Admiterea la Facultatea de Matematică și Informatică

Universitatea Babeș-Bolyai Cluj-Napoca

Probleme și teste pregătitoare

Informatică

UNIVERSITATEA BABEȘ-BOLYAI
BABEȘ-BOLYAI TUDOMÁNYEGYETEM
BABEȘ-BOLYAI UNIVERSITÄT
BABEȘ-BOLYAI UNIVERSITY

TRADITIO ET EXCELLENTIA

Probleme și Teste Grilă de Informatică pentru Admiterea la Facultatea de Matematică și Informatică Universitatea Babeș-Bolyai 2025

Coordonatori – autori:

drd. prof. Ioan Daniel Pop

stud. Rareș Cotoi

stud. Cristian Crețu

Autori:

stud. Mircea Măierean
stud. Mara Ielciu
stud. Luca Crețu
stud. Mihai Gheorgheș
stud. Luca Tudor
stud. Paul Dobrescu

Contribuitori:

prof. univ. dr. Dioșan Laura
prof. univ. dr. Pop Horia
conf. univ. dr. Șerban Camelia
lect. dr. Andor Camelia
lect. dr. Călin Alina
lect. dr. Chisăliţă-Creţu Camelia
lect. dr. Cioban Vasile
lect. dr. Ionescu Clara
lect. dr. Lupșa Dana
lect. dr. Mureșan Horea
lect. dr. Dragoș Radu
drd. Berciu Liviu
drd. Ciprian-Viorel Stupinean
drd. Dicu Mădălina
drd. Ion Iulia
drd. Măduța Adrian-Pavel
drd. Manole Alexandru
drd. Văran Andrei
prog. Dragomir Manuel
prog. Sarkozi Melisa Alexandra
prog. Zoltan Răzvan
stud. Daniș Cătălin
stud. Deaconu Mihai

Probleme și Teste Grilă de Informatică pentru Admiterea la Facultatea de Matematică și Informatică Universitatea Babeș-Bolyai 2025

Coordonatori:

Ioan Daniel Pop
Rareș Cotoi Cristian Crețu

Prefață

Această carte se adresează elevilor de liceu care doresc să își aprofundeze cunoștințele de Informatică și să se pregătească în mod eficient pentru examenul de admitere la Facultatea de Matematică și Informatică a Universității Babeș-Bolyai din Cluj-Napoca, la disciplina Informatică.

Fiecare capitol din programa oficială a examenului de Admitere este aprofundat în două părți: În prima parte sunt prezentate toate conceptele teoretice aferente capitolului, alături de exemple concrete și elemente grafice explicative, iar a doua parte conține probleme de antrenament specifice capitolului, după modelul problemelor prezente la examen. Mai apoi, sunt prezentate problemele de la toate sesiunile de admitere și de la toate edițiile concursului Mate-Info UBB dintre anii 2021 și 2024, alături de soluțiile aferente. Ultima parte a cărții conține 10 teste de antrenament a câte 24 de probleme grilă, respectând cu exactitate modelul subiectelor de la examen. Toate aceste probleme conțin, de asemenea, rezolvări și explicații, la finalul cărții.

Inițiativa și efortul principal de realizare a acestei cărți se datorează următorilor studenți: Rareș-Andrei Cotoi, Cristian-Emanuel Crețu, Mircea Măierean, Mara Ielciu, Tudor-Luca Crețu, Mihai Gheorgheș, Luca Tudor și Paul Dobrescu. Aceștia au propus probleme, au redactat soluții și au editat materialul sub coordonarea drd. prof. preuniv. Ioan Daniel Pop, cu sprijinul departamentului de Informatică și al comisiei de admitere nivel licență, coordonată de conf. univ. dr. Dan Mircea Suciu (prodecan). Le mulțumim tuturor cadrelor didactice și studenților care au contribuit la realizarea materialului prin propunerea de probleme, redactarea de soluții și realizarea de corecturi. Adresăm mulțumiri speciale pentru ajutorul acordat lect. dr. Clara Ionescu, conf. dr. Camelia Șerban și conf. dr. Marcel Șerban (decan).

Îi rugăm pe cei care observă erori (care aproape sigur s-au strecurat, în ciuda eforturilor noastre) sau au sugestii de îmbunătățire să îi contacteze pe coordonatorii culegerii.

De asemenea, le urăm mult succes elevilor în pregătirea lor pentru examenul de admitere și sperăm să-i întâlnim pe mulți dintre ei în sălile Facultății de Matematică și Informatică.

Cluj-Napoca, februarie 2025 *Coordonatorii*

Precizare: Problemele din subiectele de la concursurile Mate-Info sau examenele de Admitere **nu** se vor regăsi printre problemele din această carte. Culegerea are ca scop pregătirea și înțelegerea tipurilor de probleme abordate. Problemele propuse în subiecte la următoarele ediții de concurs/examen vor fi complet noi.

Notă importantă

În lipsa altor precizări:

- Toate operațiile aritmetice se efectuează pe tipuri de date nelimitate (nu există overflow / underflow).
- Numerotarea indicilor tuturor vectorilor, matricelor și șirurilor de caractere începe de la 1.
- Toate restricțiile se referă la valorile parametrilor actuali în momentul apelului inițial.
- O subsecvență a unui vector este formată din elemente care ocupă poziții consecutive în vector.
- Un subșir al unui vector/șir este format din elemente situate nu neapărat pe poziții consecutive în vectorul/șirul respectiv, în ordinea în care acestea apar în șirul dat.
- Dacă pe un același rând apar mai multe instrucțiuni de atribuire consecutive, acestea sunt delimitate prin "; ".

Cuprins

Partea

I

Teorie și probleme

1 Algoritmi

Acest capitol acoperă

- Noţiunea de algoritm
- Date, variabile, expresii, operaţii
- Structuri de bază (liniară, alternativă şi repetitivă)

1.1 Teorie

1.1.1 Introducere

Un algoritm reprezintă un set finit de pași bine definiți, care descriu o secvență logică de acțiuni necesare pentru a rezolva o problemă sau pentru a obține un rezultat specific. Acesta poate fi implementat ca:

- **Algoritm principal:** conține logica principală a programului și poate include operații de citire/scriere
- **Procedură (subalgoritm):** o secvență de instrucțiuni care efectuează o sarcină specifică, comunică doar prin parametri și nu returnează valori
- **Funcție (subalgoritm):** similar cu procedura, dar returnează întotdeauna o valoare și nu ar trebui să aibă efecte laterale (precum citire/scriere)

Notă: Subalgoritmii (proceduri și funcții) nu ar trebui să efectueze operații de citire sau scriere direct. Acestea ar trebui realizate în algoritmul principal sau în subalgoritmi special dedicați pentru operații I/O. Comunicarea între subalgoritmi se face exclusiv prin parametri (pentru proceduri) și prin valoarea returnată (pentru funcții).

Datele sunt informațiile procesate de către un algoritm și pot fi de diferite tipuri, precum numerice, logice, textuale, etc. O variabilă este un spațiu de memorie care poate stoca o valoare ce poate fi modificată pe parcursul execuției unui algoritm. Fiecare variabilă este identificată printr-un nume și are un tip de date asociat. O operație este compusă din operanzi și un operator. Operanzii reprezintă datele asupra cărora se aplică operația, iar operatorul este simbolul care indică tipul de operație efectuată asupra acestora. În funcție de scopul acestora, operațiile pot fi clasificate în aritmetice, logice sau relaționale.

Operanzii pot fi constante, variabile, literali, rezultatele unor funcții, rezultatele altor operații. O expresie este o operație care are ca operanzi alte operații.

Interacțiunea cu utilizatorul sau cu fișierele externe se face prin instrucțiuni de intrare (citire) și ieșire (scriere). Aceste operații asigură comunicarea dintre program și mediul înconjurător. În cele ce urmează, vom utiliza instrucțiunile `Write` și `Return` pentru a evidenția valorile afișate sau returnate de algoritmul respectiv.

1.1.2 Structuri de bază

- **Structura liniară:** expresiile care sunt evaluate secvențial pe o singură linie de cod, precum declarări, atribuiri și operații similare.

Structura liniară

$a \leftarrow 3$
$b \leftarrow 5$
Write a + b ▷ se afișează 8

- **Structura alternativă (condițională):** permite luarea unei decizii prin executarea unei ramuri de cod, în funcție de îndeplinirea unei condiții.

Structura condițională `If`

```
1: If n MOD 2 = 0 then
2:     Write "par"
3: Else
4:     Write "impar"
5: EndIf                ▷ se afișează "par" dacă n este par sau "impar" altfel
```

- **Structura repetitivă:** execută un set de instrucțiuni de mai multe ori, de obicei în funcție de o condiție specifică.

Exemplu: Structura repetitivă cu test inițial `For`

```
1: For i ← 0, 9 execute
2:     Write i, ' '
3: EndFor               ▷ se afișează toate cifrele separate de un spațiu
```

Exemplu: Structura repetitivă cu test inițial `While`

```
1: i ← 0
2: While i ≤ 9 execute
3:     Write i, ' '
4:     i ← i + 1
5: EndWhile             ▷ se afișează toate cifrele separate de un spațiu
```

Exemplu: Structura repetitivă cu test final `Do...While`

```
1: i ← 0
2: Do
3:     Write i, ' '
4:     i ← i + 1
5: While i ≤ 9          ▷ se afișează toate cifrele separate de un spațiu
```

1.2 Probleme

1. Fie algoritmul `ceFace(a, b)`, unde a și b sunt numere întregi: ✓ ?

```
Algorithm ceFace(a, b)
   a ← a DIV b
   b ← a * b
   a ← b DIV a
   Write a, ' ', b
EndAlgorithm
```

Precizați valorile inițiale ale lui a și b astfel încât algoritmul să afișeze 10 20.

A. $a = 18$, $b = 20$

B. $a = 28$, $b = 10$

C. $a = 20$, $b = 10$

D. $a = 10$, $b = 20$

2. Fie a și b două numere întregi ($-10^4 \leq a, b \leq 10^4$). Precizați care dintre următorii algoritmi realizează corect interschimbarea valorilor a și b și afișează valorile acestora interschimbate. De exemplu: Pentru $a = 12$ și $b = 6$, algoritmul ar trebui să afișeze 6 12. ✓ ?

A.

```
Algorithm swap1(a, b)
   a ← b + a
   b ← a - b
   a ← b - a
   Write a, ' ', b
EndAlgorithm
```

B.

```
Algorithm swap2(a, b)
   a ← a * b
   b ← a DIV b
   a ← a DIV b
   Write a, ' ', b
EndAlgorithm
```

C.

```
Algorithm swap3(a, b)
   a ← a DIV b
   b ← a * b
   a ← b DIV a
   Write a, ' ', b
EndAlgorithm
```

D.

```
Algorithm swap4(a, b)
   a ← a - b
   b ← a - b
   a ← b + a
   Write a, ' ', b
EndAlgorithm
```

3. Fie algoritmul `pink(a, b, k)`, unde a, b, k sunt numere naturale ($1 \leq a, b, k \leq 10^4$). ✓ ?

```
Algorithm pink(a, b, k)
   If a > b then
      a ← a ^ b
      b ← a ^ b
      a ← a ^ b
   EndIf
   While a ≤ b execute
      a ← a + k
      b ← b - k
      Write a, ' '
   EndWhile
EndAlgorithm
```

Operatorul ^ este operatorul XOR pe biți; tabelul de adevăr este următorul:

^	0	1
0	0	1
1	1	0

Exemplu: 3^5 convertit în binar este 011^101 = 110 = 6, iar 1^4 convertit în binar este 001^100 = 101 = 5.

Care dintre următoarele afirmații de mai jos nu sunt adevărate?

A. Numărul de valori afișate de algoritm este egal cu $(b - a)$ **DIV** $(k \cdot 2) + 1$.

B. Pentru $a = 12$, $b = 100$ și $k = 4$, algoritmul afișează 11 valori.

C. Pentru $a = 70$, $b = 20$ și $k = 6$, algoritmul afișează 26 32 38 44 50.

D. Algoritmul afișează întotdeauna valori din intervalul $[a, \left(\frac{a+b}{2}\right)]$, multiple de k.

4. Fie algoritmii `good(a, b)` și `bad(a, b)`, unde a și b sunt numere întregi ($-10^4 \leq a, b \leq 10^4$). ✓ ?

```
Algorithm GOOD(a, b)
    a ← a * b
    b ← a DIV b
    a ← a DIV b
    Write a, ' ', b
EndAlgorithm
```

```
Algorithm BAD(a, b)
    a ← a − b
    ____________
    a ← b − a
    Write a, ' ', b
EndAlgorithm
```

Alegeți variantele care completează corect spațiul subliniat de mai sus astfel încât cei doi algoritmi să afișeze mereu aceleași valori pentru a și b, indiferent de valorile inițiale ale acestora.

A. $b \leftarrow a + b$

B. $b \leftarrow a - b$

C. $b \leftarrow b - a$

D. $b \leftarrow a$ * b DIV $a + a$

5. Fie algoritmul `container(n)`, unde n este un număr natural ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm CONTAINER(n)
    a ← 0
    b ← 1
    While n > 1 execute
        a ← a + 1
        b ← b * 2
        n ← n − b
    EndWhile
    Return a
EndAlgorithm
```

Care dintre următoarele afirmații de mai jos sunt adevărate?

A. Pentru $n = 1024$, algoritmul returnează 9.

B. Pentru $n = 336$, algoritmul returnează 8.

C. Algoritmul returnează întotdeauna o putere a lui 2.

D. Algoritmul `container(n)` returnează cel mai mare număr k, cu proprietatea că numărul 2^k este mai mic sau egal decât n.

6. Fie expresia $E = n$ `DIV` $100 + n$ `MOD` 100 `DIV` $10 + n$ `MOD` 10, unde n este un număr natural nenul ($1 \leq n < 10^3$). Precizați pentru câte valori ale lui n, care respectă specificațiile din enunț, expresia E are valoarea 9. ✓ ?

A. 9 B. 10 C. 45 D. 55

2

Tipuri de date. Operatori.

Acest capitol acoperă

- Care sunt principalele tipuri de date și cum se reprezintă?
- Care sunt tipurile de operatori și cum sunt aceștia utilizați?

2.1 Teorie

2.1.1 Noțiunea de tip de date

Tipul de dată definește domeniul de valori pe care le poate lua o variabilă și operațiile permise asupra acesteia. În limbajele de programare, tipurile de date pot fi: numerice întregi, numerice cu virgulă mobilă (detaliat mai jos), caracter, logice sau structurate (tablou, structură).

2.1.2 Definirea tipurilor de date

Tipurile de date fundamentale pot fi descrise, din punct de vedere numeric, prin intervale ce derivă din numărul de biți alocați fiecărui tip. De obicei, un tip de date reprezentat pe n biți (cu semn, folosind complementul lui 2 - se va detalia în cadrul materiei Arhitectura Sistemelor de Calcul) poate reprezenta valori în intervalul $[-2^{n-1}, 2^{n-1} - 1]$, iar același tip, fără semn, poate reprezenta valori în $[0, 2^n - 1]$. Totuși, pentru tipurile uzuale de date avem:

- `char` (8 biți):
  - `signed char`: $[-2^7, 2^7 - 1]$;
  - `unsigned char`: $[0, 2^8 - 1]$.
- `short` (16 biți):
  - `short`: $[-2^{15}, 2^{15} - 1]$;
  - `unsigned short`: $[0, 2^{16} - 1]$.
- `int` (**Minim** 16 biți, dar poate fi și mai mult, în funcție de limbaj):
  - `int`: **Minim** $[-2^{15}, 2^{15} - 1]$;
  - `unsigned int`: **Minim** $[0, 2^{16} - 1]$.
- `long` (**Minim** 32 biți, dar poate fi și mai mult, în funcție de limbaj):
  - `long`: **Minim** $[-2^{31}, 2^{31} - 1]$;
  - `unsigned long`: **Minim** $[0, 2^{32} - 1]$.
- `long long` (**Minim** 64 biți, dar poate fi și mai mult, în funcție de limbaj):
  - `long`: **Minim** $[-2^{63}, 2^{63} - 1]$;

  - `unsigned long`: **Minim** $[0, 2^{64} - 1]$.

- `float` și `double`: Acestea reprezintă numere reale în format cu virgulă mobilă, codificate conform standardului IEEE 754. Intervalele nu se exprimă simplu sub formă de puteri ale lui 2 pentru întregul domeniu numeric, deoarece reprezintă numere în formă normalizată (se va detalia în cadrul materiei Logică computațională). Totuși, precizia lor în biți este:
  - `float`: 32 biți (aprox. 24 biți pentru mantisă);
  - `double`: 64 biți (aprox. 53 biți pentru mantisă).

2.1.3 Operatori

Operatorii sunt de 3 tipuri: aritmetici, logici și relaționali. Cei pe care noi îi vom folosi în cele ce urmează sunt:

- **Operatori aritmetici:** `+`, `-`, `*`, `/`, `%`;
- **Operatori logici:** `AND`, `OR`, `NOT`;
- **Operatori relaționali:** `<`, `>`, $\leq$, $\geq$, `=`, $\neq$

Operatorii aritmetici sunt evaluați conform regulilor standard de prioritate:

- Operatorii `*`, `/` și `%` au prioritate mai mare decât `+` și `-`;
- Expresiile grupate prin paranteze au prioritate în timpul evaluării unei expresii;
- Pentru operatorii de aceeași prioritate se aplică evaluarea de la stânga la dreapta.

Operatorii logici operează pe valori de adevăr (`True` sau `False`) și se evaluează după următoarele principii:

- `AND`: Returnează `True` doar dacă ambii operanzi sunt `True`;
- `OR`: Returnează `True` dacă cel puțin unul dintre operanzi este `True`;
- `NOT`: Inversează valoarea operandului (adică, `NOT False = True` și `NOT True = False`).

A	B	A `AND` B	A `OR` B
`False`	`False`	`False`	`False`
`False`	`True`	`False`	`True`
`True`	`False`	`False`	`True`
`True`	`True`	`True`	`True`

Tabela 2.1 Tabela de adevăr pentru operatorii logici `AND` și `OR`.

Operatorii relaționali compară două valori și returnează un rezultat boolean, după cum urmează:

- `<`: Verifică dacă valoarea din stânga este mai mică decât cea din dreapta.
- `>`: Verifică dacă valoarea din stânga este mai mare decât cea din dreapta.

- $\leq$: Verifică dacă valoarea din stânga este mai mică sau egală cu cea din dreapta.
- $\geq$: Verifică dacă valoarea din stânga este mai mare sau egală cu cea din dreapta.
- `=`: Verifică egalitatea dintre cele două valori.
- $\neq$: Verifică inegalitatea dintre cele două valori.

Evaluarea expresiilor relaționale se efectuează după evaluarea expresiilor aritmetice, respectând regulile standard de comparare.

2.2 Probleme

7. Fie variabila `n` care memorează un număr natural. Care dintre expresiile de mai jos are valoarea `True` dacă și numai dacă `n` este divizibil cu 5 și cu 9? ✓ ?

A. $(n$ `MOD` $5 \neq 1)$ `AND` $(n$ `MOD` $9 = 0)$

B. $((n$ `DIV` $10)$ `MOD` $5 \neq 1)$ `AND` $(n$ `DIV` $9 = 0)$

C. $(n$ `MOD` $5 \neq 1)$ `OR` $(n$ `MOD` $9 = 0)$

D. $(n$ `MOD` $5 = 0)$ `AND` $(n$ `MOD` $9 = 0)$

8. Fie variabila `y` care memorează un număr natural. Care dintre expresiile de mai jos are valoarea `True` dacă și numai dacă `y` este mai mare decât 10 și nu este divizibil cu 4? ✓ ?

A. $(y > 10)$ `AND` $(y$ `MOD` $4 \neq 0)$

B. $(y > 10)$ `OR` $(y$ `MOD` $4 = 0)$

C. `NOT`$(y \leq 10)$ `AND` $(y$ `MOD` $4 = 0)$

D. $(y > 10)$ `AND` $(y$ `MOD` $2 \neq 0)$

9. Fie variabila `k` care memorează un număr natural. Care dintre expresiile de mai jos are valoarea `True` dacă și numai dacă `k` este un număr par mai mic decât 20? ✓ ?

A. $(k$ `MOD` $2 = 0)$ `AND` $(k < 20)$

B. $(k$ `MOD` $2 \neq 0)$ `AND` $(k \geq 20)$

C. $(k$ `MOD` $2 = 0)$ `OR` $(k \geq 20)$

D. $(k$ `MOD` $2 = 0)$ `AND` $(k \leq 20)$

10. Fie variabila `m` care memorează un număr întreg. Care dintre expresiile de mai jos este `True` dacă și numai dacă `m` este un număr negativ divizibil cu 8? ✓ ?

A. $(m > 0)$ `AND` $(m$ `MOD` $8 \neq 0)$

B. $(m < 0)$ `OR` $(m$ `MOD` $8 \neq 0)$

C. $(m < 0)$ `AND` $(m$ `MOD` $8 = 0)$

D. $(m$ `MOD` $8 = 0)$ `AND` $(m > 0)$

11. Fie variabilele `a = 16`, `b = 4` şi `c = 2`. Ce valoare va avea expresia următoare? ✓ ?

$$(a \texttt{ MOD } b + b * c) \texttt{ DIV } (a \texttt{ DIV } b - c) + \Big((a + 1) \texttt{ MOD } (b - 1)\Big)$$

A. 4 B. 5 C. 6 D. 7

12. Fie variabilele `x = 20`, `y = 7` şi `z = 3`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big((x+1)\ \texttt{MOD}\ y+z\Big) * \Big((x-1)\ \texttt{DIV}\ z\Big) - \Big(y\ \texttt{MOD}\ (z-1)\Big)$$

A. 18 B. 17 C. 21 D. 23

13. Fie variabilele `a = 10`, `b = 5` şi `c = 8`. Ce valoare va avea `a` după executarea următoarei instrucţiuni? ✓ ?

$$a = a + (b * c)\ \texttt{DIV}\ (c\ \texttt{MOD}\ (b+2)) - ((c-1) * (b+2))$$

A. 1 B. 8 C. 9 D. 10

14. Fie variabilele `x = 8`, `y = 15` şi `z = 5`. Ce valoare va avea următoarea expresie? ✓ ?

$$\Big((x\ \texttt{MOD}\ (y-1)) + (z * (x+1))\Big)\ \texttt{DIV}\ ((y-1)\ \texttt{MOD}\ (z-1))$$

A. 25 B. 26 C. 27 D. 28

15. Se consideră următoarea expresie logică: ✓ ?

$$(A\ \texttt{OR}\ B)\ \texttt{AND}\ (\ \texttt{NOT}\ A\ \texttt{OR}\ C)\ \texttt{OR}\ (B\ \texttt{AND}\ (\ \texttt{NOT}\ C\ \texttt{OR}\ A))$$

Precizați pentru ce valori ale lui A, B, C, expresia are valoarea `True`.

A. $A =$ `True`, $B =$ `False`, $C =$ `True`
B. $A =$ `False`, $B =$ `True`, $C =$ `False`
C. $A =$ `True`, $B =$ `True`, $C =$ `False`
D. $A =$ `False`, $B =$ `False`, $C =$ `True`

16. Se consideră următoarea expresie logică: ✓ ?

$$(\ \texttt{NOT}\ C\ \texttt{AND}\ B)\ \texttt{OR}\ (A\ \texttt{AND NOT}\ B)\ \texttt{AND}\ (\ \texttt{NOT}\ A\ \texttt{OR}\ C)$$

Precizați pentru ce valori ale lui A, B, C, expresia are valoarea `True`.

A. $A =$ `True`, $B =$ `False`, $C =$ `False`
B. $A =$ `False`, $B =$ `True`, $C =$ `False`
C. $A =$ `True`, $B =$ `True`, $C =$ `False`
D. $A =$ `False`, $B =$ `False`, $C =$ `True`

17. Se consideră următoarea expresie logică: ✓ ?

$$((A\ \texttt{AND}\ (\ \texttt{NOT}\ B\ \texttt{OR}\ C))\ \texttt{OR}\ ((\ \texttt{NOT}\ A\ \texttt{AND}\ B)\ \texttt{AND}\ (\ \texttt{NOT}\ C\ \texttt{OR}\ A)))\ \texttt{AND}\ (B\ \texttt{OR NOT}\ C)$$

Precizați pentru ce valori ale lui A, B, C, expresia are valoarea `True`.

A. $A = \texttt{True}, B = \texttt{False}, C = \texttt{False}$
C. $A = \texttt{True}, B = \texttt{True}, C = \texttt{False}$
B. $A = \texttt{False}, B = \texttt{True}, C = \texttt{False}$
D. $A = \texttt{True}, B = \texttt{False}, C = \texttt{True}$

18. Se consideră următoarea expresie logică: ✓ ?

(NOT ((A OR B) AND C) OR ((B AND NO C) OR (NOTA AND (C OR NOT B)))) AND A

Precizați pentru ce valori ale lui A, B, C, expresia are valoarea `True`.

A. $A = \texttt{True}, B = \texttt{True}, C = \texttt{False}$
C. $A = \texttt{True}, B = \texttt{False}, C = \texttt{False}$
B. $A = \texttt{False}, B = \texttt{False}, C = \texttt{True}$
D. $A = \texttt{True}, B = \texttt{False}, C = \texttt{True}$

19. Se consideră următoarea expresie logică: ✓ ?

(((A AND B) OR (NOT C AND A)) AND (B OR NOT A)) OR ((NOT B AND C) AND (NOT A OR B))

Precizați pentru ce valori ale lui A, B, C, expresia are valoarea `True`.

A. $A = \texttt{True}, B = \texttt{False}, C = \texttt{False}$
C. $A = \texttt{True}, B = \texttt{False}, C = \texttt{True}$
B. $A = \texttt{False}, B = \texttt{False}, C = \texttt{True}$
D. $A = \texttt{True}, B = \texttt{True}, C = \texttt{False}$

20. Se consideră următoarea expresie logică: ✓ ?

(((A OR (NOT B AND C)) AND (NOT A OR B)) OR ((NOT C OR A) AND (NOT B OR NOT A))) AND C

Precizați pentru ce valori ale lui A, B, C, expresia are valoarea `False`.

A. $A = \texttt{True}, B = \texttt{True}, C = \texttt{False}$
C. $A = \texttt{False}, B = \texttt{True}, C = \texttt{False}$
B. $A = \texttt{True}, B = \texttt{False}, C = \texttt{True}$
D. $A = \texttt{False}, B = \texttt{False}, C = \texttt{True}$

21. Fie variabilele `p = 12`, `q = 3` şi `r = 4`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big(((p+1)\ \texttt{MOD}\ q) + (r * (p+1))\Big) - \Big(q\ \texttt{MOD}\ (r-1)\Big) + \Big((p+1)\ \texttt{DIV}\ (q+2)\Big)$$

A. 50; B. 52; C. 55; D. 56.

22. Fie variabilele `a = 9`, `b = 2` şi `c = 5`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big((a-1)\ \texttt{MOD}\ b\Big) + \Big(b * (c+1)\Big) - \Big((a-1)\ \texttt{DIV}\ b\Big)$$

A. 7; B. 8; C. 9; D. 10.

23. Fie variabilele `x = 25`, `y = 4` şi `z = 2`. Ce valoare va avea următoarea expresie? ✓ ?

$$(x\ \texttt{MOD}\ y) + \Big((y+1) * z\Big) - \Big(x\ \texttt{DIV}\ (y+1)\Big)$$

A. 4; B. 6; C. 8; D. 10.

24. Fie variabilele `a = 30`, `b = 6`, `c = 2`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big(a \texttt{ DIV } b - c\Big) + \Big((b-1) \texttt{ MOD } (c+1)\Big) + \Big(a \texttt{ MOD } (b-1)\Big)$$

A. 3; B. 4; C. 5; D. 6.

25. Fie variabilele `m = 10`, `n = 3` şi `p = 3`. Ce valoare va avea `m` după instrucţiunea: ✓ ?

$$m = ((m+1) \texttt{ MOD } n) + \Big((n-1) * p\Big) - \Big((m+1) \texttt{ DIV } (p+0)\Big)$$

A. 3; B. 4; C. 5; D. 6.

26. Fie variabilele `a = 14`, `b = 7` şi `c = 2`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big(a \texttt{ MOD } (b-1)\Big) + \Big((a+1) \texttt{ DIV } (b-1) + c\Big) * \Big(2 \texttt{ MOD } (b-1)\Big)$$

A. 10; B. 16; C. 18; D. 20.

27. Fie variabilele `x = 4`, `y = 2` şi `z = 1`. Ce valoare va avea `x` după executarea acestei instrucțiuni? ✓ ?

$$x = (x * (y+1)) - \Big((y+1) \texttt{ DIV } (z+1)\Big) + \Big((x+1) \texttt{ MOD } y\Big)$$

A. 10; B. 11; C. 12; D. 13.

28. Fie variabilele `a = 6`, `b = 3`, `c = 2`. Ce valoare va avea următoarea expresie? ✓ ?

$$\Big(a \texttt{ DIV } (b-1)\Big) + \Big((b-1) * (c \texttt{ MOD } (a+1))\Big) - \Big(a \texttt{ MOD } (c+1)\Big)$$

A. 5; B. 6; C. 7; D. 8.

29. Fie variabilele `u = 12`, `v = 4`, `w = 3`. Ce valoare va avea expresia? ✓ ?

$$\Big((u-1) * v\Big) \texttt{ DIV } \Big((u-1) \texttt{ MOD } w\Big) + \Big((v-1) \texttt{ MOD } (w-1)\Big)$$

A. 20; B. 21; C. 22; D. 23.

30. Fie variabilele `x = 5`, `y = 3`, `z = 10`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big(x \texttt{ MOD } y\Big) + \Big(z \texttt{ DIV } (y+1)\Big) - \Big((z-1) \texttt{ MOD } (x-1)\Big)$$

A. 2; B. 3; C. 4; D. 5.

31. Fie variabilele `x = 7`, `y = 4`, `z = 2`. Care este valoarea următoarei expresii? ✓ ?

$$\Big((x \texttt{ MOD } (y-1)) + (z * (x+2))\Big) - \Big((x+2) \texttt{ DIV } z\Big)$$

A. 15; B. 16; C. 17; D. 18.

32. Fie variabilele `a = 12`, `b = 5`, `c = 3`. Care este valoarea returnată de expresia următoare? ✓ ?

$$\Big((a \texttt{ DIV } b) > c\Big) \texttt{ AND } \Big((b-1) < c\Big) \texttt{ OR } \Big(a \texttt{ MOD } (b-1) = 2\Big)$$

A. `True`;
B. `False`;
C. Eroare de compilare;
D. Niciuna dintre variante.

33. Fie tipul de date `signed char`, reprezentat pe 8 biți cu semn. Care este domeniul de valori care se pot reprezenta pe acest tip de date? ✓ ?

A. $[-128, 127]$; B. $[-127, 128]$; C. $[0, 255]$; D. $[-256, 255]$.

34. Fie variabila `k = 3`. Ce valoare va avea `k` după executarea acestei instrucțiuni? ✓ ?

$$k = k + \Big((k+2) \texttt{ MOD } (k+2)\Big) + \Big((k+1) \texttt{ DIV } 2\Big)$$

A. 1; B. 2; C. 3; D. 5.

35. Fie tipul de date `unsigned long`, reprezentat pe 32 de biți. Care dintre următoarele intervale este valabil pentru valorile reprezentabile pe acest tip de date? ✓ ?

A. $[0, 2^{31} - 1]$; B. $[0, 2^{32} - 1]$; C. $[0, 2^{63} - 1]$; D. $[0, 2^{64} - 1]$.

36. Fie variabilele `x = 10`, `y = 5`. Ce valoarea va returna expresia următoare? ✓ ?

$$\Big(x > 5\Big) \texttt{ AND } \Big((x+y) < 20\Big) \texttt{ OR } \Big(y \texttt{ MOD } 2 = 1\Big)$$

A. `True`;
B. `False`;
C. Eroare de compilare;
D. Niciuna dintre variante.

37. Fie variabila `n = 255` de tipul `short`. Poate valoarea lui `n` să fie stocată fără depăşire într-o altă variabilă de tipul `signed char`? ✓ ?

A. Da;
B. Nu;
C. Depinde de limbajul de programare;
D. Depinde de implementare.

38. Fie variabilele `a = 8`, `b = 3`, `c = 1`. Ce valoare va returna expresia următoare? ✓ ?

$$\Big((a+1)\ \texttt{DIV}\ b\Big) - \Big(b\ \texttt{MOD}\ c\Big) + \Big((a+1)\ \texttt{MOD}\ b\Big)$$

A. 1; B. 6; C. 2; D. 3.

39. Fie variabila `x` de tipul `signed short`. Inițializarea `x` = 200 este: ✓ ?

A. Incorectă, deoarece depășește intervalul lui `signed short`;
B. Corectă, deoarece `signed short` stochează valori până la 127;
C. Corectă, deoarece `signed short` acceptă valori din [-32768, 32767];
D. Depinde de implementare.

40. Fie variabilele `m = 15`, `n = 4`, `p = 2`. Care e valoarea finală a lui `m` după instrucţiunea următoare? ✓ ?

$$m = \Big(m\ \texttt{MOD}\ (n+p)\Big) + \Big(n\ \texttt{DIV}\ p\Big)$$

A. 1; B. 3; C. 5; D. 7.

41. Care este limita inferioară garantată pentru tipul de dată `long long` reprezentat pe minim 64 de biți **cu semn**? ✓ ?

A. -2^{63}; B. -2^{64}; C. 0; D. Depinde.

42. Fie variabilele `x = 13`, `y = 3`, `z = 5`. Ce valoare va returna expresia următoare? ✓ ?

$$\Big((x\ \texttt{MOD}\ y) = 1\Big)\ \texttt{AND}\ \Big((z*y) > x\Big)\ \texttt{OR}\ \Big(x < (z+y)\Big)$$

A. `True`;
B. `False`;
C. Eroare de compilare;
D. Niciuna dintre variante.

43. Fie variabilele `a = 1`, `b = 2`, `c = 3`. Ce valoare va returna expresia următoare? ✓ ?

$$\Big((b\ \texttt{DIV}\ a) > 0\Big)\ \texttt{OR}\ \Big((c-1) < (b+1)\Big)\ \texttt{AND}\ \Big(a == 2\Big)$$

A. `True`;
B. `False`;
C. Eroare de compilare;
D. Niciuna dintre variante.

44. Fie variabilele `x = 6` și `y = 10`. Ce valoarea va returna următoarea expresie? ✓ ?

$$\Big((x+y) \leq 15\Big)\ \texttt{AND}\ \Big((y-x) \geq 4\Big)\ \texttt{OR}\ \Big((x*y)\ \texttt{MOD}\ 2 = 0\Big)$$

A. `True`;
B. `False`;
C. Eroare de compilare;
D. Niciuna dintre variante.

45. Care dintre următoarele expresii logice sunt adevărate dacă și numai dacă valoarea naturală memorată în variabila `x` este multiplu de 5 și aparține intervalului $(a, b]$? ✓ ?

A. `NOT`$(x$ `MOD` $5 \neq 0$ `OR` $x \leq a)$ `AND` $(x \leq b)$
B. x `MOD` $5 \neq 1$ `AND` $x > a$ `AND` $x \leq b$
C. x `MOD` $5 = 0$ `AND` `NOT`$(x < a$ `AND` $x > b)$
D. `NOT`$(x$ `MOD` $5 \neq 0$ `AND` $x \leq a$ `AND` $x > b)$

3

Tipuri structurate de date

Acest capitol acoperă

- Care este importanța tipurilor structurate de date?
- Care sunt tipurilor de date structurate studiate și cum le folosim?

3.1 Teorie

3.1.1 Introducere

Tipurile structurate de date reprezintă un concept fundamental în informatică, acestea permițând organizarea eficientă a informațiilor în programe și aplicații, facilitând atât manipularea, cât și accesarea rapidă a datelor. Ele sunt utilizate pentru a gestiona volume mari de date și pentru a rezolva probleme complexe, cum ar fi căutarea, sortarea sau determinarea celui mai scurt drum între două puncte într-o rețea.

Analogic, putem compara acest concept cu o listă de cumpărături: în loc să ai obiectele împrăștiate peste tot, le organizezi într-o listă bine structurată, unde știi exact ce trebuie să cumperi și în ce ordine. Similar, în programare, structurile de date sunt utilizate pentru a organiza informația într-un mod eficient, permițând accesarea și manipularea rapidă și corectă a datelor.

Tipuri structurate de date:

- Tipul tablou;
- Tipul șir de caractere;
- Tipul înregistrare.

3.1.2 Tipul tablou: tablouri în memorie

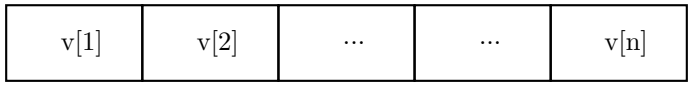
Un **tablou** este o structură de date compusă dintr-un număr fix de elemente, toate având același tip de bază. Acesta reprezintă **o zonă de memorie** identificată printr-un **nume specific**, permițând stocarea mai multor valori de **același tip**. Datele dintr-un tablou pot fi gestionate fie ca un tot unitar, fie ca elemente independente, facilitând accesul rapid și organizat la fiecare valoare.

Tablourile de memorie studiate sunt:

- Tablouri de memorie unidimensionale;
- Tablouri de memorie bidimensionale.

3.1.3 Tablouri de memorie unidimensionale

Tablourile unidimensionale, cunoscute și sub denumirea de **vectori**, sunt structuri de date caracterizate prin **un singur indice**, fiecare element fiind accesibil prin intermediul acestuia. Dacă un tablou conține un număr de elemente, indicii acestor elemente sunt valori întregi cuprinse în intervalul [**1, numărul de elemente**]. Astfel, vectorii permit organizarea eficientă a datelor în memorie și facilitează efectuarea de operații asupra fiecărui element în parte, eliminând necesitatea stocării individuale a fiecărei variabile. De exemplu, vectorul **v** cu un număr de **n** elemente poate fi reprezentat astfel:



Operații cu vectori:

Citirea elementelor unui vector:

```
Algorithm Citire(v,n)
    For i ← 1, n execute
        Read v[i]
    EndFor
EndAlgorithm
```

Afișarea elementelor unui vector:

```
Algorithm Afișare(v,n)
    For i ← 1, n execute
        Write v[i]
    EndFor
EndAlgorithm
```

Afișarea elementelor unui vector în ordine inversă:

```
Algorithm AfișareR(v,n)
    For i ← n, 1, −1 execute
        Write v[i]
    EndFor
EndAlgorithm
```

Cele mai întâlnite probleme cu vectori sunt:

- Inversarea unui vector
- Suma elementelor unui vector
- Determinarea valorii minime/maxime
- Inserarea sau eliminarea unui element
- Căutarea unui element

- Secvențe, interclasare, sortare
- Vectori de frecvență

Elementele unui tablou pot fi de orice tip de date disponibil în limbajul de programare utilizat. Astfel, ele pot fi chiar și alte tablouri, ceea ce permite crearea de tablouri bidimensionale, cunoscute și sub numele de matrice.

3.1.4 Tablouri de memorie bidimensionale

Un tablou bidimensional este o structură de date cu **două dimensiuni**: una corespunzătoare **liniilor** și alta **coloanelor**. Fiecare element este accesibil folosind doi indici, unul pentru linia pe care se află și altul pentru coloana respectivă. De exemplu, $matrice[i][j]$ reprezintă elementul aflat la intersecția liniei i și a coloanei j. O matrice cu numărul de linii egal cu numărul de coloane se numește o **matrice pătratică**.

Tablourile bidimensionale sunt utilizate pentru a reprezenta date sub formă de grilă sau tabel, cum ar fi matrice matematice, tabele de scoruri, imagini sau hărți. Acestea oferă un mod eficient de a organiza și accesa date structurale complexe și sunt esențiale în diverse domenii, cum ar fi calculul numeric, grafica computerizată și știința datelor.

De exemplu, matricea **mat** cu **n** linii și **m** coloane poate fi reprezentată în următorul mod:

mat[1][1]	mat[1][2]	···	mat[1][m]
mat[2][1]	mat[2][2]	···	mat[2][m]
...	...	...	...
mat[n][1]	mat[n][2]	···	mat[n][m]

Operații cu matrice:

Citirea elementelor unei matrice:

```
Algorithm CITIRE(mat,n,m)
    For i ← 1, n execute
        For j ← 1, m execute
            Read mat[i][j]
        EndFor
    EndFor
EndAlgorithm
```

Afișarea elementelor unei matrice:

```
Algorithm AFIȘARE(mat,n,m)
    For i ← 1, n execute
        For j ← 1, m execute
            Write mat[i][j]
        EndFor
    EndFor
EndAlgorithm
```

3.1.5 Matrici pătratice

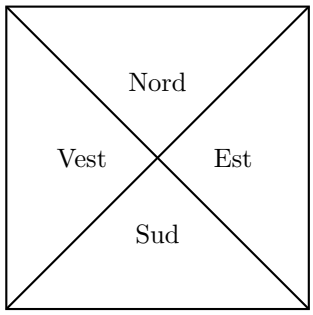
Notăm cu n numărul de linii și coloane ale matricei, i reprezentând indicele liniei, iar j indicele coloanei.

Relațiile dintre indicii unei matrice pătratice:

- **Diagonala principală**: Elementele de pe diagonala principală sunt cele pentru care indicele liniei este egal cu indicele coloanei, adică $i = j$. Deasupra diagonalei principale, se respectă relația $i < j$, iar sub diagonala principală $i > j$.

- **Diagonala secundară**: Elementele de pe diagonala secundară sunt cele pentru care suma indicilor liniei și coloanei este egală cu $n-1$, adică $i+j = n+1$. Deasupra diagonalei secundare, se respectă relația $i + j < n + 1$, iar sub diagonala secundară $i + j > n + 1$.

Prin trasarea ambelor diagonale, se delimitează patru zone diferite ale matricei: **Nord**, **Sud**, **Est** și **Vest**



- **Nord**: Elemente situate deasupra diagonalei principale și a diagonalei secundare, unde $i < j$ și $i + j < n + 1$.
- **Sud**: Elemente situate sub diagonala principală și sub diagonala secundară, unde $i > j$ și $i + j > n + 1$.
- **Vest**: Elemente situate sub diagonala secundară, dar deasupra diagonalei principale, unde $i > j$ și $i + j < n + 1$.
- **Est**: Elemente situate deasupra diagonalei secundare, dar sub diagonala principală, unde $i < j$ și $i + j > n + 1$.

Cele mai întâlnite probleme cu matrice sunt:

- Interschimbarea a două linii sau coloane
- Rotirea unei matrice
- Labirint

3.1.6 Tipul șir de caractere

Un **șir de caractere** este o structură de date utilizată pentru a stoca și manipula secvențe de caractere. Acesta poate reprezenta cuvinte, propoziții sau alte combinații de caractere,

fiind unul dintre cele mai utilizate tipuri de date în programare. În majoritatea limbajelor de programare, un șir este tratat ca un tablou unidimensional de caractere.

Reprezentarea șirurilor de caractere și codurile ASCII:
Fiecare caracter dintr-un șir este reprezentat intern printr-un cod numeric, conform standardului **ASCII** (American Standard Code for Information Interchange). De exemplu:
-'A' are codul ASCII 65.
-'a' are codul ASCII 97.
-'0' are codul ASCII 48.
Această reprezentare permite manipularea directă a caracterelor folosind operații pe codurile lor. În memorie, un șir de caractere este stocat sub forma unui tablou de caractere, fiecare celulă conținând codul ASCII al unui caracter. Șirul este terminat de caracterul special **'\0'** (null) pentru a marca sfârșitul.

Accesarea unui caracter dintr-un șir:
Fiecare caracter al unui șir poate fi accesat folosind un **indice** (începând de la 1). De exemplu, pentru șirul de caractere $s =$ "admitere UBB", $s[1] =$ 'a', $s[5] =$ 't', $s[8] =$ ' ', $s[12] =$ 'B' și $s[13] =$ '\0'.

3.2 Probleme

46. Se consideră algoritmul `Verifica(n, a)`, unde n este un număr natural ($1 \leq n \leq 10^3$) și a este un vector cu n elemente numere întregi $(a[1], a[2], ..., a[n])$, unde $-100 \leq a[i] \leq 100$, pentru $i = 1, 2, ..., n$: ✓ ?

```
Algorithm VERIFICA(n, a)
    m ← n DIV 2
    For i ← 1,m execute
        If n MOD 2 = 0 then
            aux ← a[m + i]
            a[m + i] ← a[n − i + 1]
            a[n − i + 1] ← aux
        Else
            aux ← a[m + i + 1]
            a[m + i + 1] ← a[n − i + 1]
            a[n − i + 1] ← aux
        EndIf
    EndFor
    For i ← 1,m execute
        If a[i] ≠ a[n − i + 1] then
            Return False
        EndIf
    EndFor
    Return True
EndAlgorithm
```

Se cere să se determine pentru ce valori ale vectorului a, apelul `Verifica(n, a)` returnează `True`.

A. $a = [1, 2, 3, 4, 3, 2, 1]$;

B. $a = [1, 2, 3, 3, 3, 1, 2, 1]$;

C. $a = [1, 2, 1, 2]$;

D. $a = [-200, 10, -200]$.

47. Se consideră algoritmul `Alg(x, m)`, unde m este un număr natural ($1 \leq m \leq 10^3$) și x este un vector cu m elemente numere întregi $(x[1], x[2], ..., x[m])$, unde $-100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., m$: ✓ ?

```
Algorithm ALG(x, m)
    fx ← 0
    For p ← 1,m execute
        a ← 0
        For q ← p,m execute
            a ← a + x[q]
            If a = 0 then
                w ← q − p + 1
                If w > fx then
                    fx ← w
                EndIf
            EndIf
        EndFor
    EndFor
    Return fx
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Algoritmul are o complexitate de timp $O(m^3)$;

B. Variabila a este folosită pentru a calcula lungimea unei subsecvențe;

C. Pentru $x = [2, 2, 2]$, rezultatul returnat este 0;

D. Pentru $x = [0, 2, -2, 1, 1, -1, 1, 6, 0, -1]$, rezultatul returnat este 4.

48. Se consideră algoritmul `Contor(x,n)`, unde n este un număr natural ($1 \leq n \leq 10^3$) și x este un vector cu n elemente numere întregi ($x[1], x[2], ..., x[n]$), unde $0 \leq x[i] \leq 10^5$, pentru $i = 1, 2, ..., n$ și algoritmul `Alg(numar)`, unde *numar* este un număr natural ($0 \leq numar$):

```
Algorithm ALG(numar)
   If numar < 2 then Return False
   EndIf
   For d ← 2,√numar execute
      If numar MOD d = 0 then
         Return False
      EndIf
   EndForReturn True
EndAlgorithm
Algorithm CONTOR(x, n)
   count ← 0
   For i ← 1,n − 1 execute
      For j ← i + 1 to n execute
         v ← x[i]
         asc ← True
         ok ← 1
         For k ← i,j − 1 execute
            If x[k] ≥ x[k + 1] then
               asc ← False
               ok ← 0
            EndIf
            If ok = 1 then
               v ← v + x[k + 1]
            EndIf
         EndFor
         If asc AND ALG(v) then
            count ← count + 1
         EndIf
      EndFor
   EndFor
   Return count
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $x = [7, 7, 7, 7, 7, 7, 7]$, rezultatul returnat de `Contor(x,7)` este 7;

B. Pentru $x = [2, 3, 5, 7]$, rezultatul returnat de `Contor(x,4)` este 2;

C. Pentru $x = [7, 5, 3, 2]$, rezultatul returnat de `Contor(x,4)` este 2;

D. Pentru $x = [7, 2, 2, 9, 6]$, rezultatul returnat de `Contor(x,5)` este 1.

49. Se consideră 2 algoritmi: `A(x,n,k)` și `B(x,n,k)`, unde n este un număr natural ✓ ? $(1 \leq n \leq 10^3)$ și x este un vector cu n elemente numere întregi $(x[1], x[2], ..., x[n])$, unde $0 \leq x[i] \leq 10^5$, pentru $i = 1, 2, ..., n$, k număr natural.

```
Algorithm A(x, n, k)
   k ← k MOD n
   initializează temp[1 . . . k]
   For i ← 1, k execute
      temp[i] ← x[i]
   EndFor
   For i ← k + 1, n execute
      x[i − k] ← x[i]
   EndFor
   For i ← 1, k execute
      x[n − k + i] ← temp[i]
   EndFor
EndAlgorithm
```

```
Algorithm HELPER(x, n)
   primul ← x[1]
   For i ← 1, n − 1 execute
      x[i] ← x[i + 1]
   EndFor
   x[n] ← primul
EndAlgorithm
Algorithm B(x, n, k)
   For i ← 1, k execute
      HELPER(x, n)
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul A face același lucru ca algoritmul B, dar algoritmul A este mai eficient;

B. Algoritmul A nu face același lucru ca algoritmul B;

C. Algoritmul A face același lucru ca algoritmul B, dar algoritmul B este mai eficient;

D. Algoritmul A face același lucru ca algoritmul B doar dacă $k = n$.

50. Se consideră algoritmul `Calculeaza(x,n)`, unde n este un număr natural ($1 \leq n \leq 10^3$) și x este un vector cu n elemente numere întregi ($x[1], x[2], ..., x[n]$), unde $-10^4 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$: ✓ ?

```
Algorithm CALCULEAZA(x, n)
    s ← x[1]
    a ← x[1]
    For i ← 2, n execute
        If a + x[i] > x[i] then
            a ← a + x[i]
        Else
            a ← x[i]
        EndIf
        If a > s then
            s ← a
        EndIf
    EndFor
    Return s
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $x = [-2, 1, -3, 4, -1, 2, 1, -5, 4]$, rezultatul returnat de `Calculeaza(x,9)` este 6;

B. Pentru $x = [1, -2, 3, 5, -1]$, rezultatul returnat de `Calculeaza(x,5)` este 8;

C. Pentru $x = [-1, -2, -3, -4]$, rezultatul returnat de `Calculeaza(x,4)` este -1;

D. Pentru $x = [2, 4, -1, 3]$, rezultatul returnat de `Calculeaza(x,4)` este 6.

Problemele **51.** și **52.** se referă la următorii 2 algoritmi:
Se consideră algoritmul `Afiseaza(x,n,k)`, unde n este un număr natural ($1 \leq n \leq 10^3$), x este un vector cu n elemente numere întregi ($x[1], x[2], ..., x[n]$), unde $-10^4 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$, și k este un număr natural ($1 \leq k \leq n$), și algoritmul `A(x,n)`, unde n, x au aceleași restricții ca pentru algoritmul `Afiseaza(x,n,k)`:

```
Algorithm A(x, n)
    For i ← 1,n − 1 execute
        For j ← i + 1,n execute
            If x[i] > x[j] then
                a ← x[i]
                x[i] ← x[j]
                x[j] ← a
            EndIf
        EndFor
    EndFor
EndAlgorithm
```

```
Algorithm AFISEAZA(x, n, k)
    A(x, n)
    i ← 1
    While i ≤ n execute
        count ← 1
        While i + count ≤ n and x[i] = x[i +
count] execute
            count ← count + 1
        EndWhile
        If count MOD k ≠ 0 then
            scrie x[i]
        EndIf
        i ← i + count
    EndWhile
EndAlgorithm
```

51. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Apelul algoritmului `A(x,n)` nu afectează rezultatul afișat de algoritmul `Afiseaza(x,n,k)`, indiferent de parametri;

B. Apelul algoritmului `A(x,n)` nu afectează rezultatul afișat de algoritmul `Afiseaza(x,n,k)`, dacă și numai dacă șirul `x` este ordonat crescător;

C. Algoritmul `A(x,n)` sortează crescător șirul `x`;

D. Algoritmul `A(x,n)` sortează descrescător șirul `x`.

52. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Pentru $x = [1, 4, 1, 2, 3, 3, 2]$ și $k = 2$, rezultatul afișat de `Afiseaza(x,7,2)` este 4;

B. Pentru $x = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]$ și $k = 3$, rezultatul afișat de `Afiseaza(x,10,3)` este $1, 2, 4$;

C. Pentru $x = [5, 6, 5, 7, 6, 6, 7]$ și $k = 2$, rezultatul afișat de `Afiseaza(x,7,2)` este $5, 7$;

D. Pentru $x = [2, 2, 4, 4, 4, 6, 4, 5]$ și $k = 2$, rezultatul afișat de `Afiseaza(x,8,2)` este $4, 6$.

53. Se consideră algoritmul `Peak(x,n)`, unde n este un număr natural ($3 \leq n \leq 10^3$), x este un vector cu n elemente numere întregi $(x[1], x[2], ..., x[n])$, unde $-10^4 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$: ✓ ?

```
Algorithm Peak(x, n)
    i ← 1
    While i < n and x[i] < x[i + 1] execute
        i ← i + 1
    EndWhile
    If i = 1 or i = n then
        Return False
    EndIf
    While i < n and x[i] > x[i + 1] execute
        i ← i + 1
    EndWhile
    Return i = n
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $x = [1, 2]$, rezultatul returnat de `Peak(x,2)` este **True**;

B. Pentru $x = [1, 4, 3, 2, 1]$, rezultatul returnat de `Peak(x,5)` este **True**;

C. Pentru $x = [1, 3, 3, 2, 1]$, rezultatul returnat de `Peak(x,5)` este **False**;

D. Pentru $x = [1, 2, 3, 4]$, rezultatul returnat de `Peak(x,3)` este **True**.

54. Se consideră algoritmul `Down(x,n)`, unde n este un număr natural ($3 \leq n \leq 10^3$), x este un vector cu n elemente numere întregi $(x[1], x[2], ..., x[n])$, unde $-10^4 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$: ✓ ?

```
Algorithm Down(x, n)
    i ← 1
    While i < n and x[i] > x[i + 1] execute
        i ← i + 1
    EndWhile
    If i = 1 or i = n then
        Return False
    EndIf
    While i < n and x[i] < x[i + 1] execute
```

```
        i ← i + 1
    EndWhile
    If i = n then
        Return True
    Else
        Return False
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $x = [4, 2, 3, 5]$, rezultatul returnat de `Down(x,4)` este **True**;

B. Pentru $x = [4, 3, 2, 1]$, rezultatul returnat de `Down(x,4)` este **True**;

C. Pentru $x = [1, 2, 3, 4]$, rezultatul returnat de `Down(x,4)` este **False**;

D. Pentru $x = [4, 3, 3, 2, 1]$, rezultatul returnat de `Down(x,5)` este **True**.

55. Se consideră algoritmul `Get(a,b,n,m)`, unde n și m sunt lungimile sirurilor de caractere a și b, respectiv, iar a și b sunt două șiruri de caractere formate din litere mici ale alfabetului englez $(a[1], a[2], ..., a[n])$ și $(b[1], b[2], ..., b[m])$: Presupunem că avem definit următorul subalgoritm: `ascii(c)` - returnează codul ASCII al caracterului `c`. Presupunem că operațiile aritmetice nu produc depășire pe mulțimea numerelor întregi. ✓ ?

```
Algorithm GET(a, b, n, m)
    i ← 1
    While i ≤ n and i ≤ m and a[i] = b[i] execute
        i ← i + 1
    EndWhile
    Return i − 1
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $a =$ `abcdef`, $b =$ `abcxyz`, rezultatul returnat de `Get(a,b,6,6)` este 3;

B. Pentru $a =$ `hello`, $b =$ `world`, rezultatul returnat de `Get(a,b,5,5)` este 1;

C. Pentru $a =$ `test`, $b =$ `testcase`, rezultatul returnat de `Get(a,b,4,8)` este 4;

D. Pentru $a =$ `abcd`, $b =$ `abef`, rezultatul returnat de `Get(a,b,4,4)` este 2.

56. Se dă un șir a de n caractere din alfabetul englez, $(a[1], a[2], ..., a[n])$. Dorim să aflăm cea mai lungă subsecvență a sa care este palindrom. Care dintre următoarele implementări returnează lungimea acestei subsecvențe la apelul `LongestPalindrome(a,n)`? ✓ ?

A.

```
Algorithm LONGESTPALINDROME(a, n)
    maxLength ← 1
    For i ← 1,n execute
        len1 ← EXPANDAROUNDCENTER(a, n, i, i)
        len2 ← EXPANDAROUNDCENTER(a, n, i, i+1)
        If len1 > maxLength then
            maxLength ← len1
        EndIf
        If len2 > maxLength then
```

```
            maxLength ← len2
         EndIf
      EndFor
      Return maxLength
   EndAlgorithm
   Algorithm ExpandAroundCenter(a, n, l, r)
      While l ≥ 1 and r ≤ n and a[l] = a[r] execute
         l ← l − 1
         r ← r + 1
      EndWhile
      Return r − l − 1
   EndAlgorithm
```

B.

```
   Algorithm LongestPalindrome(a, n)
      maxLength ← 0
      For i ← 1 to n execute
         For j ← i + 1 to n execute
            If a[i] = a[j] then
               If j − i + 1 > maxLength then
                  maxLength ← j − i + 1
               EndIf
            EndIf
         EndFor
      EndFor
      Return maxLength
   EndAlgorithm
```

C.

```
   Algorithm LongestPalindrome(a, n)
      maxLength ← 1
      start ← 1
      For i ← 1,n execute
         dp[i][i] ← True
      EndFor
      For i ← 1,n − 1 execute
         If a[i] = a[i + 1] then
            dp[i][i + 1] ← True
            start ← i
            maxLength ← 2
         EndIf
      EndFor
      For len ← 3,n execute
         For i ← 1,n − len + 1 execute
            j ← i + len − 1
            If a[i] = a[j] and dp[i + 1][j − 1] = True then
               dp[i][j] ← True
               If len > maxLength then
                  maxLength ← len
                  start ← i
               EndIf
            EndIf
         EndFor
      EndFor
      Return maxLength
```

```
    EndAlgorithm
```

D. Niciuna dintre variante.

57. Se consideră algoritmul `Compute(m,n)`, unde n este dimensiunea unei matrici pătratice ✓ ? m de dimensiune n x n, iar m este o matrice cu elemente întregi ($m[1][1], m[1][2], ..., m[n][n]$):

```
Algorithm COMPUTE(m, n)
    p1 ← 1
    p2 ← 1
    For i ← 1 to n execute
        p1 ← p1 · m[i][i]
        p2 ← p2 · m[i][n − i + 1]
    EndFor
    Return p1 · p2
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $m = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, rezultatul returnat de `Compute(m,2)` este 11;

B. Pentru $m = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$, rezultatul returnat de `Compute(m,3)` este 23040;

C. Pentru $m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, rezultatul returnat de `Compute(m,3)` este 1;

D. Pentru $m = \begin{bmatrix} 3 & 1 & 2 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, rezultatul returnat de `Compute(m,3)` este 11430.

Problemele **58.** și **59.** se referă la următorii 2 algoritmi:
Se consideră algoritmul algoritmul `B(m,n)`, unde n este dimensiunea unei matrici pătratice m de dimensiune n x n, iar m este o matrice cu elemente întregi ($m[1][1], m[1][2]$, $..., m[n][n]$), și algoritmul `A(m,n,i,s1,s2)`, unde m, n au aceleași restricții ca pentru algoritmul `B(m,n)`, iar *i*,*s1*,*s2* sunt numere întregi.

```
Algorithm A(m, n, i, s1, s2)
    If i = n + 1 then
        Return s1 − s2
    EndIf
    x ← 0
    For j ← 1,n execute
        x ← x + m[i][j]
    EndFor
    If i MOD 2 = 0 then
        s1 ← s1 + x
    Else
        s2 ← s2 + x
    EndIf
    Return A(m, n, i + 1, s1, s2)
EndAlgorithm
```

```
Algorithm B(m, n)
   Return A(m, n, 1, 0, 0)
EndAlgorithm
```

58. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Pentru $m = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, rezultatul returnat de `B(m,2)` este 4;

B. Pentru $m = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$, rezultatul returnat de `B(m,3)` este -18;

C. Pentru $m = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, rezultatul returnat de `B(m,3)` este 0;

D. Pentru $m = \begin{bmatrix} 10 & 20 & 30 \\ 5 & 15 & 25 \\ 40 & 50 & 60 \end{bmatrix}$, rezultatul returnat de `B(m,3)` este 65.

59. Dacă în subalgoritmul `A(m,n,i,s1,s2)`, înlocuim apelul *Return* $A(m, n, i+1, s1, s2)$ cu *Return* $A(m, n, i-1, s1, s2)$, ce ar trebui modificat în subalgoritmul `B(m,n)` pentru a oferi același rezultat? ✓ ?

A. Înlocuim *Return* $A(m, n, 1, 0, 0)$ cu *Return* $A(m, n, 0, n, 0)$;

B. Înlocuim *Return* $A(m, n, 1, 0, 0)$ cu *Return* $A(m, n, 0, n, n-1)$;

C. Înlocuim *Return* $A(m, n, 1, 0, 0)$ cu *Return* $A(m, n, 0, 0, n)$;

D. Înlocuim *Return* $A(m, n, 1, 0, 0)$ cu *Return* $A(m, n, n, 0, 0)$;

60. Se consideră algoritmii `H(m,mx,nx)` și `G(m,mx,nx)`, unde m este o matrice cu mx linii și nx coloane $(m[1][1], m[1][2], ..., m[mx][nx])$, mx și nx sunt numere naturale mai mari decât 2: ✓ ?

```
Algorithm H(m, mx, nx)
   For j ← 1,nx execute
      temp ← m[mx − 2][j]
      m[mx − 2][j] ← m[mx − 1][j]
      m[mx − 1][j] ← temp
   EndFor
   For i ← 1,mx execute
      temp ← m[i][nx − 2]
      m[i][nx − 2] ← m[i][nx − 1]
      m[i][nx − 1] ← temp
   EndFor
   Return m
EndAlgorithm
```

```
Algorithm G(m, mx, nx)
   For i ← 1,mx execute
      temp ← m[i][nx − 2]
      m[i][nx − 2] ← m[i][nx − 1]
      m[i][nx − 1] ← temp
   EndFor
   For j ← 1,nx execute
      temp ← m[mx − 2][j]
      m[mx − 2][j] ← m[mx − 1][j]
      m[mx − 1][j] ← temp
   EndFor
   Return m
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Ambii algoritmi au același efect asupra unei matrici m;

B. Pentru $m = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$, rezultatul returnat de `H(m,4,4)` este $\begin{bmatrix} 1 & 2 & 4 & 3 \\ 5 & 6 & 8 & 7 \\ 13 & 14 & 16 & 15 \\ 9 & 10 & 12 & 11 \end{bmatrix}$;

C. Algoritmul `H(m,mx,nx)` returnează un rezultat diferit față de rezultatul returnat de algoritmul `G(m,mx,nx)`;

D. Pentru $m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, rezultatul returnat de `G(m,3,3)` este $\begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{bmatrix}$.

61. Se consideră algoritmul `Cauta(a, n , i, j)`, unde s este un vector cu n elemente, ✓ ? inițial egale cu 0, iar a un șir de numere întregi $(a[1], a[2], ..., a[n])$ cu n elemente, $1 \leq i \leq j < n + 1$:

```
Algorithm CAUTA(a, n, i, j)
    s ← CONSTRUIESTE(a, n)
    If i = 1 then
        Return s[j]
    Else
        Return s[j] − s[i − 1]
    EndIf
EndAlgorithm
Algorithm CONSTRUIESTE(a, n)
    s[1] ← a[1]
    For k ← 2,n execute
        s[k] ← s[k − 1] + a[k]
    EndFor
    Return s
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $a = [1, 2, 3, 4, 5]$ și $i = 2, j = 4$, rezultatul returnat de `Cauta(a,5,2,4)` este 9;

B. Pentru $a = [1, 1, 1, 1, 1]$ și $i = 1, j = 4$, rezultatul returnat de `Cauta(a,5,1,4)` este 5;

C. Algoritmul returnează suma elementelor dintre indicii i și j, inclusiv, ai vectorului a;

D. Algoritmul returnează suma elementelor dintre indicii i și j, fără $a[i], a[j]$, ai vectorului a.

62. Se consideră algoritmul `Find(m, n)`, unde n este dimensiunea unei matrici pătratice ✓ ? m de dimensiune n x n.

```
Algorithm FIND(m, n)
    For i ← 1, n execute
        For j ← i + 1, n execute
            If m[i][j] ≠ m[j][i] then
```

```
            Return False
          EndIf
        EndFor
    EndFor
    Return True
EndAlgorithm
```

Ce face algoritmul?

A. Verifică dacă toate elementele de pe marginea matricei sunt egale;

B. Verifică dacă toate elementele de pe diagonala principală sunt egale;

C. Verifică dacă matricea este simetrică față de diagonala secundară;

D. Verifică dacă matricea este simetrică față de diagonala principală.

63. Se consideră algoritmul `bin(arr, n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$) și arr este un șir binar de n elemente. ✓ ?

```
Algorithm BIN(arr, n)
    cont ← 0
    For i ← 1, n − 1 execute
        cont ← cont + arr[i] * arr[i + 1]
    EndFor
    Return cont
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `bin([0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1], 20)`, algoritmul returnează 6.

B. Pentru $n = 11$, există exact 233 șiruri binare arr cu proprietatea că apelul `bin(arr, n)` returnează 0.

C. Pentru $n = 6$, există exact 20 șiruri binare arr cu proprietatea că apelul `bin(arr, n)` returnează 0.

D. Pentru $n = 8$, există exact 201 șiruri binare arr cu proprietatea că apelul `bin(arr, n)` returnează o valoare diferită de 0.

64. Fie vectorul `v = [3, 5, 2, 2, 7, 9, 1, 1, 4, 6]`. Primul element este situat pe poziția 0. Care este valoarea expresiei următoare? ✓ ?

$$v[2 * v[2 * v[3] - 1] + 1] + v[2 * v[7 - 5] - 1] + v[3 - v[2 * v[2] - 3] + 6]$$

A. 11

B. 24

C. 18

D. Niciuna din variantele de mai sus.

65. Se consideră algoritmul `Algo(a, n, m, k)`, unde a este un tablou bidimensional cu dimensiunea $n \times m$ ($1 \leq n, m \leq 10^3$), unde n și m sunt numere naturale iar fiecare element din a este un număr natural. Parametrul k este un număr natural ($1 \leq k \leq \min(n, m)$). Fie `x` o matrice de dimenisune $n \times m$, unde fiecare element este inițializat cu 0. ✓ ?

```
Algorithm ALGO(a, n, m, k)
   max ← 0
   For i ← 1, n execute
      For j ← 1, m execute
         If CHECK(a[i][j]) then
            x[i][j] ← a[i][j]
         EndIf
      EndFor
   EndFor
   For i ← 1, n − k + 1 execute
      For j ← 1, m − k + 1 execute
         sum ← 0
         For p ← 0, k − 1 execute
            For q ← 0, k−1 execute
               sum ← sum +
                 x[i + p][j + q]
            EndFor
         EndFor
         If sum > max then
            max ← sum
         EndIf
      EndFor
   EndFor
   Return max
EndAlgorithm
```

```
Algorithm CHECK(num)
   If num ≤ 1 then
      Return False
   EndIf
   For i ← 2, √num execute
      If num MOD i = 0 then
         Return False
      EndIf
   EndFor
   Return True
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Complexitatea algoritmului este $O(n \cdot m \cdot k^2)$.

B. Algoritmul `Algo` determină suma maximă a elementelor prime dintr-o submatrice $k \times k$ din a.

C. Pentru apelul `f(a, 5, 5, 3)` și matricea $a = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 7 & 9 & 11 \\ 4 & 5 & 7 & 10 & 13 \\ 5 & 9 & 13 & 17 & 4 \\ 6 & 11 & 16 & 4 & 9 \end{bmatrix}$ algoritmul va returna valoarea 62.

D. Algoritmul verifică dacă există o submatrice $k \times k$ în care toate elementele sunt numere prime și returnează suma acestora.

66. Fie vectorul `v = [4, 7, 1, 9, 6, 3, 8, 5, 5, 2]`. Primul element este situat pe poziția 0. Care este valoarea expresiei următoare? ✓ ?

$$v[2 * v[4] \text{ MOD } 3] + v[v[7] \text{ DIV } 2 - 1] * v[3 - v[2] + 1]$$

A. 42

B. 67

C. Același rezultat ca al expresiei $v[v[9] * v[4] \text{ MOD } v[5]] + v[v[8] \text{ DIV } v[9] - v[2]] * v[3 - v[2] + v[2]]$.

D. Același rezultat ca al expresiei $v[v[8] \text{ MOD } v[5]*3]*v[3]-v[v[9]]-v[0]-v[v[2] \text{ DIV } 3]* v[4 - v[5]]$.

Algoritmi elementari

Acest capitol acoperă

- Baze de numerație
- Divizibilitate
- Determinare minim/maxim
- Căutări secvențiale/binare pe șiruri de numere
- Interclasare
- Metode de sortare

4.1 Teorie

4.1.1 Baze de numerație

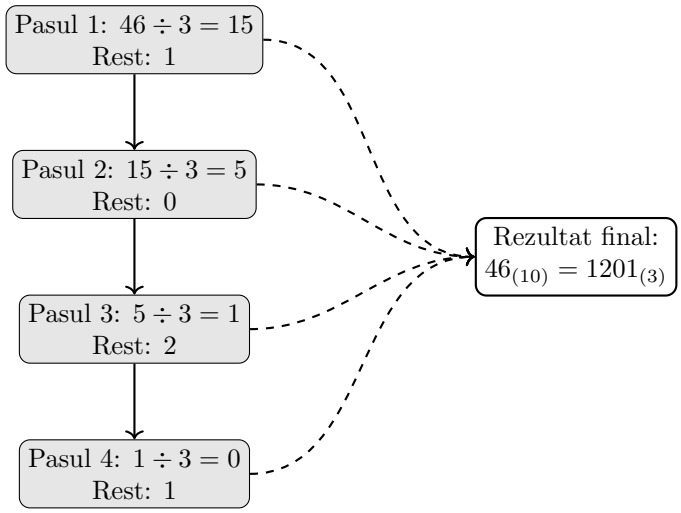
Cu excepția bazei 10, alte baze de numerație utilizate frecvent sunt:

- **Baza 2 (binar)**: cifrele 0 și 1, utilizată în computere.
- **Baza 8 (octal)**: cifrele 0-7, folosită pentru simplificarea reprezentării binare.
- **Baza 16 (hexazecimal)**: cifrele 0-9 și literele $A - F (A = 10, B = 11, ..., F = 15)$.
- **Baza b (caz general)**: cifrele 0, 1, ..., $b - 1$

Transformarea din baza 10 în baza b

Transformarea unui număr n din baza 10 în baza b, se realizează utilizând metoda împărțirii repetate, conform următorilor pași:

- Împarte cu rest numărul n la baza b.
- Notează restul (care va deveni o cifră în baza b).
- Împarte câtul obținut anterior la baza b.
- Repetă până când câtul devine 0.
- Resturile obținute, scrise în ordinea inversă obținerii, reprezintă scrierea în baza b a lui n.



Transformare iterativă din baza 10 în baza b:

```
1:  Algorithm ITERATIV(n, b)
2:      poz ← 0
3:      While n > 0 execute
4:          poz ← poz + 1
5:          cifre[poz] ← n MOD b
6:          n ← n DIV b
7:      EndWhile
8:      For i ← poz, 1, −1 execute
9:          Write cifre[i]
10:     EndFor
11: EndAlgorithm
```

Complexitate: $O(\log(n))$

Transformare recursivă din baza 10 în baza b:

```
1: Algorithm RECURSIV(n, b)
2:     If n > 0 then
3:         recursiv(n DIV b, b)
4:         Write n MOD b
5:     EndIf
6: EndAlgorithm
```

Complexitate: $O(\log(n))$

Transformarea din baza b în baza 10

Pentru a converti un număr din baza b în baza 10, folosim o formulă care exprimă numărul ca o sumă de puteri ale bazei b. Fie un număr $c_k c_{k-1} \dots c_1 c_{0(b)}$, atunci conversia acestuia în baza 10 se face conform formulei:

$$c_k c_{k-1} \dots c_1 c_{0(b)} = c_k \cdot b^k + c_{k-1} \cdot b^{k-1} + \dots + c_1 \cdot b^1 + c_0 \cdot b^0$$

`Exemplu:`

$$\begin{aligned} 11010001_{(2)} &= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 \\ &= 128 + 64 + 0 + 16 + 0 + 0 + 0 + 1 \\ &= 128 + 64 + 16 + 1 \end{aligned}$$

$$= 209$$

Astfel, rezultatul este:

$$11010001_{(2)} = 209_{(10)}$$

Transformare iterativă din baza b în baza 10:

```
1: Algorithm ITERATIV(n, b)
2:     result ← 0
3:     p ← 1
4:     While n > 0 execute
5:         result ← result + (n MOD 10) * p
6:         p ← p * b
7:         n ← n DIV 10
8:     EndWhile
9:     Return result
10: EndAlgorithm
```

Transformare recursivă din baza b în baza 10:

```
1: Algorithm RECURSIV(n, b)
2:     If n = 0 then
3:         Return 0
4:     EndIf
5:     Return (n MOD 10) + recursiv(n DIV 10, b) * b
6: EndAlgorithm
```

Transformarea din baza b în baza d

Pentru transformarea unui număr din baza b în baza d, procesul poate fi realizat în două etape principale:

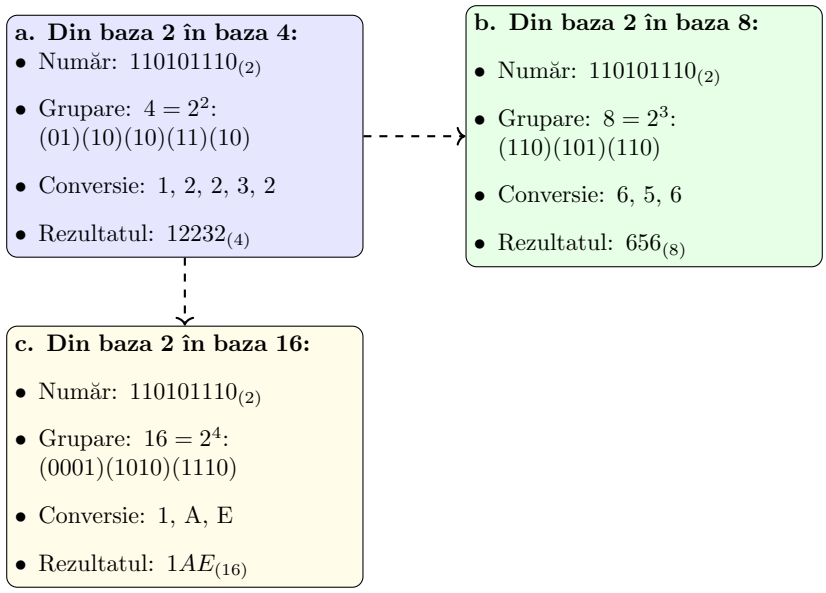
- 1. Conversia numărului din baza b în baza 10.
- 2. Conversia rezultatului din baza 10 în baza d.

4.1.2 Conversii rapide

Transformările rapide între baze care sunt puteri ale lui 2 (cum ar fi 2, 4, 8 sau 16) se bazează pe gruparea cifrelor binare în subgrupuri specifice. Această abordare permite conversia directă a unui număr din baza 2 în alte baze fără a trece prin baza 10.
1. Conversia din baza 2 în baza 4, 8, sau 16 se realizează în următorul mod:

- Grupare: Împărțim numărul binar în grupuri de câte 2 cifre (pentru baza 4), 3 cifre (pentru baza 8) sau 4 cifre (pentru baza 16) cifre. Dacă numărul de cifre binare nu este multiplu al dimensiunii grupului, se completează cu zerouri în partea stângă.
- Conversie: Fiecare grup de cifre binare se transformă direct în cifra corespunzătoare din baza dorită.

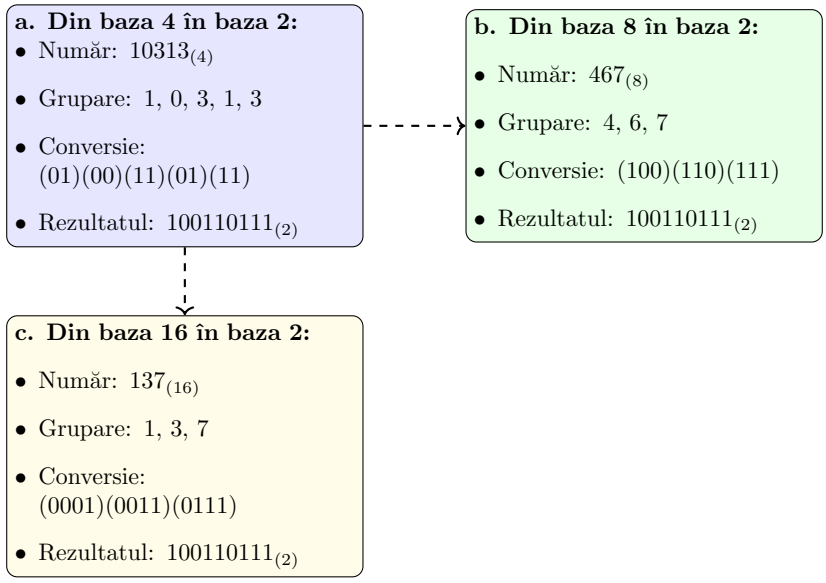
Exemplu:



2. Conversia din baza 4, 8, sau 16 în baza 2:

   - Fiecare cifră din baza respectivă se transformă în echivalentul său binar, folosind 2 (pentru baza 4), 3 (pentru baza 8) sau 4 (pentru baza 16) cifre binare pentru fiecare cifră din baza respectivă.

Exemplu:



4.1.3 Divizibilitate

Divizibilitatea se referă la proprietatea unui număr a de a fi împărțit exact la un alt număr b (notat $b \mid a$).

- **Divizor:** Un număr care împarte exact alt număr, adică restul împărțirii întregi este 0. De exemplu, 2 este divizor al lui 8.
- **Multiplu:** Rezultatul obținut prin înmulțirea unui număr întreg cu alt număr. De exemplu, 12 este multiplu al lui 3.
- **Rest:** Numărul rămas după împărțire. De exemplu, restul împărțirii lui 10 la 3 este 1.

Divizorii unui număr

Un număr d este divizor al lui n dacă $n : d = k$, unde k este un număr întreg. Exemplu: Pentru $n = 12$, divizorii sunt 1, 2, 3, 4, 6, 12.

Exemplu 1: Afișarea divizorilor unui număr

```
1: Algorithm AFISARE(n)
2:     For d ← 1, n execute
3:         If n MOD d = 0 then
4:             Write d, ' '
5:         EndIf
6:     EndFor
7: EndAlgorithm
```

Complexitate: $O(n)$

Exemplu 2: Afișarea divizorilor unui număr

```
1: Algorithm AFISARE(n)
2:     For d ← 1, d * d ≤ n execute
3:         If n MOD d = 0 AND d *
   d < n then
4:             Write d, ' ', n DIV d, ' '
5:         Else
6:             If d * d = n then
7:                 Write d
8:             EndIf
9:         EndIf
10:    EndFor
11: EndAlgorithm
```

Complexitate: $O(\sqrt{n})$

Observații importante

- Divizorii unui număr sunt simetrici: dacă d este un divizor al lui n, atunci n DIV d este de asemenea un divizor.
- Numerele care nu sunt pătrate perfecte au număr par de divizori.
- Singurele numere cu număr impar de divizori sunt pătratele perfecte.
- Cel mai mic divizor propriu al unui număr natural (diferit de 1 și de numărul însuși) este întotdeauna un număr prim.

CMMDC (Cel Mai Mare Divizor Comun)

Cel mai mare divizor comun (CMMDC) al două numere naturale a și b este cel mai mare număr natural care divide atât pe a, cât și pe b. CMMDC al numerelor a și b se notează (a, b). Dacă $(a, b) = 1$, spunem că a și b sunt prime între ele.

Descompunerea în factori primi sau Algoritmul lui Euclid reprezintă metode prin care putem calcula CMMDC.

Algoritmul lui Euclid prin scăderi: Cel mai mare divizor a două numere divide și diferența acestora.

- Dacă $a = b$, CMMDC este a sau b.
- Dacă $a > b$, se înlocuiește a cu $a - b$.
- Dacă $a < b$, se înlocuiește b cu $b - a$.
- Se repetă pașii 2 și 3 cât timp $a \neq b$. Rezultatul final va fi CMMDC.

Algoritmul lui Euclid prin scăderi, iterativ

```
Algorithm ITERATIV(a, b)
    While a ≠ b execute
        If a > b then
            a ← a − b
        Else
            b ← b − a
        EndIf
    EndWhile
    Return a
EndAlgorithm
```

Complexitate: $O(\min(a, b))$

Algoritmul lui Euclid prin scăderi, recursiv

```
Algorithm RECURSIV(a, b)
    If a = b then
        Return a
    Else
        If a > b then
            Return recursiv(a − b, b)
        Else
            Return recursiv(a, b − a)
        EndIf
    EndIf
EndAlgorithm
```

Complexitate: $O(\min(a, b))$

Algoritmul lui Euclid prin împărțiri (cu resturi): Cel mai mare divizor a două numere divide și restul împărțirii acestora, conform teoremei împărțirii cu rest.

- Dacă $b = 0$, CMMDC este a.
- Dacă $b \neq 0$, calculează $r = a \mod b$.
- Înlocuiește a cu b și b cu r.
- Repetă pașii 2-3 cât timp $b \neq 0$. CMMDC este valoarea finală a lui a.

Exemplu: Algoritmul lui Euclid pentru a = 56 și b = 42:

Algoritmul lui Euclid prin împărțiri, iterativ

```
Algorithm ITERATIV(a, b)
    While b > 0 execute
        r ← a MOD b
        a ← b
        b ← r
    EndWhile
    Return a
EndAlgorithm
```

Complexitate: $O(\log(\min(a, b)))$

Algoritmul lui Euclid prin împărțiri, recursiv

```
Algorithm RECURSIV(a, b)
    If b = 0 then
        Return a
    Else
        Return recursiv(b, a MOD b)
    EndIf
EndAlgorithm
```

Complexitate: $O(\log(\min(a, b)))$

CMMMC (Cel Mai Mic Multiplu Comun)

Cel mai mic multiplu comun (CMMMC) al două sau mai multe numere întregi este cel mai mic număr întreg pozitiv care este divizibil cu toate numerele respective. Pentru două numere a și b, CMMMC se calculează folosind formula:

$$\text{CMMMC}(a, b) = \frac{a \cdot b}{\text{CMMDC}(a, b)},$$

unde CMMDC este cel mai mare divizor comun.

Primalitate

Un număr prim este un număr întreg pozitiv mai mare decât 1 care are exact doi divizori: 1 și el însuși.
Pentru a verifica algoritmic dacă un număr n este prim:

- Presupunem că n este prim.
- Dacă $n \leq 1$, atunci nu este prim.
- Căutăm un divizor între 2 și $\sqrt{n}$.
- Dacă n nu este divizibil cu niciunul dintre aceste numere, atunci este prim.
- Dacă găsim un divizor d, atunci n nu este prim.

Metoda 1: Verificare dacă un număr este prim

```
1:  Algorithm PRIMALITATE(n)
2:      If n ≤ 1 then
3:          Return False
4:      EndIf
5:      For d ← 2, n − 1 execute
6:          If n MOD d = 0 then
7:              Return False
8:          EndIf
9:      EndFor
10:     Return True
11: EndAlgorithm
```

Complexitate: $O(n)$

Metoda 2: Verificare dacă un număr este prim

```
1:  Algorithm PRIMALITATE(n)
2:      If n ≤ 1 then
3:          Return False
4:      EndIf
5:      If n MOD 2 = 0 AND n > 2 then
6:          Return False
7:      EndIf
8:      For d ← 3, d ∗ d ≤ n, 2 execute
9:          If n MOD d = 0 then
10:             Return False
11:         EndIf
12:     EndFor
13:     Return True
14: EndAlgorithm
```

Complexitate: $O(\sqrt{n})$

Observație importantă

- Putem stabili dacă un număr n este prim și prin următoarele feluri:
  - Numărăm divizorii lui n. Dacă are exact 2 divizori, atunci n este prim.
  - Determinăm suma divizorilor lui n. Dacă suma este egală cu $n + 1$, numărul este prim.

Descompunerea în factori primi

Descompunerea în factori primi este procesul prin care un număr întreg pozitiv este exprimat ca produs de numere prime, posibil ridicate la o putere. De exemplu, numărul 60 poate fi descompus în factori primi astfel:

$$60 = 2^2 \cdot 3^1 \cdot 5^1$$

Pentru a descompune un număr întreg pozitiv n în factori primi, se urmează pașii următori:

- Începem cu cel mai mic număr prim, 2.
- Împărțim n la 2. Dacă n este divizibil cu 2, notăm 2 ca factor și împărțim n cu 2.
- Continuăm împărțirea lui n la 2 până când n nu mai este divizibil cu 2.
- Trecem la următorul număr prim (3, 5, 7, etc.) și repetăm procesul până când n devine egal cu 1.

Descompunerea în factori primi

```
1:  Algorithm FACTORIZARE(n)
2:      d ← 2
3:      While n > 1 execute
4:          putere ← 0
5:          While n MOD d = 0 execute
6:              putere ← putere + 1
7:              n ← n DIV d
8:          EndWhile
9:          If putere > 0 then
10:             Write d, ^, putere
11:             Write new line
12:         EndIf
13:         d ← d + 1
14:         If d * d > n then
15:             d ← n
16:         EndIf
17:     EndWhile
18: EndAlgorithm
```

Complexitate: $O(\sqrt{n})$

4.1.4 Şirul lui Fibonacci

Șirurile cu termen general sunt frecvent întâlnite în matematică, însă în informatică ele sunt adesea utilizate pentru implementarea structurilor repetitive sau a programelor recursive. Cu toate acestea, Șirul lui Fibonacci nu este un șir oarecare cu termen general, deoarece al n-lea termen al său este dependent de termenii $n-1$ și $n-2$. Astfel, pentru Șirul lui Fibonacci, sunt valabile următoarele precizări: fiecare termen n este suma termenilor $n-1$ și $n-2$, iar șirul începe cu termenii 0 și 1 (unii programatori consideră primii termeni ca fiind 1 și 1, dar ambele convenții sunt acceptate). În cele ce urmează, vom considera primul termen al șirului $f_0 = 0$. Deci, dacă $f_n, n \in \mathbb{N}$ este șirul lui Fibonacci, avem:

$$f_n = \begin{cases} 0, \text{ dacă } n = 0; \\ 1, \text{ dacă } n = 1; \\ f_{n-1} + f_{n-2}, \text{ dacă } n \geq 2. \end{cases}$$

Astfel, un posibil program care afișează primii n termeni din șirul lui Fibonacci și care folosește un vector este:

Șirul lui Fibonacci folosind un array:

```
1: Algorithm FIBO(n)
2:     f[0] ← 1
3:     f[1] ← 1
4:     For i ← 2, n − 1 execute
5:         f[i] = f[i − 1] + f[i − 2]
6:     EndFor
7:     For i ← 0, n − 1 execute
8:         afișează f[i]
9:     EndFor
10: EndAlgorithm
```

Această implementare este una simplă și intuitivă, dar nu este cea mai optimă din punct de vedere al complexității spațiale, deoarece folosește un vector de dimensiune n(complexitatea spațiu este $O(n)$). Un algoritm care afișează primii n termeni din șirul lui Fibonacci în complexitate spațiu $O(1)$ ar fi următorul, care folosește doar 3 variabile, numere naturale:

Șirul lui Fibonacci folosind trei variabile:

```
1: Algorithm FIBO(n)
2:     a ← 1
3:     b ← 1
4:     Write a, Write b
5:     For i ← 2, n − 1 execute
6:         aux ← a + b
7:         a ← b
8:         b ← aux
9:         Write aux
10:    EndFor
11: EndAlgorithm
```

Complexitatea de timp pentru afișarea primelor n numere din șirul lui Fibonacci este $O(n)$. Însă, verificarea dacă un număr x este în Șirul lui Fibonacci se poate implementa în complexitatea de timp $O(1)$, folosind următoarea proprietate:

$$x \text{ face parte din șirul Fibonacci} \Leftrightarrow$$

$$\Leftrightarrow 5x^2 + 4 \textbf{ este pătrat perfect sau } 5x^2 - 4 \textbf{ este pătrat perfect } .$$

4.1.5 Determinare minim/maxim

Determinarea maximului și minimului reprezintă procesul de identificare a valorii celei mai mari (maxim) sau celei mai mici (minim) dintr-un set de valori.

Maximul și minimul a două valori

Determinarea minimului unei valori

```
Algorithm MINIM(a, b)
    If a < b then
        Return a
    Else
        Return b
    EndIf
EndAlgorithm
```

Complexitate: $O(1)$

Determinarea maximului unei valori

```
Algorithm MAXIM(a, b)
    If a > b then
        Return a
    Else
        Return b
    EndIf
EndAlgorithm
```

Complexitate: $O(1)$

Determinarea minimului pentru un număr oarecare de valori

Algoritmul *primaValoare* va păstra în variabila *min* cu cea mai mică valoare citită (iniția, prima valoarea citită), în timp ce *valoareMare* inițializează variabila *min* cu o valoare suficient de mare pentru a permite găsirea minimului, ambii algoritmi terminându-se când se introduce valoarea 0.

Variabila *min* inițializată cu prima valoare citită

```
1:  Algorithm PRIMAVALOARE
2:      Read nr
3:      min ← nr
4:      Read nr
5:      While nr ≠ 0 execute
6:          If nr < min then
7:              min ← nr
8:          EndIf
9:          Read nr
10:     EndWhile
11:     Write min
12: EndAlgorithm
```

Complexitate: $O(n)$

Variabila *Min* inițializată cu o valoare mare

```
1:  Algorithm VALOAREMARE
2:      min ← ∞
3:      Read nr
4:      While nr ≠ 0 execute
5:          If nr < min then
6:              min ← nr
7:          EndIf
8:          Read nr
9:      EndWhile
10:     Write min
11: EndAlgorithm
```

Complexitate: $O(n)$

Determinarea maximului pentru un număr oarecare de valori

Algoritmul *primaValoare* actualizează variabila *max* cu prima valoare citită, în timp ce *valoareMica* inițializează variabila *max* cu o valoare suficient de mică pentru a permite găsirea maximului, ambii algoritmi terminându-se când se introduce 0.

Variabila Max inițializată cu cea mai mare valoarea citită

```
1:  Algorithm PRIMAVALOARE
2:      Read nr
3:      max ← nr
4:      Read nr
5:      While nr ≠ 0 execute
6:          If nr > max then
7:              max ← nr
8:          EndIf
9:          Read nr
10:     EndWhile
11:     Write max
12: EndAlgorithm
```

Complexitate: $O(n)$

Variabila Max inițializată cu o valoarea mică

```
1:  Algorithm VALOAREMICA
2:      max ← −999999
3:      Read nr
4:      While nr ≠ 0 execute
5:          If nr > max then
6:              max ← nr
7:          EndIf
8:          Read nr
9:      EndWhile
10:     Write max
11: EndAlgorithm
```

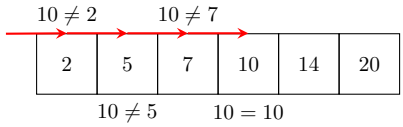
Complexitate: $O(n)$

4.1.6 Căutări pe șiruri de numere

Căutarea Secvenţială

Căutarea secvenţială (numită și *căutare liniară*) presupune parcurgerea element cu element a şirului până când fie se găseşte elementul căutat, fie se ajunge la finalul şirului fără a găsi elementul respectiv. Din punct de vedere al complexităţii timp, în cel mai bun caz (când elementul se află la începutul şirului), complexitatea căutării este $O(1)$, însă în medie şi în cel mai rău caz, complexitatea va fi $O(n)$, unde n este lungimea şirului.

Exemplu: Presupunem că avem şirul $A = [2, 5, 7, 10, 14, 20]$, iar elementul căutat este 10. Căutarea secvenţială va compara în ordine elementele: 2, 5, 7, 10, iar la al patrulea element găseşte valoarea căutată. Ilustrarea grafică a căutării lui 10 în A este aceasta:



Considerăm A - şir de numere, n - dimensiunea şirului și x - elementul căutat. O implementare în pseudocod a căutării secvențiale arată astfel:

Căutare Secvenţială

```
1: For i = 0, n − 1 execute
2:     If A[i] = x then
3:         Return i                          ▷ Returnăm poziția lui x
4:     EndIf
5: EndFor
6: Return −1                                 ▷ x nu a fost găsit
```

Căutarea Binară

Pentru a putea căuta un element într-un șir folosind Căutarea Binară, **șirul trebuie să fie sortat** (vezi următorul capitol referitor la Metode de Sortare). Ideea de bază este împărţirea şirului în două subintervale de dimensiune aproximativ egală. Astfel, la fiecare pas:

- Se calculează indicele mijlocului secvenței curente $\rightarrow m$;
- Dacă $A[m]$ este elementul căutat, returnăm m.
- Dacă $A[m] > x$, continuăm căutarea în subsecvența stângă $[st, m-1]$.
- Dacă $A[m] < x$, continuăm căutarea în subsecvența dreaptă $[m+1, dr]$.

Căutarea binară are o complexitate timp atât în cazul mediu, cât şi în cel mai rău caz de $O(\log n)$, unde n este lungimea şirului.

Observație: Complexitatea Căutării Binare nu include și complexitatea sortării șirului. Atunci când începem să căutăm binar un anumit element, șirul trebuie să fie deja sortat.

Exemplu: Considerăm acelaşi şir sortat: $A = [2, 5, 7, 10, 14, 20]$, unde vom căuta elementul 10.
Paşii sunt următorii:

- Interval iniţial: $st = 0$, $dr = 5$, $m = \lfloor (0+5)/2 \rfloor = 2$. Avem elementul $A[2] = 7$;
- Cum $7 < 10$, căutăm în secvența $[3, 5]$;
- Avem $m = \lfloor (3+5)/2 \rfloor = 4$ și elementul $A[4] = 14$;
- $14 > 10$, deci căutăm în secvența $[3, 3]$;
- $m = \lfloor (3+3)/2 \rfloor = 3$, $A[3] = 10$ și oprim căutarea.

Ilustrarea grafică a căutării lui 10 în A este aceasta:

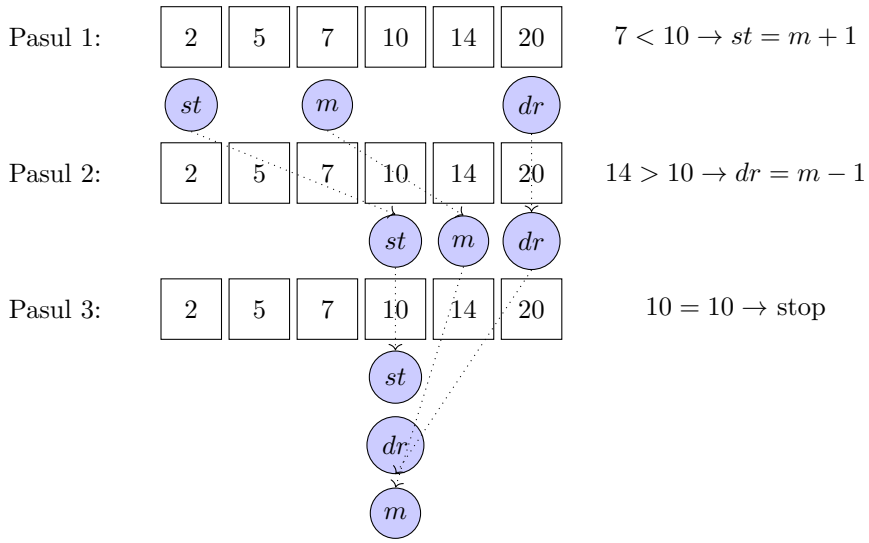


Figura 4.1 Căutare binară pentru elementul 10

Considerăm A - şir de numere, n - dimensiunea şirului și x - elementul căutat. O implementare în pseudocod a căutării secvențiale arată astfel:

Căutare Binara

```
1: While st ≤ dr execute
2:     m ← (st + dr) DIV 2
3:     If A[m] = x then
4:         Return m
5:     Else If A[m] > x then
6:         dr ← m − 1
7:     Else
8:         st ← m + 1
9:     EndIf
10: EndWhile
11: Return −1
```

4.1.7 Interclasarea

Interclasarea este o tehnică fundamentală utilizată pentru combinarea a două secvențe sortate într-o singură secvență sortată. Această metodă este folosită frecvent în algoritmi precum sortarea prin interclasare (Merge Sort) sau pentru alte aplicații care presupun manipularea datelor sortate.

Principiile Interclasării

- **Intrările:**
  - Două liste sortate (sau secvențe sortate)
- **Ieșirea:**

  - O singură listă sortată obținută prin combinarea celor două liste

- **Procesul:**
  - Compararea elementelor curente din cele două liste
  - Inserarea elementului mai mic în lista finală
  - Repetarea procesului până la epuizarea ambelor liste

Modelul Matematic al Interclasării

Considerăm două liste sortate $A = \{a_1, a_2, \ldots, a_m\}$ și $B = \{b_1, b_2, \ldots, b_n\}$. Interclasarea produce o listă $C = \{c_1, c_2, \ldots, c_{m+n}\}$ astfel încât C este de asemenea sortată:

$$C = \text{Merge}(A, B)$$

Algoritmul pentru Interclasare

Algoritmul de Interclasare

```
1:  Algorithm MERGE(A, B)
2:      i ← 1, j ← 1, k ← 1
3:      C ← listă goală
4:      While i ≤ |A| && j ≤ |B| execute        ▷ S-a folosit |A| = lungimea șirului A
5:          If A[i] ≤ B[j] then
6:              C[k] ← A[i]
7:              i ← i + 1
8:          Else
9:              C[k] ← B[j]
10:             j ← j + 1
11:         EndIf
12:         k ← k + 1
13:     EndWhile
14:     While i ≤ |A| execute
15:         C[k] ← A[i]
16:         i ← i + 1, k ← k + 1
17:     EndWhile
18:     While j ≤ |B| execute
19:         C[k] ← B[j]
20:         j ← j + 1, k ← k + 1
21:     EndWhile
22:     Return C
23: EndAlgorithm
```

Complexitatea Algoritmului

- **Complexitatea temporală:** $O(m+n)$, unde m și n sunt dimensiunile celor două liste de intrare. Acest lucru rezultă din faptul că fiecare element este comparat și inserat o singură dată.
- **Complexitatea spațială:** $O(m+n)$ pentru stocarea listei rezultate.

Aplicații

- **Sortarea prin interclasare:**
  - Algoritmul Merge Sort utilizează interclasarea pentru a combina subsecvențe sortate.
- **Îmbinarea datelor sortate:**
  - Utilizat în baze de date și fluxuri de date mari pentru a uni seturi sortate.

4.1.8 Metode de sortare

Ce sunt metodele de sortare?

Metodele de sortare ne ajută să aducem elementele unui tablou unidimensional (vector) într-o ordine **crescătoare** sau **descrescătoare**. Există mai multe metode de a realiza acest lucru, iar diferența majoră care ne va determina să alegem o anumită metodă va fi cu siguranță **complexitatea timp a metodei**. Atunci când procesăm un șir cu o lungime imensă, suntem interesați ca operația de sortare să fie făcută cât mai eficient din punct de vedere al timpului de execuție. În cele ce urmează, vom prezenta cele mai întâlnite metode de sortare în Informatica de liceu, iar la finalul prezentării vom face o scurtă comparație între eficiența acestora.

Metoda Bubble Sort

Bubble Sort este un algoritm de sortare care parcurge în mod repetat un șir de numere, comparând perechi de elemente adiacente și interschimbându-le dacă sunt în ordinea greșită față de cum ne dorim să sortăm șirul (crescător/descrescător). Procesul continuă până când șirul este complet ordonat, adică nu mai sunt necesare interschimbări.

Exemplu: Vom sorta șirul $[5, 1, 4, 2, 8]$ crescător, folosind metoda Bubble Sort:

- **Iterația 1:**
  - Comparăm 5 și 1: $5 > 1$, interschimbăm $\rightarrow \{1, 5, 4, 2, 8\}$
  - Comparăm 5 și 4: $5 > 4$, interschimbăm $\rightarrow \{1, 4, 5, 2, 8\}$
  - Comparăm 5 și 2: $5 > 2$, interschimbăm $\rightarrow \{1, 4, 2, 5, 8\}$
  - Comparăm 5 și 8: $5 < 8$, nu interschimbăm.
- **Iterația 2:**
  - Comparăm 1 și 4: $1 < 4$, nu interschimbăm.
  - Comparăm 4 și 2: $4 > 2$, interschimbăm $\rightarrow \{1, 2, 4, 5, 8\}$
  - Comparăm 4 și 5: $4 < 5$, nu interschimbăm.
  - Comparăm 5 și 8: $5 < 8$, nu interschimbăm.
- **Iterația 3:**
  - Comparăm 1 și 2: $1 < 2$, nu interschimbăm.
  - Comparăm 2 și 4: $2 < 4$, nu interschimbăm.
  - Comparăm 4 și 5: $4 < 5$, nu interschimbăm.

La finalul execuției, obținem șirul sortat crescător: $\{1, 2, 4, 5, 8\}$. O ilustrare grafică a rezultatelor în urma celor 3 iterații se regăsește mai jos:

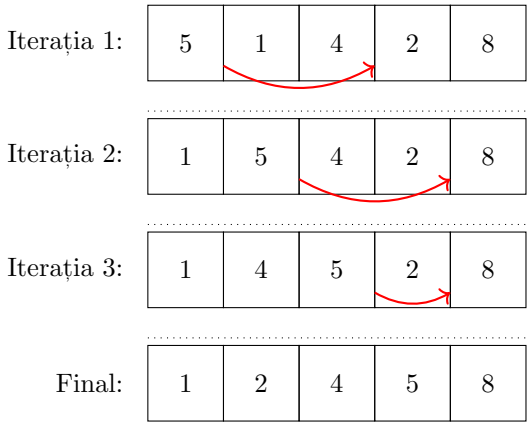


Figura 4.2 Sortare cu metoda Bubble Sort

O implementare a algoritmului în pseudocod arată astfel:

Bubble Sort

```
1: Do
2:     ok ← True
3:     For i ← 1, n − 1 execute
4:         If A[i] > A[i + 1] then
5:             Swap(A[i], A[i + 1])
6:             ok ← False
7:         EndIf
8:     EndFor
9: While NOT ok
```

Complexitate timp: $O(n^2)$ în cel mai rău caz, $O(n)$ în cel mai bun caz.

Metoda Selection Sort

Sortarea prin metoda Selection Sort funcționează prin găsirea în mod repetat a elementului minim din vectorul nesortat și mutarea lui pe poziția corespunzătoare în vectorul sortat.

Exemplu: Vom sorta șirul $\{29, 10, 14, 37, 13\}$ folosind Selection Sort:

- **Iterația 1:**
  - Găsim minimul în $\{29, 10, 14, 37, 13\}$, care este 10.
  - Interschimbăm 29 cu 10 $\rightarrow \{10, 29, 14, 37, 13\}$.
- **Iterația 2:**
  - Găsim minimul în $\{29, 14, 37, 13\}$, care este 13.
  - Interschimbăm 29 cu 13 $\rightarrow \{10, 13, 14, 37, 29\}$.

- **Iterația 3:**
  - Găsim minimul în $\{14, 37, 29\}$, care este 14.
  - Cum 14 este deja la locul său, nu interschimbăm.
- **Iterația 4:**
  - Găsim minimul în $\{37, 29\}$, care este 29.
  - Interschimbăm 37 cu 29 $\rightarrow \{10, 13, 14, 29, 37\}$.

Cum șirul este sortat, algoritmul se oprește în acest punct. O reprezentare grafică a pașilor se regăsește mai jos:

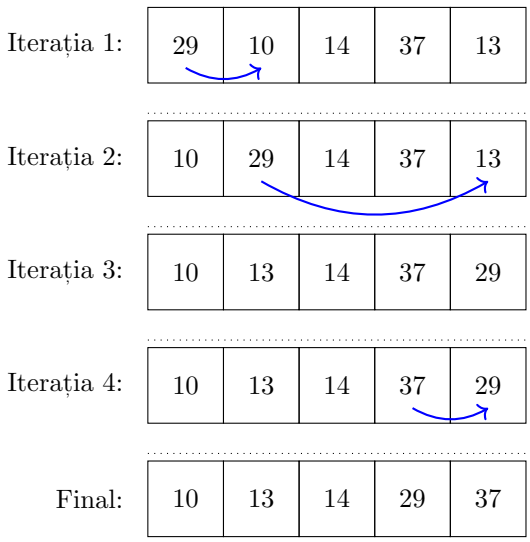


Figura 4.3 Sortare cu metoda Selection Sort

O implementare a algoritmului în pseudocod arată astfel:

Selection Sort

```
1:  For i ← 1, n − 1 execute
2:      min ← i
3:      For j ← i + 1, n execute
4:          If A[j] < A[min] then
5:              min ← j
6:          EndIf
7:      EndFor
8:      If min ≠ i then
9:          Swap(A[i], A[min])
10:     EndIf
11: EndFor
```

Complexitate: $O(n^2)$, atât în cel mai bun, cât și în cel mai rău caz.

Metoda Insertion Sort

Insertion Sort este un algoritm de sortare simplu și eficient pentru seturi mici de date. Funcționează pe principiul construirii treptate a șirului sortat, inserând fiecare element în poziția corectă. Algoritmul parcurge șirul de la stânga la dreapta și la fiecare pas, mută elementul curent în poziția corespunzătoare față de elementele deja sortate din partea stângă a șirului.

Exemplu: Vom sorta șirul $\{12, 11, 13, 5, 6\}$ folosind metoda Insertion Sort:

- **Iterația 1:**
  - Element curent: 11
  - Comparăm cu 12, deoarece $11 < 12$, mutăm 12 o poziție la dreapta.
  - Inserăm 11 pe poziția 1.
  - Șirul devine: $\{11, 12, 13, 5, 6\}$
- **Iterația 2:**
  - Element curent: 13
  - Comparăm cu 12, deoarece $13 \geq 12$, nu facem modificări.
  - Șirul rămâne: $\{11, 12, 13, 5, 6\}$
- **Iterația 3:**
  - Element curent: 5
  - Comparăm cu 13, $5 < 13$, mutăm 13 la dreapta.
  - Comparăm cu 12, $5 < 12$, mutăm 12 la dreapta.
  - Comparăm cu 11, $5 < 11$, mutăm 11 la dreapta.
  - Inserăm 5 pe poziția 0.
  - Șirul devine: $\{5, 11, 12, 13, 6\}$
- **Iterația 4:**
  - Element curent: 6
  - Comparăm cu 13, $6 < 13$, mutăm 13 la dreapta.
  - Comparăm cu 12, $6 < 12$, mutăm 12 la dreapta.
  - Comparăm cu 11, $6 < 11$, mutăm 11 la dreapta.
  - Comparăm cu 5, $6 \geq 5$, inserăm 6 pe poziția 1.
  - Șirul devine: $\{5, 6, 11, 12, 13\}$

La finalul execuției, obținem șirul sortat: $\{5, 6, 11, 12, 13\}$. O ilustrare grafică a pașilor algoritmului este prezentată mai jos:

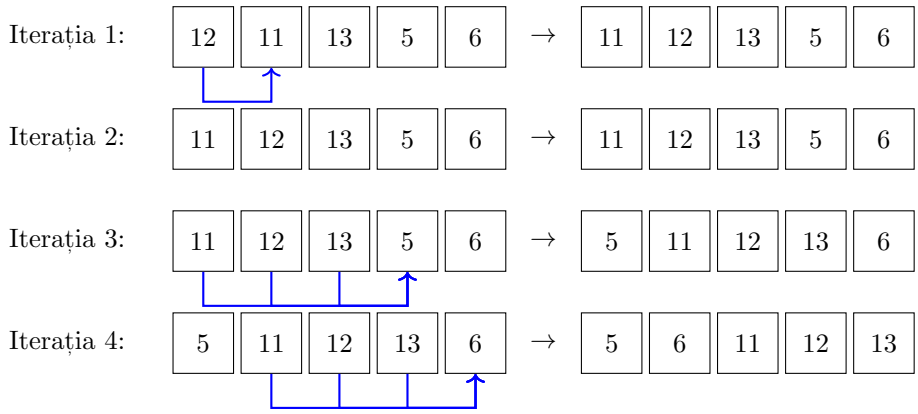


Figura 4.4 Sortare cu metoda Insertion Sort

O implementare a algoritmului în pseudocod arată astfel:

Insertion Sort

```
1: For i ← 1 to n − 1 execute
2:     key ← A[i]
3:     j ← i − 1
4:     While j ≥ 0 and A[j] > key execute
5:         A[j + 1] ← A[j]
6:         j ← j − 1
7:     EndWhile
8:     A[j + 1] ← key
9: EndFor
```

Complexitate: $O(n^2)$, atât în cel mai bun, cât și în cel mai rău caz.

Metoda Quick Sort

Quick Sort este un algoritm de sortare care utilizează strategia "Divide et Impera". Algoritmul funcționează prin selectarea unui element numit "pivot" și divizarea șirului astfel încât toate elementele mai mici decât pivotul să fie plasate înaintea acestuia, iar cele mai mari după el. Acest proces se repetă recursiv pentru subșirurile rezultate până când șirul este complet sortat.

Exemplu: Vom sorta șirul $\{33, 10, 55, 71, 29, 3, 18\}$ folosind metoda Quick Sort:

- **Pasul 1:**
  - Alegem pivotul: 33.
  - Divizăm șirul în funcție de pivot:
    - Elemente mai mici decât 33: $\{10, 29, 3, 18\}$
    - Pivot: $\{33\}$
    - Elemente mai mari decât 33: $\{55, 71\}$
- **Pasul 2:** Aplicăm recursiv Quick Sort pe subșirurile rezultate.

  - Sortăm $\{10, 29, 3, 18\}$
    - Pivot: 10
    - Elemente mai mici: $\{3\}$
    - Elemente mai mari: $\{29, 18\}$
  - Sortăm $\{3\}$ (deja sortat)
  - Sortăm $\{29, 18\}$
    - Pivot: 29
    - Elemente mai mici: $\{18\}$
    - Elemente mai mari: $\{\}$

- **Pasul 3:** Reunim subșirurile sortate.
  - Subșirul stâng sortat: $\{3, 10, 18, 29\}$
  - Pivotul inițial: $\{33\}$
  - Sortăm $\{55, 71\}$
    - Pivot: 55
    - Elemente mai mici: $\{\}$
    - Elemente mai mari: $\{71\}$
  - Subșirul drept sortat: $\{55, 71\}$

La final, obținem șirul sortat: $\{3, 10, 18, 29, 33, 55, 71\}$. O ilustrare grafică a procesului este prezentată mai jos:

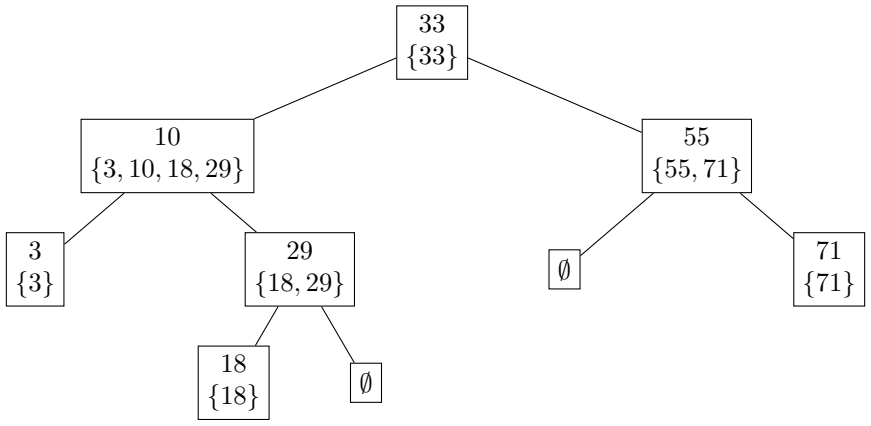


Figura 4.5 Sortare cu metoda Quick Sort

O implementare a algoritmului în pseudocod arată astfel:

Quick Sort

```
1: Algorithm QuickSort(A, low, high)
2:     If low < high then
3:         pivot_index ← Partition(A, low, high)
4:         QuickSort(A, low, pivot_index − 1)
5:         QuickSort(A, pivot_index + 1, high)
6:     EndIf
7: EndAlgorithm
8: Algorithm Partition(A, low, high)
9:     pivot ← A[high]
10:    i ← low − 1
11:    For j ← low to high − 1 execute
12:        If A[j] ≤ pivot then
13:            i ← i + 1
14:            Swap(A[i], A[j])
15:        EndIf
16:    EndFor
17:    Swap(A[i + 1], A[high])
18:    Return i + 1
19: EndAlgorithm
```

Complexitate timp:

- Caz mediu: $O(n \log n)$
- Cazul cel mai rău: $O(n^2)$ (dacă șirul este deja sortat invers sau pivotul ales este mereu cel mai mic/mare element)

Metoda Merge Sort

Merge Sort este, de asemenea, un algoritm care utilizează "Divide et Impera". Spre deosebire de Quick Sort, Merge Sort împarte în mod recursiv șirul în jumătăți până când ajunge la subșiruri de un singur element, apoi le îmbină *(merge)* într-un mod ordonat pentru a obține șirul sortat final.

Exemplu: Vom sorta șirul $\{38, 27, 43, 3, 9, 82, 10\}$ folosind metoda Merge Sort:

- **Pasul 1:** Divizăm recursiv șirul până la subșiruri de un element:
  - Șirul inițial: $\{38, 27, 43, 3, 9, 82, 10\}$;
  - Împărțim în două: $\{38, 27, 43\}$ și $\{3, 9, 82, 10\}$;
  - Continuăm împărțirea:
    - $\{38, 27, 43\} \rightarrow \{38\}, \{27, 43\} \rightarrow \{27\}, \{43\}$;
    - $\{3, 9, 82, 10\} \rightarrow \{3, 9\}, \{82, 10\} \rightarrow \{3\}, \{9\}, \{82\}, \{10\}$.
- **Pasul 2:** Îmbinăm subșirurile într-un mod ordonat:
  - Îmbinăm $\{27\}$ și $\{43\} \rightarrow \{27, 43\}$;
  - Îmbinăm $\{38\}$ și $\{27, 43\} \rightarrow \{27, 38, 43\}$;
  - Îmbinăm $\{3\}$ și $\{9\} \rightarrow \{3, 9\}$;

  - Îmbinăm $\{82\}$ și $\{10\} \rightarrow \{10, 82\}$;
  - Îmbinăm $\{3, 9\}$ și $\{10, 82\} \rightarrow \{3, 9, 10, 82\}$.

- **Pasul 3:** Îmbinăm cele două subșiruri sortate:
  - Îmbinăm $\{27, 38, 43\}$ și $\{3, 9, 10, 82\} \rightarrow \{3, 9, 10, 27, 38, 43, 82\}$.

La final, obținem șirul sortat: $\{3, 9, 10, 27, 38, 43, 82\}$. Diagrama de mai jos ilustrează procesul:

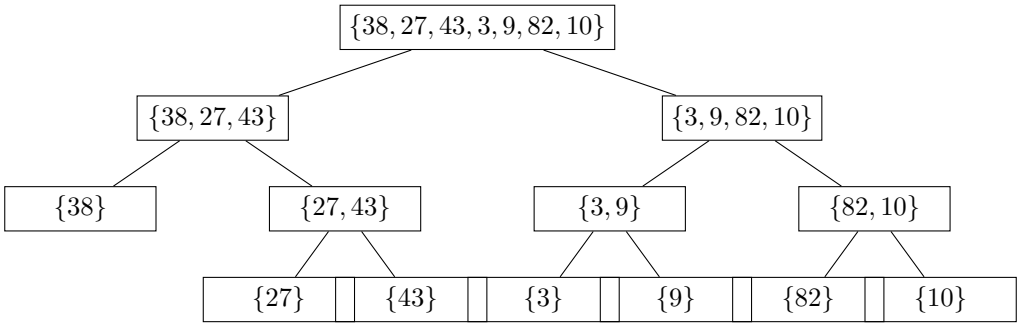


Figura 4.6 Divizarea șirului în Merge Sort

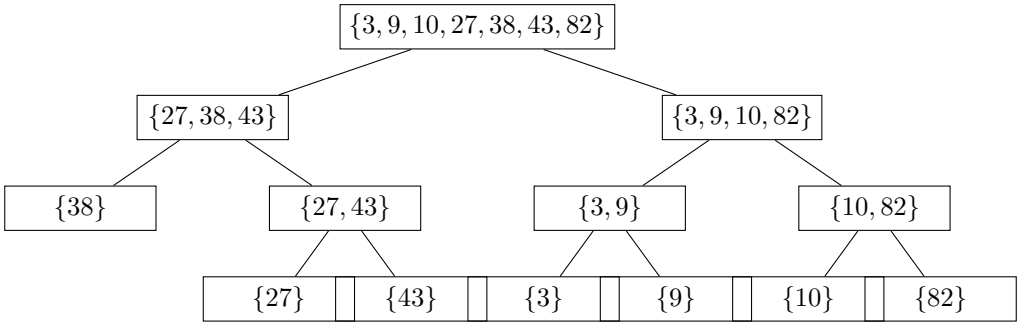


Figura 4.7 Îmbinarea subșirurilor în Merge Sort

O implementare a algoritmului în pseudocod arată astfel:

Merge Sort

```
1: Algorithm MergeSort(A, left, right)
2:     If left < right then
3:         mid ← ⌊(left + right)/2⌋
4:         MergeSort(A, left, mid)
5:         MergeSort(A, mid + 1, right)
6:         Merge(A, left, mid, right)          ▷ Definită la capitolul interclasare
7:     EndIf
8: EndAlgorithm
```

Complexitate timp: $O(n \log n)$ în toate cazurile, datorită divizării și îmbinării constante a subșirurilor.

Comparație între metodele de sortare

Până acum am prezentat multiple metode de sortare, iar acum a venit timpul să punem următoarea problemă: Pe care dintre ele este cel mai bine să o utilizăm? Răspunsul se bazează foarte mult pe analiza complexităților acestora, explicate în următorul capitol. Pe scurt, complexitățile sortărilor arată astfel:

Algoritm	Cazul mediu	Cazul cel mai rău
Bubble Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n \log n)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$

Tabela 4.1 Comparație algoritmi de sortare

Având în vedere comparația de mai sus, observăm că algoritmul Merge Sort este cel mai stabil și eficient din punct de vedere al complexității. Așadar, atunci când ne dorim să sortăm un șir de numere cât mai eficient, putem folosi cu certitudine Merge Sort.

4.2 Probleme

67. Care este rezultatul conversiei numărului binar 1010101100 în baza 8? ✓ ?

A. 684

B. 1254

C. 1002

D. Niciunul din răspunsurile A., B., C.

68. Care este rezultatul conversiei numărului zecimal $2^4 + 2^6 + 2^{10} - 3$ în baza 2? ✓ ?

A. 10001001101

B. 1101

C. 10001010000

D. Niciunul dintre răspunsurile A., B., C.

69. Fie expresia $E = 10_{(16)} - 10_{(15)} + 10_{(14)} - 10_{(13)} + \cdots + 10_{(2)}$, unde notația $x_{(b)}$ semnifică numărul x scris în baza b. Care valoare corespunde expresiei E? ✓ ?

A. 0 B. $1001_{(2)}$ C. $11_{(8)}$ D. $9_{(16)}$

70. Se consideră algoritmul `ceFace(arr, n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$) și `arr` este un șir de numere întregi cu n elemente ($-10^6 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^6$). ✓ ?

```
Algorithm ceFace(arr, n)
   c ← arr[1]
   For i ← 2, n execute
      dup ← false
      For j ← 1, i − 1 execute
         If arr[i] = arr[j] then
            dup ← true
         EndIf
      EndFor
      If NOT dup then
         If arr[i] < c then
            c ← arr[i]
         EndIf
      EndIf
   EndFor
   Return c
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `ceFace([7, 3, 3, 4, 8], 5)` algoritmul returnează 4.

B. Algoritmul `ceFace(arr, n)` determină elementul minim din vectorul `arr`.

C. Algoritmul `ceFace(arr, n)` determină elementul maxim din vectorul `arr`.

D. Dacă toate elementele din șirul `arr` sunt egale, algoritmul `ceFace(arr, n)` returnează oricare dintre aceste elemente.

71. Se consideră algoritmii `ceFace1()` și `ceFace2()` definiți alăturat, unde x este un număr întreg, cel mult 10^9. ✓ ?

```
Algorithm ceFace1(x)
   While x > 9 execute
      s ← 0
      While x > 0 execute
         s ← s + x MOD 10
         x ← [x DIV 10]
      EndWhile
      x ← s
   EndWhile
   Return x
EndAlgorithm
```

```
Algorithm ceFace2(x)
   If x = 0 OR x MOD 9 = 0 then
      Return 0
   Else
      Return x MOD 9
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul `ceFace1(x)` returnează suma cifrelor lui x;

B. Algoritmul `ceFace2(x)` returnează întotdeauna restul împărțirii lui x la 9;

C. Algoritmii `ceFace1(x)` și `ceFace2(x)` vor returna mereu aceeași valoare $\forall x$;

D. Algoritmul `ceFace1(x)` returnează cifra de control a numărului x.

72. Precizați care dintre următoarele afirmații referitoare la egalitatea unor valori în diferite baze de numerație sunt adevărate, unde $x_{(b)}$ semnifică numărul x scris în baza b. ✓ ?

A. $1010101_{(2)} = 85_{(10)}$

B. $1110010001011111010_{(2)} = E46FA_{(16)}$

C. $21030010121110_{(3)} = 7203543_{(9)}$

D. $1234_{(8)} = 1010011100_{(2)}$

73. Se consideră algoritmul `ceFace(n)`, unde `n` este un număr natural ($0 \leq$ `n` $\leq 10^4$). ✓ ?

```
Algorithm ceFace(n)
   If n = 1 then
      Return False
   EndIf
   If n = 2 then
      Return True
   EndIf
   If n MOD 2 = 0 AND n ≥ 4 then
      Return False
   EndIf
   For index ← 3, index * index ≤ n
execute
      If n MOD index = 0 then
         Return False
      EndIf
   EndFor
   Return True
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În cazul apelului `ceFace(21)`, algoritmul returnează True.

B. Algoritmul returnează False pentru toate valorile pare diferite de 2.

C. Algoritmul returnează True dacă și numai dacă numărul `n` este prim, False altfel.

D. Există exact 5 numere cu proprietatea că `n` $\leq$ 10, pentru care apelul `ceFace(n)` returnează True.

74. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^9$): ✓ ?

```
Algorithm ceFace(n)
   top ← 0
   While n > 0 execute
      If n MOD 6 = 0 then
         top ← top + 1
      EndIf
      n ← n DIV 6
   EndWhile
   Return top
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 1302$, algoritmul returnează valoarea 3.

B. Pentru $n = 216$, algoritmul returnează valoarea 2.

C. Dacă n este un număr par, algoritmul returnează o valoare impară.

D. Algoritmul returnează numărul de cifre de 0 din reprezentarea în baza 6 a numărului n.

75. Se consideră 3 numere în diferite baze de numerație: $x = 429$ în baza 10, $y = 1AD$ în baza 16 și $z = 110101100$ în baza 2. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. $x = y$ și $y \neq z$. B. $x = y = z$. C. $x \neq y$ și $y \neq z$. D. $x \neq y$ și $y = z$.

76. Se consideră subalgoritmul `ceFace()` definit alăturat. Şirul `a` de lungime `n` este citit de la tastatură, iar şirul `sp` este iniţializat cu 0. Funcţia `max(a, b)` returnează maximul dintre numerele `a` şi `b`. ✓ ?

```
Algorithm ceFace()
    c ← a[0]
    sp[0] ← a[0]
    If c < 0 then
        c ← 0
    EndIf
    For i ← 1, n − 1 execute
        c ← c + a[i]
        sp[i] ← max(sp[i − 1], c)
        If c < 0 then
            c ← 0
        EndIf
    EndFor
    c ← a[n − 1]
    b ← a[n − 1]
    If c < 0 then
        c ← 0
    EndIf
    d ← b + sp[n − 2]
    For i ← n − 2 to 1, −1 execute
        c ← c + a[i]
        If b < c then
            b ← c
        EndIf
        If c < 0 then
            c ← 0
        EndIf
        d ← max(d, b + sp[i − 1])
    EndFor
    Output d
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Subalgoritmul `ceFace` determină suma maximă posibilă care se poate obține din două secvențe oarecare din şirul `a`.

B. Subalgoritmul `ceFace` determină suma maximă posibilă care se poate obține din două secvențe care nu au niciun element comun din şirul `a`.

C. Subalgoritmul `ceFace` determină secvența de lungime maximă și sumă maximă din şirul `a`.

D. Subalgoritmul `ceFace` determină numărul de secvențe cu sumă și lungimea maximă din şirul `a`.

77. Se consideră algoritmul `ceFace(arr, n, i)`, unde n și i sunt numere naturale nenule ✓ ? $(1 \leq n \leq 10^4,\ 1 \leq i \leq n)$ și `arr` este un șir de numere întregi cu n elemente $(-10^6 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^6)$.

```
Algorithm ceFace(arr, n, i)
    If i = n then
        Return arr[i]
    EndIf
    m ← ceFace(arr, n, i + 1)
    If arr[i] > m then
        m ← arr[i]
    EndIf
    Return m
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate referitoare la algoritmul `ceFace(arr, n, i)`?

A. Pentru apelul `ceFace([10, -3, 7, 5, -11, 12, 6, 0], 8, 1)` algoritmul returnează -11.

B. Pentru apelul `ceFace([10, -3, 7, 5, -11, 12, 6, 0], 8, 1)` algoritmul returnează 12.

C. Pentru apelul `ceFace([5, -4, 3, -2, 1], 5, 2)` algoritmul returnează 5.

D. Algoritmul `ceFace(arr, n, i)` determină maximul din întregul vector `arr`.

78. Se consideră algoritmul `special`(v, n, k), unde n și v sunt două numere naturale ✓ ? $(1 \leq n \leq 10^4,\ 1 \leq k \leq 9)$, iar v este un șir de numere naturale cu n elemente $(v[1], v[2], \ldots, v[n])$.

```
Algorithm SPECIAL(v, n, k)
   If n = 0 then
      Return 0
   EndIf
   Return ((v[n] DIV 10 − v[n] MOD 10 ∗ 2) MOD k = 0) + special(v, n − 1, k)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `special`([11, 33, 36, 44, 66, 67], 6, 3) algoritmul returnează 3.

B. Pentru apelul `special`([28, 70, 30, 35], 4, 7) algoritmul returnează 3.

C. Algoritmul `special`(v, n, k) calculează și returnează câte numere sunt divizibile cu k în vectorul v.

D. Pentru apelurile `special`([21, 15, 6, 8, 49], 5, 7) și `special`([77, 147, 5, 16, 25], 5, 7) algoritmul returnează în ambele cazuri o valoare pară.

79. Se consideră algoritmul `taste(arr, n)`, unde n este un număr natural nenul $(1 \leq$ ✓ ? $n \leq 10^4)$, iar `arr` este un șir de numere naturale cu n elemente $(1 \leq$ `arr`$[1],$ `arr`$[2], \ldots,$ `arr`$[n] \leq 10^6)$.

```
Algorithm TASTE(arr, n)
   If n = 0 then
      Return 0
   EndIf
   p ← true
   If arr[n] ≤ 1 then
      p ← false
   EndIf
   For j ← 2 to j ∗ j ≤ arr[n] execute
      If arr[n] MOD j = 0 then
         p ← false
      EndIf
   EndFor
   If p then
      Return taste(arr, n - 1)
   EndIf
   Return arr[n] + taste(arr, n - 1)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul `taste(arr, n)` returnează suma numerelor prime din vectorul `arr`.

B. Pentru apelul `taste([3, 8, 9, 13, 15, 21, 23, 24], 8)` algoritmul returnează 39.

C. Pentru apelul `taste([1, 5, 6, 9, 11, 16, 17, 19, 20], 9)` algoritmul returnează 52.

D. Algoritmul `taste(arr, n)` returnează 0 dacă niciun element din vectorul `arr` nu este prim.

80. Se consideră algoritmul `extra(n)`, unde n este un număr natural nenul $(1 \leq n \leq 10^9)$. ✓ ?

```
Algorithm EXTRA(n)
   If n > 0 then
      Write n MOD 2
      extra(n DIV 2)
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 73$, algoritmul afișează `1001001`.

B. Pentru $n = 21$, algoritmul afișează `10101`.

C. Algoritmul afișează reprezentarea numărului n în baza 2.

D. Algoritmul are complexitate logaritmică.

81. Se consideră algoritmul `ceFace(arr, n)`, unde n este un număr natural nenul ($2 \leq n \leq 10^4$), iar `arr` este un șir de numere întregi cu n elemente ($-10^4 \leq$ `arr`$[1]$, `arr`$[2], \ldots,$ `arr`$[n] \leq 10^4$). ✓ ?

```
Algorithm CEFACE(arr, n)
   a ← 10001
   b ← −10001
   For i ← 2 to n execute
      If arr[i] > arr[1] AND arr[i] < a then
         a ← arr[i]
      EndIf
      If arr[i] < arr[1] AND arr[i] > b then
         b ← arr[i]
      EndIf
   EndFor
   If a = 10001 OR b = −10001 then
      Return arr[1]
   Else
      Return a + b
   EndIf
EndAlgorithm
```

Pentru care dintre următoarele apeluri ale algoritmului `ceFace(arr, n)` algoritmul returnează valoarea 31?

A. `ceFace([31, 31, 31, 31, 31], 5)`

B. `ceFace([31, -21, -11, -1, 32, -5, 49], 7)`

C. `ceFace([18, 20, 5, 6], 4)`

D. `ceFace([15, 17, 5, 49, 37, 14], 6)`

82. Se consideră algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale pozitive ($1 \leq a \leq b \leq 10^9$). ✓ ?

```
Algorithm ceFace(a, b)
    k ← 0
    m ← 10^9
    M ← 0
    For i ← a, b execute
        x ← 1
        nr ← 0
        d ← 2
        cn ← i
        While cn > 1 AND d * d ≤ cn execute
            nr ← 0
            If cn MOD d = 0 then
                While cn MOD d = 0 execute
                    cn ← cn DIV d
                    nr ← nr + 1
                EndWhile
                x ← x * (nr + 1)
            EndIf
            d ← d + 1
        EndWhile
        If cn > 1 then
            x ← x * 2
        EndIf
        If x > M then
            M ← x
            m ← i
            k ← 1
        Else
            If x = M then
                k ← k + 1
            EndIf
        EndIf
    EndFor
    Write m, M, k
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `ceFace(200, 200)`, algoritmul afişează valorile 200 10 1.

B. Pentru apelul `ceFace(2, 10)`, algoritmul afişează valorile 6 4 3.

C. Algoritmul `ceFace(a, b)` determină și afişează cel mai mic număr din intervalul $[a, b]$ cu număr maxim de divizori, numărul de divizori al acestuia și numărul de numere cu această proprietate.

D. Algoritmul `ceFace(a, b)` determină și afişează cel mai mic număr din intervalul $[a, b]$ cu număr maxim de factori primi distincți, numărul acestora și numărul de numere care au această proprietate.

83. Care ar putea fi elementele unui vector astfel încât, aplicând metoda căutării binare pentru valoarea `64`, aceasta să fie comparată succesiv cu valorile `36`, `45`, `64`? ✓ ?

A. $[4, 7, 9, 15, 36, 40, 42, 45, 64, 67]$

B. $[2, 4, 7, 12, 24, 36, 50]$

C. $[12, 24, 36, 42, 54, 66]$

D. $[4, 8, 9, 36, 16, 45, 64]$

84. Fie expresia $E = AB_{(16)} + 120_{(3)} - 120_{(4)}$, unde notația $x_{(b)}$ semnifică numărul x scris în baza b. Care valoare corespunde expresiei E? ✓ ?

A. $162_{(10)}$; B. $278_{(8)}$; C. $1000101_{(2)}$; D. $242_{(8)}$.

85. Se consideră algoritmul `dificil(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^6$). ✓ ?

```
Algorithm DIFICIL(n)
    k ← 0
    nr ← 0
    For index ← 1 to n execute
        nr ← index
        While nr > 0 execute
            k ← k + (nr MOD 2)
            nr ← nr DIV 2
        EndWhile
    EndFor
    Return k
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 509$, algoritmul returnează 2287.

B. Pentru $n = 1025$, algoritmul returnează 5123.

C. Pentru $n = 120$, algoritmul returnează 406.

D. Algoritmul `dificil(n)` calculează și returnează numărul total de cifre de 1 din reprezentarea binară a tuturor numerelor de la 1 până la n.

86. Se consideră algoritmul `indigo(n)`, unde n este un număr natural ($1 \leq n \leq 10^6$). ✓ ?

```
Algorithm INDIGO(n)
    cnt ← 0
    For i ← 2, i * i ≤ n execute
        If n MOD i = 0 then
            cnt ← cnt + 1
            While n MOD i = 0 execute
                n ← n DIV i
            EndWhile
        EndIf
    EndFor
    If n > 1 then
        cnt ← cnt + 1
    EndIf
    Return cnt
EndAlgorithm
```

Care dintre următoarele apeluri ale algoritmului `indigo(n)` returnează valoarea 3?

A. `indigo(120)`

B. `indigo(9016)`

C. `indigo(630)`

D. `indigo(8064)`

87. Se consideră algoritmul `lambda(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^9$). ✓ ?

```
Algorithm LAMBDA(n)
    If n < 10 then
        Return n
    EndIf
    b ← LAMBDA(n DIV 10)
    a ← n MOD 10
    If a MOD 4 > b MOD 4
then
        Return a
    EndIf
    Return b
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate referitoare la algoritmul `lambda(n)`?

A. Pentru apelul `lambda(12569)` algoritmul returnează 2.

B. Pentru apelul `lambda(783031)` algoritmul returnează 8.

C. Pentru apelurile `lambda(2024)` și `lambda(2025)` algoritmul returnează aceeași valoare.

D. Algoritmul `lambda(n)` returnează prima cifră divizibilă cu 4 din reprezentarea numărului n.

88. Se consideră algoritmul `book(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^9$). ✓ ?

```
Algorithm BOOK(n)
   If n < 10 then
      x ← n MOD 3
      y ← n MOD 2
      If x = 0 AND y ≠ 0 then
         Return n
      Else
         Return -1
      EndIf
   EndIf
   u ← n MOD 10
   v ← n DIV 10
   p ← book(v)
   t ← u MOD 3
   q ← u MOD 2
   z ← p
   If t = 0 AND q ≠ 0 AND u > z then
      w ← u
   Else
      w ← z
   EndIf
   r ← w * 2 − w
   Return r
EndAlgorithm
```

Care dintre următoarele afirmații sunt false referitoare la algoritmul `book(n)`?

A. Pentru apelul `book(3063528)` algoritmul returnează 3.

B. Pentru apelul `book(24680)` algoritmul returnează -1.

C. Pentru apelul `book(5792041)` algoritmul returnează 9.

D. Algoritmul `book(n)` returnează -1 dacă numărul n conține doar cifre din mulțimea $\{3, 4, 5, 6, 7, 8\}$.

89. Se consideră algoritmii `switch(a, b)` și `case(a, b)`, unde a și b sunt numere naturale $(1 \leq a, b \leq 10^6)$. ✓ ?

```
Algorithm SWITCH(a, b)
   If a = b then
      Return a
   EndIf
   If a < b then
      Return switch(a, b - a)
   EndIf
   Return switch(a - b, b)
EndAlgorithm
```

```
Algorithm CASE(a, b)
   Return a * b DIV switch(a, b)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă $a = 2024$ și $b = 32$, atunci apelul `switch(a, b)` returnează 8.

B. Dacă $a = 48$ și $b = 22$, atunci apelul `case(a, b)` returnează 528.

C. Algoritmul `switch(a, b)` returnează cel mai mic multiplu comun al lui a și b.

D. Algoritmul `case(a, b)` returnează produsul celor două numere, dacă acestea sunt prime între ele.

90. Se consideră algoritmul `red(arr, n)`, unde n este un număr natural nenul $(1 \leq n \leq 10^4)$ și `arr` este un șir de numere întregi cu n elemente $(0 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^6)$. ✓ ?

```
Algorithm RED(arr, n)
   For i ← n − 1, 1, −1 execute
      If arr[i] < arr[i + 1] then
         arr[i] ← arr[i] − arr[i + 1]
         arr[i + 1] ← arr[i + 1] + arr[i]
         arr[i] ← arr[i + 1] − arr[i]
      EndIf
   EndFor
   Return arr[1]
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `red([15, 23, 7, 12, 81, 49, 35, 31], 8)` algoritmul returnează 7.

B. Pentru apelul `red([23, 10, 44, 52, 19, 81, 3, 72, 22], 9)` algoritmul returnează 81.

C. Algoritmul `red(arr, n)` sortează descrescător vectorul `arr`.

D. Algoritmul `red(arr, n)` determină elementul minim din vectorul `arr`.

91. Se consideră algoritmul `great(arr, n, i)`, unde n și i sunt numere naturale nenule ($1 \leq n, i \leq 10^4$), iar `arr` este un șir de numere naturale cu n elemente ($1 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^6$). ✓ ?

```
Algorithm GREAT(arr, n, i)
   If i = n then
      Return arr[i]
   EndIf
   g ← GREAT(arr, n, i + 1)
   While g ≠ 0 execute
      If arr[i] > g then
         arr[i] ← arr[i] MOD g
      Else
         temp ← arr[i]
         arr[i] ← g
         g ← temp MOD g
      EndIf
   EndWhile
   Return arr[i]
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate referitoare la algoritmul `great(arr, n, i)`.

A. Pentru apelul `great([30, 60, 12, 48, 72], 5, 1)` algoritmul returnează 3.

B. Pentru apelul `great([21, 14, 42], 3, 2)` algoritmul returnează 7.

C. Pentru apelul `great([45, 35, 20, 80, 50], 5, 3)` algoritmul returnează 10.

D. Algoritmul are complexitate liniară $O(n)$.

92. Se consideră algoritmul `Algo(v, n)`, unde `v` este un vector cu `n` numere naturale ($1 \leq n \leq 10^5$): ✓ ?

```
Algorithm ALGO(v, n)
   s ← 0
   For i ← 1, n execute
      v[i] ← n + 1 − i
   EndFor
   For i ← 1, n execute
      If i MOD 2 ≠ 0 then
         s ← s + v[i]
      EndIf
   EndFor
   Write s
EndAlgorithm
```

Ce valoare afișează algoritmul în urma apelului `Algo(v, 10)`?

A. 25

B. 35

C. 50

D. 30

93. Se consideră algoritmii `ceFace1(v, n)` și `ceFace2(v, n, i, fl, sl)`, unde n și i sunt numere naturale ($1 \leq n \leq 10^4$, $0 \leq i \leq n$), iar v este un șir de numere întregi cu n elemente ($0 \leq v[1], v[2], \ldots, v[n] \leq 10^6$). fl si sl sunt 2 numere naturale ($0 \leq fl, sl \leq 10^6$). ✓ ?

```
Algorithm ceFace2(v, n, i, fl, sl)
   If i ≤ n then
      If v[i] > fl then
         ceFace2(v, n, i + 1, v[i], fl)
      Else
         If v[i] > sl AND v[i] ≠ fl then
            ceFace2(v, n, i + 1, fl, v[i])
         Else
            ceFace2(v, n, i + 1, fl, sl)
         EndIf
      EndIf
   Else
      Write '(', fl, ',', sl, ')', ' '
   EndIf
EndAlgorithm
```

```
Algorithm ceFace1(v, n)
   For k ← 0, n execute
      ceFace2(v, n, k, 0, 0)
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate referitoare la algoritmii `ceFace1(v, n)` și `ceFace2(v, n, i, fl, sl)`?

A. Pentru apelul `ceFace2([9, 8, 6, 3, 7, 2, 4], 6, 1, 0, 0)`, algoritmul afișează $(8, 7)$.

B. Pentru apelul `ceFace1([7, 9, 1, 4, 5, 3], 5)`, algoritmul afișează $(9, 7)$ $(9, 5)$ $(5, 4)$ $(5, 4)$ $(5, 3)$.

C. Algoritmul `ceFace1(v, n)` afișează $n + 1$ perechi de numere din vectorul v, de forma (a, b), unde $a > b$ și a este cel mai mare element din vectorul v.

D. Pentru apelul `ceFace2([5, 5, 4, 2, 1, 3, 1], 6, 3, 0, 0)`, algoritmul afișează $(3, 2)$.

94. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^9$). ✓ ?

```
Algorithm ceFace(n)
   While n ≠ 0 execute
      If n MOD 4 > 1 then
         Return False
      EndIf
      n ← n DIV 4
   EndWhile
   Return True
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Algoritmul returnează True dacă și numai dacă n conține doar cifrele 0 și 1 în reprezentarea sa în baza 4.

B. Algoritmul verifică dacă numărul n poate fi scris ca suma de puteri distincte ale lui 4.

C. Pentru $n = 806$, algoritmul returnează False.

D. Algoritmul returnează False pentru orice număr care nu este multiplu de 4.

95. Se consideră algoritmul `Algo(v)`, unde `v` este un vector cu $n = 8$ elemente: ✓ ?

```
Algorithm ALGO(v)
   For i ← 8, 1, −1 execute
      If i MOD 2 ≠ 0 then
         v[i] ← i + 1
      Else
         v[i] ← i − i/2
      EndIf
   EndFor
EndAlgorithm
```

Cum va arăta vectorul v în urma execuției instrucțiunilor algoritmului `Algo(v)`?

A. 3 1 5 3 7 5 9 7

B. 2 1 4 3 6 5 8 7

C. 2 1 4 2 6 3 8 4

D. 3 1 5 3 5 7 7 9

96. Se consideră algoritmul `Algo(v, n, e)`, unde v este un vector cu n elemente întregi distincte $(v[1], v[2], \ldots, v[n])$, $1 \leq n \leq 10^4$ și $v[i] \neq v[j]$, pentru $1 \leq i \leq j \leq n$), iar e este un număr întreg. Algoritmul caută elementul e în vectorul v și, dacă îl găsește, mută toate elementele de după elementul e la începutul vectorului și returnează `True`, nemodificând ordinea celorlalte elemente. Dacă elementul e nu se găsește în v, algoritmul returnează `False` și nu modifică nimic. De exemplu, pentru vectorul v cu elementele $[3, 5, 7, 9, 11]$ și $e = 5$, algoritmul va returna `True` și vectorul v va deveni $[7, 9, 11, 3, 5]$. Care dintre următoarele implementări de mai jos este o variantă corectă pentru algoritmul descris? ✓ ?

A.

```
Algorithm ALGO(v, n, e)
   i ← 0
   While i < n execute
      If v[i] = e then
         Break
      EndIf
      i ← i + 1
   EndWhile
   If i = n then
      Return False
   EndIf
   j ← i
   While j < n execute
      v[j − i] ← v[j]
      j ← j + 1
   EndWhile
   k ← 0
   While k < i execute
      v[n − i + k] ← v[n − k − 1]
      k ← k + 1
   EndWhile
   Return True
EndAlgorithm
```

B.

```
Algorithm ALGO(v, n, e)
   i ← 0
   found ← False
   While i < n execute
      If v[i] = e then
         found ← True
         Break
      EndIf
      i ← i + 1
   EndWhile
   If NOT found then
      Return False
   EndIf
   j ← i
   While j < n execute
      v[j − i] ← v[j]
      j ← j + 1
   EndWhile
   k ← 0
   While k < i execute
      v[k] ← v[n − i + k]
      k ← k + 1
   EndWhile
   Return True
EndAlgorithm
```

C.

```
Algorithm ALGO(v, n, e)
    i ← 1
    found ← False
    While i ≤ n execute
        If v[i] = e then
            found ← True
            Break
        EndIf
        i ← i + 1
    EndWhile
    If NOT found then
        Return False
    EndIf
    elements ← n − i
    temp ← []                    ▷ vector vid
    For j ← 1, elements execute
        temp[j] ← v[i + j]
    EndFor
    For j ← i, 1, −1 execute
        v[j + elements] ← v[j]
    EndFor
    For j ← 1, elements execute
        v[j] ← temp[j]
    EndFor
    Return True
EndAlgorithm
```

D.

```
Algorithm ALGO(v, n, e)
    i ← 0
    found ← False
    While i < n execute
        If v[i] = e then
            found ← True
            Break
        EndIf
        i ← i + 1
    EndWhile
    If NOT found then
        Return False
    EndIf
    temp ← [0] ∗ i
    k ← 0
    tempIndex ← 0
    While k ≤ i execute
        temp[tempIndex] ← v[k]
        tempIndex ← tempIndex + 1
        k ← k + 1
    EndWhile
    j ← i
    While j < n execute
        v[j − i] ← v[j]
        j ← j + 1
    EndWhile
    k ← 0
    While k < i execute
        v[n − i + k] ← temp[k]
        k ← k + 1
    EndWhile
    Return True
EndAlgorithm
```

97. Se consideră algoritmul `algo(v, n, k)`, unde n este un număr natural ($1 \leq n \leq 10^4$), ✓ ? v este un vector cu n elemente numere întregi ($v[1], v[2], \ldots, v[n]$), iar k este un număr natural:

```
Algorithm ALGO(v, n, k)
    If n MOD k ≠ 0 then
        Return False
    EndIf
    s ← n DIV k
    For i ← 1, k execute
        start ← (i − 1) ∗ s
        end ← i ∗ s
        For j ← start, end − 2 execute
            If v[j] ≥ v[j + 1] then
                Return False
            EndIf
        EndFor
    EndFor
```

```
    Return True
EndAlgorithm
```

Precizați care dintre afirmațiile de mai jos sunt adevărate:

A. Algoritmul verifică dacă un vector poate fi împărțit în k subsecvențe egale, fiecare fiind strict crescător.

B. Algoritmul verifică dacă un vector poate fi împărțit în k subsecvențe, fiecare fiind ordonat crescător.

C. Apelul `algo([1, 2, 1, 2, 1, 2, 1, 2], 8, 4)` returnează valoarea `True`.

D. Apelul `algo([1, 2, 3, 4, 3, 4], 6, 2)` returnează valoarea `True`.

98. Se consideră algoritmul `algo(n, k)`, unde n și k sunt numere naturale ($1 \leq n, k \leq 10^6$): ✓ ?

```
Algorithm ALGO(n, k)
    nr ← 0
    p ← 1
    While (n ≠ 0)
        AND (k ≠ 0) execute
        a ← n MOD 10
        b ← (n DIV 10) MOD 10
        If (a + b) MOD 3 = 0 then
            nr ← nr + (a * b) * p
            p ← p * 10
        Else
            k ← k − 1
        EndIf
        n ← n DIV 10
    EndWhile
    Return nr
EndAlgorithm
```

Care dintre următoarele perechi de apeluri returnează valori identice?

A. `algo(676458, 3)` și `algo(655413, 3)`

B. `ceFace(2314587, 4)` și `ceFace(1314579, 5)`

C. `algo(321458, 7)` și `algo(120459, 6)`

D. `ceFace(227459, 6)` și `algo(227454, 4)`

99. Se consideră algoritmul `transform(secv, n, m)`, unde n este un număr natural ($1 \leq n \leq 10^4$), $secv$ este un vector cu n elemente numere naturale ($secv[1], secv[2], \ldots, secv[n]$), iar m este un număr întreg. Se consideră $|m|$ ca fiind valoarea absolută a numărului. ✓ ?

```
Algorithm TRANSFORM(secv, n, m)
    For i ← 1, n execute
        rez[i] ← 1
    EndFor
    For i ← 1, n execute
        If m = 0 then
            rez[i] ← 1
        Else If m > 0 then
            prod ← 1
            For j ← 1, m execute
                prod ← prod * secv[(i + j − 1) MOD n + 1]
            EndFor
            rez[i] ← prod
        Else
            prod ← 1
```

```
            For j ← 1, |m| execute
                prod ← prod * secv[(i − j + n − 1) MOD n + 1]
            EndFor
            rez[i] ← prod
        EndIf
    EndFor
    Return rez
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru apelul `transform([1, 3, 8, 5, 9], 5, 2)` algoritmul va returna $[25, 40, 45, 30, 9]$.

B. Pentru apelul `transform([1, 3, 2, 7], 4, -3)` algoritmul va returna $[42, 14, 21, 6]$.

C. Pentru apelul `transform([2, 3, 4, 5, 6], 5, 3)` algoritmul va returna $[60, 120, 60, 36, 24]$.

D. Pentru apelul `transform([1, 3, 7, 5, 9], 5, -2)` algoritmul va returna $[45, 9, 3, 21, 35]$.

100. Se consideră algoritmii `g(v, n)`, și `f(v, n, t)`,unde n este un număr natural ($1 \leq n \leq 10^4$), v este un vector cu n elemente numere naturale ($v[1], v[2], \ldots, v[n]$), iar t este un număr natural ($1 \leq n \leq 10^4$). Valoarea INF reprezintă valoarea cea mai mare valoarea posibilă.

```
Algorithm F(v, n, t)
    G(v, n)
    c ← INF
    aux ← [ ]
    For i ← 1, 3 execute
        aux[i] ← 0
    EndFor
    For i ← 1, n − 2 execute
        l ← i + 1
        r ← n − 1
        While l < r execute
            s ← v[i] + v[l] + v[r]
            If |s − t| < |c − t| then
                c ← s
                aux ← [v[i], v[l], v[r]]
            EndIf
            If s < t then
                l ← l + 1
            Else If s > t then
                r ← r − 1
            Else
                Return aux
            EndIf
        EndWhile
    EndFor
    Return aux
EndAlgorithm
```

```
Algorithm G(v, n)
    For i ← 1, n − 1 execute
        For j ← 1, n − i execute
            If v[j] > v[j + 1] then
                temp ← v[j]
                v[j] ← v[j + 1]
                v[j + 1] ← temp
            EndIf
        EndFor
    EndFor
EndAlgorithm
```

Precizați care dintre afirmațiile de mai jos sunt adevărate:

A. Algoritmul indică tripletul de elemente din vector, a cărui sumă reprezintă cea mai apropiată valoare de o valoare dată.

B. Algoritmul indică tripletul de elemente din vector, a cărui sumă reprezintă elementul strict mai mic al unei valori date.

C. Pentru apelul `f([-1, 2, 1, -3, 5, -4], 6, 1)` algoritmul returnează $[-3, -1, 5]$.

D. Pentru apelul `f([-1, 7, 1, 5, -8], 5, 6)` algoritmul returnează $[-1, 1, 5]$.

101. Se consideră algoritmul `one(nr)`, unde nr este un număr natural nenul ($1 \leq nr \leq 10^6$) și algoritmul `two(v, n)`, unde n este un număr natural nenul ($1 \leq n \leq 10000$) și v un vector cu n numere naturale pozitive ($1 \leq v[1], v[2], \ldots, v[n] \leq 10^6$). ✓ ?

```
1: Algorithm TWO(v, n)
2:     cnt ← 0
3:     p ← 1
4:     For i ← 1, n − 1 execute
5:         For j ← n, i + 1, −1 execute
6:             p ← p + 1
7:             If (one(v[i]) MOD 2 = one(v[j]) MOD 2) AND (one(v[i]) MOD 2) then
8:                 cnt ← cnt + 1
9:             EndIf
10:        EndFor
11:    EndFor
12:    Return p − cnt − 1
13: EndAlgorithm
```

```
Algorithm ONE(nr)
    c ← 0
    For d ← 1, d * d ≤ nr execute
        If nr MOD d = 0 then
            c ← c + 1
            If d * d < nr then
                c ← c + 1
            EndIf
        EndIf
    EndFor
    Return c
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Pentru apelul `one(2027)`, se returnează 2.

B. Pentru apelul `two([144, 12, 24, 1024, 4, 36], 6)`, se returnează 6.

C. Algoritmul `two(v, n)` returnează numărul de perechi (a, b) de elemente din vector, cu număr par de divizori.

D. Dacă linia 12 ar fi schimbată cu "`Return cnt`", algoritmul `two(v, n)` ar returna numărul de perechi de elemente din vector, cu proprietatea că cele 2 numere sunt pătrate perfecte.

102. Se consideră algoritmul `poate(n, b)`, unde n și b sunt numere naturale nenule ($1 \leq n \leq 10^9$, $2 \leq b \leq 10$). ✓ ?

```
Algorithm POATE(n, b)
    If n = 0 then
        Return 1
    EndIf
```

```
    Return n MOD b MOD 2 * poate(n DIV b, b)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 42$ și $b = 4$, algoritmul returnează 1.

B. Pentru $n = 502$ și $b = 7$, algoritmul returnează 1.

C. Algoritmul `poate(n, b)` returnează produsul cifrelor pare ale lui n din reprezentarea în baza b.

D. Algoritmul `poate(n, b)` returnează 1 dacă și numai dacă toate cifrele lui n din reprezentarea în baza b sunt impare.

103. Fie nr un număr întreg ($0 < nr \leq 10^9$) și b o bază întreagă ($1 < b < 11$). Precizați care dintre următorii algoritmi transformă corect numărul nr din baza b în baza 10 și returnează acest număr. ✓ ?

A.

```
Algorithm ALG1(nr, b)
    rez ← 0
    k ← 1
    ax ← 1
    While nr > 0 execute
        ax ← (nr MOD 10) * k
        rez ← rez + ax
        nr ← nr DIV 10
        k ← k * b
    EndWhile
    Return rez
EndAlgorithm
```

B.

```
Algorithm ALG2(nr, b)
    rez ← 0
    k ← 1
    While nr > 0 execute
        k ← k * b
        rez ← rez + (nr MOD 10) * k
        nr ← nr DIV 10
    EndWhile
    Return rez
EndAlgorithm
```

C.

```
Algorithm ALG3(nr, b)
    If nr = 0 then
        Return 0
    EndIf
    Return nr MOD 10+alg3(nr DIV 10, b*
b)
EndAlgorithm
```

D.

```
Algorithm ALG4(nr, b)
    If nr = 0 then
        Return 0
    EndIf
    ax ← nr
    ax ← ax MOD 10
    nr1 ← nr DIV 10
    Return ax + alg4(nr1, b) * b
EndAlgorithm
```

104. Se consideră algoritmul `clock(n)`, unde n este un număr natural ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm CLOCK(n)
    nd ← 0
    For i ← 1, n execute
        If n MOD i = 0 then
            If i MOD 2 = 0 then
                nd ← nd + 1
            Else
                nd ← nd − 1
            EndIf
```

```
      EndIf
   EndFor
   Return nd
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă $n = 12$, algoritmul returnează 2.

B. Algoritmul calculează și returnează diferența dintre numărul de divizori pari și numărul de divizori impari ai lui n.

C. Dacă n este un număr prim, algoritmul `clock(n)` returnează întotdeauna -2.

D. Algoritmul `clock(n)` returnează 0 pentru toate numerele din mulțimea $\{2, 6, 18, 30, 72\}$.

105. Se consideră algoritmul `algo(n, x, p)`, unde n este un număr natural ($1 \leq n \leq 10^5$), x este un vector cu n elemente numere întregi $(x[1], x[2], \ldots, x[n])$, iar p este un număr natural:

```
Algorithm ALGO(n, x, p)
   If n MOD p ≠ 0 then
      Return False
   EndIf
   i ← 1
   While i ≤ n − p execute
      If x[i] ≠ x[i + p] then
         Return False
      EndIf
      i ← i + 1
   EndWhile
   Return True
EndAlgorithm
```

Pentru care dintre următoarele condiții algoritmul returnează `True`?

A. Dacă vectorul `x` este periodic, iar lungimea `n` este un multiplu al perioadei `p`.

B. Dacă în vectorul `x` toate elementele au aceeași valoare, indiferent de valoarea lui `p`.

C. Dacă `x[i]` este egal cu `x[i+p]` pentru `i` $= \overline{1, n}$, iar lungimea `n` nu este multiplu al lui `p`.

D. Dacă vectorul `x` este periodic cu perioada `p`, iar $n \bmod p = 0$.

106. Se consideră algoritmul `algo(n, d)`, unde n este un număr natural pozitiv ($1 \leq n \leq 10^4$), iar d este o cifră ($0 \leq d \leq 9$).

```
Algorithm ALGO(n, d)
   count ← 0
   total ← 0
   While n > 0 execute
      r ← n MOD 10
      If r = d then
         count ← count + 1
      EndIf
      total ← total + 1
      n ← n DIV 10
   EndWhile
   Return count > total DIV 2
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $n = 1223333$ și $d = 3$, algoritmul returnează `True`.

B. Pentru $n = 444555$ și $d = 4$, algoritmul returnează `True`.

C. Algoritmul verifică dacă cifra d apare de mai multe ori decât jumătate din numărul total de cifre ale lui n.

D. Pentru $n = 77431$ și $d = 7$, algoritmul returnează `True`.

107. Se consideră algoritmul `algo(n, b)`, unde n este un număr natural pozitiv ($1 \leq n \leq 10^6$) și b este un număr natural. ✓ ?

```
Algorithm ALGO(n, b)
    count ← 0
    While n > 0 execute
        r ← n MOD b
        If r MOD (b − 1) = 0 then
            count ← count + 1
        EndIf
        n ← n DIV b
    EndWhile
    Return count
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $n = 31$ și $b = 4$, algoritmul returnează 2.

B. Pentru $n = 255$ și $b = 16$, algoritmul returnează 2.

C. Algoritmul determină câte cifre din reprezentarea numărului n în baza b sunt divizibile cu $b - 1$.

D. Algoritmul determină suma cifrelor din reprezentarea numărului n în baza b care sunt divizibile cu $b - 1$.

108. Se consideră algoritmul `algo(n, x)`, unde n este un număr natural ($1 \leq n \leq 10^4$), iar x este un vector cu n elemente numere întregi ($x[1], x[2], \ldots, x[n]$). ✓ ?

```
Algorithm ALGO(n, x)
    For i ← 1, n − 1 execute
        nr ← 0
        a ← x[i]
        b ← x[i + 1]
        While a > 0  OR b > 0 execute
            If a > 0 then
                cif_a ← a MOD 10
            Else
                cif_a ← 0
            EndIf
            If b > 0 then
                cif_b ← b MOD 10
            Else
                cif_b ← 0
            EndIf
            If cif_a > cif_b then
                nr ← nr + (cif_a − cif_b)
            Else If cif_a < cif_b then
                nr ← nr − (cif_b − cif_a)
            Else
                nr ← nr + 1
            EndIf
            a ← a DIV 10
            b ← b DIV 10
        EndWhile
        x[i] ← (x[i] + nr) MOD (i + 1)
    EndFor
    s ← 0
    For i ← 1, n execute
        s ← s + x[i]
    EndFor
```

```
    x[n] ← s MOD 10
    Return x
EndAlgorithm
```

Care va fi noul conținut al vectorului x după executarea algoritmului dat dacă $n = 6$ și $x = [134, 214, 789, 578, 636, 543]$?

A. $[0, 2, 1, 2, 3, 2]$
B. $[0, 2, 1, 3, 3, 3]$
C. $[0, 2, 1, 3, 3, 2]$
D. $[1, 2, 1, 3, 3, 2]$

109. Se consideră algoritmul `algo(n)`, unde n este un număr natural ($1 \leq n \leq 10^5$). ✓ ?

```
Algorithm ALGO(n)
    a ← 0
    b ← False
    While n > 0 execute
        c ← n MOD 10
        If c MOD 2 = 0 then
            a ← a + c
            b ← True
        EndIf
        n ← n DIV 10
    EndWhile
    If NOT b then
        Return −1
    Else
        Return a
    EndIf
EndAlgorithm
```

Ce va returna apelul `algo(6356)`?

A. 13 B. 50 C. 0 D. 12

110. Se consideră algoritmul `algo(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10^5$). ✓ ?

```
Algorithm ALGO(a, b)
    c ← 0
    d ← 1
    p ← 1
    While a + b > 0 execute
        If (a MOD 10 + b MOD 10) MOD 2 = 0 then
            c ← (a MOD 10 + b MOD 10) * p
        Else
            If a MOD 10 > b MOD 10 then
                c ← (a MOD 10 − b MOD 10) * p
            Else
                c ← (b MOD 10 − a MOD 10) * p
            EndIf
        EndIf
        If c < 0 then
            c ← c * (−1)
        EndIf
        d ← d + c
        p ← p + 2
        a ← a DIV 10
        b ← b DIV 10
```

```
   EndWhile
   Return d
EndAlgorithm
```

Care va fi rezultatul apelului `algo(387, 2349)`?

A. 23 B. 145 C. 97 D. 278

111. Se consideră algoritmul `f(v, n)`, unde n este un număr natural ($1 \leq n \leq 10^4$), iar v este un vector cu n elemente numere naturale ($v[1], v[2], \ldots, v[n]$). ✓ ?

```
Algorithm F(v, n)
   If n = 0 then
      Return 0
   EndIf
   If n = 1 then
      Return v[1]
   EndIf
   p2 ← 0
   p1 ← v[1]
   For i ← 2, n execute
      If p1 > p2 + v[i] then
         c ← p1
      Else
         c ← p2 + v[i]
      EndIf
      p2 ← p1
      p1 ← c
   EndFor
   Return p1
EndAlgorithm
```

A. Apelul `f([2, 7, 9, 3, 1], 5)` va returna valoarea 12.

B. Apelul `f([2, 1, 1, 2], 4)` va returna valoarea 3.

C. Apelul `f([9, 7, 8, 4, 1, 9], 6)` va returna valoarea 26.

D. Apelul `f([9, 7, 8, 4, 1, 6, 9, 2], 8)` va returna valoarea 32.

112. Se consideră algoritmul `ceFace(n, arr)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$) și `arr` este un vector de n numere întregi ($-100 \leq arr[1], arr[2], \ldots, arr[n] \leq 100$). ✓ ?

```
Algorithm CEFACE(n, arr)
   b ← True
   i ← 1
   While i ≤ n and b execute
      b ← arr[i] mod 2 = 0
      i ← i + 1
   EndWhile
   Return b
EndAlgorithm
```

Pentru care din următoarele situații algoritmul returnează `True`?

A. Dacă vectorul `arr` este format din valorile $2, 0, 4, -2, 8, -6$.

B. Dacă vectorul `arr` este format din valorile $4, 10, -1, 8, 6, 2$.

C. Dacă vectorul `arr` are doar elemente impare.

D. Dacă vectorul `arr` are cel puțin un element par.

113. Se consideră algoritmul `X(arr, n, y)`, unde n și y sunt numere naturale ($4 \leq n \leq 10^4$, $1 \leq y \leq 10^4$) și `arr` este un vector de n numere naturale nenule ($1 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^4$). ✓ ?

```
Algorithm X(arr, n, y)
   If y > n then
      Return 0
   EndIf
   For i ← 1, n − 2 execute
      arr[i+1] ← arr[i]+2 * arr[i+2]−arr[i+1]
   EndFor
   Return arr[y]
EndAlgorithm
```

Pentru care din următoarele apeluri returnează valoarea 16?

A. `X([7, 4, 3, 5, 12], 5, 3)`

B. `X([2, 5, 7, 9, 4], 5, 4)`

C. `X([1, 4, 7, 8, 10, 16], 6, 6)`

D. `X([7, 5, 2, 3, 4, 9, 2], 7, 5)`

Problemele **114.**, **115.** se referă la următorul algoritm `ceFace(n, k, r)`, unde n, k, și r sunt numere naturale nenule ($1 \leq n, k, r \leq 10^3$).

```
Algorithm ceFace(n, k, r)
   If k * k > n then
      If n > 1 then
         r ← r * (n + 1)
      EndIf
      Return r
   EndIf
   p ← 0
   sF ← 1
   While n MOD k = 0 execute
      p ← p + 1
      n ← n DIV k
      sF ← sF * k
   EndWhile
   If p > 0 then
      r ← r * (sF * k − 1) DIV (k − 1)
   EndIf
   Return ceFace(n, k + 1, r)
EndAlgorithm
```

114. Pentru care din următoarele afirmații sunt adevărate referitoare la apelul `ceFace(n, 2, 1)`? ✓ ?

A. Algoritmul returnează numărul de divizori impari ai numărului n.

B. Algoritmul returnează numărul de divizori ai numărului n.

C. Algoritmul returnează suma divizorilor numărului n.

D. Algoritmul are complexitatea de timp $O(\log n)$.

115. Care dintre următoarele afirmații sunt false referitoare la algoritmul `ceFace(n, k, r)`? ✓ ?

A. Pentru apelul `ceFace(12, 2, 1)` algoritmul returnează 28.

B. Pentru apelurile `ceFace(16, 2, 1)` și `ceFace(30, 2, 2)` algoritmul returnează 31.

C. Pentru apelul `ceFace(360, 2, 1)` algoritmul returnează 1160.

D. Pentru apelurile `ceFace(100, 2, 1)` și `ceFace(100, 2, 2)` algoritmul returnează aceeași valoare.

116. Se consideră algoritmul `ceFacea(n, l, h)`, unde n, l și h sunt numere naturale nenule ($1 \leq n, l, h \leq 10^9$). ✓ ?

```
Algorithm ceFacea(n, l, h)
   If l > h then
      Return False
   EndIf
   m ← l + (h − l) DIV 2
   If m * m * m = n then
      Return True
   Else
      If m * m * m < n then
         Return ceFacea(n, m + 1, h)
      Else
         Return ceFacea(n, l, m - 1)
      EndIf
   EndIf
EndAlgorithm
```

Precizați care afirmații de mai jos sunt adevărate referitor la algoritmul `ceFacea(n, 1, n)`.

A. Algoritmul returnează `False` pentru $n = 8$.

B. Algoritmul returnează `True` pentru $n = 5832$.

C. Există cel puțin 27 valori în intervalul $(1, 30]$ pentru care algoritmul returnează `False`.

D. Algoritmul returnează `True` pentru exact 6 valori din intervalul $[9, 513)$.

117. Se consideră algoritmul `Magic(n)`, unde n este un număr natural ($1 \leq n \leq 10^5$) ✓ ?

```
Algorithm Magic(n)
   s ← 0
   p ← 1
   While n > 0 execute
      c ← n MOD 10
      If c MOD 2 = 0 then
         s ← s + c * p
         p ← p * 10
      Else
         s ← c + s * 10
      EndIf
      n ← n DIV 10
   EndWhile
   Return s
EndAlgorithm
```

Precizați care dintre afirmațiile următoare sunt adevărate:

A. Pentru apelul `Magic(23456)` valoarea returnată de algoritm este 1255

B. Pentru apelul `Magic(987654)` valoarea returnată de algoritm este 18579

C. Algoritmul `Magic(n)` returnează un număr construit prin rearanjarea cifrelor lui n, astfel încât cifrele impare sunt păstrate în ordine directă, iar cifrele pare în ordine inversă, menținând pozițiile relative ale acestora.

D. Algoritmul `Magic(n)` inversează ordinea tuturor cifrelor lui n.

118. Se consideră algoritmul `Algo(x, y)`, unde `x` și `y` sunt numere naturale: ✓ ?

```
Algorithm Algo(x, y)
   a ← x
   b ← y
   If a = 0  OR b = 0 then
      a ← a * b
   Else
      While a ≠ b execute
         If a < b then
            a ← a + x
         Else
            b ← b + y
         EndIf
      EndWhile
   EndIf
```

```
    Write a
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt corecte:

A. Algoritmul determină cel mai mare divizor comun al numerelor x și y.

B. Algoritmul calculează cel mai mare multiplu al numerelor x sau y.

C. Algoritmul calculează cel mai mic multiplu comun a numerelor x și y.

D. Algoritmul determină cel mai mic număr divizibil fie cu x, fie cu y.

119. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural ($1 \leq n \leq 10^6$). ✓ ?

```
Algorithm ceFace(n)
    s ← 0
    p ← 1
    While n > 0 execute
        c ← n MOD 10
        p ← 1
        x ← c
        ok ← True
        For i ← 1, c − 1 execute
            x ← x * 10 + c
            p ← p * 10
        EndFor
        t ← x
        inv ← 0
        While t > 0 execute
            inv ← inv * 10 + t MOD 10
            t ← t DIV 10
        EndWhile
        If x = inv  AND x > 0 then
            s ← s + x
            p ← p * c
        EndIf
        n ← n DIV 10
    EndWhile
    Return s MOD p
EndAlgorithm
```

Precizați care dintre afirmațiile următoare sunt adevărate:

A. Algoritmul `ceFace(n)` calculează suma numerelor formate din cifrele lui n, repetate și verificate dacă sunt palindromuri, și returnează restul împărțirii la produsul cifrelor speciale.

B. Pentru apelul `ceFace(789)`, valoarea returnată de algoritm este 464.

C. Algoritmul returnează 0 dacă n conține cel puțin o cifră 0

D. Pentru apelul `ceFace(123)`, algoritmul returnează 3 deoarece doar cifrele care generează palindromuri de lungime impară contribuie la suma finală.

120. Se consideră algoritmul `ceFace(n, p)`, unde n și p sunt numere naturale nenule ($2 \leq n \leq 20, 1 \leq p \leq n$). ✓ ?

```
Algorithm ceFace(n, p)
    nrF ← 0
    pr ← p
    While n ≥ pr execute
        nrF ← nrF + n DIV pr
        pr ← pr * p
    EndWhile
    Return nrF
EndAlgorithm
```

Precizați care afirmații de mai jos sunt adevărate:

A. Algoritmul are complexitatea $O(\log n)$.

B. Algoritmul returnează suma numerelor mai mici ca n care sunt divizibile cu p.

C. Algoritmul returnează numărul de factori p din $n!$.

D. Algoritmul returnează numărul de numere divizibile cu p din intervalul $[1, n]$.

121. Se consideră algoritmul `Cifra(n)`, unde n este un număr natural ($1 \leq n \leq 10^6$). ✓ ? $frecv$ este un șir de 10 elemente, inițializate cu 0.

```
Algorithm CIFRA(n)
    temp ← n
    While temp > 0 execute
        cifra ← temp MOD 10
        frecv[cifra] ← frecv[cifra] + 1
        If frecv[cifra] > 1 then
            Return 0
        EndIf
        temp ← temp DIV 10
    EndWhile
    s ← CHECK(n)
    If s = n then
        Return 1
    EndIf
    While s > 9 execute
        aux ← s
        s ← 0
        While aux > 0 execute
            s ← s + aux MOD 10
            aux ← aux DIV 10
        EndWhile
    EndWhile
    Return s
EndAlgorithm
```

```
Algorithm CHECK(x)
    s ← 0
    For d ← 1, √x execute
        If x MOD d = 0 then
            s ← s + d
            If d * d ≠ x then
                s ← s + x DIV d
            EndIf
        EndIf
    EndFor
    Return s
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Funcția `Cifra(n)` returnează 1 dacă n este un număr perfect și toate cifrele sale sunt distincte.

B. Pentru apelul `Cifra(7654)`, valoarea returnată de algoritm este 9.

C. Funcția `Cifra(n)` returnează restul sumei divizorilor lui n redus la o singură cifră, dacă toate cifrele lui n sunt distincte.

D. Pentru apelul `Cifra(452)`, valoarea returnată de algoritm este 6.

122. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^6$). Simbolul & reprezintă operația de **AND** pe biți ($101_2 \& 110_2 = 100_2$). ✓ ?

```
Algorithm ALGO(n)
   For i ← 1, n execute
      v[i] ← i − 1
   EndFor
   For i ← n, 2 execute
      j ← (i − 1) & ((i − 1) − 1)
      aux ← v[i]
      v[i] ← v[j + 1]
      v[j + 1] ← aux
   EndFor
   Return v
EndAlgorithm
```

```
Algorithm CEFACE(n)
   p ← ALGO(n)
   suma ← 0
   For i ← 1, n execute
      If (p[i] & (i − 1)) = (i − 1) then
         suma ← suma + 1
      EndIf
   EndFor
   Return suma
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `ceFace(2345)`, valoarea returnată de algoritm este 587.

B. Algoritmul determină suma palindromurilor formate din fiecare cifră repetată un număr egal cu valoarea sa, modulată de produsul cifrelor din număr care generează palindromurilor valide.

C. Pentru apelul `ceFace(7546)`, valoarea returnată de algoritm este 1887.

D. Algoritmul calculează rădăcina produsului dintre palindromurilor formate din fiecare cifră repetată de atâtea ori cât reprezintă valoarea cifrei.

123. Se consideră algoritmul `S(n, s, d)`, unde n, s și d sunt numere naturale nenule ✓ ? $(1 \leq n, s, d \leq 10^4)$.

```
Algorithm S(n, s, d)
   If s > n then
      Return False
   EndIf
   If s = n then
      Return True
   EndIf
   Return S(n, s + d, d + 2)
EndAlgorithm
```

Precizați care afirmații sunt adevărate referitor la apelul algoritmului `S(n, 0, 1)`:

A. Algoritmul verifică dacă n este pătrat perfect.

B. Algoritmul verifică dacă n poate fi scris ca sumă de numere naturale consecutive impare începând de la 1.

C. Există exact 19 valori în intervalul $[71, 734]$ pentru care algoritmul returnează `True`.

D. Algoritmul returnează `False` pentru exact 2 numere din mulțimea $\{73, 9, 442, 841, 576, 962\}$.

124. Se consideră algoritmul `ceFace(n, k)`, unde `n` și `k` reprezintă numere naturale: ✓ ?

```
Algorithm ceFace(n, k)
   c ← 0
   While n > 0 execute
      x ← n
      y ← 0
      len ← 0
      For i ← 1, k execute
         If x = 0 then
            Return c
         EndIf
         y ← y * 10 + (x MOD 10)
         x ← x DIV 10
         len ← len + 1
      EndFor
      If len < k then
         Return c
      EndIf
      unique ← True
      temp ← y
      While temp > 0 execute
         d ← (temp MOD 10)
         frecv[d] ← frecv[d] + 1
         If frecv[d] > 1 then
            unique ← False
            Break
         EndIf
         temp ← temp DIV 10
      EndWhile
      If unique then
         c ← c + 1
      EndIf
      n ← n DIV 10
   EndWhile
   Return c
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Secvențele verificate sunt reordonate și comparate în ordine descrescătoare pentru a determina unicitatea cifrelor.

B. Algoritmul determină numărul de subsecvențe de k cifre consecutive în n, unde toate cifrele din subsecvență sunt distincte

C. Dacă lungimea rămasă a numărului n este mai mică decât k, algoritmul returnează 0 fără a mai verifica alte secvențe.

D. Algoritmul calculează suma cifrelor distincte din secvențele de lungime k în loc să returneze numărul lor.

125. Se consideră algoritmul `SpecialBits(n)`, unde `n` reprezintă un număr natural ($1 \leq n \leq 10^5$). Operația SHL (Shift Left) este definită ca o operație de deplasare pe biți (bitwise shift left), care deplasează biții unui număr spre stânga cu un anumit număr de poziții. ✓ ?

```
Algorithm SpecialBits(n)
   If n = 0 then
      Return 0
   EndIf
   count ← 0
   mask ← 1
   While mask ≤ n execute
      temp ← n  AND  ((mask SHL 1) − 1)
      next ← temp  OR  mask
      If next ≤ n then
         count ← count + 1
         aux ← next
         While aux > 0 execute
```

```
            If (aux AND mask) ≠ 0 then
               test ← aux AND (aux −1)
               If test ≤ n then
                  count ← count + 1
                  aux ← test
               Else
                  aux ← aux AND (aux − 1)
               EndIf
            Else
               aux ← aux AND (aux − 1)
            EndIf
         EndWhile
      EndIf
      mask ← mask SHL 1
   EndWhile
   Return count
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează numărul de operații posibile asupra n, care respectă constrângerile impuse de mask și bit-shifting.

B. Dacă n este o putere a lui 2, algoritmul va returna întotdeauna $n - 1$.

C. Algoritmul verifică toate submask-urile posibile ale lui n, indiferent de constrângeri.

D. Pentru apelul `SpecialBits(12)`, algoritmul va returna valoarea 9.

126. Se consideră algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10^4$). ✓ ?

```
1: Algorithm ceFace(a, b)
2:    x ← a
3:    y ← b
4:    While x ≠ y execute
5:       If x < y then
6:          x ← x + a
7:       Else
8:          y ← y + b
9:       EndIf
10:   EndWhile
11:   Return x
12: EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru apelul `ceFace(24, 36)`, algoritmul returnează 72.

B. Pentru apelul `ceFace(52, 14)`, algoritmul returnează 728.

C. Algoritmul returnează cel mai mare divizor comun al lui a și b.

D. Dacă `Linia 11` ar fi schimbată cu `Return a * b DIV y`, algoritmul ar returna cel mai mare divizor comun al lui a și b.

Complexitatea algoritmilor

Acest capitol acoperă

- De ce sunt importante complexitățile în analiza algoritmilor;
- Cum se determină complexitatea unui algoritm;
- Exemple de complexități comune și implementările lor;
- Diferența între complexități aditive și multiplicative.

5.1 Teorie

Complexitățile algoritmilor reprezintă un concept fundamental care se află la intersecția dintre informatică și matematică. Acestea ne permit să estimăm numărul aproximativ de pași necesari pentru ca un algoritm să rezolve o problemă, constituind un instrument esențial în alegerea algoritmului optim în funcție de cerințele de eficiență. Deși complexitatea poate fi analizată atât din perspectiva temporală, cât și spațială, în acest capitol ne vom concentra pe analiza temporală a algoritmilor.

5.1.1 Elementele Fundamentale ale Complexităților

- **Complexitate constantă** $O(1)$:
  - Timpul de execuție este independent de dimensiunea datelor de intrare
  - Operațiile elementare (atribuiri, operații aritmetice simple, comparații) au complexitate constantă
- **Complexitate liniară** $O(n)$:
  - Numărul de pași este direct proporțional cu dimensiunea datelor de intrare
- **Complexitate logaritmică** $O(\log n)$:
  - Numărul de pași crește lent pe măsură ce dimensiunea datelor de intrare devine mai mare
- **Complexitate radical** $O(\sqrt{n})$:
  - La fel ca în cazul complexității logaritmice, numărul de pași crește lent pe măsură ce dimensiunea intrării devine mai mare
- **Complexitate pătratică** $O(n^2)$:
  - Numărul de pași crește proporțional cu pătratul dimensiunii datelor de intrare în cazul buclelor imbricate
- **Complexitate exponențială** $O(2^n)$:

  - Timpul de execuție crește rapid, fiind adesea ineficient pentru valori mari ale intrării

- **Teorema Master (pentru algoritmi Divide and Conquer)**:
  - Fie un algoritm recursiv de tipul Divide and Conquer. Complexitatea sa poate fi scrisă sub forma $T(n) = a \cdot T(n/b) + \Theta(n^k \cdot \log^p(n))$.
  - Clasa de complexitate a algoritmului poate fi determinată astfel:
    - Cazul 1: Dacă $a > b^k \Rightarrow T(n) = \Theta(n^{\log_b(a)})$
    - Cazul 2: Dacă $a = b^k$:
      - Pentru $p > -1 \Rightarrow T(n) = \Theta(n^{\log_b(a)} \cdot \log^{p+1}(n))$
      - Pentru $p = -1 \Rightarrow T(n) = \Theta(n^{\log_b(a)} \cdot \log(\log(n)))$
      - Pentru $p < -1 \Rightarrow T(n) = \Theta(n^{\log_b(a)})$
    - Cazul 3: Dacă $a < b^k$:
      - Pentru $p \geq 0 \Rightarrow T(n) = \Theta(n^k \cdot \log^p(n))$
      - Pentru $p < 0 \Rightarrow T(n) = O(n^k)$

5.1.2 Exemple de implementare

Exemplu de Complexitate Liniară $O(n)$

```
1: For i ← 1, n execute
2:     Write i
3: EndFor
```

Exemplu de Complexitate Logaritmică $O(\log n)$

```
1: i ← 1
2: While i ≤ n execute
3:     Write i
4:     i ← i × 2
5: EndWhile
```

Exemplu de Complexitate $O(\sqrt{n})$

```
1: i ← 1
2: While i * i ≤ n execute
3:     Write i
4:     i ← i + 1
5: EndWhile
```

Exemplu de Complexitate Pătratică $O(n^2)$

```
1: For i ← 1, n execute
2:     For j ← 1, n execute
3:         Write i, j
4:     EndFor
5: EndFor
```

Exemplu de Complexitate Exponențială $O(2^n)$

```
1: Algorithm FIBONACCI(n)
2:     If n ≤ 1 then
3:         Return 1
4:     Else
5:         Return FIBONACCI(n − 1) + FIBONACCI(n − 2)
6:     EndIf
7: EndAlgorithm
```

5.1.3 Analiza Complexității

- **Complexitate temporală:** Analiza unui algoritm în funcție de numărul de operații efectuate în timpul execuției.
- **Complexitate spațială:** Analiza resurselor de memorie utilizate de algoritm în timpul execuției.

Observație importantă: Constantele sunt ignorate în analiza Big-O, deoarece nu influențează creșterea pe termen lung a complexității. De exemplu, un algoritm de complexitate $O(2n)$ este considerat echivalent cu unul de complexitate $O(n)$.

5.2 Probleme

127. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm ceFace(n)
    k ← 0
    For i ← n, n DIV 3, −1 execute
        For j ← 3, n, j ← j * 3 execute
            k ← k + n DIV 3
        EndFor
    EndFor
    Return k
EndAlgorithm
```

Care este complexitatea de timp a algoritmului `ceFace(n)`?

A. $O(n^2)$

B. $O(n^2 \cdot \log_3(n))$

C. $O(n)$

D. $O(n \cdot \log_3(n))$

128. Se consideră algoritmul `ceFace(n, v)`, unde v este un tablou unidimensional cu cel mult 10^6 elemente și n reprezintă numărul elementelor din v. Subalgoritmul $A(n, v)$ produce o schimbare asupra tabloului v, sortând elementele crescător folosind metoda Selection Sort. ✓ ?

```
Algorithm ceFace(n, v)
    A(n, v)
    For i ← 1, n − 2 execute
        For j ← i + 1, n − 1 execute
            st ← j + 1
            dr ← n
            k ← n
            While st ≤ dr execute
                k ← (st + dr)/2
                If v[i] + v[j] > v[k] then
```

```
                st ← k + 1
            Else
                dr ← k − 1
            EndIf
        EndWhile
        cnt ← cnt + (dr − j)
    EndFor
  EndFor
  Return cnt
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul `ceFace` are complexitatea $O(n)$;

B. Algoritmul `ceFace` are complexitatea $O(n^2 \log(n))$ și calculează numărul tripletelor (i, j, k) din tabloul v cu proprietatea că $v[i] - v[j] + v[k] > 0$.

C. Algoritmul `ceFace` are complexitatea $O(n^3)$ și calculează numărul tripletelor (i, j, k) din tabloul v cu proprietatea că $v[i] + v[j] > 0$.

D. Algoritmul `ceFace` calculează numărul triunghiurilor distincte care au lungimile laturilor în tabloul v.

129. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm CEFACE(n)
  k ← 0
  m ← 1
  While k ≤ n execute
    ind ← 1
    While ind ≤ m execute
      Write j
      ind ← ind + 1
    EndWhile
    k ← k + 1
    m ← m * 3
  EndWhile
EndAlgorithm
```

Care este complexitatea de timp a algoritmului `ceFace(n)`?

A. $O(n \cdot \log_3(n))$

B. $O(n^2)$

C. $O(3^n)$

D. $O(n \cdot m)$

130. Se consideră algoritmul `Ack(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm ACK(n)
  If n ≠ 1 then
    Ack(n - 1)
    Write n
    Ack(n - 1)
    Write n − 1
  EndIf
EndAlgorithm
```

Care dintre următoarele afirmații referitoare la algoritmul `Ack(n)` sunt false?

A. Complexitatea de timp a algoritmului `Ack(n)` este $O(n)$.

B. Complexitatea de timp a algoritmului `Ack(n)` este $O(2^n)$.

C. Complexitatea de timp a algoritmului `Ack(n)` este $O(n^2)$.

D. Niciuna dintre afirmațiile de mai sus.

131. Se consideră algoritmul `ceFace(n, i)`, unde n și i sunt numere naturale nenule $(1 \leq n, i \leq 10^4)$. ✓ ?

```
Algorithm ceFace(n, i)
   If n = 1 then
      Return 1
   Else
      m ← n DIV 2
      If i MOD 2 = 0 then
         Return ceFace(m, i)−i
      Else
         Return ceFace(m, i)+i
      EndIf
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații referitoare la algoritmul `ceFace(n, i)` sunt adevărate?

A. Complexitatea de timp a algoritmului `ceFace(n, i)` este $O(\log_2(n))$.

B. Complexitatea de timp a algoritmului `ceFace(n, i)` este $O(n)$.

C. Complexitatea de timp a algoritmului `ceFace(n, i)` este $O(n^2)$.

D. Apelul `ceFace(17, 5)` returnează valoarea 21.

132. Se consideră algoritmul `X(n)`, unde n este un număr natural nenul $(1 \leq n \leq 10^4)$. ✓ ?

```
Algorithm X(n)
   s ← 0
   ind ← n
   While ind > 0 execute
      j ← 1
      While j ≤ ind execute
         s ← s + j + 5
         j ← j * 5
         ind ← ind DIV 5
      EndWhile
   EndWhile
   Return s
EndAlgorithm
```

Care este complexitatea de timp a algoritmului `X(n)`?

A. $O(n^2)$

B. $O(n \cdot \log_5 n)$

C. $O(5^n)$

D. $O(\log_5 n)$

133. Care dintre următorii algoritmi sortează în cel mai eficient mod elementele unui vector de n elemente cu valori din mulțimea $\{0, 1, 3, 5, 8, 9\}$? ✓ ?

A. QuickSort B. MergeSort C. CountSort D. BubbleSort

134. Se consideră vectorul ordonat crescător a cu n elemente numere naturale $(1 \leq n \leq 10^4)$ și un număr natural q $(1 \leq n \leq 10^9)$. Dorim să aflăm numărul maxim de elemente din a ale căror sumă nu depășește valoarea numărului q. Care este complexitatea celui mai eficient subprogram care rezolvă problema? (excluzând complexitatea sortării șirului) ✓ ?

A. $O(\log n)$; B. $O(n \log n)$; C. $O(n \log^2 n)$; D. $O(n^2 \log n)$.

Acest capitol acoperă

- Ce sunt subprogramele și de ce sunt utile?
- Cum definim și folosim subprograme?
- Tipurile de subprograme: funcții și proceduri.
- Reutilizarea subprogramelor pentru organizarea codului.

6.1 Teorie

Introducere

În acest subcapitol discutăm despre subprograme, un concept foarte important în programare. Subprogramele reprezintă un mod de a împărți un program mai mare în secțiuni mai mici și reutilizabile, facilitând organizarea și gestionarea codului.

Elementele Fundamentale ale Subprogramelor

- **Funcții:**
  - Subprograme care returnează o valoare.
- **Proceduri:**
  - Subprograme care nu returnează o valoare.

Definirea Subprogramelor

Un subprogram este definit o singură dată, dar poate fi apelat de oricâte ori este necesar în contextul unui program. Definirea unui subprogram implică specificarea unui nume și a unei liste de parametri. În cazul funcțiilor, este necesară utilizarea și a unei instrucțiuni `return` pentru a returna o valoare.

Funcția `suma`

```
1: Algorithm SUMA(a, b)
2:     Return a + b
3: EndAlgorithm
```

Exemplul prezentat mai jos ilustrează o funcție simplă care primește doi parametri, a și b, și returnează suma acestora. Funcția poate fi apelată de mai multe ori:

Apelarea funcției `suma`

```
1: result ← SUMA(3, 5)
2: Write result
```

Parametrii Subprogramelor

Subprogramele pot include parametri, care reprezintă variabilele de intrare necesare pentru a executa operațiile definite. Acești parametri pot fi de mai multe tipuri, iar atât numărul, cât și tipul lor trebuie specificate la momentul apelării subprogramului.

Funcția `inmultire`

```
1: Algorithm INMULTIRE(a, b)
2:     Return a * b
3: EndAlgorithm
```

În cazul în care un subprogram nu primește parametri, lista de parametri este omisă:

Funcția `hello`

```
1: Algorithm HELLO
2:     Write "Salutare viitor student UBB Info!"
3: EndAlgorithm
```

Reutilizarea Subprogramelor

Unul dintre cele mai mari avantaje ale subprogramelor este reutilizarea. Odată definit, un subprogram poate fi apelat de oricâte ori este nevoie, ceea ce facilitează modularitatea și reduce totodată redundanța codului:

Apelarea funcției `suma` pentru reutilizare

```
1: result1 ← SUMA(10, 20)
2: result2 ← SUMA(15, 25)
```

Concluzie

Subprogramele sunt un instrument esențial pentru organizarea codului în secțiuni reutilizabile. Acestea simplifică dezvoltarea aplicațiilor, reduc volumul de cod scris de programatori și contribuie la creșterea clarității și modularității codului.

6.2 Probleme

135. Se consideră algoritmul `ceFace(arr, n, y)`, unde `n` și `y` sunt numere naturale ✓ ?
$(1 \leq n \leq 10^4, 1 \leq y \leq 10^4)$ și `arr` este un vector de `n` numere naturale nenule $(1 \leq \texttt{arr}[1], \texttt{arr}[2], \ldots, \texttt{arr}[n] \leq 10^4)$.

```
Algorithm ceFace(arr, n, y)
   rezultat ← 0
   For i ← 1, n execute
      If arr[i] > y then
         rezultat ← rezultat + 1
      EndIf
   EndFor
   Return rezultat
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Algoritmul are complexitate $O(n)$;

B. Funcția returnează numărul elementelor din `arr` care sunt mai mari decât `y`;

C. Algoritmul returnează 0 dacă toate elementele din `arr` sunt mai mici decât `y`+1;

D. Algoritmul poate fi optimizat pentru a reduce complexitatea la $O(\log n)$.

136. Se consideră algoritmul `ceFace(v, n, k)`, unde `n` și `k` sunt numere naturale ($1 \leq n, k \leq 10^6$), iar `v` este un vector de `n` numere naturale nenule ($1 \leq v[1], v[2], \ldots, v[n] \leq 10^9$). ✓ ?

```
Algorithm ceFace(v, n, k)
   k ← k MOD n
   temp        ← vector auxiliar de
dimensiune n
   For i ← 1, n execute
      temp[(i + k − 1) MOD n + 1] ← v[i]
   EndFor
   For i ← 1, n execute
      v[i] ← temp[i]
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Algoritmul are complexitate $O(n)$;

B. Realizează o rotație a vectorului la dreapta cu k poziții;

C. Algoritmul face ca in vectorul temp doar pozitiile pare sa fie ocupate;

D. Modifică vectorul original.

137. Se consideră algoritmul `ceFace(v, n)`, unde `n` este un număr natural nenul ($1 \leq n \leq 10^6$), iar `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$). ✓ ?

```
Algorithm ceFace(v, n)
   count ← 0
   For i ← 1, n − 1 execute
      If v[i] > v[i + 1] then
         temp ← v[i]
         v[i] ← v[i + 1]
         v[i + 1] ← temp
         count ← count + 1
      EndIf
   EndFor
   Return count
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Algoritmul implementează o iterație din bubble sort;

B. Complexitatea este $O(n)$;

C. Numărul returnat reprezintă câte interschimbări s-au făcut;

D. Vectorul va fi complet sortat după execuție.

138. Se consideră algoritmul `ceFace(v, n)`, unde `n` este un număr natural nenul ($1 \leq n \leq 10^6$), iar `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$). ✓ ?

```
Algorithm ceFace(v, n)
   Sum ← v[1]
```

```
    currentSum ← v[1]
    For i ← 2, n execute
        currentSum ← max(v[i], currentSum + v[i])
        Sum ← max(Sum, currentSum)
    EndFor
    Return Sum
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul găsește suma maximă a unei subsecvențe continue;

B. Complexitatea este $O(n)$;

C. Pentru un vector cu toate elementele negative, se va returna cel mai mare element;

D. Algoritmul modifică vectorul original.

139. Se consideră algoritmul `ceFace(n)`, unde `n` este un număr natural nenul ($1 \leq n \leq 10^9$). ✓ ?

```
Algorithm ceFace(n)
    count ← 0
    For i ← 1, i · i ≤ n execute
        If n mod i = 0 then
            count ← count + 1
            If i · i ≠ n then
                count ← count + 1
            EndIf
        EndIf
    EndFor
    Return count
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Algoritmul numără divizorii proprii a lui `n`;

B. Complexitatea este $O(\sqrt{n})$;

C. Pentru $n = 1$, rezultatul este 0;

D. Pentru numere prime, rezultatul este întotdeauna 2.

140. Se consideră algoritmul `ceFace(n, k)`, unde `n` și `k` sunt numere naturale nenule ($1 \leq n, k \leq 10^4$). ✓ ?

```
Algorithm ceFace(n, k)
    s ← 1
    For i ← 1, n execute
        If i mod k = 0 then
            s ← s · i
        EndIf
    EndFor
    Return s
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Subalgoritmul poate returna orice număr din intervalul $[0, 10^4]$;

B. Dacă ar lipsi linia cu `if(i % k == 0)`, rezultatul funcției ar fi factorialul lui `n`;

C. Rezultatul funcției va fi un număr divizibil cu `k` dacă și numai dacă `n` $\geq$ `k`;

D. Pentru cazul în care `n` $<$ `k`, subalgoritmul returnează mereu o cifră.

141. Se consideră algoritmul `ceFace(n)`, unde `n` este un număr natural nenul ($2 \leq n \leq 10^9$). ✓ ?

```
Algorithm ceFace(n)
   For i ← 2, n DIV 2 execute
      If n mod i = 0 then
         Return 0
      EndIf
   EndFor
   Return 1
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează câți divizori are numărul `n`;

B. Algoritmul verifică dacă numărul este prim, dar se poate face mai eficient;

C. Algoritmul are complexitatea $O(\sqrt{n})$;

D. Niciun răspuns de mai sus.

142. Se consideră algoritmul `ceFace(v, n, k)`, unde `n` și `k` sunt numere naturale ($1 \leq k < n \leq 10^6$), iar `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$). ✓ ?

```
Algorithm ceFace(v, n, k)
   For i ← 2, n execute
      v[i] ← v[i] + v[i − 1]
   EndFor
   Return v[n] − v[k]
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Algoritmul poate returna și valori negative;

B. Algoritmul este pătratic;

C. Algoritmul calculează suma subsecvenței dintre indicii $[k + 1, n]$;

D. Dacă numerele din vector sunt pozitive și elementele ordonate crescător, rezultatul este întotdeauna pozitiv și nenul.

143. Se consideră algoritmul `ceFace(n)` definit alăturat, unde n este un număr natural, $n \leq 1000$ și a este un tablou unidimensional cu cel mult 10^5 elemente, inițial toate nule. ✓ ?

```
Algorithm ceFace(n)
    a[1] ← 1
    cf ← 1
    For i ← 1, n execute
        cr ← 0
        For j ← 1, cf execute
            temp ← a[j] * i + cr
            a[j] ← temp MOD 10
            cr ← temp DIV 10
        EndFor
        While cr ≠ 0 execute
            cf ← cf + 1
            a[cf] ← cr MOD 10
            cr ← cr DIV 10
        EndWhile
    EndFor
    For i ← cf, 1, −1 execute
        scrie a[i]
    EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul scrie pe ecran numărul de cifre al tuturor valorilor $x!$, unde $x \in [1, n]$;

B. Algoritmul scrie pe ecran valoarea lui $n!$;

C. Pentru $n = 10$, programul afișează pe ecran 3628800;

D. Pentru $n = 5$, programul afișează 11123.

144. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi ✓ ? $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `n` este un număr natural nenul $(1 \leq n \leq 10^6)$.

```
Algorithm ceFace(v, n)
    result ← 0
    i ← 1
    While i ≤ n execute
        While i < n and v[i] ≤ v[i + 1] execute
            i ← i + 1
        EndWhile
        While i < n and v[i] ≥ v[i + 1] execute
            i ← i + 1
        EndWhile
        result ← result + 1
        i ← i + 1
    EndWhile
    Return result
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul numără vârfurile din vector (elemente precedate de o secvență crescătoare și urmate de una descrescătoare);

B. Complexitatea este $O(n)$;

C. Pentru un vector strict crescător returnează $n/2$;

D. Pentru un vector sortat crescător returnează 1.

145. Se consideră algoritmul `ceFace(n)`, unde `n` este un număr natural nenul $(1 \leq n \leq$ ✓ ? $10^6)$.

```
Algorithm ceFace(n)
    sum ← 0
    For i ← 1, n execute
        If n mod i = 0 then
            d ← i
            While d > 0 execute
                sum ← sum + (d mod 10)
                d ← d DIV 10
            EndWhile
        EndIf
    EndFor
    Return sum
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Algoritmul calculează suma cifrelor tuturor divizorilor lui `n`;

B. Complexitatea este $O(n)$;

C. Pentru numere prime returnează 1 + suma cifrelor lui `n`;

D. Pentru `n = 1` returnează 1.

146. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi ✓ ? $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `n` este un număr natural nenul $(1 \leq n \leq 10^6)$.

```
Algorithm ceFace(v, n)
    maxLen ← 2
    len ← 2
    diff ← v[2] − v[1]
    For i ← 2, n − 1 execute
        If v[i + 1] − v[i] = diff then
            len ← len + 1
        Else
            maxLen ← max(maxLen, len)
            len ← 2
            diff ← v[i + 1] − v[i]
        EndIf
    EndFor
    Return max(maxLen, len)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul găsește lungimea celei mai lungi progresii aritmetice consecutive din vector;

B. Complexitatea este $O(n)$;

C. Pentru un vector cu valori constante returnează n;

D. Pentru un vector cu elemente distincte returnează întotdeauna 2.

147. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi ✓ ? $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `n` este un număr natural nenul $(1 \leq n \leq 10^3)$.

```
Algorithm ceFace(v, n)
```

```
    result ← 0
    For i ← 1, n execute
        For j ← i, n execute
            If v[i] = v[j] then
                result ← max(result, j − i)
            EndIf
        EndFor
    EndFor
    Return result
EndAlgorithm
```

Subprogramul max(a,b) returnează maximul dintre cele doua numere, unde a și b numere întregi.

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul găsește distanța maximă între două elemente egale;

B. Are complexitate $O(n)$;

C. Pentru un vector cu elemente distincte returnează 0;

D. Pentru un vector cu elemente constante returnează $n - 1$.

148. Se consideră algoritmul `ceFace(v, n, k)`, unde `v` este un vector de `n` numere întregi $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9), 1 \leq n \leq 10^3$, iar `k` este un număr natural nenul $(1 \leq k \leq n)$. ✓ ?

```
Algorithm ceFace(v, n, k)
    sum ← 0
    For i ← 1, k execute
        sum ← sum + v[i]
    EndFor
    maxSum ← sum
    For i ← k + 1, n execute
        sum ← sum + v[i] − v[i − k]
        maxSum ← max(maxSum, sum)
    EndFor
    Return maxSum
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul găsește suma maximă a `k` elemente consecutive;

B. Are complexitate $O(n \cdot k)$;

C. Pentru `k = 1` returnează maximul din vector;

D. Pentru `k = n` returnează suma tuturor elementelor.

149. Se consideră algoritmul `ceFace(n)`, unde `n` este un număr natural $(1 \leq n \leq 10^9)$. ✓ ?

```
Algorithm ceFace(n)
    result ← 0
    While n > 0 execute
        cifra ← n mod 10
        If cifra mod 2 = 0 then
            result ← result · 10 + cifra
```

```
        EndIf
        n ← n DIV 10
    EndWhile
    temp ← 0
    While result > 0 execute
        temp ← temp · 10 + result  mod 10
        result ← result DIV 10
    EndWhile
    Return temp
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul păstrează doar cifrele pare din număr în ordinea lor inițială;

B. Pentru un număr care conține doar cifre impare returnează 0;

C. Are complexitate $O(\log n)$;

D. Modifică ordinea cifrelor în numărul rezultat.

150. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi ✓ ? $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `n` este un număr natural nenul $(1 \leq n \leq 10^6)$.

```
Algorithm CEFACE(v, n)
    result ← 0
    count ← 0
    For i ← 1, n execute
        If v[i] > 0 then
            count ← count + 1
            If count > result then
                result ← count
            EndIf
        Else
            count ← 0
        EndIf
    EndFor
    Return result
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul găsește lungimea celei mai lungi secvențe de numere pozitive consecutive;

B. Pentru un vector cu toate elementele negative returnează 0;

C. Complexitatea este $O(n^2)$;

D. Pentru un vector cu toate elementele pozitive returnează $n/2$.

7 Recursivitate

Acest capitol acoperă

- Înțelegerea conceptului de recursivitate și elementele sale fundamentale
- Aplicarea modelului matematic și ecuațiilor de recurență

7.1 Teorie

Recursivitatea este o tehnică fundamentală în programare care permite rezolvarea problemelor complexe prin descompunerea lor în subprobleme mai simple. Un algoritm recursiv este un algoritm care se autoapelează pentru a rezolva o problemă mai mică de același tip, reprezentând o implementare directă a principiului matematic al inducției.[2]

Elementele Fundamentale ale Recursivității

- **Cazul de bază (condiția de terminare):**
  - Reprezintă scenariul trivial al problemei, care poate fi rezolvat direct
  - Asigură convergența algoritmului, prevenind buclele infinite
  - Pot exista unul sau mai multe cazuri de bază
- **Pasul recursiv:**
  - Reduce problema curentă la una sau mai multe instanțe mai simple
  - Fiecare apel recursiv apropie execuția de cazul de bază

Tipuri de Recursivitate

- **Recursivitate liniară:**
  - Implică un singur apel recursiv per execuție
  - Exemplu: calculul factorialului, căutare binară
- **Recursivitate arborescentă:**
  - Implică multiple apeluri recursive per execuție
  - Exemplu: Fibonacci recursiv naiv
- **Recursivitate indirectă (mutuală):**
  - Apare atunci când funcțiile se apelează reciproc
  - Este utilă în procesarea structurilor de date complexe

Modelul Matematic și Analiza Formală

Recursivitatea poate fi formalizată prin ecuații de recurență, care exprimă relația matematică dintre soluția unei probleme și soluțiile subproblemelor corespunzătoare.

Pentru factorial:

$$n! = \begin{cases} 1 & \text{dacă } n = 0 \\ n \cdot (n-1)! & \text{dacă } n > 0 \end{cases}$$

Pentru Fibonacci:

$$F_n = \begin{cases} 0 & \text{dacă } n = 0 \\ 1 & \text{dacă } n = 1 \\ F_{n-1} + F_{n-2} & \text{dacă } n > 1 \end{cases}$$

Structura Generală a unei Funcții Recursive

```
1: Algorithm FunctieRecursiva(parametri)
2:     If condiție de oprire then
3:         Return rezultat_caz_bază
4:     Else
5:         procesare_intermediară
6:         Return FunctieRecursiva(parametri_reduși)
7:     EndIf
8: EndAlgorithm
```

Analiza Complexității

Pentru algoritmi recursivi, complexitatea temporală este adesea exprimată prin relații de recurență:

$$T(n) = aT(n/b) + f(n)$$

unde:

- a = numărul de apeluri recursive
- b = factorul de reducere a problemei
- $f(n)$ = costul operațiilor non-recursive

Calculul factorialului

Pornind de la modelul matematic si structura generală a unei funcții recursive, se poate construi algoritmul pentru calculul factorialului:

Algoritmul Factorial Recursiv

```
1: Algorithm FACTORIAL(n)
2:     If n = 0 then
3:         Return 1                                   ▷ Cazul de bază
4:     Else
5:         Return n * FACTORIAL(n − 1)                ▷ Pasul recursiv
6:     EndIf
7: EndAlgorithm
```

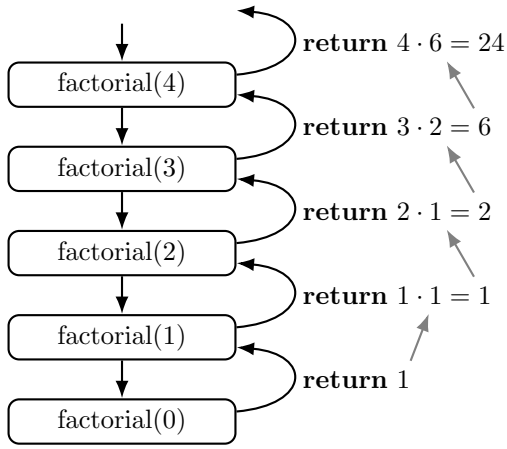


Figura 7.1 Arbore de apeluri recursive pentru calculul lui 4!. Fiecare nod reprezintă un apel al funcției `Factorial`, care se reduce până când se atinge cazul de bază.

Aplicații și Tipuri de Probleme

Recursivitatea este utilă în rezolvarea mai multor clase de probleme:

- **Probleme de tip Divide et Impera:**
  - Sortare prin interclasare (Merge Sort)
  - Căutare binară
- **Parcurgeri în structuri ierarhice:**
  - Parcurgerea arborilor binari
  - Traversarea grafurilor
- **Probleme combinatoriale:**
  - Generarea permutărilor
  - Generarea combinărilor
  - Problema turnurilor din Hanoi

7.2 Probleme

151. Se consideră algoritmul $mn(n, d, p, u)$, unde n, p, d sunt numere întregi, iar u este o variabilă de tip logic (poate lua valorile `True` sau `False`). ✓ ?

```
Algorithm MN(n, d, p, u)
   p ← n + 6
   Write n
   If d ≤ 0 OR n ≤ 0 then
      Return
   EndIf
   If u then
      MN(n + d, d − 2, p, n + d ≥ p)
   Else
      MN(n − d, d + 2, p, False)
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Ultima valoare afișată de apelul $mn(6, 3, 0, true)$ este egală cu ultima valoare afișată de apelul $mn(1, 3, 0, true)$.

B. Pentru orice $d < 0$, algoritmul va afișa o singură valoare.

C. Algoritmul generează o secvență de numere care alternează între creștere (cu d) și descreștere (cu $d + 2$) până când d devine negativ sau n devine negativ.

D. Algoritmul generează toate modalitățile posibile de a ajunge de la n la 0 prin scăderi succesive cu $d, d + 2, d + 4$, etc.

152. Se consideră următorul algoritm care primește un vector de biți b de lungime n și începe procesarea cu valoarea inițială $\ell = 1$: ✓ ?

```
Algorithm c(x, y)
   If x MOD 2 = 1 then
      Return x · y
   Else
      Return x + y
   EndIf
EndAlgorithm

Algorithm p(b, k, n, ℓ)
   If k > n then
      Return ℓ
   EndIf
   If b[k] = 1 then
      n1 ← p(b, k + 1, n, ℓ + 3)
      n2 ← p(b, k + 1, n, ℓ · 2)
      Return c(n1, n2)
   Else
      Return p(b, k + 1, n, ℓ)
   EndIf
EndAlgorithm
```

Care dintre următoarele valori pentru vectorul b va produce rezultatul 76 când `p(b, 1, n, 1)` este apelat?

A. $b = [0, 0, 1, 1]$

B. $b = [0, 0, 0, 1]$

C. $b = [1, 1, 0, 0]$

D. $b = [0, 1, 1, 1]$

153. Se consideră algoritmul `g(x)`, unde x este un număr natural nenul ($1 \leq x \leq 10^5$): ✓ ?

```
Algorithm g(x)
   If x > 1 then
      Write f(x), " "
      g(x DIV 2)
      Write x + 3, " "
      g(x DIV 2)
   EndIf
EndAlgorithm
```

```
Algorithm f(x)
   If x = 0 then
      Return 0
   Else
      Return (x MOD 10) + f(x DIV 10)
   EndIf
EndAlgorithm
```

Ce se afișează în urma apelului `g(10)`?

A. 1 5 2 5 8 3 5 13 5 2 5 8 5 5

B. 1 5 2 5 8 2 5 13 5 2 5 8 2 5

C. 1 5 2 5 8 2 5 13 5 3 5 8 2 5

D. 1 4 3 5 8 2 3 13 5 2 5 8 2 5

154. Se dă algoritmul `comp(s, a, l)`, unde s este un șir de caractere, iar a, l sunt numere întregi. Algoritmul `vocala(c)` returnează `true` dacă caracterul c este o vocală. ✓ ?

```
Algorithm comp(s, a, l)
   If l ≤ 1 then
      If l = 1 then
         Return s[a]
      Else
         Return ""
      EndIf
   EndIf
   m ← l DIV 2
   If vocala(s[a + m − 1]) then
      r1 ← comp(s, a, m)
      r2 ← _____
      Return r1 + " * " + r2
   EndIf
   Return comp(s, a, l − 1)
EndAlgorithm
```

Cu ce secvență de cod trebuie completat algoritmul, astfel încât în urma apelului `comp("babaeiou", a, l)` să se afișeze **b*b*e*i*o*u**?

A. $comp(s, a + m - 1, l - m - 1)$

B. $comp(s, a + m + 1, l - m)$

C. $comp(s, a + m, l - m)$

D. $comp(s, a + m, l - m + 1)$

155. Se consideră algoritmul `algo(v, n)`, unde n este un număr natural ($1 \leq n \leq 10^4$) și v este un vector cu n elemente naturale ($v[1], v[2], \ldots, v[n]$): ✓ ?

```
Algorithm algo(v, n)
   If n = 0 then
      Return 0
   EndIf
   If n = 1 then
      Return v[1]
   EndIf
   include ← v[n]+ algo(v, n − 2)
   exclude ← algo(v, n − 1)
   Return g(include, exclude)
EndAlgorithm
```

```
Algorithm g(a, b)
   If a > b then
      Return a
   Else
      Return b
   EndIf
EndAlgorithm
```

Precizați care dintre afirmațiile de mai jos sunt adevărate:

A. Algoritmul determină suma maximă posibilă a unei subsecvențe de elemente dintr-un vector, unde suma include cel puțin două elemente aflate pe poziții consecutive în șir.

B. Pentru apelul `algo([3, 2, 7, 10, 20, 1, 5], 7)` algoritmul va afișa valoarea 35.

C. Algoritmul determină suma maximă posibilă a tuturor elementelor dintr-un vector.

D. Algoritmul determină suma maximă posibilă a unei subsecvențe de elemente neadiacente dintr-un vector.

156. Se consideră matricea pătratică `M` de dimensiune n care conține numere naturale, unde n este un număr natural nenul ($1 \leq n \leq 10^4$) și valorile din matrice sunt astfel încât $1 \leq M[i][j] \leq 10^4$, pentru $i = 1, 2, \ldots, n$ și $j = 1, 2, \ldots, n$. Se consideră următorul algoritm `f(M, i, j, n)`: ✓ ?

```
Algorithm F(M, i, j, n)
   minVal ← -10^9
   If i = n  AND j = n then
      Return M[i][j]
   EndIf
   If i > n  OR j > n then
      Return minVal
   EndIf
   k ← F(M, i, j + 1, n)
   l ← F(M, i + 1, j, n)
   Return M[i][j]+ G(k, l)
EndAlgorithm
```

```
Algorithm G(a, b)
   If a > b then
      Return a
   Else
      Return b
   EndIf
EndAlgorithm
```

Precizați care dintre afirmațiile de mai jos sunt adevărate pentru algoritmul prezentat:

A. Funcția `f` calculează suma maximă a unui traseu de la stânga-sus la dreapta-jos într-o matrice, permițând deplasarea doar spre dreapta sau în jos.

B. Funcția `f` calculează suma minimă a unui traseu de la stânga-sus la dreapta-jos într-o matrice, permițând deplasarea doar spre dreapta sau în jos.

C. Pentru apelul `f(M, 1, 1, 3)` și matricea $M = \begin{bmatrix} 5 & 3 & 2 \\ 1 & 4 & 6 \\ 0 & 7 & 8 \end{bmatrix}$ algoritmul va returna valoarea 27.

D. Funcția `f` returnează întotdeauna valoarea cea mai mare din matrice, indiferent de poziție.

157. Se consideră algoritmul `M(n, i, j)`, unde n, i și j sunt numere naturale ($1 \leq n, i, j \leq 10^4$): ✓ ?

```
Algorithm M(n, i, j)
   If i > j then
      Write '@'
   Else
      If (i * j) MOD n = 0 then
         M(n, i + 3, j - 2)
         Write 'P'
```

```
        Else
           If (i + j) MOD n = 1 then
              M(n, i, j − 1)
              Write 'Q'
           Else
              M(n, i + 1, j)
              Write 'R'
           EndIf
        EndIf
     EndIf
  EndAlgorithm
```

Ce afișează execuția apelului `M(15, 2, 10)`?

A. @QRPRR B. @QRRPR C. @QRPPR D. @QRRPP

158. Se consideră algoritmul `F(n)`, unde n este un număr natural ($1 \leq n \leq 10^6$): ✓ ?

```
Algorithm F(n)
   If n < 3 then
      Return n ∗ 4
   EndIf
   u ← n MOD 6
   p ← F(n DIV 2)
   If (u + p) MOD 5 = 0 then
      Return (u ∗ p + n) MOD 12
   EndIf
   If n MOD 4 = 0 then
      Return (u + p ∗ n) MOD 8
   EndIf
   Return (n − p + u) MOD 10
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Apelul `f(7325)` returnează valoarea 8.

B. Apelul `f(5147)` returnează valoarea 3.

C. Apelul `f(8363)` returnează valoarea 9.

D. Apelul `f(4568)` returnează valoarea 5.

159. Se consideră algoritmul `f(v, a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10^4$), iar v este un vector cu b elemente numere întregi ($v[1], v[2], \ldots, v[b-1]$): ✓ ?

```
Algorithm F(v, a, b)
   If a > b then
      Return 0
   EndIf
   If a = b  AND  a ≠ 0 then
      Return v[a] ∗ 2
   EndIf
   m ← (a + b) DIV 2
   s1 ← F(v, a, m)
   s2 ← F(v, m + 1, b)
```

```
    If (b − a) MOD 2 = 0 then
       Return s1 + s2 + m
    Else
       Return (s1 − s2) MOD 7
    EndIf
EndAlgorithm
```

Care este rezultatul apelului `f([3, 4, 3, 7, 5, 6, 7], 1, 7)`?

A. 12
B. 33
C. 20
D. 15

160. Se consideră matricea pătratică `X` de dimensiune n care conține numere naturale, ✓ ? unde n este un număr natural nenul ($1 \leq n \leq 10^4$), iar valorile sunt astfel încât $1 \leq X[i][j] \leq 10^4$, pentru $i = 1, 2, \ldots, n$ și $j = 1, 2, \ldots, n$. Se consideră următorul algoritm:

```
Algorithm GENERATE(n, x, i, j)
   If i > n then
      For p ← 1, n execute
         For q ← 1, n execute
            Write x[p][q]
         EndFor
         Write newline
      EndFor
      Return
   EndIf
   If i = 1  AND j = 1 then
      x[i][j] ← 1
   Else If j > 1 then
      x[i][j] ← x[i][j − 1] ∗ 2
   Else
      x[i][j] ← x[i − 1][n] ∗ 2
   EndIf
   If j < n then
      GENERATE(n, x, i, j + 1)
   Else
      GENERATE(n, x, i + 1, 1)
   EndIf
EndAlgorithm
```

În urma apelului `generate(3, x, 1, 1)`, ce va afișa algoritmul dat?

A. $\begin{bmatrix} 1 & 2 & 4 \\ 8 & 16 & 32 \\ 64 & 128 & 256 \end{bmatrix}$

B. $\begin{bmatrix} 1 & 2 & 4 \\ 32 & 16 & 8 \\ 64 & 128 & 256 \end{bmatrix}$

C. $\begin{bmatrix} 1 & 2 & 4 \\ 128 & 256 & 8 \\ 64 & 32 & 16 \end{bmatrix}$

D. $\begin{bmatrix} 1 & 2 & 4 \\ 8 & 16 & 32 \\ 256 & 128 & 64 \end{bmatrix}$

161. Se consideră algoritmul `f(v, n)`, unde n este un număr natural ($1 \leq n \leq 10^4$), iar ✓ ? v este un vector cu n elemente naturale ($v[1], v[2], \ldots, v[n]$).

```
Algorithm F(v, n)
   If n = 0 then
      Return 0
   EndIf
   If n MOD 2 = 0 then
      Return v[n]+ F(v, n − 1)
   Else
      Return −v[n]+ F(v, n − 1)
   EndIf
```

```
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul calculează diferența dintre suma elementelor de pe pozițiile pare și suma elementelor de pe pozițiile impare dintr-un vector `v`.

B. Pentru apelul `f([72, 34, 120, 56, 44], 5)` algoritmul va returna valoarea `-146`.

C. Algoritmul calculează suma tuturor elementelor dintr-un vector, modificând semnul fiecărui element pe baza valorii acestuia.

D. Pentru apelul `f([89, 50, 33, 70, 20, 63], 6)` algoritmul va returna valoarea `40`.

162. Se consideră algoritmul `Algo(n)`, unde n este un număr natural ($0 \leq n \leq 10^6$). ✓ ?

```
Algorithm ALGO(n)
   If n = 0 then
      Return 0
   Else
      If (n MOD 10) MOD 2 ≠ 0 then
         Return ALGO(n DIV 10)
+n MOD 10
      Else
         Return ALGO(n DIV 10)
      EndIf
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Apelul `algo(12345)` returnează `9`.

B. Algoritmul calculează suma cifrelor aflate pe poziții impare ale numărului dat.

C. Algoritmul calculează suma cifrelor impare ale numărului dat.

D. Algoritmul calculează suma tuturor cifrelor numărului dat.

163. Se consideră algoritmul `ceFace(n)` unde n este un număr natural ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm CEFACE(n)
   If n > 0 then
      Write n − 1
      CEFACE(n DIV 2)
      Write n + 1
      CEFACE(n − 1)
      Write n
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate referitor la algoritmul `ceFace(n)`:

A. Pentru $n = 5$, se afișează 39 de valori.

B. Pentru $n = 6$, se afișează 54 de valori.

C. Oricare ar fi valoarea lui n, algoritmul `ceFace(n)` afișează întotdeauna un număr multiplu de 3 de valori.

D. Pentru $n = 2$, se afișează `1 0 2 1 3 0 2 1 2`.

164. Se consideră algoritmul `afisare(n)`. Știind că $n \in \{3003, 6006, 9009\}$, care dintre următoarele variante de răspuns sunt adevărate? ✓ ?

```
Algorithm AFISARE(n)
   If n ≥ 1000 then
      write(n MOD 10)
      AFISARE(n DIV 3)
   Else
      If n ≥ 100 then
         write(n * 2)
         If n < 2000 then
            If n MOD 2 = 0 then
               AFISARE(n DIV 3)
            Else
               AFISARE(n * 2)
            EndIf
         EndIf
         write(n)
      EndIf
   EndIf
EndAlgorithm
```

A. Suma primelor două valori afișate este un multiplu de 4, $\forall n \in \{3003, 6006, 9009\}$.

B. Cel mai mare număr afișat este 1332.

C. Valorile afișate pentru 3333 sunt aceleași valori afișate pentru 9999.

D. Pentru fiecare valoare posibilă a lui n, ultima cifră a acesteia nu afectează ultimul număr afișat.

165. Se consideră algoritmul `g(x)`, unde x este număr întreg. ✓ ?

```
Algorithm G(x)
   If x = 0 then
      Return 1
   EndIf
   If x < 10 then
      Return x
   EndIf
   If x MOD 5 = 0 then
      Return G(x DIV 100) * 2
   Else
      Return G(x DIV 100) + G(x MOD 10)
   EndIf
EndAlgorithm
```

Pentru ce valoare a lui x algoritmul va returna valoarea 8?

A. 12505. B. 10205. C. 23015. D. 30515.

166. Se consideră algoritmul `f(n, i, j)`, unde n, i și j sunt numere naturale ($1 \leq n, i, j \leq 10^4$) la momentul apelului inițial. ✓ ?

```
Algorithm F(n, i, j)
   If i ≥ j then
      Write '@'
   Else
      If (n MOD (i + j)) MOD 3 = 0 then
         F(n, i + 1, j - 2)
         Write 'A'
      Else If n DIV (i + 1) MOD 2 = 0 then
         F(n, i, j - 1)
         Write 'B'
      Else
         F(n, i + 2, j - 1)
```

```
            Write 'C'
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

Ce afișează execuția apelului `f(17, 3, 18)`?

A. @*BBACABABC*

B. @*BCAAABCAB*

C. @*BBACABBBB*

D. @*BBACACBBA*

167. Se consideră algoritmul `f(n)`, unde n este un număr natural ($1 \leq n \leq 10^6$). ✓ ?

```
Algorithm F(n)
   If n = 0 then
      Return 1
   EndIf
   If n = 1 then
      Return 2
   EndIf
   If n MOD 2 = 0 then
      Return F(n - 1) + F(n - 2)
   Else
      Return F(n - 1) * F(n - 3)
   EndIf
EndAlgorithm
```

Ce va returna algoritmul pentru apelul `f(8)`?

A. 165

B. 90

C. 168

D. 215

168. Se consideră algoritmii `X(n, i, j)` și `Y(a, b)`, unde n, i, j sunt numere naturale nenule, cu $1 \leq n \leq 10^6$, $0 \leq i \leq 6$, $0 \leq j \leq 9$, iar a, b sunt numere naturale nenule, cu $1 \leq a, b \leq 10^3$. ✓ ?

```
Algorithm Y(a, b)
   If b = 0 then
      Return 1
   EndIf
   If b MOD 2 = 0 then
      Return Y(a, b DIV 2) * Y(a, b DIV 2)
   Else
      Return Y(a, b DIV 2) * a * Y(a, b DIV 2)
   EndIf
EndAlgorithm

Algorithm X(n, i, j)
   p ← Y(10, i) DIV 10
   Return n + (n MOD 10) * p + (n DIV p) * p + j
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `X(369, 3, 5)`, algoritmul returnează 1573.

B. Pentru apelul `X(369, 2, 3)`, algoritmul returnează 822.

C. Pentru apelul `Y(6, 3)`, algoritmul returnează 216.

D. Algoritmul `Y(a, b)` returnează a^b, complexitatea lui fiind $O(\log_2 b)$.

169. Se consideră algoritmul `F(v, n)`, unde n este un număr natural ($1 \leq n \leq 10^4$), iar ✓ ? v este un vector cu n elemente numere întregi ($v[1], v[2], \ldots, v[n]$).

```
Algorithm F(v, n)
   If n = 0 then
      Return v[0]
   EndIf
   If n MOD 3 = 0 then
      Return F(v, n - 1) + v[n] * (n MOD 2)
   Else If n MOD 2 = 0 then
      Return F(v, n - 2) - v[n]+F(v, n - 1)
   Else
      Return (F(v, n - 1) * v[n]) MOD 7+ F(v, n DIV 2)
   EndIf
EndAlgorithm
```

Ce va returna algoritmul pentru apelul `f([2, 8, 7, 9, 2, 8], 6)`?

A. 7 B. 4 C. 5 D. 10

170. Se consideră algoritmul `meci(a, b)`, unde a și b sunt numere naturale nenule ($1 \leq$ ✓ ? $a, b \leq 10^3$).

```
Algorithm MECI(a, b)
   If a * b = 0 then
      Return 1
   EndIf
   Return meci(a - 1, b) + meci(a, b - 1)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `meci(3, 2)` algoritmul returnează 10.

B. Pentru apelul `meci(5, 5)` algoritmul returnează 250.

C. Algoritmul are complexitatea $O(2^{a+b})$.

D. Algoritmul are complexitatea $O(a * b)$.

171. Se consideră algoritmul `F(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm F(n)
   Write n DIV 3
   If n > 3 then
      Write n - 2
      F(n DIV 2)
      Write n
      n ← n - 2
      F(n + 1)
   Else
      If n > 0 then
         Write n + 1
         F(n DIV 3)
         Write n MOD 2
```

```
      EndIf
   EndIf
EndAlgorithm
```

Pentru care dintre următoarele valori ale lui n, algoritmul `F(n)` afișează valoarea 130300512030041402011?

A. $n = 7$ B. $n = 6$ C. $n = 5$ D. $n = 8$

Problemele **172.**, **173.** se referă la următorii algoritmi `sun(i)` și `moon(n)`, unde i și n sunt numere naturale nenule ($1 \leq i, n \leq 10^4$).

```
Algorithm SUN(i)
   If i > 1 then
      k ← i DIV 2
      Return sun(k)+1
   Else
      Return 0
   EndIf
EndAlgorithm
```

```
Algorithm MOON(n)
   j ← 1
   c ← 0
   While j < n execute
      c ← c + sun(j)
      j ← j * 2
   EndWhile
   Return c
EndAlgorithm
```

172. Precizați care afirmații de mai jos sunt adevărate referitor la algoritmii `sun(i)` și `moon(n)`. ✓ ?

A. Clasa de complexitate de timp a algoritmului `moon(n)` este $O(\log_2 n)$.
B. Clasa de complexitate de timp a algoritmului `moon(n)` este $O(\log_2(\log_2 n))$.
C. Clasa de complexitate de timp a algoritmului `sun(i)` este $O(\log_2 i)$.
D. Clasa de complexitate de timp a algoritmului `moon(n)` este $O(\log_3 n \cdot \log_4 n)$.

173. Precizați care dintre următoarele afirmații sunt adevărate referitor la algoritmii `sun(i)` și `moon(n)`. ✓ ?

A. Pentru apelul `sun(40)` algoritmul returnează valoarea 5.
B. Pentru apelul `moon(128)` algoritmul returnează valoarea 20.
C. Pentru apelurile `sun(7)` și `moon(7)` algoritmii returnează aceeași valoare.
D. Pentru apelul `moon(2048)` algoritmul returnează valoarea 55.

174. Se consideră algoritmul `ceFace(n)`, unde `n` este un număr natural ($1 \leq n \leq 30$). ✓ ?

```
Algorithm CEFACE(n)
   If n ≤ 1 then
      Return 1
   EndIf
   Return ceFace(n-1) +
ceFace(n-2)
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Algoritmul calculează al n-lea termen din șirul lui Fibonacci;
B. Complexitatea este $O(2^n)$;
C. Pentru $n = 5$ rezultatul este 13;
D. Algoritmul poate fi optimizat folosind programare dinamică.

175. Se consideră algoritmul `ceFace(n, i)`, unde n și i sunt numere naturale ($1 \leq n, i \leq 10^4$). ✓ ?

```
Algorithm ceFace(n, i)
   If n > 1 then
      i ← i * 2
      m ← n DIV 2
      ceFace(m, i - 2)
      ceFace(m, i - 1)
      ceFace(m, i + 2)
      ceFace(m, i + 1)
   Else
      For j = 1, n − 1 execute
         Write i
      EndFor
   EndIf
EndAlgorithm
```

Care este complexitatea de timp a algoritmului `ceFace(n, i)`?

A. $O(n^3)$

B. $O(n^2)$

C. $O(2^n)$

D. $O(n \cdot \log_2 n)$

176. Se consideră algoritmul `S(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^5$). ✓ ?

```
Algorithm S(n)
   If n = 0 then
      Return 0
   EndIf
   c ← n MOD 10
   s ← S(n DIV 10)
   If c MOD 3 = 0 then
      Return s + c * c
   Else If c MOD 3 = 1 then
      Return s − c
   Else
      Return s + 2 * c
   EndIf
EndAlgorithm
```

Precizați care dintre afirmațiile următoare sunt adevărate:

A. Pentru orice număr format doar din cifre multiplu de 3, rezultatul va fi suma pătratelor cifrelor

B. Algoritmul `S(n)` calculează suma cifrelor lui n, adăugând pătratul cifrelor divizibile cu 3.

C. Pentru apelul `S(369)`, valoarea returnată de algoritm este 126.

D. Algoritmul `S(n)` returnează 0 pentru orice valoare a lui n care nu conține cifre divizibile cu 3.

177. Se consideră algoritmul `ceFace(n, b)`, unde n și b sunt numere naturale ($1 \leq n, b \leq 10^5$) ✓ ?

```
Algorithm ceFace(n, b)
   If n = 0 then
      Return 0
   EndIf
   c ← n MOD b
   x ← ceFace(n DIV b, b)
   If c MOD 2 = 0 then
      Return x * 10 + c
   Else
      y ← 1
      temp ← x
      While temp > 0 execute
         y ← y * 10
         temp ← temp DIV 10
```

```
      EndWhile
      Return c * y + x
   EndIf
EndAlgorithm
```

Precizați care dintre afirmațiile următoare sunt adevărate:

A. Pentru apelul `ceFace(234, 5)`, valoarea returnată de algoritm este 543.

B. Algoritmul `ceFace(n, b)` rearanjează cifrele lui n în baza b, astfel încât cifrele pare apar la final, iar cifrele impare apar la început.

C. Pentru apelul `ceFace(5749, 5)`, valoarea returnată de algoritm este 140444.

D. Pentru apelul `ceFace(4423, 8)`, valoarea returnată de algoritm este 75100.

178. Se consideră algoritmul `Exp(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10^5$). ✓ ?

```
Algorithm Exp(a, b)
   If b = 0 then
      Return a
   EndIf
   If a MOD 2 = 0 and b MOD 2 = 0 then
      Return 2 * Exp(a DIV 2, b DIV 2)
   Else If a MOD 2 = 0 then
      Return Exp(a DIV 2, b)
   Else If b MOD 2 = 0 then
      Return Exp(a, b DIV 2)
   Else
      If a > b then
         Return Exp((a - b)  DIV  2, b)
      Else
         Return Exp(a, (b - a)  DIV  2)
      EndIf
   EndIf
EndAlgorithm
```

Precizați care dintre afirmațiile următoare sunt adevărate:

A. Algoritmul `Exp(a, b)` returnează cel mai mare divizor comun al numerelor a și b folosind metoda binară.

B. Pentru apelul `Exp(48, 36)`, valoarea returnată este 12.

C. Algoritmul returnează același rezultat ca algoritmul lui Euclid clasic pentru orice a și b

D. Pentru apelul `Exp(28, 7)`, valoarea returnată este 4.

179. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `n` este un număr natural nenul ($1 \leq n \leq 10^6$). ✓ ?

```
Algorithm ceFace(v, n)
   If n ≤ 0 then
      Return 0
   EndIf
   If v[n] MOD 2 = 0 then
      Return 1 + ceFace(v, n-1)
```

```
   EndIf
   Return ceFace(v, n-1)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul numără elementele pare din vector;

B. Algoritmul are complexitatea $O(2 * n)$;

C. Pentru un vector vid, algoritmul returnează -1;

D. Algoritmul se poate rescrie iterativ fără a modifica complexitatea timp.

180. Se consideră algoritmul `ceFace(a, n, i = 0, j = 0)`, unde `a` este o matrice pătratică ? de dimensiune `n x n` cu elemente întregi ($-10^9 \leq a[i][j] \leq 10^9$), iar `n` este un număr natural nenul ($1 \leq n \leq 100$).

```
Algorithm ceFace(a, n, i = 0, j = 0)
   If i ≥ n or j ≥ n then
      Return 0
   EndIf
   If i = n − 1 and j = n − 1 then
      Return a[i][j]
   EndIf
   Return a[i][j] + max( ceFace(a, n, i+1, j), ceFace(a, n, i, j+1))
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Calculează suma maximă pe un drum de la $(0, 0)$ la $(n - 1, n - 1)$;

B. Are complexitate $O(2^n)$;

C. Se poate optimiza folosind programare dinamică;

D. Algoritmul face suma elementelor din matrice.

181. Se consideră algoritmul `ceFace(n, d = 2)`, unde `n` este un număr natural ($1 \leq n \leq$ ✓ ? 10^9) și `d` reprezintă un divizor prim folosit în calcul.

```
Algorithm ceFace(n, d = 2)
   If n < 2 then
      Return 0
   EndIf
   If n MOD d = 0 then
      Return 1 + ceFace(n / d, d)
   EndIf
   If d · d > n then
      Return 0
   EndIf
   Return ceFace(n, d + 1)
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Calculează numărul de factori primi ai lui `n`;

B. Are complexitate $O(\sqrt{n})$;

C. Pentru numere prime, returnează 1;

D. Functia calculeaza divizorii proprii a lui n.

182. Se consideră algoritmul `ceFace(v, st, dr, x)`, unde `v` este un vector de `n` numere ✓ ? întregi sortate crescător ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), `st` și `dr` reprezintă limitele intervalului de căutare, iar `x` este numărul căutat.

```
Algorithm ceFace(v, st, dr, x)
   If st > dr then
      Return false
   EndIf
   mid ← [(st + dr)/2]
   If v[mid] = x then
      Return true
   EndIf
   If v[mid] > x then
      Return ceFace(v, st, mid - 1, x)
   EndIf
   Return ceFace(v, mid + 1, dr, x)
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Este o implementare recursivă a căutării binare;

B. Complexitatea este $O(\log n)$;

C. Funcționează doar pe vectori sortați strict crescător;

D. Returnează indexul elementului găsit.

183. Se consideră algoritmii `combine(a,b)` și `ceFace(a, b, c)`, unde a, b și c sunt ✓ ? numere întregi:

```
Algorithm combine(a, b)
   If a > b then
      d ← a − b
   Else
      d ← b − a
   EndIf
   Return (a + b − d)/2
EndAlgorithm

Algorithm ceFace(a, b, c)
   t ← combine(a, b)
   Return combine(t, c)
EndAlgorithm
```

Care dintre următoarele afirmații nu sunt adevărate?

A. Pentru orice x, y, expresia `combine(x,y)` este echivalentă cu `combine(y,x)`.

B. Pentru valorile $a = 8$, $b = 5$, $c = 6$, funcția `ceFace` va returna valoarea 5.

C. Există valori pentru a, b și c astfel încât `ceFace(a,b,c)` să returneze un rezultat diferit de `ceFace(b,a,c)`.

D. Dacă $|a| = |b|$, atunci rezultatul apelului `ceFace(a,b,c)` va fi întotdeauna minimul dintre a și c.

184. Se consideră algoritmul `Algo(n)`, unde `n` reprezintă un număr natural ($1 \leq n \leq 10^5$): ✓ ?

```
Algorithm ALGO(n)
   Write n MOD 100
   If n ≥ 10 then
      ALGO(n DIV 10)
   EndIf
   Write n MOD 10
EndAlgorithm
```

Ce valoare afișează algoritmul în urma apelului `Algo(2024)`?

A. 24022024

B. 2422022024

C. 2024202424

D. 242222024

185. Se consideră algoritmul `f(n)`, unde `n` reprezintă un număr natural ($n > 0$): ✓ ?

```
Algorithm F(n)
   Write n − 2
   If n > 2 then
      F(n − 2)
   EndIf
   Write n + 2
EndAlgorithm
```

Ce valoare afișează algoritmul în urma apelului `f(10)`?

A. 8 8 6 2 0 4 6 8 10 12

B. 8 6 4 2 0 4 6 8 10 12

C. 12 10 8 6 4 0 6 8 10 12

D. 12 10 8 6 4 0 2 4 6 8

186. Se consideră algoritmul `spirala(n, sens)`, unde n este un număr natural ($n \geq 10$). ✓ ?
Algoritmul utilizează operatorul `<<`, definit ca o operație de deplasare pe biți spre stânga (*bitwise left shift*). Aceasta deplasează toți biții unui număr spre stânga cu un anumit număr de poziții, echivalent cu înmulțirea numărului inițial cu 2 ridicat la puterea specificată.

```
Algorithm SPIRALA(n, sens)
   If n < 10 then
      Return n
   EndIf
   u ← n MOD 10
   p ← n
   While p ≥ 10 execute
      p ← p DIV 10
   EndWhile
   temp ← n DIV 10
   mij ← temp MOD (1 <<
(CIFRE(temp) − 1))
   If sens = 1 then
      rez ← u ∗ 10 + p
   Else
      rez ← p ∗ 10 + u
   EndIf
   If mij = 0 then
      Return rez
   EndIf
   Return rez+ SPIRALA(mij,
1−sens)∗100
EndAlgorithm
```

```
Algorithm CIFRE(x)
   If x = 0 then
      Return 0
   EndIf
   Return 1+ CIFRE(x DIV 10)
EndAlgorithm
```

Pentru apelul `Spirala(67812, 1)`, ce valoare va returna algoritmul?

A. 427 B. 527 C. 526 D. 326

187. Se consideră algoritmul `F(n)`, unde `n` reprezintă un număr natural ($1 \leq n \leq 10^5$): ✓ ?

```
Algorithm F(n)
   Write n * 2 and "!"
   i ← 1
   While i ≤ n − 1 execute
      F(i)
      i ← i + 1
   EndWhile
   Write "*"
EndAlgorithm
```

Ce valoare afișează algoritmul în urma apelului `F(3)`?

A. 6!2! ∗ 4!2! ∗ ∗

B. 6!2!4!2! ∗ ∗∗

C. 6!4! ∗ 2!4! ∗ ∗

D. 6!2! ∗ 4!2! ∗ ∗∗

Metodele Backtracking, Divide et Impera și Greedy

Acest capitol acoperă

- Explorarea tehnicilor de backtracking pentru construirea soluțiilor
- Ce este metoda divide et impera și de ce este utilă?
- Analiza complexității algoritmilor divide et impera
- Ce este metoda Greedy? Probleme clasice rezolvate cu această metodă

8.1 Backtracking

8.1.1 Teorie

Algoritmii de backtracking explorează toate posibilitățile pentru a construi o soluție validă, revenind la pașii anteriori („backtracking") atunci când o anumită cale nu duce la o soluție validă.

Elementele Backtracking-ului

- **Spațiul de căutare:**
  - Reprezintă toate posibilele configurații ale soluției.
  - Este organizat sub forma unei structuri de tip arbore, unde fiecare nivel corespunde unei decizii sau alegeri.
- **Candidatul pentru soluție:**
  - Reprezintă o opțiune posibilă selectată într-un anumit pas al algoritmului.
  - Este definit ca un element dintr-o mulțime de opțiuni valide.
- **Verificarea validității:**
  - La fiecare pas al algoritmului, se verifică dacă alegerea curentă respectă toate constrângerile problemei.
  - În cazul în care alegerea nu este validă, algoritmul revine la pasul anterior pentru a explora o altă opțiune posibilă.
- **Cazul de succes:**
  - Reprezintă o configurație completă care satisface toate constrângerile problemei.
  - Poate exista fie o soluție unică, fie mai multe soluții, în funcție de cerințe.

Backtracking-ul poate fi descris ca o metodă recursivă care explorează toate posibilitățile într-un mod structurat. Pentru o problemă dată, această metodă construiește soluția pas

cu pas și revine la pașii anteriori atunci când o alegere nu poate conduce la o soluție validă.

$$\text{Backtrack}(k) = \begin{cases} \text{Dacă } k > n, \text{ salvează soluția} \\ \text{Dacă valid}(c, k) \Rightarrow x[k] = c \Rightarrow \text{Backtrack}(k+1) \end{cases}$$

Această ecuație de recurență descrie procesul de construire a soluției reprezentat de vectorul x. Soluția este obținută prin selectarea unor candidați c din mulțimile A_k, verificare validității fiecărei alegeri și continuarea recursivă cu pasul următor.

Structura Generală a unui Algoritm de Backtracking

```
1: Algorithm BACKTRACK(k)
2:     If k > n then
3:         Solutie ← x[1..n]
4:         Afiseaza(Solutie)
5:     Else
6:         For fiecare candidat c în A_k execute
7:             If VALID(c, k) then
8:                 x[k] ← c
9:                 BACKTRACK(k + 1)
10:            EndIf
11:        EndFor
12:    EndIf
13: EndAlgorithm
```

8.1.2 Probleme

188. Luca dorește să își cumpere mâncare de la magazin după ce a terminat ora de matematică, unde a învățat despre combinații. În drum spre magazin, Luca este curios să afle cum poate implementa un algoritm care să îl ajute să determine numărul combinațiilor posibile ale monedelor pe care le are în buzunar, pentru a atinge suma dorită. ✓ ?

Se consideră vectorul `c` cu `n` elemente $(c[1], c[2], \ldots, c[n])$, unde $1 \leq n \leq 10^4$, iar `s` este un număr întreg care reprezintă suma pe care acesta dorește să o atingă. Analizați următoarele variante și precizați care sunt corecte pentru a-l ajuta pe Luca să determine combinațiile posibile de monede pentru suma dorită.

A.

```
Algorithm COUNT(c, n, s)
   If s = 0 then
      Return 1
   EndIf
   If n = 0  OR s < 0 then
      Return 0
   EndIf
   include ← COUNT(c, n, s−c[n−
2])
   exclude ← COUNT(c, n − 1, s)
   Return include + exclude
EndAlgorithm
```

B.

```
Algorithm COUNT(c, n, s)
   If s = 0 then
      Return 1
   EndIf
   If n = 0 then
      Return 0
   EndIf
   include ← COUNT(c, n, s−c[n−
1])
   exclude ← COUNT(c, n − 1, s)
   Return include + exclude
EndAlgorithm
```

C.

```
Algorithm COUNT(c, n, s)
   If s = 0 then
      Return 1
   EndIf
   If n = 0  OR s < 0 then
      Return 0
   EndIf
   include ← COUNT(c, n − 1, s −
c[n − 1])
   exclude ← COUNT(c, n − 1, s)
   Return include + exclude
EndAlgorithm
```

D.

```
Algorithm COUNT(c, n, s)
   If s = 0 then
      Return 1
   EndIf
   If n = 0  OR s < 0 then
      Return 0
   EndIf
   include ← COUNT(c, n, s−c[n−
1])
   exclude ← COUNT(c, n − 1, s)
   Return include + exclude
EndAlgorithm
```

189. Se consideră algoritmul `g(v, n, t)`, unde n este un număr natural ($1 \leq n \leq 10^4$), v este un vector cu n elemente naturale ($v[1], v[2], \ldots, v[n]$), iar t este un număr natural.

```
Algorithm G(v, n, t)
   If t = 0 then
      Return True
   EndIf
   If n = 0 then
      Return False
   EndIf
   include ← G(v, n − 1, t − v[n − 1])
   exclude ← G(v, n − 1, t)
   Return include  OR exclude
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `g([5, 10, 15, 30, 4, 9], 6, 10)` algoritmul va returna valoarea `False`.

B. Algoritmul determină dacă există o submulțime dintr-un vector care produce exact o anumită sumă specificată.

C. Pentru apelul `g([3, 34, 4, 12, 5, 2], 6, 9)` algoritmul va returna valoarea `True`.

D. Algoritmul returnează valoarea `True` numai și numai dacă suma tuturor elementelor vectorului atinge suma specificată.

190. Se consideră algoritmul `xenon(x, n, p)`, unde n și p sunt numere naturale nenule ($1 \leq n, p \leq 10$) și x este un vector cu *n + 1* elemente naturale ($x[0], x[1], \ldots, x[n]$). Presupunem că *x[0]* este inițializat cu 0. Algoritmul `neon(x, k)` este definit alăturat și are ca parametri un vector x și un număr natural k ($1 \leq k \leq 10$):

```
Algorithm XENON(x, n, p)
   For i ← x[p − 1] + 1, n execute
      x[p] ← i
      NEON(x, p)
      If p < n then
         XENON(x, n, p + 1)
      EndIf
   EndFor
EndAlgorithm
```

```
Algorithm NEON(x, k)
   For i ← 1, k execute
      Write x[i], ' '
   EndFor
   Write NewLine
EndAlgorithm
```

Precizați care dintre afirmațiile de mai jos sunt adevărate:

A. Dacă `x[0]` se inițializează cu valoarea 2, algoritmul `xenon(x, n, 1)` apelează algoritmul `neon(x, k)` de $2^{n-2} - 1$ ori.

B. Pentru apelul `xenon(x, 5, 1)`, algoritmul afișează pe a doisprezecea linie `1 3 4 5`.

C. Pentru apelul `xenon(x, 4, 1)`, algoritmul afișează pe a șasea linie `1 3 4`.

D. Pentru apelul `xenon(x, 3, 1)`, algoritmul se autoapelează de 7 ori.

191. Se consideră algoritmul `ceFace(arr, k, n)`, unde `arr` este un vector de n numere naturale ($1 \leq n \leq 10^2$), iar k este un număr natural ($1 \leq k \leq n$). Se presupune că toate elementele vectorului `arr` sunt egale cu pozițiile lor la apelul inițial. ✓ ?

```
Algorithm ceFace(arr, k, n)
   For i ← 1, n execute
      If k ≠ i then
         arr[k] ← i
         p ← 1
         For j ← k − 1, 1, −1 execute
            If arr[j] = arr[k] then
               p ← 0
            EndIf
         EndFor
         If p then
            If k = n then
               For m ← 1, n execute
                  Write arr[m], ' '
               EndFor
               Write '*'
               Write newline
            Else
               ceFace(arr, k + 1, n)
            EndIf
         EndIf
      EndIf
   EndFor
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate referitor la apelul algoritmului `ceFace(arr, 1, n)`.

A. Algoritmul afișează toate permutările vectorului `arr`.

B. Pentru apelul `ceFace(arr, 1, 4)` algoritmul afișează 9 steluțe.

C. Pentru apelul `ceFace(arr, 1, 5)` pe a opta linie se va afișa 2 4 5 3 1∗.

D. Pentru apelul `ceFace(arr, 1, 5)` pe a zecea linie se va afișa 2 5 4 3 1∗.

192. Luca vrea să determine toate submulțimile mulțimii $M = \{4, 5, 17, 24, 11, 16\}$ cu 6 elemente, unde suma elementelor din submulțime nu depășește un prag minim și un prag maxim. Pentru acest lucru el a creat un algoritm. Mulțimea este reprezentată cu ajutorul vectorului M cu n elemente numere naturale. Submulțimile generate se afișează cu ajutorul funcției `afișare`, unde v este un vector auxiliar indexat de la 0, iar l reprezintă lungimea vectorului curent v. Înainte de apelarea funcției `algo(1, 6, v, M, 0, 20, 40)` elementul $v[0]$ a fost inițializat cu 0. ✓ ?

```
Algorithm ALGO(i, n, v, M, suma,
s_min, s_max)
   For j ← v[i − 1] + 1, n execute
      v[i] ← j
      suma ← suma + M[j]
      If s_min ≤ suma ≤ s_max then
         AFIȘARE(i, v, M)
      EndIf
      If suma < s_max then
         ALGO(i + 1, n, v, M,
suma, s_min, s_max)
      EndIf
      suma ← suma − M[j]
   EndFor
EndAlgorithm
```

```
Algorithm AFIȘARE(d, v, M)
   Write "{", M[v[1]]
   For i ← 2, d execute
      Write ", ", M[v[i]]
   EndFor
   Write "}", newline
EndAlgorithm
```

Luca știe că primele 4 submulțimi afișate sunt, în această ordine: $\{4, 5, 17\}$, $\{4, 5, 17, 11\}$, $\{4, 5, 24\}$, $\{4, 5, 11\}$, dar vrea să știe care este a 8-a submulțime generată. Care dintre cele de mai jos reprezintă răspunsul corect?

A. $\{4, 11, 24\}$ B. $\{5, 24, 11\}$ C. $\{5, 17\}$ D. $\{4, 17, 11\}$

193. Se consideră algoritmul `four(c, l, n)`, unde c, l și n sunt numere naturale nenule ✓ ? $(1 \leq l, n \leq 9, 1 \leq c \leq 10^9)$.

```
Algorithm FOUR(c, l, n)
   If l = n then
      Write c, ' '
   Else
      For i ← 0, 9 execute
         cn ← 0
         For j ← 1, i execute
            If i MOD j = 0 then
               cn ← cn + 1
            EndIf
         EndFor
         If cn = 2 then
            FOUR(c * 10 + i, l + 1, n)
         EndIf
      EndFor
   EndIf
EndAlgorithm
```

Precizați care afirmații sunt adevărate referitoare la apelul algoritmului `four(0, 0, n)`.

A. Algoritmul afișează 4^n valori întotdeauna.

B. Algoritmul afișează numere formate din n cifre din mulțimea $\{2, 3, 5, 7, 9\}$.

C. Algoritmul afișează toate numerele prime formate din n cifre.

D. Algoritmul afișează toate numerele formate din n cifre, iar fiecare cifră are exact 2 divizori.

8.2 Divide et impera

8.2.1 Teorie

Metoda divide et impera reprezintă o abordare care împarte o problemă complexă în subprobleme mai mici și mai simple, le rezolvă, iar apoi combină soluțiile acestora pentru a obține rezultatul final. De menționat că, în majoritatea cazurilor, problemele rezolvate prin această metodă pot fi abordate si cu ajutorul unei bucle simple `for`, cu anumite excepții specifice.

Elementele Fundamentale ale Metodei Divide et Impera

- **Divide (Împarte):** Se împarte problema inițială în una sau mai multe subprobleme de dimensiuni mai mici, de același tip ca problema inițială.
- **Conquer (Rezolvă):** Se rezolvă subproblemele obținute. Dacă subproblemele sunt suficient de simple, se rezolvă direct.
- **Combine (Combină):** Se combină soluțiile subproblemelor pentru a forma soluția problemei inițiale.

Implementare

De acum încolo vom folosi sintaxa {`arr[a:b]`}. Conceptul se numește *slicing* și este întâlnit în aproape toate limbajele moderne, cum ar fi Python.

Cum funcționează?
Presupunem că avem un vector `arr` cu numere de la 0 la n. Expresia `arr[a:b]` ne va returna sub-vectorul care începe de la poziția a și se oprește la poziția $b - 1$.

Exemplu:
Dacă `arr = {0, 1, 2, 3, 4, 5}`, atunci:

- `arr[1:4]` va returna {`1, 2, 3`}.

Un exemplu clasic de algoritm divide et impera este algoritmul Merge Sort. Acesta sortează un vector împărțindu-l în subsecțiuni mai mici, sortându-le și apoi combinându-le.
Funcția de combinare `interclasare` este definită la capitolul **Algoritmi elementari**.

Analiza Complexității

Algoritmii divide et impera sunt eficienți datorită modului prin care reduc dimensiunea problemei. În cazul algoritmului Merge Sort, complexitatea temporală este $O(n \log n)$.

Alte Exemple cu Metoda Divide et Impera

Un alt exemplu de utilizare a metodei divide et impera este găsirea elementului maxim dintr-un vector:

Algoritmul Merge Sort

```
1: Algorithm MERGE_SORT(A)
2:     If len(A) ≤ 1 then
3:         Return A
4:     EndIf
5:     m ← len(A) DIV 2
6:     For i ← 1, m execute
7:         L[i] ← A[i]
8:     EndFor
9:     For i ← 1, len(A) − m execute
10:        R[i] ← A[m + i]
11:    EndFor
12:    L ← MERGE_SORT(L)
13:    R ← MERGE_SORT(R)
14:    Return INTERCLASARE(L, R)
15: EndAlgorithm
```

Găsirea Elementului Maxim

```
1: Algorithm MAX_ELEMENT(A)
2:     If len(A) = 1 then
3:         Return A[1]
4:     EndIf
5:     m ← len(A) DIV 2
6:     For i ← 1, m execute
7:         L[i] ← A[i]
8:     EndFor
9:     For i ← 1, len(A) − m execute
10:        R[i] ← A[m + i]
11:    EndFor
12:    left_max ← MAX_ELEMENT(L)
13:    right_max ← MAX_ELEMENT(R)
14:    Return MAX(left_max, right_max)
15: EndAlgorithm
```

Concluzie

Metoda divide et impera este un concept esențial care poate fi aplicat într-o gamă largă de probleme. Aceasta poate fi utilizată atât în situații evidente, precum cele prezentate mai sus, cât și în contexte mai subtile, unde structura metodei nu este imediat evidentă.

8.2.2 Probleme

194. Se consideră algoritmul `f(e)`, unde e este un număr natural nenul ($1 \leq e \leq 10^6$) și algoritmul `g(arr, a, b)`, unde a și b sunt numere naturale nenule ($1 \leq a, b \leq 10^3$) și arr este un vector de n numere naturale nenule ($1 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^6$). Operatorul & este operatorul AND pe biți; tabelul de adevăr este următorul: ✓ ?

Exemplu:

&	0	1
0	0	0
1	0	1

i. 6&1 convertit în binar: 110&001 = 000 = $0_{(10)}$

ii. 2&7 convertit în binar: 010&111 = 010 = $2_{(10)}$

```
Algorithm F(e)
    k ← 0
    While e > 0 execute
        k ← k + (e & 1)
        e ← e DIV 2
    EndWhile
    Return k MOD 2 = 1
EndAlgorithm
```

```
Algorithm G(arr, a, b)
    If a = b then
        Return f(arr[a])
    EndIf
    c ← (a + b) DIV 2
    Return g(arr, a, c) AND g(arr, c+1, b)
EndAlgorithm
```

Pentru ce valori ale numărului n și ale vectorului `arr` apelul `g(arr, 1, n)` returnează True?

A. $n = 5$, `arr` = (14, 7, 11, 1, 4)

B. $n = 4$, `arr` = (31, 7, 15, 3)

C. $n = 5$, `arr` = (16, 19, 21, 22, 25)

D. Pentru oricare $n < 100$ și elementele vectorului sunt de forma 2^k, unde k este un număr natural ($1 \leq k \leq 16$).

195. Se consideră algoritmul `ceFace(ts, td, V)`, unde `ts`, `td` sunt numere naturale nenule ($1 \leq ts, td \leq 10^3$) și `V` este un vector de n numere naturale nenule ($1 \leq V[1], V[2], \ldots, V[n] \leq 10^9$). ✓ ?

```
Algorithm ceFace(ts, td, V)
    If ts = td then
        x ← V[td]
        c ← 0
        While x > 0 execute
            x ← x DIV 10
            c ← c + 1
        EndWhile
        Return c MOD 2 = 0
    EndIf
    Return ceFace(ts, (ts + td) DIV 2, V) AND ceFace((ts + td) DIV 2 + 1, td, V)
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Pentru apelul `ceFace(1, 9, [12, 34, 56, 78, 90, 67, 22, 10, 52])` algoritmul returnează True.

B. Pentru apelul `ceFace(1, 7, [323, 941, 105, 984, 603, 174, 405])` algoritmul returnează True.

C. Pentru apelul `ceFace(2, 6, [384, 29 , 9923, 23, 18, 1784, 44, 85])` algoritmul returnează False.

D. Algoritmul `ceFace(ts, td, V)` verifică dacă toate elementele din intervalul indicilor $[ts, td]$ sunt pare.

196. Se consideră algoritmii `star(m, n)` și `sky(x, y, arr)`, unde m și n sunt numere naturale nenule ($1 \leq m, n \leq 10^6$), iar arr este un vector de n numere naturale nenule ($1 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^6$). ✓ ?

```
Algorithm STAR(m, n)
   If n = 0 then
      Return m
   EndIf
   Return star(n, m MOD n)
EndAlgorithm
```

```
Algorithm SKY(x, y, arr)
   If x = y then
      Return arr[x]
   EndIf
   m ← (x + y) DIV 2
   a ← sky(x, m, arr)
   b ← sky(m + 1, y, arr)
   Return star(a, b)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `sky(1, 5, [12, 72, 78, 42, 90])` algoritmul returnează 6.

B. Pentru apelul `sky(1, 4, [10, 20, 30, 40, 55])` algoritmul returnează 10.

C. Algoritmul `sky(x, y, arr)` are complexitatea O(n).

D. Algoritmul `star(m, n)` returnează cel mai mare divizor comun al numerelor m și n, doar în cazul în care m și n nu sunt prime.

197. Se consideră algoritmul `ceFace(n, arr)`, unde n, s sunt numere naturale nenule ($1 \leq n, s \leq 10^3$) și `arr` este un vector de n numere naturale nenule ($1 \leq$ `arr`$[1],$ `arr`$[2], \ldots,$ `arr`$[n] \leq 10^6$). ✓ ?

```
Algorithm CEFACE(n, arr, s)
   If n = 1 then
      Return arr[s]
   EndIf
   a ← CEFACE(m, arr, s)
   b ← CEFACE(n − n DIV 2, arr, s + n DIV 2)
   Return a + b
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate referitor la algoritmul `ceFace(n, arr, 1)`:

A. Pentru apelul `ceFace(5, [1, 2, 3, 4, 5], 1)`, algoritmul returnează 16.

B. Pentru apelul `ceFace(7, [32, 12, 45, 67, 23, 11, 9], 1)`, algoritmul returnează 199.

C. Pentru orice n și orice vector `arr`, algoritmul returnează suma primelor n elemente din vectorul `arr`.

D. Algoritmul `ceFace(n, arr, s)` are complexitatea de timp $O(\log(n))$.

198. Se consideră algoritmul `ceFace(st, dr, n, m, matrix)`, unde n și m sunt numere naturale nenule ($1 \leq n, m \leq 10^3$), iar `matrix` este o matrice de $n \times m$ numere naturale nenule ($1 \leq$ `matrix`$[i][j] \leq 10^6$). ✓ ?

```
Algorithm ceFace(st, dr, n, m, matrix)
   If st = dr then
      Return matrix[st][2]
   EndIf
   mid ← (st + dr) DIV 2
   a ← ceFace(st, mid, n, m, matrix)
   b ← ceFace(mid + 1, dr, n, m, matrix)
   Return a + b
EndAlgorithm
```

Pentru care dintre următoarele valori ale lui n, m și `matrix`, apelul `ceFace(1, n, n, m, matrix)` returnează 31?

A. $n = 3$, $m = 3$,

$$\begin{bmatrix} 11 & 23 & 3 \\ 4 & 17 & 10 \\ 41 & 8 & 18 \end{bmatrix}$$

B. $n = 2$, $m = 4$,

$$\begin{bmatrix} 1 & 22 & 3 & 4 \\ 5 & 9 & 16 & 1 \end{bmatrix}$$

C. $n = 3$, $m = 2$,

$$\begin{bmatrix} 22 & 13 \\ 37 & 4 \\ 29 & 6 \end{bmatrix}$$

D. $n = 5$, $m = 5$,

$$\begin{bmatrix} 13 & 18 & 23 & 28 & 33 \\ 9 & 5 & 11 & 15 & 17 \\ 7 & 3 & 8 & 9 & 5 \\ 5 & 1 & 3 & 14 & 11 \\ 6 & 4 & 9 & 2 & 4 \end{bmatrix}$$

199. Se consideră algoritmul `once(a, b, arr)`, unde a și b sunt numere naturale nenule ✓ ? $(1 \leq a, b \leq 10^3)$ și `arr` este un vector de n numere naturale $(0 \leq arr[1], arr[2], \ldots, arr[n] \leq 10^9)$.

```
Algorithm once(a, b, arr)
   If a = b then
      x ← arr[a]
      c ← True
      While x > 0 execute
         x ← x DIV 10
         c ← NOT c
      EndWhile
      Return c
   EndIf
   m ← (a + b) DIV 2
   Return once(a, m, arr)  OR once(m + 1, b, arr)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `once(1, 5, [12, 132, 56, 782, 90])` algoritmul returnează `True`.

B. Pentru apelul `once(1, 10, [9, 8, 7, 6, 5, 4, 3, 2, 1, 0])` algoritmul returnează `False`.

C. Pentru apelurile `once(1, 4, [983, 50942, 132, 1])` și `once(1, 5, [91832, 2, 132, 195, 32])` algoritmul returnează aceeași valoare.

D. Pentru apelul `once(3, 7, [123, 32, 24, 9043, 2025, 12, 1302, 134, 543])` algoritmul returnează `False`.

200. Se consideră algoritmul `ceFace`(il, rl, x), unde il, rl sunt numere naturale nenule ✓ ? ($1 \leq il, rl \leq 10^3$) și x este un vector de n numere naturale ($0 \leq x[1], x[2], \ldots, x[n] \leq 10^9$).

```
Algorithm ceFace(il, rl, x)
   If il = rl then
      nr ← x[il]
      op ← 0
      While nr > 0 execute
         If nr MOD 3 = 1 then
            op ← op + nr MOD 10
         EndIf
         nr ← nr DIV 10
      EndWhile
      Return op MOD 2 = 1
   EndIf
   sl ← il * 2 + rl * 2
   sl ← sl DIV 4
   Return ceFace(il, sl, x) AND ceFace(sl + 1, rl, x)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `ceFace(1, 7, [374, 12, 39777, 7848, 97, 765, 30282])`, algoritmul returnează True.

B. Pentru apelul `ceFace(1, 8, [85, 1428, 9961, 1119, 9978, 571, 4818, 691])`, algoritmul returnează True.

C. Pentru apelul `ceFace(1, 8, [1257, 55, 143, 3749, 1369, 7746, 49122, 9846])`, algoritmul returnează False.

D. Algoritmul verifică dacă toate numerele din intervalul indicilor $[il, rl]$ au proprietatea că suma cifrelor a căror rest la împărțirea cu 3 este 1 este un număr impar.

201. Se consideră algoritmul `ceFace(a, n, m, st_i, st_j, dr_i, dr_j)`, unde `a` este ✓ ? o matrice de dimensiune `n x m` cu elemente întregi ($-10^9 \leq a[i][j] \leq 10^9$), iar `st_i, st_j, dr_i, dr_j` sunt indicii care definesc o submatrice.

```
Algorithm ceFace(a, n, m, st_i, st_j, dr_i, dr_j)
   If st_i > dr_i or st_j > dr_j then
      Return 0
   EndIf
   If st_i = dr_i and st_j = dr_j then
      Return a[st_i][st_j]
   EndIf
   mij_i ← (st_i + dr_i) DIV 2
   mij_j ← (st_j + dr_j) DIV 2
   Return max(
            ceFace(a, n, m, st_i, st_j, mij_i, mij_j),
            ceFace(a, n, m, st_i, mij_j+1, mij_i, dr_j),
            ceFace(a, n, m, mij_i+1, st_j, dr_i, mij_j),
            ceFace(a, n, m, mij_i+1, mij_j+1, dr_i, dr_j))
```

```
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Găsește maximul dintr-o matrice împărțind-o în 4 submatrice.

B. Are complexitate $O(nm \log(nm))$.

C. Face exact $\log(nm)$ apeluri recursive.

D. Se poate optimiza folosind un algoritm iterativ în $O(n \cdot m)$.

202. Se consideră algoritmul `ceFace(v, st, dr, k)`, unde `v` este un vector de `n` numere ✓ ? întregi sortate crescător ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `k` este un număr întreg.

```
Algorithm ceFace(v, st, dr, k)
   If st = dr then
      Return v[st] = k
   EndIf
   mij ← (st + dr) DIV 2
   left ← ceFace(v, st, mij, k)
   right ← ceFace(v, mij+1, dr, k)
   count ← 0
   i ← mij, j ← mij + 1
   While i ≥ st and j ≤ dr execute
      If v[i] + v[j] = k then
         If v[i] = v[i + 1] then
            count ← count + 1
         EndIf
         i ← i − 1, j ← j + 1
      Else If v[i] + v[j] < k then
         j ← j + 1
      Else
         i ← i − 1
      EndIf
   EndWhile
   Return left + right + count
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt adevărate?

A. Numără perechile de sumă `k` într-un vector sortat.

B. Are complexitate $O(n^2)$.

C. Pentru `k = 0` și vector vid returnează 0.

D. Funcționează corect doar pe vectori sortați crescător.

203. Se consideră algoritmul `ceFace(v, st, dr)`, unde `v` este un vector de `n` numere ✓ ? întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `st` și `dr` sunt limitele indicilor ($1 \leq st \leq dr \leq n$).

```
Algorithm ceFace(v, st, dr)
   If st = dr then
      Return {v[st], v[st]}
   EndIf
   If dr − st = 1 then
```

```
        Return {min(v[st], v[dr]), max(v[st], v[dr])}
    EndIf
    mij ← (st + dr) DIV 2
    p1 ← ceFace(v, st, mij)
    p2 ← ceFace(v, mij+1, dr)
    Return {min(p1.first, p2.first), max(p1.second, p2.second)}
EndAlgorithm
```

Un `pair` este o structură de date care reține două valori, notate convențional `first` și `second`. În contextul algoritmului, notăm un `pair` ca o pereche ordonată de forma $\{a, b\}$, unde:

i. a reprezintă primul element (`first`)

ii. b reprezintă al doilea element (`second`)

iii. $a, b \in \mathbb{Z}$ și $a \leq b$

Care dintre următoarele afirmații sunt adevărate?

A. Găsește minimul și maximul dintr-un vector.

B. Face mai puține comparații decât metoda clasică(o parcurgere și comparații la fiecare pas).

C. Are complexitate $O(n^2)$.

D. Pentru un vector cu elemente egale returnează aceeași valoare în ambele componente.

204. Se consideră algoritmul `ceFace(v, st, dr)`, unde `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `st` și `dr` sunt limitele indicilor ($1 \leq st \leq dr \leq n$). ✓ ?

```
Algorithm ceFace(v, st, dr)
    If st = dr then
        Return v[st]
    EndIf
    mij ← (st + dr) DIV 2
    s1 ← ceFace(v, st, mij)
    s2 ← ceFace(v, mij + 1, dr)
    Return max(s1, s2)
EndAlgorithm
```

Care dintre următoarele afirmații despre funcție sunt false?

A. Găsește minimul din vector folosind Divide et Impera.

B. Complexitatea este $O(n)$.

C. Complexitatea este $O(nlog(n))$.

D. Pentru un vector cu un singur element returnează acel element.

8.3 Greedy

8.3.1 Teorie

Metoda Greedy este o tehnică fundamentală în proiectarea algoritmilor, care rezolvă probleme printr-o serie de alegeri locale optime, în speranța obținerii unei soluții globale bune. Ideea principală constă în a alege, la fiecare pas, varianta considerată cea mai bună la momentul respectiv, fără a reconsidera ulterior deciziile luate.[2]

Elementele Fundamentale ale Metodei Greedy

- **Alegerea local optimă:**
  - La fiecare pas se selectează soluția care pare cea mai bună în acel moment
  - Nu există revenire sau reconsiderarea alegerilor anterioare
- **Opțiunea Greedy:**
  - Există o structură a problemei care permite ca o alegere local optimă să conducă spre o soluție globală optimă (sau suficient de bună)
  - Nu toate problemele permit aplicarea cu succes a acestei strategii
- **Substructura optimă:**
  - O soluție optimă a problemei poate fi construită din soluțiile optime ale subproblemelor rezultate în urma alegerilor locale
  - Nu este necesară explorarea tuturor soluțiilor posibile

Tipuri de Probleme Rezolvate prin Metoda Greedy

- **Selecția activităților (planificarea intervalelor):**
  - Alegerea unui set maxim de activități care nu se suprapun
  - Exemplu: planificarea spectacolelor într-o sală unică
- **Problema fracționară a rucsacului:**
  - Împachetarea obiectelor pentru maximizarea valorii, fiind permisă fracționarea obiectelor
- **Probleme de acoperire și optimizare pe grafuri:**
  - Algoritmul lui Prim sau Kruskal pentru arborele parțial minim
  - Algoritmul Dijkstra pentru drum minim

Modelul Matematic și Analiza Formală

Metoda Greedy poate fi formalizată prin analiza structurii problemelor ce posedă proprietatea de substructură optimă și opțiunea Greedy. Pentru o problemă de optimizare cu mulțimea de candidați S, la fiecare pas i se alege un element $x_i \in S_i$ (mulțimea de candidați rămași la pasul i) care maximizează sau minimizează o anumită funcție obiectiv f. Problema se reduce astfel la:

$$x_i = \arg\max_{x \in S_i} f(x) \quad \text{sau} \quad x_i = \arg\min_{x \in S_i} f(x)$$

Proprietatea cheie este că prin selectarea acestui x_i, nu afectăm capacitatea de a obține o soluție optimă pentru subproblema rămasă.

Structura Generală a unui Algoritm Greedy

Structura Generală a unui Algoritm Greedy

```
1:  Algorithm AlgoritmGreedy(candidați)
2:      soluție ← ∅
3:      While există candidați disponibili execute
4:          x ← Candidatul cu cea mai bună valoare locală
5:          If sePoateAdauga(x, soluție) then
6:              soluție ← soluție ∪ {x}
7:          EndIf
8:          elimină x din lista de candidați
9:      EndWhile
10:     Return soluție
11: EndAlgorithm
```

Analiza Complexității

Complexitatea temporală a algoritmilor Greedy depinde de:

- **Sortarea inițială** (dacă este necesară), care poate necesita $O(n \log n)$
- **Selecția repetată a celui mai bun candidat**, adesea optimizată prin structuri de date (grămezi, cozi de priorități)
- În general, se urmărește obținerea unei soluții în timp polynomial, pentru probleme altfel dificile

Exemplu Clasic: Selecția Activităților

Pentru a ilustra metoda Greedy, considerăm problema selecției activităților. Avem o mulțime de activități, fiecare cu un interval de timp $[S_i, F_i]$. Scopul este de a selecta un număr maxim de activități care nu se suprapun.

- **Modelul matematic:**

$$\max |A| \quad \text{a. î.} \quad \forall (i,j) \in A, [S_i, F_i] \cap [S_j, F_j] = \emptyset$$

- **Strategia Greedy:**
  - Sortează activitățile crescător după timpul de finalizare
  - Selectează prima activitate (cea cu cel mai mic F_i)
  - Pentru fiecare altă activitate, selecteaz-o doar dacă nu se suprapune cu cele deja alese

Algoritmul Greedy pentru Selecția Activităților

```
1: Algorithm SelectieActivitati(activități)
2:     Sortează activitățile după F_i
3:     selectate ← ∅
4:     ultim ← −∞
5:     For activitate a în ordine execute
6:         If S_a ≥ ultim then
7:             selectate ← selectate ∪ {a}
8:             ultim ← F_a
9:         EndIf
10:    EndFor
11:    Return selectate
12: EndAlgorithm
```

Aplicații și Tipuri de Probleme

Metoda Greedy este utilă în rezolvarea diverselor probleme:

- **Probleme de optimizare combinatorică:**
  - Rucsacul fracționar
  - Selectarea codurilor cu lungime minimă (Coduri Huffman)
- **Probleme pe grafuri:**
  - Afișarea arborelui parțial de cost minim (Prim, Kruskal)
  - Calcularea drumurilor minime (Dijkstra)
- **Probleme de programare orară și alocare de resurse:**
  - Alocarea intervalelor de timp, sarcinilor sau resurselor

8.3.2 Probleme

205. Se dă un șir cu n elemente numere întregi și un număr natural $k \leq n$. Se dorește determinarea celei mai mari sume care poate fi obținută schimbând semnul a exact k elemente aflate pe poziții distincte din șirul dat. ✓ ?

Care dintre următoarele strategii nu rezolvă corect problema?

A. Se sortează elementele șirului crescător după valoarea absolută. Se schimbă semnul primelor k elemente negative din șir. Se calculează suma finală a șirului.

B. Se sortează elementele șirului crescător. Se schimbă semnul primelor k elemente negative din șir. Se calculează suma finală a șirului.

C. Se sortează elementele șirului descrescător după valoarea absolută. Se schimbă semnul primelor k elemente negative, iar dacă mai rămân schimbări disponibile și elemente pozitive, se schimbă și semnul celor mai mici elemente pozitive rămase. Se calculează suma finală a șirului.

D. Se sortează elementele șirului descrescător după valoarea absolută. Se schimbă semnul primelor k elemente negative, iar dacă numărul de elemente negative este mai mic decât k, se schimbă semnul celui mai mic număr dintre elementele pozitive de k ori. Se calculează suma finală a șirului.

206. Se dă un șir de numere a, având lungimea n, $\forall a_i \in \mathbb{N}$. Pentru a obține cel mai mare număr posibil, concatenând aceste cifre, trebuie sortat șirul respectând anumite condiții. Fiind date funcțiile `nrCifre` (care returnează numărul de cifre a unui număr) și `concat` care are implementarea de mai jos, care variantă de răspuns reprezintă soluția optimă? ✓ ?

```
Algorithm CONCAT(x, y)
    p ← nrCifre(y)
    Return x · 10^p + y
EndAlgorithm
```

A.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        x ← nrCifre(a[j])
        y ← nrCifre(a[i])
        If a[i] · 10^x < a[j] · 10^y then
            temp ← a[i]
            a[i] ← a[j]
            a[j] ← temp
        EndIf
    EndFor
EndFor
```

B.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        If concat(a[j], a[i]) > concat(a[i], a[j]) then
            temp ← a[i]
            a[i] ← a[j]
            a[j] ← temp
        EndIf
    EndFor
EndFor
```

C.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        If a[i] < a[j] then
            temp ← a[i]
            a[i] ← a[j]
            a[j] ← temp
        EndIf
    EndFor
EndFor
```

D.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        If a[i] mod 10 < a[j] mod 10 then
            temp ← a[i]
            a[i] ← a[j]
            a[j] ← temp
        EndIf
    EndFor
EndFor
```

207. Pe o masă sunt așezate n bețișoare, pentru fiecare din ele cunoscându-se lungimea acestuia, fiind $L[i]$. Asupra acestora se pot efectua operații de tăiere, în care lungimea bețișorului se scurtează. Se dorește ca lungimile bețișoarelor să fie în ordine descrescătoare, fără a se schimba ordinea acestora. Notăm cu t această lungime. ✓ ?

Pentru care din următoarele seturi de date, lungimea totală minimă pentru a le tăia este corectă?

A. n=10, L=[18 24 14 43 10 9 8 5 7 3], t=37.

B. n=20, L=[23 15 27 31 15 18 19 19 16 20 29 40 28 42 18 27 49 65 2 86], t=83.

C. n=5, L=[8 13 21 34 55], t=90.

D. n=6, L=[438243 2831231 4304392 8123120 2322138103 21321312], t=2321699842.

208. Se dă o matrice 6×6 cu costuri de traversare. Care dintre următoarele variante de răspuns sunt corecte, știind că trebuie să găsim drumul de cost minim de la $(1, 1)$ la $(6, 6)$ (singurele direcții de deplasare valabile fiind sus, jos, stânga, dreapta) ? ✓ ?

A. Pentru matricea dată, costul minim va fi atins prin alegerea minimului dintre celulele din dreapta și jos.

S	7	3	2	4	2
5	5	7	4	7	3
9	5	4	9	2	5
2	4	8	2	9	1
5	2	9	1	2	4
2	4	5	2	9	F

B. Pentru matricea dată, costul minim va fi atins prin alegerea direcției cu suma minimă până la celula finală.

C. Orice drum valid trebuie să conțină exact 10 pași.

D. Costul minim pentru matricea dată este 41.

209. În cadrul unui festival de muzică sunt n artiști care trebuie să susțină un spectacol. Pentru fiecare artist i, se cunosc două valori: d_i - durata spectacolului în minute și f_i - numărul de fani prezenți la spectacolul său. Organizatorii vor să determine ordinea optimă a artiștilor pentru a maximiza satisfacția generală a publicului. Care dintre următoarele strategii sunt corecte, știind că un fan devine nemulțumit proporțional cu timpul de așteptare până la artistul său favorit? ✓ ?

A.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        If f[i] · d[i] > f[j] · d[j] then
            swap(d[i], d[j])
            swap(f[i], f[j])
        EndIf
    EndFor
EndFor
```

B.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        If f[i] > f[j] then
            swap(d[i], d[j])
            swap(f[i], f[j])
        EndIf
    EndFor
EndFor
```

C.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        If d[i] > d[j] then
            swap(d[i], d[j])
            swap(f[i], f[j])
        EndIf
    EndFor
EndFor
```

D.
```
For i ← 1, n − 1 execute
    For j ← i + 1, n execute
        If d[i] DIV f[i] >
                d[j] DIV f[j] then
            swap(d[i], d[j])
            swap(f[i], f[j])
        EndIf
    EndFor
EndFor
```

210. Într-o companie, mai multe echipe au solicitat sala de ședințe pentru prezentări. Pentru fiecare prezentare se cunosc: ✓ ?

(a) Ora de început solicitată - vectorul `start`

(b) Ora de sfârșit estimată - vectorul `finish`

Index	1	2	3	4	5	6	7	8	9	10
Start	1	3	0	5	3	5	6	8	8	2
Finish	4	5	6	7	8	9	10	11	12	13

Fiind o singură sală de ședințe disponibilă, două prezentări nu pot avea loc în același timp. Care este numărul minim de prezentări care trebuie reprogramate pentru o altă zi, astfel încât să nu existe suprapuneri în programul sălii?

A. 4. B. 5. C. 6. D. 7.

211. O inversiune într-o permutare p este o pereche (i, j), unde $i < j$ și $p[i] > p[j]$. De exemplu, în permutarea $[3, 1, 2]$, inversiunile sunt $(1, 2)$ și $(1, 3)$. ✓ ?

Care este complexitatea minimă în care se poate genera o permutare de lungime n minim lexicografic, având k inversiuni?

A. $O(\log n)$. B. $O(n \log n)$. C. $O(n)$. D. $O(n^2)$.

212. Într-un magazin sunt n obiecte; pentru fiecare se cunoaște greutatea G și valoarea V. Un hoț intră în magazin având un rucsac ce poate transporta o greutate maximă $GMax$. El va fura anumite obiecte, sau porțiuni de obiecte, astfel încât suma greutăților obiectelor furate să nu depășească $GMax$. Pentru următoarele obiecte, care este câștigul maxim pe care îl poate obține hoțul, știind că $GMax = 30$? ✓ ?

Greutatea (G)	**Valoarea (V)**
10	60
5	50
12	60
20	140

A. 220. B. 200. C. 190. D. 100.

213. Pentru interclasarea a două șiruri ordonate crescător, A având lungime a, și B, având lungimea b, numărul total de operații va fi $a + b$. Fie n șiruri, fiecare având lungimea $L[i]$. Toate aceste șiruri sunt ordonate crescător. Se dorește interclasarea tuturor celor n șiruri, fiind în total $n - 1$ interclasări. Pentru $n = 7$, iar $L = [2, 4, 7, 3, 1, 5, 6]$, care este numărul minim de operații necesare pentru a realiza interclasarea tuturor șirurilor? ✓ ?

A. 173. B. 28. C. 219. D. 375.

214. Într-un cuier există $n+1$ agățători, numerotate de la 1 la $n+1$. Primele n agățători conțin fiecare câte un palton, fiecare din ele având câte un număr de la 1 la n. Victor dorește mutarea lor, astfel încât, pentru fiecare i, paltonul cu numărul i să fie pus în agățătoarea cu numărul i. O mutare este reprezentată sub forma (i, j), indicând că paltonul aflat în agățătoarea i va fi mutat în agățătoarea j, iar aceasta se poate realiza doar dacă în agățătoarea j **NU** se află niciun palton. Care informații sunt adevărate? ✓ ?

A. Există unele configurații valide pentru care nu vom avea niciodată soluție.

B. Pentru $n = 6$, și configurația inițială $[6, 5, 1, 2, 4, 3]$, o succesiune validă de mutări este [(1, 7), (3, 1), (6, 3), (7, 6), (2, 7), (4, 2), (5, 2), (6, 3), (7, 6)].

C. O succesiune validă de mutări poate fi reprezentată sub forma unui graf neorientat, în care fiecare mutare reprezintă o muchie de la un nod spre altul. Fie c numărul de componente conexe ale acestui graf. Numărul minim de mutări necesare este egal cu $n + c$.

D. Mutarea unui palton deja aflat pe poziția corectă nu îl va ajuta pe Victor să-și realizeze scopul.

215. O tablă de șah de dimensiune $n \times n$, n par, este împărțită în dreptunghiuri de arii diferite, iar numărul de pătrățele albe este egal cu numărul de pătrățele negre, pentru fiecare dreptunghi în parte. Care este numărul maxim de dreptunghiuri pe care se pot obține? ✓ ?

A. n B. n^2 C. $n - 1$ D. $2 * n$.

216. Un depozit primește N comenzi. Pentru comanda i se știe: timpul de procesare t_i (minute), deadline-ul d_i (minute de la începutul zilei) și profitul p_i. O comandă aduce profit doar dacă e finalizată înainte de deadline. Dacă comenzile se procesează una după alta, care dintre următoarele afirmații nu sunt adevărate? ✓ ?

A. Pentru orice set de comenzi, sortarea după deadline produce soluția optimă.

B. Pentru setul $[(t = 3, d = 9, p = 6), (t = 2, d = 9, p = 8), (t = 4, d = 9, p = 4), (t = 1, d = 9, p = 7)]$, profitul maxim este 19.

C. Maximizarea profitului impune o sortare duală a comenzilor: mai întâi după deadline-uri în ordine descrescătoare pentru a gestiona urgența, apoi după raportul profit/timp (descrescător) pentru eficiență, asigurând procesarea în limita timpului disponibil.

D. Când toate deadline-urile sunt egale, sortarea după raportul profit/timp produce soluția optimă.

217. Se dă un set de n intervale de timp, fiecare interval $[a_i, b_i]$ reprezentând începutul și sfârșitul unui eveniment. Se dorește determinarea celui mai lung segment comun tuturor intervalelor. ✓ ?

Care dintre următoarele strategii nu rezolvă corect problema?

A. Se sortează intervalele în ordine crescătoare după punctul de început. Se începe cu intersecția primelor două intervale și se continuă prin intersecția cu fiecare interval următor.

B. Se sortează intervalele în ordine descrescătoare după punctul de sfârșit. Se ia intersecția dintre primul și ultimul interval și se continuă inter-secționând cu fiecare interval din mijloc.

C. Se sortează intervalele după punctul de început crescător. Se calculează intersecția dintre primul interval și fiecare interval următor, iar dacă intersecția devine nulă, se revine la intersecția maximă cunoscută.

D. Se sortează intervalele după punctul de sfârșit descrescător. Se ia intersecția dintre primul și ultimul interval, apoi se continuă inter-secționând cu fiecare interval, dar se revine la intersecția maximă dacă intersecția curentă devine nulă.

Combinatorică

Acest capitol acoperă

- Produs cartezian
- Submulțimi
- Permutări
- Aranjamente
- Combinări

9.1 Teorie

Introducere

Combinatorica este un domeniu al matematicii care explorează metodele de numărare, selecție și aranjare a elementelor din mulțimi finite. Multe dintre conceptele specifice combinatoricii sunt esențiale și în problemele de algoritmică, având o importanță majoră în informatică. Elementele de combinatorică stau la baza dezvoltării algoritmilor eficienți, optimizării soluțiilor pentru probleme complexe și îmbunătățirii performanțelor sistemelor informatice.

Produs cartezian

Definiție: Produsul cartezian a două mulțimi A și B este mulțimea tuturor perechilor ordonate (a, b), unde $a \in A$ și $b \in B$. Produsul cartezian se notează $A \times B$ și se definește formal astfel:

$$A \times B = \{(a, b) \mid a \in A \text{ și } b \in B\}.$$

Exemplu: Fie $A = \{1, 2\}$ și $B = \{x, y\}$. Produsul cartezian $A \times B$ este:

$$A \times B = \{(1, x), (1, y), (2, x), (2, y)\}.$$

Pași pentru generarea produsului cartezian:

- Pentru fiecare element a din mulțimea A, iterăm prin toate elementele din mulțimea B.
- Construim perechile ordonate (a, b), unde $b \in B$.
- Adăugăm fiecare pereche în mulțimea rezultat.

Rezultatul: Mulțimea tuturor perechilor ordonate este formată astfel:

$$A \times B = \{(a, b) \mid \forall a \in A, \forall b \in B\}.$$

Exemplu concret: Dacă $A = \{1, 2\}$ și $B = \{x, y\}$, atunci produsul cartezian este format în pași:

- Pentru $a = 1$: $(1, x)$, $(1, y)$.
- Pentru $a = 2$: $(2, x)$, $(2, y)$.

Rezultatul final este:

$$\{(1, x), (1, y), (2, x), (2, y)\}.$$

Procedură pentru generarea produsului cartezian

```
Algorithm GenerareProdusCartezian(A,B,n,m)
    For i ← 0, n − 1 execute                ▷ Mulțimea A cu n elemente
        For j ← 0, m − 1 execute            ▷ Mulțimea B cu m elemente
            Scrie (A[i], B[j])
        EndFor
    EndFor
EndAlgorithm
```

Submulțimi

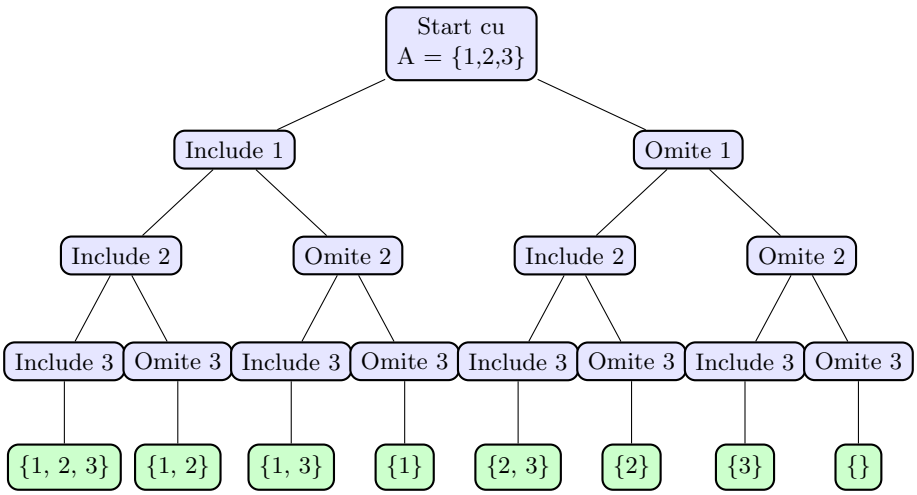
Definiție: Fie o mulțime A. O mulțime B este o **submulțime** a lui A dacă fiecare element al lui B aparține și mulțimii A. Formal, notăm acest lucru astfel:

$$B \subseteq A \iff \forall x \, (x \in B \implies x \in A).$$

Exemplu: Fie $A = \{1, 2, 3\}$. Toate submulțimile lui A sunt:

$$\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}.$$

Pentru a înțelege și vizualiza formarea submulțimilor, se examinează pe rând fiecare element al mulțimii A și se decide dacă se include sau nu. Acest proces se continuă până când se obțin toate submulțimile cu 2 elemente



Cardinalitatea mulțimii de submulțimi: O mulțime cu n elemente are 2^n submulțimi. De exemplu, dacă A are 4 elemente, atunci $2^4 = 16$ submulțimi.
Pași pentru generarea tuturor submulțimilor:

- Fiecare element poate fi inclus sau exclus dintr-o submulțime.
- Dacă mulțimea A are n elemente, generăm toate combinațiile posibile de n biți (0 sau 1), unde:
  - '1' indică includerea elementului.
  - '0' indică excluderea elementului.
- Generăm submulțimile corespunzătoare fiecărei combinații.

Exemplu concret: Fie $A = \{a, b\}$.
Generăm submulțimile:
1. Reprezentăm fiecare submulțime folosind biți: $00, 01, 10, 11$.
2. Interpretăm combinațiile:
- 00: $\{\}$ (submulțimea vidă).
- 01: $\{b\}$.
- 10: $\{a\}$.
- 11: $\{a, b\}$.
3. Rezultatul final este:

$$\{\{\}, \{a\}, \{b\}, \{a, b\}\}.$$

Algoritmul prezentat mai jos determină generarea tuturor submulțimilor prin intermediul metodei de backtracking și a apelurilor recursive.

Procedură pentru afișarea submulțimilor

```
Algorithm AfisareSubmultimi(A,x,k)
    For i ← 1, k execute
        If x[i] = 1 then
            Scrie A[i]
        EndIf
    EndFor
    Scrie linie nouă
EndAlgorithm

Algorithm GenerareSubmultimi(A,x,k)
    For i ← 0, 1 execute                    ▷ Valori posibile: 0 sau 1
        x[k] ← i
        If k = n then
            AfisareSubmultimi(A,x,k)
        Else
            GenerareSubmultimi(A,x,k+1)
        EndIf
    EndFor
EndAlgorithm
```

Procedură pentru generarea iterativă a submulțimilor

```
1:  Algorithm GenerareSubmultimiIterativ(A, n)
2:      totalSubmultimi ← 1
3:      For i ← 0, n − 1 execute
4:          totalSubmultimi ← totalSubmultimi ∗ 2
5:      EndFor
6:      For masca ← 0, totalSubmultimi − 1 execute
7:          Scrie " "
8:          m ← masca
9:          For i ← 0, n − 1 execute
10:             If m MOD 2 = 1 then
11:                 Scrie A[i]
12:             EndIf
13:             m ← m DIV 2
14:         EndFor
15:         Scrie ""
16:         Scrie linie nouă
17:     EndFor
18: EndAlgorithm
```

Permutări

Definiție: Fie o mulțime $A = \{a_1, a_2, \ldots, a_n\}$ cu n elemente distincte. O permutare este orice înșiruire a acestor elemente, în care fiecare element apare o singură dată, iar ordinea elementelor este luată în considerare. Fiecare astfel de înșiruire reprezintă un aranjament unic.
O altă definiție spune că o permutare a unei mulțimi finite și nevide de n elemente este o aplicație bijectivă, adică atât injectivă, cât și surjectivă, de la această mulțime în ea însăși. Cu alte cuvinte o permutare asupra mulțimii A este o funcție $f : A \to A$ astfel încât f este bijectivă.
Numărul total de permutări ale unei mulțimi cu n elemente este $n!$, unde:

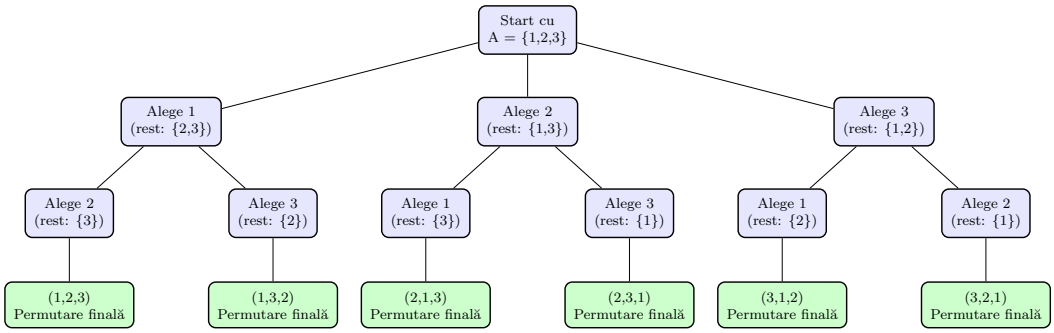
$$n! = n * (n-1) \ldots * 2 * 1.$$

Exemplu: Fie mulțimea $A = \{1, 2, 3\}$ și $n = 3$. Permutările posibile ale mulțimii sunt:

$$(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1).$$

În total, sunt $3! = 6$ permutări pentru o mulțime de 3 elemente.

Funcția ilustrată mai jos are ca scop generarea tuturor permutărilor unei mulțimi utilizând o abordare recursivă combinată cu tehnica de backtracking. Algoritmul explorează în mod sistematic toate permutările posibile, asigurându-se că fiecare dintre acestea este construită incremental.



Procedură pentru generarea permutărilor

```
1:  Algorithm GenerarePermutari(sir, n, index)
2:      If index = n then
3:          For i ← 0, n − 1 execute
4:              Scrie sir[i]
5:          EndFor
6:          Scrie linie nouă
7:          Return
8:      EndIf
9:      For i ← index, n − 1 execute
10:         Interschimba(sir[index], sir[i])
11:         GenerarePermutari(sir, n, index + 1)
12:         Interschimba(sir[index], sir[i])
13:     EndFor
14: EndAlgorithm
```

Pentru a se putea analiza diferențele dintre varianta recursivă și cea iterativă, este prezentată mai jos implementarea iterativă a algoritmului. Comparativ cu algoritmul recursiv, care explorează toate permutările unei mulțimi construind incremental soluțiile prin apeluri recursive și revenire la stările anterioare, algoritmul iterativ se bazează pe manipularea directă a vectorului și inversarea secțiunilor acestuia pentru a obține următoarea permutarea în ordine lexicografică.

Procedură pentru generarea permutărilor în ordine lexicografică

```
1:  Algorithm GenerarePermutariLexicografic(sir, n)
2:      For i ← 0, n − 1 execute
3:          Scrie sir[i]
4:      EndFor
5:      Scrie linie nouă
6:      i ← n − 2
7:      While i ≥ 0  AND sir[i] ≥ sir[i + 1] execute
8:          i ← i − 1
9:      EndWhile
10:     If i < 0 then
11:         Return fals
12:     EndIf
13:     j ← n − 1
14:     While sir[j] ≤ sir[i] execute
15:         j ← j − 1
16:     EndWhile
17:     Interschimba(sir[i], sir[j])
18:     stanga ← i + 1, dreapta ← n − 1
19:     While stanga < dreapta execute
20:         Interschimba(sir[stanga], sir[dreapta])
21:         stanga ← stanga + 1, dreapta ← dreapta − 1
22:     EndWhile
23:     Return adevarat
24: EndAlgorithm
```

Aranjamente

Definiție: Un aranjament de k elemente dintr-o mulțime A de n elemente este o selecție **ordonată** de k elemente. Numărul total de aranjamente se calculează cu formula:

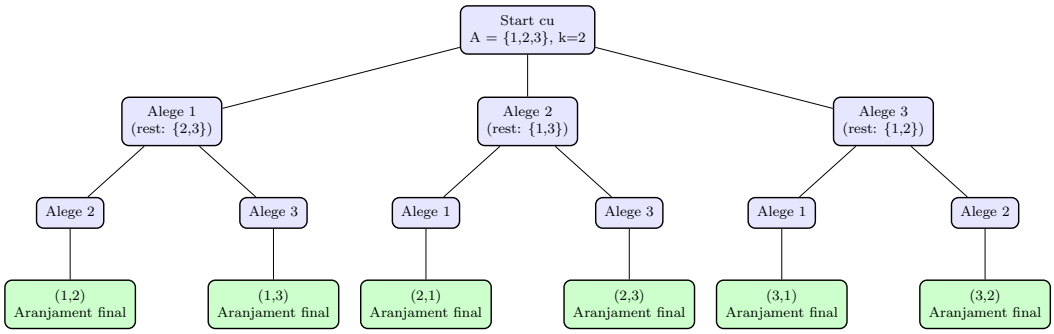
$$A_n^k = \frac{n!}{(n-k)!}.$$

Exemplu: Fie $A = \{1, 2, 3\}$ și $k = 2$. Aranjamentele de 2 elemente sunt:

$$(1,2), (1,3), (2,1), (2,3), (3,1), (3,2).$$

Pentru a vizualiza procesul de generare a aranjamentelor, se explorează toate posibilitățile de selecție ordonată a elementelor din A. Se alege succesiv câte un element disponibil, până când se obțin toate aranjamentele de dimensiune 2.

Funcția descrisă mai jos prezintă o metodă recursivă care construiește toate aranjamentele posibile ale unei mulțimi date. La fiecare pas, aceasta adaugă un element valid în soluția curentă și continuă recursiv până când soluția completă este obținută.



Procedură pentru generarea aranjamentelor

```
1:  Algorithm GenerareAranjamente(sol, pas, n, k)
2:      If pas = k then
3:          For i ← 0, k − 1 execute
4:              Scrie sol[i]
5:          EndFor
6:          Scrie linie nouă
7:          Return
8:      EndIf
9:      For i ← 1, n execute
10:         If nu EsteUtilizat(sol, i, pas) then
11:             sol[pas] ← i
12:             GenerareAranjamente(sol, pas + 1, n, k)
13:         EndIf
14:     EndFor
15: EndAlgorithm
```

Pentru a verifica faptul că un element există deja în soluția parțială generată până la un anumit pas, se utilizează funcția următoare, care este esențială pentru evitarea includerii unor elemente duplicate în soluție.

Funcție pentru verificarea elementelor utilizate

```
1:  Algorithm EsteUtilizat(sol, element, pas)
2:      For i ← 0, pas − 1 execute
3:          If sol[i] = element then
4:              Return adevarat
5:          EndIf
6:      EndFor
7:      Return fals
8:  EndAlgorithm
```

Următorul algoritm oferă o abordare iterativă, care gestionează soluția direct într-un vector.

Procedură pentru generarea iterativă a aranjamentelor

```
1:  Algorithm GenerareAranjamenteIterativ(n, k)
2:      sol ← vector de dimensiune k
3:      index ← 0
4:      element ← 1
5:      While index ≥ 0 execute
6:          gasit ← fals
7:          While element ≤ n execute
8:              If nu EsteUtilizat(sol, index, element) then
9:                  sol[index] ← element
10:                 gasit ← adevarat
11:                 break
12:             EndIf
13:             element ← element + 1
14:         EndWhile
15:         If gasit then
16:             If index = k − 1 then
17:                 For i ← 0, k − 1 execute
18:                     Scrie sol[i]
19:                 EndFor
20:                 Scrie linie nouă
21:                 element ← sol[index] + 1
22:             Else
23:                 index ← index + 1
24:                 element ← 1
25:             EndIf
26:         Else
27:             index ← index − 1
28:             If index ≥ 0 then
29:                 element ← sol[index] + 1
30:             EndIf
31:         EndIf
32:     EndWhile
33: EndAlgorithm
```

Combinări

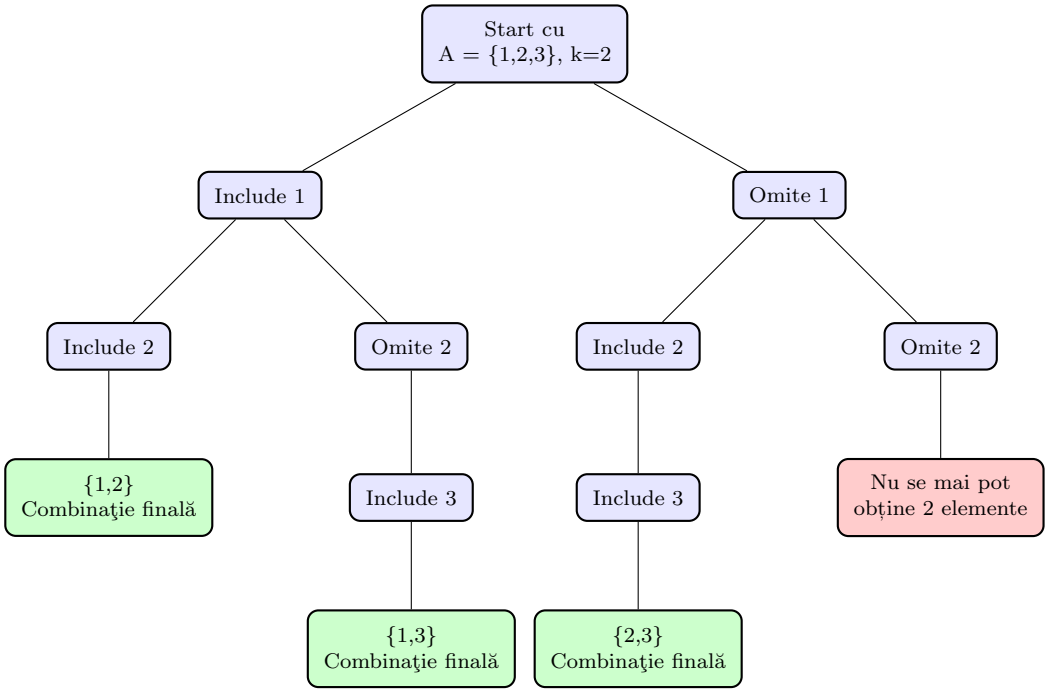
Definiție: O combinare de k elemente dintr-o mulțime A de n elemente este o selecție **neordonată** de k elemente. Numărul total de combinări se calculează cu formula:

$$C_n^k = \frac{n!}{k! \cdot (n-k)!}.$$

Exemplu: Fie $A = \{1, 2, 3\}$ și $k = 2$. Combinările de 2 elemente sunt:

$$\{1, 2\}, \{1, 3\}, \{2, 3\}.$$

Pentru a se putea vizualiza procesul de generare și pentru a se înțelege mai bine, se consideră următoare diagramă, unde se observă parcurgerea în ordine a elementelor din mulțimea A. La fiecare pas se ia decizia de a include sau nu un element, astfel încât la final să se obțină toate combinările posibile de dimensiune 2.



Următorul algoritm descris determină recursiv combinările, prin completarea treptată a vectorului soluției.

Backtracking pentru generarea combinărilor unei mulțimi

```
1:  Algorithm BACKTRACK(Sol, k, n, p)
2:      If k = 1 then
3:          start ← 1
4:      Else
5:          start ← Sol[k − 1] + 1
6:      EndIf
7:      For i ← start, n execute
8:          Sol[k] ← i
9:          If k = p then
10:             For j ← 1, k execute
11:                 Write Sol[j]
12:             EndFor
13:             Write newline
14:         Else
15:             BACKTRACK(Sol, k + 1, n, p)
16:         EndIf
17:     EndFor
18: EndAlgorithm
```

Pentru a oferi o alternativă la abordarea recursivă, următorul algoritm folosește o metodă

iterativă, care utilizează un vector pentru a stoca direct combinarea curentă și generează combinarea următoare pe baza celei actuale.

Generarea iterativă a combinărilor unei mulțimi

```
1: Algorithm GenerateCombinations(n, p)
2:     Sol ← un vector de dimensiune p + 1
3:     For i ← 1, p execute
4:         Sol[i] ← i
5:     EndFor
6:     hasNext ← true
7:     While hasNext execute
8:         For i ← 1, p execute
9:             Write Sol[i]
10:        EndFor
11:        Write newline
12:        i ← p
13:        While i > 0 and Sol[i] = n − p + i execute
14:            i ← i − 1
15:        EndWhile
16:        If i = 0 then
17:            hasNext ← false
18:        Else
19:            Sol[i] ← Sol[i] + 1
20:            For j ← i + 1, p execute
21:                Sol[j] ← Sol[j − 1] + 1
22:            EndFor
23:        EndIf
24:    EndWhile
25: EndAlgorithm
```

9.2 Probleme

218. Într-o sală de clasă, toate cele m scaune sunt ocupate de către m elevi. Fiecare elev are scaunul atribuit după un sistem simplu, elevului i fiindu-i atribuit scaunul i. Datorită faptului că toți elevii sunt buni prieteni, aceștia ocupă primul loc liber în momentul în care ajung în sala de clasă, fără a se supăra unul pe celălalt. Care este numărul posibilităților de aranjare a elevilor astfel încât exact doi elevi nu se află la locul atribuit? ✓ ?

A. $\frac{m(m-1)(m-2)}{6}$

B. $\frac{m(m-1)}{2}$

C. $\frac{m!}{m-2}$

D. Nicio variantă.

219. Se consideră algoritmul `Find(a, n)`, unde n este lungimea șirului a, iar a este un șir de n elemente întregi. De asemenea, se consideră existența unui algoritm `Sort(a, n)` care returnează șirul a de n elemente întregi ordonat crescător. ✓ ?

```
Algorithm FIND(a, n)
   a1 ← SORT(a, n)
   x ← 1
   For i ← 1,n execute
      x ← x* COUNT(a1, n, a[i])
   EndFor
   Return x
EndAlgorithm
Algorithm COUNT(v, n, val)
   a ← 0
   For i ← 1,n execute
      If v[i] = val then
         a ← a + 1
      EndIf
   EndFor
   Return a
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează permutările valide ale șirului a fără a genera explicit toate permutările;

B. Algoritmul determină câte permutări ale lui a respectă condiția ca pentru orice pereche (i, j) cu $i < j$ să fie $a[i] \geq a[j]$;

C. Complexitatea algoritmului nu depinde de sortare;

D. Nicio variantă.

220. În orașul tău au avut loc alegeri la care au participat doi candidați, candidatul A și candidatul B. Având în vedere că primul candidat a primit 5 voturi, iar candidatul B a primit 3, în câte moduri s-ar fi putut număra voturile astfel încât candidatul A să fie mereu în fața candidatului B sau la egalitate, știind că nu au fost numărate mai mult de 2 voturi consecutive pentru candidatul A? ✓ ?

A. 12; B. 5; C. 25; D. 56;

221. Fie n un număr natural par, diferit de 0. Știind că 'U' reprezintă o urcare, iar 'D' coborâre, se dorește să se construiască o secvență validă de n caractere astfel încât poți ajunge la nivelul de start (0) cu un număr egal de urcări și coborâri, fără a coborî sub nivelul 0. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Pentru $n = 6$, există 5 secvențe valide.

B. Pentru $n = 2$ există o singură secvență validă.

C. Numărul de secvențe valide pentru $n = 8$ este de 2 ori mai mare decât pentru $n = 4$.

D. Numărul de secvențe valide pentru $n = 8$ este de 3 ori mai mare decât pentru $n = 6$

222. Se consideră algoritmul `P(n, m, i, j, s)`, unde n și m sunt dimensiunile unei matrici, iar i și j sunt indicii poziției curente în matrice. De asemenea, se consideră existența algoritmilor `append(a,x)` care adaugă la finalul șirului a elementul x, și `remove(a)` care șterge ultimul element din șirul a. ✓ ?

```
Algorithm P(n, m, i, j, s)
   If i = n and j = m then
      print(s)
      Return
   EndIf
   If j < m then
      APPEND(s, "dreapta")
      P(n, m, i, j + 1, s)
      REMOVE(s)
   EndIf
   If i < n then
      APPEND(s, "jos")
      P(n, m, i + 1, j, s)
      REMOVE(s)
   EndIf
EndAlgorithm
```

Ce face algoritmul?

A. Determină un traseu valid de la colțul din stânga sus la colțul din dreapta jos al unei matrici și se oprește la primul traseu găsit;

B. Afișează toate traseele posibile de la colțul din stânga sus la colțul din dreapta jos al unei matrici;

C. Determină numărul de trasee posibile de la colțul din stânga sus la colțul din dreapta jos al unei matrici;

D. Determină dacă matricea conține un număr impar de celule și decide traseul pe baza acestui fapt.

223. Se consideră algoritmul `Count(n, k)`, unde n și k sunt două numere naturale cu $0 \leq k \leq n$. ✓ ?

```
Algorithm COUNT(n, k)
   If k = 0 or k = n then
      Return 1
   EndIf
   Return COUNT(n - 1, k - 1) + COUNT(n - 1, k)
EndAlgorithm
```

Ce face algoritmul?

A. Determină toate permutările posibile ale șirului de n elemente distincte;

B. Determină numărul de submulțimi de dimensiune k care se pot forma dintr-un șir de n elemente distincte;

C. Afișează toate submulțimile de dimensiune k care se pot forma din șir;

D. Determină toate permutările posibile ale șirului de n elemente.

224. Se consideră algoritmul `back(x, n, k)`, unde x este un șir de cel mult $n \leq 1000$ elemente, inițial toate 0. Algoritmul `print(x, n)` afișează toate elementele din x începând cu poziția 1 și până la poziția n, iar apoi afișează caracterul `newline`. În metoda principală se realizează, inițial, apelul `back(x, n, 1)`. ✓ ?

```
Algorithm BACK(x, n, k)
   For i ← n, 1, −1 execute
      x[k] ← i
      If ok(x, k) = 1 then
         If k = n then
            print(x, n)
         Else back(x, n, k+1)
         EndIf
      EndIf
   EndFor
EndAlgorithm
```

```
Algorithm OK(x, k)
   For i ← 1, k − 1 execute
      If x[k] = x[i] then
         Return 0
      EndIf
   EndFor
   Return True
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În urma executării apelului inițial, pe ecran se vor afișa toate submulțimile mulțimii $\{1, 2, \ldots n\}$;

B. În urma executării apelului inițial, pe ecran se vor afișa toate permutările mulțimii $\{1, 2, \ldots n\}$ în ordine lexicografică și are complexitatea $O(n!)$;

C. În urma executării apelului inițial, pe ecran se vor afișa toate permutările mulțimii $\{1, 2, \ldots n\}$, în ordine invers lexicografică și are complexitatea $O(n!)$;

D. În urma executării apelului inițial, pe ecran se vor afișa toate permutările mulțimii $\{1, 2, \ldots n\}$, în ordine invers lexicografică și are complexitatea $O(n! \cdot n)$.

225. Se consideră șirul $1, 2, 3, \ldots, n$. În câte moduri se pot aranja elementele șirului astfel încât în șirurile rezultate niciun element să nu își păstreze poziția inițială? ✓ ?

A. $n! - (n-2)!$

B. $\frac{n!}{0!} + \frac{n!}{1!} - \frac{n!}{2!} + \cdots + (-1)^{n+1} * \frac{n!}{n!}$

C. $n! * \sum_{k=1}^{n} \frac{(-1)^k}{k!}$

D. Niciuna dintre variantele de mai sus.

Problemele **226.**, **227.** se referă la următorii algoritmi `old(a, b)` și `new(a, b)`, unde a și b sunt numere naturale nenule ($1 \leq a \leq b \leq 10^3$).

```
Algorithm OLD(a, b)
   p ← 1
   For i ← 1, b execute
      p ← p * (a − i + 1)
   EndFor
   Return p
EndAlgorithm
```

```
Algorithm NEW(a, b)
   If b = 0 then
      Return 1
   EndIf
   _____________________
EndAlgorithm
```

226. Precizați cu ce trebuie completat spațiul liber din algoritmul `new(a, b)` pentru ca ambii algoritmi să returneze aceeași valoare pentru orice valori ale lui a și b: ✓ ?

A. `Return new(a - 1, b - 1) + new(a - 1, b)`

B. `Return new(a - 1, b - 1) * a`

C. `Return (a - b + 1) * new(a, b - 1)`

D. `Return new(a - 1, b - 1) * (a - b)`

227. Referitor la algoritmii `new(a, b)` și `old(a, b)` de mai sus, precizați care afirmații nu sunt adevărate: ✓ ?

A. Algoritmul `old(a, b)` calculează și returnează aranjamente de a luate câte b.

B. Pentru apelul `old(9, 4)` algoritmul returnează 3025.

C. Complexitatea de timp a algoritmului `old(a, b)` este $O(\log b)$.

D. Dacă în spațiul liber din algoritmul `new(a, b)` se completează cu `Return a * new(a, b - 1);` atunci algoritmul returnează a^b.

228. Se consideră algoritmul `generate(arr, c, n, k, idx, st, prod)`, unde `arr` este un vector ce conține n elemente naturale, `c` este un vector utilizat pentru generarea combinărilor, `prod` reprezintă produsul elementelor curente din combinare, iar `k` și `idx` sunt două variabile de tip întreg, cel mult 10^5. ✓ ?

```
Algorithm G(num)
   low ← 0, high ← num
   result ← 0
   While low ≤ high execute
      mid ← (low + high) / 2
      If mid * mid = num then
         Return mid
      Else If mid * mid < num then
         low ← mid + 1
         result ← mid
      Else
         high ← mid - 1
      EndIf
   EndWhile
   Return result
EndAlgorithm
```

```
Algorithm GENERATE(arr, c, n, k, idx,
st, prod)
   If idx = k then
      r ← G(prod)
      If r * r = prod then
         For i ← 0, k − 1 execute
            Write c[i], " "
         EndFor
         Write newline
      EndIf
      Return
   EndIf
   For i ← st, n − 1 execute
      c[idx] ← arr[i]
      GENERATE(arr, c, n, k, idx + 1,
i + 1, prod * arr[i])
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate pentru algoritmul prezentat?

A. Combinările generate sunt sortate lexicografic și sunt afișate indiferent de condiția asupra produsului lor.

B. Produsul fiecărei combinări generate este verificat pentru a determina dacă este egal cu suma pătratelor elementelor din combinare.

C. Algoritmul generează toate combinările de k elemente din `arr` și le afișează doar pe cele care au produsul elementelor un pătrat perfect.

D. Dacă funcția `G` returnează un rezultat diferit de rădăcina pătrată exactă, algoritmul omite configurația curentă.

229. Se consideră algoritmul `f(arr, used, curr, n, index)`, unde n este un număr natural ($1 \leq n \leq 10^5$), *arr*, *curr* și *used* sunt vectori cu n elemente naturale, iar *index* este un număr natural. ✓ ?

```
Algorithm F(arr, used, curr, n,
index)
   If index = n then
      diff ← curr[0] - curr[n -
1]
      If diff < 0 then
         diff ← -diff
      EndIf
      If G(diff) then
         For i ← 1, n execute
            Write curr[i], " "
         EndFor
         Write newline
      EndIf
      Return
   EndIf
   For i ← 1, n execute
      If not used[i] then
         used[i] ← 1
         curr[index] ← arr[i]
         F(arr, used, curr, n,
index + 1)
         used[i] ← 0
      EndIf
   EndFor
EndAlgorithm
```

```
Algorithm G(num)
   If num ≤ 1 then
      Return False
   EndIf
   For i ← 2, √num execute
      If num  MOD i = 0 then
         Return False
      EndIf
   EndFor
   Return True
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate pentru algoritmul prezentat?

A. Algoritmul generează toate permutările posibile ale vectorului `arr` și verifică pentru fiecare permutare dacă valoarea absolută a diferenței dintre primul și ultimul element este un număr prim.

B. Funcția `G(num)` verifică dacă un număr este prim, și algoritmul afișează permutările în care diferența dintre elementele consecutive este primă.

C. Pentru apelul funcției `f([1,2,3,4,5], [0,0,0,0,0], [0,0,0,0,0], 5, 0)` a cincea permutare afișată este $\{1, 3, 2, 5, 4\}$

D. Algoritmul afișează doar permutările care respectă condiția ca suma elementelor să fie număr prim.

230. Se consideră expresia: $E(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{2024} \cdot x^{2024}$, unde ✓ ? $a_0, a_1, a_2, \ldots, a_{2024}$ sunt numere reale nenule. Precizați numărul minim de operații de adunare și înmulțire necesare pentru a calcula $E(x)$:

A. 4048

B. 2024

C. 2049300

D. Niciuna dintre variantele de mai sus.

231. Se consideră algoritmul `Algo(n, k, idx, s, part)`, unde n este un număr natural ✓ ? $(1 \leq n \leq 10^5)$, k este un număr natural, `part` este un vector cu n elemente naturale $(arr[1], arr[2], \ldots, arr[n])$, iar `s` este un număr natural.

```
Algorithm ALGO(n, k, idx, s, part)
   If idx = k + 1 then
      If s = n then
         For i ← 1, k execute
            Write part[i]
            If i ≠ k then
               Write " "
            EndIf
         EndFor
         Write newline
      EndIf
      Return
   EndIf
   For i ← 1, n − s execute
      part[idx] ← i
      ALGO(n, k, idx + 1, s + i,
part)
   EndFor
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate pentru algoritmul prezentat

A. Algoritmul generează toate modurile în care se poate scrie numărul n ca sumă de k numere naturale distincte.

B. Algoritmul generează toate modurile în care se poate scrie n ca sumă de k numere naturale, fiecare mai mic decât n.

C. Dacă $k > n$, algoritmul nu va genera nicio soluție validă.

D. Algoritmul generează toate combinările de k elemente diferite din intervalul $[1, n]$.

232. Se consideră algoritmul `F(arr, used, perm, n, idx)`, unde `arr` este un vector cu n elemente, `used` este un vector utilizat pentru marcarea elementelor folosite, iar `perm` este un vector pentru permutarea curentă. Funcția `Ok` verifică dacă suma elementelor de pe pozițiile pare dintr-o permutare este mai mare decât suma celor de pe pozițiile impare. ✓ ?

```
Algorithm F(arr, used, perm, n,
idx)
   If idx > n then
      If OK(perm, n) then
         For i ← 1, n execute
            Write perm[i], " "
         EndFor
         Write newline
      EndIf
      Return
   EndIf
   For i ← 1, n execute
      If not used[i] then
         used[i] ← 1
         perm[idx] ← arr[i]
         F(arr, used, perm, n, idx
+ 1)
         used[i] ← 0
      EndIf
   EndFor
EndAlgorithm
```

```
Algorithm OK(perm, n)
   evenSum ← 0, oddSum ← 0
   For i ← 1, n execute
      If i MOD 2 = 1 then
         oddSum ← oddSum + perm[i]
      Else
         evenSum ← evenSum + perm[i]
      EndIf
   EndFor
   Return evenSum > oddSum
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate pentru algoritmul prezentat?

A. Algoritmul generează toate permutările vectorului `arr` care au suma elementelor de pe pozițiile pare mai mare decât suma celor de pe pozițiile impare.

B. Algoritmul `Ok` returnează `true` dacă suma pozițiilor impare este mai mare decât suma pozițiilor pare.

C. Algoritmul generează toate permutările vectorului `arr` și afișează doar acelea în care diferența dintre suma elementelor de pe pozițiile pare și suma elementelor de pe pozițiile impare este pozitivă.

D. Pentru apelul funcției `F([1,2,3,4], [0,0,0,0], [0,0,0,0], 4, 0)` a șasea permutare afișată este $\{3, 2, 1, 4\}$

233. Se consideră algoritmul `generate(current, size, n, k, number)`, unde unde `n` ✓ ? este un număr natural ($1 \leq n \leq 10^5$),, `current` este un vector cu n elemente naturale ($current[1], current[2], \ldots, current[n]$), `k`, `size` și `number` sunt numere naturale:

```
Algorithm GENERATE(current, size,
n, k, number)
   If size = k then
      isValid ← true
      For i ← 1, k − 1 execute
         If GCD(current[i],
current[i + 1]) ≠ 1 then
            isValid ← false
            Break
         EndIf
      EndFor
      If isValid then
         For i ← 1, k execute
            Write current[i]
            If i ≠ k then
               Write " "
            EndIf
         EndFor
         Write newline
      EndIf
      Return
   EndIf
   For i ← number + 1, n execute
      current[size + 1] ← i
      GENERATE(current, size + 1,
n, k, i)
   EndFor
EndAlgorithm
```

```
Algorithm GCD(a, b)
   While b ≠ 0 execute
      temp ← b
      b ← a MOD b
      a ← temp
   EndWhile
   Return a
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul generează toate secvențele strict crescătoare de k numere din mulțimea $1, 2, ..., n$ cu proprietatea că oricare două numere consecutive sunt prime între ele;

B. Pentru apelul funcției `Generate(current, 0, 5, 3, 0)`, secvența $(2, 3, 5)$ este o soluție validă;

C. Pentru apelul funcției `Generate(current, 0, 7, 4, 0)`, secvența $(2, 5, 6, 7)$ nu este o soluție validă;

D. Complexitatea algoritmului este $O(n^k * k * \log n)$.

Problemele **234.** și **235.** se referă la următorii algoritmi `f(n, k)` și `g(n, k)`, unde n și k sunt numere naturale nenule ($1 \leq k \leq n \leq 10^3$).

```
Algorithm F(n, k)
   If n < k then
      Return 0
   EndIf
   If k = 0 OR k = n then
      Return 1
   EndIf
   If k = 1 then
      Return n
   EndIf
   Return n * F(n − 1, k − 1) DIV k
EndAlgorithm
```

```
Algorithm G(n, k)
   If n < k then
      Return 0
   EndIf
   If k = 0 OR k = n then
      Return 1
   EndIf
   If k = 1 then
      Return n
   EndIf
   _____________
EndAlgorithm
```

234. Cu ce poate fi completat spațiul liber din algoritmul `g(n, k)` pentru ca ambii algoritmi să returneze aceeași valoare pentru orice valori ale lui n și k? ✓ ?

A. `Return g(n - 1, k) * n DIV (n - k)`

B. `Return g(n - 1, k - 1) + g(n - 1, k)`

C. `Return g(n - 1, k - 1) DIV k * n`

D. `Return f(n - 1, k - 1) * n DIV k`

235. Dacă spațiul liber din algoritmul `f(n, k)` se completează astfel încât funcția să returneze același rezultat ca algoritmul `g(n, k)`, precizați care afirmații sunt false: ✓ ?

A. Pentru apelul `f(13, 7)` algoritmul returnează 1715.

B. Apelurile `f(n, k)` și `f(n, n - k)` returnează întotdeauna aceeași valoare.

C. Algoritmul `f(n, k)` calculează și returnează aranjamente de n luate câte k.

D. Algoritmul `f(n, k)` calculează și returnează combinări de n luate câte k.

236. Teo vrea să aranjeze pe un raft n cărți în așa fel încât nici o carte să nu își păstreze poziția inițială. Care dintre următorii algoritmi o ajută pe Teo să afle în câte moduri poate aranja cele n cărți pe raft? Algoritmul `factorial(n)` calculează $n!$. ✓ ?

A.

```
Algorithm CHAPTER(n)
   d ← 0
   s ← 1
   For k = 0, n execute
      d ← d + s * factorial(n) DIV factorial(k)
      s ← −s
   EndFor
   Return d
EndAlgorithm
```

B.

```
Algorithm FICTION(n)
   If n = 0 then
      Return 1
   EndIf
   If n = 1 then
      Return 0
```

```
        EndIf
        Return (n − 1) * (fiction(n − 1) + fiction(n − 2))
    EndAlgorithm
```

C.

```
Algorithm NOVEL(n)
    d ← 0
    s ← 1
    For k = 0 to n execute
        d ← d + s * factorial(n) DIV factorial(n − k)
        s ← s * (−1)
    EndFor
    Return d
EndAlgorithm
```

D.

```
Algorithm STORY(n)
    d ← 0
    s ← 1
    For k = 1, n execute
        d ← d + s * factorial(n) DIV factorial(k)
        s ← s * (−1)
    EndFor
    Return d
EndAlgorithm
```

237. Se consideră toate submulţimile unei mulţimi M cu n elemente şi $x \in M$. De câte ori apare x în toate aceste submulţimi? ✓ ?

A. $\frac{n!}{2}$ B. n^2 C. 2^{n-1} D. 2^n

238. Se consideră o mulţime S cu n elemente şi o submulţime T cu k elemente, deci $k \leq n$. Care este numărul de submulţimi ale mulţimii iniţiale S, în care sunt incluse toate elementele submulţimii T? ✓ ?

A. $2^{\frac{n}{k}}$ B. 2^{n-2} C. $\frac{2^n}{k}$ D. 2^{n-k}

239. Se consideră o mulțime M cu n elemente distincte din mulțimea numerelor naturale şi toate submulţimile acesteia. Este aleasă, apoi, o valoare $k \leq n$ şi submulțimile A_k, $|A_k| = k$, iar pentru fiecare A_k, se construieşte mulțimea B_k, astfel încât $A_k \cup B_k = M$, dar $A_k \cap B_k = \emptyset$. Care este suma tuturor elementelor din toate mulţimile B_k? ✓ ?

A. $C_{n-1}^{k-1} \cdot \sum_{x \in M} x$

B. $C_{n-1}^{k} \cdot \sum_{x \in M} x$

C. $C_{n}^{k} \cdot \sum_{x \in M} x$

D. $(n-k) \cdot \sum_{x \in M} x$

Acest capitol acoperă

- Grafuri neorientate;
- Grafuri orientate;
- Arbori.
- Conceptul de arbore binar, tipurile de arbori și metode de reprezentare și parcurgere a arborilor
- Conceptul de arbore binar de căutare, proprietățile acestuia și operațiile efectuate posibile

10.1 Teorie

10.1.1 Grafuri neorientate

Un **graf neorientat** este o pereche ordonată $G = (V, A)$, unde:

- V este o mulțime finită și nevidă de elemente, numite **noduri** (sau **vârfuri**);
- A este o mulțime finită de submulțimi cu două elemente din V, numite **muchii**

Într-un graf neorientat, muchiile sunt bidirecționale, ceea ce înseamnă că existența unei muchii dintr-un nod u înspre un nod v implică și existența unei muchii din v înspre u
Exemplu: Considerăm un graf neorientat $G = (V, A)$ cu:

- $V = \{1, 2, 3, 4, 5\}$
- $A = \{(1, 2), (2, 3), (3, 1), (3, 4)\}$

Graful poate fi reprezentat astfel:

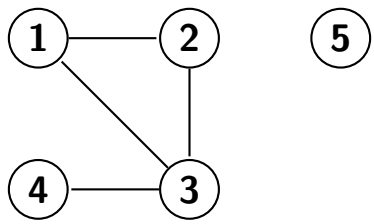


Figura 10.1 Exemplu de graf neorientat

Metode de reprezentare a unui graf neorientat

Există mai multe metode de reprezentare a unui graf neorientat, dintre care cele mai utilizate sunt:

- **Matricea de adiacență**

  Matricea de adiacență a unui graf neorientat cu n noduri este o matrice binară A (deci $A[i][j] \in \{0,1\}\ \forall i,j$) de dimensiune $n \times n$, unde:

  $$A[i][j] = A[j][i] = \begin{cases} 1 & \text{dacă există o muchie între nodurile } i \text{ și } j, \\ 0 & \text{altfel.} \end{cases}$$

  Exemplu: Matricea de adiacență pentru graful anterior este:

  $$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

  Se observă că matricea de adiacență a unui graf neorientat este simetrică față de diagonala principală.

- **Listă de muchii**

  În această metodă, muchiile sunt stocate într-un tablou static, fiecare element fiind o pereche de noduri care reprezintă o muchie.

  Exemplu: Pentru graful de mai sus, lista de muchii este:

  $$A = \{(1,2),\ (2,3),\ (3,1),\ (3,4)\}$$

- **Liste de adiacență**

  În această metodă, fiecare nod v are atribuită o listă a succesorilor săi, adică nodurile u pentru care există o muchie de la v la u. Lista de adiacență pentru un graf neorientat reprezintă pentru fiecare nod lista nodurilor către care există muchii.

  Exemplu: Pentru graful de mai sus, lista de adiacență este:

  $$1 : \{2,3\}$$
  $$2 : \{1,3\}$$
  $$3 : \{1,2,4\}$$
  $$4 : \{3\}$$
  $$5 : \{\}$$

Gradele nodurilor

Într-un graf neorientat, se numește grad al unui vârf numărul de vârful adiacente cu acesta, care este echivalent cu numărul de muchii incidente cu acesta.
Exemplu: Pentru graful de mai sus, avem:

- Grad nodului 1 egal cu 2
- Grad nodului 2 egal cu 2
- Grad nodului 3 egal cu 3
- Grad nodului 4 egal cu 1
- Grad nodului 5 egal cu 0

Graf parțial și subgraf

- **Subgraf**: Un graf $H = (V_H, A_H)$ este un subgraf al grafului $G = (V, A)$ dacă $V_H \subseteq V$ și $A_H \subseteq A$.
- **Graf parțial**: Un graf parțial al grafului G este un graf obținut prin eliminarea unor muchii din G, dar păstrând toate nodurile.

  Exemplu: Dacă eliminăm muchia $(3, 4)$ din graful de mai sus, obținem un graf parțial.

Graf complet

- Un graf neorientat în care există o muchie de la fiecare nod la fiecare alt nod (cu excepția arcelor de la un nod la el însuși) se numește **complet**.

  Într-un graf complet cu n noduri, numărul total de muchii este dat de formula combinatorică:

  $$C_n^2 = \frac{n(n-1)}{2}$$

  unde C_n^2 reprezintă numărul de modalități de a alege două noduri din n, iar fiecare pereche de noduri va fi conectată printr-o muchie. Aceasta este echivalent cu formula

  $$\frac{n(n-1)}{2}$$

  deoarece fiecare nod este conectat la celelalte $n-1$ noduri, iar fiecare muchie este numărată de două ori (o dată pentru fiecare nod conectat).

Lanțuri în grafuri neorientate

- **Lanț**: Numim lanț o succesiune de noduri $L = [v_1, v_2, v_3, \dots v_k]$ cu proprietatea că oricare două vârfuri consecutive sunt adiacente.
- Vârfurile v_1 şi v_k se numesc extremitățile lanțului. Lungimea unui lanț este $k-1$, reprezentând numărul total de muchii din care este format.
- Un lanț care conține toate nodurile distincte, două câte două, se numește **lanț elementar**.

Exemplu: În graful nostru neorientat menționat anterior:

- Lanțul $L = [1, 2, 3]$ este un lanț cu lungimea 2, deoarece muchiile sunt $(1, 2)$ și $(2, 3)$.
- Lanțul $L = [3, 4]$ este un lanț de lungime 1, având muchia $(3, 4)$.
- Lanțul $L = [1, 2, 3, 4]$ este un lanț elementar, deoarece toate nodurile sunt distincte și toate muchiile sunt adiacente între ele.

Cicluri în grafuri neorientate

- **Ciclu**: Numim ciclu un lanț simplu în care primul vârf este identic cu ultimul. Dacă toate vârfurile sunt distincte, mai puțin primul și ultimul, se numește **ciclu elementar**.

- Lungimea unui ciclu este egală cu numărul de muchii din ciclu. Lungimea minimă a unui ciclu este 3.
- Un graf neorientat care nu conține niciun ciclu se numește **aciclic**.

Exemplu: În graful nostru neorientat, [1, 2, 3, 1] este un ciclu elementar

Componente conexe

- Un graf neorientat este **conex** dacă, pentru fiecare două noduri distincte din graf, există un lanț având extremitățile în cele două noduri.
- Se numește **componentă conexă** a unui graf $G = (X, U)$ un subgraf $H = (Y, V)$, conex, al lui G care are proprietatea că nu există nici un lanț în G care să lege un vârf din Y cu un vârf din X–Y. Subgraful H este conex și maximal cu această proprietate (dacă s-ar mai adăuga un vârf nu ar mai fi conex.)

Graf Hamiltonian. Graf Eulerian

- Se numește **graf Hamiltonian** un graf care conține un **ciclu Hamiltonian**. Se numește **ciclu Hamiltonian** un ciclu elementar care conține toate vârfurile grafului.
- Un graf neorientat care are $n \geq 3$ vârfuri şi gradul oricărui vârf verifică inegalitatea este mai mare sau egal cu $\lfloor \frac{n}{2} \rfloor$, atunci graful este Hamiltonian.
- Se numește **graf Eulerian** un graf care conține un **ciclu Eulerian**. Se numește **ciclu Eulerian** un ciclu elementar care conține toate muchiile grafului.

10.1.2 Grafuri orientate

Un **graf orientat** (sau digraf) este o pereche ordonată $G = (V, A)$, unde:

- V este un set nevid de noduri (sau vârfuri);
- A este un set de arce, fiecare arc fiind o pereche ordonată de noduri (u, v), unde $u, v \in V$.

Într-un graf orientat, arcele au o direcție, ceea ce înseamnă că (u, v) este diferit de (v, u).
Exemplu: Considerăm un graf orientat $G = (V, A)$ cu:

- $V = \{1, 2, 3, 4\}$
- $A = \{(1, 2), (2, 3), (3, 1), (3, 4)\}$

Graful poate fi reprezentat astfel:

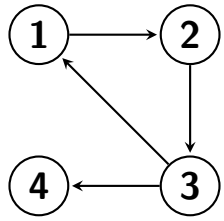


Figura 10.2 Exemplu de graf orientat

Metode de reprezentare a unui graf orientat

Există mai multe metode de reprezentare a unui graf orientat, dintre care cele mai utilizate sunt:

- **Matricea de adiacență**

  Matricea de adiacență a unui graf orientat cu n noduri este o matrice binară A (deci $A[i][j] = 0$ sau 1 $\forall i, j$) de dimensiune $n \times n$, unde $A[i][j] = 1$ dacă există un arc de la nodul i la nodul j, și $A[i][j] = 0$ în caz contrar.

  Exemplu: Matricea de adiacență pentru graful anterior este:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Listă de arce**

  În această metodă, arcele sunt stocate într-un tablou static, fiecare element fiind o pereche de noduri care reprezintă un arc.

  Exemplu: Pentru graful de mai sus, lista de arce este:

$$A = \{(1, 2),\ (2, 3),\ (3, 1),\ (3, 4)\}$$

- **Liste de adiacență**

  În această metodă, fiecare nod v are atribuită o listă a succesorilor săi, adică nodurile u pentru care există un arc de la v la u. Lista de adiacență pentru un graf orientat reprezintă pentru fiecare nod lista nodurilor către care există arce.

  Exemplu: Pentru graful de mai sus, lista de adiacență este:

$$1 : \{2\}$$
$$2 : \{3\}$$
$$3 : \{1, 4\}$$
$$4 : \{\}$$

Gradele nodurilor

Spre deosebire de grafurile neorientate, când vine vorba despre grafurile orientate, fiecare nod are **două** grade:

- **Gradul intern**: numărul de arce care intră în nod.
- **Gradul extern**: numărul de arce care ies din nod.

Exemplu: Pentru nodul 3 în graful de mai sus:

- Grad intern: 1 (arcul $(2, 3)$)
- Grad extern: 2 (arcele $(3, 1)$ și $(3, 4)$)

Graf parțial și subgraf

- **Subgraf**: Un graf $H = (V_H, A_H)$ este un subgraf al grafului $G = (V, A)$ dacă $V_H \subseteq V$ și $A_H \subseteq A$.
- **Graf parțial**: Un graf parțial al grafului G este un graf obținut prin eliminarea unor arce din G, dar păstrând toate nodurile.

  Exemplu: Dacă eliminăm arcul $(3, 4)$ din graful de mai sus, obținem un graf parțial.

Graf complet și graf turneu

- **Graf complet orientat**: Un graf orientat în care există un arc de la fiecare nod la fiecare alt nod (cu excepția arcelor de la un nod la el însuși) se numește **complet**.
- **Graf turneu**: Un graf orientat complet fără cicluri de lungime 2 (adică pentru orice pereche de noduri u și v, există exact un arc între ele, fie (u, v), fie (v, u), dar nu ambele) se numește graf **turneu**.

Lanț, drum și circuit în grafuri orientate

- **Lanț orientat**: Numim lanț orientat o succesiune de arce $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ în care nodurile pot fi repetate. Lungimea lanțului este egală cu numărul de arce din care este alcătuit.
- **Drum orientat**: Un drum orientat este un lanț orientat în care nodurile sunt distincte.
- **Circuit orientat**: Numim circuit orientat un lanț orientat în care primul și ultimul nod coincid, și toate celelalte noduri sunt distincte.

Exemplu: În graful nostru, $(1, 2), (2, 3), (3, 1)$ formează un circuit orientat.

Componente tare conexe

Un graf orientat este **tare conex** dacă există un drum orientat de la orice nod la orice alt nod. O **componentă tare conexă** este un subgraf tare conex maximal.

Exemplu: În graful de mai sus, nodurile $\{1, 2, 3\}$ formează o componentă tare conexă, deoarece există drumuri orientate între oricare două noduri din acest set.
Mai jos este prezentată o implementare în pseudocod pentru parcurgerea în adâncime (DFS - explicată anterior la subsecțiunea *Grafuri neorientate*) a unui graf orientat, care poate fi utilizată pentru a determina componentele tare conexe.

DFS pentru grafuri orientate

```
1: Algorithm DFS(G, v)
2:     Marcăm v ca vizitat
3:     For fiecare nod u adiacent cu v (adică există un arc (v, u)) execute
4:         If u nu este vizitat then
5:             DFS(G, u)
6:         EndIf
7:     EndFor
8: EndAlgorithm
```

Pentru a găsi componentele tare conexe, se poate utiliza algoritmul lui Kosaraju, care constă în două parcurgeri DFS și transpunerea grafului.
Complexitate timp: $O(n+m)$, unde n este numărul de noduri și m numărul de arce.

10.1.3 Arbori

Introducere

Arborii sunt structuri de date fundamentale, utilizate în multiple domenii ale informaticii, fiind utili pentru a reprezenta relații ierarhice între elemente. Aceștia sunt întâlniți în contextul gestionării bazelor de date, optimizării algoritmilor de căutare și sortare.

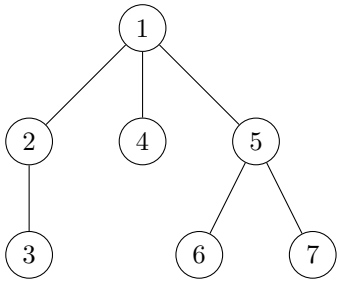
Arbori liberi

Un arbore este un graf aciclic și conex, format dintr-o mulțime finită de noduri și muchii. Acesta este cunoscut și sub denumirea de **arbore liber**.
Un arbore liber se caracterizează prin următoarele proprietăți:

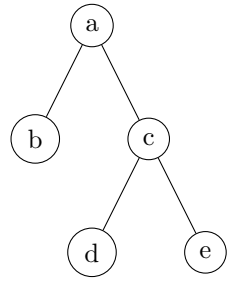
- Un arbore cu un număr de n vârfuri are întotdeauna $n-1$ muchii
- Acesta respectă proprietatea de minimalitate, adică eliminarea oricărei muchii determină pierderea conexității grafului
- Într-un arbore, între oricare două vârfuri există un lanț elementar unic
- Respectă proprietatea maximalității, adică adăugarea oricărei muchii are ca rezultat generarea unui ciclu
- Orice arbore cu un număr de $n \geq 2$ noduri conține cel puțin două noduri terminale

Arbori cu rădăcină

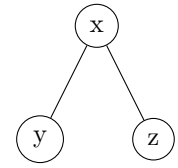
Un **arbore cu rădăcină** este un arbore în care unul dintre noduri este desemnat ca rădăcină.



Rădăcina este nodul 1.



Rădăcina este nodul a.



Rădăcina este nodul x.

O astfel de reprezentare a acestora, care definește un punct de referință unic, conferă arborelui obținut o structură ierarhică bine definită. Mai mult decât atât, aceasta permite introducerea și definirea unor concepte precum **relația părinte - copil**, **nivelurile de**

adâncime, și diverse metode de parcurgere a arborilor cu rădăcină, precum **preordine**, **inordine** și **postordine** și structuri mai avansate, cum ar fi arborii binari de căutare, simpli sau balansați.

Proprietățile nodurilor într-un arbore cu rădăcină

Se dă un arbore a și un nod x în arborele dat. Proprietățile și conceptele specifice nodurilor în astfel de structuri sunt definite după cum urmează:

Ascendent:

- Un nod y este considerat **ascendent** al unui nod x dacă se află pe lanțul de la rădăcină până la x, excluzând însuși x.
- Relația de ascendență respectă următoarele proprietăți:
  - Rădăcina arborelui este ascendentul tuturor nodurilor din arbore și aceasta nu are ascendenți
  - Dacă există o muchie (y, x), iar y este ascendentul lui x, atunci y este cunoscut sub denumirea de **ascendent direct** al lui x sau **tatăl** nodului x

Descendent

- Un nod y este numit **descendent** al unui nod x dacă nodul x aparține lanțului care unește rădăcina r cu y.
- Descendența poate fi descrisă prin următoarele proprietăți:
  - Dacă există o muchie (x, y), atunci nodul y se numește **descendent direct** al lui x sau **fiul** nodului x
  - Un nod fără descendenți poartă denumirea de **frunză**

Frate:

- Două noduri care au același nod părinte sunt denumite noduri **frați**.

Nivel sau adâncimea:

- Lungimea lanțului care leagă rădăcina arborelui de un nod x definește **nivelul** sau **adâncimea** nodului x

Înălțimea arborelui:

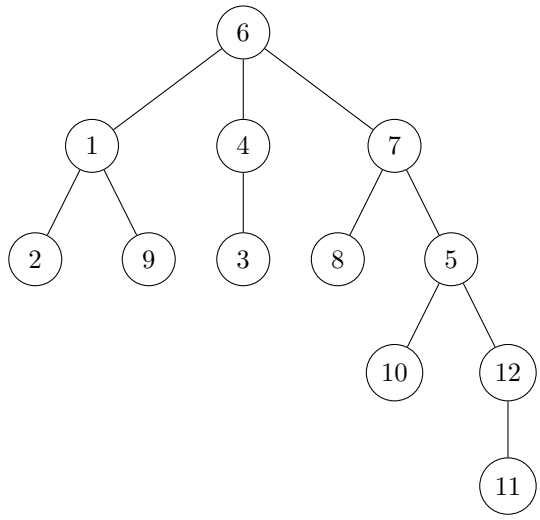
- Lungimea maximă a unui lanț care pornește de la rădăcină și ajunge la oricare nod din arbore definește **înălțimea arborelui**

Subarbore:

- Orice nod din arbore, împreună cu toți descendenții săi, constituie un **subarbore** al arborelui

Exemplu

Se dă arborele următor:



Pe baza acestuia putem exemplifica următoarele:

- Rădăcina arborelui este nodul 6
- Înălțimea arborelui este 4
- Nodurile 2, 9, 3, 8, 10 și 11 reprezintă nodurile frunză ale arborelui
- Nodurile 2 și 9 sunt noduri frați
- Nodurile 5, 10, 12, 11 formează un subarbore
- Descendenții nodului 7 sunt nodurile 5, 8, 9, 10 și 11
- Nodurile 10 și 12 sunt descendenții direcți ai nodului 5
- Ascendenții nodului 10 sunt nodurile 5, 7 și 6
- Nodul 4 reprezintă ascendentul direct sau tatăl nodului 3
- Descompunerea arborelui pe niveluri:
  - Nivelul 0 conține nodul rădăcină 6
  - Nivelul 1 conține nodurile 1, 4 și 7
  - Nivelul 2 conține nodurile 2, 9, 3, 8 și 5
  - Nivelul 3 conține nodurile 10 și 12
  - Nivelul 4 conține nodul 11

Există două modalități prin care un arbore poate fi reprezentat în memorie: ascendent și descendent

Reprezentarea ascendentă

Această metodă se bazează pe ideea faptului că pentru fiecare nod al arborelui se memorează informații despre ascendenții direcți ai acestora. Astfel se obține un vector de părinți, unde:

- Fiecare poziție k din vector corespunde unui nod din arbore
- Valoarea $p[k]$ indică părintele nodului k
- Pentru rădăcina arborelui r, valoarea este 0: $v[r] = 0$

Pentru arborele prezentat ca exemplu mai sus, vectorul de părinți este:

Nod (k)	1	2	3	4	5	6	7	8	9	10	11	12
Părinte ($p[k]$)	6	1	4	6	7	0	6	7	1	5	12	5

Observații importante

- Valoarea 0 este unică în vectorul de părinți și corespunde rădăcinii
- Nodurile frunză sunt acele noduri ale căror valori nu apar în vectorul de părinți

Vectorul părinților este util, deoarece permite determinarea lanțurilor în arbore, de la oricare nod spre rădăcină. Acesta presupune pornirea de la un nod dat x, iar mai apoi identificarea părintelui acestuia, $y = v[x]$, acest proces repetându-se până când algoritmul ajunge la un nod z pentru care $v[z] = 0$, aceasta reprezentând rădăcina. Mai jos este prezentat un exemplu de implementare a determinării lanțurilor

Structura **Nod** reprezintă un nod individual al arborelui, incluzând următoarele câmpuri: `data: număr întreg`, care stochează informația asociată nodului corespunzător, și `părinte: Nod`, care reprezintă un pointer către nodul părinte al nodului curent. Prin intermediul acestui câmp este permisă construirea unei reprezentări a arborelui prin legături ascendente.

Structura Nod

1: *data* : număr întreg
2: *parinte* : *Nod*

Funcția ilustrată mai jos construiește relațiile ale unui arbore prin actualizarea vectorului `tați` și a vectorului `valueToIndex`. Pentru fiecare nod din arbore, funcția verifică dacă există un părinte corespunzător. Dacă da, câmpul din vectorul `tati` este actualizat la valoarea `data` a părintelui, iar valoarea nodului este asociată cu indexul său în `valueToIndex`. În caz contrar, nodul este considerat ca fiind rădăcina, iar valoarea atribuită este 0. Valoarea n reprezintă numărul de noduri din arbore.

Construire Vector de Tați și Corespondență

```
1: Algorithm CONSTRUIESTETATISICORESPONDENTA(noduri, tati, valueToIndex, n)
2:     For i ← 0, n − 1 execute
3:         nodeValue ← noduri[i].data
4:         valueToIndex[nodeValue] ← i
5:         If noduri[i].parinte ≠ nul then
6:             tati[i] ← noduri[i].parinte.data
7:         Else
8:             tati[i] ← 0
9:         EndIf
10:    EndFor
11: EndAlgorithm
```

Construirea și afișarea lanțului unui nod x către rădăcina arborelui sunt realizate prin intermediul funcției implmentate mai jos. Vectorul `valueToIndex` asociază valorile nodurilor cu indexurile corespunzătoare în vectorul `tați`.

Funcția Determină Lanț

```
1: Algorithm DETERMINĂLANȚ(x, tati, valueToIndex)
2:     Write x
3:     While x ≠ 0 execute
4:         i ← valueToIndex[x]
5:         x ← tati[i]
6:         If x ≠ 0 then
7:             Write → x
8:         EndIf
9:     EndWhile
10: EndAlgorithm
```

Reprezentarea descendentă

Reprezentarea descendentă a arborilor presupune organizarea informației astfel încât fiecare nod al arborelui să memoreze și să fie asociat cu datele despre descendenții săi direcți. Pentru această reprezentare se pot folosi listele de adiacență, similar cu reprezentarea prin liste de adiacență a grafurilor. Asta înseamnă că fiecare nod al arborelui este asociat cu o listă de descendenți direcți.
Pentru exemplul prezentat mai sus, avem următoarea reprezentare descendentă pentru fiecare nod al arborelui:
$\{6 : \{1, 4, 7\}, 1 : \{2, 9\}, 4 : \{3\}, 7 : \{8, 5\}, 5 : \{10, 12\}, 12 : \{11\}, 2 : \{\}, 3 : \{\}, 8 : \{\}, 9 : \{\},$
$10 : \{\}, 11 : \{\}\}$
În continuare, este prezentat un exemplu detaliat care ilustrează implementarea acestei reprezentări prin intermediul unor structuri și algoritmi specifici. Structurile **Descendent** și **Nod** sunt esențiale pentru gestionarea relațiilor dintre noduri într-un arbore.

Structura **Nod** definește un nod individual al arborelui, incluzând următoarele elemente: `data: număr întreg`, pentru informația stocată în nod, și `descendenți: Descendent`, un pointer către primul descendent al nodului.

Structura **Descendent** reprezintă un element al listei de descendenți ai unui nod și are următoarele câmpuri: `nod: Nod`, care este un pointer către nodul copil, și `next: Descendent`, un pointer către următorul descendent.

Structura Nod

1: *data* : număr întreg
2: *descendenti* : *Descendent*

Structura Descendent

1: *nod* : *Nod*
2: *next* : *Descendent*

Adăugarea unui copil nou la lista de descendenți a unui nod existent este facilitată de funcția prezentată mai jos. Prin modul de implementare prezentat, structura listei de descendenți este actualizată în mod eficient, fără a fi nevoie de o parcurgere a întregii liste.

Funcția Adaugă Descendent

```
1: Algorithm AdaugaDescendent(parinte, copil)
2:     nou ← Nod
3:     nou ← copil
4:     nou.next ← parinte.descendenti
5:     parinte.descendenti ← nou
6: EndAlgorithm
```

Funcția implementată mai jos are rolul de a afișa toate nodurile de pe un anumit nivel al arborelui. Aceasta funcționează prin intermediul unor combinații de verificări condiționale și apeluri recursive pentru a parcurge arborele pe nivelul specificat.

Funcția Afișează Nivel

```
1: Algorithm afiseazaNivel(nod, nivel)
2:     If nod = nul then
3:         Return
4:     EndIf
5:     If nivel = 0 then
6:         Write nod.data
7:         curent ← nod.descendenti
8:         Write "["
9:         While curent ≠ nul execute
10:            Write curent.nod.data
11:            If curent.next ≠ nul then
12:                Write ", "
13:            EndIf
14:            curent ← curent.next
15:        EndWhile
16:        Write "]"
17:    Else
18:        curent ← nod.descendenti
19:        While curent ≠ nul execute
20:            afiseazaNivel(curent.nod, nivel - 1)
21:            curent ← curent.next
22:        EndWhile
23:    EndIf
24: EndAlgorithm
```

Afișarea pe niveluri a unui arbore, pornind de la rădăcină și continuând până la frunzele arborelui, se realizează cu ajutorul următoarei funcții:

Funcția Afișează Arbore pe Niveluri

```
1: Algorithm afiseazaArborePeNiveluri(radacina)
2:     h ← inaltimeArbore(radacina)
3:     For nivel ← 0, h − 1 execute
4:         afiseazaNivel(radacina, nivel)
5:     EndFor
6: EndAlgorithm
```

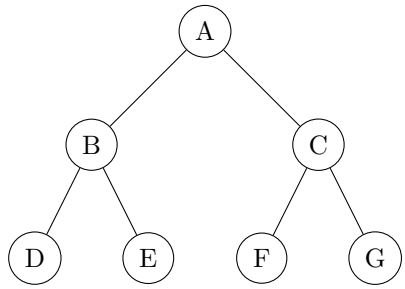
Funcția ilustrată mai jos are rolul de a determina înălțimea unui arbore pornind de la un nod dat. Funcția este utilizată ulterior pentru afișarea arborelui pe niveluri. Totodată, aceasta folosește o abordare recursivă, parcurgând lista descendenților pentru fiecare nod.

Funcția Înălțime Arbore

```
1: Algorithm InaltimeArbore(nod)
2:     If nod = nul then
3:         Return 0
4:     EndIf
5:     maxInaltime ← 0
6:     curent ← nod.descendenti
7:     While current ≠ nul execute
8:         maxInaltime ← max(maxInaltime, inaltimeArbore(curent.nod))
9:         current ← current.next
10:    EndWhile
11:    Return maxInaltime + 1
12: EndAlgorithm
```

Arbori binari

Un **arbore binar** este un arbore cu rădăcină în care fiecare nod are cel mult doi descendenți direcți, denumiți descendentul sau fiul stâng și descendentul sau fiul drept. Alternativ, un arbore binar poate fi definit recursiv ca fiind fie vid, fie format dintr-un nod rădăcină și doi subarbori binari (stâng și drept).
Următorul arbore exemplificat reprezintă un arbore binar:

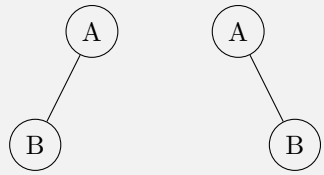


În exemplul de mai sus:

- nodul A reprezintă rădăcina
- Nodul B este descendentul stâng direct a nodului rădăcină A, iar nodul C este descendentul direct drept. Similar nodurile D și E pentru nodul părinte B și nodurile F și G pentru nodul părinte C
- Nodurile D, E, F și G sunt frunze

Observație importantă

- Dacă un nod are un singur nod descendent, atunci este necesar să fie precizat dacă acesta este fie un descendent stâng, fie unul drept. Următorii doi arbori sunt considerați ca fiind distincți:



Proprietățile arborelui binar

- Într-un arbore binar cu n noduri și înălțimea h se respectă relația $h \geq \log_2(n+1)-1$. Această relație spune că fiecare nivel din arbore, trebuie să aibă cel puțin un element, iar astfel înălțimea nu poate fi mai mare decât n.
- Numărul maxim de noduri pe care îl poate avea un nivel i este de 2^i noduri.
- Numărul maxim de noduri într-un arbore binar cu înălțimea h este $2^(h+1)-1$
- Într-un arbore binar în care fiecare nod are fie 0, fie 2 copii, numărul nodurilor frunză este mereu cu unul mai mare decât numărul nodurilor care au doi copii.
- Într-un arbore binar care conține cel puțin un nod, dacă n este numărul total de noduri și m este numărul total de muchii, atunci se verifică egalitatea $m = n - 1$. Fiecare nod dintr-un arbore are exact un părinte, cu excepția rădăcinii. Astfel, pentru un număr total de n noduri, $n - 1$ noduri vor avea exact un părinte.

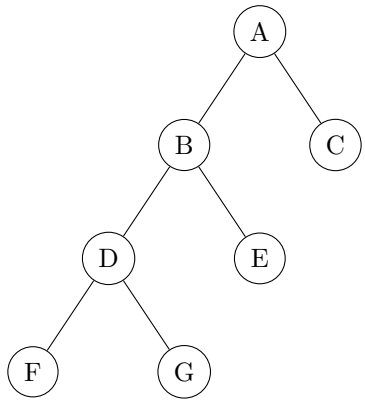
Tipuri speciale de arbori binari

Arbore binar strict

Un **arbore binar strict** este un arbore binar în care fiecare nod care nu este terminal are exact doi copii.

Proprietăți:

- Pentru un arbore binar cu x noduri frunză, numărul de noduri este dat de relația $n = 2 * x - 1$.
- Un arbore binar strict are un număr impar de noduri.

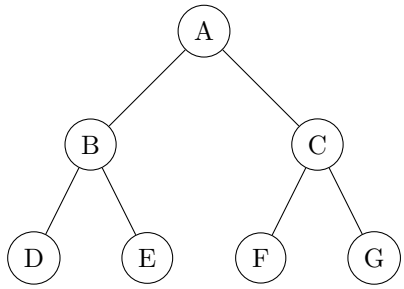


Arbore binar plin

Un **arbore binar plin** este un arbore binar în care toate nodurile au doi copii și toate nodurile frunză se află pe același nivel.

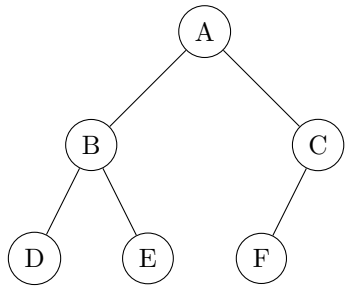
Proprietăți:

- Toate nodurile frunză sunt situate pe același nivel, iar astfel se poate spune că arborele binar plin este un caz particular al arborelui binar strict.
- Un arbore binar plin de înălțime h conține $2^{h+1} - 1$ noduri.
- Un arbore binar are care un număr de x noduri terminale, are un număr de $n = 2 * k - 1$ noduri.



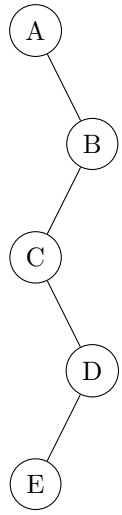
Arbore binar complet

Un arbore binar se numește **arbore binar complet** dacă toate nivelurile, cu excepția ultimului, sunt complet populate, iar nodurile de pe ultimul nivel sunt poziționate cât mai la stânga posibil. Totodată, un arbore binar plin este și un arbore binar complet. O proprietate importantă a arborilor binari completi este aceea că înălțimea unui astfel de arbore este $\lfloor \log_2(n) \rfloor$, unde n reprezintă numărul de noduri.



Arborele binar degenerat

Un **arbore binar degenrat** este un tip special de arbore binar, în care fiecare nod are fie 0, fie 1 copil. Structura unui arbore degenerat se aseamănă cu cea a unei liste înlănțuite, deoarece fiecare nod are un singur descendent direct, cu excepția frunzei.



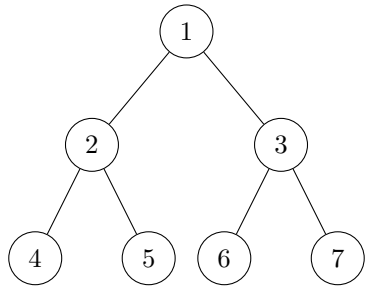
Reprezentarea arborilor binari

La fel ca în cazul precedent al arborilor cu rădăcină, aceștia pot fi reprezentați prin utilizarea unor metode similare deja cunoscute, dar anumite ajustări sunt necesare pentru a respecta structura specifică arborilor binari

Reprezentarea prin referințe ascendente

Această reprezentare necesită informații suplimentare față de arbori cu rădăcină generală. Pentru fiecare nod al arborelui, în plus față de părintele acestuia este nevoie să se cunoască și dacă nodul este un descendent stâng sau drept. Așadar este necesară utilizarea a doi vectori pentru realizarea reprezentării:

- Vectorul părinților $p[k]$:
  - Stochează părintele nodului k.
  - În cazul în care nodul k este rădăcina arborelui, atunci $p[k] = 0$
- Vectorul poziției ($tip[k]$), care indică ce tip de descendent este nodul k:
  - Dacă k este rădăcina, atunci $tip[k] = 0$
  - Dacă k este copilul stâng al părintelui din $p[k]$, atunci $tip[k] = -1$
  - Dacă k este copilul drept al părintelui din $p[k]$, atunci $tip[k] = 1$



Pentru arborele dat mai sus ca exemplu vectorii corespunzători reprezentării ascendente sunt următorii:

k	1	2	3	4	5	6	7
Părinte[$p(k)$]	0	1	1	2	2	3	3
Tip[$tip(k)$]	0	-1	1	-1	1	-1	1

Reprezentarea prin referințe descendente

Pentru realizarea reprezentării prin referințe descendente este necesară identificarea rădăcinii arborelui binar, precum și stocarea informațiilor despre copilul stâng și copilul drept pentru fiecare nod din arbore. Această reprezentare se poate realiza fie folosind vectori, fie prin utilizarea alocării dinamice.

- **Reprezentarea folosind vectori:**
  - Vectorul $l[k]$ reține indexul copilului stâng al nodului k. Dacă nodul k nu are copil stâng, atunci $l[k] = -1$.
  - Vectorul $r[k]$ reține indexul copilului drept al nodului k. Dacă nodul k nu are copil drept, atunci $r[k] = -1$.

  Reprezentarea descendentă folosind vectori pentru arborele prezentat mai sus ca exemplu este următoarea:

k	1	2	3	4	5	6	7
l[k]	2	4	6	-1	-1	-1	-1
r[k]	3	5	7	-1	-1	-1	-1

- **Reprezentarea folosind alocare dinamică:** Fiecare nod al arborelui reprezentat sub forma unei structuri conține următoarele câmpuri:
  - Valoarea nodului *valoare* stochează informația din nod.
  - Pointerii *stanga* și *dreapta* indică înspre copilul stâng și, respectiv copilul drept al nodului.

Structura Nod

1: $valoare$: număr întreg
2: $stanga$: Nod
3: $dreapta$: Nod

Parcurgerea arborilor binari

Parcurgerea arborilor binari reprezintă procesul prin care fiecare nod al arborelui este vizitat o singură dată, într-o anumită ordine. Acest proces este esențial pentru efectuarea operațiilor asupra arborilor, deoarece permite manipularea și analiza datelor stocate în arbore. Există mai multe tipuri de parcurgeri, printre care se numără: parcurgerea în **preordine**, parcurgerea în **inordine** și parcurgerea în **postordine**

Parcurgerea în preordine

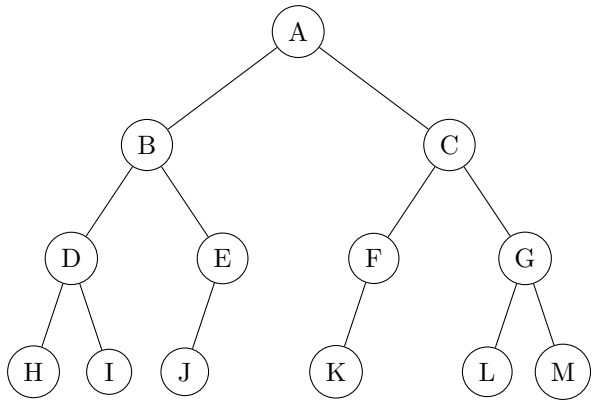
- Rădăcina este vizitată prima, iar mai apoi este urmată de subarborele stâng și apoi de subarborele drept
- Ordinea vizitării este: Rădăcină → Subarborele stâng → Subarborele drept

Parcurgerea în inordine

- Subarborele stâng este vizitat primul, urmat de nodul rădăcină și apoi subarborele drept
- Ordinea vizitării este: Subarborele stâng → Rădăcină → Subarborele drept

Parcurgerea în postordine

- Prima oară este vizitat subarborele stâng, iar mai apoi acesta este urmat de subarborele drept și, în final, de rădăcină
- Ordinea vizitării este: Subarborele stâng → Subarborele drept → Rădăcină



Pentru arborele dat ca exemplu mai sus, parcurgerile sunt următoarele:

- **Preordine:** A → B → D → H → I → E → J → C → F → K → G → L → M
- **Inordine:** H → D → I → B → J → E → A → K → F → C → L → G → M
- **Postordine:** H → I → D → J → E → B → K → F → L → M → G → C → A

Construirea unui arbore pe baza metodelor de parcurgere

Un arbore binar poate fi reconstruit în mod unic atunci când sunt cunoscute parcurgerile sale în **preordine** și **inordine**, sau în **postordine** și **inordine**. Acest lucru este posibil datorită informațiilor complementare furnizate de fiecare dintre aceste parcurgeri.

Pentru a oferi un contraexemplu, dacă se folosesc parcurgerile în **preordine** și **postordine**, nu putem reconstrui un arbore unic, deoarece ambele parcurgeri descriu ordinea vizitării nodurilor, dar nu oferă informații suficiente despre relațiile dintre noduri și subarborii lor. În consecință, acest lucru poate duce la mai multe configurații valide care satisfac aceleași parcurgeri.

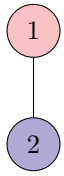
Pentru a înțelege mai bine procesul de construire a unui arbore binar, se consideră următoarele parcurgeri:

- **Preordine**: $1, 2, 4, 5, 3, 6, 7$
- **Inordine**: $4, 2, 5, 1, 6, 3, 7$

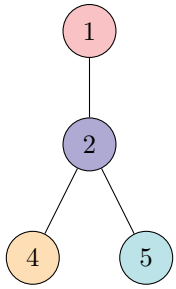
Se construiește arborele pas cu pas:

- **Determinarea rădăcinii:**
  Primul nod din parcurgerea în **preordine** este 1. Acesta reprezintă rădăcina arborelui. În parcurgerea în **inordine**, poziția nodului 1 împarte lista în două subliste:
  - Subarborele stâng: [4, 2, 5]
  - Subarborele drept: [6, 3, 7]

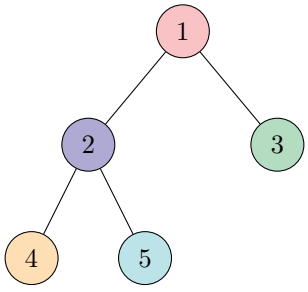
- **Construirea arborelui stâng:**
  Subarborele stâng este definit de lista [4, 2, 5] în parcurgerea în **inordine**. Următorul nod din **preordine** este 2, care devine rădăcina subarborelui stâng. În **inordine**, nodul 2 împarte lista în:
  - Subarborele stâng: [4]
  - Subarborele drept: [5]



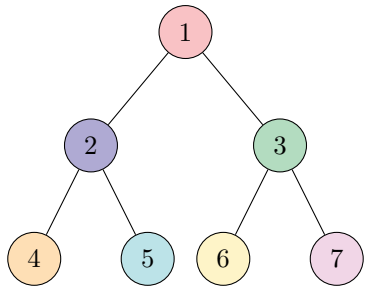
- **Adăugarea nodurilor subarborelui stâng al lui 2:**
  Subarborele stâng al lui 2 este definit de nodul 4, iar subarborele drept este reprezentat de nodul 5. Acestea sunt noduri frunză, deoarece listele lor corespunzătoare sunt goale.



- **Construirea subarborelui drept**:
  Subarborele drept este reprezentat de lista [6, 3, 7] în parcurgerea în **inordine**. Următorul nod din **preordine** este 3, care devine rădăcina subarborelui drept. În parcurgerea în **inordine**, nodul 3 împarte lista în:
  - Subarborele stâng: [6]
  - Subarborele drept: [7]



- **Adăugarea nodurilor subarborelui drept al lui 3**:
  Subarborele stâng al lui 3 este nodul 6, iar subarborele drept este nodul 7. Acestea sunt noduri terminale sau frunză.



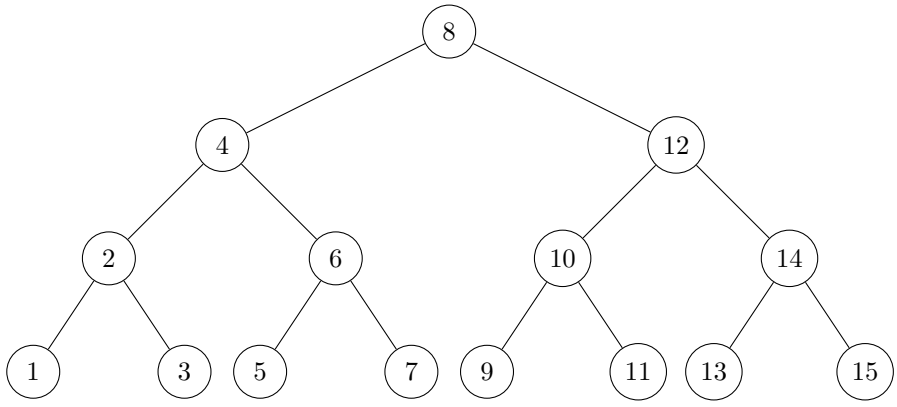
Astfel, prin urmarea pașilor prezentați mai sus, arborele construit este complet.

Arbori binari de căutare

Un **arbore binar de căutare** este un arbore binar care respectă următoarele proprietăți pentru fiecare nod x:

- Toate valorile din subarborele stâng al lui x sunt mai mici decât valoarea nodului x
- Toate valorile din subarborele drept al lui x sunt mai mari decât valoarea nodului x
- Fiecare subarbore drept și stâng este la rândul său un arbore binar de căutare
- În general, arborele nu conține noduri cu valori duplicate, dar acest lucru poate fi permis prin ajustarea definițiilor de comparare

Următorul exemplu ilustrează un arbore binar de căutare, unde se pot observa proprietățile prezentate mai sus:



Observații importante

- Parcurgerea în inordine a arborelui se realizează în ordinea crescătoare a valorilor nodurilor.
- Căutarea unui nod care are o valoarea dată se realizează eficient, deoarece dacă rădăcina nu este egală cu valoarea căutată, căutarea va continua fie în subarborele drept, fie în subarborele stâng, în funcție de relația dintre valoarea dată și valoarea rădăcinii.

În exemplele implementate ce urmează se va folosi reprezentarea prin alocare dinamică a unui nod într-un arbore binar de căutare, deoarece aceasta este cel mai des întâlnită în situații reale când se lucrează cu astfel de structuri de date.

Pentru reamintire, un nod într-un arbore binar de căutare are următoarea structură, unde `valoare` reprezintă valoarea stocată în nod, `stanga` reprezintă copilul stâng al nodului, iar `dreapta` copilul drept al nodului

Structura Nod

1: $valoare$: număr întreg
2: $stanga$: Nod
3: $dreapta$: Nod

Principalele operații efectuate cu un arbore binare de căutare sunt:

- Căutarea unui nod în arbore
- Inserarea unui nod nou
- Ștergerea unui nod din arbore
- Traversarea nodurilor din arbore

Căutarea unui nod din arbore

Căutarea unui nod reprezintă procesul de traversare a arborelui pentru a verifica dacă în arbore există un element dat sau nu, utilizând proprietățile specifice ale acestuia.

Pentru a căuta un nod cu o valoare specifică x într-un arbore binar se compară valoarea căutată x cu valoarea nodului curent y:

- Dacă $x = y$, atunci nodul căutat a fost găsit și căutarea se oprește
- Dacă $x < y$, căutarea continuă în subarborele stâng
- Dacă $x > y$, căutare continuă în subarborele drept

În continuare sunt prezentate cele două modalități de căutare a unui element în arborele binar de căutare:

Funcția de căutare implementată în mode iterativ

```
1: Algorithm CautaNod(arbore, x)
2:     nod_curent ← arbore.radacina
3:     While nod_curent ≠ nul execute
4:         If nod_curent.valoare = x then
5:             Return nod_curent
6:         Else If x < nod_curent.valoare then
7:             nod_curent ← nod_curent.stanga
8:         Else
9:             nod_curent ← nod_curent.dreapta
10:        EndIf
11:    EndWhile
12:    Return nul
13: EndAlgorithm
```

Funcția de căutare implementată în mod recursiv

```
1: Algorithm CautaNod(radacina, x)
2:     If radacina = nul or radacina.valoare = x then
3:         Return radacina
4:     EndIf
5:     If x < radacina.valoare then
6:         Return CautaNod(radacina.stanga, x)
7:     Else
8:         Return CautaNod(radacina.dreapta, x)
9:     EndIf
10: EndAlgorithm
```

Inserarea unui nod nou

Inserarea unui nod presupune adăugarea unei valori noi în arbore astfel încât acesta să respecte în continuare proprietățile specifice arborilor de căutare.
Pentru a insera un nod cu valoarea x într-un arbore binar de căutare se procedează astfel:

- Dacă arborele este vid, se creează un nod nou care devine rădăcina arborelui
- Dacă arborele este nevid, atunci se compară valoarea x cu valoarea nodului curent:
  - Dacă $x <$ *valoarea nodului curent*, atunci inserarea continuă în subarborele stâng
  - Dacă $x >$ *valoarea nodului curent*, atunci inserarea continuă în subarborele drept
- Când se ajunge la un loc gol (`nul`), se creează un nod nou ce conține valoarea dorită x și se adaugă în poziția corespunzătoare din partea stângă sau dreaptă a părintelui său, astfel încât să păstreze strucutra unui arbore binar de căutare

Următoarele două metode ilustrează implementarea funcției de inserare a unei valori în arborele binar de căutare:

Funcția de inserare implementată în mod iterativ

```
1:  Algorithm InserareIterativa(radacina, valoare)
2:      nodNou ← Nod()
3:      nodNou.valoare ← valoare
4:      nodNou.stanga ← nul
5:      nodNou.dreapta ← nul
6:      If radacina = nul then
7:          Return nodNou
8:      EndIf
9:      nodCurent ← radacina
10:     parinte ← nul
11:     While nodCurent ≠ nul execute
12:         parinte ← nodCurent
13:         If valoare < nodCurent.valoare then
14:             nodCurent ← nodCurent.stanga
15:         Else If valoare > nodCurent.valoare then
16:             nodCurent ← nodCurent.dreapta
17:         Else
18:             Return radacina
19:         EndIf
20:     EndWhile
21:     If valoare < parinte.valoare then
22:         parinte.stanga ← nodNou
23:     Else
24:         parinte.dreapta ← nodNou
25:     EndIf
26:     Return radacina
27: EndAlgorithm
```

Funcția de inserare implementată în mod recursiv

```
1:  Algorithm Inserare(radacina, valoare)
2:      If radacina = NULL then
3:          nodNou ← Nod
4:          nodNou.valoare ← valoare
5:          nodNou.stanga ← nul
6:          nodNou.dreapta ← nul
7:          Return nodNou
8:      EndIf
9:      If valoare < radacina.valoare then
10:         radacina.stanga ← Inserare(radacina.stanga, valoare)
11:     Else If valoare > radacina.valoare then
12:         radacina.dreapta ← Inserare(radacina.dreapta, valoare)
13:     EndIf
14:     Return radacina
15: EndAlgorithm
```

Ștergerea unui nod din arbore

Ștergerea unui nod dintr-un arbore binar de căutare este o operație ce presupune eliminarea unui nod specificat, menținând în același timp proprietățile arborelui. Această operație poate fi mai complexă decât operațiile de căutare și inserare, deoarece trebuie să fie tratate diferite cazuri în funcție de poziția nodului de șters.
Pentru realizarea ștergerii unui nod x din arbore se iau în considerare următorii pași:

- Nodul x este un nod frunză:
  - Nodul este eliminat din arbore.
  - Referința părintelui către nod este actualizată la `nul`, astfel încât părintele să nu mai indice către nodul șters.
  - Ștergerea unui nod frunză nu modifică relațiile dintre celelalte moduri.
- Nodul x are un singur copil:
  - Copilul stâng sau drept al nodului șters înlocuiește nodul șters.
  - Referința părintelui către nodul șters este actualizată pentru a indica copilul existent
  - Nodul șters este eliminat.
  - Subarborele copilului existent devine conectat direct la părintele nodului șters.
- Nodul x are doi copii
  - Nodul x este înlocuit fie cu **succesorul în inordine** care reprezintă cel mai mic nod din subarborele drept (acesta nu poate avea copil stâng), fie cu **predecesorul în preordine**, care reprezintă cel mai mare nod din subarborele stâng (acesta nu poate avea copil drept).
  - Valoarea nodului succesor sau predecesor este copiată în locul nodului x, ce trebuie șters.
  - Nodul succesor sau predecesor este eliminat din poziția sa inițială, deoarece devine duplicat în arbore.

Pașii algoritmului general pentru ștergerea unui nod din arbore:

- Se aplică algoritmul prezentat de căutare a unui nod în arbore, pentru a găsi nodul de șters.
- Se identifică unul dintre cele trei cazuri de mai sus și se aplică strategia corespunzătoare de ștergere a nodului, pentru a menține structura corectă și proprietățile arborelui.

În cele ce urmează este ilustrată implementarea ștergerii unui nod dintr-un arbore binar de căutare:

Funcția de ștergere a unui nod implementată în mod recursiv

```
1: Algorithm StergeNod(radacina, valoare)
2:     If radacina = nul then
3:         Return nul
4:     EndIf
5:     If valoare < radacina.valoare then
6:         Return StergeNod(radacina.stanga, valoare)
7:     Else If valoare > radacina.valoare then
8:         Return StergeNod(radacina.dreapta, valoare)
9:     Else
10:        nodSters ← radacina.valoare
11:        If radacina.stanga = nul and radacina.dreapta = nul then
12:            Delete radacina
13:            Return nodSters
14:        Else If radacina.stanga = nul then
15:            temp ← radacina.dreapta
16:            Delete radacina
17:            radacina ← temp
18:            Return nodSters
19:        Else If radacina.dreapta = nul then
20:            temp ← radacina.stanga
21:            Delete radacina
22:            radacina ← temp
23:            Return nodSters
24:        Else
25:            succesor ← Minimul(radacina.dreapta)
26:            radacina.valoare ← succesor.valoare
27:            StergeNod(radacina.dreapta, succesor.valoare)
28:            Return nodSters
29:        EndIf
30:    EndIf
31: EndAlgorithm
```

Funcția de găsire a succesorului în inordine, adică a celui mai mic nod din subarborele drept

```
1: Algorithm GasesteMinim(radacina)
2:     While radacina.stanga ≠ nul execute
3:         radacina ← radacina.stanga
4:     EndWhile
5:     Return radacina
6: EndAlgorithm
```

Traversarea nodurilor din arbore

Nodurile arborilor binari de căutare pot fi traversați în oricare mod prezentat anterior, adică inordine, preordine și postordine. Mai jos se pot observa implementările celor trei metode:

Traversare în Preordine

```
1: Algorithm PREORDINE(nod)
2:     If nod ≠ nul then
3:         Write nod.valoare
4:         PREORDINE(nod.stanga)
5:         PREORDINE(nod.dreapta)
6:     EndIf
7: EndAlgorithm
```

Traversare în Inordine

```
1: Algorithm INORDINE(nod)
2:     If nod ≠ nul then
3:         INORDINE(nod.stanga)
4:         Write nod.valoare
5:         INORDINE(nod.dreapta)
6:     EndIf
7: EndAlgorithm
```

Traversare în Postordine

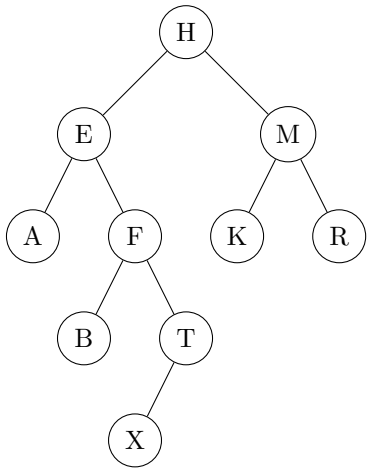
```
1: Algorithm POSTORDINE(nod)
2:     If nod ≠ nul then
3:         POSTORDINE(nod.stanga)
4:         POSTORDINE(nod.dreapta)
5:         Write nod.valoare
6:     EndIf
7: EndAlgorithm
```

Concluzie

Arborii reprezintă una dintre structurile de date fundamentale și printre cele mai des utilizate în informatică, oferind o soluție eficientă pentru organizarea și manipularea datelor ierarhice. Au fost analizate concepte importante, precum arborii liberi, arborii cu rădăcină, arborii binari și cei de căutare. Fiecare dintre aceste structuri are aplicabilități practice, de la gestionarea bazelor de date până la implementarea și optimizarea diverșilor algoritmi utilizați în industrie și în situații reale.

10.2 Probleme

240. Se dă următorul arbore binar: ✓ ?



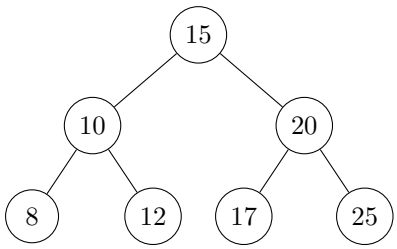
Care dintre următoarele șiruri de noduri corespund traversării arborelui în inordine?

A. H, E, B, F, X, T, A, K, M, R
B. A, E, B, F, T, X, H, K, M, R
C. A, E, B, F, X, T, K, M, R, H
D. A, E, B, F, X, T, H, K, M, R

241. Care este numărul maxim de noduri într-un arbore binar plin de înălțime h? ✓ ?

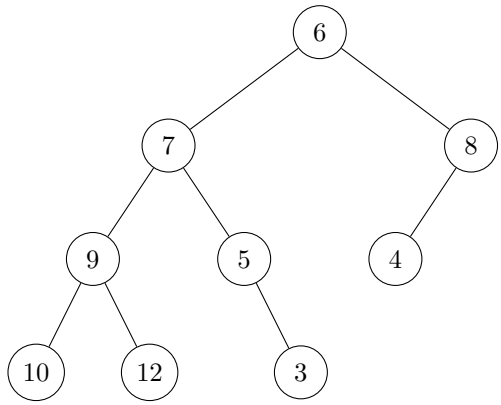
A. 2^h B. $2^{h+1} - 1$ C. $2^h - 1$ D. $h * 2$

242. Într-un arbore binar de căutare se inserează, pe rând, câte un element din mulțimea $\{20, 10, 15, 8, 12, 25, 17\}$. În ce ordine putem insera elementele astfel încât să obținem arborele binar definit alăturat? ✓ ?



A. 15, 10, 20, 8, 12, 17, 25
B. 10, 15, 20, 8, 12, 25, 17
C. 20, 10, 15, 8, 12, 25, 17
D. 15, 20, 10, 12, 8, 17, 25

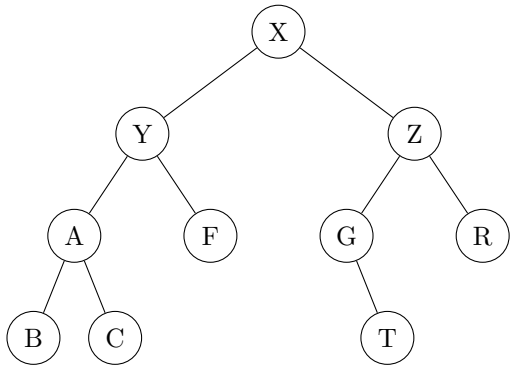
243. Se consideră următorul arbore binar: ✓ ?



Care dintre următoarele șiruri de noduri corespund traversării arborelui în preordine?

A. 7, 9, 10, 12, 5, 3, 6, 8, 4
B. 6, 7, 9, 10, 12, 5, 3, 8, 4
C. 7, 9, 10, 12, 5, 3, 8, 4, 6
D. 6, 7, 9, 12, 10, 5, 3, 8, 4

244. Se consideră următorul arbore binar: ✓ ?



Care dintre următoarele șiruri de noduri corespund traversării arborelui în postordine?

A. X, B, C, A, F, Y, T, G, R, Z
B. B, C, A, F, Y, T, R, G, Z, X
C. B, C, A, F, Y, T, G, R, Z, X
D. X, Y, Z, A, F, G, R, B, C, T

245. În traversarea inordine a unui arbore binar de căutare, ce proprietate este întotdeauna adevărată? ✓ ?

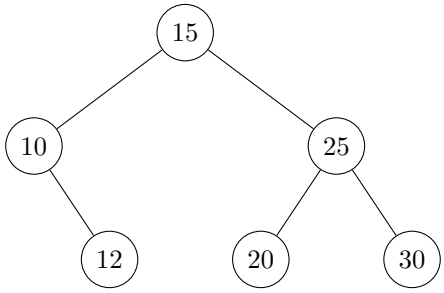
A. Nodurile sunt vizitate în ordinea descrescătoare.
B. Nodurile sunt vizitate în ordinea crescătoare.
C. Nodurile frunze sunt întotdeauna vizitate primele.

D. Nodurile rădăcină sunt vizitate ultimele.

246. Care este înălțimea unui arbore binar complet cu 31 de noduri? ✓ ?

A. 4 B. 5 C. 6 D. 7

247. Într-un arbore binar de căutare, care este succesorul în inordine al nodului cu valoarea 20? ✓ ?



A. 12

B. 25

C. 30

D. 15

248. Dacă traversările preordine și inordine ale unui arbore sunt următoarele: ✓ ?

Preordine: A, B, D, E, C, F
Inordine: D, B, E, A, F, C

Care este structura arborelui binar?

A. A are copiii B și C; B are copiii D și E; C are copil F.

B. A are copiii B și C; B are copiii E și D; C are copil F.

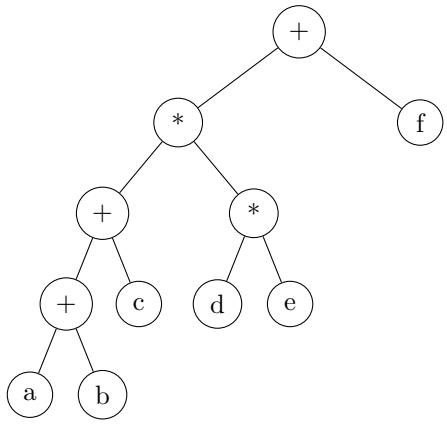
C. A are copiii B și C; C are copiii D și F; B are copil E.

D. A are copiii B și C; B are copil F; C are copiii D și E.

249. Într-un arbore binar complet, care este numărul minim de niveluri pentru a avea cel puțin 50 de noduri? ✓ ?

A. 5 B. 6 C. 7 D. 8

250. Unei expresii matematice îi corespunde un arbore binar în care orice nod care este un nod frunză are ca valoare un operator și are exact doi fii. Nodurile terminale sunt considerate operanzii. Operatorii cu prioritate mai mare sunt plasați pe un nivel mai mare, reflectând ordinea corectă a operațiilor. Parantezele nu sunt prezente în arbore, dar influențează ordinea operațiilor. ✓ ?

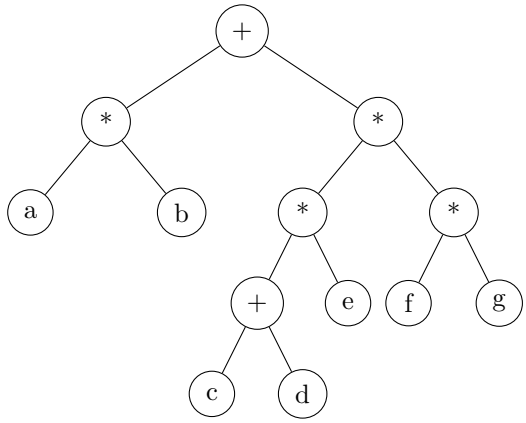


Pentru arborele ilustrat mai sus, determinați expresia corespunzătoare acestuia:

A. $(a+b)+(c*(d*e))+f$
B. $(a+b+c)*(d+e+f)$
C. $(a+b+c)*(d*e)+f$
D. $(a+b)*c+(d+e)*f$

251. Unei expresii matematice îi corespunde un arbore binar care descrie ordinea operațiilor pentru evaluarea acesteia. Fiecare nod care nu este o frunză reprezintă un operator și are exact doi fii. Nodurile frunză sunt operanzii expresiei. Structura arborelui reflectă prioritățile operatorilor și ordinea în care aceștia trebuie evaluați. Evaluarea corectă a expresiei presupune procesarea nodurilor în ordinea în care operațiile devin complet definite.

Fie arborele binar de mai jos, care reprezintă ordinea operațiilor într-o expresie matematică:



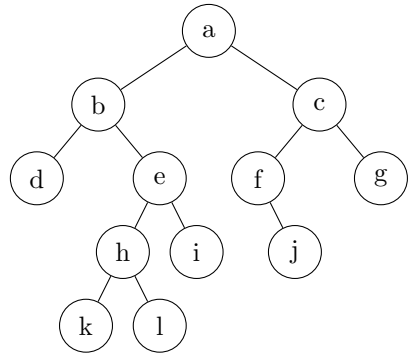
Care dintre următoarele ordini de procesare este corectă pentru evaluarea expresiei $(a*b)+((c+d)*e)*(f*g)$?

A. $ab * cd + e * fg * *+$ C. $cd + e * fg * ab * +$

B. $ab * cd + efg * *+$ D. $ab * efg * cd + *+$

252. Se consideră algoritmul de mai jos, împreună cu arborele binar alăturat acestuia. ✓ ? Pentru un nod *node*, prin intermediul câmpurilor $node.left$ şi $node.right$, poate fi accesat descendentul stâng, respectiv descendentul drept al acestuia.

```
Algorithm ALGORITHM(node)
   If node = null then
      Return -1
   EndIf
   left ← algorithm(node.left)
   right ← algorithm(node.right)
   If left > right then
      max ← left
   Else
      max ← right
   EndIf
   Return 1 + max
EndAlgorithm
```

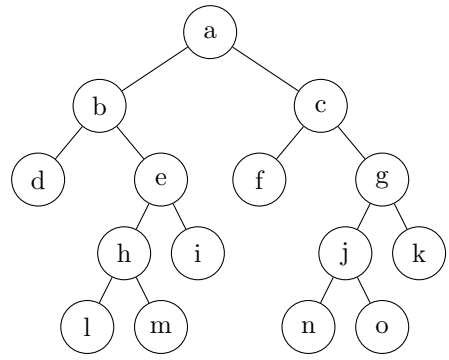


Care este valoarea returnată de algoritm, considerând că apelul acestuia a avut ca parametru rădăcina arborelui?

A. 1 B. 2 C. 3 D. 4

253. Se consideră algoritmul de mai jos, împreună cu arborele binar alăturat acestuia. ✓ ? Pentru un nod *node*, prin intermediul câmpurilor $node.left$ şi $node.right$, poate fi accesat descendentul stâng, respectiv descendentul drept al acestuia.

```
Algorithm ALGORITM(node)
   If node = null then
      Return 0
   EndIf
   count ← 0
   If node.left ≠ null then
      count ← count + 1
      count ← count + algoritm(node.left)
   EndIf
   If node.right ≠ null then
      count ← count + algoritm(node.right)
   EndIf
   Return count
EndAlgorithm
```

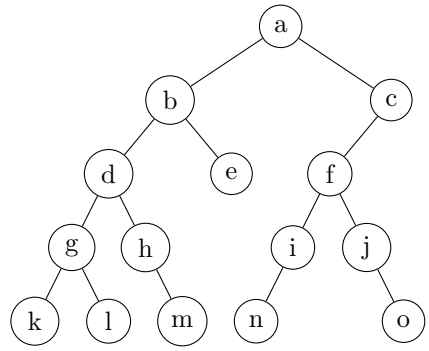


Care este valoarea returnată de algoritm, considerând că apelul acestuia a avut ca parametru rădăcina arborelui?

A. 5 B. 6 C. 7 D. 8

254. Se consideră arborele binar de mai jos, împreună cu algoritmul alăturat acestuia. ✓ ? Pentru un nod *node*, prin intermediul câmpurilor *node.left* şi *node.right*, poate fi accesat descendentul stâng, respectiv descendentul drept al acestuia.

```
Algorithm ALGORITHM(node)
    If node = null then
        Return 0
    EndIf
    count ← 0
    If (node.left ≠ null AND node.right
= null) OR (node.left = null AND
node.right ≠ null) then
        count ← count + 1
    EndIf
    Return count + algorithm(node.left) +
algorithm(node.right)
EndAlgorithm
```



Care este valoarea returnată de algoritm, considerând că apelul acestuia a avut ca parametru rădăcina arborelui?

A. 3 B. 4 C. 5 D. 6

255. Se consideră următorul algoritm $algorithm(node, inf, sup)$. Pentru un nod *node*, ✓ ? prin intermediul câmpurilor $node.left$, *node.right* şi *node.value* poate fi accesat descendentul stâng, descendentul drept, respectiv valoarea acestuia. Ştiind că apelul iniţial are ca parametri rădăcina unui arbore binar pentru *node*, cea mai mică valoare posibilă pentru *inf* şi cea mai mare valoare posibilă pentru *sup*, care dintre afirmaţii sunt adevărate?

```
Algorithm ALGORITM(node, inf, sup)
    If node = null then
        Return true
    EndIf
    If node.value < inf OR node.value > sup then
        Return false
    EndIf
    Return algoritm(node.left, inf, node.value) AND algoritm(node.right, node.value, sup)
EndAlgorithm
```

A. Algoritmul verifică dacă arborele binar dat este complet.

B. Algoritmul verifică dacă descendenţii arborelui binar dat au mereu valori mai mari sau egale cu ascendenţii.

C. Algoritmul verifică dacă arborele binar dat este arbore binar de căutare.

D. Algoritmul verifică dacă în arborele binar dat valoarea oricărui nod este mai mare sau egală cu cea a descendentului stâng, împreună cu toţi descendeţii lui şi mai mică sau egală cu cea a descendentului drept, împreună cu toţi descendenţii lui.

256. Precizați care este numărul ciclurilor hamiltoniene distincte într-un graf neorientat ✓ ? complet cu 5 noduri. Două cicluri sunt distincte dacă diferă prin cel puțin o muchie.

A. $4!/2$ B. $4!$ C. $4^1 \cdot 2 + 4^2$ D. $6!/5! \cdot 2!$

257. Se consideră algoritmul `ceFace()` definit alăturat, unde $a[][]$ este un tablou bidimensional care reprezintă matricea de adiacență a unui graf **orientat** cu n noduri, $n \leq 1000$. Dacă $a[i][j] = 1$, înseamnă că există o muchie de la nodul i la nodul j. $v[]$, $comp[]$ şi $order[]$ sunt tablouri unidimensionale, fiecare cu n elemente, inițial toate 0. Metodele `d1(i, a, v, n, order, idx)` şi `d2(i, t, comp, n, c)` apelate de `ceFace()` sunt, de asemenea, definite mai jos. ✓ ?

```
Algorithm ceFace(a, n)
    idx ← 0
    For i ← 1, n execute
        v[i] ← 0
        comp[i] ← 0
    EndFor
    For i ← 1, n execute
        If v[i] = 0 then
            d1(i, a, v, n, order, idx)
        EndIf
    EndFor
    For i ← 1, n execute
        For j ← 1, n execute
            t[i][j] ← a[j][i]
        EndFor
    EndFor
    c ← 0
    For i ← n, 1, −1 execute
        u ← order[i]
        If comp[u] = 0 then
            c ← c + 1
            d2(u, t, comp, n, c)
        EndIf
    EndFor
EndAlgorithm
```

```
Algorithm d1(i, a, v, n, order, idx)
    v[i] ← 1
    For j ← 1, n execute
        If a[i][j] = 1 & v[j] = 0 then
            d1(j, a, v, n, order, idx)
        EndIf
    EndFor
    idx ← idx + 1
    order[idx] ← i
EndAlgorithm
Algorithm d2(i, t, comp, n, c)
    comp[i] ← c
    For j ← 1, n execute
        If t[i][j] = 1 & comp[j] = 0 then
            d2(j, t, comp, n, c)
        EndIf
    EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

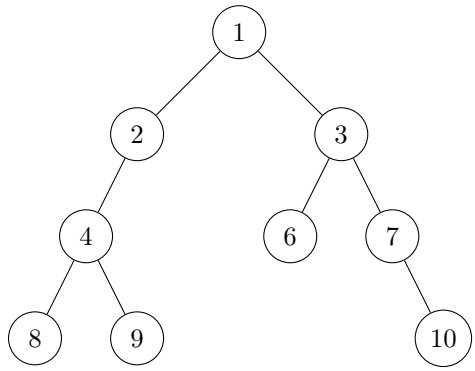
A. Algoritmul `ceFace(a,n)` verifică dacă graful dat este tare conex;

B. La apelul `ceFace(a,n)` se vor parcurge atât graful inițial, cât și graful transpus al grafului inițial;

C. La finalul execuției algoritmului `ceFace(a,n)`, pentru orice nod i, $comp[i]$ va conține numărul de ordine al componentei tare conexe din care i face parte;

D. Algoritmul produce o sortare topologică asupra grafului.

258. Se consideră următorul arbore binar: ✓ ?



Care dintre următoarele afirmații sunt adevărate?

A. Parcurgerea în preordine a arborelui este $1, 2, 4, 8, 9, 3, 6, 7, 10$.

B. Parcurgerea în inordine a arborelui este $8, 4, 9, 2, 1, 6, 3, 7, 10$.

C. Parcurgerea în postordine a arborelui este $8, 9, 4, 2, 6, 10, 7, 3, 1$.

D. Arborele binar nu este complet.

259. Se consideră un arbore binar complet cu 1023 noduri, numerotate de la 1 la 1023, în ordinea apariției de la stânga la dreapta pe fiecare nivel. Considerând că rădăcina se află pe nivelul 0 al arborelui, care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Înălțimea arborelui este 10;

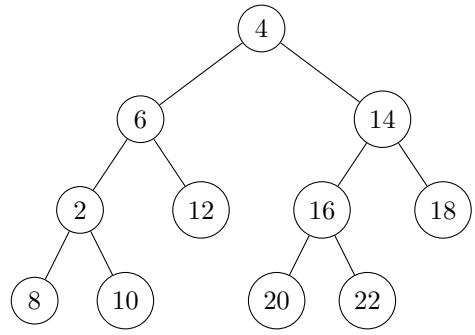
B. Arborele are 512 noduri terminale;

C. Nodul 1000 se află pe nivelul 9 și este fiul stâng al unui alt nod;

D. Nodul 8 este strămoș pentru nodul 768.

260. Se consideră algoritmul `Transform(node, k, s, c)`, unde `node` reprezintă un nod al arborelui, iar k, s și c sunt numere naturale, și arborele binar: ✓ ?

```
Algorithm TRANSFORM(node, k, s, c)
   If node = null then
      s ← 0
      c ← 0
      Return
   EndIf
   left_s ← 0, left_c ← 0
   right_s ← 0, right_c ← 0
   TRANSFORM(node.left, k, left_s,
left_c)
   TRANSFORM(node.right, k, right_s,
right_c)
   If node.left ≠ null  AND
node.right ≠ null then
      If (left_s+right_s)  MOD  k = 0
then
         node.value ← left_c +
right_c
      Else
         node.value ← left_c *
right_c
      EndIf
      s ← left_s + right_s +
node.value
      c ← left_c + right_c + 1
   Else If node.value  MOD  2  =  0
then
      s ← node.value
      c ← 1
   Else
      s ← 0
      c ← 0
   EndIf
EndAlgorithm
```

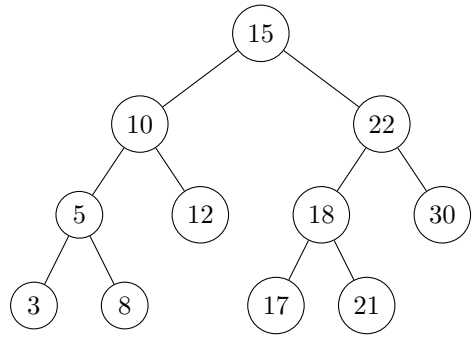


Pentru apelul `Transform(4, 3, 0, 0)`, care va fi valoarea nodului rădăcină după execuția algoritmului?

A. 12 B. 15 C. 20 D. 25

261. Se consideră algoritmul `Algo(node)`, unde `node` reprezintă un nod al arborelui, și arborele binar:

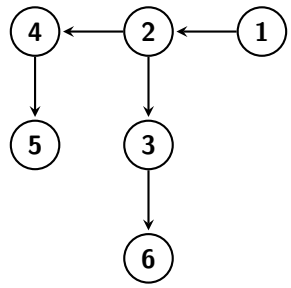
```
Algorithm ALGO(node)
   If node = null then
      Return 0
   EndIf
   If node.stanga ≠ null  AND
node.dreapta ≠ null then
      x ← 0
      temp ← node.stanga
      While temp.dreapta ≠ null
execute
         temp ← temp.dreapta
      EndWhile
      x ← temp.valoare
      y ← 0
      temp ← node.dreapta
      While temp.stanga ≠ null
execute
         temp ← temp.stanga
      EndWhile
      y ← temp.valoare
      If y − x = node.valoare then
         Return 1+
ALGO(node.stanga) +
ALGO(node.dreapta)
      EndIf
   EndIf
   Return ALGO(node.stanga) +
ALGO(node.dreapta)
EndAlgorithm
```



Pentru apelul `Algo(node)`, ce va returna algoritmul, dacă valoarea transmisă ca parametru este nodul rădăcină al arborelui ?

A. 0 B. 3 C. 1 D. 21

262. Se consideră următorul graf orientat cu 6 noduri. ✓ ?

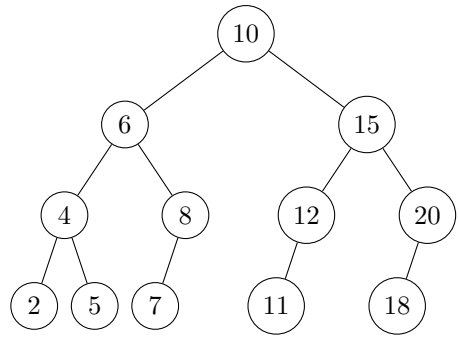


Câte componente tare conexe are acest graf?

A. 2; B. 3; C. 1; D. 6.

263. Se consideră algoritmul `Algo(node)`, unde `node` reprezintă un nod al arborelui, și ✓ ? arborele binar:

```
Algorithm PROCESARE(nod)
    If nod = null then
        Return 0
    EndIf
    nivel_st ← PROCESARE(nod.stanga)
    nivel_dr ←
PROCESARE(nod.dreapta)
    If nivel_st = nivel_dr then
        n ← nivel_st + 1
        p ← (n > 0 AND (n&(n − 1)) = 0)
        If p then
            nod.valoare ← nod.valoare ∗ 2
        EndIf
    EndIf
    Return 1 + max(nivel_st, nivel_dr)
EndAlgorithm
```



Pentru apelul `Algo(node)`, cum va arăta parcurgerea în preordine a arborelui modificat, dacă valoarea transmisă ca parametru este nodul rădăcină al arborelui ?

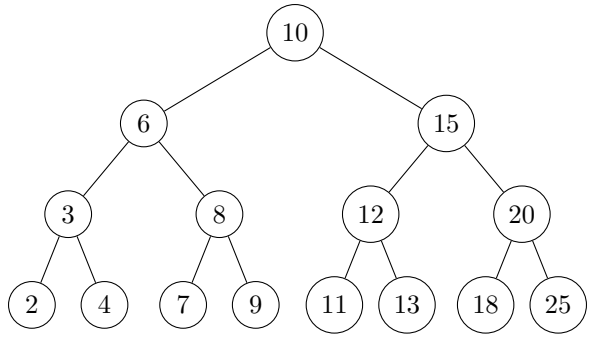
A. 4 8 10 6 14 8 20 22 12 15 36 20

B. 20 6 8 4 10 8 14 15 12 22 20 36

C. 4 10 8 14 8 6 22 12 36 20 15 20

D. 20 6 8 4 10 8 15 14 12 22 20 36

264. Se consideră algoritmul `Algo(nod)`, unde `nod` reprezintă un nod al arborelui binar, ✓ ? iar arborele binar este definit astfel:



```
Algorithm ALGO(nod)
   If nod = null then
      Return 0, 0
   EndIf
   s1, c1 ← ALGO(nod.stanga)
   s2, c2 ← ALGO(nod.dreapta)
   If nod.stanga ≠ null  AND
nod.dreapta ≠ null then
      If CHECK(s1 + s2) then
         nod.valoare ← c1 * c2
      Else
         nod.valoare ← c1 + c2
      EndIf
      Return s1+s2+nod.valoare, c1+
c2 + 1
   Else If nod.valoare > 0 then
      Return nod.valoare, 1
   Else
      Return 0, 0
   EndIf
EndAlgorithm
```

```
Algorithm CHECK(n)
   If n ≤ 1 then
      Return False
   EndIf
   For i ← 2, √n execute
      If n  mod i = 0 then
         Return False
      EndIf
   EndFor
   Return True
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Valoarea finală a nodului rădăcină va fi produsul numărului de noduri din subarborele stâng cu numărul de noduri din subarborele drept

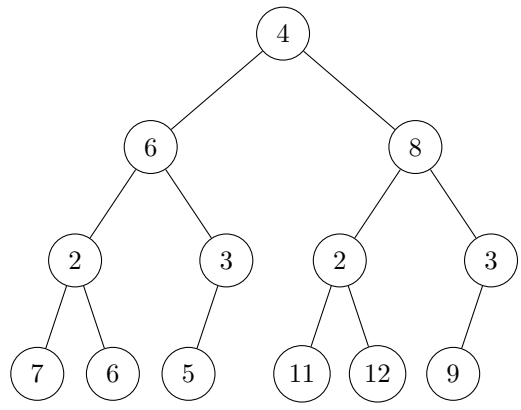
B. Pentru apelul algoritmului `Algo(nod)`, unde *nod* reprezintă rădăcina arborelui, în acest caz, valoarea rădăcinii după procesarea arborelui este 14

C. După execuția algoritmului, orice nod intern a cărui sumă a descendenților e număr prim va avea valoarea egală cu produsul numărului de noduri din subarborii săi

D. Algoritmul modifică valorile tuturor nodurilor interne ale arborelui

265. Se consideră algoritmul `Algo(nod, k, total)`, unde `nod` reprezintă nodul rădăcină ✓ ? al arborelui binar transmis ca parametru, `k` și `total` reprezintă numere naturale, și arborele binar corespunzător:

```
Algorithm ALGO(nod, k, total)
   If nod = null then
      Return 0
   EndIf
   val_st ← ALGO(nod.st, k, total)
   val_dr ← ALGO(nod.dr, k, total)
   If nod.st ≠ null  AND nod.dr ≠
null then
      If (val_st * val_dr)  MOD k =
nod.val  MOD k then
         total ← total + 1
         nod.val ← val_st + val_dr
      EndIf
   EndIf
   Return nod.val
EndAlgorithm
```



Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru apelul `Algo(nod, 4, 0)`, numărul de noduri care își vor schimba valoarea este 4

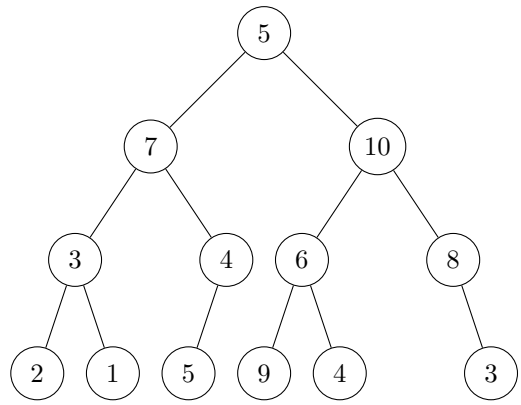
B. Pentru orice arbore binar și $k > 1$, valoarea variabilei total după execuția algoritmului este strict mai mică decât numărul de noduri interne care au ambii copii frunze

C. Dacă doi frați au valori egale, părintele lor își va modifica valoarea dacă și numai dacă valoarea sa inițială modulo k este egală cu pătratul valorii copiilor modulo k

D. Pentru orice k prim, dacă un nod intern își modifică valoarea, atunci cel puțin unul dintre copiii săi trebuie să aibă o valoare divizibilă cu k

266. Se consideră algoritmul `Mister(nod, p, sum, cnt)`, unde `nod` reprezintă un nod al arborelui binar, `p`, `sum`, `cnt` sunt numere naturale, și arborele binar definite astfel: ✓ ?

```
Algorithm MISTER(nod, p, sum, cnt)
    If nod = null then
        sum ← 0, cnt ← 0
        Return
    EndIf
    s_st ← 0, c_st ← 0
    s_dr ← 0, c_dr ← 0
    If nod.stanga ≠ null then
        MISTER(nod.stanga, p, s_st, c_st)
    EndIf
    If nod.dreapta ≠ null then
        MISTER(nod.dreapta, p, s_dr, c_dr)
    EndIf
    If nod.stanga ≠ null AND
nod.dreapta ≠ null then
        If (s_st + s_dr) MOD p = 0 then
            nod.val ← c_st * c_dr
        Else If nod.val MOD p = 0
then
            nod.val ← c_st + c_dr
        EndIf
    EndIf
    sum ← s_st + s_dr + nod.val
    cnt ← c_st + c_dr + 1
EndAlgorithm
```



Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru apelul `Mister(nod, 3, 0, 0)`, valorile finale în urma execuției algoritmului vor fi $sum = 61$ și $cnt = 13$

B. Dacă un nod are suma valorilor din subarbori divizibilă cu p, noua sa valoare va fi strict mai mare decât valoarea sa inițială

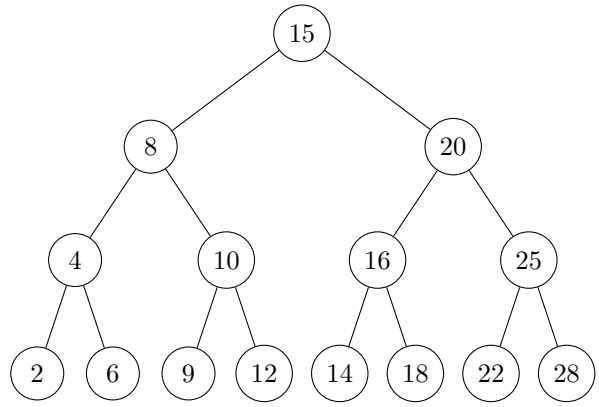
C. Pentru orice p prim mai mare decât maximul din arbore, cel puțin un nod își va modifica valoarea

D. Dacă un nod își modifică valoarea bazat pe a doua condiție (când valoarea sa e divizibilă cu p), noua valoare va fi întotdeauna mai mică decât valoarea inițială

267. Se consideră algoritmul `Algo(nod, dist, val, x)`, unde `nod` reprezintă nodul ✓ ? rădăcină al arborelui binar, `dist`, `val` și `x` reprezintă numere naturale, și arborele binar corespunzător:

```
Algorithm ALGO(nod, dist, val, x)
   If nod = null then
      val ← 0
      x ← 0
      Return
   EndIf
   val_st ← 0, max_st ← 0
   val_dr ← 0, max_dr ← 0
   If nod.st ≠ null then
      ALGO(nod.st, dist, val_st, max_st)
   EndIf
   If nod.dr ≠ null then
      ALGO(nod.dr, dist, val_dr, max_dr)
   EndIf
   If nod.st ≠ null AND nod.dr ≠ null
then

      If (val_st − val_dr ≤
dist) AND (val_dr − val_st ≤ dist)
then
         If max_st > max_dr then
            nod.val ← val_st
         Else
            nod.val ← val_dr
         EndIf
      Else
         min ← 0
         If val_st < val_dr then
            min ← val_st
         Else
            min ← val_dr
         EndIf
         nod.val ← min
      EndIf
      max ← 0
      If max_st > max_dr then
         max ← max_st
      Else
         max ← max_dr
      EndIf
      x ← max + 1
      val ← nod.val
   Else
      val ← nod.val
      x ← 1
   EndIf
EndAlgorithm
```



Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru apelul `Algo(nod, 5, 0, 0)`, valoarea finală a nodului rădăcină după

execuția algoritmului va fi 6.

B. Un nod intern își modifică valoarea cu maximul dintre valorile returnate de subarbori dacă și numai dacă diferența dintre aceste valori nu depășește dist.

C. Pentru orice $dist > 0$, cel puțin un nod intern își va modifica valoarea după execuția algoritmului

D. Un nod care își modifică valoarea va prelua întotdeauna una dintre valorile returnate de copiii săi

268. Se consideră un arbore cu 10 noduri reprezentat prin vectorul de tați `t = (2, 3, 0, 1, 3, 1, 10, 5, 6, 5)`. Care dintre următoarele afirmații sunt adevărate? ✓ ?

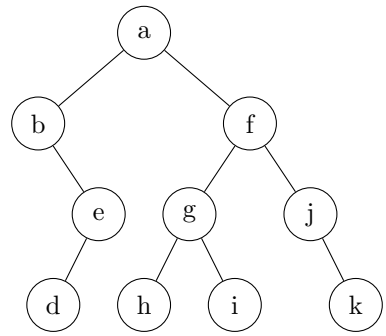
A. Nodurile 4, 7, 8 și 9 sunt noduri frunză.

B. Nodul 2 este rădăcina arborelui

C. Nodul 10 nu are descendenți.

D. Nodurile 8 și 10 sunt frați.

269. Se dă următorul arbore binar. Care dintre următoarele şiruri de noduri reprezintă traversarea acestuia în inordine? ✓ ?



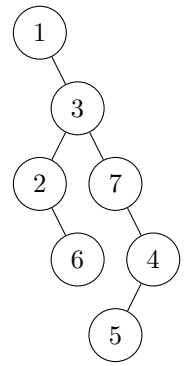
A. b, d, a, e, h, g, i, f, j, k

B. b, d, e, d, f, g, h, i, j, k

C. b, d, e, a, h, g, i, f, j, k

D. a, b, e, d, f, g, h, i, j, k

270. Se dă următorul arbore binar. Care dintre următoarele şiruri de noduri reprezintă traversarea acestuia în postordine? ✓ ?



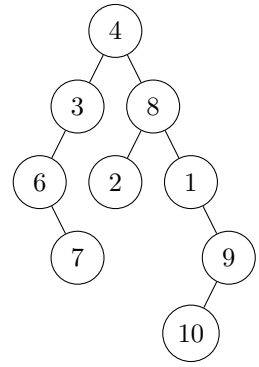
A. $6, 2, 5, 4, 7, 3, 1$

B. $6, 2, 4, 5, 7, 3, 1$

C. $2, 6, 5, 4, 7, 3, 1$

D. $2, 6, 4, 5, 7, 3, 1$

271. Se dă următorul arbore binar. Care dintre următoarele şiruri de noduri reprezintă traversarea acestuia în preordine? ✓ ?



A. $4, 3, 6, 7, 2, 8, 1, 9, 10$

B. $4, 3, 6, 7, 8, 2, 1, 9, 10$

C. $3, 6, 7, 4, 8, 2, 1, 9, 10$

D. $3, 6, 7, 4, 2, 8, 1, 9, 10$

Partea

II

Teste

Admitere nivel licență, sesiunea septembrie 2024

272. Se consideră algoritmul `decide(n, x)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar x este un vector cu n elemente numere întregi $(x[1], x[2], ..., x[n])$, unde $-100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm DECIDE(n, x)
    b ← True
    i ← 1
    While b AND (i < n) execute
        b ← (x[i] < x[i + 1])
        i ← i + 1
    EndWhile
    Return b
EndAlgorithm
```

Pentru care din următoarele situații algoritmul returnează `True`?

A. Dacă vectorul x este format din valorile $1, 2, 3, ..., 10$

B. Dacă vectorul x este strict crescător

C. Dacă vectorul x nu are elemente negative

D. Dacă vectorul x are elementele negative situate înaintea celor pozitive

273. Se consideră algoritmul `afiseaza(n, a)`, unde n este număr natural ($1 \leq n \leq 10^3$), iar a este un vector cu n elemente numere întregi $(a[1], a[2], ..., a[n])$, unde $-100 \leq a[i] \leq 100$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm AFISEAZA(n, a)
    i ← 1; j ← n
    While i ≤ j execute
        If a[i] < a[j] then
            Write a[i], " "
            i ← i + 1
        Else
            Write a[j], " "
            j ← j − 1
        EndIf
    EndWhile
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă vectorul dat este sortat crescător, valorile din vector se afișează în ordine descrescătoare.

B. Dacă vectorul dat este sortat descrescător, ultimul element afișat este elementul maxim.

C. Dacă $n = 10$ și $a = [0, 2, 4, 6, 8, 10, 8, 6, 4, 2]$, valorile din vector se afișează în ordine crescătoare.

D. Dacă elementul maxim este pe prima poziție, valorile din vector se afișează în ordine inversă.

274. Care este relația dintre numerele $X = 6543_{(8)}$ în baza 8 și $Y = CEF_{(16)}$ în baza 16? ✓ ?

A. $X > Y$ B. $X < Y$ C. $X \geq Y$ D. $X = Y$

275. Se consideră algoritmul `f(n)`, unde n este număr natural nenul ($1 \leq n \leq 15$). ✓ ?

```
Algorithm F(n)
    x ← 10; y ← 13
    While n ≠ 0 execute
        z ← (x + y) MOD 2
        n ← n DIV 2
```

```
        If z MOD 2 = 0 then
            x ← (x * 3 + y * 4) MOD z
            y ← (y + x) * z
        Else
            x ← x + 1
            y ← y − 1
        EndIf
    EndWhile
    Return z
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează aceeași valoare pentru orice număr natural $1 \leq n \leq 15$.

B. Algoritmul returnează valori distincte pentru numerele naturale n cu proprietatea $1 \leq n \leq 10$.

C. Dacă $n = 11$, algoritmul returnează 0.

D. Dacă schimbăm instrucțiunea de pe linia 10 cu $x \leftarrow x - 1$, și cea de pe linia 11 cu $y \leftarrow y + 1$ algoritmul returnează aceeași valoare ca în varianta originală pentru orice număr natural $1 \leq n \leq 15$.

276. Se consideră algoritmul `numere(n, x)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar x este un vector cu n elemente numere întregi $(x[1], x[2], ..., x[n])$, unde $-100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm NUMERE(n, x)
    i ← 1; nr ← n
    While i ≤ n execute
        If (x[i] MOD 10) MOD 2
              = 0 then
            nr ← nr + 1
        Else
            nr ← nr − 1
        EndIf
        i ← i + 1
    EndWhile
    Return nr = n
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Apelul `numere(3, [1, 2, 3])` returnează `True`.

B. Apelul `numere(3, [1, -2, 3])` returnează `False`.

C. Apelul `numere(4, [1, 2, 3, -4])` returnează `False`.

D. Apelul `numere(4, [1, 2, 3, 4])` returnează `True`.

277. Se consideră algoritmul `ceFace(v, n)`, unde v este un vector de n ($1 \leq n \leq 10^4$) numere naturale $(v[1], v[2], ..., v[n])$, unde $1 \leq v[i] \leq 10^4$, pentru $i = 1, 2, ..., n$). ✓ ?

```
Algorithm ceFace(v, n)
   a ← 0; b ← 1
   For i ← n, 2, −1 execute
      If v[i] = v[i − 1] + 1
then
         b ← b + 1
      Else
         b ← 1
      EndIf
      If b > a then
         a ← b
      EndIf
   EndFor
   Return a
EndAlgorithm
```

Ce returnează algoritmul `ceFace(v, n)`?

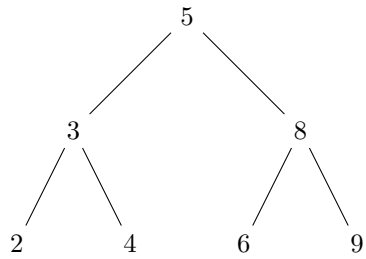
A. Lungimea celei mai lungi subsecvențe formate din numere consecutive crescătoare din vectorul v.

B. Lungimea celei mai lungi subsecvențe formate din numere consecutive descrescătoare din vectorul v.

C. Numărul subsecvențelor crescătoare din vectorul v.

D. Lungimea celei mai lung subșir format din numere consecutive crescătoare din vectorul v.

278. Se consideră următorul arbore binar: ✓ ?



Care dintre următoarele șiruri de noduri corespund traversării arborelui în postordine?

A. 2, 3, 4, 5, 6, 8, 9 B. 4, 3, 2, 9, 8, 6, 5 C. 2, 4, 3, 6, 9, 8, 5 D. 9, 6, 8, 5, 3, 2, 4

279. Se consideră algoritmul `prelucrare(n, m, x)`, unde n și m sunt numere naturale ✓ ? $(1 \leq n \leq 100, 1 \leq m \leq 100)$, iar x este o matrice cu n * m elemente numere naturale $(x[1][1], x[1][2], ..., x[n][m])$, unde inițial $x[i][j] = 0$, pentru $i = 1, 2, ..., n; j = 1, 2, ..., m)$:

```
Algorithm prelucrare(n, m, x)
   k ← 1; i ← k
   While i ≤ n execute
      j ← k + 1
      While j ≤ m execute
         If k MOD 2 = 0 then
            x[i][j] ← k ∗ k
         EndIf
         Write x[i][j], " "
         k ← k + 1; j ← j + 1
      EndWhile
      i ← i + 1
   EndWhile
EndAlgorithm
```

Ce afișează acest algoritm?

A. Un șir de n valori.

B. Dacă m este par, un șir de valori în care valoarea 0 alternează cu valori care reprezintă pătrate perfecte pare, iar prima și ultima valoare sunt 0.

C. Un șir de m - 1 valori.

D. Un șir de valori în care valoarea 0 alternează cu valori care reprezintă pătrate perfecte impare.

280. Se consideră doi vectori de biți, x cu n, și y cu m elemente, unde n și m sunt numere naturale nenule ($0 < n, m \leq 64$). Elementele vectorilor sunt 0 sau 1. Fie *b1* și *b2* doi biți pentru care definim operația $op(b1, b2) = \begin{cases} 1 & \text{dacă } b1 = b2 \\ 0 & \text{dacă } b1 \neq b2 \end{cases}$. Definim operația *os* ca aplicare a operației *op* pe elementele din x și y, dar pornind de la finalul vectorilor (deci prima dată aplicăm *op* pe *x[n]* și *y[m]*). Dacă cei doi vectori au număr diferit de elemente, elementele de la începutul vectorului cu lungime mai mare, care nu au pereche în celălalt vector, rămân nemodificate. De exemplu, pentru vectorii $[1, 1, 0, 1, 0]$ și $[1, 1, 1, 0]$, rezultatul operației *os* va fi $[1, 1, 0, 1, 0, 0]$. Algoritmul creează un vector r cu $\max(n, m)$ elemente.

```
1: Algorithm OperatieSpeciala(x, n, y,
   m)
2:     length ← n
3:     lenF ← m
4:     r ← Zero(m)
5:     If m < n then
6:         length ← m
7:         lenF ← n
8:         r ← Zero(n)
9:     EndIf
10:    For i ← 1, length execute
11:        If (x[i] + y[i]) MOD 2 = 0 then
12:            r[i] ← 0
13:        Else
14:            r[i] ← 1
15:        EndIf
16:    EndFor
17:    For i ← length + 1, m execute
18:        r[i] ← y[i]
19:    EndFor
20:    For i ← length + 1, n execute
21:        r[i] ← x[i]
22:    EndFor
23:    Return r, lenF
24: EndAlgorithm
```

Care dintre afirmațiile de mai jos sunt adevărate?

A. Algoritmul `OperatieSpeciala(x, n, y, m)` implementează corect operația *os* și returnează vectorul rezultat la lungimea lui.

B. Pentru acele date de intrare pentru care instrucțiunile de pe rândul 18 și 21 se execută de același număr de ori, rezultatul returnat este corect.

C. Implementarea ar deveni corectă, dacă înlocuim instrucțiunea de pe rândul 11 cu `If (x[n - i] + y[m - i]) MOD 2 = 0 then`

D. Rezultatul algoritmului `OperatieSpeciala(x, n, y, m)` nu este corect, iar vectorul returnat conține elementele în ordine inversă.

281. Se consideră algoritmul `ceFace(x, m, y, n)`, unde x este un șir de caractere de lungime m ($1 \leq m \leq 100$) iar y un șir de caractere de lungime n ($1 \leq n \leq 100$), astfel încât $m < n$.

```
1: Algorithm ceFace(x, m, y, n)
2:     i ← 1
3:     ok ← True
4:     While ok AND i ≤ m execute
5:         If i ≤ m AND x[i] ≠ y[i] then
6:             ok ← False
7:         Else
8:             i ← i + 1
9:         EndIf
10:    EndWhile
11:    _____
12: EndAlgorithm
```

Ce instrucțiune ar trebui să conțină linia 11, astfel încât algoritmul să returneze `True` în cazul în care șirul x este prefix al șirului y? Exemplu: dacă $x =$ "abc" și $y =$ "abcd", x este prefix al lui y și algoritmul returnează `True`.

A. `Return` $(i = m)$ `OR` $(i = n)$

B. `Return` $i = m$

C. `Return` $i > m$

D. `Return` ok

282. Se consideră o matrice A cu m linii și n coloane $(A[1][1], A[1][2], ..., A[m][n])$, unde m și n sunt numere naturale $(1 < m \leq 25, 1 < n \leq 25)$, și $1 \leq A[i][j] \leq 10^3$, pentru $i = 1, 2, ..., m; j = 1, 2, ..., n$. ✓ ?

Care dintre următorii algoritmi returnează suma elementelor de pe coloana k $(1 < k \leq n)$?

A.
```
Algorithm suma(A, m, n, k)
    s ← 0
    For i ← n, 1, −1 execute
        s ← s + A[i][k]
    EndFor
    Return s
EndAlgorithm
```

B.
```
Algorithm suma(A, m, n, k)
    s ← 0; i ← 1
    While i ≤ m execute
        s ← s + A[i][k]
        i ← i + 1
    EndWhile
    Return s
EndAlgorithm
```

C.
```
Algorithm suma(A, m, n, k)
    s ← 0
    For i ← 1, m execute
        s ← s + A[k][i]
    EndFor
    Return s
EndAlgorithm
```

D.
```
Algorithm suma(A, m, n, k)
    s ← 0; k ← 1
    While k ≤ n execute
        s ← s + A[k][k]
        k ← k + 1
    EndWhile
    Return s
EndAlgorithm
```

283. Se consideră algoritmul `F(n)`, unde n este număr natural nenul $(1 \leq n \leq 10^6)$. Algoritmul `sqrt(n)` returnează radicalul lui n și are complexitatea $O(1)$. Notația $[a]$ reprezintă partea întreagă a valorii lui a. Operatorul "/" reprezintă împărțirea reală, de exemplu: $3 / 2 = 1.5$. ✓ ?

```
Algorithm F(n)
    If n = 1 then
        Return 1
    EndIf
    i ← [n/sqrt(n)]
    Return 1 + F(i)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul `F(n)` are complexitatea timp $O(\log \log n)$.

B. În urma apelului `F(200)` se obține valoarea 4.

C. În urma apelului `F(250)` se obține valoarea 5.

D. Algoritmul `F(n)` are complexitatea timp $O(1)$.

284. Se consideră algoritmul `check(n, x)`, unde n este număr natural $(1 \leq n \leq 10^4)$, iar ✓ ?

x este un vector cu n elemente numere întregi $(x[1], x[2], ..., x[n], -100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n)$:

```
Algorithm CHECK(n, x)
   If n < 3 then
      Return False
   EndIf
   p ← select(n, x)
   If p = 1 OR p = n then
      Return False
   EndIf
   For i ← 2, p execute
      If x[i] ≥ x[i − 1] then
         Return False
      EndIf
   EndFor
   For i ← p + 1, n − 1 execute
      If x[i] ≥ x[i + 1] then
         Return False
      EndIf
   EndFor
   Return True
EndAlgorithm


Algorithm SELECT(n, x)
   r ← 0
   v ← x[1]
   For i ← 2, n execute
      If x[i] < v then
         r ← i
         v ← x[i]
      EndIf
   EndFor
   Return r
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă vectorul x este ordonat descrescător și are cel puțin 3 elemente, algoritmul `check(n, x)` returnează `True`.

B. Dacă vectorul $x = [12, 10, 8, 5, 9, 11, 15, 18]$ și $n = 8$ algoritmul `check(n, x)` returnează `True`.

C. Dacă vectorul $x = [20, 10, 5, 1, 2, 4, 6, 10, 8]$ și $n = 9$ algoritmul `check(n, x)` returnează `False`.

D. Dacă vectorul x este ordonat strict crescător și are cel puțin 3 elemente, algoritmul `check(n, x)` returnează `True`.

285. Se consideră algoritmul `f(x, n)`, unde n este număr natural $(3 \leq n \leq 10^4)$, iar x este un vector de n numere naturale $(x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n)$. Prin $[\,]$ s-a notat un vector vid, iar prin $[a, b]$ s-a notat un vector cu 2 elemente a și b.

```
Algorithm F(x, n)
   If n < 2 then
      Return []
   EndIf
   If n = 2 then
      If x[1] > x[2] then
         Return [x[1], x[2]]
      Else
         Return [x[2], x[1]]
      EndIf
   EndIf
   y ← f(x, n − 1)
   If x[n] > y[1] then
      Return [x[n], y[1]]
   Else
      If x[n] > y[2] then
         Return [y[1], x[n]]
      Else
         Return y
      EndIf
   EndIf
EndAlgorithm
```

Ce va returna algoritmul pentru apelul `f([4, 15, 5, 8, 10, 18, 16, 19, 1, 12], 10)`?

A. $[19, 18]$

B. $[18, 19]$

C. $[16, 19]$

D. $[19, 16]$

286. Se consideră algoritmul `numere(x, n, e)`, unde n este număr natural ($1 \leq n \leq 10^4$), x este un vector cu n elemente numere întregi ($x[1], x[2], ..., x[n], -100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$), iar e este valoarea unui element din vector: ✓ ?

```
Algorithm NUMERE(x, n, e)
   i ← 1
   c ← 0
   b ← True
   If n MOD 2 = 0 then
      Return False
   EndIf
   While (i ≤ n) AND b execute
      If x[i] < e then
         c ← c + 1
      Else
         b ← False
      EndIf
      i ← i + 1
   EndWhile
   Return c = (n − i + 1)
EndAlgorithm
```

În care din următoarele situații returnează algoritmul `True`?

A. Dacă vectorul are număr par de elemente și este ordonat descrescător până la elementul cu valoarea e inclusiv, care se află pe poziția n DIV 2.

B. Dacă vectorul are număr impar de elemente și este ordonat strict crescător până la elementul cu valoarea e inclusiv, care se află pe poziția n DIV 2+1.

C. Dacă vectorul are număr impar de elemente și este ordonat descrescător până la elementul cu valoarea e inclusiv, care se află pe poziția n DIV 2+1.

D. Dacă vectorul are număr impar de elemente, iar valoarea e se găsește pe poziția n DIV 2 + 1 și înainte de e sunt doar valori mai mici, iar după e sunt doar valori mai mari.

287. Care dintre algoritmii următori afișează reprezentarea numărului a în baza b, unde a, b sunt numere naturale date în baza 10 ($1 \leq a \leq 10^4, 2 \leq b \leq 9, a > b$)? ✓ ?

A.
```
Algorithm AFISEAZA(a, b)
   If a ≠ 0 then
      Write a MOD b
      afiseaza(a DIV b, b)
   EndIf
EndAlgorithm
```

B.
```
Algorithm AFISEAZA(a, b)
   If a ≠ 0 then
      afiseaza(a DIV b, b)
      Write a MOD b
   EndIf
EndAlgorithm
```

C.
```
Algorithm AFISEAZA(a, b)
   While a > 0 execute
      Write a MOD b
      a ← a DIV b
   EndWhile
EndAlgorithm
```

D.
```
Algorithm AFISEAZA(a, b)
   nrNou ← 0
   putere ← 1
   While a > 0 execute
      nrNou ← nrNou+
        (a MOD b) * putere
      a ← a DIV b
      putere ← putere * b
   EndWhile
   Write nrNou
EndAlgorithm
```

288. Se consideră algoritmul `f(x, y)`, unde x și y sunt două numere naturale ($1 \leq x \leq 100, 1 \leq y \leq 100$). ✓ ?

```
Algorithm F(x, y)
   If x = y then
      Write "start:  "
   Else
      If x MOD y = 0 then
         f(x + 1, y + 2)
      Else
         If (x DIV y) MOD 2
              = 0 then
            f(x + 2, y + 1)
            Write "*"
         Else
            f(x − 1, y + 1)
            Write "#"
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Ca urmare a apelurilor `f(12, 15)` și `f(8, 12)` nu se afișează același șir de caractere

B. Ca urmare a apelurilor `f(15, 12)` și `f(12, 8)` se afișează același șir de caractere

C. Ca urmare a apelului `f(17, 23)` nu se afișează niciun caracter "#"

D. Ca urmare a apelului `f(23, 17)` șirul de caractere afișat conține cel puțin un caracter "#"

289. Se consideră algoritmul `decide(n, x, t)`, unde x este un vector de n numere naturale ($2 \leq n \leq 10^4, 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$) iar t este un număr natural ($1 \leq t \leq 10^4$). ✓ ?

```
Algorithm DECIDE(n, x, t)
    left ← 1; right ← n
    While x[left] + x[right] ≠ t execute
        If x[left] + x[right] < t then
            left ← left + 1
        Else
            right ← right − 1
        EndIf
    EndWhile
    Return left, right
EndAlgorithm
```

În care dintre următoarele situații algoritmul `decide(n, x, t)` determină indicii *left*, *right* ($1 \leq left < right \leq n$) astfel încât $x[left] + x[right] = t$?

A. Dacă și numai dacă vectorul x conține numere distincte.

B. Dacă și numai dacă vectorul x este ordonat crescător.

C. Dacă vectorul x este ordonat descrescător și conține numere distincte.

D. Dacă vectorul x este ordonat crescător și în vector există cel puțin o pereche de elemente având suma t.

290. Se consideră algoritmul `perechi(x, y)`, unde x și y sunt numere naturale nenule ($1 \leq x, y \leq 100$): ✓ ?

```
Algorithm PERECHI(x, y)
    nr ← 0; d ← 2
    While d ≤ x AND d ≤ y execute
        If (x MOD d = 0) AND
              (y MOD d = 0) then
            nr ← nr + 1
            x ← x DIV d
            y ← y DIV d
        Else
            d ← d + 1
        EndIf
    EndWhile
    Write nr, " ", x, " ", y
EndAlgorithm
```

În care dintre următoarele variante de răspuns avem doar perechi de numere (x, y) pentru care algoritmul `perechi(x, y)` afișează valorile 1 7 11?

A. $(14, 22)$, $(21, 33)$, $(35, 55)$, $(49, 77)$

B. $(7, 11)$, $(14, 22)$, $(21, 33)$, $(28, 44)$

C. $(1, 7)$, $(1, 11)$

D. $(2, 2)$, $(3, 3)$, $(4, 4)$, $(5, 5)$

291. Se consideră algoritmul `first(x, n)`, unde n este un număr natural nenul ($2 \leq n \leq 10^4$), iar x este un vector de n numere naturale ($x[1], x[2], ..., x[n]$, unde $1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$). ✓ ?

```
1: Algorithm FIRST(x, n)
2:     f1 ← False
3:     f2 ← False
4:     For i ← 1, n execute
5:         If x[i] = 1 then
6:             f1 ← True
7:         EndIf
8:         If x[i] = n then
9:             f2 ← True
10:        EndIf
11:        If x[i] ≥ n then
12:            x[i] ← 1
13:        EndIf
14:    EndFor
```

```
15:    If NOT f1 then
16:       Return 1
17:    EndIf
18:    For i ← 1, n execute
19:       _____________
20:    EndFor
21:    If f2 then
22:       x[n] ← n
23:    EndIf
24:    For i ← 1, n execute
25:       If _____________ then
26:          Return i
27:       EndIf
28:    EndFor
29:    Return n + 1
30: EndAlgorithm
```

Cu ce ar trebui înlocuite liniile 19 și 25, astfel încât algoritmul să returneze cel mai mic număr natural nenul care nu se află în vectorul x?

A. **19:** $x[x[i]\ MOD\ (n+1)] \leftarrow x[x[i]\ MOD\ (n+1)] + n$
25: $x[i]\ DIV\ (n+1) = 0$

B. **19:** $x[x[i]\ MOD\ n] \leftarrow x[x[i]\ MOD\ n] + n$
25: $x[i]\ DIV\ n = 0$

C. **19:** $x[x[i]\ MOD\ n] \leftarrow 1$
25: $x[i] = 1$

D. **19:** $x[x[i]\ MOD\ n] \leftarrow x[x[i]\ MOD\ n] + n$
25: $x[i]\ MOD\ n = 0$

292. Se consideră algoritmul `ceFace(arr, n)` unde *arr* este un vector cu n ($1 \leq n \leq 100$) ✓ ? elemente numere întregi ($\text{arr}[1], \text{arr}[2], ..., \text{arr}[n], unde - 10^5 \leq \text{arr}[i] \leq 10^5$, pentru $i = 1, 2, ..., n$).

```
Algorithm ceFace(arr, n)
   sum ← 0
   For i ← 1, n execute
      sum ← sum + arr[i]
   EndFor
   If sum MOD 2 ≠ 0 then
      Return False
   EndIf
   Return
     auxiliar(arr, n, sum DIV 2)
EndAlgorithm
```

```
Algorithm auxiliar(arr, n, sum)
   If sum = 0 then
      Return True
   EndIf
   If n = 1 AND sum ≠ 0 then
      Return False
   EndIf
   If arr[n − 1] > sum then
      Return auxiliar(arr, n - 1, sum)
   EndIf
   Return auxiliar(arr, n - 1, sum) OR
auxiliar(arr, n - 1, sum - arr[n - 1])
EndAlgorithm
```

Care din următoarele afirmații sunt adevărate?

A. Apelul `ceFace([11, 5, 6, 22, 0, 7, 6, 13], 8)` returnează `True`.

B. Apelul `ceFace([-5, -6, -22, -7, -6, -13], 6)` NU returnează `True`.

C. Dacă vectorul *arr* conține doar valori negative, algoritmul `auxiliar(arr, n, sum)` va intra în ciclu infinit.

D. Dacă și numai dacă elementele din *arr* pot fi partiționate în două mulțimi astfel încât media elementelor din cele două mulțimi să fie egală, algoritmul `ceFace(arr, n)` returnează `True`.

293. Se consideră algoritmul `f(n, x)`, unde n este număr natural ($3 \leq n \leq 10^4$), iar x este un vector de n numere naturale ($x[1], x[2], ..., x[n]$, unde $1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm F(n, x)
   s1 ← h(n, x)
   For i ← 1, 2 * n execute
      x[((i + 1) MOD n) + 1] ←
         g(x[(i MOD n) + 1],
         x[((i + 1) MOD n) + 1])
   EndFor
   s2 ← h(n, x)
   Return x[n]
EndAlgorithm
```

```
Algorithm G(a, b)
   If a * b = 0 then
      Return a + b
   EndIf
   If a = b then
      Return a
   EndIf
   If a > b then
      Return g(a − b, b)
   EndIf
   Return g(a, b − a)
EndAlgorithm
```

```
Algorithm H(n, x)
   s ← 0
   For i ← 1, n execute
      s ← s + x[i]
   EndFor
   Return s
EndAlgorithm
```

Care din următoarele afirmații sunt adevărate?

A. În cazul apelului `f(6, [12, 16, 80, 40, 28, 144])` algoritmul returnează valoarea 4.

B. Pentru orice vector de intrare, valoarea *s1* (calculată pe rândul 2 din algoritmul `f(n, x)`) va fi strict mai mare ca valoarea *s2* (calculată pe rândul 6 din algoritmul `f(n, x)`).

C. Dacă în algoritmul `f(n, x)` înlocuim instrucțiunile de de pe liniile 3, 4 și 5 cu secvența de mai jos, la finalul executării algoritmului `f(n, x)` vectorul x va avea același conținut ca în algoritmul original.

```
For j ← 1, 2 execute
   For i ← 1, n − 1 execute
      x[i + 1] ← g(x[i], x[i + 1])
   EndFor
EndFor
```

D. Există vector de intrare cu n elemente pentru care complexitatea timp a algoritmului `f(n, x)` este $O(n)$.

294. Se consideră un număr natural nenul par n ($2 \leq n \leq 12$). Dorim să generăm în șirul de caractere x toate șirurile formate din n paranteze rotunde care se deschid și se închid corect. Algoritmul `paranteze(i, desc, inc, x, n)` se apelează sub forma `paranteze(2, 1, 0, x, n)`, știind că au avut loc inițializările `x[1]` ← `'('` și `x[n]` ← `')'`. Algoritmul `afisare(n, x)` afișează șirul de caractere x de lungime n. ✓ ?

```
1: Algorithm PARANTEZE(i, desc, inc,
   x, n)
2:    If i = n then
3:       afisare(n, x)
4:    Else
5:       If _____ then
6:          x[i] ← '('
7:          paranteze(i + 1, desc +
   1, inc, x, n)
8:       EndIf
9:       If _____ then
10:         x[i] ← ')'
11:         paranteze(i + 1, desc,
   inc + 1, x, n)
12:      EndIf
13:   EndIf
14: EndAlgorithm
```

Cu ce ar trebui completate liniile precizate mai jos, astfel încât algoritmul să afișeze doar șirurile de caractere corecte conform cerinței?

A. Linia 5 trebuie completată cu `desc < n`, iar linia 9 trebuie completată cu `inc < desc`

B. Linia 5 trebuie completată cu `desc < n DIV 2`, iar linia 9 trebuie completată cu `inc < desc`

C. Linia 5 trebuie completată cu `desc < n`, iar linia 9 trebuie completată cu `inc < n DIV 2`

D. Indiferent de comparațiile cu care se completează liniile 5 și 9, algoritmul nu va afișa toate șirurile de caractere conforme cerinței.

295. La ora de educație fizică n copii stau unul lângă altul, cu fața către profesorul lor când acesta le cere să se întoarcă toți la stânga. Încă o dată când se întorc cu fața către profesor. Într-o unitate de timp, toți copiii se vor întoarce într-o direcție. Dacă un copil se întoarce la stânga, el va face acest lucru (întorcându-se cu 180°), fiecare copil făcând câte un întoarce. Mișcarea copiilor continuă până nu mai sunt copii situați față în față. Precizăm că în urma acestui proces, algoritmul `intoarceri(n, c)` determină numărul unităților de timp t care trebuie până când nu mai sunt copii situați față în față. Variabila n este număr natural nenul ($1 \leq n \leq 100$), iar șirul de caractere c are n elemente, unde `c[i]` este fie `'s'` (reprezentând *stânga*), fie `'d'` (reprezentând *dreapta*) în funcție de direcția în care se întorc copiii după comanda profesorului. Exemplu: dacă $n = 6$ și $c =$ "sdsssd" atunci $t = 3$; dacă $n = 3$ și $c =$ "sdd" atunci $t = 0$. Algoritmul `copiza(c, n)` returnează o copie a vectorului c cu n elemente.

Care din următoarele algoritmi `intoarceri(n, c)` determină numărul unităților de timp t corect?

A.

```
Algorithm INTOARCERI(n, c)
   t ← 0; aux ← copiza(c, n); ok ← False
   While NOT ok execute
      ok ← True
      For i ← 1, n − 1 execute
         If (aux[i] =' d') AND (aux[i + 1] =' s') then
            c[i] ←' s'; c[i + 1] ←' d'
            ok ← False
         EndIf
      EndFor
      aux ← copiza(c, n)
      If NOT ok then
         t ← t + 1
      EndIf
```

```
      EndWhile
      Return t
   EndAlgorithm
```

B.

```
   Algorithm INTOARCERI(n, c)
      stop ← False; t ← 0
      While NOT stop execute
         stop ← True
         i ← 1
         While i < n execute
            If (c[i] =' d') AND (c[i + 1] =' s') then
               c[i] ←' s'; c[i + 1] ←' d'
               i ← i + 2
               stop ← False
            Else
               i ← i + 1
            EndIf
         EndWhile
         If NOT stop then
            t ← t + 1
         EndIf
      EndWhile
      Return t
   EndAlgorithm
```

C.

```
   Algorithm INTOARCERI(n, c)
      t ← 0; dr ← 0; st ← 0
      For i ← 1, n execute
         If c[i] =' d' then
            If dr > 0 then
               t ← t + st
               st ← st + 1
            Else
               dr ← dr + 1
            EndIf
         Else
            If st > 0 then
               t ← t + dr
               dr ← dr + 1
            Else
               st ← st + 1
            EndIf
         EndIf
      EndFor
      Return t
   EndAlgorithm
```

D.

```
   Algorithm INTOARCERI(n, c)
      t ← 0; dr ← 0; st ← 0
      For i ← 1, n execute
         If c[i] =' d' then
            If st > 0 then
               t ← t + st
               st ← st − 1
            Else
```

```
            dr ← dr + 1
         EndIf
      Else
         If dr > 0 then
            t ← t + dr
            dr ← dr − 1
         Else
            st ← st + 1
         EndIf
      EndIf
   EndFor
   Return t
EndAlgorithm
```

Admitere nivel licență, sesiunea iulie 2024

296. Se consideră algoritmul `ceFace(A, m, n)`, unde m este număr natural ($1 \leq m \leq 100$), iar A este un vector cu m elemente numere întregi ($A[1], A[2], ..., A[m], -10^5 \leq A[i] \leq 10^5$, pentru $i = 1, 2, ..., m$), iar n este un număr natural ($n \leq m$): ✓ ?

```
Algorithm ceFace(A, m, n)
   For i ← 1, n execute
      min_idx ← i
      For j ← i + 1, m execute
         If A[min_idx] > A[j] then
            min_idx ← j
         EndIf
      EndFor
      aux ← A[i]
      A[i] ← A[min_idx]
      A[min_idx] ← aux
   EndFor
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $n = m$, atunci după executarea algoritmului `ceFace(A, m, n)` elementele vectorului vor fi ordonate crescător.

B. Dacă $n = m$, atunci după executarea algoritmului `ceFace(A, m, n)` elementele vectorului vor fi ordonate descrescător.

C. Dacă $A = [4, 64, 1, 25, 12, 22, 2, 11]$ $n = 2$ și $m = 8$, după executarea algoritmului `ceFace(A, m, n)` cel puțin primele 3 elemente din vectorul A vor fi ordonate crescător.

D. Dacă $n < m$, după executarea algoritmului `ceFace(A, m, n)` cel puțin primele $n+1$ elemente din vectorul A vor fi ordonate crescător.

297. Se consideră algoritmul `h(n, a)`, unde n este un număr natural ($1 \leq n \leq 10^3$) și a este un vector cu n elemente numere întregi ($a[1], a[2], ..., a[n]$), unde $-100 \leq a[i] \leq 100$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm h(n, a)
   If n = 1 then
      Return a[n]
   Else
      If a[n] > a[n − 1] then
         a[n − 1] ← a[n] − a[n − 1]
      Else
         a[n − 1] ← a[n] + a[n − 1]
      EndIf
      Return h(n − 1, a)
   EndIf
EndAlgorithm
```

Pentru ce valori ale numărului n și a vectorului a apelul `h(n, a)` va returna valoarea 1?

A. $n = 6, a = [1, 2, 3, 4, 5, 6]$

B. $n = 6, a = [6, 5, 4, 3, 2, 1]$

C. $n = 5, a = [1, 5, 4, 2, 3]$

D. $n = 2, a = [1, 2]$

298. Se consideră expresia ✓ ?

$$E = (x \textbf{ MOD } 3 = 0) \textbf{ OR } ((y < x) \textbf{ OR } \textbf{ NOT } ((y * 3) \textbf{ MOD } 7 \leq 3))$$

Care este valoarea expresiei, dacă $x = 10$ și $y = 41$?

A. True

B. False

C. Aceeași valoare ca expresia E1, unde E1 = **NOT** $((y$ **MOD** $3 = 0)$ **OR** $((x < y)$ **OR NOT** $((x * 3)$ **MOD** $7 \leq 3)))$

D. Aceeași valoare ca expresia E2, unde E2 = $(x$ **MOD** $3 = 0)$ **OR** $((x < y)$ **AND** $((y * x)$ **MOD** $3 \leq 7))$

299. Ion implementează următorul algoritm pentru a verifica dacă numărul natural nr $(0 < nr < 10^6)$ este prim. ✓ ?

```
Algorithm PRIM(nr)
   If nr < 2 then
      Return False
   EndIf
   If (nr > 2)  AND
        (nr  MOD  2 = 0) then
      Return False
   EndIf
   d ← 3
   While d * d < nr execute
      If nr  MOD  d = 0 then
         Return False
      EndIf
      d ← d + 2
   EndWhile
   Return True
EndAlgorithm
```

Ion testează corectitudinea algoritmului pe numerele din mulțimea $M = \{2, 3, 4, 5, 10, 11, 13\}$. Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul este corect și returnează rezultat corect atât pentru numerele din M, cât și pentru orice alt număr conform specificațiilor.

B. Algoritmul este incorect, dar returnează rezultat corect pentru numerele din M.

C. Algoritmul este incorect, și returnează rezultat incorect pentru toate numerele din M.

D. Algoritmul este incorect, dar returnează rezultat corect pentru cel puțin un număr din M și rezultat incorect pentru cel puțin un alt număr din M.

300. Se consideră algoritmul $f(n, x)$, unde n este număr natural $(1 \leq n \leq 10^4)$, iar x este un vector cu n elemente numere întregi $(x[1], x[2], ..., x[n], -200 \leq x[i] \leq 200$, pentru $i = 1, 2, ..., n)$: ✓ ?

```
Algorithm F(n, x)
   a ← True
   i ← 1
   While a AND
        (i < n) execute
      a ← (x[i] > x[i + 1])
      i ← i + 1
   EndWhile
   Return a
EndAlgorithm
```

Pentru care din următoarele date de intrare algoritmul `f(n, x)` returnează True?

A. Pentru orice vector care conține elementele pozitive urmate de elementele negative

B. Pentru orice vector strict descrescător

C. Pentru orice vector care nu conține elemente pozitive

D. Pentru vectorul $x = [5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5]$ și $n = 11$

301. Fie expresia $E = AB_{(16)} + 120_{(3)} - 120_{(4)}$, unde notația $x_{(b)}$ semnifică numărul x scris în baza b. ✓ ?

Care valoare corespunde expresiei E?

A. $162_{(10)}$ B. $278_{(8)}$ C. $1000101_{(2)}$ D. $242_{(8)}$

302. Se consideră algoritmul f(a, b), unde a și b sunt numere naturale nenule ($0 < a, b < 10^4$). ✓ ?

```
Algorithm F(a, b)
   If a = 0 then
      Return b
   EndIf
   x ← f(a − 1, b + 1)
   Return f(a − 1, x − 2)
EndAlgorithm
```

Care este cel mai mic număr natural a pentru care în urma apelului `f(a, 15)` algoritmul returnează un număr strict negativ?

A. 3 B. 4 C. 5 D. 6

303. Se consideră algoritmul compute(n), unde n este număr natural ($1 \leq n \leq 10^4$). ✓ ?

```
Algorithm COMPUTE(n)
   x ← 0
   While n > 0 execute
      If n MOD 2 = 1 then
         x ← x + 1
      EndIf
      n ← n DIV 2
   EndWhile
   Return x
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă n este impar, algoritmul compute(n) returnează o valoare mai mare decât 1.

B. Algoritmul compute(n) returnează suma cifrelor din reprezentarea lui n în baza 2.

C. Algoritmul compute(n) returnează numărul divizorilor impari (proprii și improprii) ai numărului natural n.

D. Algoritmul compute(n) returnează numărul de biți 1 din reprezentarea lui n în baza 2.

304. Se consideră algoritmul f(p, q, r), unde p, q și r sunt valori booleene: ✓ ?

```
Algorithm F(p, q, r)
   While (p AND (NOT r))
         OR (NOT q) execute
      Write (q AND (p OR r))
      p ← NOT p
      r ← q OR p
   EndWhile
EndAlgorithm
```

Care din următoarele afirmații sunt adevărate pentru apelul `f(True, False, True)`?

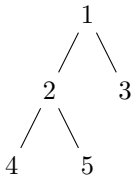
A. Algoritmul intră în ciclu infinit, afișând False în mod repetat.

B. Algoritmul nu afișează nimic.

C. Algoritmul afișează valoarea False o singură dată.

D. Algoritmul afișează valorile False True False.

305. Se consideră următorul arbore binar:



Care dintre următoarele șiruri de noduri corespund traversării arborelui în preordine? ✓ ?

A. 1, 2, 4, 5, 3

B. 4, 2, 5, 1, 3

C. 1, 2, 3, 4, 5

D. 4, 5, 2, 3, 1

306. Se consideră algoritmul mark(n, m, a), unde n și m sunt numere naturale nenule ($1 \leq n, m \leq 10$), iar a este un vector de numere naturale cu n elemente (a[1], a[2], ..., a[n]). Algoritmul tuple(i, j, k), unde i, j și k sunt numere naturale nenule ($1 \leq i, j, k \leq 10$) returnează True sau False. ✓ ?

```
Algorithm MARK(n, m, a)
    a[1] ← 1
    For i ← 2, n execute
        a[i] ← 0
    EndFor
    ready ← False
    While NOT ready execute
        ready ← True
        For i ← 1, n execute
            For j ← 1, n execute
                For s ← 1, m execute
                    If a[i] = 1 AND
                          tuple(i, s, j) AND
                          a[j] = 0 then
                        a[j] ← 1
                        ready ← False
                    EndIf
                EndFor
            EndFor
        EndFor
    EndWhile
EndAlgorithm
```

Presupunem că pentru toate tripletele de mai jos algoritmul tuple(i, j, k) returnează True. Pentru care perechi de triplete va fi efectul apelului mark(3, 3, a) acela de setare a tuturor elementelor vectorului a la valoarea 1?

A. (1, 1, 2) și (2, 2, 3)

B. (1, 1, 2) și (3, 2, 2)

C. (1, 2, 2) și (1, 3, 3)

D. (1, 2, 2) și (3, 3, 1)

307. Se consideră o matrice mat cu n linii și n coloane ($1 \leq n \leq 200, mat[1][1], \ldots, mat[1][n], mat[2][1], \ldots, mat[2][n], \ldots, mat[n][1], \ldots, mat[n][n]$) și algoritmul matrice(mat, n). ✓ ?

```
Algorithm MATRICE(mat, n)
    k ← 1
    For i ← 1, n execute
        For j ← 1, n execute
            mat[i][j] ← k
            k ← k * (−1)
        EndFor
    EndFor
    Return mat
EndAlgorithm
```

Care din afirmațiile de mai jos sunt adevărate pentru matricea returnată în urma apelului matrice(mat, n)?

A. Dacă $n = 31$, produsul elementelor de pe diagonala principală este 1.

B. Dacă $n = 32$, produsul elementelor de pe prima linie este 1.

C. Dacă $n = 127$, elementul de pe ultima linie și ultima coloană este -1.

D. Dacă $n = 128$, suma elementelor de pe prima coloană este 1.

308. Se consideră algoritmul modifica(n, a), unde n este număr natural ($1 \leq n \leq 10^3$), iar a este un vector cu n elemente numere întregi $(a[1], a[2], ..., a[n], -100 \leq a[i] \leq 100, i = 1, ..., n)$: ✓ ?

```
Algorithm MODIFICA(n, a)
    x ← a[n]
    i ← 0
    For j ← 1, n − 1 execute
        If a[j] ≤ x then
            i ← i + 1
            t ← a[i]
            a[i] ← a[j]
            a[j] ← t
        EndIf
    EndFor
    t ← a[i + 1]
    a[i + 1] ← a[n]
    a[n] ← t
    Return a
EndAlgorithm
```

Care din afirmațiile de mai jos sunt adevărate?

A. Dacă vectorul a este sortat crescător, acesta va rămâne sortat crescător la terminarea executării algoritmului.

B. Dacă vectorul a este sortat strict descrescător, atunci în vectorul returnat de algoritm elementul maxim va fi pe ultima poziție.

C. În vectorul returnat de algoritm, elementul maxim va fi întotdeauna pe ultima poziție.

D. Dacă $n = 100$, iar elementele din vectorul a au proprietatea că $a[i] = i$ **MOD** 2, pentru $i = 1, 2, ..., n$, atunci la terminarea executării algoritmului vectorul va fi sortat crescător.

309. Se consideră algoritmul f(v, n), unde n este număr natural ($2 \leq n \leq 10^4$) și v este un vector cu n numere naturale $(v[1], v[2], ..., v[n], 1 \leq v[i] \leq 10^3$, pentru $i = 1, 2, ..., n)$. ✓ ?

```
Algorithm F(v, n)
    a ← 0; b ← 0; i ← 1
    While i < n execute
        If v[i] mod 3 = 0 then
            a ← a + v[i]
            b ← b + 1
        EndIf
        i ← i + 1
    EndWhile
    If b = 0 then
        Return 0
    EndIf
    i ← 0
    While a ≥ b execute
        a ← a − b
        i ← i + 1
    EndWhile
    Return i
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează media aritmetică a elementelor care sunt multiplii de 3 din vectorul v sau 0 dacă vectorul nu conține multiplii de 3.

B. Algoritmul returnează cel mai mare divizor comun al elementelor care sunt multiplii de 3 din vectorul v sau 0 dacă vectorul nu conține multiplii de 3.

C. Algoritmul returnează numărul elementelor multiplii de 3 din vectorul v sau 0 dacă vectorul nu conține multiplii de 3.

D. Niciunul dintre răspunsurile A., B., C nu este adevărat.

310. Pentru a determina toate submulțimile mulțimii $A = \{4, 8, 9, 12, 15\}$ cu 5 elemente, un elev a scris algoritmul generare(i, n, x, A). Mulțimea este reprezentată prin vectorul A cu n elemente numere naturale. Submulțimile generate se afișează cu ajutorul algoritmului afis(m, x, A), x fiind un vector auxiliar indexat de la 0 iar m un număr natural reprezentând lungimea vectorului x curent. Înainte de apelul generare(1, 5, x, A) elementul x[0] a fost inițializat cu 0.

```
Algorithm GENERARE(i, n, x, A)
    For j ← n, x[i − 1] + 1 execute
        x[i] ← j
        AFIS(i, x, A)
        GENERARE(i + 1, n, x, A)
    EndFor
EndAlgorithm
```

```
Algorithm AFIS(m, x, A)
    Write ‘‘{’’, A[x[1]]
    For i ← 2 to m execute
        Write ‘‘,’’, A[x[i]]
    EndFor
    Write ‘‘}’’, newline
EndAlgorithm
```

Știind că primele 4 submulțimi afișate sunt, în această ordine: $\{15\}$, $\{12\}$, $\{12, 15\}$, $\{9\}$ care va fi a 8-a submulțime generată (submulțimea vidă nu se ia în considerare)?

A. $\{9, 12\}$ B. $\{8\}$ C. $\{9, 12, 15\}$ D. $\{8, 15\}$

311. Se consideră algoritmul f(x, n, k) unde n și k sunt numere naturale ($3 \leq n \leq 10^4, 1 \leq k \leq 10^4$), iar x este un vector de n numere naturale ($x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$):

```
Algorithm F(x, n, k)
    If k > n then
        Return 0
    EndIf
    For i ← 1, n − 1 execute
        x[i + 1] ← x[i + 1] + x[i]
    EndFor
    Return x[k]
EndAlgorithm
```

Pentru care din următoarele apeluri algoritmul va returna valoarea 10?

A. `f([1, 4, 6], 3, 3)`

B. `f([1, 2, 3, 4, 5], 5, 3)`

C. `f([1, 2, 3, 4], 4, 4)`

D. `f([10, 15, 25], 3, 1)`

312. Se consideră algoritmul `decide(n)`, unde n este număr natural ($10^4 \leq n \leq 10^7$): ✓ ?

```
Algorithm DECIDE(n)
    m ← 10
    abc ← n DIV m
    While abc ≥ 1000 execute
        m ← m * 10
        abc ← n DIV m
    EndWhile
    bc ← abc MOD 100
    f ← (bc < 2)
    i ← 2
    While i ≤ bc DIV 2 execute
        If bc MOD i = 0 then
            f ← True
            i ← bc
        EndIf
        i ← i + 1
    EndWhile
    Return f
EndAlgorithm
```

Pentru care din următoarele apeluri algoritmul va returna True?

A. decide(865756)

B. decide(72387)

C. decide(103983)

D. decide(10405)

313. Se consideră algoritmul ceFace(n), unde n este număr natural nenul ($1 \leq n < 10^3$). ✓ ?

```
Algorithm CEFACE(n)
    Return CEFACERECURSIV(n, 1, 1)
EndAlgorithm

Algorithm CEFACERECURSIV(n, a, b)
    If n = 0 then
        Return 1
    Else
        If n < 0  OR  b > n then
            Return 0
        Else
            Return CEFACERECURSIV(n, a + b, a) +
                   CEFACERECURSIV(n - a, a + b, a)
        EndIf
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În intervalul [11, 16] există o singură valoare x, pentru care algoritmul ceFace(x) returnează 1.

B. Pentru orice număr n, algoritmul ceFace(n) va returna valoarea 0 sau 1.

C. Algoritmul ceFace(n) returnează numărul de moduri de a scrie numărul n ca sumă de numere consecutive.

D. Algoritmul ceFace(n) returnează numărul de mulțimi diferite ale căror elemente sunt numere Fibonacci diferite de 0 și care au suma egală cu n.

314. Se consideră algoritmul ceFace(x, n), unde n este număr natural ($1 \leq n \leq 10^4$), x este un vector cu n elemente cifre ($x[1], x[2], ..., x[n], 1 \leq x[i] \leq 9$, pentru $i = 1, 2, ..., n$), iar algoritmul Zero(k) returnează un vector cu k elemente, toate egale cu zero: ✓ ?

```
Algorithm ceFace(x, n)
    f ← Zero(9)
    For i ← 1, n execute
        f[x[i]] ← f[x[i]] + 1
    EndFor
    i ← 9
    nr ← 0
    While i > 0 execute
        If f[i] = 0 then
            nr ← nr * 10 + i
        EndIf
        i ← i − 1
    EndWhile
    Return 10 * nr
EndAlgorithm
```

Ce returnează algoritmul dat?

A. Un număr format din cifrele vectorului x

B. Un număr format din cifrele vectorului x, luată fiecare cifră o singură dată

C. Cel mai mare număr posibil de format din cifre distincte care nu apar în vectorul x

D. Cel mai mic număr posibil de format din cifre distincte care nu apar în vectorul x

315. Se consideră numerele naturale nenule n și m, $(1 \leq n, m \leq 100)$ și matricea `matrix` cu n linii și m coloane, elementele ei fiind 0 sau 1. Se consideră algoritmii `prelucrare(matrix, row, col, n, m)` și `num(matrix, n, m)`, unde `row` și `col` sunt numere naturale $(1 \leq row \leq n, 1 \leq col \leq m)$.

```
Algorithm prelucrare(matrix, row, col, n, m)
    If row ≥ 1 AND row ≤ n AND col ≥ 1 AND col ≤ m
            AND matrix[row][col] = 1 then
        matrix[row][col] ← 0
        prelucrare(matrix, row - 1, col, n, m)
        prelucrare(matrix, row + 1, col, n, m)
        prelucrare(matrix, row, col - 1, n, m)
        prelucrare(matrix, row, col + 1, n, m)
    EndIf
EndAlgorithm

Algorithm num(matrix, n, m)
    c ← 0
    For row ← 1, n execute
        For col ← 1, m execute
            If matrix[row][col] = 1 then
                c ← c + 1
                prelucrare(matrix, row, col, n, m)
            EndIf
        EndFor
    EndFor
    Return c
EndAlgorithm
```

Considerând că o insulă este formată din elemente identice vecine pe orizontală sau pe verticală, care dintre următoarele afirmații sunt adevărate?

A. Dacă $n \neq m$ algoritmul num(matrix, n, m) nu verifică toate elementele din matrice.

B. Pentru matricea cu 5 linii și 5 coloane:

```
matrix =
1 1 0 0 0
1 1 0 0 0
```

```
0 0 1 0 0
0 0 0 1 1
0 0 0 1 1
```

apelul num(matrix, 5, 5) returnează 3.

C. Algoritmul num(matrix, n, m) returnează numărul de insule formate din 0 în matricea dată.

D. Algoritmul num(matrix, n, m) returnează numărul de insule formate din 1 în matricea dată.

316. Se consideră două șiruri de caractere r și s de lungimea Lung ($1 \leq Lung \leq 256$). Se ✓ ? consideră următorii algoritmi:

(a) Algoritmul copiere(a, primul, ultimul) returnează șirul de caractere format din elementele șirului de caractere a, începând cu poziția primul până la poziția ultimul inclusiv.

(b) Algoritmul egale(a, b, k) returnează True, dacă șirurile de caractere a și b, ambele de lungime k, sunt identice, și False în caz contrar.

(c) Algoritmul lungime(a) returnează lungimea șirului de caractere a.

(d) Algoritmul concatenare(a, b) returnează șirul de caractere obținut prin concatenarea șirului a cu șirul b, în această ordine.

Precizați care dintre următorii algoritmi returnează valoarea True dacă șirul de caractere r se poate obține prin rotirea de 0, 1, sau de mai multe ori a șirului s. De exemplu, șirul de caractere "abcde" poate fi obținut prin rotirea șirului "cdeab".

A.

```
Algorithm CHECK(s, r, Lung)
   For i ← 1, Lung execute
      If EGALE(s, r, Lung) then
         Return True
      EndIf
      aux ← s[1]
      For j ← 2, Lung execute
         s[j − 1] ← s[j]
      EndFor
      s[Lung] ← aux
   EndFor
   Return False
EndAlgorithm
```

B.

```
Algorithm CHECK(s, r, Lung)
   ss ← CONCATENARE(s, s)
   i ← 1
   sf ← Lung + 1
   While i ≤ sf execute
      k ← i
      j ← 1
      While j ≤ Lung AND ss[k] =
r[j] execute
         j ← j + 1
         k ← k + 1
      EndWhile
      If j > Lung then
         Return True
      EndIf
      i ← i + 1
   EndWhile
   Return False
EndAlgorithm
```

C.

```
Algorithm CHECK(s, r, Lung)
    ss ← CONCATENARE(r, s)
    i ← 1
    While i ≤ Lung execute
        k ← i
        j ← 1
        While j ≤ Lung  AND  ss[k] =
r[j] execute
            j ← j + 1
            k ← k + 1
        EndWhile
        If j > Lung then
            Return True
        EndIf
        i ← i + 1
    EndWhile
    Return False
EndAlgorithm
```

D.

```
Algorithm CHECK(s, r, Lung)
    pos1 ← 1
    ok ← False
    While r[pos1] ≠ s[1] execute
        pos1 ← pos1 + 1
    EndWhile
    If pos1 > 0 then
        ok ← EGALE(s, r, Lung)
    EndIf
    If NOT ok then
        pos2 ← Lung − pos1 + 1
        ok ← (r[1] = s[pos2])
        ss ← COPIERE(s, pos2, Lung)
        rr ← COPIERE(r, 1, pos1)
        ok ← ok  AND  EGALE(rr, ss,
lungime(ss))
    EndIf
    Return ok
EndAlgorithm
```

317. Se consideră algoritmul ceFace(a, n) unde n este număr natural ($2 < n \leq 10^4$) și a este un vector cu n numere naturale ($a[1], a[2], ..., a[n], 0 \leq a[i] \leq 10^4$ pentru $i = 1, 2, ..., n$). Considerăm algoritmul nrPalindromuri(b, p, r), unde b este un vector de m numere naturale ($b[1], b[2], ..., b[m], 0 \leq b[j] \leq 10^4$ pentru $j = 1, 2, ..., m, 2 < m < 10^4$). Parametrii p și r sunt numere naturale astfel încât $1 \leq p < r \leq m$. Algoritmul nrPalindromuri(b, p, r) returnează numărul de numere palindrom din subsecvența b[p], ..., b[r] a vectorului b.

```
Algorithm CEFACE(a, n)
    b ← 0; c ← b; e ← 0; d ← 0
    For i ← 1, n − 2 execute
        If NRPALINDROMURI(a, i, i + 2) > 1 then
            If c = 0 then
                d ← i
            EndIf
            c ← c + 1
        Else
            If c > b then
                b ← c; e ← d
            EndIf
            c ← 0
        EndIf
    EndFor
    If c > b then
        b ← c; e ← d
    EndIf
    If b = 0 then
        Write 0, " ", 0
    Else
        Write e, " ", e + b + 1
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă în cazul unui vector a de lungime 10^4 se afișează 7381 7384, rezultă că printre cele 4 numere situate în vector în intervalul de poziții [7381, ..., 7384] există exact două numere palindrom.

B. Dacă $n = 12$ și $a = [11, 33, 45, 103, 121, 343, 33, 99, 100, 22, 44, 45]$ algoritmul ceFace(a, n) afișează: 5 8

C. Dacă la terminarea executării algoritmului valoarea lui b este 0, rezultă că în vectorul a nu există niciun număr palindrom.

D. Dacă $n = 12$ și $a = [11, 33, 45, 103, 121, 343, 33, 99, 100, 22, 44, 45]$ algoritmul ceFace(a, n) afișează: 4 12

318. Se consideră algoritmul fun(a, b, len), unde len este un număr natural ($1 \leq len \leq 100$), iar a și b sunt doi vectori având aceeași lungime len ($a[1], a[2], ..., a[len], b[1], b[2], ..., b[len], 1 \leq a[i], b[i] \leq len, i = 1, 2, ..., len$). ✓ ?

```
Algorithm FUN(a, b, len)
   For i ← 1, len execute
      k ← a[b[i]]
      a[b[i]] ← b[a[i]]
      b[a[i]] ← k
   EndFor
EndAlgorithm
```

Fie $len = 7$, $a = [6, 2, 5, 4, 1, 3, 4]$ și $b = [1, 2, 3, 5, 6, 4, 4]$. În cei doi vectori înainte de executarea algoritmului `fun(a, b, len)` există câte două elemente având aceeași valoare, situate pe poziții identice ($a[2] = b[2]$ și $a[7] = b[7]$). Care din următoarele afirmații sunt adevărate în urma apelului `fun(a, b, len)`?

A. Vectorii a și b vor avea elemente identice pe pozițiile 3 și 6.

B. Vectorii a și b vor avea câte trei elemente având aceeași valoare, situate pe poziții identice.

C. Vectorul b va avea valorile: [1, 2, 3, 4, 6, 5, 4].

D. Vectorul a va avea valorile: [4, 2, 6, 3, 6, 1, 4].

319. Se consideră algoritmul `calculeaza(v, b, n, i)`, unde b, n, i sunt numere naturale nenule ($1 \leq b, n, i \leq 10^3$), iar v este un vector cu n elemente numere naturale ($v[1], v[2], ..., v[n], 0 \leq v[i] \leq 10^3$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm CALCULEAZA(v, b, n, i)
   If b = 0 then
      Return True
   EndIf
   If i = n then
      Return False
   EndIf
   Return
     CALCULEAZA(v, b - v[i], n, i + 1)
OR  CALCULEAZA(v, b, n, i + 1)
EndAlgorithm
```

Pentru care din următoarele date de intrare algoritmul returnează **True**?

A. $v = [3, 1, 7, 4, 2]$, $b = 10$, $n = 5$, $i = 1$

B. $v = [2, 6, 4, 8, 12]$, $b = 12$, $n = 5$, $i = 1$

C. $v = [3, 1, 7, 4, 2]$, $b = 10$, $n = 5$, $i = 2$

D. $v = [2, 6, 4, 8, 12]$, $b = 12$, $n = 5$, $i = 3$

Admitere nivel licență, sesiunea septembrie 2023

320. Se consideră algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale ($0 \leq a, b \leq 10^4$): ✓ ?

```
Algorithm ceFACE(a, b)
    c ← 0
    bc ← b
    While bc ≠ 0 execute
        c ← c * 10 + bc MOD 10
        bc ← bc DIV 10
    EndWhile
    If c ≠ a then
        Return ceFace(a − 1, b − 1)
    EndIf
    Return a
EndAlgorithm
```

Care este efectul apelului `ceFace(a, a)`?

A. Algoritmul returnează cel mai mic palindrom mai mare sau egal cu a.

B. Algoritmul returnează cel mai mare palindrom mai mic sau egal cu a.

C. Algoritmul returnează cel mai mic palindrom mai mare decât a.

D. Algoritmul returnează cel mai mare număr par mai mic sau egal cu a.

321. Se consideră algoritmul `creareTablou(n, m, x)`, unde n, m sunt numere naturale ($1 \leq n, m \leq 100$), iar x este un tablou bidimensional cu $n * m$ elemente numere întregi ($x[1][1], x[1][2], ..., x[n][m], 0 \leq x[i][j] \leq 10^4$, pentru $i = 1, 2, ..., n; j = 1, 2, ..., m$): ✓ ?

```
Algorithm CREARETABLOU(n, m, x)
    k ← 0
    For i ← 1, n execute
        For j ← 1, m execute
            If k MOD 2 ≠ 0 then
                x[i][j] ← k * k
            EndIf
            Write x[i][j], " "
            k ← k + 1
        EndFor
        Write new line
    EndFor
EndAlgorithm
```

Ce afișează acest algoritm dacă elementele tabloului x sunt inițializate cu 0?

A. Algoritmul afișează elementele tabloului bidimensional x, în care se află valori egale cu 0 și primele $(n * m)$ **DIV** 2 pătrate perfecte impare.

B. Algoritmul afișează elementele tabloului bidimensional x, în care se află valori egale cu 0 și primele pătrate perfecte pare.

C. Algoritmul afișează elementele tabloului bidimensional x, în care se află șirul primelor $(n * m)$ **DIV** 2 pătrate perfecte pare.

D. Algoritmul afișează elementele tabloului bidimensional x, în care - dacă am așeza elementele linie după linie - pătratele perfecte impare ar apărea în ordine crescătoare, eventual precedate și/sau urmate de valori egale cu 0.

322. Se consideră algoritmul `something(n, x)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar x este un vector de n numere naturale ($x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10^6$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm SOMETHING(n, x)
   s ← 0
   For i ← 1, n execute
      nr ← 1
      While x[i] > 9 execute
         nr ← nr + 1
         x[i] ← x[i] DIV 10
      EndWhile
      s ← s + nr
   EndFor
   Return s
EndAlgorithm
```

Ce returnează apelul `something(5, [222, 2043, 29, 2, 20035])`?

A. 16

B. 10

C. 11

D. 15

323. Fie algoritmul `ceFace(n, v, a)`, unde n și v sunt două numere naturale ($1 \leq n, v \leq 10^4$), iar a este un șir de numere naturale cu n elemente $(a[1], a[2], ..., a[n])$: ✓ ?

```
Algorithm CEFACE(n, v, a)
   For i ← 1, n execute
      d ← v
      If a[i] ≠ 0 then
         gasit ← False
         While (d ≤ v * a[i])  AND  ( NOT  gasit) execute
            If ((d DIV a[i]) * a[i] = d)  AND  ((d DIV v) * v = d) then
               gasit ← True
            Else
               d ← d + 1
            EndIf
         EndWhile
      EndIf
      v ← d
   EndFor
   Return v
EndAlgorithm
```

Care este valoarea returnată de algoritm, dacă $n = 4$, $v = 3$ și $a = [5, 4, 2, 10]$?

A. 20 B. 120 C. 60 D. 15

324. Se consideră algoritmul `calcul(v, n)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar v este un vector cu n elemente numere naturale $(v[1], v[2], ..., v[n], 1 \leq v[i] \leq 10^4$, pentru $i = 1, 2, ..., n)$: ✓ ?

```
Algorithm CALCUL(v, n)
   i ← 1
   While i ≤ n DIV 2 execute
      p ← 0
      While v[i] ≠ 0 execute
         p ← p + 1
         v[i] ← v[i] DIV 10
      EndWhile
      q ← 0
      While v[n + 1 − i] ≠ 0 execute
         q ← q + 1
         v[n + 1 − i] ← v[n + 1 − i] DIV 10
      EndWhile
      If p ≠ q then
         Return False
      EndIf
      i ← i + 1
   EndWhile
   Return True
EndAlgorithm
```

În care din următoarele situații algoritmul returnează *True*?

A. Dacă vectorul v este format din valorile $[12, 12, 2, 5466, 3, 111, 1, 3, 44]$ și $n = 9$.

B. Dacă vectorul v este format din valorile $[12, 345, 2, 5466, 3, 111, 10]$ și $n = 7$.

C. Dacă elementele vectorului v au același număr de cifre.

D. Dacă vectorul format din numărul cifrelor elementelor vectorului v formează un palindrom; de exemplu, din $v = [8, 37, 3]$ se formează vectorul $[1, 2, 1]$, care este palindrom.

325. Se consideră algoritmul `alg(n)`, unde n este număr natural ($0 \leq n \leq 10^4$): ✓ ?

```
Algorithm ALG(n)
   If n = 0 then
      Return 0
   Else
      If n MOD 2 = 0 then
         Return alg(n DIV 10)      +
n MOD 10
      Else
         Return alg(n DIV 10)
      EndIf
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Apelul `alg(123)` returnează 6.

B. Algoritmul calculează suma cifrelor aflate pe pozitii pare ale numărului dat.

C. Algoritmul calculează suma cifrelor pare ale numărului dat.

D. Algoritmul calculează suma cifrelor numărului dat.

326. Se consideră algoritmul `f(x)`, unde x este un număr natural nenul ($1 \leq x \leq 10^5$): ✓ ?

```
Algorithm F(x)
   If x > 0 then
      x ← x DIV 2
      f(x)
      Write x, " "
      x ← x DIV 2
      f(x)
   EndIf
EndAlgorithm
```

Precizați ce se afișează în urma apelului `f(10)`.

A. 0 1 2 0 5 0 1

B. 0 1 2 5 1 0

C. 1 2 1 5 2 1

D. 1 2 1 1 5 1 2

327. Se consideră matricea pătrată M de dimensiune n care conține numere naturale, ✓ ?

unde n este număr natural nenul ($1 \leq n \leq 10^4$, $M[1][1], ..., M[1][n], M[2][1], ..., M[2][n], ..., M[n][1], ..., M[n][n], 1 \leq M[i][j] \leq 10^4$, pentru $i = 1, 2, ..., n, j = 1, 2, ..., n$). Se consideră următorul algoritm:

```
Algorithm WHAT(M, n)
   up ← 1
   down ← n
   left ← 1
   right ← n
   While left ≤ right AND up ≤ down execute
      For i ← left, right execute
         Write M[up][i], " "
      EndFor
      up ← up + 1
      For i ← up, down execute
         Write M[i][right], " "
      EndFor
      right ← right − 1
      For i ← right, left, −1 execute
         Write M[down][i], " "
      EndFor
      down ← down − 1
      For i ← down, up, −1 execute
         Write M[i][left], " "
      EndFor
      left ← left + 1
   EndWhile
EndAlgorithm
```

Ce se afișează pentru următoarea matrice M?

1	2	3
8	9	4
7	6	5

✓ ?

A. 1 2 3 4 9 8 7 6 5

B. 1 2 3 4 5 6 7 8 9

C. 1 2 3 4 5 8 9 7 6

D. 1 8 7 6 5 4 3 2 9

328. Fie algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10^4$): ✓ ?

```
Algorithm CEFACE(a, b)
   If a = 1 then
      Return 1
   Else
      If a MOD b = 0 then
         Return ceFface(a DIV b, b)
      Else
         Return 0
      EndIf
   EndIf
EndAlgorithm
```

Precizați afirmațiile adevărate:

A. În cazul apelului `ceFace(1, 2)` algoritmul returnează 1.

B. În cazul apelului `ceFace(24, 2)` algoritmul returnează 0.

C. În cazul apelului `ceFace(2024, 4)` algoritmul returnează 4.

D. În cazul apelului `ceFace(8, 3)` algoritmul returnează 2.

329. Fie algoritmii `decide(n)` și `compute(m)`, unde n și m sunt numere naturale nenule ($1 \leq n, m \leq 10^4$): ✓ ?

```
Algorithm DECIDE(n)
    result ← −1
    m ← 0
    While n > 0 execute
        m ← m * 10 + n MOD 10
        n ← n DIV 10
    EndWhile
    If m MOD 3 = 0 then
        result ← 1
    EndIf
    Return result
EndAlgorithm

Algorithm COMPUTE(m)
    cnt ← 0
    For k ← 0, m − 1 execute
        cnt ← cnt + decide(k)
    EndFor
    Return cnt
EndAlgorithm
```

Pentru ce valori ale lui m algoritmul `compute(m)` va returna -33?

A. 100

B. 99

C. 98

D. 101

330. Se consideră algoritmul `f(n, x)`, unde n și x sunt numere naturale ($1 \leq n \leq 10^5, 2 \leq x \leq 10$): ✓ ?

```
Algorithm F(n, x)
    If n > 0 then
        f(n DIV x, x)
        Write n MOD x
    EndIf
EndAlgorithm
```

Care din următoarele afirmații sunt adevărate?

A. Algoritmul afișează reprezentarea numărului n în baza de numerație x.

B. Algoritmul afișează restul împărțirii întregi a numărului x la numărul n.

C. Algoritmul afișează numărul de cifre al reprezentării în baza x a numărului n.

D. Algoritmul verifică dacă numărul n este divizibil cu x.

331. Se consideră algoritmul `ceFace(n)`, unde n este număr natural ($1 \leq n \leq 10^9$): ✓ ?

```
Algorithm CEFACE(n)
    If n ≤ 9 then
        If n MOD 2 = 0 then
            Return n
        Else
            Return −1
        EndIf
    EndIf
    x ← n MOD 10
    y ← ceFace(n DIV 10)
    If x MOD 2 ≠ 0 then
        Return y
    EndIf
    If x > y then
        Return x
```

```
    EndIf
    Return y
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează un număr format dintr-o singură cifră, sau -1.

B. Algoritmul returnează un număr impar.

C. Algoritmul returnează cifra impară maximă a numărului n, sau -1.

D. Algoritmul returnează cifra pară maximă a numărului n, sau -1.

332. Se consideră algoritmul `decide(n, x)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar x este un vector cu n elemente numere întregi ($x[1], x[2], ..., x[n], -100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm DECIDE(n, x)
    b ← True
    i ← 1
    While b = True  AND
            i < n execute
        If x[i] < x[i + 1] then
            b ← True
        Else
            b ← False
        EndIf
        i ← i + 1
    EndWhile
    Return b
EndAlgorithm
```

În care din următoarele situații algoritmul returnează *True*?

A. Dacă vectorul $x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ și $n = 10$

B. Dacă $n > 1$ și elementele vectorului x sunt în ordine strict crescătoare

C. Dacă vectorul x nu are elemente negative

D. Dacă vectorul x are elemente pozitive situate înaintea celor negative

333. Fie x și y două numere naturale pozitive cu proprietățile: x este putere a lui 2 și y este multiplu de 3. Fie expresia logică $((x{*}y{+}3)$ **DIV** $6 = 10)$ **OR** $((x{*}y)$ **MOD** $6 = 0)$ **AND** $((x + y)$ **MOD** $4 = 0))$ Care dintre următoarele afirmații sunt adevărate pentru perechi de numere care respectă proprietățile din enunț: ✓ ?

A. Există o pereche (x, y) pentru care expresia este adevărată.

B. Există o pereche (x, y) pentru care expresia este falsă.

C. Există perechile (x_1, y_1) și (x_2, y_2), cu $x_1 \neq x_2$ și $y_1 \neq y_2$ în așa fel încât expresia este adevărată pentru ambele perechi.

D. Expresia este falsă pentru orice pereche (x, y).

334. Se consideră două numere naturale n și m ($1 \leq n, m \leq 256$) respectiv șirurile de caractere a, având n caractere ($a[1], a[2], ..., a[n]$) și șirul b având m caractere ($b[1], b[2], ..., b[m]$). Care dintre următorii algoritmi returnează *True* dacă șirul a poate fi format pornind de la șirul b prin eliminarea unor caractere, fără a modifica poziția relativă a caracterelor rămase, și *False* în caz contrar. De exemplu, șirul "ace" poate fi format prin eliminarea de caractere din șirul "abcde", dar șirul "aec" nu poate fi obținut prin acest procedeu. ✓ ?

A.

```
Algorithm hasProperty(a, b, n, m)
    If n = 0 then
        Return True
    EndIf
    If m = 0 then
        Return False
    EndIf
    If a[n] = b[m] then
        Return hasProperty(a, b, n − 1, m − 1)
    EndIf
    Return hasProperty(a, b, n, m − 1)
EndAlgorithm
```

B.

```
Algorithm hasProperty(a, b, n, m)
    i ← 1
    j ← 1
    While i ≤ n AND j ≤ m execute
        If a[i] = b[j] then
            i ← i + 1
        EndIf
        j ← j + 1
    EndWhile
    If i > n then
        Return True
    Else
        Return False
    EndIf
EndAlgorithm
```

C.

```
Algorithm hasProperty(a, b, n, m)
    i ← n
    j ← m
    While i ∗ j > 0 execute
        If a[i] = b[j] then
            i ← i − 1
        EndIf
        j ← j − 1
    EndWhile
    If i = 0 then
        Return True
    Else
        Return False
    EndIf
EndAlgorithm
```

D.

```
Algorithm hasProperty(a, b, n, m)
    If n > m then
        Return False
    EndIf
    i ← 1
    j ← 1
    While i < n execute
        If a[i] = b[j] then
            i ← i + 1
        EndIf
        j ← j + 1
    EndWhile
    If i > m then
        Return True
    Else
        Return False
    EndIf
EndAlgorithm
```

335. Se consideră algoritmul `ceva(x, n, e)`, unde x este un vector cu n elemente distincte întregi $(x[1], x[2], ..., x[n], 1 \leq n \leq 10^3$ și $x[i] \neq x[j]$, pentru $1 \leq i < j \leq n)$ și e este un număr întreg. Algoritmul caută elementul e în vectorul x, și dacă îl găsește, mută elementul pe prima poziție din vector și returnează *True*, nemodificând ordinea relativă a celorlalte elemente. Dacă e nu se găsește în x, algoritmul returnează *False* și nu modifică conținutul vectorului. De exemplu, pentru vectorul x cu elementele $[-100, 2, 71, 31, -62, 51]$ și $e = 31$, algoritmul va returna *True* și vectorul x va deveni $[31, -100, 2, 71, -62, 51]$. Care dintre următoarele variante este o implementare corectă pentru algoritmul `ceva(x, n, e)` și are complexitate timp $O(n)$? ✓ ?

A.

```
Algorithm CEVA(x, n, e)
    index ← 1
    While index ≤ n execute
        If x[index] = e then
            tmp ← x[index]
            x[index] ← x[1]
            x[1] ← tmp
            Return True
        EndIf
        index ← index + 1
    EndWhile
    Return False
EndAlgorithm
```

B.

```
Algorithm CEVA(x, n, e)
    index ← 2
    tmp ← x[1]
    While index ≤ n execute
        If x[index] = e then
            x[1] ← e
            x[index] ← tmp
            Return True
        EndIf
        tmp2 ← x[index]
        x[index] ← tmp
        tmp ← tmp2
        index ← index + 1
    EndWhile
    Return False
EndAlgorithm
```

C.

```
Algorithm CEVA(x, n, e)
    index ← n
    While index > 1 execute
        If x[index] = e then
            index2 ← index
            While index2 > 1
execute
                x[index2] ←
                    x[index2 − 1]
                index2 ← index2 − 1
            EndWhile
            x[index2] ← e
        EndIf
        index ← index − 1
    EndWhile
    If x[1] = e then
        Return True
    Else
        Return False
    EndIf
EndAlgorithm
```

D. Niciuna dintre variantele A, B, C

336. Se consideră algoritmul `expresie(x, y, z)`, unde x, y, z sunt numere naturale $(0 \leq x, y, z \leq 10^4)$:

```
Algorithm EXPRESIE(x, y, z)
    If x = 0 then
        Return z
    Else
        Return expresie(x − 1, y, x ∗ x +
y ∗ y + z)
    EndIf
EndAlgorithm
```

Precizați expresia a cărei valoare o calculează și returnează algoritmul:

A. $\sum_{i=1}^{x} i^2 + \sum_{i=1}^{y} x * y + \sum_{k=1}^{z} 1$

B. $\sum_{i=1}^{x} i^2 + \sum_{j=1}^{y} j^2 + z$

C. $\sum_{i=1}^{x} i^2 + x * y^2 + z$

D. $\sum_{i=1}^{x} i^2 + \sum_{j=1}^{y} j^2 + \sum_{k=1}^{z} k$

337. Se consideră algoritmul `ceFace(v, a, b)`, unde v este un vector cu n elemente din mulțimea $0, 1$, $(1 \leq n \leq 10^4, v[1], ..., v[n])$, iar a și b sunt numere naturale nenule. Vectorul v este ordonat crescător.

```
Algorithm ceFace(v, a, b)
   If b − a + 1 = 0 then
      Return 0
   EndIf
   If v[a] = 1 then
      Return b − a + 1
   EndIf
   If v[b] = 0 then
      Return 0
   EndIf
   c ← (a + b) DIV 2
   Return ceFace(v, a, c)+
        ceFace(v, c + 1, b)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate, considerând că apelul inițial este `ceFace(v, 1, n)`?

A. Dacă vectorul v conține cel puțin o valoare de 1, atunci se returnează lungimea vectorului.

B. Dacă vectorul v conține doar valori de 1, atunci se returnează valoarea lui n.

C. Dacă vectorul v conține doar valori de 0, atunci se returnează 0.

D. Se returnează numărul de valori 1 conținute de vectorul v.

338. Se știe că numărul total de șiruri binare (care conțin doar caracterele 0 și 1) de lungime n este 2^n. De exemplu, pentru $n = 2$ acestea sunt 00, 01, 10 și 11, numărul lor fiind $2^2 = 4$. Șirul 100011 are lungimea 6 și conține ca subsecvență toate cele 4 șiruri posibile de lungime $n = 2$, fiindcă începând cu prima poziție apare 10, începând cu a doua poziție apare 00, începând cu a patra poziție apare 01 și începând cu a cincea poziție apare 11. Care este lungimea minimă a unui șir, care conține ca subsecvență toate cele 2^n șiruri binare posibile pentru $n = 4$?

A. 18 B. 19 C. 20 D. 21

339. Se consideră algoritmul `t(q, x, y)`, unde q este un caracter oarecare, iar x și y sunt numere naturale nenule ($1 \leq x, y \leq 100$):

```
Algorithm t(q, x, y)
   If x ≤ y then
      Write q
   Else
      If x MOD y = 0 then
         t(q, x + 1, y − 2)
      Else
         If (x DIV y) MOD
                2 ≠ 0 then
            t(q, x − 1, y + 2)
            Write 'c'
         Else
            t(q, x − 1, y − 1)
            Write "cc"
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. În urma apelurilor `t('c', 33, 28)`, `t('c', 10, 6)` și `t('c', 22, 16)` se afișează aceleași caractere.

B. În urma apelurilor `t('c', 33, 28)` și `t('c', 45, 40)` nu se afișează aceleași caractere.

C. În urma apelului `t('c', 11, 8)` se afișează "cc".

D. În urma apelului `t('c', 25, 16)` nu se afișează "ccccc".

340. Se consideră algoritmul `hIndex(x, n)`, unde x este un vector cu n ($1 \leq n \leq 10^5$) elemente numere naturale nenule ($x[1], x[2], ..., x[n]$). Definim *h-index*-ul vectorului x, ca fiind cea mai mare valoare v pentru care este adevărat că există cel puțin v valori în

x care sunt mai mari sau egale cu v. De exemplu, pentru $x = [3, 10, 2, 7, 10, 8, 50, 1, 1, 5]$ h-*index*-ul este 5.

```
1: Algorithm hIndex(x, n)
2:     h ← 1
3:     cont ← True
4:     While cont = True AND h ≤ n
   execute
5:         pos ← h
6:         For i ← h + 1, n execute
7:             If x[i] > x[pos] then
8:                 pos ← i
9:             EndIf
10:        EndFor
11:        If pos ≠ h then
12:            tmp ← x[pos]
13:            x[pos] ← x[h]
14:            x[h] ← tmp
15:        EndIf
16:        If x[h] ≥ h then
17:            h ← h + 1
18:        Else
19:            cont ← False
20:        EndIf
21:    EndWhile
22:    ...
23: EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În momentul în care s-ar executa linia 22 vectorul x este sortat descrescător.

B. Algoritmul `hIndex(x, n)` returnează h-*index*-ul vectorului x dacă pe linia 22 scriem instrucțiunea `Return h`.

C. Algoritmul `hIndex(x, n)` returnează h-*index*-ul vectorului x dacă pe linia 22 scriem instrucțiunea `Return h - 1`.

D. Dacă algoritmul `hIndex(x, n)` se apelează pentru un vector x sortat strict descrescător, atunci algoritmul nu returnează h-*index*-ul vectorului x, indiferent ce instrucțiune adăugăm pe linia 22.

341. Se consideră algoritmul `ceFace(n, k, x, p)`, unde n, k și p sunt numere naturale nenule ($1 \leq n, k, p \leq 10, p \leq n$), iar x este un vector cu $p + 1$ elemente numere naturale $(x[0], x[1], ..., x[p])$. Presupunem că $x[0]$ este inițializat cu 0. ✓ ?

```
Algorithm ceFace(n, k, x, p)
   If k > p then
      For i ← 1, p execute
         Write x[i]
      EndFor
      Write " "  ▷ un singur spațiu
   Else
      For i ← x[k − 1] + 1, n execute
         x[k] ← i
         ceFace(n, k + 1, x, p)
      EndFor
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele variante de răspuns sunt corecte.

A. După ce algoritmul se apelează sub forma `ceFace(3, 1, x, 3)` acesta se va mai autoapela de 6 ori.

B. Dacă $x[0]$ se inițializează cu o valoare diferită de 0, în urma apelului `ceFace(5, 1, x, 3)` numărul de spații afișate este diferit de 10.

C. Dacă algoritmul se apelează sub forma `ceFace(5, 1, x, 4)` se afișează numerele 1245 1234 1345 1235 2345, dar în altă ordine.

D. Dacă algoritmul se apelează sub forma `ceFace(5, 1, x, 3)` rezultatul afișat este 123 124 125 134 135 145 234 235 în această ordine.

342. Se consideră algoritmul `f(sir, s, d, p)`, unde *sir* este un șir de caractere, iar *s, d, p* sunt numere naturale nenule ($0 < s, d, p < 10^6$). Operatorul "+" reprezintă operatorul de concatenare a două șiruri de caractere. Algoritmul `print(a)` afișează șirul de caractere *a*, apoi trece la o linie nouă. ✓ ?

```
1: Algorithm F(sir, s, d, p)
2:     If s = p  AND  d = p then
3:         print(sir)
4:     EndIf
5:     If s < p then
6:         f(sir + "-1 ", s + 1, d, p)
7:     EndIf
8:     If s > d then
9:         f(sir + "1 ", s, d + 1, p)
10:    EndIf
11: EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate în urma apelului `f("", 0, 0, 2)`:

A. Se afișează două șiruri de caractere pe linii separate, fiecare conținând 4 numere a căror sumă este 0 (de exemplu, suma numerelor din șirul de caractere "-1 1 -1 1" este 0)

B. Se afișează doar "-1 -1 1 1".

C. Se afișează doar "-1 -1 1 1", dar algoritmul nu își termină execuția din cauza unei erori.

D. Dacă pe linia 2 s-ar înlocui operatorul **AND** cu **OR** , atunci s-ar afișa doar "-1 -1".

343. Se consideră algoritmul `ceFace(a, i, n)`, unde i și n sunt numere naturale ($1 \leq i, n \leq 100$), iar a este un vector cu n elemente numere întregi ($a[1], a[2], ..., a[n], -100 \leq a[i] \leq 100$). În șirul a se află cel puțin un număr pozitiv. Algoritmul `max(x, y, z)` returnează maximul dintre trei numere întregi *x, y* și *z* ($-10^4 \leq x, y, z \leq 10^4$). Algoritmul `ceFace(a, 1, n)` apelează algoritmul `intermediar(a, i, m, n)`, unde parametrii *a, i* și *n* au semnificația de mai sus, iar m este un număr natural ($1 \leq m \leq n$). ✓ ?

```
Algorithm INTERMEDIAR(a, i, m, n)
    s ← 0
    left ← a[m]
    For k ← m, i, -1 execute
        s ← s + a[k]
        If s > left then
            left ← s
        EndIf
    EndFor
    s ← 0
    right ← a[m]
    For i ← m, n execute
        s ← s + a[i]
        If s > right then
            right ← s
        EndIf
    EndFor
    Return max(left, right, left + right − a[m])
EndAlgorithm
```

```
Algorithm CEFACE(a, i, n)
    If i > n then
        Return a[i]
    EndIf
    m ← (i + n) DIV 2
    v1 ← ceFace(a, i, m − 1)
    v2 ← ceFace(a, m + 1, n)
    v3 ← intermediar(a, i, m, n)
    Return max(v1, v2, v3)
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate dacă algoritmul se apelează sub forma `ceFace(a, i, n)`:

A. Algoritmul identifică o poziție m a vectorului a astfel încât fie suma elementelor de pe pozițiile $1, 2, ..., m$, fie suma elementelor de pe pozițiile $m, m+1, ..., n$ să fie maximul care se poate obține pentru orice $1 \leq m \leq n$, și returnează suma maximă obținută astfel.

B. Algoritmul returnează suma maximă care se poate obține însumând elementele unei submulțimi a valorilor vectorului a.

C. Algoritmul returnează suma maximă care se poate obține pentru o subsecvență a vectorului a.

D. În cazul în care toate elementele vectorului a sunt pozitive, algoritmul returnează suma tuturor elementelor vectorului a.

Admitere nivel licență, sesiunea iulie 2023

344. Se consideră algoritmul `F(x)`, unde x este număr natural ($1 \leq x \leq 10^6$): ✓ ?

```
Algorithm F(x)
   If x = 0 then
      Return 0
   Else
      If x MOD 3 = 0 then
         Return F(x DIV 10) + 1
      Else
         Return F(x DIV 10)
      EndIf
   EndIf
EndAlgorithm
```

Pentru care dintre următoarele apeluri se returnează 4?

A. F(21369)

B. F(6933)

C. F(4)

D. F(16639)

345. Se consideră algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10^4$) care nu conțin cifra 0: ✓ ?

```
Algorithm ceFace(a, b)
   p ← 0
   While a ≠ 0 execute
      c ← a MOD 10
      p ← p * 10 + c
      a ← a DIV 10
   EndWhile
   If p = b then
      Return True
   Else
      Return False
   EndIf
EndAlgorithm
```

Algoritmul `ceFace(a, b)` returnează *True* dacă și numai dacă:

A. numerele a și b sunt egale

B. a și b sunt numere palindrom

C. numărul a este oglinditul numărului b

D. ultima cifră a lui a este egală cu ultima cifră a lui b

346. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural ($1 \leq n \leq 10^3$). ✓ ? Operatorul „/" reprezintă împărțirea reală, de exemplu: $3/2 = 1.5$.

```
Algorithm ceFace(n)
   s ← 0
   For i ← 1, n execute
      p ← (i + 1) * (i + 2)
      s ← s + (i/p)
   EndFor
   Return s
EndAlgorithm
```

Precizați expresia a cărei valoare este returnată de algoritm.

A. $\frac{1}{1} + \frac{1}{1+2} + \cdots + \frac{1}{1+2+\cdots+n}$

B. $\frac{1}{2 \cdot 3} + \frac{2}{3 \cdot 4} + \cdots + \frac{n}{(n+1)(n+2)}$

C. $\frac{1}{1} + \frac{1}{1 \cdot 2} + \cdots + \frac{1}{1 \cdot 2 \cdot \ldots \cdot n}$

D. $\frac{1}{2 \cdot 3} + \frac{2}{3 \cdot 4} + \cdots + \frac{n-1}{n(n+1)}$

347. Se consideră algoritmul `f(n, x)`, unde n este număr natural ($3 \leq n \leq 10^4$), iar ✓ ? x este un vector de n numere naturale $(x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n)$:

```
Algorithm F(n, x)
   k ← 0
   For i ← 1, n − 1 execute
      If k = 0 then
         If x[i] = x[i + 1] then
            Return False
         EndIf
         If x[i] < x[i + 1] then
            k ← 1
         EndIf
      EndIf
      If k = 1 then
         If x[i] ≥ x[i + 1] then
            Return False
         EndIf
      EndIf
   EndFor
   If x[n − 1] ≥ x[n] then
      Return False
   EndIf
   Return True
EndAlgorithm
```

Pentru care din următoarele apeluri algoritmul va returna *True*?

A. f(6, [1000, 512, 23, 22, 1, 2])

B. f(6, [6, 4, 1, 1, 2, 3])

C. f(8, [3000, 2538, 799, 424, 255, 256, 299, 1001])

D. f(3, [3, 2, 1])

348. Se dă algoritmul `calcul(a, b, c, d)`, unde a, b, c, d sunt numere naturale nenule ✓ ? $(1 \leq a, b, c, d \leq 100)$:

```
Algorithm CALCUL(a, b, c, d)
   x ← a ∗ b
   y ← c ∗ d
   While y ≠ 0 execute
      z ← x MOD y
      x ← y
      y ← z
   EndWhile
   Return x
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează cel mai mare divizor comun al numerelor a, b, c, d.

B. Algoritmul returnează cel mai mare divizor comun al numerelor $a*b$ și $c*d$.

C. Algoritmul returnează cel mai mic multiplu comun al numerelor a, b, c, d.

D. Algoritmul returnează cel mai mic multiplu comun al numerelor $a * b$ și $c * d$.

349. Se consideră algoritmul `p(na, a, nb, b)`, unde na și nb sunt numere naturale $(0 \leq$ ✓ ? $na, nb \leq 10^4)$, a și b sunt vectori cu na, respectiv nb numere naturale $(a[1], a[2], ..., a[na], 1 \leq a[i] \leq 10^4$, pentru $i = 1, 2, ..., na$ și $b[1], b[2], ..., b[nb], 1 \leq b[i] \leq 10^4$, pentru $i = 1, 2, ..., nb)$. Variabila locală c este un vector.

```
Algorithm P(na, a, nb, b)
    i ← 1
    j ← 1
    nc ← 0
    While i ≤ na AND j ≤ nb
execute
        nc ← nc + 1
        If a[i] < b[j] then
            c[nc] ← a[i]
            i ← i + 1
        Else
            c[nc] ← b[j]
            j ← j + 1
        EndIf
    EndWhile
    Return nc
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă $na = 0$ și $nb = 0$, atunci valoarea returnată prin nc este egală cu 0.

B. Dacă elementele din a și b sunt sortate crescător, atunci elementele depuse în c sunt sortate crescător.

C. Valoarea returnată prin nc este întotdeauna egală cu $na + nb$.

D. Dacă $na, nb > 0$ și cel mai mare element din a este mai mic decât toate elementele din b, atunci c va avea exact aceleași elemente ca și a.

350. Se dă algoritmul `suma(n, a, m, b)`, unde n și m sunt numere naturale ($1 \leq n, m \leq 10^5$), iar a și b sunt două șiruri ordonate crescător cu n, respectiv m elemente numere naturale ($a[1], a[2], ..., a[n]$ și $b[1], b[2], ..., b[m]$): ✓ ?

```
Algorithm SUMA(n, a, m, b)
    s ← 0
    For i ← 1, n, 2 execute
        j ← 1
        While j ≤ a[i] AND j ≤ m
execute
            s ← s + b[j]
            j ← j + 1
        EndWhile
    EndFor
    Return s
EndAlgorithm
```

Ce valoare va returna algoritmul, dacă $n = 4$, $a = [1, 3, 4, 7]$, $m = 6$ și $b = [2, 4, 6, 8, 10, 12]$?

A. 42

B. 22

C. 20

D. Nu se poate determina ce valoare va returna

351. Se consideră algoritmul `verifica(n, p1, p2)`, unde n, *p1* și *p2* sunt numere naturale ($1 \leq n, p1, p2 \leq 10^6$): ✓ ?

```
Algorithm VERIFICA(n, p1, p2)
    bt ← (p1 + p2) DIV 2
    If p1 > p2 then
        Return False
    EndIf
    If bt * bt = n then
        Return True
    EndIf
    If bt * bt > n then
        Return verifica(n, p1, bt − 1)
    EndIf
    Return verifica(n, bt + 1, p2)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă numerele *p1*, *p2* și n sunt prime între ele, atunci apelul `verifica(n, p1, p2)` returnează *True*.

B. Algoritmul folosește metoda căutării binare și dacă numărul n este prim, apelul `verifica(n, 1, n)` returnează *True*.

C. Pentru apelul `verifica(n, 1, n)` algoritmul returnează *True* dacă și numai dacă numărul n este pătrat perfect.

D. Dacă $p1 \leq n \leq p2$ și în intervalele $[p1, n]$ și $[n, p2]$ există cel puțin câte un pătrat perfect, atunci apelul `verifica(n, p1, p2)` returnează *True*.

352. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural ($1 \leq n \leq 3000$): ✓ ?

```
Algorithm ceFace(n)
    s ← 0
    i ← 1
    While s < n execute
        s ← s + i
        If s = n then
            Return True
        Else
            i ← i + 2
        EndIf
    EndWhile
    Return False
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă $n = 36$, algoritmul returnează *True*.

B. Dacă n este egal cu o sumă de numere impare consecutive începând de la 1, algoritmul returnează *True*.

C. Dacă n este pătrat perfect, algoritmul returnează *True*, altfel returnează *False*.

D. Dacă $n = 64$, algoritmul returnează *False*.

353. Se consideră algoritmul `ceFace(a)`, unde a este număr natural ($1 \leq a \leq 10^4$): ✓ ?

```
Algorithm ceFace(a)
    ok ← 0
    While ok = 0 execute
        b ← a
        c ← 0
        While b ≠ 0 execute
            c ← c * 10 + b MOD 10
            b ← b DIV 10
        EndWhile
        If c = a then
            ok ← 1
        Else
            a ← a + 1
        EndIf
    EndWhile
    Return a
EndAlgorithm
```

Precizați efectul algoritmului.

A. Algoritmul returnează cel mai mic palindrom mai mare sau egal cu a.

B. Algoritmul returnează cel mai mare palindrom mai mic sau egal cu a.

C. Algoritmul returnează cel mai mic palindrom mai mare decât a.

D. Algoritmul returnează cel mai mic număr par mai mare decât a.

354. Se consideră algoritmul `calcul(v, n)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar v este un vector cu n elemente numere naturale ($v[1], v[2], ..., v[n], 1 \leq v[i] \leq 10^4$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm CALCUL(v, n)
    i ← 2
    x ← 0
    If v[1] MOD 2 ≠ 0 then
        Return False
    EndIf
    While i ≤ n execute
        If x = 0  AND
                v[i] MOD 2 = 0 then
            Return False
        Else
            If x = 1  AND
                    v[i] MOD 2 = 1 then
                Return False
            Else
                i ← i + 1
                x ← (x + 1) MOD 2
            EndIf
        EndIf
    EndWhile
    Return True
EndAlgorithm
```

În care din următoarele situații algoritmul returnează *True*?

A. Dacă vectorul v este format din valorile $[2, 3, 10, 7, 20, 5, 18]$ și $n = 7$

B. Dacă vectorul v are valori după următorul model: impar, par, impar, par...

C. Dacă vectorul v este format din valorile $[3, 8, 17, 20, 15, 10]$ și $n = 6$

D. Dacă vectorul v are valori după următorul model: par, impar, par, impar...

355. Se consideră algoritmul `ceFace(a, n)`, unde n este număr natural nenul ($2 \leq n \leq 10^4$) și a este un vector cu n numere întregi ($a[1], a[2], ..., a[n], -100 \leq a[i] \leq 100, i = 1, 2, ..., n$). În vectorul a există cel puțin un număr pozitiv. ✓ ?

```
Algorithm CEFACE(a, n)
    b ← 0
    c ← b
    For i ← 1, n execute
        b ← b + a[i]
        If b < 0 then
            b ← 0
        EndIf
        If b > c then
            c ← b
        EndIf
    EndFor
    Return c
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează suma tuturor elementelor din vectorul a.

B. Algoritmul returnează suma subsecvenței de lungime maximă care conține doar elemente pozitive din vectorul a.

C. Algoritmul returnează suma tuturor elementelor pozitive din vectorul a.

D. Algoritmul returnează suma unei subsecvențe cu suma maximă din vectorul a.

356. Se consideră o matrice A de numere întregi cu n linii și m coloane ($1 \leq n, m \leq 10^4$). În condițiile în care $n*m = p*q$, dorim să redimensionăm această matrice într-o matrice B de numere întregi cu p linii și q coloane ($1 \leq p, q \leq 10^4$), conform exemplului de mai jos, unde $n = 4$, $m = 6$, $p = 3$ și $q = 8$. Liniile și coloanele sunt numerotate începând de la 1. ✓ ?

A:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

B:

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24

Care din următoarele variante prezintă un algoritm care pentru perechea de numere naturale i și j $(1 \leq i \leq n, 1 \leq j \leq m)$ reprezentând indici în matricea A va returna perechea de indici din matricea B corespunzătoare valorii $A[i][j]$?

A.

```
Algorithm RESHAPE(i, j, n, m,
p, q)
   Return (i * m + j) DIV q, (i *
m + j) MOD q
EndAlgorithm
```

B.

```
Algorithm RESHAPE(i, j, n, m,
p, q)
   i ← i − 1
   j ← j − 1
   Return (i * m + j) DIV q, (i *
m + j) MOD q
EndAlgorithm
```

C.

```
Algorithm RESHAPE(i, j, n, m,
p, q)
   i ← i − 1
   j ← j − 1
   Return (i * m + j) DIV q + 1,
          (i * m + j) MOD q + 1
EndAlgorithm
```

D.

```
Algorithm RESHAPE(i, j, n, m,
p, q)
   Return (i*m+j−1) DIV q+1,
          (i*m+j−1) MOD q+1
EndAlgorithm
```

357. Se consideră algoritmul `ceFace(n, m)`, unde n este număr natural $(1 \leq n \leq 10^4)$, iar m este o matrice cu n linii și n coloane, iar elementele sunt numere naturale $(m[1][1], ..., m[1][n], m[2][1], ..., m[2][n], ..., m[n][1], ..., m[n][n])$. Considerăm că elementele matricei m sunt inițial egale cu 0.

```
Algorithm ceFace(n, m)
   a ← 0
   b ← 1
   For j ← 1, n execute
      i ← 1
      While i + j ≤ n − 1 execute
         If (i MOD 2 = 1) AND
                (j MOD 2 = 1) then
            m[i][j] ← b
            c ← a + b
            a ← b
            b ← c
         EndIf
         i ← i + 1
      EndWhile
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt FALSE?

A. Dacă $n = 11$, valoarea lui $m[6][4]$ este 21

B. Dacă $n = 7$, valoarea lui $m[3][5]$ este 4

C. Dacă $n = 10$, valoarea lui $m[6][4]$ este 21

D. Dacă $n = 7$, valoarea maximă din matrice este 8

358. Algoritmii de mai jos prelucrează un vector x ordonat crescător, având n elemente numere naturale $(1 \leq n \leq 10^4, x[1], x[2], ..., x[n])$. Parametrii *first* și *last* sunt numere naturale $(1 \leq first \leq last \leq n)$. Alegeți algoritmii care au complexitatea timp cea mai scăzută, dacă se apelează sub forma `A(x, 1, n, n)`.

A.

```
Algorithm A(x, first, last, n)
   If first > last then
      Return 0
   EndIf
   m ← (first + last) DIV 2
   If x[m] = n then
      Return m
   Else
      If x[m] > n then
         Return
              A(x, first, m − 1, n)
      Else
         If x[m] < n then
            Return
                 A(x, m + 1, last, n)
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

B.

```
Algorithm A(x, first, last, n)
   While first < last execute
      m ← (first + last) DIV 2
      If x[m] = n then
         Return m
      Else
         If x[m] > n then
            last ← m − 1
         Else
            If x[m] < n then
               first ← m + 1
            EndIf
         EndIf
      EndIf
   EndWhile
   Return 0
EndAlgorithm
```

C.

```
Algorithm A(x, first, last, n)
   For i ← first, last execute
      If x[i] = n then
         Return i
      EndIf
   EndFor
   Return 0
EndAlgorithm
```

D.

```
Algorithm A(x, first, last, n)
   For i ← first, last execute
      If x[i] = n then
         x[i] ← 3 * n
      EndIf
   EndFor
EndAlgorithm
```

359. Andrei se joacă cu următorul algoritm, unde n și m sunt numere naturale nenule ✓ ? $(1 \leq n, m \leq 10^4)$. Algoritmul `abs(x)` returnează valoarea absolută a lui x.

```
Algorithm PROBLEMA(n, m)
   b ← abs(m − n)
   c ← n − m
   If b − c = 0 then
      a ← n MOD m
   Else
      a ← (m + 2) MOD n
   EndIf
   Return a
EndAlgorithm
```

El observă că indiferent de valoarea variabilei n corespunzătoare specificației, există cel puțin două valori ale lui m în cazul cărora algoritmul `problema(n, m)` returnează 0. Care sunt aceste valori ale lui m?

A. 1 și n

B. 1 și $n + 2$

C. n și $n + 2$

D. 1 și $n - 2$

360. Un elev dorește să genereze, folosind metoda backtracking, toate numerele impare ✓ ? cu câte trei cifre, cifre care iau valori din vectorul $[4, 3, 8, 5, 7, 6]$, în ordinea dată. Știind că primele 5 numere generate sunt, în această ordine: 443, 445, 447, 433, 435, care va fi cel de-al zecelea număr generat?

A. 487 B. 453 C. 457 D. 455

361. Se consideră algoritmul `f(k, n, x)`, unde k, n sunt numere naturale ($1 \leq k, n \leq 10^3$) și x este un vector de n numere naturale ($x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm F(k, n, x)
   If n = 0 then
      Return 0
   Else
      d ← 0
      For i ← 2, x[n] DIV 2 execute
         If (x[n] MOD i) = 0 then
            d ← d + 1
         EndIf
      EndFor
      If d = k then
         Return 1 + f(k, n − 1, x)
      Else
         Return f(k, n − 1, x)
      EndIf
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $x = [4, 9, 26, 12]$ rezultatul apelului `f(1, 4, x)` va fi 3.

B. Pentru $x = [4, 8, 6, 144]$ rezultatul apelului `f(2, 4, x)` va fi 3.

C. Pentru $x = [4, 9, 25, 144]$ rezultatul apelului `f(1, 4, x)` va fi 3.

D. Pentru $x = [8, 27, 25, 121]$ rezultatul apelului `f(2, 4, x)` va fi 3.

362. Fie algoritmul `check(n)`, unde n este număr natural ($1 \leq n \leq 10^5$): ✓ ?

```
Algorithm CHECK(n)
   While n > 0 execute
      If n MOD 3 > 1 then
         Return False
      EndIf
      n ← n DIV 3
   EndWhile
   Return True
EndAlgorithm
```

Precizați efectul algoritmului.

A. Algoritmul returnează *True* dacă n este o putere a lui 3 și *False* în caz contrar.

B. Algoritmul returnează *True* dacă scrierea în baza 3 a lui n conține doar cifrele 0 și 1 și *False* în caz contrar.

C. Algoritmul returnează *True* dacă n poate fi scris ca o putere a lui 3 sau ca sumă de puteri distincte ale lui 3 și *False* în caz contrar.

D. Algoritmul returnează *True* dacă scrierea în baza 3 a lui n conține doar cifra 2 și *False* în caz contrar.

363. Un eveniment trebuia să aibă loc într-o anumită sală I, dar trebuie mutat în sala II, unde numerotarea scaunelor diferă. În ambele săli există L rânduri de scaune ($2 \leq L \leq 50$), fiecare rând fiind împărțit la mijloc de un culoar și având K scaune ($2 \leq K \leq 50$) în fiecare parte a culoarului (deci, sala conține în total $2 * K * L$ scaune). În sala I fiecare loc este identificat printr-un singur număr. Locurile din stânga culoarului au numere pare, iar numerotarea scaunelor începe pe rândul din fața scenei. Deci scaunele din primul rând au numerele (pornind dinspre culoar spre marginea sălii) 2, 4, 6 etc. După ce toate scaunele de pe un rând au fost numerotate, pe rândul următor se continuă numerotarea, reîncepând cu scaunul de lângă culoar cu următorul număr par. Locurile din partea dreaptă a culoarului sunt numerotate la ✓ ?

fel, dar folosind numere impare. Deci scaunele din primul rând au numerele (pornind dinspre culoar spre marginea sălii) 1, 3, 5 etc. În sala II fiecare loc este identificat prin trei valori. Numărul rândului (o valoare între 1 și L, rândul 1 fiind cel din fața scenei), direcția locului față de culoar (valoarea "stânga" sau "dreapta") și numărul scaunului în cadrul rândului (o valoare între 1 și K, scaunul 1 fiind cel de lângă culoar). Din cauza mutării spectacolului, locurile de pe bilete din sala I (reprezentate printr-un singur număr) trebuie transformate în locuri valabile în sala II (reprezentate prin *rând*, *loc*, *direcție*). Care dintre următorii algoritmi, având ca date de intrare L, K, $nrLoc$ conform enunțului execută în mod corect transformarea? O transformare este corectă dacă fiecare spectator va avea un loc unic în sala II.

A.

```
Algorithm TRANSFORMA(L, K,
nrLoc)
   If nrLoc MOD 2 = 1 then
      directie ← "dreapta"
      nrLoc ← nrLoc + 1
   Else
      directie ← "stanga"
   EndIf
   If nrLoc MOD (2 * K) = 0 then
      rand ← nrLoc DIV (2 * K)
   Else
      rand ← nrLoc DIV (2*K)+1
   EndIf
   loc ← (nrLoc − (rand − 1) * 2 *
K) DIV 2
   Return rand, loc, directie
EndAlgorithm
```

B.

```
Algorithm TRANSFORMA(L, K,
nrLoc)
   If nrLoc MOD 2 = 1 then
      directie ← "dreapta"
   Else
      directie ← "stanga"
   EndIf
   If nrLoc MOD (2 * K) = 0 then
      rand ← nrLoc DIV (2 * K)
   Else
      rand ← nrLoc DIV (2*K)+1
   EndIf
   loc ← (nrLoc − (rand − 1) * 2 *
K) DIV 2
   Return rand, loc, directie
EndAlgorithm
```

C.

```
Algorithm TRANSFORMA(L, K,
nrLoc)
   If nrLoc MOD 2 = 1 then
      directie ← "dreapta"
      nrLoc ← nrLoc + 1
   Else
      directie ← "stanga"
   EndIf
   rand ← nrLoc DIV (2 * K) + 1
   loc ← (nrLoc − (rand − 1) * 2 *
K) DIV 2
   Return rand, loc, directie
EndAlgorithm
```

D.

```
Algorithm TRANSFORMA(L, K,
nrLoc)
   If nrLoc MOD 2 = 1 then
      directie ← "dreapta"
      nrLoc ← nrLoc + 1
   Else
      directie ← "stanga"
   EndIf
   If nrLoc MOD (2 * K) = 0 then
      rand ← nrLoc DIV (2 * K)
   Else
      rand ← nrLoc DIV (2*K)+1
   EndIf
   loc ← (nrLoc − (rand − 1) * 2 *
K) DIV 2 + 1
   Return rand, loc, directie
EndAlgorithm
```

364. Se consideră algoritmul `p(x, n, k, final)`, unde x este un vector de $n + 1$ numere naturale $(x[0], x[1], x[2], ..., x[n])$. Inițial $x[i] = 0$, pentru $i = 0, 1, 2, ..., n$. Variabilele n și k sunt numere naturale nenule ($1 \leq n, k \leq 20$), iar *final* este de tip boolean. Algoritmul `Afis(x, 1, n)` afișează elementele $x[1], x[2], ..., x[n]$.

```
Algorithm p(x, n, k, final)
   While final = False execute
      While x[k] < n execute
         x[k] ← x[k] + 1
         If OK(x, k) = True then
            If k = n then
               Afis(x, 1, n)
            Else
               k ← k + 1
               x[k] ← 0
            EndIf
         EndIf
      EndWhile
      _____
   EndWhile
EndAlgorithm
```

```
Algorithm OK(x, k)
   For i ← 1, k − 1 execute
      If x[k] = x[i] then
         Return False
      EndIf
   EndFor
   Return True
EndAlgorithm
```

Cu ce secvență de cod trebuie completat algoritmul, astfel încât în urma apelului `p(x, n, 1, False)` să se afișeze toate permutările de ordin n, fiecare o singură dată.

A.
```
If k > 1 then
   k ← k − 1
Else
   final ← True
EndIf
```

B.
```
If k > 0 then
   k ← k − 1
Else
   final ← True
EndIf
```

C.
```
final ← True
```

D.
```
If k > 1 then
   k ← k − 1
   final ← True
EndIf
```

365. Se dau algoritmii `problema(n)` și `calcul(a, b)`, unde n, a, b sunt numere naturale $(0 \leq n, a, b \leq 9)$: ✓ ?

```
Algorithm problema(n)
   rezultat ← 0
   For k ← 0, n execute
      For p ← 0, n execute
         For j ← 0, n execute
            If p MOD 2 = 0 then
               rezultat ← rezultat + 1
            EndIf
         EndFor
      EndFor
   EndFor
   Return rezultat
EndAlgorithm

Algorithm calcul(a, b)
   t ← 0
   For cifra ← a, b execute
      t ← t + problema(cifra)
   EndFor
```

```
    Write t
EndAlgorithm
```

Care din următoarele afirmații sunt adevărate?

A. În urma apelului `calcul(1, 8)` se afișează 1095.

B. În urma apelului `calcul(1, 8)` se afișează 1094.

C. În urma apelului `calcul(0, 9)` se afișează 1095.

D. În urma apelului `calcul(0, 9)` se afișează 1595.

366. Se consideră algoritmul `checkAcc(n, f, w, lw)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$), f este număr natural, w este un șir de lw ($1 \leq lw \leq 10^4$) numere naturale ($w[1], w[2], ..., w[lw]$, unde $0 \leq w[p] \leq 10^4$, pentru $p = 1, 2, ..., lw$). Algoritmul `checkAcc(n, f, w, lw)` apelează algoritmul `t(i, j, k)`, unde i, j și k sunt numere naturale. Algoritmul `t(i, j, k)` returnează rezultat boolean.

```
Algorithm CHECKACC(n, f, w, lw)
    acc ← True
    If lw = 0 AND f ≠ 1 then
        acc ← False
    Else
        index ← 1
        q ← 1
        While (acc = True) AND
                (index ≤ lw) execute
            crt ← 1
            changed ← False
            While (changed = False)
                    AND (crt ≤ n)
execute
                If t(q, w[index], crt) then
                    q ← crt
                    changed ← True
                Else
                    crt ← crt + 1
                EndIf
            EndWhile
            If changed = False then
                acc ← False
            Else
                index ← index + 1
            EndIf
        EndWhile
        If (index > lw) AND
                (acc = True) AND
                (q ≠ f) then
            acc ← False
        EndIf
    EndIf
    Return acc
EndAlgorithm
```

În care dintre situațiile de mai jos algoritmul `checkAcc(2, f, w, lw)` va returna *True*, știind că algoritmul `t(i, j, k)` returnează *True* în cazurile din tabel, altfel returnează *False*?

i	j	k
1	0	1
1	1	2
2	1	2

A. $w = [0, 0, 1, 1]$, $lw = 4$ și $f = 1$

B. $w = [1, 1, 1, 0]$, $lw = 4$ și $f = 2$

C. $w = [0, 0, 1, 1]$, $lw = 4$ și $f = 2$

D. $w = [0, 0, 0, 0]$, $lw = 4$ și $f = 1$

367. Se consideră vectorul de cifre $a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$. Cu scopul de a afișa

elementele vectorului a într-o altă ordine, se construiește vectorul b (inițial vid). La fiecare pas, se poate alege una din următoarele două operații:

(a) *Adaugă* – se adaugă primul element din vectorul a la finalul vectorului b și se elimină din vectorul a.

(b) *Șterge* – se afișează, apoi se șterge ultimul element din vectorul b.

Observații:

(a) Elementele vectorului a se prelucrează în ordinea dată.

(b) Nu se poate folosi operația *Adaugă* dacă vectorul a este vid și nu se poate folosi operația *Șterge*, dacă vectorul b este vid.

(c) Prelucrarea se termină când vectorii a și b sunt vizi.

Respectând regulile de mai sus, în ce ordine NU pot fi afișate cifrele?

A. 0 1 2 3 4 5 6 7 8 9

B. 9 8 7 6 5 4 3 2 1 0

C. 2 4 6 5 3 7 0 1 9 8

D. 2 3 1 4 5 0 8 9 7 6

Admitere nivel licență, sesiunea septembrie 2022

368. Se consideră algoritmul `decide(n, x)`, unde n este număr natural ($1 \leq n \leq 10000$), iar x este un vector cu n elemente numere întregi ($x[1], x[2], \ldots, x[n]$, $-100 \leq x[i] \leq 100, \forall i$): ✓ ?

```
Algorithm DECIDE(n, x)
    b ← True
    i ← 1
    While b = True & i < n execute
        If x[i] < x[i + 1] then
            b ← True
        Else
            b ← False
        EndIf
        i ← i + 1
    EndWhile
    Return b
EndAlgorithm
```

Pentru care dintre următoarele situații algoritmul returnează `True`?

A. Dacă vectorul x este format din valorile $1, 2, 3, \ldots, 10$.

B. Dacă vectorul x este strict crescător.

C. Dacă vectorul x nu are elemente negative.

D. Dacă vectorul x are elemente pozitive situate înaintea celor negative.

369. Se consideră un număr natural fără cifre egale cu zero, dat prin șirul a ($a[1], \ldots, a[n]$) în care se află cele n cifre ale sale ($1 \leq n \leq 10$ la momentul apelului inițial). Precizați care dintre următorii algoritmi returnează `True` dacă un număr dat sub această formă este palindrom și `False` în caz contrar. Un număr este palindrom dacă citit de la stânga la dreapta are aceeași valoare ca atunci când se citește de la dreapta la stânga. ✓ ?

A.

```
Algorithm PALINDROM1(a, x)
    i ← 1
    j ← n
    k ← True
    While (i ≤ j) AND (k = True)
    execute
        If a[i] = a[j] then
            i ← i + 1
            j ← j − 1
        Else
            k ← False
        EndIf
    EndWhile
    Return k
EndAlgorithm
```

B.

```
Algorithm TRANSLATARE(a, n)
    For i ← 1, n − 1 execute
        a[i] ← a[i + 1]
    EndFor
EndAlgorithm
Algorithm PALINDROM2(a, n)
    j ← n
    If (j = 0) OR (j = 1) then
        Return True
    EndIf
    If a[1] = a[j] then
        translatare(a, n)
        Return palindrom2(a, n − 2)
    EndIf
    Return False
EndAlgorithm
```

C.

```
Algorithm PALINDROM3(a, n)
    i ← n
    j ← 1
    k ← True
    sum1 ← 0
    sum2 ← 0
    While (i > n DIV 2) AND (j ≤
n DIV 2) execute
        sum1 ← sum1 + a[i]
        sum2 ← sum2 + a[j]
        i ← i − 1
        j ← j + 1
    EndWhile
    If sum1 = sum2 then
        k ← True
    Else
        k ← False
    EndIf
    Return k
EndAlgorithm
```

D.

```
Algorithm PALINDROM4(a, n)
    i ← 1
    j ← n
    k ← True
    While (i ≤ j) AND (k = True)
execute
        If (a[i] = a[j]) AND (i MOD 2 =
0) AND (j MOD 2 = 0) then
            i ← i + 1
            j ← j − 1
        Else
            k = False
        EndIf
    EndWhile
    Return k
EndAlgorithm
```

370. Se consideră algoritmul `F(n)`, unde n este număr natural ($1 \leq n \leq 10^9$): ✓ ?

```
Algorithm F(n)
    If n < 10 then
        Return n
    EndIf
    u ← n MOD 10
    p ← F(n DIV 10)
    If u MOD 5 ≤ p MOD 5 then
        Return u
    EndIf
    Return p
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt corecte:

A. Dacă $n = 812376$, valoarea returnată de algoritm este 6.

B. Dacă $n = 8237631$, valoarea returnată de algoritm este 1.

C. Dacă $n = 4868$, valoarea returnată de algoritm este 8.

D. Dacă $n = 51$, valoarea returnată de algoritm este 0.

371. Se consideră algoritmul `f(n)`, unde parametrul n este număr natural ($1 \leq n \leq 10^9$): ✓ ?

```
Algorithm F(n)
    v ← 0; z ← 0
    For c ← 0, 9 execute
        x ← n
        k ← 0
        While x > 0 execute
            If x MOD 10 = c then
                k ← k + 1
            EndIf
            x ← x DIV 10
```

```
        EndWhile
        If k > v then
            v ← k
            z ← c
        EndIf
    EndFor
    Return z
EndAlgorithm
```

Care dintre afirmațiile următoare sunt adevărate?

A. Algoritmul returnează numărul cifrelor numărului n.

B. Algoritmul returnează numărul de apariții ale cifrei cu valoarea cea mai mare în numărul n.

C. Algoritmul returnează una dintre cifrele cu cel mai mare număr de apariții în numărul n.

D. Algoritmul returnează numărul cifrelor având cel mai mare număr de apariții în numărul n.

372. Care dintre următorii algoritmi afișează reprezentarea binară a numărului natural x dat ca parametru ($0 < x \leq 10^9$) la momentul apelului inițial)? ✓ ?

A.

```
Algorithm IMP(x)
    If x = 0 then
        r ← x MOD 2
        imp(x DIV 2)
        write r
    EndIf
EndAlgorithm
```

B.

```
Algorithm IMP(x)
    If x ≠ 0 then
        r ← x MOD 2
        imp(x DIV 2)
        write r
    EndIf
EndAlgorithm
```

C.

```
Algorithm IMP(x)
    If x = 0 then
        r ← x DIV 2
        imp(x DIV 2)
        write r
    EndIf
EndAlgorithm
```

D.

```
Algorithm IMP(x)
    If x ≠ 0 then
        r ← x MOD 2
        imp(x)
        write r
    EndIf
EndAlgorithm
```

373. Care dintre următoarele afirmații referitoare la variantele de răspuns ale problemei **372.** sunt adevărate? ✓ ?

A. În timpul execuției algoritmului de la varianta A nu se afișează nimic.

B. Algoritmul de la varianta B nu se va apela recursiv pentru nicio valoare validă a parametrului x

C. Algoritmul de la varianta C ar fi corect, dacă am schimba "=" cu "≠"

D. Algoritmul de la varianta D ar fi corect, dacă am schimba "imp(x)" cu "imp(x **DIV** 2)".

374. Se consideră numerele întregi a și b ($-1000 \leq a, b \leq 1000$) și expresia: ✓ ?

$$\textbf{NOT}\ ((a > 0)\ \textbf{AND}\ (b > 0)).$$

Care dintre următoarele expresii sunt echivalente cu expresia dată mai sus:

A. (**NOT** $(a < 0)$) **AND** (**NOT** $(b < 0)$)

B. $(a \leq 0)$ **AND** $(b \leq 0)$

C. (**NOT** $(a > 0)$) **OR** (**NOT** $(b > 0)$)

D. **NOT** $((a > 0)$ **OR** $(b < 0))$

375. Se consideră algoritmul `s(n)`, unde n este număr natural ($2 \leq n \leq 10$). Operatorul / reprezintă împărțirea reală (ex. $3/2 = 1,5$). ✓ ?

```
Algorithm s(n)
    p ← 1
    x ← 0
    For k = 0, n − 1 execute
        p ← p ∗ (k + 1)
        x ← x + 1/p
    EndFor
    Return x
EndAlgorithm
```

Precizați care dintre următoarele sume este returnată de algoritmul dat.

A. $\sum_{k=0}^{n} \frac{1}{k!}$

B. $\sum_{k=0}^{n} \frac{1}{k}$

C. $\sum_{k=0}^{n-1} \frac{1}{k!}$

D. $\sum_{k=1}^{n} \frac{1}{k!}$

376. Se consideră algoritmul `ceFace(n)`, unde n este număr natural pozitiv ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm ceFace(n)
    m ← 0
    p ← 10
    While p < n execute
        r ← n MOD p
        m ← m + r
        p ← p ∗ 10
    EndWhile
    Return m
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $n = 125$ algoritmul returnează valoarea 521.

B. Algoritmul `ceFace(n)` returnează oglinditul numărului n.

C. Pentru $n = 125$ algoritmul returnează valoarea 155.

D. Pentru $n = 340$ algoritmul returnează valoarea 40.

377. Se consideră algoritmul `f(v, n)`, unde n este număr natural nenul ($1 \leq n \leq 10000$) și v este un vector cu n numere naturale pozitive ($v[1], v[2], ..., v[n]$). Presupunem că algoritmul `prim(d)` returnează *True* dacă d (număr natural) este prim și *False* în caz contrar. ✓ ?

```
Algorithm F(v, n)
    x ← 1
    a ← 0
    For i ← 1, n execute
        For d ← 2, (v[i] DIV 2) execute
            If (prim(d) = True) AND (v[i] MOD d = 0) then
                x ← x * d
            EndIf
        EndFor
    EndFor
    For d ← 2, (x DIV 2) execute
        If (x MOD d = 0) AND (prim(d) = True) then
            a ← a + 1
        EndIf
    EndFor
    Return a
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează numărul divizorilor proprii primi distincți ai tuturor numerelor din vectorul v.

B. Algoritmul returnează produsul divizorilor primi ai numerelor din vectorul v.

C. Algoritmul returnează numărul numerelor prime din vectorul v.

D. Algoritmul returnează numărul total al tuturor divizorilor numerelor din vectorul v.

378. Se consideră algoritmul `f(n)`, unde n este număr natural ($0 < n \leq 10^9$ la momentul apelului). Variabila locală v este un vector. ✓ ?

```
Algorithm F(n)
    m ← 0
    While n > 0 execute
        m ← m + 1
        v[m] ← n MOD 10
        n ← n DIV 10
    EndWhile
    x ← 0
    mx ← 0
    While mx > −1 execute
        x ← x * 10 + mx
        mx ← −1
        j ← 1
        For i = 1, m execute
            If v[i] > mx then
                j ← i
                mx ← v[i]
            EndIf
        EndFor
        v[j] ← −1
    EndWhile
    Return x
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează cel mai mare număr care se poate obține folosind cifrele lui n.

B. Algoritmul returnează cea mai mare putere a lui 10 care divide numărul n.

C. Algoritmul returnează prima cifră din stânga a numărului n.

D. Algoritmul returnează suma cifrelor numărului n.

379. Se consideră algoritmul `f(n)`, unde parametrul n este număr natural ($1 \leq n \leq 1000^2$ la momentul apelului). ✓ ?

```
Algorithm F(n)
    z ← 0; p ← 1
    While n ≠ 0 execute
```

```
        c ← n MOD 10
        n ← n DIV 10
        If c MOD 3 = 0 then
            z ← z + p * (9 − c)
            p ← p * 10
        EndIf
    EndWhile
    Return z
EndAlgorithm
```

Care este valoarea returnată dacă algoritmul se apelează pentru $n = 103456$?

A. 639 B. 963 C. 693 D. 369

380. Se consideră algoritmul `f(n)` dat în enunțul problemei **379.**, dar acum parametrul n este număr natural cu două cifre ($10 \leq n \leq 99$ la momentul apelului). ✓ ?

Care dintre următoarele variante conțin doar numere pentru care algoritmul returnează valoarea 3?

A. 61, 65, 67 B. 62, 66, 68 C. 16, 56, 76 D. 26, 66, 86

381. Se dă algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale pozitive ($1 \leq a, b \leq 10000$). ✓ ?

```
Algorithm ceFace(a, b)
    For i ← 2, a, 2 execute
        If a MOD i = 0 then
            If b MOD i = 0 then
                write i
                write new line
            EndIf
        EndIf
    EndFor
EndAlgorithm
```

Dacă $a = 600$, precizați pentru care valori ale lui b se afișează 4 numere în urma executării algoritmului `ceFace(a, b)`:

A. $b = 20$ B. $b = 50$ C. $b = 12$ D. $b = 90$

382. Considerând algoritmul de la problema **381.**, precizați care dintre următoarele afirmații sunt adevărate: ✓ ?

A. Algoritmul afișează divizorii comuni ai numerelor a și b.

B. Algoritmul afișează divizorii proprii comuni ai numerelor a și b.

C. Algoritmul afișează divizorii impari comuni ai numerelor a și b.

D. Algoritmul afișează divizorii pari comuni ai numerelor a și b.

383. Fie un program care generează, în ordine crescătoare, toate numerele naturale de exact 5 cifre distincte care se pot forma cu cifrele 2, 3, 4, 5, 6. ✓ ?

Precizați numărul generat imediat înainte și numărul generat imediat după secvența următoare: 34256, 34265, 34526, 34562.

A. 32645 și 34625 B. 32654 și 34655 C. 32654 și 34625 D. 32645 și 34655

384. Fie șirul $x = (1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, ...)$, care se continuă conform regulii care se poate observa din elementele enumerate. ✓ ?

Considerând că primul element din șir este pe poziția 1, în care dintre următoarele subsecvențe va apărea doar valoarea 11?

A. $x[100], \ldots, x[109]$ C. $x[140], \ldots, x[152]$

B. $x[113], \ldots, x[120]$ D. $x[123], \ldots, x[132]$

385. Câte din primele 100 de elemente ale șirului x descris în problema **384.** sunt numere prime? ✓ ?

A. 4 B. 34 C. 36 D. 30

386. Se consideră numerele naturale a și n $(1 \leq a, n \leq 1000)$, vectorul V cu n elemente numere naturale $(V[1], V[2], ..., V[n])$ și algoritmii `one(a, n, V)` și `two(a, n, V)`: ✓ ?

```
Algorithm ONE(a, n, V)
    p ← 1; i ← 1
    While (i ≤ n) AND (a > V[p])
execute
        p ← p + 1
        i ← i + 1
    EndWhile
    Return p
EndAlgorithm
```

```
Algorithm TWO(a, n, V)
    p ← 1; i ← 1
    While i ≤ n execute
        If a > V[i] then
            p ← p + 1
        EndIf
        i ← i + 1
    EndWhile
    Return p
EndAlgorithm
```

Ce proprietate poate avea vectorul V, astfel încât, pentru orice n și V cu proprietatea dată, cei doi algoritmi să returneze valori egale pentru orice valoare a lui a?

A. În vectorul V toate elementele sunt egale.

B. În vectorul V toate elementele sunt distincte și sortate crescător.

C. În vectorul V toate elementele sunt distincte și sortate descrescător.

D. În vectorul V elementele sunt sortate crescător, dar nu sunt neapărat distincte.

387. Se consideră algoritmul `suma(n)` unde n este număr natural $(0 < n \leq 10000)$ la momentul apelului inițial). ✓ ?

```
Algorithm SUMA(n)
    If n = 0 then
        Return 0
    Else
        Return suma(n − 1) + n DIV (n + 1) + (n + 1) DIV n
    EndIf
EndAlgorithm
```

Care dintre afirmațiile de mai jos NU sunt adevărate?

A. Algoritmul returnează valoarea $n + 1$

B. Algoritmul calculează și returnează suma divizorilor proprii ai lui n

C. Apelul `suma(1)` returnează 2

D. Algoritmul calculează și returnează dublul părții întregi a mediei aritmetice a primelor n numere naturale

388. Fie următorul algoritm, având ca parametri de intrare numerele naturale a și b ✓ ? $(0 \leq a, b \leq 10000)$ la momentul apelului inițial):

```
Algorithm ceFace(a, b)
   While a * b ≠ 0 execute
      If a > b then
         Return ceFace(a MOD b, b)
      Else
         Return ceFace(a, b MOD a)
      EndIf
   EndWhile
   Return a + b
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează suma numerelor a și b.

B. Algoritmul returnează numărul nenul x în urma apelului `ceFace(x, 0)` sau `ceFace(0, x)`, respectiv 0 pentru `ceFace(0, 0)`.

C. Algoritmul returnează cel mai mare divizor comun al numerelor a și b.

D. Algoritmul returnează a la puterea b.

389. Se consideră algoritmul `afișare(n)` unde n este număr natural $(1 \leq n \leq 10^9)$: ✓ ?

```
Algorithm afișare(n)
   For i = 1, n - 1 execute
      For j = i + 1, n execute
         If (j - i) < (n DIV 2) then
            write i, " ", j - i
            write new line
         Else
            If (j - i) ≠ (n DIV 2) then
               write j - i, " ", i
               write new line
            EndIf
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

Câte perechi de numere se vor afișa în urma execuției algoritmului pentru $n = 7$?

A. 21 B. 15 C. 11 D. 17

390. Considerând secvența de cod de mai jos, determinați de câte ori se afișează șirul de ✓ ? caractere UBB, știind că $n = 3^k$, unde k este număr natural $(1 \leq k \leq 30)$?

```
j ← n
While j > 1 execute
   i ← 1
   While i ≤ n execute
      i ← 3 * i
      write 'UBB'
   EndWhile
   j ← j  DIV  3
EndWhile
```

A. k^2

B. $k * 3^k$

C. $k * (k + 1)$

D. $3 * k$

391. Se dau următoarele secvențe de cod și numerele naturale i, j, a, b $(1 < a, b \leq 10^9)$. ✓ ?

Secvența 1 (S1):

```
i ← 1
While i ≠ b execute
   j ← 1
   While j ≠ a execute
      write '*'
      j ← j + 1
   EndWhile
   i ← i + 1
EndWhile
```

Secvența 2 (S2):

```
i ← 1
While i ≠ a execute
   j ← 1
   While j ≠ b execute
      write '*'
      j ← j + 1
   EndWhile
   i ← i + 1
EndWhile
```

Care dintre afirmațiile următoare sunt adevărate?

A. Numărul de caractere afișate de secvența S1 este diferit față de numărul de caractere afișate de secvența S2.

B. Ambele secvențe au aceeași complexitate timp.

C. Numărul de caractere afișate de secvența S1 este $(a - 1) * (b - 1)$.

D. Numărul de caractere afișate de secvența S2 este $a * b$.

392. Se consideră algoritmul `ceFace(nr)`, unde nr este un număr natural $(100 \leq nr \leq 2 * 10^9$ la momentul apelului). ✓ ?

```
Algorithm testProprietateNr(n)
   If n ≤ 1 then
      Return False
   EndIf
   d ← 2
   While d * d ≤ n execute
      If n  MOD  d = 0 then
         Return False
      EndIf
      d ← d + 1
   EndWhile
   Return True
EndAlgorithm
```

```
Algorithm ceFace(nr)
   s ← 0
   c1 ← nr  MOD  10
   nr ← nr  DIV  10
   c2 ← nr  MOD  10
   nr ← nr  DIV  10
   While nr ≠ 0 execute
      c3 ← nr  MOD  10
      t ← c3 * 100 + c2 * 10 + c1
      If testProprietateNr(t) then
         s ← s + c1 + c2 + c3
      EndIf
      c1 ← c2
      c2 ← c3
      nr ← nr  DIV  10
   EndWhile
   Return s
EndAlgorithm
```

Precizați valoarea pe care o returnează algoritmul `ceFace(nr)` pentru $nr = 1271211312$?

A. 31 B. 32 C. 33 D. 34

393. Care dintre următorii algoritmi determină corect și returnează valoarea rădăcinii pătrate a numărului natural n ($0 < n < 10^5$), rotunjit în jos la cel mai apropiat întreg. Operatorul / reprezintă împărțirea reală (ex. $3/2 = 1,5$).

A.

```
Algorithm RADICAL_A(n)
    x ← 0
    z ← 1
    While z ≤ n execute
        x ← x + 1
        z ← z + 2 * x
        z ← z + 1
    EndWhile
    Return x
EndAlgorithm
```

B.

```
Algorithm RADICAL_B(n)
    s ← 1
    d ← n  DIV  2
    While s < d execute
        k ← (s + d)  DIV  2
        If k * k ≥ n then
            d ← k
        Else
            s ← k + 1
        EndIf
    EndWhile
    If s * s ≤ n then
        Return s + 1
    Else
        Return s − 1
    EndIf
EndAlgorithm
```

C.

```
Algorithm RADICAL_C(n, x)
    //Algoritmul se apelează inițial
    //în forma radical_C(n, n)
    eps ← 0.001
    y ← 0.5 * (x + n/x)
    If x − y < eps then
        //se returnează partea
        //întreagă a lui x
        Return [x]
    EndIf
    Return radical_C(n, y)
EndAlgorithm
```

D.

```
Algorithm RADICAL_D(n)
    s ← 0
    p ← 0
```

```
    k ← 0
    While k < n execute
        k ← k + 3 + p
        p ← p + 2
        s ← s + 1
    EndWhile
    Return s
EndAlgorithm
```

394. Știind că x este număr natural, care dintre următoarele expresii au valoarea *True* dacă și numai dacă x este număr par care NU aparține intervalului deschis (10, 20)?

A. **NOT** $((x > 10)$ **AND** $(x < 20))$ **AND** (**NOT** $(x$ **MOD** $2 = 1))$

B. $(x$ **MOD** $2 = 0)$ **AND** $((x < 10)$ **OR** $(x > 20))$

C. **NOT** $(x$ **MOD** $2 = 1)$ **AND** $((x > 10)$ **AND** $(x < 20))$

D. **NOT** $((x$ **MOD** $4 = 1)$ **OR** $(x$ **MOD** $4 = 3)$ **OR** $((x > 10)$ **AND** $(x < 20)))$

395. Se dă un șir a de n numere naturale distincte $(a[1], a[2], ..., a[n], 2 \leq n \leq 1000)$ ordonate strict crescător. Într-un șir se numește vârf local un număr cu proprietatea că este strict mai mare decât numărul de pe poziția anterioară, dar și decât numărul de pe poziția următoare. Primul și ultimul element din șir nu pot fi vârfuri locale. Se dorește un algoritm `rearanjare(a, n)` care rearanjează valorile din șir astfel încât șirul să aibă un număr maxim de vârfuri locale și returnează noul șir. Variabila locală b este un șir. Care dintre următorii algoritmi sunt corecți?

A.

```
Algorithm REARANJARE(a, n)
    i ← n
    For p ← 2, n, 2 execute
        b[p] ← a[i]
        i ← i − 1
    EndFor
    For p ← 1, n, 2 execute
        b[p] ← a[i]
        i ← i − 1
    EndFor
    Return b
EndAlgorithm
```

B.

```
Algorithm REARANJARE(a, n)
    i ← n
    For p ← 2, n, 2 execute
        b[p] ← a[i]
        i ← i − 1
        b[p − 1] ← a[i]
        i ← i − 1
    EndFor
    If n MOD 2 = 1 then
        b[n] ← a[i]
    EndIf
    Return b
EndAlgorithm
```

C.

```
Algorithm REARANJARE(a, n)
    i ← n
    For p ← 2, n, 2 execute
        b[p] ← a[i]
        i ← i − 1
    EndFor
    i ← 1
    For p ← 1, n, 2 execute
        b[p] ← a[i]
        i ← i + 1
    EndFor
    Return b
EndAlgorithm
```

D.

```
Algorithm REARANJARE(a, n)
    i ← n
    For p ← 2, n, 3 execute
        b[p] ← a[i]
        i ← i − 1
        b[p − 1] ← a[i]
        i ← i − 1
        If p + 1 ≤ n then
            b[p + 1] ← a[i]
            i ← i − 1
        EndIf
    EndFor
    If n MOD 3 = 1 then
        b[n] ← a[i]
    EndIf
    Return b
EndAlgorithm
```

396. Se consideră algoritmul `f(n, p1, p2)`, unde n, *p1* și *p2* sunt numere naturale strict pozitive ($1 < n, p1, p2 \leq 10^4$) la momentul apelului). ✓ ?

```
Algorithm F(n, p1, p2)
    c ← 0
    While p1 ≤ n execute
        c ← c + n DIV p1
        p1 ← p1 ∗ p2
    EndWhile
    Return c
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă cei trei parametri au valori egale ($n = p1 = p2$), atunci algoritmul returnează întotdeauna valoarea 1.

B. Dacă $p1 = 5$ și $p2 = 5$, algoritmul returnează numărul de cifre de 0 pe care le are $n!$ la sfârșit.

C. Dacă *p1* și *p2* au valori egale și mai mari decât 2, atunci algoritmul returnează $[\log_{p1} n]$.

D. Niciuna dintre celelalte trei afirmații nu este adevărată.

397. Care dintre următorii algoritmi returnează numărul de numere sumative din intervalul $[a, b]$ ($0 < a < b < 10^6$)? Un număr natural nenul n este sumativ dacă n^2 se poate scrie ca sumă a n numere naturale nenule consecutive. De exemplu, 1 și 7 sunt sumative deoarece $1 = 1$, respectiv $49 = 4 + 5 + 6 + 7 + 8 + 9 + 10$. ✓ ?

A.

```
Algorithm SUMATIVE(a, b)
    k ← 0
    For i ← a, b execute
        If i MOD 2 ≠ 0 then
            k ← k + 1
        EndIf
```

```
   EndFor
   Return k
EndAlgorithm
```

B.

```
Algorithm SUMATIVE(a, b)
   Return (b − a) DIV 2 + (b − a) MOD 2 + (a MOD 2 + b MOD 2) DIV 2
EndAlgorithm
```

C.

```
Algorithm SUMATIVE(a, b)
   k ← 0
   For i ← a, b execute
      i2 ← i ∗ i
      For j ← 2, i − 1 execute
         If i2 = j ∗ i + (i ∗ (i + 1) DIV 2) then
            k ← k + 1
         EndIf
      EndFor
   EndFor
   Return k
EndAlgorithm
```

D.

```
Algorithm SUMATIVE(a, b)
   k ← 0
   For i ← a, b execute
      i2 ← i ∗ i
      For j ← 2, i DIV 2 execute
         If i2 = j ∗ i + (i ∗ (i + 1) DIV 2) then
            k ← k + 1
         EndIf
      EndFor
   EndFor
   Return k
EndAlgorithm
```

Admitere nivel licență, sesiunea iulie 2022

398. Se consideră algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10000$) la momentul apelului). ✓ ?

```
Algorithm ceFace(a, b)
   While (a MOD 10 = b MOD 10) AND (a ≠ 0) AND (b ≠ 0) execute
      a ← a DIV 10
      b ← b DIV 10
   EndWhile
   If (a = 0 AND b = 0) then
      Return True
   Else
      Return False
   EndIf
EndAlgorithm
```

Algoritmul `ceFace(a, b)` returnează *True* dacă și numai dacă:

A. numerele a și b au același număr de cifre

B. a și b sunt egale

C. a și b sunt formate din aceleași cifre, dar asezate în altă ordine

D. ultima cifră a lui a este egală cu ultima cifră a lui b

399. Se consideră algoritmul `f(a, n)` unde n este număr natural nenul ($2 \leq n \leq 10000$) și a este un vector cu n numere întregi ($a[1], a[2], ..., a[n], -100 \leq a[i] \leq 100$, pentru $i = 1, 2, ..., n$). Variabila locală b este vector. ✓ ?

```
Algorithm f(a, n)
   i ← 2
   b[1] ← a[1]
   While i ≤ n execute
      b[i] ← b[i − 1] + a[i]
      i ← i + 1
   EndWhile
   Return b[n]
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează suma tuturor elementelor din vectorul a.

B. Algoritmul returnează suma ultimelor două elemente din vectorul a.

C. Algoritmul returnează ultimul element din vectorul a.

D. Algoritmul returnează suma ultimelor n - 1 elemente din vectorul a.

400. Care dintre algoritmii următori returnează numărul factorilor primi distincți ai unui număr natural n dat ($5 < n < 10^5$) la momentul apelului). ✓ ?

A.

```
// Vectorul prime are lungimea n
// prime[i] are valoarea True, dacă
// numărul i este prim și False altfel
Algorithm nrFactoriPrimi_A(n, prime)
   d ← 2
   nr ← 0
```

```
        p ← 0
        While n > 0 execute
            While n MOD d = 0 execute
                p ← p + 1
                n ← n DIV d
            EndWhile
            If p ≠ 0 then
                nr ← nr + 1
            EndIf
            d ← d + 1
            While prime[d] = False execute
                d ← d + 1
            EndWhile
            p ← 0
        EndWhile
        Return nr
    EndAlgorithm
```

B.

```
    Algorithm nrFactoriPrimi_B(n)
        d ← 2
        nr ← 0
        While n > 1 execute
            p ← 0
            While n MOD d = 0 execute
                p ← p + 1
                n ← n DIV d
            EndWhile
            If p > 0 then
                nr ← nr + 1
            EndIf
            If d = 2 then
                d ← d + 1
            Else
                d ← d + 2
            EndIf
        EndWhile
        Return nr
    EndAlgorithm
```

C.

```
    Algorithm nrFactoriPrimi_C(n)
        nr ← 0
        For d ← 2, n execute
            If n MOD d = 0 then
                nr ← nr + 1
            EndIf
            While n MOD d = 0 execute
                n ← n DIV d
            EndWhile
        EndFor
        Return nr
    EndAlgorithm
```

D.

```
    Algorithm nrFactoriPrimi_D(n)
```

```
        nr ← 0
        d ← 2
        While d * d ≤ n execute
            If n MOD d = 0 then
                nr ← nr + 1
            EndIf
            While n MOD d = 0 execute
                n ← n DIV d
            EndWhile
            d ← d + 1
        EndWhile
        Return nr
    EndAlgorithm
```

401. Se consideră algoritmul `ceFace(n, m)`, unde n este număr natural ($0 \leq n \leq 1000$) cu ultima cifră diferită de 0. ✓ ?

```
Algorithm ceFace(n, m)
    If n = 0 then
        Return m
    Else
        Return ceFace(n DIV 10, m * 10 + n MOD 10)
    EndIf
EndAlgorithm
```

Care este rezultatul apelului `ceFace(n, 0)`?

A. 0 (indiferent de valoarea lui n)

B. n (indiferent de valoarea lui n)

C. Suma cifrelor numărului n

D. Oglinditul numărului n

402. Se consideră algoritmul `f(x, n)` unde n este număr natural ($2 \leq n \leq 10000$), iar x este un șir de n numere naturale ($x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10000$), pentru $i = (1, 2, ..., n)$. ✓ ?

```
Algorithm f(x, n)
    For i = 1, n - 1 execute
        If x[i] = x[i + 1] then
            Return False
        EndIf
    EndFor
    Return True
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează *False* dacă două elemente oarecare din șirul x sunt distincte.

B. Algoritmul returnează *False* dacă două elemente oarecare din șirul x sunt egale.

C. Algoritmul returnează *False* dacă două elemente consecutive din șirul x sunt egale.

D. Algoritmul returnează *False* dacă primele două elemente din șirul x sunt egale.

403. Se consideră algoritmul `f(x, n)` unde x și n sunt numere naturale ($0 \leq n \leq 10000, 0 < x \leq 10000$). ✓ ?

```
1: Algorithm F(x, n)
2:     If n = 0 then
3:         Return 1
4:     EndIf
5:     m ← n DIV 2
6:     p ← f(x, m)
7:     If n MOD 2 = 0 then
8:         Return p * p
9:     EndIf
10:    Return x * p * p
11: EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează x la puterea n.

B. Dacă pe linia 7, în loc de n **MOD** 2 ar fi m **MOD** 2, atunci algoritmul ar returna x la puterea n.

C. Din cauza autoapelului de pe linia 6, liniile 7, 8, 9, 10 nu se vor executa niciodată.

D. Algoritmul returnează 1 dacă n este număr par sau x dacă n este număr impar.

404. Considerând că toate operațiile de înmulțire și împărțire se realizează în timp constant, ce putem spune despre complexitatea timp a algoritmului din enunțul problemei **403.**? ✓ ?

A. Complexitatea timp depinde de parametrii x și n.

B. Complexitatea timp nu depinde de parametrul x.

C. Complexitatea timp este $O(\log \log n)$.

D. Complexitatea timp este logaritmică în raport cu parametrul n ($O(\log n)$).

405. Se consideră algoritmul `afișare(n)`, unde n este număr natural ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm AFIȘARE(n)
   If n ≤ 4000 then
      write n, " "
      afișare(2 * n)
      write n, " "
   EndIf
EndAlgorithm
```

Ce se afișează pentru apelul `afișare(1000)`?

A. 1000 2000 4000

B. 1000 2000 4000 4000 2000 1000

C. 1000 2000 4000 2000 1000

D. 1000 2000 2000 1000

406. Care ar putea fi elementele unui vector astfel încât, aplicând metoda căutării binare pentru valoarea 36, aceasta să fie comparată succesiv cu valorile 12, 24, 36: ✓ ?

A. $[2, 4, 7, 12, 24, 36, 50]$
B. $[2, 4, 8, 9, 12, 16, 20, 24, 36, 67]$
C. $[4, 8, 9, 12, 16, 24, 36]$
D. $[12, 24, 36, 42, 54, 66]$

407. Care dintre următoarele expresii sunt echivalente cu x **MOD** y pentru toate numerele naturale strict pozitive x și y $(0 < x, y \leq 10000)$? ✓ ?

A. x **DIV** y
B. $x - (y * (x$ **DIV** $y))$
C. $x - (x * (x$ **DIV** $y))$
D. x **DIV** $y + y$ **DIV** x

408. Fie variabila n care memorează un număr natural. Care dintre expresiile de mai jos are valoarea *True* dacă și numai dacă n este divizibil cu 2 și cu 3? ✓ ?

A. $(n$ **DIV** $2 = 0)$ **OR** $(n$ **DIV** $3 \neq 0)$
B. $(n$ **MOD** $3 = 2)$ **OR** $(n$ **MOD** $2 = 3)$
C. $(n$ **MOD** $2 \neq 1)$ **AND** $(n$ **MOD** $3 = 0)$
D. $(n$ **MOD** $2 = 0)$ **AND** $(n$ **MOD** $3 \neq 1)$

409. Fie variabila n care memorează un număr natural. Care dintre expresiile de mai jos are valoarea *True* dacă și numai dacă n este divizibil cu 2 și cu 3? ✓ ?

A. $(n$ **MOD** $2) - (n$ **MOD** $3) = 0$
B. $(n$ **MOD** $2) - (n$ **MOD** $3) < 0$
C. $(n$ **MOD** $2) + (n$ **MOD** $3) > 0$
D. $(n$ **MOD** $2) + (n$ **MOD** $3) = 0$

410. Se consideră algoritmul `f(n)`, unde n este număr natural $(1 \leq n \leq 100)$. Operatorul "/" reprezintă împărțirea reală (ex. $3/2 = 1,5$). Precizați efectul algoritmului. ✓ ?

```
Algorithm F(n)
    s ← 0; p ← 1
    For i ← 1, n execute
        s ← s + i
        p ← p * (1/s)
    EndFor
    Return p
EndAlgorithm
```

A. Evaluează expresia $1/1 * 1/2 * 1/3 * ... * 1/n$
B. Evaluează expresia $1/1 * 1/(1 * 2) * 1/(1 * 2 * 3) * ... * 1/(1 * 2 * 3 * ... * n)$
C. Evaluează expresia $1/1 * 1/(1 + 2) * 1/(1 + 2 + 3) * ... * 1/(1 + 2 + 3 + ... + n)$
D. Evaluează expresia $1/1 + 1/(1 * 2) + 1/(1 * 2 * 3) + ... + 1/(1 * 2 * 3 * ... * n)$

411. Se consideră algoritmul `prelucrare(s1, lung1, s2, lung2)`, unde *s1* și *s2* sunt două șiruri de caractere de lungime *lung1*, respectiv *lung2* $(1 \leq lung1, lung2 \leq 1000)$. Cele două șiruri conțin doar caractere, având codul ASCII din intervalul [1, 125]. Variabila locală x este vector. Considerăm algoritmul `ascii(s, i)` care returnează codul ASCII al celui de-al i-lea caracter al șirului de caractere s. ✓ ?

```
Algorithm PRELUCRARE(s1, lung1, s2, lung2)
    For i = 1, 125 execute
        x[i] ← 0
    EndFor
```

```
    For i = 1, lung1 execute
        x[ascii(s1, i)] ← x[ascii(s1, i)] + 1
    EndFor
    For i = 1, lung2 execute
        x[ascii(s2, i)] ← x[ascii(s2, i)] − 1
    EndFor
    ok ← True
    For i = 1, 125 execute
        If x[i] ≠ 0 then
            ok ← False
        EndIf
    EndFor
    Return ok
EndAlgorithm
```

Precizați efectul algoritmului.

A. Algoritmul returnează *True* dacă șirurile de caractere *s1* și *s2* au aceeași lungime și *False* în caz contrar.

B. Algoritmul returnează *True* dacă șirurile de caractere *s1* și *s2* sunt formate din aceleași caractere având aceleași frecvențe corespunzătoare, și *False* în caz contrar.

C. Algoritmul returnează *True* dacă în fiecare dintre cele două șiruri de caractere *s1* și *s2* apar toate caracterele având codul ASCII din intervalul [1, 125] și *False* în caz contrar.

D. Algoritmul returnează *True* dacă cele două șiruri de caractere *s1* și *s2* sunt formate din caractere diferite și *False* în caz contrar.

412. Care este rezultatul conversiei numărului binar 1001011001111 în baza 10? ✓ ?

A. 2407

B. 2408

C. 1203

D. Niciunul dintre A., B., C.

413. Se consideră un vector a cu n numere naturale $(a[1], a[2], ..., a[n])$, numărul natural n $(1 \leq n \leq 10000)$ și un număr natural x. Care din următoarele secvențe de cod afișează poziția cu indicele minim unde se află valoarea x în vectorul a, sau afișează -1 dacă x nu apare în vectorul a? ✓ ?

A.

```
i ← 1
While (i ≤ n) AND
      (a[i] = x) execute
    i ← i + 1
EndWhile
If i ≤ n then
    write i
Else
    write −1
EndIf
```

B.

```
i ← 1
While (i ≤ n) AND
      (a[i] ≠ x) execute
    i ← i + 1
EndWhile
If i = n + 1 then
    write i
Else
    write −1
EndIf
```

C.

```
i ← 1
While (i ≤ n) AND
      (a[i] = x) execute
   i ← i + 1
EndWhile
If i = n + 1 then
   write i
Else
   write −1
EndIf
```

D.

```
i ← 1
While (i ≤ n) AND
      (a[i] ≠ x) execute
   i ← i + 1
EndWhile
If i ≤ n then
   write i
Else
   write −1
EndIf
```

414. Se consideră algoritmul `f(x)`, unde x este număr întreg: ✓ ?

```
Algorithm F(x)
   If x = 0 then
      Return 0
   Else
      If x MOD 3 = 0 then
         Return f(x DIV 10) + 1
      Else
         Return f(x DIV 10)
      EndIf
   EndIf
EndAlgorithm
```

Pentru ce valoare a lui x algoritmul va returna valoarea 4?

A. 13369

B. 21369

C. 4

D. 1233

415. Se consideră algoritmul `f(n, i, j)` unde n, i și j sunt numere naturale ($1 \leq n, i, j \leq 10000$ la momentul apelului inițial). ✓ ?

```
Algorithm F(n, i, j)
   If i > j then
      write '*'
   Else
      If n MOD i = 0 then
         f(n, i − 1, j)
      Else
         If n DIV i ≠ j then
            f(n, i + 1, j − 1)
            write '0'
         Else
            f(n, i + 2, j − 2)
            write '#'
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

Ce se afișează în urma execuției apelului `f(15, 3, 10)`?

A. *000000

B. *0#000

C. *0#0000

D. *0000000

416. Se consideră algoritmul `ceFace(n, x)`, unde n este număr natural ($1 \leq n \leq 100$) și x este un vector cu n elemente numere naturale ($x[1], x[2], ..., x[n]$). ✓ ?

```
Algorithm CEFACE(n, x)
   For i = 1, n execute
      c ← x[i]
      x[i] ← x[n − i + 1]
```

```
        x[n − i + 1] ← c
    EndFor
EndAlgorithm
```

Care va fi noul conținut al vectorului x după executarea algoritmului dat dacă $n = 6$ și $x = [5, 3, 2, 1, 1, 1]$?

A. $[1, 1, 2, 1, 3, 5]$
B. $[1, 1, 1, 2, 3, 5]$
C. $[5, 3, 2, 1, 1, 1]$
D. Niciuna dintre variante.

417. Se consideră algoritmul `what(n)`, unde n este număr natural ($1 \leq n \leq 1000$ la apelul inițial). ✓ ?

```
Algorithm WHAT(n)
    If n = 0 then
        Return True
    EndIf
    If (n MOD 10 = 3) OR (n MOD 10 = 7) then
        Return what(n DIV 10)
    Else
        Return False
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează *True* dacă și numai dacă fie n este format doar din cifre de 3, fie n este format doar din cifre de 7
B. Algoritmul returnează *False* dacă n conține cel puțin o cifră pară
C. Algoritmul returnează *False* dacă și numai dacă n conține cel puțin o cifră c unde $c \neq 3$ și $c \neq 7$
D. Algoritmul returnează *True* dacă și numai dacă n nu conține nicio cifră din mulțimea $\{0, 1, 2, 4, 5, 6, 8, 9\}$

418. Se consideră algoritmul `calcul(x, n)`, unde x și n sunt numere naturale ($1 \leq x \leq 10000, 1 \leq n \leq 10000$), și $x \leq n$. ✓ ?

```
Algorithm CALCUL(x, n)
    b ← 1
    For i ← 1, n − x execute
        b ← b ∗ i
    EndFor
    a ← b
    For i ← n − x + 1, n execute
        a ← a ∗ i
    EndFor
    Return a DIV b
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $x = 2$ și $n = 5$, atunci algoritmul returnează 10.
B. Algoritmul returnează numărul acelor submulțimi ale mulțimii $\{1, 2, ..., n\}$ care au câte x elemente.

C. Algoritmul returnează numărul aranjamentelor de n elemente, luate câte x.

D. Algoritmul returnează numărul combinărilor de n elemente, luate câte x.

419. O fermă crește găini și iepuri, fiecare găină având două picioare și fiecare iepure patru picioare. Numărul total de capete este n și numărul total de picioare al animalelor din fermă este m ($0 \leq n, m \leq 10^4$). Care dintre următorii algoritmi returnează *True* și afișează toate perechile de numere posibile pentru numărul găinilor și al iepurilor din fermă, sau returnează *False* dacă nu există soluție? ✓ ?

A.
```
Algorithm FERMA_A(n, m)
    found ← False
    For i ← 0, n execute
        j ← n − i
        If 2 * i + 4 * j = m then
            found ← True
            write i, ' ', j
            write newline
        EndIf
    EndFor
    Return found
EndAlgorithm
```

B.
```
Algorithm FERMA_B(n, m)
    found ← False
    For i ← 0, n execute
        For j ← 0, n execute
            If 2 * i + 4 * j = m  AND
                 i + j = n then
                found ← True
                write i, ' ', j
                write newline
            EndIf
        EndFor
    EndFor
    Return found
EndAlgorithm
```

C.
```
Algorithm FERMA_C(n, m)
    found ← False
    For i ← 0, n execute
        For j ← 0, n − i execute
            If 2 * i + 4 * j = m  AND
                 i + j = n then
                found ← True
                write i, ' ', j
                write newline
            EndIf
        EndFor
    EndFor
    Return found
EndAlgorithm
```

D.
```
Algorithm FERMA_D(n, m)
    found ← False
    For i ← 0, n execute
        For j ← 0, i execute
            If 2 * i + 4 * j = m  AND
                 i + j = n then
                found ← True
                write i, ' ', j
                write newline
            EndIf
        EndFor
    EndFor
    Return found
EndAlgorithm
```

420. Se dă un număr natural n, care poate fi scris ca produs de trei numere naturale a, b, c, ($n = a * b * c$). Care dintre următoarele expresii are ca valoare restul împărțirii lui n la numărul natural d ($1 \leq n, a, b, c, d \leq 10000$)? ✓ ?

A. $(a \ \textbf{MOD} \ d) * b * c$

B. $((a \ \textbf{MOD} \ d) * (b \ \textbf{MOD} \ d) * (c \ \textbf{MOD} \ d)) \ \textbf{MOD} \ d$

C. $(a \ \textbf{MOD} \ d) * (b \ \textbf{MOD} \ d) * (c \ \textbf{MOD} \ d)$

D. $(a \ \textbf{DIV} \ d) * (b \ \textbf{DIV} \ d) * (c \ \textbf{DIV} \ d)$

421. Se consideră algoritmul `det(a, n, m)`, unde a este un șir de n numere naturale ($a[1], a[2], ..., a[n]$ dacă $n \geq 1$) sau șir vid dacă $n = 0$. n și m sunt numere naturale ($0 \leq n < 100, 0 \leq m < 10^6$). ✓ ?

```
1: Algorithm DET(a, n, m)
2:     For i ← 1, n − 1 execute
3:         For j ← i + 1, n execute
4:             If a[i] > a[j] then
5:                 tmp ← a[i]
6:                 a[i] ← a[j]
7:                 a[j] ← tmp
8:             EndIf
9:         EndFor
10:    EndFor
11:    i ← 1
12:    j ← n
13:    b ← False
14:    While i < j execute
15:        If a[i] + a[j] = m then
16:            b ← True
17:        EndIf
18:        If a[i] + a[j] < m then
19:            i ← i + 1
20:        Else
21:            j ← j − 1
22:        EndIf
23:    EndWhile
24:    Return b
25: EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează *True* dacă în șirul a există o pereche de numere care au suma egală cu m.

B. Algoritmul returnează întotdeauna *False*.

C. Algoritmul returnează *False* dacă $n = 0$.

D. În liniile 2, ..., 10 algoritmul sortează crescător șirul a.

422. Se consideră algoritmul `magic(n, a)`, unde a este un vector cu n numere naturale ✓ ? $(a[1], a[2], ..., a[n], 1 \leq n \leq 10000)$.

```
Algorithm MAGIC(n, a)
    If n < 2 then
        Return False
    EndIf
    For i ← 2, n execute
        If a[i − 1] = a[i]
then
            Return True
        EndIf
    EndFor
    Return False
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru `magic(5, [2, 5, 4, 5, 4])` algoritmul returnează *False*.

B. Algoritmul indică dacă există duplicate în șirul a, dacă și numai dacă vectorul a este sortat crescător/descrescător.

C. Pentru `magic(9, [1, 2, 3, 4, 4, 5, 6, 7, 9])` algoritmul returnează *True*.

D. Pentru `magic(5, [9, 5, 5, 2, 4])` algoritmul returnează *True*.

423. Fie algoritmul `f(n, a, b, c)` unde n este număr natural ($n \leq 20$) și a, b, c trei ✓ ? numere întregi.

```
Algorithm F(n, a, b, c)
    If n = 0 then
        Return 1
    Else
        Return f(n − 1, a ∗ a, b + 1, c ∗ 2)+ f(n − 1, a − 1, b ∗ b, c + 1) + 1
    EndIf
EndAlgorithm
```

Care este rezultatul returnat la apelul `f(n, 1, 1, 2)`?

A. $2^{n+1}-1$
C. $2^0+2^1+2^2+...+2^n$
B. n
D. 2^{n+1}

424. Se consideră algoritmii `f(n, p)` și `g(n)`, unde n și p sunt inițial numere naturale ✓ ? ($1 \leq n, p \leq 10^6$ la momentul apelului inițial).

```
Algorithm g(n)
   If n < 2 then
      Return False
   EndIf
   i ← 2
   While i * i ≤ n execute
      If n MOD i = 0 then
         Return False
      EndIf
      i ← i + 1
   EndWhile
   Return True
EndAlgorithm
```

```
Algorithm f(n, p)
   If n = 0 then
      Return 1
   EndIf
   If n > 0 AND n ≥ p then
      c ← 0
      If g(p) = True then
         c ← c+ f(n − p, p + 1)
      EndIf
      Return c+ f(n, p + 1)
   EndIf
   Return 0
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul `g(n)` returnează *True* dacă numărul n este prim și *False* în caz contrar.

B. Pentru apelul `f(n, 2)` se returnează numărul de moduri diferite în care numărul n poate fi scris ca sumă de cel puțin un termen de numere prime distincte în ordine strict crescătoare.

C. Pentru apelul `f(n, 2)` se returnează suma divizorilor primi ai numărului n.

D. Apelurile `f(n, 1)` și `f(n, 2)` vor returna același rezultat, oricare ar fi n.

425. Se consideră algoritmul `AlexB(value, n, k, p)`, unde *value* este un șir cu n numere naturale $(value[1], value[2], ..., value[n])$, iar n, k și p sunt numere naturale. Inițial șirul *value* are n elemente egale cu zero. Algoritmul `afișare(value, n)` afișează pe o linie șirul *value*. ✓ ?

```
Algorithm AlexB(value, n, k, p)
   p ← p + 1
   value[k] ← p
   If p = n then
      afișare(value, n)
   Else
      For i ← 1, n execute
         If value[i] = 0 then
            AlexB(value, n, i, p)
         EndIf
      EndFor
   EndIf
   p ← p − 1
   value[k] ← 0
EndAlgorithm
```

Precizați șirul afișat pe a zecea linie, dacă $n = 5$ și algoritmul se apelează sub forma `AlexB(value, 5, 1, 0)`.

A. 1 5 2 3 4

B. 1 5 4 0 4

C. 5 5 5 5 5

D. 1 2 5 4 3

426. Se consideră algoritmul `f(n)` unde n este număr natural ($1 \leq n \leq 10000$) la momentul apelului). ✓ ?

```
Algorithm F(n)
   c ← 0
   If n ≠ 0 then
      c ← c + 1
      n ← n & (n−1) // and pe biți
      While n ≠ 0 execute
         c ← c + 1
         n ← n & (n − 1) // and pe
biți
      EndWhile
   EndIf
   Return c
EndAlgorithm
```

Operatorul & este operatorul AND pe biți; tabelul de adevăr este următorul:

&	0	1
0	0	0
1	0	1

✓ ?

Exemplu: 2 & 7 convertit în binar: 010 & 111 = 010 care este 2 în baza 10. 6 & 1 convertit în binar: 110 & 001 = 000 care este 0 în baza 10.

Care dintre afirmațiile de mai jos **NU** sunt adevărate?

A. Dacă n este o putere a lui 2, atunci `f(n)` returnează valoarea 1.

B. Dacă $n > 16$ și $n < 32$, atunci valoarea returnată de `f(n)` aparține mulțimii $\{2, 3, 4, 5\}$.

C. Algoritmul returnează numărul de numere pare strict mai mici decât n.

D. Algoritmul returnează numărul de numere impare mai mici decât n.

427. Se consideră algoritmul `calcul(v, n)`, unde n este număr natural nenul ($1 \leq n \leq 10000$) și v este un șir cu n numere întregi $(v[1], v[2], ..., v[n])$. Instrucțiunea `return x, y` returnează perechea de valori (x, y). ✓ ?

```
Algorithm CALCUL(v, n)
   i ← n DIV 2 + 1
   j ← i + 1
   k ← i
   p ← j
   While j ≤ n execute
      While (j ≤ n) AND (v[i] =
v[j]) execute
         j ← j + 1
      EndWhile
      If j − i > p − k then
         k ← i
         p ← j
      EndIf
      i ← j
      j ← j + 1
   EndWhile
   If j − i > p − k then
      k ← i
      p ← j
   EndIf
   Return p − k, k
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă șirul are un singur element, algoritmul returnează valorile 0, -1

B. Dacă $n = 2$ și cele două elemente ale șirului sunt simetrice față de 0 (de ex. -5, 5), rezultatul va fi -1, 1

C. Dacă $n = 2$ și cele două elemente ale șirului au valori consecutive (de ex. 3, 4), se returnează întotdeauna valorile 1, 2

D. Unul dintre numerele returnate de algoritm reprezintă lungimea celei mai lungi secvențe cu valori egale, din a doua jumătate a șirului, pentru orice $n > 1$ număr par

Admitere nivel licență, sesiunea septembrie 2021

428. Se consideră subalgoritimul `ceFace(n)`, unde n este un număr natural ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm ceFace(n)
    nr ← 0
    For d ← 1, n execute
        If n MOD d = 0 then
            nr ← nr + 1
        EndIf
    EndFor
    If nr = 2 then
        Return adevărat
    Else
        Return fals
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritimul returnează *adevărat* dacă numărul n este impar.

B. Subalgoritimul returnează *adevărat* dacă numărul n este par.

C. Subalgoritimul returnează *adevărat* dacă numărul n este prim.

D. Subalgoritimul returnează *adevărat* dacă numărul n este pătrat perfect.

429. Știind că $x < y$ (x și y sunt numere reale), care din următoarele expresii are valoarea *adevărat* dacă și numai dacă numărul memorat în t (t număr real) NU aparține intervalului (x, y)? ✓ ?

A. $(t > x)$ **SAU** $(t < y)$

B. $(t \leq x)$ **SAU** $(t \geq y)$

C. $(t \leq x)$ **ȘI** $(t \geq y)$

D. $(t > x)$ **ȘI** $(t < y)$

430. Fie subalgoritimul `f(n)` unde n este un număr natural ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm f(n)
    r ← 0
    While n > 0 execute
        r ← r + (n MOD 10) * (n MOD 2)
        n ← n DIV 10
    EndWhile
    Return r
EndAlgorithm
```

Alegeți variantele care completează corect spațiul subliniat din subalgoritimul de mai jos astfel încât cei doi subalgoritmi să returneze mereu aceeași valoare.

```
Algorithm fr(n)
    If n > 0 then
        Return _____________
    EndIf
    Return 0
EndAlgorithm
```

A. $(n$ **MOD** $2) * (n$ **MOD** $10) +$ `fr`$(n$ **DIV** $10)$

B. $(n$ **MOD** $2) * (n$ **MOD** $10) *$ `fr`$(n$ **DIV** $10)$

C. $(n$ **MOD** $10) +$ `fr`$(n$ **DIV** $10)$

D. $(n$ **MOD** $2) * (n$ **MOD** $10) +$ `fr`$(n$ **MOD** $10)$

431. Fie subalgoritmul `f(n)` unde n este un număr natural ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm F(n)
   For i ← 1, n execute
      For j ← 1, 2 * i − 1 execute
         write '*'
      EndFor
   EndFor
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $n = 3$ subalgoritmul afișează 3 steluțe

B. Pentru $n = 3$ subalgoritmul afișează 9 steluțe

C. Pentru ca subalgoritmul să afișeze 1154 de steluțe valoarea lui n trebuie să fie 34

D. Pentru ca subalgoritmul să afișeze 289 de steluțe valoarea lui n trebuie să fie 17

432. Subalgoritmul de mai jos are ca parametri de intrare un vector v cu n numere naturale ($v[1], v[2], ..., v[n]$) și numărul întreg n ($2 \leq n \leq 10000$). Operatorul / reprezintă împărțirea reală (ex. $3/2 = 1,5$). Vectorul v conține cel puțin un număr par și cel puțin un număr impar. ✓ ?

```
Algorithm FN(v, n)
   a ← 0
   b ← 0
   For i ← 1, n execute
      If v[i] MOD 2 = 0 then
         a ← a + v[i]
         b ← b + 1
      EndIf
   EndFor
   Return a/b
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează numărul de elemente pare din vectorul v

B. Subalgoritmul returnează media elementelor pare din vectorul v

C. Subalgoritmul returnează suma elementelor pare din vectorul v

D. Subalgoritmul returnează media elementelor impare din vectorul v

433. Subalgoritmul de mai jos are ca parametri de intrare un vector v cu n numere naturale ($v[1], v[2], ..., v[n]$) și numărul întreg n ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm FN(v, n)
   a ← 0
   For i ← 1, n execute
      ok ← 1
      b ← v[i]
      While (b ≠ 0) SI (ok = 1) execute
```

```
            If b MOD 2 = 0 then
                ok ← 0
            EndIf
            b ← b DIV 10
        EndWhile
        If ok = 1 then
            a ← a + v[i]
        EndIf
    EndFor
    Return a
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritimul returnează suma elementelor impare din vectorul v

B. Subalgoritimul returnează suma elementelor din vectorul v care sunt puteri ale lui 2

C. Subalgoritimul returnează suma elementelor din vectorul v care au în componența lor doar cifre pare

D. Subalgoritimul returnează suma elementelor din vectorul v care au în componența lor doar cifre impare

434. Precizați care dintre următorii subalgoritmi calculează modulul (valoarea absolută) unui număr întreg. Vom presupune că o expresie logică are valoarea 1 dacă este adevărată și 0 dacă este falsă. ✓ ?

A.

```
Algorithm MODUL(n)
    Return n * (-2 * (n < 0) + 1)
EndAlgorithm
```

B.

```
Algorithm MODUL(n)
    If n < 0 then
        Return n * (-1)
    EndIf
    Return n
EndAlgorithm
```

C.

```
Algorithm MODUL(n)
    If n < 0 then
        Return n * (-1)
    Else
        Return n
    EndIf
EndAlgorithm
```

D.

```
Algorithm MODUL(n)
    If n > 0 then
        Return n * (-1)
    Else
        Return n
    EndIf
EndAlgorithm
```

435. Care este valoarea expresiei de mai jos, dacă $x = 15$ și $y = 17$? ✓ ?

(**NU**(x **MOD** $10 = 0$)) **ȘI** (y **MOD** $2 = 0$) **ȘI** ($x < y$)

A. adevărat

B. fals

C. Eroare

D. Expresia nu poate fi evaluată

436. Se consideră subalgoritmul recursiv `ceFace(n, i)`, unde n este un număr natural ($2 \leq n \leq 1000$). ✓ ?

```
Algorithm ceFace(n, i)
   If i = 1 then
      Return i
   Else
      If n MOD i = 0 then
         Return i+ ceFace(n, i - 1)
      Else
         Return ceFace(n, i - 1)
      EndIf
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate pentru apelul `ceFace(n, n)`.

A. Subalgoritmul returnează succesorul celui mai mare divizor al lui n

B. Subalgoritmul returnează suma numerelor naturale neprime, până la n inclusiv

C. Subalgoritmul returnează suma divizorilor proprii ai numărului n

D. Subalgoritmul returnează suma divizorilor proprii și improprii ai numărului n

437. Subalgoritmul `magic(s, n)` are ca parametri de intrare un șir s cu n caractere ✓ ? $(s[1], s[2], ..., s[n])$ și numărul întreg n $(1 \leq n \leq 10000)$.

```
Algorithm magic(s, n)
   i ← 1
   f ← 1
   While i ≤ n DIV 2 execute
      If s[i] ≠ s[n − i + 1] then
         f ← 0
      EndIf
      i ← i + 1
   EndWhile
   Return f
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează 1 dacă s are un număr par de caractere.

B. Subalgoritmul returnează 1 dacă s are un număr impar de caractere.

C. Subalgoritmul returnează 1 dacă s este un palindrom.

D. Subalgoritmul returnează 1 dacă s conține doar caractere distincte.

438. Care dintre următoarele expresii au valoarea *adevărat* dacă și numai dacă x este ✓ ? număr impar și negativ? Notăm cu $|x|$ valoarea absolută a lui x (modulul lui x).

A. $(|x|$ **MOD** $2 = 1)$ **ȘI** $(x < 0)$

B. **NU**$((|x|$ **MOD** $2 = 0)$ **ȘI** $(x \geq 0))$

C. **NU**$((|x|$ **MOD** $2 = 0)$ **SAU** $(x \geq 0))$

D. $(|x|$ **MOD** $2 \neq 0)$ **SAU** $(x < 0)$

439. Subalgoritmul `ceFace(n)` are ca parametru de intrare un număr natural n $(0 \leq$ ✓ ? $n \leq 10000)$.

```
Algorithm ceFace(n)
    s ← 0
    While n > 0 execute
        c ← n MOD 10
        If c MOD 2 ≠ 0 then
            s ← s + c
        EndIf
        n ← n DIV 10
    EndWhile
    Return s
EndAlgorithm
```

Ce va returna apelul `ceFace(1234)`?

A. 4 B. 10 C. 60 D. 0

440. Considerăm un șir de caractere și o funcție f care primește ca parametru un caracter și returnează 1 dacă acel caracter este cifră și 0 altfel. Care dintre următoarele abordări determină dacă șirul de caractere este format numai din cifre? ✓ ?

A. Verificăm dacă funcția *f*, aplicată pe fiecare caracter al șirului de caractere, returnează întotdeauna 1.

B. Verificăm dacă suma valorilor returnate de *f*, aplicată pe fiecare caracter al șirului de caractere, este egală cu lungimea șirului de caractere.

C. Verificăm dacă funcția *f*, aplicată pe fiecare caracter al șirului de caractere, returnează cel puțin o dată 1.

D. Aplicăm funcția f pe caractere alese aleatoriu din șir până când sunt returnate un număr de valori egale cu 1 egal cu lungimea șirului.

441. Care dintre algoritmii următori pot fi implementați în așa fel încât să aibă complexitate de timp liniară ($O(n)$)? ✓ ?

A. Algoritmul de căutare secvențială a unui element într-un vector de n numere

B. Algoritmul de sortare prin inserție a unui tablou unidimensional de n numere

C. Algoritmul de căutare al numărului maxim într-un vector nesortat de n numere

D. Algoritmul de determinare a sumei elementelor de pe diagonala principală a unei matrice pătratice cu n linii și n coloane.

442. Se consideră subalgoritmul `f(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10000$). ✓ ?

```
Algorithm f(a, b)
    m ← a
    While b MOD m > 0 execute
        m ← m + 1
    EndWhile
    Return m
EndAlgorithm
```

Pentru care dintre următoarele apeluri corpul buclei **While** se va executa cel mult o dată?

A. `f(10, 11)` B. `f(10, 10)` C. `f(10, 9)` D. `f(10, 15)`

443. Se consideră subalgoritmul `f(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq$ 10000). ✓ ?

```
Algorithm F(a, b)
    c ← 0
    d ← 0
    p ← 1
    While a + b + c > 0 execute
        c ← a MOD 10 + b MOD 10 + c
        d ← d + (c MOD 10) * p
        p ← p * 10
        a ← a DIV 10
        b ← b DIV 10
        c ← c DIV 10
    EndWhile
    Return d
EndAlgorithm
```

Ce va returna apelul `f(493, 1836)`?

A. 2329 B. 2229 C. 2430 D. 3292

444. Se consideră subalgoritmul `afisare(M, n)` care primește ca și parametru un șir M cu n ($n \leq 10$) numere întregi $(M[1], M[2], ..., M[n])$ reprezentând o mulțime. ✓ ?

```
Algorithm AFISARE(M, n)
    nr ← 2^n
    k ← 0
    While k < nr execute
        curent ← k
        Scrie ''
        For j = 1, n execute
            r ← curent MOD 2
            curent ← curent DIV 2
            If r = 1 then
                Scrie M[j]
            EndIf
        EndFor
        Scrie ''
        Scrie linie nouă
        k ← k + 1
    EndWhile
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul afișează toate permutările mulțimii M.

B. Subalgoritmul afișează toate combinările elementelor mulțimii M luate câte i, $i = 0, 1, ..., n$ (nu neapărat în această ordine).

C. Subalgoritmul afișează toate aranjamentele elementelor mulțimii M luate câte i, $i = 0, 1, ..., n$ (nu neapărat în această ordine).

D. Subalgoritmul afișează toate submulțimile mulțimii M.

445. Se dă subalgoritmul `s(a, b, c)`, unde a, b, c sunt numere naturale pozitive ($1 \leq a, b, c \leq 10000$). ✓ ?

```
Algorithm s(a, b, c)
   If (a = 1)  SAU  (b = 1)  SAU  (c = 1) then
      Return 1
   Else
      If a > b then
         Return a* s(a - 1, b, c)
      Else
         If a < b then
            Return b* s(a, b - 1, c)
         Else
            Return c* s(a - 1, b - 1, c - 1)
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate în cazul în care $a = b$ și $a < c$:

A. Subalgoritmul calculează și returnează $c!$

B. Subalgoritmul calculează și returnează $c!/(c - a + 1)!$

C. Subalgoritmul calculează și returnează $c!/(c - a - 1)!$

D. Subalgoritmul calculează și returnează aranjamente de c luate câte $(a - 1)$

446. Subalgoritmul de mai jos are ca parametri de intrare un șir A cu n numere naturale $(A[1], A[2], ..., A[n])$ și numărul natural n $(1 \leq n \leq 10000)$. Pentru numerele naturale x și y, x^y semnifică x la puterea y (x^y). ✓ ?

```
Algorithm H(A, n)
   If n = 0 then
      Return 0
   Else
      Return A[n] * (-1)^(1-A[n] MOD 2)+ H(A, n - 1)
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează diferența dintre suma elementelor de pe poziții pare și suma elementelor de pe pozițiile impare din șirul A

B. Subalgoritmul returnează diferența dintre suma elementelor pare și suma elementelor impare din șirul A

C. Subalgoritmul returnează diferența dintre suma elementelor impare și suma elementelor pare din șirul A

D. Niciunul din celelalte răspunsuri nu este corect

447. Un fișier Excel conține n înregistrări cu număr de ordine de la 1 la n. Aceste înregistrări trebuie copiate într-un fișier Word în care înregistrările se vor aranja în maxim r rânduri și exact c coloane pe fiecare pagină. Se garantează că valoarea lui n întotdeauna permite aranjarea pe exact c coloane. ✓ ?

Să notăm cu $x_1, ..., x_c$ numărul de înregistrări, care sunt copiate pe fiecare coloană pe o anumită pagină.

Pe prima pagină a documentului Word, datorită prezenței unui antet, numărul de rânduri este r_1, $r_1 < r$ (numărul de rânduri prezent pe prima pagina este mai mic), adică $x_p = r_1$, $\forall 1 \leq p \leq c$.

Înregistrările vor fi aranjate în fișierul Word pe fiecare pagină de sus în jos pe fiecare coloană, coloanele fiind completate de la stânga la dreapta: dacă prima înregistrare de pe o pagină are numărul de ordine i, înregistrarea cu numărul de ordine $(i + 1)$ va fi prezentă sub ea, iar înregistrarea cu numărul de ordine $(i + x_1)$ va fi prima înregistrare de pe coloana 2 de pe pagina respectivă ș.a.m.d.

Pe ultima pagină a documentului Word se dorește ca pe toate coloanele numărul înregistrărilor să fie echilibrat, adică diferența dintre numărul înregistrărilor de pe oricare două coloane să fie cel mult 1 ($|x_j - x_k| \leq 1$, $\forall 1 \leq j, k \leq c$, $j \neq k$).

În cazul celorlalte pagini (în afară de prima și ultima) $x_p = r$, $\forall 1 \leq p \leq c$.

Pentru $n = 5883$, $r = 46$, $r_1 = 12$ și $c = 2$ pe ce rând al paginii se poate regăsi ultima înregistrare din document (cea cu număr de ordine $i = 5883$)?

A. 29 B. 30 C. 31 D. 32

448. Se consideră subalgoritmul `prelucreaza(a, b, c, d, e)`, care primește ca parametri cinci numere întregi a, b, c, d și e ($1 \leq a, b \leq 10000, 2 \leq c \leq 16, 1 \leq d < c$). ✓ ?

```
Algorithm PRELUCREAZA(a, b, c, d, e)
   If a = 0 ȘI b = 0 then
      If e = 0 then
         Return 1
      Else
         Return 0
      EndIf
   EndIf
   If (a MOD c = d) ȘI (b MOD c = d) then
      Return PRELUCREAZA(a DIV c, b DIV c, c, d, e)
   EndIf
   If a MOD c = d then
      Return PRELUCREAZA(a DIV c, b DIV c, c, d, e + 1)
   EndIf
   If b MOD c = d then
      Return PRELUCREAZA(a DIV c, b DIV c, c, d, e - 1)
   EndIf
   Return PRELUCREAZA(a DIV c, b DIV c, c, d, e)
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate pentru apelul `prelucreaza(a, b, c, d, 0)`:

A. Returnează 1 dacă reprezentările în baza c a numerelor a și b conțin cifra d de număr egal de ori, 0 în caz contrar

B. Returnează 1 dacă cifra d apare în reprezentarea în baza c a numărului a și în reprezentarea în baza c a numărului b, 0 în caz contrar

C. Returnează 1 dacă cifra d apare în reprezentarea în baza c a numărului a sau în reprezentarea în baza c a numărului b, 0 în caz contrar

D. Returnează 1 dacă cifra d nu apare deloc în reprezentările în baza c a numerelor a și b, 0 în caz contrar

449. Se consideră subalgoritmii `val(p, s, i, n, x)` și `val_exp(p, n, x)` a căror parametri au următoarea specificație: un șir p cu n numere întregi $(p[1], p[2], ..., p[n])$, numerele naturale s, i și n ($n \leq 1000, n = 2^k, k < 10$), și numărul real x. Valorile șirului p reprezintă coeficienții expresiei în ordine crescătoare a exponenților, exponentul maxim fiind egal cu $n-1$, într-o expresie de forma $p[1]+p[2]\cdot x+p[3]\cdot x^2+...+p[n]\cdot x^{n-1}$ ✓ ?

Exemplu: $p = [1, 2, 3, 4]$ corespunde expresiei $E(x) = 1 + 2x + 3x^2 + 4x^3$.

```
Algorithm VAL(p, s, i, n, x)
   If s + i ≤ n then
      Return _____________________________
   Else
      Return p[s]
   EndIf
EndAlgorithm
Algorithm VAL_EXP(p, n, x)
   Return VAL(p, 1, 1, n, x)
EndAlgorithm
```

Care dintre următoarele variante completează corect spațiul subliniat astfel încât subalgoritmul `val_exp(p, n, x)` să returneze valoarea expresiei $E(x)$?

A. **Return** $p[s] + x*$ `val(p, s + i, i * 2, n, x * x)`

B. **Return** `val(p, s, i * 2, n - i, x * x)` $+ x *$ `val(p, s + i, i * 2, n, x * x)`

C. **Return** `val(p, s + i, i * 2, n, x * x)` $+ x *$ `val(p, s, i * 2, n - i, x * x)`

D. **Return** $p[s] + x*$ `val(p, s + i, i, n, x)`

450. Se consideră subalgoritmul `f(a)`, care primește ca și parametru un număr natural a ($2 \leq a < 1000000$) și returnează *adevărat* dacă există un număr natural d, $1 < d < a$ cu proprietatea că d divide a, și *fals* în caz contrar. Notația $[\sqrt{x}]$ reprezintă partea întreagă a numărului x. ✓ ?

Care dintre variantele următoare ale subalgoritmului `f(a)` sunt corecte?

A.

```
Algorithm F(a)
   If a = 2 then
      Return fals
   EndIf
   If a MOD 2 = 0 then
      Return adevărat
   EndIf
   For d ← 3, [√a] − 1, 2 execute
      If a MOD d = 0 then
         Return adevărat
      EndIf
   EndFor
   Return fals
EndAlgorithm
```

B.

```
Algorithm F(a)
   For d ← 2, [√a] execute
      If a MOD d = 0 then
         Return adevărat
      EndIf
   EndFor
   Return fals
EndAlgorithm
```

C.

```
Algorithm F(a)
   If a ≤ 2 then
      Return fals
   EndIf
   If a MOD 2 = 0 then
      Return adevărat
   EndIf
   For d ← 3, [√a], 2 execute
      If a MOD d = 0 then
         Return adevărat
      EndIf
   EndFor
   Return fals
EndAlgorithm
```

D.

```
Algorithm F(a)
   d ← a − 1
   While adevărat execute
      If a MOD d = 0 then
         Return adevărat
      EndIf
      d ← d − 1
   EndWhile
   Return fals
EndAlgorithm
```

451. Fie expresia de mai jos, unde $1 < A < 2021$ și $1 < n < 10202110$. ✓ ?

$$E(A,n) = (A + A^2 + A^3 + ... + A^n) \textbf{ MOD } 2021$$

Care dintre următorii subalgoritmi calculează corect valoarea $E(A,n)$ și are complexitatea timp specificată?

Presupuneți că toate calculele se realizează pe tipuri de date pe 32 de biți. Presupuneți că x^k se calculează în $O(\log k)$.

A.

```
Algorithm E(A, n)
   Return (A ∗ (A^n − 1) DIV (A − 1)) MOD 2021
EndAlgorithm
```

Complexitatea timp: $O(\log n)$

B.

```
Algorithm E(A, n)
   Return ((A ∗ (A^n − 1)) MOD 2021) DIV ((A − 1) MOD 2021)
EndAlgorithm
```

Complexitatea timp: $O(\log n)$

C.

```
Algorithm E1(A, n)
   If n = 1 then
      Return (A, A) //returnează o pereche de valori
   EndIf
   If n MOD 2 = 1 then
      (t1, t2) ← E1(A, n - 1)
      p ← (t1 * A) MOD 2021
      Return (p, (p + t2) MOD 2021)
   Else
      (t1, t2) ← E1(A, n DIV 2)
      p ← (t1 * t1) MOD 2021
      Return (p, ((1 + t1) * t2) MOD 2021)
   EndIf
EndAlgorithm
Algorithm E(A, n)
   (aux1, aux2) ← E1(A, n)
   Return aux2
EndAlgorithm
```

`Complexitatea timp:` $O(\log n)$

D.

```
Algorithm E(A, n)
   raspuns ← A
   For i = 2, n execute
      raspuns ← raspuns + A^i
   EndFor
   Return raspuns MOD 2021
EndAlgorithm
```

`Complexitatea:` $O(n \cdot \log n)$

452. Pe un cerc se scriu, în ordine crescătoare, toate numerele de la 1 la 1000, în sensul acelor de ceasornic. Începând de la 1, colorăm, în sensul acelor de ceasornic, fiecare al k-lea număr $(1, k+1, 2 \cdot k+1, ...)$. Procedeul se continuă până când ajunge la un număr deja colorat, fiind colorate la final x numere. Care dintre următoarele afirmații sunt adevărate?

A. Dacă $k = 15$ atunci $x = 300$
B. Dacă $k = 45$ atunci $x = 200$
C. Dacă $k = 25$ atunci $x = 40$
D. Dacă $k = 30$ atunci $x = 150$

453. Se consideră subalgoritmul `ceFace(n, k)` unde n și k sunt numere naturale ($1 \leq n, k \leq 1000000$).

```
Algorithm ceFace(n, k)
   nr ← 0
   p ← 1
   While (n ≠ 0) ȘI (k ≠ 0) execute
      If n MOD 2 ≠ 0 then
         nr ← nr + ((n DIV 10) MOD 10) * p
         p ← p * 10
      Else
         k ← k - 1
```

```
      EndIf
      n ← n DIV 10
   EndWhile
   Return nr
EndAlgorithm
```

Care dintre următoarele perechi de apeluri returnează valori identice?

A. `ceFace(32345, 3)` și `ceFace(321458, 7)`

B. `ceFace(321458, 4)` și `ceFace(2314587, 4)`

C. `ceFace(2314, 3)` și `ceFace(23145, 4)`

D. `ceFace(23145, 3)` și `ceFace(231458, 4)`

454. Se consideră subalgoritmii: * `putere(b, p)` – determină b^p (b la puterea p), b, p - numere naturale ($1 \leq b \leq 20, 1 \leq p \leq 20$); * `nrCifre(nr)` – returnează numărul cifrelor unui număr natural nenul nr ($0 < nr \leq 1000000$), sau valoarea 0 atunci când $nr = 0$; * `produs(st, dr)` – subalgoritmul de mai jos, unde st, dr – numere naturale ($100 < st < 1000000, 0 \leq dr < 1000000$, st – număr care, reprezentat în baza 10, are cel puțin două cifre nenule).

```
Algorithm PRODUS(st, dr)
   If st > 0 then
      drCrt ← ----------------------------
      stCrt ← st DIV 10
      If st * dr < stCrt * drCrt then
         Return PRODUS(stCrt, drCrt)
      Else
         Return st * dr
      EndIf
   Else
      Return st * dr
   EndIf
EndAlgorithm
```

Care dintre următoarele variante completează corect spațiul subliniat astfel încât subalgoritmul `produs(st, dr)` prin executarea secvenței de instrucțiuni

```
scrie produs(1092, 0)
scrie produs(75981, 0)
```

să se afișeze 920 și 73575?

A. $(st$ **MOD** $10) *$ `putere(10, nrCifre(dr))` $+dr$

B. $(st$ **MOD** $10) *$ `putere(10, dr)` $+dr$

C. $(st$ **MOD** $10) *$ `putere(10, nrCifre(dr))`

D. $(st$ **MOD** $10) *$ `nrCifre(dr)`

455. Se consideră subalgoritmul `ceFace(a, n, i, f)`, care primește ca parametru un șir a cu n numere întregi ($a[1], a[2], ..., a[n]$) și numerele întregi i, f și n ($2 \leq n \leq 10000$).

```
Algorithm CEFACE(a, n, i, f)
   If (i = n) ȘI (f = 2) then
      Return ADEVĂRAT
   Else
```

```
        If (i = n) then
           Return FALS
        Else
           If (f ≤ 1)  ȘI  (a[i] < a[i + 1]) then
              Return ceFace(a, n, i + 1, 1)
           EndIf
           If (1 ≤ f)  ȘI  (a[i] > a[i + 1]) then
              Return ceFace(a, n, i + 1, 2)
           EndIf
           Return FALS
        EndIf
     EndIf
  EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate, considerând apelul inițial `ceFace(a,n,1,0)`.

A. Subalgoritmul returnează *adevărat* dacă și numai dacă maximul șirului a se află pe o poziție i, $1 < i < n$.

B. Subalgoritmul returnează *adevărat* dacă și numai dacă $\exists k$, $(1 < k < n)$, astfel încât $a[1] < a[2] < ... < a[k] > a[k+1] > ... > a[n]$.

C. Subalgoritmul returnează *fals* dacă șirul a este strict crescător.

D. Subalgoritmul returnează *adevărat* dacă și numai dacă $\exists k$, $(1 < k < n)$, astfel încât $a[k] > a[k+1] > ... > a[n]$.

456. Fie următorul subalgoritm, având ca parametru numărul natural nenul n și care returnează un număr natural. ✓ ?

```
Algorithm f(n)
   j ← n
   While j > 1 execute
      i ← 1
      While i ≤ n^4 execute
         i ← 4 * i
      EndWhile
      j ← j  DIV  2
   EndWhile
   Return j
EndAlgorithm
```

În care dintre următoarele clase de complexitate se încadrează complexitatea timp a algoritmului?

A. $O(\log_2 n^2)$ B. $O(\log_2^2 n^2)$ C. $O(\log_4^2 n)$ D. $O(\log_2 \log_4 n)$

457. Se dă un șir s de n caractere din alfabetul englez, $(s[1], s[2], ..., s[n])$. Dorim să aflăm cel mai lung sufix al său care este palindrom. Un sufix al unui șir de caractere este o subsecvență a șirului care conține ultimul caracter. De exemplu, pentru șirul *abab*, cel mai lung sufix palindrom al său este *bab*. ✓ ?

Presupunem că avem definit următorul subalgoritm: * `ascii(c)` - returnează codul ASCII al caracterului `c`.

Presupunem că operațiile aritmetice nu produc depășire pe mulțimea numerelor întregi. Care dintre următoarele implementări returnează lungimea acestui sufix la apelul `sufix(s, n)`?

A.

```
Algorithm SUFIX(s, n)
    hf ← 0
    hb ← 0
    raspuns ← 1
    For i ← n, 1, −1 execute
        hf ← ascii(s[i]) + 2021 ∗ hf
        hb ← hb + ascii(s[i]) ∗ 2021^(n−i)
        If hf = hb then
            raspuns ← n − i + 1
        EndIf
    EndFor
    Return raspuns
EndAlgorithm
```

B.

```
Algorithm SUFIX(s, n)
    hf ← 0
    hb ← 0
    raspuns ← 1
    For i ← n, 1, −1 execute
        hf ← ascii(s[i]) + 3 ∗ hf
        hb ← hb + ascii(s[i]) ∗ 3^(n−i)
        If hf = hb then
            raspuns ← n − i + 1
        EndIf
    EndFor
    Return raspuns
EndAlgorithm
```

C.

```
Algorithm SUFIX(s, n)
    hf ← 0
    hb ← 0
    raspuns ← 1
    For i ← n, 1, −1 execute
        hf ← ascii(s[i]) + 2021 ∗ hb
        hb ← hf + ascii(s[i]) ∗ 2021^(n−i)
        If hf = hb then
            raspuns ← n − i + 1
        EndIf
    EndFor
    Return raspuns
EndAlgorithm
```

D. Niciuna dintre celelalte variante nu este corectă.

Admitere nivel licență, sesiunea iulie 2021

458. Fie următorul subalgoritm, având ca parametru de intrare numărul natural n și care returnează un număr natural. ✓ ?

```
Algorithm CALCUL(n)
    E ← 1
    P ← 1
    i ← 2
    While i ≤ n execute
        P ← (−1) * P * i
        E ← E + P
        i ← i + 1
    EndWhile
    Return E
EndAlgorithm
```

Care este valoarea returnată de subalgoritm, în condițiile în care $n \geq 1$?

A. $1! - 2! + 3! - 4! + ... + (-1)^{n+1} \cdot n!$

B. $1 - 1! + 2! - 3! + ... + (-1)^{n} \cdot n!$

C. $1 - 1 \cdot 2 + 1 \cdot 2 \cdot 3 - 1 \cdot 2 \cdot 3 \cdot 4 + ... + (-1)^{n+1} \cdot 1 \cdot 2 \cdot ... \cdot n$

D. $1 + 1 \cdot 2 - 1 \cdot 2 \cdot 3 + 1 \cdot 2 \cdot 3 \cdot 4 + ... + (-1)^{n} \cdot 1 \cdot 2 \cdot ... \cdot n$

459. Un fișier Excel conține n înregistrări numerotate de la 1 la n. Aceste înregistrări trebuie copiate într-un fișier Word în care înregistrările se vor aranja în câte r rânduri și c coloane pe fiecare pagină (cu excepția primei și ultimei pagini). Pe prima pagină a documentului Word, datorită prezenței unui antet, numărul de rânduri este r_1, $r_1 < r$ (numărul de rânduri prezent pe prima pagina este mai mic). ✓ ?

Înregistrările vor fi aranjate în fișierul Word pe fiecare pagină de sus în jos pe fiecare coloană, coloanele fiind completate de la stânga la dreapta: dacă prima înregistrare de pe o pagină are numărul de ordine i, înregistrarea cu numărul de ordine $(i + 1)$ va fi prezentă sub ea, iar înregistrarea cu numărul de ordine $(i + r)$ va fi prima înregistrare de pe coloana 2 de pe pagina respectivă ș.a.m.d.

Pentru $n = 5000$, $r = 46$, $r_1 = 12$ și $c = 2$ pe ce pagină a documentului Word și pe ce coloană se va regăsi înregistrarea cu număr de ordine $i = 3245$?

A. Pagina 36, ultima coloană

B. Pagina 37, prima coloană

C. Pagina 37, ultima coloană

D. Pagina 38, prima coloană

460. Se consideră subalgoritmul `ceFace(m)`, unde m este un număr natural ($10 \leq m \leq 10000$). ✓ ?

```
Algorithm CEFACE(m)
    If m = 0 then
        Return 0
    EndIf
    If m MOD 9 = 0 then
        Return 9
    EndIf
    Return m MOD 9
```

```
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează restul împărțirii numărului m la 9.

B. Subalgoritmul returnează numărul divizorilor care sunt divizibili cu 9 ai numărului m.

C. Subalgoritmul returnează cifra de control a numărului m (suma cifrelor sale, apoi suma cifrelor acestei sume, până când suma obținută este un număr format dintr-o singură cifră).

D. Subalgoritmul returnează cifra de control a numărului m (suma cifrelor sale, apoi suma cifrelor acestei sume, până când suma obținută este un număr format dintr-o singură cifră) dacă și numai dacă numărul m este divizibil cu 9.

461. Pentru a genera numerele cu n cifre formate doar din cifrele 0, 2, 9, se utilizează un algoritm care, pentru $n = 2$, generează în ordine crescătoare numerele 20, 22, 29, 90, 92, 99. Dacă $n = 4$ și se utilizează același algoritm, care este numărul generat imediat după numărul 2009? ✓ ?

A. 2022

B. 2090

C. 2010

D. Niciuna dintre celelalte variante

462. Se consideră subalgoritmul `cauta(n)`, unde n este un număr natural ($0 \leq n \leq 1000000$). ✓ ?

```
Algorithm CAUTA(n)
    v ← 0
    If n = 0 then
        Return 1
    Else
        m ← n
        While m > 0 execute
            If m MOD 10 = 0 then
                v ← v + 1
            EndIf
            m ← m DIV 10
        EndWhile
        Return v
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul determină și returnează câte cifre are numărul n.

B. Subalgoritmul returnează 1 dacă numărul n este o putere a lui 10 și 0 altfel.

C. Subalgoritmul returnează 1 dacă numărul n se termină cu cifra 0 și 0 altfel.

D. Subalgoritmul determină și returnează numărul de cifre 0 din numărul n.

463. Se consideră subalgoritmul `abc(a, n, p)`, unde n este număr natural ($1 \leq n \leq 10000$), p este număr întreg ($-10000 \leq p \leq 10000$), iar a este un șir cu n numere naturale nenule $(a[1], a[2], ..., a[n])$. ✓ ?

```
Algorithm ABC(a, n, p)
   If n < 1 then
      Return 0
   Else
      If (1 ≤ p)  ŞI  (p ≤ n) then
         Return a[p]
      Else
         Return −1
      EndIf
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează -1 dacă și numai dacă p este negativ sau mai mare decât n.

B. Subalgoritmul returnează elementul de pe poziția p dacă p este strict mai mare decât 0 și mai mic sau egal decât lungimea șirului.

C. Subalgoritmul nu returnează niciodată 0 pentru valori ale parametrilor care respectă precondiţiile din enunţ.

D. Subalgoritmul returnează elementul de pe poziția p dacă p este mai mare sau egal cu 0 și mai mic strict decât lungimea șirului. În cazul în care p nu este între 1 și n, returnează -1.

464. Care dintre secvențele următoare determină în variabila i lungimea unui șir de caractere care se termină cu caracterul '*' (asterisc)? Primul caracter se află la indicele 1, iar caracterul asterisc este parte a șirului de caractere. ✓ ?

A.
```
i ← 1
While x[i] ≠ '*' execute
   i ← i + 1
EndWhile
```

B.
```
i ← 1
While x[i] = '*' execute
   i ← i + 1
EndWhile
i ← i − 1
```

C.
```
i ← 1
While x[i] ≠ '*' execute
   i ← i + 1
EndWhile
i ← i + 1
```

D.
```
i ← 1
While x[i] ≠ '*' execute
   i ← i + 1
EndWhile
i ← i − 1
```

465. Fie următorul subalgoritm, având ca parametru numărul natural nenul n și care returnează un număr natural. ✓ ?

```
Algorithm F(n)
   j ← n
   While j > 1 execute
      i ← 1
      While i ≤ n execute
         i ← 2 * i
      EndWhile
      j ← j  DIV  3
```

```
   EndWhile
   Return j
EndAlgorithm
```

În care dintre următoarele clase de complexitate se încadrează complexitatea timp a algoritmului?

A. $O(\log_2 n)$ B. $O(\log_2^2 n)$ C. $O(\log_3 n)$ D. $O(\log_2 \log_3 n)$

466. Subalgoritmul `cate(n, m)` primește ca parametri numerele naturale n și m. ✓ ?

```
Algorithm CATE(n, m)
   If n ≤ m then
      If (n MOD 2 = 0) ȘI (n MOD 3 ≠ 0) then
         Return 1+ cate(n + 1, m)
      Else
         Return cate(n + 1, m)
      EndIf
   Else
      Return 0
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $n = 0$ și $m = 1$, subalgoritmul returnează valoarea 0.

B. Dacă $n = 4$ și $m = 21$, subalgoritmul returnează valoarea 6.

C. Dacă $n = 7$ și $m = 120$, subalgoritmul returnează valoarea 36.

D. Dacă $n = 1$ și $m = 215$, subalgoritmul returnează valoarea 72.

467. Se consideră subalgoritmul `verifica(n)`, unde n este un număr natural ($1 \leq n \leq$ ✓ ? 100000).

```
Algorithm VERIFICA(n)
   While n > 0 execute
      If (n MOD 3) > 1 then
         Return 0
      EndIf
      n ← n DIV 3
   EndWhile
   Return 1
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează 1 dacă n este o putere a lui 3, 0 în caz contrar.

B. Subalgoritmul returnează 1 dacă scrierea în baza 3 a lui n conține doar cifrele 0 și/sau 1, 0 în caz contrar.

C. Subalgoritmul returnează 1 dacă n poate fi scris ca sumă a puterilor distincte ale lui 3, 0 în caz contrar.

D. Subalgoritmul returnează 1 dacă scrierea în baza 3 a lui n conține doar cifra 2, 0 în caz contrar.

468. Pentru un număr natural nr ($1000 \leq nr \leq 1000000$), definim operația de decrementare în modul următor: dacă ultima cifră a lui nr nu este 0, scădem 1 din nr, altfel, împărțim nr la 10 și păstrăm doar partea întreagă. Care dintre următorii subalgoritmi returnează, la apelul `decrementare(nr, k)`, numărul obținut aplicând de k ori ($0 \leq k \leq 100$) operația de decrementare pe numărul nr? De exemplu, pentru $nr = 15243$ și $k = 10$, rezultatul este 151.

A.

```
Algorithm DECREMENTARE(nr, k)
   If k = 0 then
      Return nr
   Else
      If nr MOD 10 ≠ 0 then
         Return decrementare(nr
DIV 10, k - 1)
      Else
         Return decrementare(nr
- 1, k - 1)
      EndIf
   EndIf
EndAlgorithm
```

B.

```
Algorithm DECREMENTARE(nr, k)
   While k > 0 execute
      If nr MOD 10 = 0 then
         nr ← nr DIV 10
      Else
         nr ← nr - 1
      EndIf
   EndWhile
   Return nr
EndAlgorithm
```

C.

```
Algorithm DECREMENTARE(nr, k)
   For i ← 1, k execute
      If nr MOD 10 > 0 then
         nr ← nr - 1
      Else
         nr ← nr DIV 10
      EndIf
   EndFor
   Return nr
EndAlgorithm
```

D.

```
Algorithm DECREMENTARE(nr, k)
   If k = 0 then
      Return nr
   Else
      If k > nr MOD 10 then
         nr1 ← nr DIV 10
         Return decrementare(nr1,
k - nr MOD 10 - 1)
      Else
         Return decrementare(nr
- k, 0)
      EndIf
   EndIf
EndAlgorithm
```

469. Se dă următorul subalgoritm care are ca parametri de intrare un șir x cu n numere naturale $(x[1], x[2], ..., x[n])$ și numărul întreg n.

```
Algorithm F(x, n)
   If n = 1 then
      Return 100
   Else
      If x[n] > f(x, n - 1) then
         Return x[n]
      Else
         Return f(x, n - 1)
      EndIf
   EndIf
EndAlgorithm
```

Care va fi rezultatul execuției subalgoritmului pentru $x = [101, 7, 6, 3]$ și $n = 4$?

A. 101
C. 100
B. 3
D. 7

470. Subalgoritmul de mai jos are ca parametri de intrare un șir a cu n numere naturale $(a[1], a[2], ..., a[n])$ și numărul natural n $(2 \leq n \leq 10000)$. ✓ ?

```
Algorithm H(a, n)
   If n ≤ 0 then
      Return 0
   EndIf
   If (n MOD 2 = 0) ȘI (a[n] MOD 2 = 0) then
      Return h(a, n - 1) + a[n]
   EndIf
   Return h(a, n - 1) - a[n]
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează diferența dintre suma elementelor care au aceeași paritate cu poziția pe care se află și suma elementelor care au paritate diferită față de poziția pe care se află din șirul a.
B. Subalgoritmul returnează diferența dintre suma elementelor pare de pe pozițiile pare și suma elementelor impare de pe pozițiile impare din șirul a.
C. Subalgoritmul returnează diferența dintre suma elementelor pare și suma elementelor impare din șirul a.
D. Subalgoritmul returnează diferența dintre suma elementelor pare de pe poziții pare și suma celorlalte elemente din șirul a.

471. Se consideră subalgoritmul `ceFace(n)`, cu parametrul n număr natural nenul. ✓ ?

```
Algorithm CEFACE(n)
   i ← 1
   While n > 0 execute
      If n MOD 2 ≠ 0 then
         scrie i
      EndIf
      i ← i + 1
      n ← n DIV 2
   EndWhile
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul afișează secvența: 12345 pentru $n = 31$.
B. Subalgoritmul afișează secvența: 234 pentru $n = 14$.
C. Subalgoritmul afișează 1 la începutul secvenței, pentru n impar.
D. Subalgoritmul afișează un singur număr pentru $n = 2^k$, unde k este un număr natural.

472. Se dă o mulțime S, care conține n intervale specificate prin capătul stâng s_i și capătul drept d_i $(s_i < d_i \ \forall \ i = 1...n)$. Se dorește determinarea unei submulțimi $S' \subseteq S$ de m elemente, astfel încât să nu existe două intervale în S' care se intersectează și m să aibă cea mai mare valoare posibilă. ✓ ?

Care dintre următoarele strategii rezolvă corect problema?

A. Se sortează intervalele din mulțimea S crescător după capătul stâng. Se adaugă primul interval din șirul sortat în S'. Se parcurg celelalte elemente din șir în ordinea sortată și când se întâlnește un interval care nu se intersectează cu intervalul care a fost adăugat ultima oară în S', se adaugă și acesta în S'.

B. Se sortează intervalele din mulțimea S crescător după capătul drept. Se adaugă primul interval din șirul sortat în S'. Se parcurg celelalte elemente din șir în ordinea sortată și când se întâlnește un interval care nu se intersectează cu intervalul care a fost adăugat ultima oară în S', se adaugă și acesta în S'.

C. Se sortează intervalele din mulțimea S crescător după lungimea intervalului. Se adaugă primul interval din șirul sortat în S'. Se parcurg celelalte elemente din șir în ordinea sortată și când se întâlnește un interval care nu se intersectează cu intervalul care a fost adăugat ultima oară în S', se adaugă și acesta în S'.

D. Se sortează intervalele din mulțimea S crescător după numărul de intervale din S cu care se intersectează. Se adaugă primul interval din șirul sortat în S'. Se parcurg celelalte elemente din șir în ordinea sortată și când se întâlnește un interval care nu se intersectează cu intervalul care a fost adăugat ultima oară în S', se adaugă și acesta în S'.

473. Se consideră subalgoritmul `f(a, b)`, care primește ca parametri două numere naturale a și b ($1 \leq a < b \leq 1000$). ✓ ?

```
Algorithm F(a, b)
    m ← 0
    For n ← a, b execute
        c ← 0
        For d ← 1, n execute
            If n MOD d = 0 then
                c ← c + 1
            EndIf
        EndFor
        If c > m then
            m ← c
        EndIf
    EndFor
    For n ← a, b execute
        c ← 0
        For d ← 1, n execute
            If n MOD d = 0 then
                c ← c + 1
            EndIf
        EndFor
        If c = m then
            scrie n
        EndIf
    EndFor
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul afișează maximul dintre numărul de divizori ai lui a și numărul de divizori ai lui b.

B. Subalgoritmul afișează numerele naturale din intervalul $[a, b]$ care au proprietatea că au cel mai mare număr de divizori.

C. Subalgoritmul afișează numărul de divizori pentru fiecare număr natural din intervalul $[a, b]$.

D. Subalgoritmul afișează numerele naturale din intervalul $[a, b]$ care au proprietatea că au cel mai mare număr de divizori proprii.

474. Fie numerele naturale a și b, cu $b \neq 0$. Care dintre următoarele variante calculează: ✓ ? a **DIV** b, dacă a **MOD** $b = 0$ **sau** (a/b) rotunjit în sus către cel mai apropiat întreg, dacă a **MOD** $b \neq 0$.

A. $(a-1)$ **DIV** b

B. $(a+b+1)$ **DIV** b

C. $(a+b-1)$ **DIV** b

D. $((a+2*b-1)$ **DIV** $b)-1$

475. Ionel trebuie să implementeze algoritmul de căutare binară a unui element a într-un ✓ ? șir V cu n $(1 \leq n \leq 1000)$ numere întregi ordonate crescător $(V[1], V[2], ..., V[n])$. El scrie următorul subalgoritm:

```
Algorithm CautareBinara(a, n, V)
    st ← 1
    dr ← n
    While dr − st > 1 execute
        m ← (st + dr)  DIV  2
        If a ≤ V[m] then
            dr ← m
        Else
            st ← m
        EndIf
    EndWhile
    Return dr
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $n = 1$ atunci valoarea returnată de subalgoritm este întotdeauna 1.

B. Pentru orice $n \geq 1$, subalgoritmul scris de Ionel returnează valoarea 1 atunci când a este mai mic decât toate elementele din șir.

C. Atunci când elementul a apare în șir, subalgoritmul scris de Ionel NU returnează întotdeauna poziția (indicele în vectorul V) pe care acesta apare.

D. Pentru orice $n > 1$, subalgoritmul scris de Ionel returnează valoarea n atunci când a este mai mare decât toate elementele din șir.

476. Se consideră subalgoritmul `calcul(x, n)`, unde parametrii de intrare sunt numerele ✓ ? naturale n și x, cu condiția $1 \leq x \leq n < 10$.

```
Algorithm calcul(x, n)
    b ← 1
    For i ← 1, n − x execute
        b ← b ∗ i
    EndFor
    a ← b
    For i ← n − x + 1, n execute
        a ← a ∗ i
    EndFor
```

```
    Return a DIV b
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $n = 5$ și $x = 2$, atunci subalgoritmul returnează 20.

B. Dacă $n = 3$ și $x = 2$, atunci subalgoritmul returnează 6.

C. Subalgoritmul returnează cardinalitatea mulțimii $\{\overline{c_1c_2...c_x} : c_i \neq c_j \forall 1 \leq i, j \leq x, i \neq j, 1 \leq c_i \leq n\}$

D. Subalgoritmul efectuează n operații de înmulțire.

477. Se consideră subalgoritmul `ceFace(n, k)`, care primește ca și parametru două numere naturale nenule n și k ($1 \leq n, k \leq 1000000$). ✓ ?

```
Algorithm ceFace(n, k)
   While n ≥ 1 execute
      If k ≤ n then
         i ← k
      Else
         i ← n
      EndIf
      n ← n − i
      x ← 1
      While i ≥ 1 execute
         scrie x, ' '
         x ← x + 1
         i ← i − 1
      EndWhile
   EndWhile
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $n = 8$ și $k = 3$ subalgoritmul afișează șirul 1 2 3 1 2 3 1 2

B. Pentru $k = 2$, cea mai mică valoare a lui n pentru care se afișează de 3 ori valoarea 1 pe ecran este $n = 3$.

C. Pentru $k = 5$, cea mai mică valoare a lui n pentru care se afișează de 37 ori valoarea 2 pe ecran este $n = 182$.

D. Pentru $n = 7$ și $k = 3$ subalgoritmul afișează 1 2 3 1 2 3

478. Se consideră subalgoritmul `calculeaza(a, b, c)`, cu parametrii de intrare numere naturale nenule, care calculează cel mai mare divizor comun al celor trei numere. ✓ ?

Care dintre următoarele sunt implementări corecte ale subalgoritmului:

A.

```
Algorithm CALCULEAZA(a, b, c)
   While (a ≠ b) SAU (a ≠ c) SAU
(b ≠ c) execute
      x ← a
      If a ≠ x then
         a ← a − x
      EndIf
      If b ≠ x then
         b ← b − x
      EndIf
      If c ≠ x then
         c ← c − x
      EndIf
   EndWhile
   Return x
EndAlgorithm
```

B.

```
Algorithm CALCULEAZA(a, b, c)
   x ← a
   y ← b
   While x ≠ y execute
      If x > y then
         x ← x − y
      Else
         y ← y − x
      EndIf
   EndWhile
   z ← c
   While x ≠ z execute
      If x > z then
         x ← x − z
      Else
         z ← z − x
      EndIf
   EndWhile
   Return x
EndAlgorithm
```

C.

```
Algorithm CALCULEAZA(a, b, c)
   While (a ≠ b) SAU (a ≠ c) SAU
(b ≠ c) execute
      x ← a
      If b < x then
         x ← b
      EndIf
      If c < x then
         x ← c
      EndIf
      If a ≠ x then
         a ← a − x
      EndIf
      If b ≠ x then
         b ← b − x
      EndIf
      If c ≠ x then
         c ← c − x
      EndIf
   EndWhile
   Return x
EndAlgorithm
```

D.

```
Algorithm CALCULEAZA(a, b, c)
   x ← a
   y ← b
   r ← x MOD y
   While r ≠ 0 execute
      x ← y
      y ← r
      r ← x MOD y
   EndWhile
   z ← c
   r ← y MOD z
   While r ≠ 0 execute
      y ← z
      z ← r
      r ← y MOD z
   EndWhile
   Return z
EndAlgorithm
```

479. Subalgoritmul `ceFace(n)` are ca parametru numărul natural n $(1 \leq n \leq 100)$. ✓ ?

```
Algorithm CEFACE(n)
   s ← 0
   If n MOD 2 = 0 then
      a ← 1
      While a < n execute
         s ← s + a
```

```
        a ← a + 2
      EndWhile
    Else
      b ← 2
      While b < n execute
        s ← s + b
        b ← b + 2
      EndWhile
    EndIf
    Return s
  EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă n este par, subalgoritmul returnează suma numerelor naturale mai mici strict decât n; dacă n este impar, returnează suma numerelor naturale pare mai mici decât n.

B. Dacă n este par, subalgoritmul returnează suma numerelor naturale pare mai mici strict decât n; dacă n este impar, returnează suma numerelor naturale impare mai mici decât n.

C. Dacă n este par, subalgoritmul returnează suma numerelor naturale impare mai mici decât n; dacă n este impar, returnează suma numerelor naturale pare mai mici decât n.

D. Dacă n este par, subalgoritmul returnează suma numerelor naturale pare mai mici strict decât n; dacă n este impar, returnează suma numerelor naturale mai mici strict decât n.

480. Subalgoritmul `ceFace(a)` primește ca parametru numărul natural a ($1 \leq a \leq$ ✓ ? 100000).

```
Algorithm ceFace(a)
    b ← 0
    c ← 0
    d ← 0
    e ← 1
    While a > 0 execute
      d ← a MOD 10
      If (d ≠ 4) ȘI (d < 7) then
        b ← b + e * (d DIV 2)
        c ← c + e * (d - d DIV 2)
      Else
        b ← b + e
        c ← c + e * (d - 1)
      EndIf
      a ← a DIV 10
      e ← e * 10
    EndWhile
    scrie b
    scrie c
EndAlgorithm
```

Care dintre următoarele perechi de valori nu vor fi afișate pentru nicio valoare de intrare validă?

A. 1112 și 11233
B. 1111 și 88888
C. 21001 și 33011
D. 3141 și 3258

481. Se consideră subalgoritmii `f(n, c)` și `g(n, c)`, care primesc ca parametri numerele naturale n și c. ✓ ?

```
Algorithm F(n, c)
   If n ≤ 9 then
      If n = c then
         Return 1
      Else
         Return 0
      EndIf
   Else
      If n  MOD  10 = c then
         Return f(n  DIV  10, c) + 1
      Else
         Return f(n  DIV  10, c)
      EndIf
   EndIf
EndAlgorithm

Algorithm G(n, c)
   If c = 0 then
      Return 0
   Else
      If f(n, c) > 0 then
         Return g(n, c - 1) + 1
      Else
         Return g(n, c - 1)
      EndIf
   EndIf
EndAlgorithm
```

Ce returnează apelul `g(n, 9)`?

A. Returnează numărul de cifre ale numărului n.
B. Returnează numărul de cifre distincte ale numărului n.
C. Returnează numărul de cifre mai mari decât 1 ale numărului n.
D. Niciunul dintre celelalte răspunsuri nu este corect.

482. Pe un site fiecare utilizator înregistrat are în loc de parolă un cod secret alcătuit din n cifre. Pentru a se loga pe site, utilizatorul nu trebuie să introducă codul complet, ci pagina generează aleator 3 poziții distincte, *p1*, *p2* și *p3*, astfel încât $1 \leq p1 < p2 < p3 \leq n$ iar utilizatorul trebuie să introducă doar cifrele de pe acele 3 poziții. De exemplu, dacă codul utilizatorului este 987654321 și pagina generează aleator pozițiile 2, 5 și 7, utilizatorul trebuie să introducă cifrele 8, 5, 3. ✓ ?

Mai jos aveți valorile introduse de un utilizator pentru 9 logări pe această pagină:
1, 2, 3
2, 9, 0
6, 3, 2
2, 0, 2
1, 4, 7
9, 3, 2
4, 4, 3
4, 3, 1
5, 6, 0

Presupunând că toate cele 9 logări sunt valide și codul utilizatorului nu a fost schimbat între timp, precizați care dintre următoarele afirmații sunt adevărate.

A. Codul utilizatorului sigur nu conține cifra 8.
B. Cel mai scurt cod posibil are 12 cifre.
C. Cel mai scurt cod posibil conține cifra 2 de minimum 3 ori.
D. Suma cifrelor în cel mai scurt cod posibil poate fi 44.

483. Se consideră subalgoritmul `f(x, n)` unde x, n sunt numere naturale și $x > 0$. ✓ ?

```
Algorithm F(x, n)
   If n = 0 then
      Return 1
   EndIf
   m ← n DIV 2
   p ← f(x, m)
   If n MOD 2 = 0 then
      Return p * p
   EndIf
   Return x * p * p
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează x^n.
B. Dacă în loc de "n **MOD** 2" ar fi "m **MOD** 2" atunci subalgoritmul ar returna x^n.
C. Liniile după autoapelul funcției nu se vor executa niciodată.
D. Subalgoritmul returnează x^n dacă și numai dacă n este par.

484. Se consideră subalgoritmul `f2(a,b)` cu parametrii a și b numere naturale, și subalgoritmul `f(arr, i, n, p)` având ca parametri șirul arr cu n numere întregi ($arr[1], arr[2]$, ..., $arr[n]$), și numerele întregi i și p: ✓ ?

```
Algorithm F2(a, b)
   If a > b then
      Return a
   Else
      Return b
   EndIf
EndAlgorithm
Algorithm F(arr, i, n, p)
   If i = n then
      Return 0
   EndIf
   n1 ← f(arr, i + 1, n, p)
   n2 ← 0
   If p + 1 ≠ i then
      n2 ← f(arr, i + 1, n, i) + arr[i]
   EndIf
   Return f2(n1, n2)
EndAlgorithm
```

Precizați care este rezultatul apelului `f(arr, 1, 9, -10)`, dacă șirul arr conține valorile $(10, 1, 3, 4, 8, 12, 1, 11, 6)$.

A. 24 B. 37 C. 26 D. 56

485. Fie subalgoritmul `verifica(n)`, care primește ca parametru un număr întreg n ✓ ? $(1 \leq n \leq 100000)$ și returnează adevărat dacă n conține o cifră care este egală cu suma celorlalte cifre. De exemplu, `verifica(1517)` returnează adevărat pentru că $7 = 1 + 5 + 1$.

Care din următoarele variante reprezintă implementări corecte ale subalgoritmului `verifica(n)`?

A.

```
Algorithm VERIFICA(n)
    s ← 0
    c ← n
    r ← fals
    While c > 0 execute
        s ← s + c MOD 10
        c ← c DIV 10
    EndWhile
    c ← n
    While c > 0 execute
        d ← c MOD 10
        If d = s − d then
            r ← adevărat
        Else
            r ← fals
        EndIf
        c ← c DIV 10
    EndWhile
    Return r
EndAlgorithm
```

B.

```
Algorithm VERIFICA(n)
    m ← −1
    c ← n
    r ← fals
    While c > 0 execute
        d ← c MOD 10
        c ← c DIV 10
        If d > m then
            m ← d
        EndIf
    EndWhile
    c ← n
    s ← 0
    While c > 0 execute
        d ← c MOD 10
        If d ≠ m then
            s ← s + d
        EndIf
        c ← c DIV 10
    EndWhile
    If s = m then
        r ← adevărat
    EndIf
    Return r
EndAlgorithm
```

C.

```
Algorithm VERIFICA(n)
    v ← [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    r ← fals
    While n > 0 execute
        d ← n MOD 10
        If d > 0 then
            v[d] ← v[d] + 1
        EndIf
        n ← n DIV 10
    EndWhile
    m ← 9
    While v[m] = 0 execute
        m ← m − 1
    EndWhile
    If v[m] = 1 then
        d ← m ▷ Continuarea var. C
        s ← 0
        m ← m − 1
        While m > 0 execute
            s ← s + v[m] ∗ m
            m ← m − 1
        EndWhile
        If d = s then
            r ← adevărat
        EndIf
    EndIf
    Return r
EndAlgorithm
```

D. Niciuna dintre celelalte variante nu este corectă.

486. Se consideră subalgoritmul `f(x, n, e, y, m)`, care primește ca parametri un șir x cu n elemente numere întregi $(x[1], x[2], ..., x[n])$, un șir y cu m elemente numere întregi $(y[1], y[2], ..., y[m])$, și un număr întreg e care nu aparține șirului y. Subalgoritmul returnează un șir și un număr natural. Se dau subalgoritmii: ✓ ?

1. $(c, p) \leftarrow$ `concatenare(a, n, b, m)` care are ca parametri de intrare un șir a cu n elemente numere întregi și un șir b cu m elemente numere întregi, și returnează șirul c cu p elemente numere întregi care reprezintă concatenarea celor două șiruri a și b, adică: $a[1], a[2], ..., a[n], b[1], b[2], ..., b[m]$

2. $(c, p) \leftarrow$ `diferență(a, n, b, m)` care are ca parametri de intrare un șir a cu n elemente numere întregi și un șir b cu m elemente numere întregi, și returnează șirul c cu p elemente numere întregi care conține toate elementele din șirul a (elementele rămase în șir păstrându-și ordinea inițială) care nu sunt în șirul b

```
Algorithm F(x, n, e, y, m)
    If n = 0 then
        Return [], 0
    EndIf
    If x[1] ≠ e then
        s ← []
        s[1] ← x[1]
        (r1, l1) ← diferență(x, n, s, 1)
        (r2, l2) ← f(r1, l1, e, y, m)
        (r3, l3) ← concatenare(s, 1, r2, l2)
        Return r3, l3
    Else
        (r1, l1) ← f(y, m, e, x, n)
        s ← []
        s[1] ← x[1]
        (r2, l2) ← diferență(x, n, s, 1)
        (r3, l3) ← f(r2, l2, e, y, m)
        (r4, l4) ← concatenare(r1, l1, r3, l3)
        Return r4, l4
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul `f(x, n, e, y, m)` construiește un tablou unidimensional pornind de la șirul x în care aparițiile elementului e sunt șterse și în locul fiecărei apariții sunt inserate elementele din y. Subalgoritmul returnează tabloul construit și dimensiunea acestuia.

B. Dacă șirurile x și y nu au elemente comune, atunci șirul returnat de subalgoritmul `f(x, n, e, y, m)` va conține doar elemente distincte.

C. Lungimea șirului returnat de subalgoritmul `f(x, n, e, y, m)`, având ca parametri de intrare șirurile x și y nevide, poate fi mai mică decât n.

D. Dacă pe linia 18, în loc de *r1* și *l1* am avea y și m atunci funcția ar returna un tablou unidimensional (și dimensiunea lui) care ar începe cu elementele din y, urmate de elementele din x, aparițiile lui e fiind înlocuite de elementele din y.

487. Se dă subalgoritmul `s(a, b, c)`, unde a, b, c sunt numere naturale nenule, $b \geq a$ ✓ ?

```
Algorithm s(a, b, c)
   If c = 0 then
      Return 1
   Else
      If a > b then
         Return (1/a) * s(a − 1, b, c)
      Else
         If a < b then
            Return (1/b) * s(a, b − 1, c)
         Else
            Return c * s(a − 1, b − 1, c − 1)
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

Care trebuie să fie relația dintre a, b și c pentru a se obține $1/C_b^a$ (unde C_b^a reprezintă combinări de b elemente luate câte a)

A. $a + b = c$

B. $a + c = b$

C. $b - c = a$

D. $b + c = a - b$

Subiect Concurs Mate-Info UBB 2024

488. Se consideră algoritmul `calcul(v, n)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar v este un vector cu n elemente numere naturale ($v[1], v[2], ..., v[n], 1 \leq v[i] \leq 10^4$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm CALCUL(v, n)
    i ← 1; j ← n
    While i < j execute
        While i < j  AND  v[i] mod 2 =
1 execute
            i ← i + 1
        EndWhile
        While i < j  AND  v[j] mod 2 =
1 execute
            j ← j − 1
        EndWhile
        If v[i] ≠ v[j] then
            Return False
        EndIf
        i ← i + 1
        j ← j − 1
    EndWhile
    Return True
EndAlgorithm
```

Pentru care din următoarele situații algoritmul returnează *True*?

A. Dacă vectorul v este format din valorile $[1, 11, 2, 4, 3, 4, 7, 6, 4, 21, 23, 25, 2]$ și $n = 13$

B. Dacă vectorul v este format din valorile $[1, 11, 2, 4, 3, 7, 6, 4, 21, 23, 25, 2]$ și $n = 12$

C. Dacă și numai dacă valoarea absolută a diferenței dintre două elemente pare ale vectorului v între care există cel puțin un element impar, este egală cu 2

D. Dacă vectorul format din elementele pare ale vectorului v parcurs de la stânga la dreapta este egal cu vectorul format din elementele pare ale vectorului v parcurs de la dreapta la stânga

489. Se consideră algoritmul `g(a, b)` unde a și b sunt numere naturale ($0 \leq a, b \leq 10^4$): ✓ ?

```
Algorithm G(a, b)
    If a = b then
        Return a
    EndIf
    If a > b then
        Return g(a − b, b)
    Else
        Return g(a, b − a)
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `g(2, 2)` algoritmul returnează 2.

B. Dacă $a = b$, algoritmul nu se autoapelează niciodată.

C. Dacă $a = 0$ și $0 \leq b \leq 10^4$, algoritmul se autoapelează o singură dată.

D. Dacă $a \neq 0$, $b \neq 0$ și $a \neq b$, algoritmul se autoapelează de $a + b - 1$ ori.

490. Un graf orientat are 8 vârfuri, numerotate de la 1 la 8, și arcele $(1, 7), (1, 8), (3, 5), (3, 7), (4, 3), (4, 7), (6, 3), (6, 5), (6, 7), (6, 8), (8, 5), (8, 7)$. Numărul vârfurilor care au gradul extern nul este: ✓ ?

A. 1 B. 2 C. 3 D. 4

491. Care este valoarea expresiei **NOT** $((x \bmod 2 = 0)$ **AND** (**NOT** $((y > x)$ **AND** $(x \bmod 7 \neq 5))))$, dacă $x = 12$ și $y = 23$? ✓ ?

A. *True*

B. *False*

C. Aceeași valoare ca a expresiei **NOT** $((x \bmod 2 = 0)$ **AND** (**NOT** $((x > y)$ **AND** $(x \bmod 7 \neq 5))))$

D. Aceeași valoare ca a expresiei **NOT** $((y \bmod 2 = 0)$ **AND** (**NOT** $((x > y)$ **AND** $(y \bmod 7 \neq 5))))$

492. Se consideră algoritmul `ghici(n)`, unde n este număr natural ($1 \leq n \leq 10^9$): ✓ ?

```
Algorithm GHICI(n)
    f ← 0; y ← −1
    For c ← 0, 9 execute
        x ← n
        k ← 0
        While x > 0 execute
            If x mod 10 = c then
                k ← k + 1
            EndIf
            x ← x DIV 10
            If k > f then
                f ← k
                y ← c
            EndIf
        EndWhile
    EndFor
    Return y
EndAlgorithm
```

Precizați ce returnează algoritmul:

A. Numărul de cifre al numărului n

B. Frecvența maximă a frecvențelor cifrelor din numărul n

C. Una dintre cifrele cu frecvența maximă din numărul n

D. Una dintre cifrele cu valoare maximă din numărul n

493. Se consideră algoritmul `divizori(n)`, unde n este număr întreg ($-10^3 \leq n \leq 10^3$): ✓ ?

```
Algorithm DIVIZORI(n)
    nr ← 0; d ← 1
    While d * d ≤ n execute
        If n mod d = 0 then
            nr ← nr + 1
        EndIf
        d ← d + 1
    EndWhile
    Return 2 * nr
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă $n = 5$, algoritmul returnează 2.

B. Dacă $n > 1$, algoritmul returnează numărul tuturor divizorilor (proprii și improprii) ai numărului n.

C. Dacă $n = 0$, algoritmul returnează 0.

D. Dacă $n < 0$, algoritmul returnează numărul tuturor divizorilor (proprii și improprii) corespunzător valorii absolute a lui n.

494. Se consideră algoritmul `ceReturneaza(a, b)`, unde a și b sunt numere naturale ($0 \leq a, b \leq 10^3$): ✓ ?

```
Algorithm ceReturneaza(a, b)
   If a > b then
      c ← a; a ← b; b ← c
   EndIf
   d ← 0
   For i ← a, b execute
      If i MOD 2 = 0 then
         d ← d + 1
      EndIf
   EndFor
   Return d
EndAlgorithm
```

În care din următoarele situații rezultatul returnat este 0?

A. $a = 11$, $b = 11$

B. $a = 4$, $b = 8$

C. $a = 12$, $b = 12$

D. $a = 0$, $b = 0$

495. Se consideră algoritmul `ceFace(n)`, unde n este număr natural ($1 \leq n \leq 10^4$): ✓ ?

```
Algorithm ceFace(n)
   k ← 0; s ← 0
   While k ≠ n execute
      k ← k + 1
      s ← s + 2 * k − 1
      Write s, " "
   EndWhile
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 3$, algoritmul va afișa: 0 9

B. Pentru $n = 10$, penultima valoare atribuită variabilei s în timpul executării este 81

C. Algoritmul afișează pătratele numerelor naturale $1, 2, ..., n$

D. Pentru $n = 4$, algoritmul va afișa: 1 4 8 16

496. Se consideră algoritmii `verificareAux(a, b)` și `verificare(a, b)`, unde a și b ✓ ? sunt numere naturale ($1 \leq a, b \leq 10^9$):

```
Algorithm verificareAux(a, b)
   c ← b
   While a > 0 execute
      While (c > 0) AND (a mod
10 ≠ c mod 10) execute
         c ← c DIV 10
      EndWhile
      If c = 0 then
         Return False
      EndIf
      c ← b
      a ← a DIV 10
   EndWhile
   Return True
EndAlgorithm
Algorithm verificare(a, b)
   Return verificareAux(a, b) AND
verificareAux(b, a)
EndAlgorithm
```

Pentru care dintre condițiile următoare algoritmul `verificare(a, b)` returnează *True*?

A. Dacă a și b au același număr de cifre.

B. Dacă $a = 1001$ și $b = 10$.

C. Dacă vectorul de frecvență a cifrelor lui a este identic cu vectorul de frecvență a cifrelor lui b.

D. Dacă $a = 123$ și $b = 321$.

497. Se consideră algoritmul `verifica(n)`, unde n este număr natural ($1 \leq n \leq 10^4$): ✓ ?

```
Algorithm VERIFICA(n)
    a ← n mod 10
    n ← n DIV 10
    While n > 0 execute
        b ← n mod 10
        If a ≤ b then
            Return False
        EndIf
        a ← b
        n ← n DIV 10
    EndWhile
    Return True
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În urma apelului `verifica(2024)` algoritmul returnează *False*.

B. Algoritmul returnează *True* dacă și numai dacă n este un număr în care cifrele sunt în ordine strict crescătoare.

C. Algoritmul returnează *True* dacă și numai dacă n este un număr în care cifrele sunt în ordine strict descrescătoare.

D. Algoritmul returnează *True* dacă și numai dacă cifra cea mai semnificativă a numărului n este mai mică decât cifra sa cea mai nesemnificativă.

498. Se consideră algoritmul `F(x, n, i, s, k)`, unde x este un vector de n ($1 \leq n \leq 10^4$) numere întregi ($x[1], x[2], ..., x[n], -10^3 \leq x[i] \leq 10^3$ pentru $i = 1, 2, ..., n$), S este număr real, iar i și k sunt numere naturale. Operatorul "/" reprezintă împărțirea reală, de exemplu: $3/2 = 1.5$. ✓ ?

```
Algorithm F(x, n, i, s, k)
    If n < i then
        If k = n then
            Return 0
        Else
            Return s/(n − k)
        EndIf
    Else
        If x[i] mod 2 = 0 then
            Return F(x, n, i+1, s+x[i], k)
        Else
            Return F(x, n, i + 1, s, k + 1)
        EndIf
    EndIf
EndAlgorithm
```

Știind că algoritmul se apelează în forma `F(x, n, 1, 0.0, 0)`, precizați care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează suma numerelor pare din vectorul x, împărțită la numărul numerelor impare din vector.

B. Dacă $n = 10$ și $x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, algoritmul returnează valoarea 6.0.

C. Algoritmul returnează media aritmetică a numerelor pare din vectorul x.

D. Dacă $n = 10$ și $x = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]$, algoritmul returnează valoarea 0.

499. Se consideră o matrice pătrată x de dimensiunea n cu elemente numere naturale distincte ($2 \leq n \leq 50, x[1][1], ..., x[1][n], x[2][1], ..., x[2][n], ..., x[n][1], ..., x[n][n], 1 \leq x[i][j] \leq 10^4$, pentru $i = 1, 2, ..., n, j = 1, 2, ..., n$). Elementele fiecărei linii și elementele fiecărei coloane sunt ordonate crescător. Algoritmul `cauta(n, x, v)` caută o valoare v în matricea x și returnează perechea formată din indicele de linie și indicele de coloană a poziției pe care se află valoarea v în matrice sau $(-1, -1)$ dacă valoarea v nu se află printre elementele matricei. Presupunem că algoritmul `cautareBinara(t, n, v)` implementează algoritmul căutării binare pentru a determina dacă un număr v este prezent în vectorul t cu n elemente ordonate crescător. Dacă v nu se află în a i-a ✓ ?

linie a matricei, în urma apelului `cautareBinara(x[i], n, v)` se returnează -1. Care dintre următorii algoritmi sunt cei mai eficienți din punct de vedere al complexității timp și realizează cerințele descrise?

A.

```
Algorithm CAUTA(n, x, v)
    a ← −1
    b ← −1
    For i ← 1, n execute
        For j ← 1, n execute
            If x[i][j] = v then
                a ← i
                b ← j
            EndIf
        EndFor
    EndFor
    Return a, b
EndAlgorithm
```

B.

```
Algorithm CAUTA(n, x, v)
    a ← −1
    b ← −1
    For i ← 1, n execute
        j ← cautareBinara(x[i], n, v)
        If j ≠ −1 then
            a ← i
            b ← j
        EndIf
    EndFor
    Return a, b
EndAlgorithm
```

C.

```
Algorithm CAUTA(n, x, v)
    a ← −1
    b ← −1
    i ← 1; j ← n
    While i ≤ n AND j > 0
execute
        If x[i][j] = v then
            a ← i
            b ← j
        EndIf
        If x[i][j] > v then
            j ← j − 1
        Else
            i ← i + 1
        EndIf
    EndWhile
    Return a, b
EndAlgorithm
```

D.

```
Algorithm CAUTA(n, x, v)
    a ← −1
    b ← −1
    i ← 1; j ← 1
    While i ≤ n AND x[i][j] < v
execute
        i ← i + 1
    EndWhile
    While j ≤ n AND x[i][j] < v
execute
        j ← j + 1
    EndWhile
    If x[i][j] = v then
        a ← i
        b ← j
    EndIf
    Return a, b
EndAlgorithm
```

500. Fiind dată o matrice pătrată M de 3×3 elemente, care dintre următoarele secvențe de cod implementează corect o rotire cu 90 de grade în sens trigonometric a matricei în jurul elementului de pe poziția $(2, 2)$? ✓ ?

A.
```
For i ← 0, 1 execute
    X ← M[1][1]
    M[1][1] ← M[1][2]
    M[1][2] ← M[1][3]
    M[1][3] ← M[2][3]
    M[2][3] ← M[3][3]
    M[3][3] ← M[3][2]
    M[3][2] ← M[3][1]
    M[3][1] ← M[2][1]
    M[2][1] ← X
EndFor
```

B.
```
For i ← 0, 2 execute
    X ← M[1][1]
    M[1][1] ← M[1][2]
    M[1][2] ← M[1][3]
    M[1][3] ← M[2][3]
    M[2][3] ← M[3][3]
    M[3][3] ← M[3][2]
    M[3][2] ← M[3][1]
    M[3][1] ← M[2][1]
    M[2][1] ← X
EndFor
```

C.
```
For i ← 1, 2 execute
    X ← M[1][1]
    M[1][1] ← M[1][2]
    M[1][2] ← M[1][3]
    M[1][3] ← M[2][3]
    M[2][3] ← M[3][3]
    M[3][3] ← M[3][2]
    M[3][2] ← M[3][1]
    M[3][1] ← M[2][1]
    M[2][1] ← X
EndFor
```

D.
```
For i ← 1, 3 execute
    X ← M[1][1]
    M[1][1] ← M[1][i]
    M[1][i] ← M[1][3]
    M[1][3] ← M[i][3]
    M[i][3] ← M[3][3]
    M[3][3] ← M[3][i]
    M[3][1] ← M[3][1]
    M[3][1] ← M[i][1]
    M[i][1] ← X
EndFor
```

501. Se consideră algoritmul `rearanjeaza(x, n)`, unde n este număr natural ($1 \leq n \leq 200$), iar x este un vector de n numere întregi distincte ($x[1], x[2], ..., x[n], -100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$). Algoritmul `interschimba(x, i, j)` interschimbă elementele *x[i]* și *x[j]*. ✓ ?

```
Algorithm REARANJEAZA(x, n)
    v ← x[n]
    i ← 0; j ← 1
    While j ≤ n − 1 execute
        If x[j] ≤ v then
            i ← i + 1
            interschimba(x, i, j)
        EndIf
        j ← j + 1
    EndWhile
    i ← i + 1
    interschimba(x, i, n)
    Return i
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul sortează în ordine crescătoare elementele vectorului x.

B. Dacă vectorul x este sortat crescător, nu se va modifica ordinea elementelor din vector.

C. Vectorul x va fi rearanjat astfel încât ultimul element din vectorul inițial va avea în stânga sa doar elemente cu valoare mai mică și în dreapta doar elemente cu valoare mai mare.

D. Algoritmul returnează indicele inițial al elementului cu valoarea minimă din vectorul x.

502. Se consideră algoritmul `calcul(v, n)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar v este un vector cu n elemente numere naturale ($v[1], v[2], ..., v[n], 1 \leq v[i] \leq 200$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm CALCUL(v, n)
   If n = 1 then
      Return v[1]
   EndIf
   If v[1] mod v[n] = 0 then
      v[1] ← v[n]
      n ← n − 1
      Return calcul(v, n)
   Else
      aux ← v[n]
      v[n] ← v[1] mod v[n]
      v[1] ← aux
      Return calcul(v, n)
   EndIf
EndAlgorithm
```

Pentru care din următoarele valori algoritmul returnează valoarea 12?

A. $v = [60, 96, 120, 84]$, $n = 4$

B. $v = [75, 24, 12, 84]$, $n = 4$

C. $v = [75, 24, 49, 80]$, $n = 4$

D. $v = [60, 24, 12, 84]$, $n = 4$

503. Se consideră algoritmul `ceFace(n)`, unde n este număr întreg ($-10^4 \leq n \leq 10^4$): ✓ ?

```
Algorithm CEFACE(n)
   If n = 0 then
      Return "0"
   EndIf
   If n < 0 then
      Return "-" + ceFace(−n)
   EndIf
   If n mod 3 = 0 then
      Return ceFace(n DIV 3) + "0"
   EndIf
   If n mod 3 = 1 then
      Return ceFace(n DIV 3) + "1"
   EndIf
   Return ceFace(n DIV 3) + "2"
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă numărul n este o putere a lui 3, șirul de caractere returnat conține un singur caracter "1".

B. Pentru $n = 3$ și $n = -3$ algoritmul `ceFace(n)` returnează valori identice.

C. Dacă $n = 82$, algoritmul returnează "010001".

D. Dacă n este număr negativ, algoritmul intră în ciclu infinit.

504. Se consideră algoritmul `decide(n, x)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar x este un vector cu n elemente numere naturale ($x[1], x[2], ..., x[n], 0 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$). Care dintre următoarele afirmații sunt adevărate? ✓ ?

```
Algorithm DECIDE(n, x)
   a ← x[1]
   i ← 2; j ← 1
   While i ≤ n execute
      If x[i] = a then
         j ← j + 1
      Else
         If j > 0 then
            j ← j − 1
         Else
            a ← x[i]
            j ← 1
```

```
        EndIf
      EndIf
      i ← i + 1
   EndWhile
   i ← 1; j ← 0
   While i ≤ n execute
      If x[i] = a then
         j ← j + 1
      EndIf
      i ← i + 1
   EndWhile
   If j > (n DIV 2) then
      Return a
   Else
      Return −1
   EndIf
EndAlgorithm
```

A. Dacă $n = 5$ și $x = [1, 2, 1, 3, 1]$ algoritmul returnează 1.

B. Dacă $n = 5$ și $x = [1, 2, 2, 3, 1]$ algoritmul returnează -1.

C. Pentru orice vector de intrare algoritmul returnează -1.

D. Algoritmul returnează primul element al vectorului x.

505. Se consideră algoritmul `ceFace(n)`, în cadrul căruia se vor citi n numere, unde n este număr natural ($1 \leq n \leq 10^9$): ✓ ?

```
Algorithm ceFace(n)
   nr ← 0
   Read a
   For i ← 2, n execute
      Read b
      If a ≠ b then
         nr ← nr + 1
      EndIf
      a ← b
   EndFor
   Return nr
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează numărul numerelor care se repetă printre numerele citite (de exemplu, dacă numerele sunt 3, 34, 34, 7, 3, 34, atunci valoarea returnată este 2).

B. Algoritmul returnează lungimea celei mai lungi subsecvențe de numere citite ce au valori egale (de exemplu, dacă numerele sunt 2, 34, 34, 34, 5, atunci valoarea returnată este 3).

C. Algoritmul returnează numărul perechilor de elemente consecutive cu valori diferite printre numerele citite (de exemplu, dacă numerele sunt 2, 34, 34, 7, atunci (2, 34), (34, 7) sunt perechi de elemente consecutive cu valori diferite și se returnează 2).

D. Algoritmul returnează numărul perechilor de elemente consecutive cu valori egale printre numerele citite (de exemplu, dacă numerele sunt 2, 2, 3, 3, atunci (2, 2), (3, 3) sunt perechi de elemente consecutive cu valori egale și se returnează 2).

506. Se consideră algoritmul `f(a)`, unde a este un număr natural ($0 \leq a \leq 10^4$): ✓ ?

```
Algorithm F(a)
    n ← 0
    While a > 1 execute
        b ← 1
        While b ≤ a execute
            b ← 3 * b
            n ← n + 1
        EndWhile
        a ← a DIV 3
    EndWhile
    Return n
EndAlgorithm
```

Care este valoarea returnată de algoritm dacă se apelează pentru $a = 81$?

A. 0

B. 14

C. 16

D. 9

507. Se consideră algoritmul `h(n, a)`, unde n este număr natural ($1 \leq n \leq 10^3$), iar a este un vector cu n elemente numere întregi ($a[1], a[2], ..., a[n], -10^4 \leq a[i] \leq 10^4$, pentru $i = 1, 2, ..., n$) ordonate crescător. ✓ ?

```
Algorithm H(n, a)
    t ← 0; i ← n
    While i > 2 execute
        k ← 1
        j ← i − 1
        b ← a[i]
        While k < j execute
            If a[k] + a[j] = b then
                t ← t + 1
                k ← k + 1
                j ← j − 1
            Else
                If a[k] + a[j] < b then
                    k ← k + 1
                Else
                    j ← j − 1
                EndIf
            EndIf
        EndWhile
        i ← i − 1
    EndWhile
    Return t
EndAlgorithm
```

Care dintre următoarele apeluri va returna valoarea 4?

A. `h(5, [1, 2, 3, 4, 5])`

B. `h(6, [2, 4, 6, 10, 18, 20])`

C. `h(7, [2, 2, 3, 4, 6, 9, 13])`

D. `h(5, [2, 2, 2, 4, 6])`

508. Se consideră algoritmul `f(x, n, m)`, unde n și m sunt numere naturale ($1 \leq n, m \leq 10^4$), iar x este un vector de n numere naturale ($x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm F(x, n, m)
   If m = 0 then
      Return 1
   EndIf
   If n = 0 then
      Return 0
   EndIf
   If x[n] > m then
      Return f(x, n − 1, m)
   Else
      Return f(x, n − 1, m) + f(x, n −
1, m − x[n])
   EndIf
EndAlgorithm
```

Ce valoare va returna algoritmul, dacă apelul are forma `f(x, 9, 41)`, unde $x = [41, 15, 5, 8, 10, 1, 16, 18, 19]$?

A. 1 B. 3 C. 5 D. 7

509. Se consideră algoritmul `select(v, x, n)`, unde n este număr natural ($1 \leq n \leq 10^4$), v este un vector cu n elemente numere întregi ($v[1], v[2], ..., v[n], -100 \leq v[i] \leq 100$, pentru $i = 1, 2, ..., n$), iar x este un număr întreg, $-100 \leq x \leq 100$: ✓ ?

```
Algorithm SELECT(v, x, n)
   i ← 1; j ← n
   While i ≤ j execute
      k ← (i + j) DIV 2
      If v[k] = x then
         Return k
      EndIf
      If v[i] ≤ v[k] then
         If v[i] ≤ x AND x < v[k]
then
            j ← k − 1
         Else
            i ← k + 1
         EndIf
      Else
         If v[k] < x AND x ≤ v[j]
then
            i ← k + 1
         Else
            j ← k − 1
         EndIf
      EndIf
   EndWhile
   Return −1
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În cazul apelului `select([0, 1, 2, 4, 5, 8, 9, 10, 7, 6], 10, 10)`, algoritmul returnează 10.

B. Algoritmul returnează poziția pe care apare elementul x în vectorul v dacă și numai dacă vectorul v este sortat crescător.

C. Complexitatea algoritmului este $O(\log n)$.

D. În cazul apelului `select([0, 1, 2, 4, 5, 8, 9, 10, 7, 6], 7, 10)`, algoritmul returnează -1.

510. Se consideră algoritmul `maiMare(n)` unde n este număr natural nenul ($1 \leq n < 10^6$) alcătuit din cifre distincte. Algoritmul ar trebui să returneze numărul numerelor strict mai mari ca n, formate din cifrele lui n. De exemplu, `maiMare(213)` = 3. Presupunem că n nu are cifre 0 la început și că avem următorii algoritmi implementați conform specificațiilor: ✓ ?

- `factorial(n)` – returnează factorialul numărului natural n ($1 \leq n \leq 10$) ✓ ?
- `nrCifre(n)` – returnează numărul cifrelor numărului natural n ($1 \leq n < 10^6$) ✓ ?

- `imparte(n)` – returnează un vector având ca elemente cifrele numărului natural n ($1 \leq n < 10^6$), în ordine inversă. De exemplu: `imparte(1352)` returnează vectorul $[2, 5, 3, 1]$. ✓ ?

```
1: Algorithm MAIMARE(n)
2:     cifre ← imparte(n)
3:     nrCif ← nrCifre(n)
4:     Return calculeaza(cifre,
   nrCif)
5: EndAlgorithm
```

```
1: Algorithm CALCULEAZA(v, n)
2:     If n < 2 then
3:         Return 0
4:     EndIf
5:     mm ← 0
6:     For i ← 1, n − 1 execute
7:         If v[i] > v[n] then
8:             mm ← mm + 1
9:         EndIf
10:    EndFor
11:    ...
12: EndAlgorithm
```

Care dintre următoarele instrucțiuni trebuie scrisă la linia 11 a algoritmului `calculeaza(v, n)`?

A. `Return factorial(n) - ((n - mm - 1) * factorial(n - 1) + calculeaza(v, n - 1))`

B. `Return calculeaza(v, n - 1) * mm + factorial(n - 1)`

C. `Return (mm * factorial(n) + calculeaza(v, n - 1)) DIV n`

D. `Return calculeaza(v, n - 1) + mm * factorial(n - 1)`

511. Un eveniment trebuia să aibă loc în sala I, dar trebuie mutat în sala II, unde numerotarea scaunelor diferă. În ambele săli există L rânduri de scaune ($2 \leq L \leq 50$), fiecare rând fiind împărțit la mijloc de un culoar și având K scaune ($2 \leq K \leq 50$) în fiecare parte a culoarului (deci, sala conține în total $2 * K * L$ scaune). În sala II fiecare loc este identificat printr-un singur număr. Locurile din stânga culoarului au numere pare, iar numerotarea scaunelor începe pe rândul din fața scenei. Deci scaunele din primul rând au numerele (pornind dinspre culoar spre marginea sălii) 2, 4, 6 etc. După ce toate scaunele de pe un rând au fost numerotate, pe rândul următor se continuă numerotarea, reîncepând cu scaunul de lângă culoar cu următorul număr par. Locurile din partea dreaptă a culoarului sunt numerotate la fel, dar folosind numere impare. Deci scaunele din primul rând au numerele 1, 3, 5, etc, pornind dinspre culoar spre marginea sălii. În sala I fiecare loc este identificat prin trei valori. Numărul rândului (o valoare între 1 și L inclusiv, rândul 1 fiind cel din fața scenei), direcția locului față de culoar (valoarea "stanga" sau "dreapta") și numărul scaunului în cadrul rândului (o valoare între 1 și K inclusiv, scaunul 1 fiind cel de lângă culoar). Din cauza mutării spectacolului, locurile de pe bilete din sala I (reprezentate prin *rand*, *loc*, *directie*) trebuie transformate în locuri valabile în sala II (un singur număr). Care dintre algoritmii de mai jos, având ca date de intrare K, *rand*, *loc*, *directie* conform enunțului, execută în mod corect transformarea (o transformare este corectă dacă fiecare spectator va avea un loc unic în sala II)? ✓ ?

A.

```
Algorithm TRANSFORMA(K, rand,
loc, directie)
   If directie = "stanga" then
      rez ← 2*(loc+K*(rand−1))
   Else
      rez ← 2 * (loc + K * (rand −
1) + 1)
   EndIf
   Return rez
EndAlgorithm
```

B.

```
Algorithm TRANSFORMA(K, rand,
loc, directie)
   rez ← rand * (K − 1) * 2
   rez ← rez + (loc * 2)
   If directie = "dreapta" then
      rez ← rez − 1
   EndIf
   Return rez
EndAlgorithm
```

C.

```
Algorithm TRANSFORMA(K, rand,
loc, directie)
   rez ← (rand − 1) * K * 2
   rez ← rez + (loc * 2)
   If directie = "dreapta" then
      rez ← rez − 1
   EndIf
   Return rez
EndAlgorithm
```

D.

```
Algorithm TRANSFORMA(K, rand,
loc, directie)
   rez ← (rand − 1) * K * 2
   rez ← rez + (loc * 2)
   If directie = "dreapta" then
      rez ← rez + 1
   EndIf
   Return rez
EndAlgorithm
```

Subiect Concurs Mate-Info UBB 2023

512. Se consideră algoritmul `f(a, b)`, unde a și b sunt numere naturale nenule ($1 \leq a, b \leq 10^9$): ✓ ?

```
1: Algorithm F(a, b)
2:     If a = b then
3:         Return a
4:     EndIf
5:     If a > b then
6:         Return f(a − b, b)
7:     EndIf
8:     Return f(a, b − a)
9: EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate.

A. În urma apelului `f(2000, 21)` algoritmul returnează 1.

B. În cazul apelului `f(2000, 21)` algoritmul nu își termină execuția din cauza condiției de pe linia 2.

C. Pentru ca algoritmul să returneze cel mai mare divizor comun al lui a și b, linia 8 ar trebui schimbată astfel: `Return f(b - a, b)`.

D. Pentru ca în cazul apelului `f(2000, 21)` algoritmul să returneze valoarea 1, linia 8 ar trebui schimbată astfel: `Return f(b - a, b - a)`.

513. Se consideră următoarea secvență de algoritm, unde a este un vector de n numere naturale ($a[1], a[2], ..., a[n], 1 \leq a[i] \leq 10^4$, pentru $i = 1, 2, ..., n$), iar n este un număr natural nenul ($1 \leq n \leq 10^4$): ✓ ?

```
For i ← 1, n − 1 execute
    poz ← i
    For j ← i + 1, n execute
        If a[j] < a[poz] then
            poz ← j
        EndIf
    EndFor
    If poz ≠ i then
        temp ← a[i]
        a[i] ← a[poz]
        a[poz] ← temp
    EndIf
EndFor
```

Care dintre următoarele afirmații sunt adevărate în momentul în care i devine 2?

A. $a[1] \leq a[k]$ pentru orice $k \in 1, 2, ..., n$

B. $a[n] \leq a[k]$ pentru orice $k \in 1, 2, ..., n$

C. $a[1] \geq a[k]$ pentru orice $k \in 1, 2, ..., n$

D. $a[k] \leq a[k + 1]$ pentru orice $k \in 1, 2, ..., n - 1$

514. Se consideră algoritmul `alg(n)`, unde n este un număr natural ($0 \leq n \leq 10^9$): ✓ ?

```
Algorithm ALG(n)
    If n MOD 2 = 0 then
        Return n + alg(n − 1)
    Else
        Return n
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă $n = 4$, valoarea returnată de algoritm este 7.

B. Algoritmul returnează suma numerelor naturale mai mici decât n.

C. Algoritmul returnează suma numerelor naturale mai mici sau egale cu n.

D. Dacă $n = 7$, valoarea returnată de algoritm este 7.

515. Se consideră algoritmul `f(nr)`, unde nr este un număr întreg ($-10^4 \leq nr \leq 10^4$): ✓ ?

```
Algorithm F(nr)
   If nr < 0 then
      Return f(-nr)
   EndIf
   If (nr = 0) OR (nr = 7) then
      Return 1
   EndIf
   If nr < 10 then
      Return 0
   EndIf
   Return f((nr DIV 10) - 2 *
(nr MOD 10))
EndAlgorithm
```

Pentru ce valori ale lui nr algoritmul returnează valoarea 1?

A. 308
B. -7
C. 7098
D. 57

516. Se consideră algoritmul `afis(n)`, unde n este un număr natural ($1 \leq n \leq 10^4$): ✓ ?

```
Algorithm AFIS(n)
   If n > 9 then
      If n MOD 2 = 0 then
         afis(n DIV 100)
         Write n MOD 10, " "
      Else
         afis(n DIV 10)
      EndIf
   EndIf
EndAlgorithm
```

Pentru care dintre următoarele apeluri se afișează valorile 2 4, în această ordine?

A. `afis(1234)`
B. `afis(1224)`
C. `afis(4224)`
D. `afis(4321)`

517. Se consideră algoritmul `Afișare(a)`, unde a este un număr natural ($1 \leq a \leq 10^4$): ✓ ?

```
Algorithm AFIȘARE(a)
   If a < 9000 then
      Write a, " "
      Afișare(3 * a)
      Write a, " "
   EndIf
EndAlgorithm
```

Ce se afișează pentru apelul `Afișare(1000)`?

A. 1000 3000 9000 9000 3000 1000
B. 1000 3000 9000 3000 1000
C. 1000 3000 3000 1000
D. 1000 3000 9000

518. Se consideră algoritmul `f(n, x)`, unde n este un număr natural ($3 \leq n \leq 10^4$), iar x este un vector de n numere naturale ($x[1], x[2], ..., x[n], 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm F(n, x)
   For i ← 1, n - 2 execute
      If x[i] + x[i + 1] ≠ x[i + 2] then
         Return False
      EndIf
   EndFor
   Return True
EndAlgorithm
```

Pentru care dintre următoarele apeluri algoritmul va returna *True*?

A. `f(3, [10, 15, 251])`

B. `f(4, [0, 0, 0, 0])`

C. `f(5, [100, 535, 635, 1170, 1805])`

D. `f(4, [0, 1, 0, 1])`

519. Care este rezultatul conversiei numărului zecimal $2^{10} - 2^5 - 1$ în baza 2? ✓ ?

A. 1111011111

B. 1010011001

C. 1000011001

D. Niciunul dintre răspunsurile A, B, C

520. Se consideră algoritmii `one(a, b)` și `two(n, m)` unde parametrii de intrare a, b, n și m sunt numere naturale ($2 \leq a, b, n, m \leq 10^6, n < m$). ✓ ?

```
Algorithm ONE(a, b)
    s ← 0
    For i ← 1, a execute
        If a MOD i = 0 then
            s ← s + i
        EndIf
    EndFor
    For i ← 1, b execute
        If b MOD i = 0 then
            s ← s + i
        EndIf
    EndFor
    Return s
EndAlgorithm

Algorithm TWO(n, m)
    For i ← n, m execute
        If one(i, i) = 2 * i + 2 then
            Write i, " "
        EndIf
    EndFor
EndAlgorithm
```

Care dintre afirmațiile de mai jos sunt adevărate?

A. Algoritmul `two(n, m)` nu afișează nimic, indiferent de valoarea parametrilor de intrare.

B. Algoritmul `two(n, m)` afișează numerele prime din intervalul $[n, m]$.

C. Algoritmul `two(n, m)` afișează numerele divizibile cu 2 din intervalul $[n, m]$.

D. Nici una din celelalte variante nu este corectă.

521. Se consideră algoritmul `decide(n, x)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$), iar x este un vector cu n elemente numere naturale ($x[1], x[2], ..., x[n], 0 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm DECIDE(n, x)
    i ← 1
    j ← n
    While i < j  AND
          x[i] = x[j] execute
        i ← i + 1
        j ← j − 1
    EndWhile
    If i ≥ j then
        Return True
    Else
        Return False
    EndIf
EndAlgorithm
```

Când returnează *True* algoritmul `decide(n, x)`?

A. Întotdeauna

B. Dacă elementele vectorului x sunt $[1, 2, 3]$

C. Dacă elementele vectorului x sunt $[1, 1, 1]$

D. Dacă elementele vectorului x formează un palindrom, adică $x[i] = x[n - i + 1]$ pentru orice $i = 1, 2, ..., n$

522. Se consideră algoritmul `alg(a, b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10^3$): ✓ ?

```
Algorithm ALG(a, b)
    If b = 0 then
        Return 1
    Else
        Return a ∗ alg(a, b − 1)
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru apelul `alg(2, 3)` algoritmul returnează 7.

B. Pentru apelul `alg(2, 3)` algoritmul se apelează de 4 ori, luând în calcul și apelul inițial.

C. Algoritmul calculează și returnează valoarea a^{b-1}.

D. Algoritmul calculează și returnează valoarea a^b.

523. Se consideră algoritmul `ceFace(a, b)`, unde a și b sunt numere naturale ($1 < a, b \leq 10^5$). Algoritmul `prim(n)` returnează *True* dacă numărul n ¿ 1 este prim și *False* altfel. ✓ ?

```
Algorithm CEFACE(a, b)
    If prim(a) = True then
        Write a, " "
    Else
        If prim(b) ≠ True then
            ceFace(a, b + 1)
        Else
            If b > a then
                Write a, " "
            Else
                If a MOD b = 0 then
                    Write b, " "
                    ceFace(a DIV b, b)
                Else
                    ceFace(a, b + 1)
                EndIf
            EndIf
        EndIf
    EndIf
EndAlgorithm
```

Ce se afișează pentru apelul `ceFace(100, 2)`?

A. 2 5 5 5

B. 5 5 2 2

C. 2 2 2 5

D. 2 2 5 5

524. Se consideră algoritmul `f(n, p)` unde n este un număr natural nenul ($1 \leq n \leq 10^9$), iar p este un număr natural ($0 \leq p \leq 10^9$): ✓ ?

```
Algorithm F(n, p)
   If n ≤ 9 then
      If n MOD 2 = 0 then
         Return 10 * p + n
      Else
         Return p
      EndIf
   Else
      If n MOD 2 = 0 then
         p ← p * 10 + n MOD 10
      EndIf
      Return f(n DIV 10, p)
   EndIf
EndAlgorithm
```

Care din următoarele apeluri vor returna valoarea 22?

A. `f(23572, 0)`

B. `f(23527, 0)`

C. `f(2, 0)`

D. `f(1242, 0)`

525. Se consideră algoritmul `cifre(n)`, unde n este un număr natural ($0 \leq n \leq 10^3$): ✓ ?

```
Algorithm CIFRE(n)
   If n ≥ 1 then
      If (n * 5) MOD 10 = 0 then
         Return cifre(n DIV 10)
      Else
         Return n MOD 10
      EndIf
   Else
      Return −1
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează întotdeauna un număr mai mic decât 10.

B. Algoritmul returnează -1 dacă și numai dacă valoarea inițială a lui n este 0.

C. Pentru $n \geq 1$, algoritmul returnează cifra cea mai puțin semnificativă a lui n care este impară, sau -1, dacă aceasta nu există.

D. Pentru $n \geq 1$ algoritmul returnează cifra cea mai semnificativă a lui n care este impară, sau -1, dacă aceasta nu există.

526. Se consideră algoritmul `ceFace(a, b)`, unde a și b sunt două numere naturale ($0 \leq a, b \leq 10^6$): ✓ ?

```
Algorithm CEFACE(a, b)
   c ← 0
   p ← 1
   While a * b ≠ 0 execute
      If (a MOD 10) = (b MOD 10) then
         c ← (a MOD 10) * p + c
      Else
         If (a MOD 10) < (b MOD 10) then
            c ← ((b MOD 10 − a MOD 10) DIV 2) * p + c
         Else
            c ← ((a MOD 10 − b MOD 10) DIV 2) * p + c
         EndIf
```

```
        EndIf
        p ← p * 10
        a ← a DIV 10
        b ← b DIV 10
    EndWhile
    Return c
EndAlgorithm
```

Care dintre următoarele afirmații sunt corecte?

A. Dacă $a = 0$ și $b = 0$, algoritmul returnează 1.

B. Dacă $a = 11$ și $b = 111$, algoritmul returnează 11.

C. Dacă $a = 5678$ și $b = 5162738$, algoritmul returnează 1024.

D. Dacă $a = 112233$ și $b = 331122$, algoritmul returnează 110000.

527. Se consideră algoritmii `ceva(n, m)` și `altceva(n, m)`, unde n și m sunt numere naturale nenule ($1 \leq n, m \leq 10^{12}$ și $m \leq n$): ✓ ?

```
Algorithm CEVA(n, m)
    nc ← n
    mc ← m
    While nc > 0  AND
          mc > 0 execute
        nc ← nc DIV 10
        mc ← mc DIV 10
    EndWhile
    If nc = mc then
        Return True
    Else
        Return False
    EndIf
EndAlgorithm

Algorithm ALTCEVA(n, m)
    c ← 0
    While ceva(n, m) = False execute
        m ← m * 10 + 1
        c ← c + 1
    EndWhile
    Write n, " ", m
    Return c
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Complexitatea timp a algoritmului `ceva(n, m)` este $O(\log m)$.

B. Algoritmul `altceva(n, m)` returnează 0 dacă și numai dacă $n = m$.

C. Avem nevoie de precondiția $m \leq n$ din enunț, deoarece dacă $m > n$ algoritmul `altceva(n, m)` intră întotdeauna în ciclu infinit.

D. Există numere n și m (care respectă precondiția) pentru care algoritmul `altceva(n, m)` afișează două valori în ordine crescătoare.

528. Se consideră algoritmul `h(s, d, A)`, unde s și d sunt numere naturale nenule ($1 \leq s, d \leq 10^3$) și A este un vector de n numere naturale nenule ($A[1], A[2], ..., A[n], 1 \leq A[i] \leq 10^3$, pentru $i = 1, 2, ..., n$): ✓ ?

```
Algorithm H(s, d, A)
   If s = d then
      x ← A[s]
      y ← x MOD 10
      x ← x DIV 10
      While x > 0 execute
         z ← x MOD 10
         If z − y ≠ 2 then
            Return 0
         EndIf
         y ← z
         x ← x DIV 10
      EndWhile
      Return 1
   Else
      Return h(s, (s + d) DIV 2, A)+
          h((s + d) DIV 2 + 1, d, A)
   EndIf
EndAlgorithm
```

Pentru ce valori ale numărului n și a vectorului A apelul `h(1, n, A)` va returna valoarea 5?

A. $n = 7$, $A = (20, 53, 10, 42, 31, 131, 42)$

B. $n = 10$, $A = (420, 75, 68, 86, 97, 975, 53, 64, 24, 57)$

C. $n = 10$, $A = (402, 75, 6, 86, 7, 9, 35, 46, 24, 57)$

D. $n = 10$, $A = (642, 97, 6, 64, 7, 9, 75, 4, 53, 31)$

529. Se consideră algoritmul `f(a, x)`, unde x este un număr natural nenul ($1 \leq x \leq 10^4$) ✓ ? și a este un vector cu 10 numere naturale nenule ($a[1], a[2], ..., a[10]$):

```
Algorithm F(a, x)
   i ← 1, j ← 10
   k ← 1
   While a[k] ≠ x  AND  i < j execute
      k ← (i + j)  DIV  2
      If a[k] < x then
         i ← k
      Else
         j ← k
      EndIf
   EndWhile
   If a[k] = x then
      Return True
   Else
      Return False
   EndIf
EndAlgorithm
```

Pentru care dintre următoarele date de intrare algoritmul intră în ciclu infinit?

A. $a = [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]$ și $x > 3$

B. $a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ și $x < 10$

C. $a = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]$ și $1 < x < 20$, x - număr impar

D. $a = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]$ și $1 < x < 20$, x - număr par

530. Se consideră algoritmul `f(a)`, unde a este un număr natural ($1 \leq a \leq 10^9$): ✓ ?

```
Algorithm F(a)
   x ← a  MOD  10
   If x = a then
      If x  MOD  2 = 0 then
         Return a
```

```
      Else
         Return 0
      EndIf
   EndIf
   If x  MOD  2 = 0 then
      Return 10 * f(a  DIV  10) + x
   EndIf
   Return f(a  DIV  10)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $a = 253401976$ algoritmul f(a) se apelează de 8 ori. Se numără și apelul inițial.

B. Pentru $a = 253401976$ algoritmul f(a) se apelează de 9 ori. Se numără și apelul inițial.

C. Pentru $a = 253401976$ rezultatul returnat de algoritm este 2406.

D. Rezultatul returnat de algoritmul f(a) pentru numărul a format doar din cifre pare este egal cu a.

531. Se consideră algoritmul `A(k)`, unde parametrul k este un număr natural nenul ($1 \leq k \leq 10^9$): ✓ ?

```
Algorithm A(k)
   gr ← (-1 + radical(1 + 8 * k))
/ 2
   If gr = [gr] then
      p ← gr
   Else
      p ← [gr] + 1
   EndIf
   Return p - (k - p * (p - 1)
DIV  2 - 1)
EndAlgorithm
```

Cu $[gr]$ s-a notat partea întreagă din gr.
Algoritmul `radical(x)` returnează valoarea radicalului lui x.
Operatorul / reprezintă împărțirea numerelor reale, de exemplu: 7 / 2 = 3.5

Care dintre următoarele afirmații sunt corecte?

A. Algoritmul A1(k), definit mai jos, este echivalent cu algoritmul A(k).

```
Algorithm A1(k)
   c ← 0
   i ← 1
   While c < k  execute  execute
      j ← 1
      While j ≤ i  execute  execute
         If c < k then
            c ← c + 1
            If c = k then
               Return j
            Else
               j ← j + 1
            EndIf
         Else
            Return j
         EndIf
```

```
      EndWhile
      i ← i + 1
   EndWhile
EndAlgorithm
```

B. Algoritmul A2(k), definit mai jos, este echivalent cu algoritmul A(k).

```
Algorithm A2(k)
   c ← 0
   i ← 1
   While c < k  execute  execute
      j ← i
      While j ≥ 1  execute  execute
         If c < k then
            c ← c + 1
            If c = k then
               Return j
            Else
               j ← j - 1
            EndIf
         Else
            Return j
         EndIf
      EndWhile
      i ← i + 1
   EndWhile
EndAlgorithm
```

C. Algoritmul A(k) returnează al k-lea termen din șirul format din concatenarea șirurilor de forma [1, 2, ..., i], pentru fiecare i = 1, 2, ..., k, în această ordine (adică șirul [1, 1, 2, 1, 2, 3, 1, 2, 3, 4, ...]).

D. Algoritmul A(k) returnează al k-lea termen din șirul format din concatenarea șirurilor de forma [i, ..., 2, 1] pentru fiecare i = 1, 2, ..., k, în această ordine (adică șirul [1, 2, 1, 3, 2, 1, 4, 3, 2, 1, ...])

532. Se consideră algoritmul `ceFace(a, lung)`, unde *lung* este un număr natural ($1 \leq$ ✓ ? $lung \leq 10^5$), iar a este un vector cu *lung* elemente numere întregi ($a[1], a[2], ..., a[lung]$). În vectorul a se află cel puțin un număr pozitiv.

```
Algorithm ceFace(a, lung)
   value1 ← 0
   value2 ← 0
   For i ← 1, lung  execute
execute
      value2 ← value2 + a[i]
      If value1 < value2 then
         value1 ← value2
      EndIf
      If value2 < 0 then
         value2 ← 0
      EndIf
   EndFor
   Return value1
EndAlgorithm
```

Știind că o subsecvență a vectorului $x = [x[1], x[2], ..., x[n]]$ este formată din elemente ale vectorului x care ocupă poziții consecutive (de exemplu $y = [x[3], x[4], x[5], x[6]]$ este o subsecvență a vectorului x de lungime 4) precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă în vectorul a există un singur număr pozitiv, algoritmul returnează valoarea acestuia.

B. Algoritmul returnează lungimea uneia dintre subsecvențele care au suma maximă în vectorul a.

C. Algoritmul returnează suma uneia dintre subsecvențele care au sumă maximă în vectorul a.

D. Algoritmul returnează suma numerelor pozitive aflate pe poziții consecutive la finalul vectorului a.

533. Se consideră algoritmul `ceFace(sir, a, b)`, unde *sir* este un vector format din n ✓ ? $(1 \leq n \leq 100)$ numere naturale nenule distincte ordonate crescător $(sir[1], sir[2], ..., sir[n])$, a și b sunt numere naturale $(1 \leq a, b \leq n)$:

```
Algorithm ceFace(sir, a, b)
   If a > b then
      Return a
   EndIf
   c ← a + (b - a)  DIV  2
   If sir[c] = c then
      Return ceFace(sir, c + 1, b)
   Else
      Return ceFace(sir, a, c - 1)
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate, considerând că apelul inițial este ceFace(sir, 1, n)?

A. Dacă vectorul *sir* este format din primele n numere naturale distincte, atunci algoritmul returnează valoarea $n + 1$.

B. Algoritmul returnează cea mai mare poziție p mai mică sau egală cu n **DIV** 2 pentru care $sir[p] = p$ sau 1, dacă nu există o astfel de poziție $(1 \leq p \leq n)$.

C. Algoritmul returnează cea mai mare poziție p mai mică sau egală cu n **DIV** 2 pentru care $sir[p] \neq p$ sau $n + 1$, dacă nu există o astfel de poziție $(1 \leq p \leq n)$.

D. Algoritmul returnează cel mai mic număr natural nenul care nu apare în vectorul *sir*.

534. Se consideră algoritmul `ceFace(s, x, c, y, n, m, k)`, unde s este șir de ca- ✓ ? ractere $(s[1], s[2], ..., s[x])$ de lungime x, iar c este șir de caractere $(c[1], c[2], ..., c[y])$

de lungime y. Identificatorii x, y, n, m și k memorează numere naturale nenule ($1 \leq x, y, n, m, k \leq 100$).

```
1: Algorithm ceFace(s, x, c, y, n, m, k)
2:    If (n ≥ 0)  AND  (m ≥ 0)  AND  (n ≤ x)  AND  (m ≤ y) then
3:       If k  MOD  2 = 0 then
4:          Write s[(n + k)  MOD  x + 1]
5:          ...
6:          ceFace(s, x, c, y, n - 1, m, k)
7:       EndIf
8:       If k  MOD  2 = 1 then
9:          Write c[(m + k)  MOD  y + 1]
10:         ...
11:         ceFace(s, x, c, y, n, m - 1, k)
12:      EndIf
13:   EndIf
14: EndAlgorithm
```

Dorim, ca în urma apelului ceFace("+-", 2, "123", 3, 2, 2, 4) să obținem o expresie aritmetică validă (adică o expresie aritmetică reprezentând o alternanță din câte un operator și câte un operand; poate să înceapă cu unul din operatorii '+' sau '-' și trebuie să se termine cu un operand). Care dintre următoarele afirmații NU sunt adevărate?

A. Liniile 5 și 10 pot fi completate cu instrucțiunea k ← k + 7.

B. Linia 5 poate fi completată cu instrucțiunea k ← k + 2, iar linia 10 cu instrucțiunea k ← k + 5.

C. Liniile 5 și 10 pot fi completate cu instrucțiunea k ← k + 2.

D. Linia 5 poate fi completată cu instrucțiunea k ← k + 7, iar linia 10 cu instrucțiunea k ← k - 1.

535. Se consideră numărul natural n ($1 \leq n \leq 50$) și vectorul x având n elemente numere întregi ($x[1], x[2], ..., x[n]$). Care dintre următoarele afirmații sunt adevărate, indiferent de valoarea lui n și de valorile elementelor vectorului? ✓ ?

A. Există un număr natural k ($1 \leq k \leq n$), astfel încât suma $x[1] + x[2] + ... + x[k]$ să fie divizibilă cu n.

B. Există (i, j), $0 \leq i < j \leq n$, astfel încât suma $x[i + 1] + x[i + 2] + ... + x[j]$ să fie divizibilă cu n.

C. Niciuna dintre afirmațiile A și B nu este adevărată.

D. Știind că o subsecvență a vectorului $x = [x[1], x[2], ..., x[n]]$ este formată din elemente ale vectorului x care ocupă poziții consecutive (de exemplu, $y = [x[3], x[4], x[5], x[6]]$ este o subsecvență a vectorului x de lungime 4), există un număr natural k, ($1 \leq k \leq n$), astfel încât în vectorul x există o subsecvență de k elemente ($1 \leq k \leq n$) a căror sumă este divizibilă cu n.

Subiect Concurs Mate-Info UBB 2022

536. Se consideră algoritmul `magic(x)`, unde x este un număr natural ($1 \leq x \leq 32000$). ✓ ?

```
Algorithm MAGIC(x)
    st ← 1
    dr ← x
    While st ≤ dr execute
        mj ← (st + dr) DIV 2
        If mj * mj = x then
            Return True
        EndIf
        If mj * mj < x then
            st ← mj + 1
        Else
            dr ← mj − 1
        EndIf
    EndWhile
    Return False
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru orice valoare de intrare x strict mai mică decât 10 algoritmul returnează **False**.

B. Algoritmul descompune numărul x în factorii săi primi.

C. Algoritmul returnează **True** dacă numărul x este pătrat perfect.

D. Algoritmul nu returnează **True** pentru nici o valoare validă a parametrului de intrare x.

537. Se consideră algoritmul `calculeaza(a,b)`, unde a și b sunt numere naturale ($1 \leq a, b \leq 10000$). ✓ ?

```
Algorithm CALCULEAZA(a, b)
    x ← 1
    For i ← 1 to b execute
        x ← (x MOD 10) * a
    EndFor
    Return x
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $a = 107$ și $b = 101$, valoarea returnată este 107.

B. Dacă $a = 1001$ și $b = 101$, valoarea returnată este 1001.

C. Pentru toate apelurile algoritmului cu $1 \leq a \leq 10000$ și $b = 101$, valoarea returnată este valoarea lui a.

D. Pentru toate apelurile algoritmului cu $a = 1001$ și $1 \leq b \leq 10000$, valoarea returnată este 1001.

538. Se consideră algoritmul `afis(n)`, unde n este un număr natural ($0 \leq n \leq 10000$). ✓ ?

```
Algorithm AFIS(n)
    Write n, " "
    If n > 0 then
        AFIS(n  DIV  2)
        Write n, ", "
    EndIf
EndAlgorithm
```

Ce se va afișa la apelul `afis(n)`?

A. Se afișează un șir de numere în care primul element este egal cu ultimul, al doilea cu penultimul etc. (cu excepția elementului din mijloc).

B. Se afișează un șir de numere pare.

C. Se afișează un șir de numere în ordine crescătoare urmate de numere în ordine descrescătoare.

D. Se afișează un șir de numere în ordine descrescătoare urmate de numere în ordine crescătoare.

539. Se consideră algoritmul `cauta(n, b)`, unde n și b sunt numere naturale ($0 \leq n \leq 10^6, 2 \leq b < 10$). ✓ ?

```
Algorithm CAUTA(n, b)
    v ← 0
    If n = 0 then
        Return 1
    Else
        m ← n
        While m > 0 execute
            If m MOD b = 0 then
                v ← v + 1
            EndIf
            m ← m DIV b
        EndWhile
        Return v
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul determină și returnează câte cifre are numărul n.

B. Algoritmul returnează 1 dacă numărul n este o putere a lui b și 0 altfel.

C. Algoritmul determină și returnează numărul de cifre 0 din reprezentarea în baza b a numărului n.

D. Algoritmul returnează 1 dacă numărul n se termină cu cifra b și 0 altfel.

540. Se consideră algoritmul `abc(a, n, p)`, unde n este număr natural ($1 \leq n \leq 10000$), p este număr întreg ($-10000 \leq p \leq 10000$), iar a este un șir cu n numere naturale nenule ($a[1], a[2], ..., a[n]$). ✓ ?

```
Algorithm ABC(a, n, p)
    If n < 1 then
        Return −1
    Else
        If (1 ≤ p) AND (p ≤ n) then
            Return a[p]
        Else
            Return 0
        EndIf
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează 0 dacă și numai dacă p este negativ sau mai mare decât n.

B. Algoritmul returnează elementul de pe poziția p dacă p este strict mai mare decât 0 și mai mic sau egal decât lungimea șirului.

C. Algoritmul nu returnează niciodată 0 pentru valori ale parametrilor care respectă precondiţiile din enunț.

D. Algoritmul returnează elementul de pe poziția p dacă p este mai mare sau egal cu 0 și mai mic strict decât lungimea șirului.

541. Pentru a genera numerele cu n cifre formate doar din cifrele 0, 6, 7, se utilizează un algoritm care, pentru $n = 2$, generează în ordine crescătoare numerele 60, 66, 67, 70, 76, 77. Dacă $n = 4$ și se utilizează același algoritm, care este numărul generat imediat după numărul 6767? ✓ ?

A. 7667
B. 6760
C. 6776
D. Niciuna dintre celelalte variante

542. Pentru un număr natural nr ($1000 \leq nr \leq 1000000$), definim operația de decrementare în modul următor: dacă ultima cifră a lui nr nu este 0, scădem 1 din nr, altfel, împărțim nr la 10 și păstrăm doar partea întreagă. Care dintre următorii algoritmi returnează, la apelul `decrementare(nr, k)`, numărul obținut aplicând de k ori ($0 \leq k \leq 100$) operația de decrementare pe numărul nr? De exemplu, pentru $nr = 15243$ și $k = 10$, rezultatul este 151. ✓ ?

A.
```
Algorithm DECREMENTARE(nr, k)
   If k = 0 then
      Return nr
   Else
      If nr MOD 10 ≠ 0 then
         Return DECREMENTARE(nr
- 1, k - 1)
      Else
         Return DECREMENTARE(nr
DIV  10, k - 1)
      EndIf
   EndIf
EndAlgorithm
```

B.
```
Algorithm DECREMENTARE(nr, k)
   While k > 0 execute
      If nr MOD 10 = 0 then
         nr ← nr DIV 10
      Else
         nr ← nr − 1
      EndIf
      k ← k − 1
   EndWhile
   Return nr
EndAlgorithm
```

C.
```
Algorithm DECREMENTARE(nr, k)
   For i ← 1 to k execute
      If nr MOD 10 > 0 then
         nr ← nr DIV 10
      Else
         nr ← nr − 1
      EndIf
   EndFor
   Return nr
EndAlgorithm
```

D.
```
Algorithm DECREMENTARE(nr, k)
   If k = 0 then
      Return nr
   Else
      If k > nr MOD 10 then
         nr1 ← nr DIV 10
         Return DECREMENTARE(nr1,
k - nr  MOD  10 - 1)
      Else
         Return DECREMENTARE(nr
- k, 0)
      EndIf
   EndIf
EndAlgorithm
```

543. Algoritmul de mai jos are ca parametri de intrare un vector v cu n numere naturale $(v[1], v[2], ..., v[n])$ și numărul întreg n ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm FN(v, n)
   a ← 0
   For i ← 1 to n execute
      ok ← True
      b ← v[i]
      While (b ≠ 0) AND (ok = True) execute
```

```
            If b MOD 2 = 1 then
               ok ← False
            EndIf
            b ← b DIV 10
         EndWhile
         If ok = True then
            a ← a + 1
         EndIf
      EndFor
      Return a
   EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează numărul elementelor impare din vectorul v.

B. Algoritmul returnează numărul elementelor din vectorul v care sunt puteri ale lui 2.

C. Algoritmul returnează numărul elementelor din vectorul v care au în componența lor doar cifre pare.

D. Algoritmul returnează numărul elementelor din vectorul v care au în componența lor doar cifre impare.

544. Algoritmul `magic(s, n)` are ca parametri de intrare un șir s cu n caractere ($s[1], s[2], ..., s[n]$) și numărul întreg n ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm MAGIC(s, n)
   i ← n
   While 1 ≤ i execute
      If s[i] ≠ s[n − i + 1] then
         Return 0
      EndIf
      i ← i − 1
   EndWhile
   Return 1
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează 1 dacă s are un număr par de caractere.

B. Algoritmul returnează 1 dacă s este un palindrom.

C. Algoritmul are o eroare deoarece expresia $n - i + 1$ poate avea valori negative în cursul execuției.

D. Algoritmul returnează 1 dacă s conține doar caractere alfanumerice.

545. Se consideră următoarea secvență de cod în pseudocod: ✓ ?

```
Read a
For i ← 1 to a − 1 execute
   For j ← i + 2 to a execute
      If i + j > a − 1 then
         Write a, " ", i, " ", j
         Write new line
      EndIf
   EndFor
EndFor
```

Câte perechi de soluții se vor afișa în urma execuției secvenței de cod pentru $a = 9$?

A. 13

B. 15

C. 19

D. Niciuna dintre celelalte variante

546. Algoritmul `ceFace(n)` are ca parametru de intrare un număr natural n ($0 \leq n \leq$ 10000). ✓ ?

```
Algorithm ceFace(n)
   s ← 0
   While n > 0 execute
      c ← n MOD 10
      If c MOD 2 = 0 then
         s ← s + c
      EndIf
      n ← n DIV 10
   EndWhile
   Return s
EndAlgorithm
```

Ce va returna apelul `ceFace(9876)`?

A. 16

B. 48

C. 14

D. 63

547. Algoritmul `generare(n)` prelucrează un număr natural n ($0 < n < 100$). ✓ ?

```
Algorithm generare(n)
   nr ← 0
   For i ← 1 to 1801 execute
      used[i] ← False
   EndFor
   While not used[n] execute
      sum ← 0
      used[n] ← True
      While n ≠ 0 execute
         digit ← n MOD 10
         n ← n DIV 10
         sum ← sum + digit * digit *
digit
      EndWhile
      n ← sum
      nr ← nr + 1
   EndWhile
   Return nr
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $n = 10$, valoarea returnată este 2.

B. Dacă $n = 10$, valoarea returnată este 1.

C. Dacă $n = 3$, valoarea returnată este 4.

D. Cele două apeluri `generare(3)` și `generare(30)` vor returna aceeași valoare.

548. Se dă algoritmul `f(a, b)` care primește ca parametri două numere naturale a și b ($1 \leq a < b \leq 1000$): ✓ ?

```
Algorithm f(a, b)
   If a > 0 then
      Return b + f(a DIV 2, b * 2)
   EndIf
   Return b + f(a * 2, b DIV 2)
EndAlgorithm
```

Din păcate, algoritmul se autoapelează de o infinitate de ori. Precizați care este valoarea pe care o va avea parametrul b, atunci când parametrul a devine 0 pentru prima dată. Algoritmul se apelează cu instrucțiunea: $c \leftarrow f(20, 10)$

A. 100 B. 160 C. 320 D. 640

549. Precizați care dintre următoarele expresii are valoarea adevărat dacă și numai dacă numărul natural n este divizibil cu 3 și are ultima cifră 4 sau 6: ✓ ?

A. $(n$ **MOD** $3 = 0)$ **OR** $((n$ **MOD** $10 = 4)$ **AND** $(n$ **MOD** $10 = 6))$

B. $(n$ **MOD** $6 = 0)$ **AND** $((n$ **MOD** $10 = 4)$ **OR** $(n$ **MOD** $10 = 6))$

C. $((n$ **MOD** $9 = 0)$ **AND** $(n$ **MOD** $10 = 4))$ **OR** $((n$ **MOD** $3 = 0)$ **AND** $(n$ **MOD** $10 = 6))$

D. $(n$ **MOD** $3 = 0)$ **AND** $(((n$ **MOD** $2 = 0)$ **AND** $(n$ **MOD** $5 = 0))$ **OR** $((n$ **MOD** $2 = 0)$ **AND** $(n$ **MOD** $5 = 1)))$

550. Se consideră următoarea expresie logică $(X$ **OR** $Z)$ **AND** $(X$ **OR** $Y)$. Alegeți valorile pentru X, Y, Z astfel încât evaluarea expresiei să dea rezultatul **True**: ✓ ?

A. $X \leftarrow$ **False**; $Y \leftarrow$ **False**; $Z \leftarrow$ **True**;

B. $X \leftarrow$ **True**; $Y \leftarrow$ **False**; $Z \leftarrow$ **False**;

C. $X \leftarrow$ **False**; $Y \leftarrow$ **True**; $Z \leftarrow$ **False**;

D. $X \leftarrow$ **True**; $Y \leftarrow$ **True**; $Z \leftarrow$ **True**;

551. Se consideră toate șirurile de lungime $l \in \{1, 2, 3\}$ formate din litere din mulțimea $\{a, b, c, d, e\}$. Câte dintre aceste șiruri au elementele ordonate strict crescător (conform alfabetului) și un număr impar de consoane? (b, c și d sunt consoane) ✓ ?

A. 14 B. 13 C. 26 D. 81

552. Se dorește afișarea unui pătrat împreună cu diagonalele sale folosind doar caracterele * (asterisc) și . (punct) (pentru spațiul din interiorul pătratului cu excepția diagonalelor). Exemplul de mai jos ilustrează un pătrat având laturile de $n = 6$ asteriscuri. Pentru acesta a fost necesară utilizarea a 28 asteriscuri și 8 puncte. ✓ ?

```
* * * * * *
* *.. * *
* . * *. *
* . * *. *
* *.. * *
* * * * * *
```

Care din afirmațiile de mai jos sunt adevărate?

A. Pentru $n = 5$, este nevoie de exact 22 asteriscuri și 4 puncte.

B. Pentru $n = 7$, este nevoie de exact 34 asteriscuri și 15 puncte.

C. Pentru $n = 7$, este nevoie de exact 33 asteriscuri și 16 puncte.

D. Pentru $n = 18$, este nevoie de exact 100 asteriscuri și 224 puncte.

553. Se consideră algoritmul `ceFace(T, n, e)`, care primește ca și parametru un șir T cu n numere naturale ordonate crescător $(T[1], T[2], ..., T[n])$ și numerele naturale n și e $(1 \leq n, e \leq 10000)$. ✓ ?

```
Algorithm ceFace(T, n, e)
   If e MOD 2 = 0 then
      a ← 1
      b ← n
      While a ≤ b execute
         m ← (a + b) DIV 2
         If e < T[m] then
            b ← m − 1
         Else
            If e > T[m] then
               a ← m + 1
            Else
               Return m
            EndIf
         EndIf
      EndWhile
      Return 0
   Else
      c ← 1
      g ← 0
      While (c ≤ n) AND (g = 0)
execute
         If e = T[c] then
            g = c
         EndIf
         c ← c + 1
      EndWhile
      Return g
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul returnează 0 dacă numărul e nu se află în șirul T.

B. Dacă numărul e este impar și se află în șirul T, algoritmul returnează poziția din șirul T pe care se află e folosind algoritmul de căutare binară.

C. Dacă numărul e este impar și se află în șirul T, algoritmul returnează poziția din șirul T pe care se află e folosind algoritmul de căutare secvențială.

D. Algoritmul returnează poziția din șirul T pe care se află numărul e.

554. Se consideră algoritmul `calcul(x, n)`, unde parametrii de intrare sunt numerele naturale n și x, având proprietatea $1 \leq x \leq n < 10$.

```
Algorithm calcul(x, n)
   b ← 1
   For i ← 1 to n − x execute
      b ← b + i
   EndFor
   a ← b
   For i ← n − x + 1 to n execute
      a ← a + i
   EndFor
   Return a − b
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $n = 5$ și $x = 2$, atunci algoritmul returnează 20.

B. Dacă $n = 3$ și $x = 2$, atunci algoritmul returnează 5.

C. Algoritmul returnează cardinalitatea mulțimii $\{\overline{c_1c_2...c_x} : c_i \neq c_j \forall 1 \leq i, j \leq x, i \neq j, 1 \leq c_i \leq n\}$

D. Algoritmul returnează întotdeauna o valoare strict mai mare decât 0.

Problemele **555.** și **556.** se referă la algoritmul `s(a, b, c)`, unde a, b, c sunt numere naturale pozitive ($1 \leq a, b, c \leq 10000$).

```
Algorithm s(a, b, c)
   If (a = 1) OR (b = 1) OR (c = 1)
then
      Return 1
   Else
      If a > b then
         Return a * s(a − 1, b, c)
      Else
         If a < b then
            Return b * s(a, b − 1, c)
         Else
            Return c*s(a−1, b−1, c−
1)
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

555. Precizați care dintre următoarele afirmații sunt adevărate în cazul în care $a = b$ și $a < c$: ✓ ?

A. Algoritmul calculează și returnează $c!/(c-a)!$

B. Algoritmul calculează și returnează $c!/(c-a+1)!$

C. Algoritmul calculează și returnează $c!/(c-a-1)!$

D. Algoritmul calculează și returnează combinări de c luate câte $(a-1)$

556. Pentru $a = 3$, $b = 4$, $c = 7$, algoritmul returnează: ✓ ?

A. 224 B. 56 C. 336 D. 168

557. Se consideră algoritmul `numere(a, b, c, d, e)`, care primește ca parametri cinci numere întregi a, b, c, d și e ($1 \leq a, b \leq 10000, 2 \leq c \leq 16, 1 \leq d < c$). ✓ ?

```
Algorithm NUMERE(a, b, c, d, e)
   If a = 0 AND b = 0 then
      If e = 0 then
         Return True
      Else
         Return False
      EndIf
   EndIf
   If a MOD c = d then
      e ← e + 1
   EndIf
   If b MOD c = d then
      e ← e − 1
   EndIf
   Return NUMERE(a DIV c, b DIV c, c, d, e)
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate pentru apelul `numere(a, b, c, d, 0)`:

A. Algoritmul returnează **True** dacă reprezentările în baza c a numerelor a și b conțin cifra d de număr egal de ori, **False** în caz contrar

B. Algoritmul returnează **True** dacă cifra d apare în reprezentarea în baza c a numărului a și în reprezentarea în baza c a numărului b, **False** în caz contrar

C. Apelul `numere(a, b, c, d, 0)` returnează aceeași valoare ca și apelul `numere(b, a, c, d, 0)`

D. Algoritmul returnează **True** dacă cifra d apare pe aceleași poziții în reprezentarea în baza c a numerelor a și b, **False** în caz contrar

558. Fie s un șir de numere naturale unde elementele s_i sunt de forma: ✓ ?

$$s_i = \begin{cases} x, & \text{dacă } i = 1 \\ x+1, & \text{dacă } i = 2, \ (i = 1, 2, ...) \\ s_{(i-1)} @ s_{(i-2)} & \text{dacă } i > 2 \end{cases}$$

unde operatorul `@` concatenează cifrele operandului stâng cu cifrele operandului drept, în această ordine (cifre aferente reprezentării în baza 10), iar x este un număr natural ($1 \leq x \leq 99$). De exemplu, dacă $x = 3$, șirul s va conține valorile 3, 4, 43, 434, 43443, ...

Pentru un număr natural k ($1 \leq k \leq 30$) precizați numărul cifrelor acelui termen din șirul s care precede termenul format din *k1* cifre, unde *k1* este cel mai mic număr cu proprietatea că $k \leq k1 \leq 30$ și există un termen format din *k1* cifre.

A. dacă $x = 15$ și $k = 8$, numărul cifrelor termenului căutat este 6.

B. dacă $x = 2$ și $k = 6$, numărul cifrelor termenului căutat este 6.

C. dacă $x = 14$ și $k = 27$, numărul cifrelor termenului căutat este 26.

D. dacă $x = 5$ și $k = 12$, numărul cifrelor termenului căutat este 8.

559. Se consideră următorul algoritm recursiv `fibonacci(n)`, unde n este un număr natural ($1 \leq n \leq 100$). Să se determine de câte ori se afișează mesajul "Aici" în cazul unui apel `fibonacci(n)`. ✓ ?

```
Algorithm FIBONACCI(n)
   If n ≤ 1 then
      Write "Aici"
      Return 1
   Else
      Return FIBONACCI(n - 1) +
FIBONACCI(n - 2)
   EndIf
EndAlgorithm
```

A. De `fibonacci(n)` ori.

B. De `fibonacci(n-1)` ori.

C. De `fibonacci(n)`-1 ori.

D. De `fibonacci(n) - fibonacci(n-1)` ori.

560. Se consideră expresia: $E(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3 + a_4 * x^4$, unde a_0, a_1, a_2, a_3, a_4 și x sunt numere reale nenule. Numărul minim de înmulțiri necesare pentru a calcula valoarea expresiei $E(x)$ este: ✓ ?

A. 4 B. 5 C. 7 D. 3

Problemele **561.** şi **562.** se referă la algoritmul `f(x, n)` unde x, n sunt numere naturale și $x > 0$.

```
1: Algorithm F(x, n)
2:    If n = 0 then
3:       Return 1
4:    EndIf
5:    m ← n DIV 2
6:    p ← f(x, m)
7:    If n MOD 2 = 0 then
```

```
 8:       Return p * p
 9:    EndIf
10:    Return x * p * p
11: EndAlgorithm
```

561. Precizați care dintre următoarele afirmații sunt adevărate: ✓ ?

A. Algoritmul returnează x^n efectuând aproximativ n apeluri recursive.

B. Algoritmul returnează x^n efectuând aproximativ $\log_2 n$ apeluri recursive.

C. Algoritmul returnează x^n dacă și numai dacă n este putere a lui 2.

D. Algoritmul returnează x^n dacă și numai dacă n este par.

562. Considerăm linia 10 înlocuită cu: ✓ ?

```
10.  return x * f(x, n - 1)
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Algoritmul nu mai returnează x^n

B. Algoritmul returnează x^{n+1}

C. Algoritmul efectuează aproximativ n^2 apeluri recursive.

D. Algoritmul returnează x^n

563. Se consideră algoritmul `f2(a,b)` cu parametrii a și b numere naturale, și algoritmul `f(arr, i, n, p)` având ca parametri șirul arr cu n numere întregi ($arr[1], arr[2], ...$, $arr[n]$), și numerele întregi i și p: ✓ ?

```
Algorithm F2(a, b)
   If a > b then
      Return a
   Else
      Return b
   EndIf
EndAlgorithm
Algorithm F(arr, i, n, p)
   If i = n then
      Return 0
   EndIf
   n1 ← f(arr, i + 1, n, p)
   n2 ← 0
   If p + 1 ≠ i then
      n2 ← f(arr, i + 1, n, i) + arr[i]
   EndIf
   Return f2(n1, n2)
EndAlgorithm
```

Precizați care este rezultatul apelului `f(arr, 1, 9, -10)`, dacă șirul arr conține valorile $(10, 1, 5, 4, 7, 12, 1, 12, 6)$.

A. 24

B. 37

C. 39

D. 56

Problemele **564.** și **565.** se referă la algoritmul `f(n)`, care are ca parametru numărul natural nenul n și care returnează un număr natural.

```
Algorithm F(n)
   j ← n
   While j > 1 execute
      i ← 1
```

```
        While i ≤ n^4 execute
            i ← 4 * i
            Write "*"
        EndWhile
        If j DIV 2 > 1 then
            Write " "
        EndIf
        j ← j DIV 2
    EndWhile
    Return j
EndAlgorithm
```

564. În care dintre următoarele clase de complexitate se încadrează complexitatea timp a algoritmului? ✓ ?

A. $O(\log_2 n)$ B. $O(\log_2^2 n)$ C. $O(\log_4^2 n)$ D. $O(\log_2 \log_4 n)$

565. Care dintre afirmațiile de mai jos sunt adevărate? ✓ ?

A. Dacă $n = 10$, algoritmul afișează grupuri formate din câte 7 steluțe, grupurile fiind despărțite prin câte un spațiu.

B. Dacă $n = 20$, algoritmul afișează 4 grupuri de steluțe și 4 caractere spațiu.

C. Dacă $n = 25$, algoritmul afișează 48 de steluțe, iar după fiecare grup afișează un spațiu.

D. Dacă $n = 100$, algoritmul afișează 84 de steluțe și 5 caractere spațiu.

Subiect Concurs Mate-Info UBB 2021

566. Se consideră expresia următoare, în care a este un număr natural. ✓ ?

$((a < 4)$ **SAU** $(a < 5))$ **ȘI** $(a > 2)$

Pentru ce valori ale lui a va avea expresia valoarea **ADEVĂRAT**?

A. $a = 3$

B. $a = 4$

C. $a = 2$

D. Expresia nu va avea niciodată valoarea **ADEVĂRAT**

567. Subalgoritmul de mai jos are ca parametri de intrare un șir v cu n numere naturale nenule $(v[1], v[2], ..., v[n])$ și numărul întreg n $(1 \leq n \leq 10000)$. ✓ ?

```
Algorithm F(v, n)
    x ← 0
    For i ← 1 to n execute
        c ← v[i]
        While c MOD 3 = 0 execute
            x ← x + 1
            c ← c DIV 3
        EndWhile
    EndFor
    Return x
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează numărul numerelor divizibile cu 3 din șirul v

B. Subalgoritmul returnează cel mai mare număr k astfel încât $v[1] * v[2] * ... * v[n]$ este divizibil cu 3^k

C. Subalgoritmul returnează cel mai mare număr k astfel încât $v[1] + v[2] + ... + v[n]$ este divizibil cu 3^k

D. Subalgoritmul returnează suma numerelor divizibile cu 3 din șirul v

568. Se consideră expresia următoare, în care x este un număr natural pozitiv. ✓ ?

$(x$ **MOD** $2) + ((x + 1)$ **MOD** $2)$

Care din afirmațiile de mai jos sunt adevărate?

A. Expresia are valoarea 1 pentru orice număr natural pozitiv x.

B. Expresia are valoarea 1 dacă și numai dacă x este un număr par.

C. Expresia are valoarea 1 dacă și numai dacă x este un număr impar.

D. Există număr natural x pentru care expresia are o valoare strict mai mare decât 1.

569. Fie subalgoritmul `prelucrare(x, n)` definit mai jos, care primește ca și parametru un șir x cu n numere reale nenule $(x[1], x[2], ..., x[n])$ și numărul întreg n $(1 \leq n \leq 10000)$. Operatorul / reprezintă împărțirea reală (ex. 3/2=1,5). ✓ ?

```
Algorithm PRELUCRARE(x, n)
```

```
    p ← 1
    For k ← 1 to n − 1 execute
        p ← p + 1
        For i ← 1 to n − 1 execute
            If x[i] > x[i + 1] then
                x[i] ← x[i] ∗ x[i + 1]
                x[i + 1] ← x[i]/x[i + 1]
                x[i] ← x[i]/x[i + 1]
            EndIf
        EndFor
    EndFor
    n ← p
EndAlgorithm
```

Care dintre următoarele afirmații descriu modificarea aplicată șirului x în urma apelului subalgoritmului `prelucrare(x, n)`?

A. Elementele șirului x vor rămâne nemodificate

B. Elementele șirului x vor fi în ordine descrescătoare

C. Elementele șirului x vor fi în ordine crescătoare

D. Numărul n este decrementat cu o unitate

570. Se consideră subalgoritmul `calcul(a, n)`, care primește ca parametru un șir a cu n numere naturale $(a[1], a[2], ..., a[n])$ și numărul întreg n $(1 \leq n \leq 10000)$. ✓ ?

```
Algorithm CALCUL(a, n)
    If n = 0 then
        Return 0
    Else
        Return a[n] ∗ (a[n] MOD 2) + CALCUL(a, n − 1)
    EndIf
EndAlgorithm
```

Pentru ce valori a numărului n și a șirului a funcția `calcul(a,n)` va returna valoarea 10?

A. $n = 4$, $a = (2, 4, 7, 5)$

B. $n = 6$, $a = (3, 1, 2, 5, 8, 1)$

C. $n = 6$, $a = (2, 4, 5, 3, 8, 5)$

D. $n = 7$, $a = (1, 1, 2, 1, 1, 1, 3)$

571. Se consideră subalgoritmul `calcul(v, n)`, care primește ca parametru un șir v cu n numere naturale $(v[1], v[2], ..., v[n])$ și numărul întreg n $(1 \leq n \leq 10000)$. ✓ ?

```
Algorithm CALCUL(v, n)
    m ← 0
    x ← 0
    s ← 0
    For i ← 1 to n execute
        s ← s + v[i]
        m ← m + (s MOD 2 + x) MOD 2
        x ← s MOD 2
    EndFor
```

```
    Return m
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul calculează și returnează suma numerelor impare din șirul v

B. Subalgoritmul calculează și returnează suma numerelor pare din șirul v

C. Subalgoritmul calculează și returnează numărul de numere impare din șirul v

D. Subalgoritmul calculează și returnează numărul de numere pare din șirul v

572. Se consideră subalgoritmul `magic(x)`, unde x este un număr natural ($1 \leq x \leq$ ✓ ? 32000).

```
Algorithm MAGIC(x)
    st ← 1
    dr ← x
    While st ≤ dr execute
        mj ← (st + dr) DIV 2
        If mj * mj = x then
            Return adevărat
        EndIf
        If mj * mj < x then
            st ← mj + 1
        Else
            dr ← mj − 1
        EndIf
    EndWhile
    Return fals
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul verifică dacă există un pătrat perfect mai mic decât x.

B. Subalgoritmul numără divizorii primi ai numărului x.

C. Subalgoritmul verifică dacă numărul x este prim.

D. Subalgoritmul verifică dacă numărul x este pătrat perfect.

573. Se consideră subalgoritmul `ceFace(n)`, unde n este un număr natural ($1 \leq n \leq$ ✓ ? 10000).

```
Algorithm CEFACE(n)
    a ← n
    b ← 0
    While a ≠ 0 execute
        b ← b * 10 + a MOD 10
        a ← a DIV 10
    EndWhile
    If n = b then
        Return adevărat
    Else
        Return fals
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul verifică dacă numărul n este prim.

B. Subalgoritmul verifică dacă numărul n este palindrom.

C. Subalgoritmul returnează întotdeauna **adevărat**.

D. Subalgoritmul verifică dacă numărul n este divizibil cu 10.

574. Se consideră subalgoritmul `calculeaza(a,b)`, unde a și b sunt numere naturale $(1 \leq a, b \leq 10000)$. ✓ ?

```
Algorithm CALCULEAZA(a, b)
    x ← 1
    For i ← 1 to b execute
        x ← (x MOD 10) * a
    EndFor
    Return x
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă $a = 2021$ și $b = 2021$, valoarea returnată de subalgoritm este 2021.

B. Pentru toate apelurile subalgoritmului cu $a = 2021$ și $1 \leq b \leq 10000$, valoarea returnată este 2021.

C. Dacă $a = 7777$ și $b = 2021$, valoarea returnată este 7777.

D. Pentru toate apelurile subalgoritmului cu $1 \leq a \leq 10000$ și $b = 2021$, valoarea returnată este valoarea lui a.

575. Câte elemente se găsesc pe cele două diagonale ale unei matrice pătratice cu n linii și n coloane $(10 \leq n \leq 1000)$? Se numără elementele de pe poziții distincte. ✓ ?

A. $2 * n$

B. $n * n$

C. $2 * n - 1$

D. $2 * n - (n \textbf{ MOD } 2)$

576. Care dintre expresiile logice următoare au valoarea **ADEVĂRAT** pentru $a = 1$ și $b = 0$? ✓ ?

A. **NU** $(((a > 0)$ **ȘI** $(b < 1))$ **SAU** $(a > 1))$

B. $((b > 0)$ **ȘI** $(b < 1))$ **SAU** $((a > 0)$ **ȘI** $(a < 2))$

C. $(\textbf{NU } (a > b))$ **SAU** $(\textbf{NU } (b > 0))$

D. $(a > 0)$ **SAU** $((b > 0)$ **ȘI** $(b < 0))$ **SAU** $(a < 1)$

577. Subalgoritmii $\texttt{calcul}_i\texttt{(e, n)}$, $1 \leq i \leq 4$, primesc ca parametri o matrice e de n linii și n coloane $(e[1][1], ..., e[1][n], e[2][1], ..., e[n][n])$ și un număr natural n $(1 \leq n \leq 1000)$. Alegeți variantele de răspuns care conțin definiția subalgoritmului $\texttt{calcul}_i\texttt{(e, n)}$, care are rezultat diferit față de toate celelalte trei variante, adică $\texttt{calcul}_i\texttt{(e, n)} \neq \texttt{calcul}_j\texttt{(e, n)}$ $\forall$ e, n, j, $1 \leq j \leq 4$, $i \neq j$ (e și n conform specificației anterioare). ✓ ?

A.

```
Algorithm CALCUL1(e, n)
    s ← 0
    For i ← 1 to n execute
        s ← s + e[1][i]
    EndFor
    Return s
EndAlgorithm
```

B.

```
Algorithm CALCUL2(e, n)
    s ← 0
    For i ← 1 to n execute
        For j ← 1 to n execute
            If i = j then
                s ← s + e[i][j]
            EndIf
        EndFor
    EndFor
    Return s
EndAlgorithm
```

C.

```
Algorithm CALCUL3(e, n)
    s ← 0
    i ← 1
    While i ≤ n execute
        s ← s + e[i][i]
        i ← i + 1
    EndWhile
    Return s
EndAlgorithm
```

D.

```
Algorithm CALCUL4(e, n)
    s ← 0
    For i ← 1 to n execute
        For j ← i + 1 to n execute
            If i = j then
                s ← s + e[i][j]
            EndIf
        EndFor
    EndFor
    Return s
EndAlgorithm
```

578. Se consideră subalgoritmul `ceFace(a,b)`, unde a și b sunt numere naturale ($1 \leq a < b \leq 10000$). ✓ ?

```
Algorithm CEFACE(a, b)
    m ← a
    While b MOD m > 0 execute
        m ← m + 1
    EndWhile
    Return m
EndAlgorithm
```

Ce va returna apelul `ceFace(47, 100)`?

A. 48 B. 50 C. 3 D. 100

579. Se consideră subalgoritmul `afis(n)`, unde n este un număr natural ($0 \leq n \leq 10000$). ✓ ?

```
Algorithm AFIS(n)
    Scrie n
    If n > 0 then
        AFIS(n - 1)
        Scrie n
    EndIf
EndAlgorithm
```

Ce se va afișa la apelul `afis(4)`?

A. 43210123 B. 123401234 C. 12340043 D. 43201234

580. Care dintre următoarele baze de numerație x satisfac condiția $232_{(x)} \leq 67_{(10)}$? ✓ ?

A. $x = 5$ B. $x = 3$ C. $x = 4$ D. $x = 6$

581. Subalgoritmul `mutaZero(a, n)` primește ca și parametru un șir a de numere întregi, ✓ ? $(a[1], a[2], ..., a[n])$ și numărul întreg n $(1 \leq n \leq 10000)$. Subalgoritmul mută valorile de zero la finalul șirului, păstrând ordinea relativă a elementelor diferite de zero. De exemplu, dacă a este [4, 0, 2, 5, 1, 0, 7, 11, 0, 3], după apelarea subalgoritmului, elementele lui a sunt [4, 2, 5, 1, 7, 11, 3, 0, 0, 0]. Care din implementările următoare ale subalgoritmului `mutaZero(a, n)` sunt corecte?

A.

```
Algorithm MUTAZERO(a, n)
    s ← ADEVĂRAT
    While s = ADEVĂRAT execute
        s ← FALS
        For i ← 1 to n − 1 execute
            If a[i] = 0 then
                tmp ← a[i]
                a[i] ← a[i + 1]
                a[i + 1] ← tmp
                s ← ADEVĂRAT
            EndIf
        EndFor
    EndWhile
EndAlgorithm
```

B.

```
Algorithm MUTAZERO(a, n)
    c ← 0
    For i ← 0 to n execute
        If a[i] = 0 then
            c ← c + 1
        EndIf
    EndFor
    i ← n
    While c > 0 execute
        a[i] ← 0
        i ← i − 1
        c ← c − 1
    EndWhile
EndAlgorithm
```

C.

```
Algorithm MUTAZERO(a, n)
    d ← 0
    i ← 1
    While i + d ≤ n execute
        While (i + d ≤ n) ȘI (a[i + d] =
0) execute
            d ← d + 1
        EndWhile
        If i + d ≤ n then
            a[i] ← a[i + d]
            i ← i + 1
        EndIf
    EndWhile
    While i ≤ n execute
        a[i] ← 0
        i ← i + 1
    EndWhile
EndAlgorithm
```

D.

```
Algorithm MUTAZERO(a, n)
    i ← 1
    f ← n
    While i < f execute
        While (i < f) ȘI (a[i] ≠ 0)
execute
            i ← i + 1
        EndWhile
        While (i < f) ȘI (a[f] = 0)
execute
            f ← f − 1
        EndWhile
        If i < f then
            tmp ← a[i]
            a[i] ← a[f]
            a[f] ← tmp
        EndIf
    EndWhile
EndAlgorithm
```

582. Fie șirul $X = 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, ...$, în care fiecare ✓ ? număr n apare de n ori pe poziții consecutive. Considerând că primul element din șir este pe poziția 1, pe ce poziții din șir apare valoarea 21?

A. Pe pozițiile din intervalul [210,230]

B. Pe pozițiile din intervalul [211,231]

C. Pe pozițiile din intervalul [212,232]

D. Pe pozițiile din intervalul [209,229]

583. Se consideră subalgoritmul `f(a, b)`, unde a și b sunt numere întregi ($-10000 \leq a, b \leq 10000$). ✓ ?

```
Algorithm F(a, b)
   Scrie "FMI"
   If (a = 0) SAU (b = 0) then
      Return 1
   EndIf
   If a > b then
      Return f(a − b ∗ b, a ∗ (a − b) − b ∗ (a − b))
   EndIf
   If a ≤ b then
      Return f(b − a ∗ a, a ∗ (a − b) − b ∗ (a − b))
   EndIf
EndAlgorithm
```

Precizați de câte ori se scrie textul *FMI* la executarea secvenței de cod: `f(f(3, 2), f(2, 3))`

A. De 8 ori B. De 6 ori C. De 3 ori D. De o infinitate de ori

584. Se consideră subalgoritmul recursiv `ceFace(n, i)`, unde n este un număr natural ($2 \leq n \leq 1000$). ✓ ?

```
Algorithm ceFace(n, i)
   If i ∗ i > n then
      Return 0
   EndIf
   If i ∗ i = n then
      Return i
   EndIf
   If n MOD i = 0 then
      Return i + n DIV i + ceFace(n, i + 1)
   Else
      Return ceFace(n, i + 1)
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate pentru apelul `ceFace(n, 2)`:

A. Subalgoritmul calculează și returnează dublul sumei tuturor divizorilor proprii ai numărului n.

B. Subalgoritmul calculează și returnează suma divizorilor proprii ai numărului n.

C. Subalgoritmul calculează și returnează suma divizorilor proprii și improprii ai numărului n.

D. Subalgoritmul verifică dacă n este pătrat perfect. În caz afirmativ, returnează rădăcina lui pătrată. Altfel, returnează 0.

585. Se consideră subalgoritmul `ceFace(T, n, e)`, care primește ca și parametru un șir T cu n numere naturale ordonate crescător $(T[1], T[2], ..., T[n])$ și numerele naturale n și e $(1 \leq n, e \leq 10000)$. ✓ ?

```
Algorithm ceFace(T, n, e)
   If e MOD 2 = 0 then
      a ← 1
      b ← n
      While a ≤ b execute
         m ← (a + b) DIV 2
         If e < T[m] then
            b ← m − 1
         Else If e > T[m] then
            a ← m + 1
         Else
            Return adevărat
         EndIf
      EndWhile
      Return fals
   Else
      c ← 1
      While c ≤ n execute
         If e = T[c] then
            Return adevărat
         EndIf
         c ← c + 1
      EndWhile
      Return fals
   EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul nu verifică dacă numărul e se află pe o poziție pară în șirul T.

B. Subalgoritmul verifică dacă numărul e se află în șirul T, iar dacă numărul e este impar, algoritmul de căutare folosit este căutarea binară.

C. Subalgoritmul verifică dacă numărul e se află în șirul T, iar dacă numărul e este par, algoritmul de căutare folosit este căutarea binară.

D. Subalgoritmul verifică dacă numărul e se află în șirul T doar dacă numărul e este impar.

586. Se dorește afișarea triunghiurilor echilaterale folosind doar caracterele * (asterisc) și . (punct). Exemplul de mai jos ilustrează un triunghi având latura de $n = 5$ asteriscuri. Pentru acesta a fost necesară utilizarea a 12 asteriscuri și 23 de puncte. ✓ ?

```
.... *
... * . *
.. * ... *
. * .... *
* . * . * .*
```

Care din afirmațiile de mai jos sunt adevărate?

A. Pentru $n = 2$, este nevoie de exact 3 asteriscuri și 4 puncte.

B. Pentru $n = 7$, este nevoie de exact 18 asteriscuri și 52 puncte.

C. Pentru $n = 7$, este nevoie de exact 18 asteriscuri și 48 puncte.

D. Pentru $n = 15$, este nevoie de exact 42 asteriscuri și 288 puncte.

587. Spunem că un șir având n caractere este antipalindrom dacă toate perechile de caractere egal depărtate de extremități sunt distincte două câte două (cu excepția ✓ ?

celui din mijloc dacă n este impar). De exemplu, *asdfg* și *xlxe* sunt antipalindroame, dar *asdsg* nu este.

Fie subalgoritmul `antipalindrom(s, stânga, dreapta)` care primește ca și parametru un șir s cu n ($1 \leq n \leq 10000$) caractere ($s[1], s[2], ..., s[n]$), și numerele naturale *stânga* și *dreapta*.

Care din următoarele implementări vor returna **adevărat** pentru apelul `antipalindrom(s, 1, n)` dacă și numai dacă șirul s este antipalindrom?

A.

```
Algorithm ANTIPALINDROM(s, stânga, dreapta)
   If stânga = dreapta then
      Return adevărat
   Else
      prim ← s[stânga]
      ultim ← s[dreapta]
      If prim = ultim then
         Return fals
      Else
         Return ANTIPALINDROM(s, stânga + 1, dreapta - 1)
      EndIf
   EndIf
EndAlgorithm
```

B.

```
Algorithm ANTIPALINDROM(s, stânga, dreapta)
   If stânga ≥ dreapta then
      Return adevărat
   EndIf
   prim ← s[stânga]
   ultim ← s[dreapta]
   If prim = ultim then
      Return fals
   Else
      Return ANTIPALINDROM(s, stânga + 1, dreapta - 1)
   EndIf
EndAlgorithm
```

C.

```
Algorithm ANTIPALINDROM(s, stânga, dreapta)
   If stânga > dreapta then
      Return adevărat
   Else
      prim ← s[stânga]
      ultim ← s[dreapta]
      If prim ≠ ultim then
         Return fals
      Else
         Return ANTIPALINDROM(s, stânga + 1, dreapta - 1)
      EndIf
   EndIf
EndAlgorithm
```

D.

```
Algorithm ANTIPALINDROM(s, stânga, dreapta)
   If stânga > dreapta then
      Return adevărat
   EndIf
   prim ← s[stânga]
   ultim ← s[dreapta]
   If prim ≠ ultim then
      Return adevărat
   EndIf
   Return ANTIPALINDROM(s, stânga + 1, dreapta - 1)
EndAlgorithm
```

588. Fie subalgoritmul `ordo(n, a)` care primește ca și parametru un număr natural n ✓ ? $(1 \leq n \leq 10000)$ și un șir a având $2n$ elemente numere naturale $(a[1], a[2], ..., a[2n])$.

Considerând că numărul de elemente pare ale șirului a este egal cu numărul de elemente impare, care din următorii subalgoritmi rearanjează elementele șirului a astfel încât elementele impare să aibă indici impari, iar elementele pare să aibă indici pari?

A.

```
Algorithm ORDO(n, a)
   For i ← 1, 2 * n - 1 execute
      If a[i]  MOD  2 ≠ i  MOD  2 then
         For j ← i + 1, 2 * n execute
            If a[j]  MOD  2 ≠ j  MOD  2 then
               a[i] ← a[i] + a[j]
               a[j] ← a[i] - a[j]
               a[i] ← a[i] - a[j]
            EndIf
         EndFor
      EndIf
   EndFor
EndAlgorithm
```

B.

```
Algorithm ORDO(n, a)
   For i ← 1, 2 * n - 1 execute
      If a[i]  MOD  2 ≠ i  MOD  2 then
         For j ← i + 1, 2 * n execute
            If (a[i]  MOD  2 ≠ i  MOD  2)  ȘI  (a[j]  MOD  2 ≠ j  MOD  2)
then
               a[i] ← a[i] + a[j]
               a[j] ← a[i] - a[j]
               a[i] ← a[j] - a[i]
            EndIf
         EndFor
      EndIf
   EndFor
EndAlgorithm
```

C.

```
Algorithm ORDO(n, a)
   For i ← 1, 2 * n - 1 execute
      If a[i]  MOD  2 ≠ i  MOD  2 then
```

```
            For j ← i + 1, 2 * n execute
               If (a[i]  MOD  2 ≠ i  MOD  2)  ȘI
(a[j]  MOD  2 ≠ j  MOD  2)  ȘI
(a[i]  MOD  2 ≠ a[j]  MOD  2) then
                  a[i] ← a[i] + a[j]
                  a[j] ← a[i] - a[j]
                  a[i] ← a[i] - a[j]
               EndIf
            EndFor
         EndIf
      EndFor
   EndAlgorithm
```

D.

```
Algorithm ORDO(n, a)
   For i ← 1, 2 * n - 1 execute
      For j ← i + 1, 2 * n execute
         If (a[j]  MOD  2 = 0)  ȘI
((a[j]  MOD  2 ≠ 0)  SAU  (a[j]  MOD  2 ≠ 0))  ȘI
(a[j]  MOD  2 = 0) then
            a[i] ← a[i] + a[j]
            a[j] ← a[i] - a[j]
            a[i] ← a[i] - a[j]
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

589. Dorim să partiționăm un șir de n ($1 \leq n \leq 1000$) valori în k ($1 \leq k \leq n$) subsecvențe adiacente de lungimi egale (fiecare element al șirului aparține exact unei subsecvențe). Dacă n nu este divizibil cu k, acceptăm ca diferența de lungime între oricare două subsecvențe să fie cel mult 1. Se dau mai jos patru variante de a genera indicii primelor elemente ale tuturor subsecvențelor j ($1 \leq j \leq k$). Numerotând elementele șirului de la 1, care dintre aceste variante satisfac cerința de mai sus? ✓ ?

A. $((j * n - 1)$ **DIV** $k) - 1$

B. $((j - 1) * n)$ **DIV** $k + 1$

C. $(j - 1) * (n$ **DIV** $k)$

D. $((j - 1) * n + k)$ **DIV** k

590. Fie $b_n b_{n-1} ... b_0$ reprezentarea binară a numărului natural B, $2021 \leq B \leq 2021^{2021}$. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Dacă valoarea expresiei $b_0 - b_1 + b_2 - b_3 + ... + (-1)^n b_n$ este zero, atunci B este divizibil cu 3

B. Dacă valoarea expresiei $b_0 - b_1 + b_2 - b_3 + ... + (-1)^n b_n$ este divizibilă cu 3, atunci B este divizibil cu 3

C. B este divizibil cu 3 dacă suma cifrelor binare este divizibilă cu 3, dar nu cu 9

D. Dacă B este divizibil cu 3, atunci valoarea expresiei $b_0 - b_1 + b_2 - b_3 + ... + (-1)^n b_n$ este divizibilă cu 3

591. Considerăm subalgoritmul `prefix(n)`, care dat fiind numărul natural n ($9 < n < 10^{20} - 1$), caută cel mai lung prefix al său care se regăsește și în interiorul numărului (exceptând prima și ultima cifră a sa). Subalgoritmul returnează lungimea acestui prefix. ✓ ?

Exemplu:

i. pentru $n = 12133121$, prefixul este 12 și are lungimea 2.

ii. pentru $n = 34534536$, prefixul este 3453 și are lungimea 4.

iii. pentru $n = 1223$, un astfel de prefix nu există (considerăm că are lungime 0).

Știind că indexarea șirurilor începe de la 1, care din următoarele variante reprezintă implementări corecte ale subalgoritmului `prefix(n)`?

A.

```
Algorithm PREFIX(n)
   nr ← n
   c ← 0
   p ← 1
   While nr > 0 execute
      c ← c + 1
      nr ← nr  DIV  10
      p ← p * 10
   EndWhile
   f1 ← 100
   f2 ← p  DIV  100
   k ← 1
   ok ← 0
   While ok = 0 execute
      n1 ← n  DIV  f1
      f3 ← 10
      For i ← 1, k execute
         n2 ← (n  DIV  f3)  MOD
f2
         If n1 = n2 then
            ok ← 1
            Return c - k - 1
         EndIf
         f3 ← f3 * 10
      EndFor
      f1 ← f1 * 10
      f2 ← f2  DIV  10
      k ← k + 1
   EndWhile
   Return -1
EndAlgorithm
```

B.

```
Algorithm PREFIX(n)
   c ← [0] * 20   ▷ 20 de poziții cu
valoarea 0
   nr ← n
   p ← 0
   While nr > 0 execute
      c[p + 1] ← nr  MOD  10
      nr ← nr  DIV  10
      p ← p + 1
   EndWhile
   For i ← 1, p - 2 execute
      For j ← p - 1, i + 1, -1
execute
         ok ← 1
         For k ← 0, i - 1 execute
            If c[p - 1 - k] ≠ c[j
- k] then
               ok ← 0
            EndIf
         EndFor
         If ok = 1 then
            Return i
         EndIf
      EndFor
   EndFor
   Return -1
EndAlgorithm
```

C.

```
Algorithm PREFIX(n)
   c ← [0] * 20   ▷ 20 de poziții cu
valoarea 0
   nr ← n
   p ← 0
   While nr > 0 execute
      c[p + 1] ← nr  MOD  10
      nr ← nr  DIV  10
      p ← p + 1
   EndWhile
   For i ← p - 2, 1, -1 execute
      For j ← p - 1, i + 1, -1
execute
         ok ← 1
         For k ← 0, i - 1 execute
            If c[p - k] ≠ c[j -
k] then
               ok ← 0
            EndIf
         EndFor
         If ok = 1 then
            Return j
         EndIf
      EndFor
   EndFor
   Return 0
EndAlgorithm
```

D.

```
Algorithm PREFIX(n)
   nr ← n
   c ← 0
   p ← 1
   While nr > 0 execute
      c ← c + 1
      nr ← nr  DIV  10
      p ← p * 10
   EndWhile
   f1 ← p  DIV  10
   f2 ← 10
   k ← c - 2
   ok ← -1
   For t ← 1, c - 2 execute
      n1 ← n  DIV  f1
      f3 ← 10
      For i ← 1, k execute
         n2 ← (n  DIV  f3)  MOD
f2
         If n1 = n2 then
            If ok < c - k - 1
then
               ok ← c - k - 1
            EndIf
         EndIf
         f3 ← f3 * 10
      EndFor
      f1 ← f1  DIV  10
      f2 ← f2 * 10
      k ← k - 1
   EndFor
   If ok < 0 then
      Return 0
   EndIf
   Return ok
EndAlgorithm
```

592. Se consideră tabelul de mai jos, având 16 celule (4 linii notate cu 1, 2, 3, 4, și 4 coloane notate cu A, B, C, D). Unele celule conțin valori constante (de ex. celula B3), altele, care încep cu "=" conțin rezultatul unei expresii aritmetice cu 2 termeni. Fiecare termen este fie o valoare constantă, fie, dacă termenul începe cu simbolul $, o referință către valoarea dintr-o altă celulă. De exemplu, în celula A4 avem rezultatul operației aritmetice de scădere din valoarea constantă 5 a valorii din celula A2. Pentru o anumită celulă *i*, notăm cu X(i) numărul minim de celule (inclusiv cele care conțin valori constante) ale căror valori trebuie cunoscute înainte de a calcula valoarea din celula *i*. Similar, notăm cu Y(i) numărul maxim de celule (inclusiv cele care conțin valori constante, dar excluzând celula *i*) ale căror valori pot fi calculate fără a cunoaște valoarea din celula *i*. Care dintre următoarele afirmații sunt adevărate despre valorile lui X(A2) și Y(A2)? ✓ ?

	A	B	C	D
1	= $B4 - $C1	=$B3 + $D3	3	= $A4 * $C3
2	= $B1 + $B2	= $D3 + 11	= $D3 + $D2	2
3	= $B1 - $D3	11	= $D4 * $D4	= $D2 + 2
4	= 5 - $A2	= $C1 * $C1	= $A3 / 2	=15 / 3

A. X(A2) = 2 și Y(A2) = 1

B. X(A2) = 5 și Y(A2) = 13

C. X(A2) = 6 și Y(A2) = 4

D. X(A2) = X(C4) și Y(A2) = Y(B2) + 1

593. Subalgoritmul `simplifică(nr, num)` obține fracția ireductibilă `aux1/aux2` cu proprietatea că `aux1/aux2` = `nr/num` (`aux1`, `aux2`, `nr`, `num` numere naturale, `num`, `aux2` $\neq$ 0).

```
Algorithm SIMPLIFICĂ(nr, num)
   d ← funcție(nr, num)
   aux1 ← nr  DIV  d
   aux2 ← num  DIV  d
EndAlgorithm
```

Care dintre variantele următoare ale subalgoritmului `funcție(a, b)` sunt corecte?

A.
```
Algorithm FUNCȚIE(a, b)
   d ← 1
   While adevărat execute
      If (a  MOD  d = 0)  ȘI
(b  MOD  d = 0) then
         Return d
      EndIf
      d ← d + 1
   EndWhile
EndAlgorithm
```

B.
```
Algorithm FUNCȚIE(a, b)
   While b ≠ 0 execute
      c ← a  MOD  b
      a ← b
      b ← c
   EndWhile
   Return a
EndAlgorithm
```

C.
```
Algorithm FUNCȚIE(a, b)
   While a ≠ b execute
      If a > b then
         a ← a - b
      Else
         b ← b - a
      EndIf
   EndWhile
   Return a
EndAlgorithm
```

D.
```
Algorithm FUNCȚIE(a, b)
   d ← a
   While ((a  MOD  d ≠ 0)  SAU
(b  MOD  d ≠ 0)) execute
      d ← d - 1
   EndWhile
   Return d
EndAlgorithm
```

594. Se consideră următorul subalgoritm recursiv `fibonacci(n)`, unde n este un număr natural ($1 \leq n \leq 100$). Să se determine de câte ori se afișează mesajul "Aici" în cazul unui apel `fibonacci(n)` (considerând împreună apelul inițial și toate apelurile recursive generate).

```
Algorithm FIBONACCI(n)
   If n ≤ 1 then
      Return 1
```

```
    Else
       Scrie "Aici"
       Return fibonacci(n - 1) + fibonacci(n - 2)
    EndIf
 EndAlgorithm
```

A. De `fibonacci(n)` ori.
B. De `fibonacci(n-1)` ori.
C. De `fibonacci(n)-1` ori.
D. De `fibonacci(n) - fibonacci(n-1)` ori.

595. Se consideră expresia: $E(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3 + a_5 * x^5$, unde a_0, a_1, a_2, a_3, a_5 și x sunt numere reale nenule. Numărul minim de înmulțiri necesare pentru a calcula valoarea expresiei $E(x)$ este: ✓ ?

A. 4 B. 5 C. 7 D. 11

Testul 1

596. Se consideră algoritmul `ceFace(a, len)`, unde *len* este un număr natural ($1 \leq len \leq 10^3$) și a este un vector cu *len* elemente numere întregi ($a[1], a[2], ..., a[len]$), unde $-10^5 \leq a[i] \leq 10^5$, pentru $i = 1, 2, ..., len$ și N care reprezintă lungimea vectorului a: ✓ ?

```
Algorithm ceFace(a, len, N)
    i ← 1
    While len − − execute
        If i ≥ N then
            i ← 1
            continue
        EndIf
        If a[i − 1] > a[i] then
            a[i − 1] ← a[i − 1] ⊕ a[i]
            a[i] ← a[i] ⊕ a[i − 1]
            a[i − 1] ← a[i − 1] ⊕ a[i]
            len ← N
        EndIf
        i ← i + 1
    EndWhile
EndAlgorithm
```

Care dintre următoarele variante de răspuns sunt adevărate?

A. Pentru $len = 5, a = [5, 2, 8, 1, 9], N = 5$ vectorul rezultat va fi ordonat crescător

B. Dacă vectorul este deja sortat crescător, algoritmul va executa exact *len* iterații

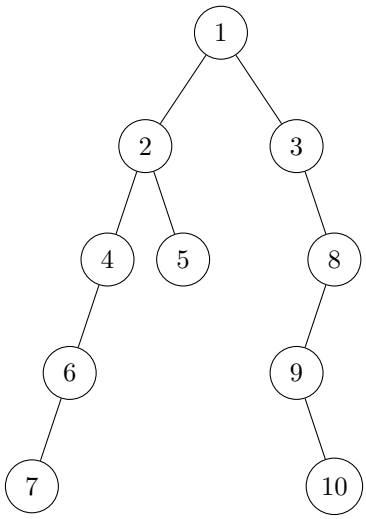
C. Algoritmul sortează vectorul descrescător

D. Algoritmul va reseta valoarea lui *len* la fiecare pas

597. Se consideră expresia $E = ABCD_{(16)} + 132_{(8)} + 24_{(6)} + A28B_{(14)}$, unde x_b reprezintă numărul x scris în baza b. Care este valoarea expresiei E în baza 10? ✓ ?

A. 51022 B. 72042 C. 36021 D. 27955

598. Se consideră următorul arbore binar: ✓ ?



Care dintre următoarele șiruri de noduri corespund traversării arborelui în preordine?

A. 7, 6, 4, 2, 5, 1, 3, 9, 10, 8

B. 1, 2, 4, 6, 7, 5, 3, 8, 9, 10

C. 7, 6, 4, 5, 2, 10, 9, 8, 3, 1

D. 7, 6, 4, 5, 2, 10, 9, 1, 3, 8

599. Se consideră algoritmul `ceFace(n, p, k)`, unde n, p și k sunt numere naturale ✓ ? $(1 \leq k \leq n)$:

```
Algorithm ceFace(n, p, k)
    For i ← 1, n execute
        For j ← 1, n execute
            a[i][j] ← a[i][j] + a[i][j − 1] + a[i − 1][j] − a[i − 1][j − 1]
        EndFor
    EndFor
    ans ← 0, cnt ← 0
    For i ← k, n execute
        For j ← k, n execute
            sum ← a[i][j] + a[i − k][j − k] − a[i][j − k] − a[i − k][j]
            If sum > ans then
                ans ← sum
                cnt ← 1
            Else If sum = ans then
                cnt ← cnt + 1
            EndIf
        EndFor
    EndFor
    Return (ans, cnt)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul găsește numărul de sume maximale de lungime k

B. Pentru $n = 3, k = 2, p = 4$ cu punctele $(1, 1), (1, 2), (2, 1), (2, 2)$, algoritmul returnează $(4, 1)$

C. Complexitatea algoritmului este $O(n^2 \cdot k)$

D. Algoritmul găsește numărul maxim de puncte într-un pătrat $k * k$ și numărul de astfel de pătrate

600. Se consideră algoritmul `ceFace(poz, s, n)`, unde *poz*, s și n sunt numere naturale, ✓ ? iar a este un vector sortat crescător cu n elemente distincte:

```
Algorithm ceFace(poz, s, n)
    If a[n] < s then
        Return n
    EndIf
    If a[poz] ≥ s then
        Return poz − 1
    EndIf
    lb ← poz, rb ← n, p ← poz − 1
    While lb ≤ rb execute
        mb ← (lb + rb)/2
        If a[mb] < s then
            p ← mb
            lb ← mb + 1
        Else
            rb ← mb − 1
        EndIf
    EndWhile
    Return p
```

```
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul este folosit pentru a găsi al treilea element care ar putea forma un triunghi valid cu două laturi date

B. Pentru vectorul $a = [2, 5, 7, 9]$, funcția va fi apelată de exact 12 ori pentru a determina toate triunghiurile posibile

C. Dacă pentru două laturi $a[i]$ și $a[j]$ avem $ceFace(j+1, a[i]+a[j]) = n$, atunci nu există niciun triunghi valid care să includă laturi

D. Pentru vectorul $a = [3, 4, 5, 6, 7]$, funcția va returna 4 când este apelată cu parametrii $(2, 9)$

601. Se consideră algoritmul `factor(n)`, unde n este un număr natural pozitiv: ✓ ?

```
Algorithm FACTOR(n)
    a ← 0
    For i ← 5, n; i ← i * 5 execute
        a ← a + [n/i]
    EndFor
    Return a
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru orice $n \geq 625$, algoritmul va returna o valoare mai mare decât suma cifrelor lui n

B. Pentru $n = 24$ și $n = 25$, algoritmul va returna aceeași valoare

C. Pentru $n = 125$, valoarea returnată va fi 31

D. Rezultatul algoritmului reprezintă numărul maxim de factori primi ai lui n mai mari decât 5

602. Se consideră algoritmul `ceFace(k)` unde k este un număr natural nenul. ✓ ?

```
Algorithm CEFACE(k)
    If k = 4 then
        write(sol)
        Return
    EndIf
    For i ← 0, 9 execute
        If k = 1 AND i ≠ 0 AND
                i MOD 2 = 0 then
            sol[k − 1] ← i
            ceFace(k + 1)
        Else If k > 1 and
                i > sol[k − 2] then
            sol[k − 1] ← i
            ceFace(k + 1)
        EndIf
    EndFor
EndAlgorithm
```

Care dintre următoarele variante de răspuns sunt adevărate?

A. Pentru apelul $ceFace(0)$, se vor afișa mai puțin de 30 de numere

B. 265 se afla printre numerele afișate

C. Pentru apelul $ceFace(0)$, se vor afișa mai mult de 30 de numere

D. Secvența $289, 456, 457, 458, 459$ se regăsește printre numerele afișate

603. Se consideră subprogramele `ceFace1(a, b)` și `ceFace2(a, b)` unde a și b sunt două numere naturale, cu proprietatea că $a \leq 20, b \leq 12$. Operația `<<= k` reprezintă ✓ ?

operația de shiftare la stânga a variabilei cu k biți, iar `&` reprezintă operația `AND` (bitwise).

```
Algorithm ceFace1(a, b)
   If b = 0 then
      Return 1
   EndIf
   If b MOD 2 = 1 then
      Return a * ceFace1(a, b-1)
   EndIf
   s ← ceFace1(a, n  DIV  2)
   Return s * s
EndAlgorithm
```

```
Algorithm ceFace2(a, b)
   s ← 1
   For i ← 1, b, i<<=1 execute
      If (n & i) then
         s ← s * a
      EndIf
      a ← a * a
   EndFor
   Return s
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Apelul `ceFace1(4, 4)` va returna valoarea 256

B. Apelul `ceFace1(a, b)` va returna mereu aceeași valoarea ca apelul `ceFace2(a,b)`

C. Algoritmul `ceFace2(a, b)` este mai eficient din punct de vedere al timpului de execuție decât algoritmul `ceFace1(a, b)`

D. Algoritmul `ceFace2(a, b)` este mai eficient din punct de vedere al memoriei utilizate decât algoritmul `ceFace1(a, b)`

604. Se consideră următoarea expresie logică: ✓ ?

$$((A \oplus B)\ \textbf{AND}\ (C \oplus D))\ \textbf{OR}\ (\ \textbf{NOT}\ (A\ \textbf{AND}\ B) \oplus (C\ \textbf{OR}\ D))$$

Unde $\oplus$ reprezintă operația XOR ($1 \oplus 1 = 0 \oplus 0 = 0$; $1 \oplus 0 = 0 \oplus 1 = 1$). Precizați pentru ce valori ale lui A, B, C, D, expresia are valoarea `True`.

A. $A =$ `False`, $B =$ `False`, $C =$ `True`, $D =$ `False`

B. $A =$ `True`, $B =$ `False`, $C =$ `True`, $D =$ `True`

C. $A =$ `True`, $B =$ `True`, $C =$ `True`, $D =$ `True`

D. $A =$ `True`, $B =$ `True`, $C =$ `True`, $D =$ `False`

605. Se consideră subprogramul `ceFace(n)` definit alăturat, cu $n \leq 10^9$. ✓ ?

```
Algorithm ceFace(n)
   For i ← 0, n execute
      v[i] ← 0
   EndFor
   v[0] ← 1
   v[1] ← 1
   For i ← 2, √n execute
      If v[i] = 0 then
         For j ← 2, [n/i] execute
            v[i * j] ← 1
         EndFor
      EndIf
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În șirul v, $v[i] = 1$ dacă și numai dacă i este număr compus

B. În șirul v, $v[i] = 0$ dacă și numai dacă i este număr prim

C. Complexitatea timp a algoritmului este $O(\sqrt{n})$

D. Complexitatea timp a algoritmului este $O(\log n)$

606. Se consideră algoritmul `solve(n, gas, cost)` - cu scopul găsirii unei rute pentru o mașină care trebuie să treacă prin n benzinării. La fiecare benzinărie i, poate alimenta $gas[i]$ litri, iar între benzinării i și $i+1$ consumă $cost[i]$ litri. ✓ ?

```
Algorithm SOLVE(n, gas, cost)
   If n ≤ 0 then
      Return 0
   EndIf
   s = 0
   d = 0
   For i = 1;n execute
      s = s + gas[i] − cost[i]
      If s < 0 then
         d = i
         s = 0
      EndIf
   EndFor
   Return d
EndAlgorithm
```

Care dintre următoarele afirmații este adevărată?

A. Algoritmul returnează întotdeauna o valoare între 0 și $n-1$

B. Algoritmul returnează poziția de start din care mașina poate parcurge toate benzinăriile

C. Algoritmul are complexitatea $O(n^2)$

D. Dacă suma elementelor din vectorul *gas* este mai mică decât suma elementelor din vectorul *cost*, algoritmul returnează -1

607. Se consideră algoritmul `ceFace(x)` definit alăturat, unde x și c sunt numere naturale, cel mult 10^9. ✓ ?

```
Algorithm CEFACE(x)
   If x < 10 then
      If x MOD 2 = 0 then
         Returneaza x
      Else
         Returneaza -1
   EndIf
Else
   If x MOD 2 = 0 then
      c = ceFace(x DIV 10)
      If c = −1 OR c > x MOD 10
then
         Returneaza x MOD 10
      Else
         Returneaza c
   EndIf
Else
   Returneaza ceFace(x DIV 10)
EndIf
EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează cea mai mare cifră pară a numărului x

B. Pentru $x = 719480888$, apelul `ceFace(x)` va returna valoarea 8

C. Pentru $x = 11317915$, apelul `ceFace(x)` va returna valoarea -1

D. Complexitatea timp a algoritmului este $O(\log x)$

608. Se consideră algoritmul `X(n)` unde n este un număr natural. ✓ ?

```
Algorithm X(n)
    r ← n
    For i ← 2; √n execute
        If n MOD i = 0 then
            While n MOD i = 0 execute
                n ← n DIV i
            EndWhile
            r ← r * (1 − 1 DIV i)
        EndIf
    EndFor
    If n > 1 then
        r ← r * (1 − 1 DIV n)
    EndIf
    Return r
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 12$, $X(n) = 4$

B. Pentru $n = 5$, $X(n) = 5$

C. Dacă p este un număr prim, atunci $X(p) = p - 1$

D. Funcția X este multiplicativă, adică pentru orice a și b prime între ele, $X(ab) = X(a) * X(b)$

609. Se consideră algoritmul `ceFace(v, n)` definit alăturat, unde v este un șir de n numere naturale nenule, cu $n \leq 10^9$, iar toate elementele sale sunt mai mici sau egale cu 10^5. f este un șir cu cel mult 10^5 elemente, inițial toate 0, iar șirul o este inițial gol. ✓ ?

```
Algorithm ceFace(v, n)
    m ← 0
    For i = 1; i ≤ n execute
        If v[i] > m then
            m ← v[i]
        EndIf
    EndFor
    For i = 1; i ≤ n execute
        f[v[i]] ← f[v[i]] + 1
    EndFor
    For i = 1; i ≤ m execute
        f[i] ← f[i − 1] + f[i]
    EndFor
    For i = n; i ≥ 1; −1 execute
        o[f[v[i]] − 1] ← v[i]
        f[v[i]] ← f[v[i]] − 1
    EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul `ceFace(v,n)` determină suma maximă a unei subsecvențe din v până la un indice i și o reține în final la poziția $o[i]$

B. La finalul execuției algoritmului, șirul o va reține forma sortată crescător a șirului inițial v

C. Pentru $n = 5$ și șirul $v = [1, 2, 2, 6, 5]$, la finalul algoritmului, $f = [1, 2, 0, 0, 1, 1]$

D. Algoritmul `ceFace(v, n)` are complexitatea timp $O(n + m)$

610. Să considerăm un vector de litere $l = [A, B, C, D, E]$. Se construiește un vector m (inițial vid). La fiecare pas, se poate alege una din următoarele două operații: ✓ ?

- **Mută** - se mută primul element din l la finalul lui m, apoi se elimină primul element din l.
- **Duplică** - se adaugă ultimul element din m la începutul lui m.

Observații:

- Elementele vectorului l se prelucrează în ordinea dată.
- Operația **Mută** nu poate fi folosită dacă l este vid.
- Operația **Duplică** nu poate fi folosită dacă m este vid.
- Prelucrarea se termină când vectorul l este vid.

Respectând regulile de mai sus, în ce ordine pot fi afișate literele?

A. $DCBAAABCDE$
B. $CDDAABCDE$
C. $CBABCDE$
D. $CBBBAAAABCDE$

611. Se consideră algoritmul `ceFace(n, A)` unde n este un număr natural, iar A este o matrice de dimensiune $n * n$. ✓ ?

```
Algorithm ceFace(n, A)
    d ← 1
    For i ← 1; n execute
        p ← i
        For j ← i + 1; n execute
            If |A[j][i]| > |A[p][i]| then
                p ← j
            EndIf
        EndFor
        If A[p][i] = 0 then
            Return 0
        EndIf
        If p ≠ i then
            A[i] = A[i] − A[p]
            A[p] = A[i] + A[p]
            A[i] = A[p] − A[i]
            d ← −d
        EndIf
        d ← d * A[i][i]
        For j ← i + 1; n execute
            f ← A[j][i]/A[i][i]
            For k ← i; n execute
                A[j][k] ← A[j][k] − f * A[i][k]
            EndFor
        EndFor
    EndFor
    Return d
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 3$, dacă matricea A este $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ și matricea B este $\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$, atunci $\texttt{ceFace}(3, A) = \texttt{ceFace}(3, B)$

B. Funcția `ceFace` este multiplicativă, adică pentru orice matrice A, de dimensiune n și B de dimensiune m, $\texttt{ceFace}(m * n, AB) = \texttt{ceFace}(n, A) * \text{ceFace}(m, B)$

C. Dacă matricea A are o coloană nulă, atunci $\texttt{ceFace}(n, A) = 0$

D. Pentru o matrice A de ordin 3 unde coloana a treia este formată numai din 1, $\texttt{ceFace}(3, A)$ calculează de fapt aria unui triunghi cu coordonatele din acea coloană

612. Fie algoritmii A, B, C, D, fiecare având ca parametrii v și n, unde v este un vector de n elemente. Pentru șirul $v = [456, 123, 998, 763]$ și $n = 4$, care dintre acești algoritmi vor avea același efect asupra șirului v? ✓ ?

A.
```
Algorithm A(v, n)
   A1(v, 1, n)
EndAlgorithm

Algorithm A1(v, l, h)
   If v[l] > v[h] then
      aux ← v[l]
      v[l] ← v[h]
      v[h] ← aux
   EndIf
   If (h − l + 1) ≥ 3 then
      t ← (h − l + 1) DIV 3
      A1(v, l, h − t)
      A1(v, l + t, h)
      A1(v, l, h − t)
   EndIf
EndAlgorithm
```

B.
```
Algorithm B(v, n)
   For i ← 1, n − 1 execute
      For j ← i + 1, n execute
         ok ← false
         If v[i] DIV 100 >
               v[j] DIV 100 then
            ok ← true
         EndIf
         If ok then
            aux ← v[i]
            v[i] ← v[j]
            v[j] ← aux
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

C.
```
Algorithm C(v, n)
   For i ← 1, n − 1 execute
      For j ← i + 1, n execute
         ok ← false
         If (v[i] DIV 100) ∗ (v[i] mod 7) >
               (v[j] DIV 100) ∗ (v[j] mod 7) then
            ok ← true
         Else If (v[i] DIV 100) ∗ (v[i] mod 7) =
               (v[j] DIV 100) ∗ (v[j] mod 7) AND
               (v[i] > v[j]) then
            ok ← true
         EndIf
         If ok then
            aux ← v[i]
            v[i] ← v[j]
            v[j] ← aux
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

D.
```
Algorithm D(v, n)
   For q ← 1, n − 1, q ← q · 2 execute
      For lb ← 1, n − q, q ← 2 · q execute
         mb ← lb + q
         If mb + q ≤ n then
            rb ← mb + q
         Else
            rb ← n + 1
         EndIf
         i ← lb
         j ← mb
         While i < mb AND j < rb execute
            If v[i] ≤ v[j] then
               i ← i + 1
            Else
               aux ← v[j]
```

```
                    For k ← j, i+1, -1 execute
                        v[k] ← v[k-1]
                    EndFor
                    v[i] ← aux
                    i ← i+1
                    mb ← mb+1
                    j ← j+1
                EndIf
            EndWhile
        EndFor
    EndFor
EndAlgorithm
```

613. Se dă subalgoritmul `Comp(a, b, c)`, unde a, b, c sunt numere naturale pozitive ✓ ? $(1 \leq a, b, c \leq 10000)$.

```
Algorithm Comp(a, b, c)
    If (a + b = 0) then
        Return 1
    Else If (a = c)  OR  (b = c) then
        Return 0
    Else If a ≥ b then
        Return c * Comp(a - 1, b, c - 1)
    Else
        Return Comp(a, b - 1, c - 1) + Comp(a, b, c - 1)
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate atunci când $a + b = c$ și $a \neq b$:

A. Subalgoritmul calculează numărul de partiții neordonate ale unui set de c elemente

B. Rezultatul este întotdeauna 0 dacă $a > b$

C. Valoarea returnată este C_{c-1}^{a-1}

D. Apelurile recursive generează coeficienți binomiali

614. Fie subalgoritmul `F(n)`, unde n este un număr întreg. ✓ ?

```
Algorithm F(n)
    If n ≤ 1 then
        Write 1
    Else
        m ← n DIV 2
        F(m)
        F(m)
    EndIf
EndAlgorithm
```

Care dintre următoarele variante de răspuns indică complexitatea subalgoritmului?

A. $O(n/2)$

B. $O(2^{\log_2 n})$

C. $O(n)$

D. $O(\log_2 n)$

615. Se dă subalgoritmul `Desc(n, k, m)`, unde n, k, m sunt numere naturale $(1 \leq$ ✓ ? $n, k, m \leq 10000)$ și k este impar.

```
Algorithm DESC(n, k, m)
   If (n = 0) ȘI (m = 0) then
      Return 1
   Else If (n < 0) SAU (m ≤ 0) SAU (k^2 > n) then
      Return 0
   Else
      If k ≡ 1 (mod 2) then
         Return DESC(n − k^2, k + 2, m − 1) + DESC(n, k + 2, m)
      Else
         Return DESC(n, k + 1, m)
      EndIf
   EndIf
EndAlgorithm
```

Precizați care afirmații sunt adevărate atunci când n este un pătrat perfect impar și $m = 1$:

A. Subalgoritmul returnează 1 dacă și numai dacă $k = \sqrt{n}$

B. Pentru $n = 25$, $k = 3$, $m = 1$, valoarea returnată este 0

C. Numărul de descompuneri ale lui n în suma de m pătrate impare distincte este $C^m_{\lfloor\sqrt{n}\rfloor}$

D. Apelul `Desc(n, 1, 1)` verifică dacă n este un pătrat impar

616. Doi jucători, A și B, alternează mutările, eliminând pietre dintr-un grup. Regulile sunt: la fiecare mutare, un jucător poate elimina 2^k pietre, unde $k \geq 0$ (adică $1, 2, 4, 8, \ldots$); iar câștigătorul este cel care elimină ultima piatră. ✓ ?

Care afirmații sunt adevărate pentru orice număr inițial de pietre $n \geq 1$?

A. Dacă n este o putere a lui 2, jucătorul A (primul) poate câștiga întotdeauna

B. Dacă $n = 2^m - 1$, jucătorul B are o strategie de câștig

C. Toate numerele de forma $4t + 3$ sunt poziții pierzătoare pentru A

D. Numărul minim de mutări pentru a câștiga este egal cu numărul de biți de 1 din reprezentarea binară a lui n

617. Se consideră subalgoritmul `ceFace(s, a)`, unde s este un șir de cel mult 10^5 numere întregi ($\leq 10^9$) sortate crescător, iar a, p, x, y sunt numere întregi, cel mult 10^9. Operatorul `>>` reprezintă operația de shiftare la dreapta pe biți. ✓ ?

```
1: Algorithm CEFACE(s, a)
2:     p ← 0
3:     x ← 1
4:     y ← n
5:     While x ≤ y execute
6:         m ← ------------------
7:         If s[m] ≤ a then
8:             poz ← m
9:             x ← m + 1
10:        Else y ← m − 1
11:        EndIf
12:    EndWhile
13:    Return poz
14: EndAlgorithm
```

Cu ce instrucțiune poate fi înlocuită linia 6 astfel încât algoritmul să determine cel mai mare număr din s mai mic sau egal cu a?

A. $m \leftarrow (x + y)$ `DIV 2`

B. $m \leftarrow x + ((y - x)/2)$

C. $m \leftarrow (x + y) >> 1$

D. $m \leftarrow (x/2 + y/2)$

618. Se dă subalgoritmul `ceFace(n)`, unde n este un număr natural nenul. ✓ ?

```
Algorithm ceFace(n)
   If n = 0 then
      Return 1
   Else If n = 1 then
      Return 2
   Else
      Return ceFace(n - 1) + ceFace(n - 2)
   EndIf
EndAlgorithm
```

Ce calculează acest subalgoritm?

A. Numărul de șiruri binare de lungime n care nu conțin două zerouri consecutive

B. Numărul de șiruri binare de lungime n care nu conțin doi de 1 consecutivi

C. Al n-lea termen din șirul lui Fibonacci

D. Numărul de permutări ale unei mulțimi cu n elemente

619. Considerăm un șir de elemente $S = \{s[1], s[2], \ldots, s[n]\}$ de numere întregi. Care dintre următoarele afirmații sunt adevărate pentru orice șir S? ✓ ?

A. Există cel puțin un subșir nevid cu suma divizibilă cu n

B. Numărul de subșiruri nevide cu sumă impară este 2^{n-1} dacă S conține cel puțin un număr impar

C. Dacă toate elementele sunt pare, atunci toate subșirurile nevide au sume pare

D. Dacă $n \geq 2$, există cel puțin două subșiruri diferite cu aceeași sumă

Testul 2

620. Se consideră algoritmul `G(arr, m, n)`, unde m este un număr natural ($1 \leq m \leq 100$), iar arr este un vector cu m elemente numere întregi ($arr[1], arr[2], \ldots, arr[m]$, $-10^5 \leq arr[i] \leq 10^5$, pentru $i = 1, 2, \ldots, m$) și n și i sunt două numere naturale ($1 \leq n, i \leq m$). ✓ ?

```
Algorithm G(arr, m, n, i)
   If i < n then
      For j ← 1, m − i execute
         If arr[j] > arr[j + 1] then
            arr[j] ← arr[j] * arr[j + 1]
            arr[j + 1] ← arr[j] DIV arr[j + 1]
            arr[j] ← arr[j] DIV arr[j + 1]
         EndIf
      EndFor
      G(arr, m, n, i + 1)
   EndIf
EndAlgorithm
```

Precizați care din următoarele afirmații sunt adevărate referitor la apelul algoritmului `G(arr, m, n)`.

A. După executarea apelului `G(arr, m, m, 1)`, vectorul arr va fi sortat în crescător

B. Dacă $arr = [17, 47, 13, 8, 4, 11, 68, 30, 14]$, $n = 5$ și $m = 9$, elementele din vectorul arr vor fi sortate în ordine crescătoare în urma apelului `G(arr, m, n, 1)`

C. Dacă $n < m$, după executarea algoritmului `G(arr, m, n, 1)`, primele n elemente din vector vor fi sortate în ordine crescătoare

D. Dacă $n < m$, după executarea algoritmului `G(arr, m, n, 1)`, ultimele $n + 1$ elemente din vector vor fi sortate în ordine crescătoare

621. Se consideră algoritmul `ceFace(x, y)`, unde x și y sunt numere naturale ($0 \leq x, y \leq 10^3$). ✓ ?

```
Algorithm ceFace(x, y)
   If x > y then
      x ← x − y
      y ← y + x
      x ← y − x
   EndIf
   z ← 0
   For i ← x, y execute
      If i mod 3 = 0 and i mod 2 ≠ 0 then
         z ← z + i
      EndIf
   EndFor
   Return z
EndAlgorithm
```

Care dintre următoarele situații algoritmul returnează valoarea 24?

A. $x = 4, y = 19$

B. $x = 15, y = 8$

C. $x = 6, y = 14$

D. $x = 20, y = 9$

622. Fie expresia $E = 10C_{(16)} + 136_{(7)} + 433_{(5)} - 5 * 33_{(9)}$. ✓ ?

Care valoare corespunde expresiei E?

A. $3AE9_{(16)}$
B. 312
C. $624_{(7)}$
D. Niciuna din variantele de mai sus

623. Se consideră algoritmii `f(arr, n, i)` și `call(arr, n)`, unde `arr` este un vector cu `n` elemente numere naturale ($1 \leq n \leq 10^4$, $1 \leq arr[i] \leq 10^6$, pentru $i = 1, 2, \ldots, n$) și `i` este un număr natural ($1 \leq i \leq n$). ✓ ?

```
Algorithm f(arr, n, i)
   If i = 1 then
      Return False
   EndIf
   If i < n AND arr[i] < arr[i + 1] then
      Return f(arr, n, i + 1)
   EndIf
   If i > 1 AND arr[i] > arr[i − 1] then
      If i = n OR arr[i] > arr[i + 1] then
         Return True
      EndIf
      Return f(arr, n, i + 1)
   EndIf
   Return False
EndAlgorithm
```

```
Algorithm call(arr, n)
   If n < 3 then
      Return False
   EndIf
   If arr[1] < arr[2] then
      Return f(arr, n, 2)
   EndIf
   Return False
EndAlgorithm
```

Care dintre următoarele situații algoritmul `call(arr, n)` returnează True?

A. Dacă vectorul `arr` este format din valorile [14, 16, 123, 22, 48, 15, 102, 723, 800] și $n = 9$
B. Dacă vectorul `arr` este format din valorile [14, 345, 123, 5414, 32, 15, 102, 723] și $n = 8$
C. Dacă vectorul `arr` are toate elementele ordonate strict crescător
D. Dacă și numai dacă vectorul `arr` are valori după următorul model: a, $a + 1$, ..., $a + k$, $a + k - 1$, ..., $a + 1$, a, unde k este un număr natural nenul

624. Se consideră algoritmul `redirect(n)`, unde `n` este un număr nenul ($-10^5 \leq n \leq 10^5$, `n` $\neq 0$). Operatorul / realizează împărțirea reală (ex: $3/2 = 1.5$). ✓ ?

```
Algorithm redirect(n)
   If n < 0 then
      n ← −n
   EndIf
   p ← 0; a ← 0; b ← 0
   cp ← n
   While cp > 0 execute
      a ← a + cp MOD 10
      b ← b + 1
      p ← cp MOD 10
      cp ← cp DIV 10
   EndWhile
   Return a / b ≥ p
EndAlgorithm
```

Care din următoarele afirmații sunt adevărate?

A. Algoritmul `redirect(n)` returnează True dacă și numai dacă media aritmetică a cifrelor lui `n` este strict mai mare decât prima cifră a lui `n`

B. Algoritmul `redirect(n)` returnează True dacă și numai dacă media aritmetică a ultimelor $p-1$ cifre ale lui `n` este mai mare sau egală decât prima cifră a lui `n`, unde p este numărul de cifre ale lui `n`

C. Pentru $n = -5825$, algoritmul returnează True

D. Complexitatea de timp a algoritmului este $O(\lg n)$

625. Se consideră algoritmul `ceFace(a, b)`, unde `a` și `b` sunt numere naturale ($1 \leq a, b \leq 10^6$). ✓ ?

```
1: Algorithm ceFace(a, b)
2:     If b = 0 then
3:         Return a
4:     Else
5:         k ← a MOD b
6:         Return ceFace(b, k)
7:     EndIf
8: EndAlgorithm
```

Precizați care din următoarele afirmații sunt adevărate:

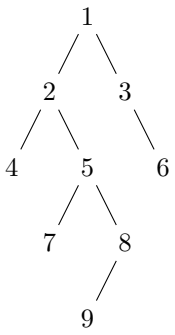
A. În urma apelului `ceFace(2025, 21)` algoritmul returnează 1

B. Algoritmul returnează cel mai mare divizor comun al numerelor `a` și `b`, doar dacă `a` sau `b` nu sunt prime

C. Pentru ca algoritmul să returneze cel mai mare divizor comun al numerelor `a` și `b`, linia 6 trebuie schimbată astfel: `Return ceFace(a MOD b, b)`

D. Algoritmul returnează 1 dacă `a` și `b` sunt prime între ele

626. Se consideră următorul arbore binar:



Care din următoarele afirmații sunt adevărate? ✓ ?

A. Parcurgerea în postordine este 4 7 9 8 5 2 6 3 1

B. Parcurgerea în preordine este 1 2 4 5 7 8 9 3 6

C. Parcurgerea în inordine este 6 3 9 8 7 5 4 2 1

D. Arborele binar nu este strict

627. Se consideră algoritmul `ceFace(n, p)`, unde n și p sunt numere naturale nenule ($1 \leq n \leq 10^6$ și $1 \leq p \leq 10^9$). ✓ ?

```
Algorithm ceFace(n, p)
    If n = 0 then
        Return 0
    EndIf
    If n MOD 2 = 0 then
        Return n MOD 10 * p * 10 + n MOD 10 DIV 2 * p + ceFace(n DIV 10, p * 100)
```

```
   EndIf
   Return n MOD 10 * p + ceFace(n DIV 10, p * 10)
EndAlgorithm
```

Precizați care din următoarele afirmații sunt adevărate referitor la apelul algoritmului `ceFace(n, 1)`.

A. Algoritmul poate returna valoarea lui n

B. Algoritmul poate returna valoarea 1213425

C. Algoritmul poate returna valoarea 1223

D. Algoritmul returnează numărul format prin dublarea cifrelor pare ale lui n și păstrarea cifrelor impare

628. Se consideră algoritmul `sim(x, y, z)`, unde x, y și z sunt valori booleene: ✓ ?

```
Algorithm SIM(x, y, z)
   While NOT x AND (y OR NOT z) execute
      Write (z OR NOT (x AND y)) OR y, ' '
      y ← NOT y
      z ← NOT (x OR y)
   EndWhile
EndAlgorithm
```

Care din următoarele afirmații sunt false?

A. Dacă $x = False$, $y = False$ și $z = False$, algoritmul afișează $True$

B. Există 3 perechi de valori (x, y, z) pentru care algoritmul afișează $True$

C. Algoritmul afișează întotdeauna cel puțin o valoare booleană

D. Pentru apelurile `sim(`$False, True, False$`)` și `sim(`$False, True, True$`)` algoritmul afișează aceeași valoare

629. Se consideră algoritmul `ceFace(a, n, b, m)`, unde n și m sunt numere naturale ✓ ? $(0 \leq n, m \leq 10^4)$, iar `a` și `b` sunt vectori cu n, respectiv m elemente numere naturale $(a[1], a[2], \ldots, a[n],\ 1 \leq a[i] \leq 10^4$, pentru $i = 1, 2, \ldots, n$ și $b[1], b[2], \ldots, b[m],\ 1 \leq b[i] \leq 10^4$, pentru $i = 1, 2, \ldots, m)$. Variabila `c` este un vector fără niciun element la apelul inițial. Vectorii `a` și `b` sunt sortați crescător.

```
Algorithm CEFACE(a, n, b, m)
   i ← 1; j ← 1; k ← 0
   While j ≤ m execute
      If i ≤ n AND a[i] < b[j] then
         i ← i + 1
      Else
         If i ≤ n AND a[i] = b[j] then
            j ← j + 1
         Else
            k ← k + 1
            c[k] ← b[j]
            j ← j + 1
         EndIf
      EndIf
   EndWhile
EndAlgorithm
```

Precizați efectul algoritmului asupra vectorului `c` în urma apelului `ceFace(a, n, b, m)`.

A. Vectorul `c` va conține toate elementele din vectorul `b` care nu sunt în vectorul `a`

B. Vectorul `c` va conține toate elementele din vectorul `a` care nu sunt în vectorul `b`

C. Vectorul `c` va conține toate elementele din vectorul `a` care sunt în vectorul `b`

D. Vectorul `c` va conține toate elementele din vectorul `b` care sunt în vectorul `a`

630. Se consideră algoritmul `pg(arr, n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^4$), iar `arr` este un vector de n elemente numere naturale ($1 \leq arr[i] \leq 10^6$, pentru $i = 1, 2, \ldots, n$). ✓ ?

```
Algorithm PG(arr, n)
    If n < 3 then
        Return True
    EndIf
    d ← arr[2] − arr[1]
    For i ← 2, n − 1 execute
        If arr[i + 1] − arr[i] ≠ d then
            Return False
        EndIf
    EndFor
    Return True
EndAlgorithm
```

Precizați pentru care dintre următoarele situații se va returna valoarea `True` în urma apelului `pg(arr, n)`.

A. Dacă `arr` = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024] și $n = 10$

B. Dacă `arr` = [19, 38, 57, 76, 95, 114, 132, 151] și $n = 8$

C. Dacă elementele vectorului `arr` formează o progresie aritmetică

D. Dacă elementele vectorului `arr` formează o progresie geometrică

631. Se consideră algoritmul `ceFace(arr, n)`, unde `arr` este un vector de `n` numere naturale ($1 \leq n \leq 10^5$, $1 \leq arr[i] \leq 10^5$, `arr[1]`, `arr[2]`, ..., `arr[n]`, pentru $i = 1, 2, \ldots, n$). Se presupune că toate elementele vectorului `arr` sunt nule la apelul inițial. ✓ ?

```
Algorithm CEFACE(arr, n)
    p ← 0
    For i ← 2, n − 1 execute
        If arr[i] = 0 then
            arr[i ∗ i] ← 2
            k ← i + 1
            While k ∗ i < n execute
                arr[k ∗ i] ← 1
                k ← k + 1
            EndWhile
        EndIf
        If arr[i] = 2 then
            p ← p + 1
        EndIf
    EndFor
    Return p
EndAlgorithm
```

Precizați care din următoarele afirmații sunt adevărate referitor la apelul algoritmului `ceFace(n)`.

A. Pentru apelul `ceFace(100)` algoritmul returnează 5

B. Pentru apelul `ceFace(256)` algoritmul returnează 6

C. Algoritmul returnează numărul de numere care au exact 3 divizori, mai mici decât n

D. Algoritmul returnează numărul de pătrate perfecte mai mici decât n

632. Se consideră algoritmul `frozen(n)`, unde `n` este un număr natural nenul ($1 \leq n \leq 10^2$). ✓ ?

```
Algorithm FROZEN(n)
   For i ← n, 2, −3 execute
      frozen(n - 1)
      Write i − 2
      frozen((i + 1) DIV 2)
      Write n + 3
   EndFor
EndAlgorithm
```

În urma apelului `frozen(6)` se afișează un șir de cifre. Care sunt cifrele de pe pozițiile 16 - 20, prima poziție fiind numerotată cu 1?

A. 10568 C. 62057

B. 80510 D. 56805

633. Se consideră algoritmul `ceFace(arr, n, k, idx)`, unde `n`, `k` și `idx` sunt numere ✓ ? naturale ($3 \leq n \leq 10^4$, $1 \leq k, idx \leq n$), iar `arr` este un vector de `n` elemente numere naturale ($1 \leq arr[i] \leq 10^4$, pentru $i = 1, 2, \ldots, n$).

```
Algorithm CEFACE(arr, n, k, idx)
   If idx = n − 1 then
      Return arr[k]
   EndIf
   arr[idx + 1] ← arr[idx] + arr[idx + 1] + arr[idx + 2] DIV 3
   Return ceFace(arr, n, k, idx + 1)
EndAlgorithm
```

Pentru care dintre următoarele afirmații pentru apelul `ceFace(arr, n, k, idx)` sunt adevărate?

A. Pentru apelul `ceFace([17, 23, 55, 31, 22, 103, 245, 322, 129, 100], 10, 3, 1)` algoritmul returnează 93

B. Pentru apelul `ceFace([14, 191, 32, 10, 77, 2, 12, 22, 72], 9, 7, 1)` algoritmul returnează 387

C. Pentru apelul `ceFace([30, 44, 121, 19, 35, 144, 2], 7, 5, 2)` algoritmul returnează 284

D. Pentru apelul `ceFace([30, 44, 121, 19, 35, 144, 2], 7, 5, 3)` algoritmul returnează 234

634. Precizați care dintre următorii algoritmi afișează corect numărul tuturor divizorilor ✓ ? lui n, unde n este un număr natural nenul ($2 \leq n \leq 10^6$). Presupunem că apelul lui `divisors3(n, i, c)` este făcut cu parametrii `divisors3(n, 1, 0)`.

A.

```
Algorithm DIVISORS1(n)
   c ← 0
   For  i ← 1, i * i ≤ n  execute
      If  n MOD i = 0  then
         c ← c + 2
      EndIf
   EndFor
   Return c
EndAlgorithm
```

B.

```
Algorithm DIVISORS2(n)
   d ← 2
   k ← 1
   While  n > 1  execute
      p ← 0
      While  n MOD d = 0  execute
         p ← p + 1
         n ← n DIV d
      EndWhile
      d ← d + 1
      k ← k * (p + 1)
   EndWhile
   Return k
EndAlgorithm
```

C.

```
Algorithm DIVISORS3(n, i, c)
   If  i * i > n  then
      Return c
   EndIf
   If  n MOD i = 0  then
      If  i * i = n  then
         c ← c + 1
      Else
         c ← c + 2
      EndIf
   EndIf
   Return divisors3(n, i + 1, c)
EndAlgorithm
```

D.

```
Algorithm DIVISORS4(n)
   d ← 2
   k ← 1
   While  d * d ≤ n  execute
      e ← 0
      While  n MOD d = 0  execute
         e ← e + 1
         n ← n DIV d
      EndWhile
      d ← d + 1
      If  e > 0  then
         k ← k * (e + 1)
      EndIf
   EndWhile
   If  n > 1  then
      k ← k * 2
   EndIf
   Return k
EndAlgorithm
```

635. Se consideră algoritmul `matrix(A, n, k)`, unde `A` este o matrice cu n linii și n coloane $(A[1][1], ..., A[1][n], ..., A[n][n])$ și n, k sunt numere naturale nenule ($1 \leq n \leq 10^2$, $1 \leq k \leq 9$). Se presupune că matricea `A` are toate elementele egale cu 0 la apelul inițial. ✓ ?

```
Algorithm MATRIX(A, n, k)
   A[1][1] ← 1
   For j ← 2, n execute
      A[1][j] ← A[1][j − 1] · k
      For i ← 2, j execute
         A[i][j] ← k · (A[i][j − 1] + A[i − 1][j − 1])
      EndFor
   EndFor
   Return A[n − 3][n − 2]
EndAlgorithm
```

Precizați ce valoare returnează algoritmul pentru apelul `matrix(A, 9, 3)`.

A. 4374

B. 4377

C. 4380

D. Niciuna din variantele de mai sus

636. Se consideră algoritmul `sec(arr, n, k)`, unde `n` este un număr natural nenul ($2 \leq n \leq 10^5$), iar `arr` este un vector de `n` numere naturale ($0 \leq arr[i] \leq 10^5$, pentru $i = 1, 2, \ldots, n$) și `k` este un număr natural nenul ($1 \leq k \leq 10^5$). ✓ ?

```
Algorithm SEC(arr, n, k)
   a ← 0; b ← 0; c ← 0
   For i ← 1,n execute
      If arr[i] MOD k = 0 then
         a ← a + 1
      Else
         If a > b then
            b ← a
            c ← 1
         Else
            If a = b then
               c ← c + 1
            EndIf
         EndIf
         a ← 0
      EndIf
   EndFor
   Write b, ' ', c
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate referitor la apelul `sec(arr, n, k)`.

A. Pentru apelul `sec([7, 3, 200, 100, 10, 9, 6, 100, 1000, 40, 1002, 20, 30], 13, 10)` algoritmul afisează 3 2

B. Pentru apelul `sec([8, 92, 35, 91, 14, 18, 21, 7, 4, 105, 28, 63], 12, 7)` algoritmul afisează 3 2

C. Dacă `arr = [2, 6, 15, 35, 63, 72, 4, 1, 48, 144, 5]` și `n = 11`, pentru orice `k` din mulțimea `M = {2,3,4,5,6,7,8,9}`, prima valoare pe care o afișează algoritmul este 2

D. Algoritmul determină lungimea maximă a unei secvențe de numere din vector care sunt multipli de `k` și numără câte secvențe de lungime maximă sunt în vector și afișează cele două valori

637. Se consideră algoritmul `ceFace(a, b)`, unde `a` și `b` sunt numere naturale nenule ✓ ? $(1 \leq a \leq 10^2, 0 \leq b \leq 25)$.

```
1: Algorithm CEFACE(a, b)
2:    If b = 0 then
3:       Return 1
4:    EndIf
5:    c ← CEFACE(a, b DIV 2)
6:    If b MOD 2 = 0 then
7:       Return c * c
8:    Else
9:       Return c* CEFACE(a, b DIV 2) *
   a
10:   EndIf
11: EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate referitor la apelul algoritmului `ceFace(a, b)`.

A. Pentru apelul `ceFace(2, 14)` algoritmul returnează 16384, algoritmul autoapelându-se de 14 ori

B. Algoritmul returnează a^b, în complexitatea de timp $O(\log_2(b))$

C. Algoritmul returnează a^b, în complexitatea de timp $O(b)$

D. Dacă "Linia 9" ar fi înlocuită astfel: `Return ceFace(a, b - 1) * a`, apelul `ceFace(2, 6)` se autoapelează de 22 ori

638. Se consideră algoritmul `X(a, b, c, arr)`, unde `a`, `b`, `c` sunt numere naturale nenule ✓ ? $(1 \leq a, b, c \leq 10^3)$ și `arr` este un vector de n numere naturale nenule $(1 \leq arr[i] \leq 10^3$, pentru $i = 1, 2, ..., n)$.

```
Algorithm X(a, b, c, arr)
   If a = b then
      d ← arr[a]
      f ← arr[b]
      While d execute
         e ← d MOD 10
         If e = c then
            Return 0
         EndIf
         d ← d DIV 10
      EndWhile
      If f ≠ b then
         Return 1
      Else
         Return 0
      EndIf
   Else
      Return X(a, (a + b) DIV 2,
c, arr) +
          X((a + b) DIV 2 + 1, b,
c, arr)
   EndIf
EndAlgorithm
```

Pentru care dintre următoarele apeluri ale algoritmului `X(a, b, c, arr)` se va returna valoarea 5?

A. `X(1, 9, 3, [91, 23, 4, 12, 33, 403, 87, 60, 302])`

B. `X(1, 9, 4, [34, 201, 39, 404, 77, 14, 7, 42, 92])`

C. `X(1, 6, 4, [85, 104, 872, 602, 44, 73, 92])`

D. `X(1, 13, 7, [28, 103, 694, 4, 333, 901, 19, 844, 24, 10, 11, 77, 13])`

639. Se consideră algoritmul `ceFace(arr, l, r, val)`, unde `arr` este un vector ordonat descrescător, având n elemente numere naturale ($1 \leq n \leq 10^4$, $1 \leq$ `arr[i]` $\leq 10^6$, pentru $i = 1, 2, \ldots, n$) și val este un număr natural ($1 \leq val \leq 10^6$ și `arr[n]` $\leq val \leq$ `arr[1]`). Parametrii l și r sunt numere naturale ($1 \leq l \leq r \leq n$).

```
Algorithm ceFace(arr, l, r, val)
   If l > r then
      Return arr[l]
   EndIf
   m ← (l + r) DIV 2
   ____________________
EndAlgorithm
```

Precizați cu ce secvență de cod trebuie completat algoritmul, astfel încât în urma apelului `ceFace(arr, 1, n, val)` să se returneze valoarea maximă din vectorul `arr` care este mai mică sau egală cu `val`.

A.

```
If arr[m] = val then
   Return arr[m]
EndIf
If arr[m] < val then
   Return ceFace(arr, l, m - 1,
val)
Else
   Return ceFace(arr, m + 1, r,
val)
EndIf
```

B.

```
If arr[m] = val then
   Return arr[m]
EndIf
If arr[m] < val then
   Return ceFace(arr, m + 1, r,
val)
Else
   Return ceFace(arr, l, m - 1,
val)
EndIf
```

C.

```
If arr[m] ≤ val then
   Return ceFace(arr, l, m - 1,
val)
Else
   Return ceFace(arr, m + 1, r,
val)
EndIf
```

D.

```
If arr[m] ≥ val then
   Return ceFace(arr, m + 1, r,
val)
Else
   Return ceFace(arr, l, m - 1,
val)
EndIf
```

640. Se consideră algoritmul `algo(q, w, arr, z)`, unde q, w, z sunt numere naturale nenule ($1 \leq q, w, z \leq 10, z \leq q$) și arr este un vector cu $z + 1$ elemente naturale ($arr[0], arr[1], \ldots, arr[z]$). Presupunem că *arr[0]* este inițializat cu 0.

```
Algorithm ALGO(q, w, arr, z)
   If w ≤ z then
      For i ← arr[w−1]+1, q−z+w execute
         arr[w] ← i
         ALGO(q, w + 1, arr, z)
      EndFor
   Else
      For i ← 1, z execute
         Write arr[i]
      EndFor
      Write ' '              ▷ **Spațiu**
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt false referitoare la algoritmul alăturat?

A. Pentru apelul `algo(8, 1, arr, 3)`, algoritmul afișează 55 caractere de spațiu

B. Pentru apelul `algo(4, 1, arr, 2)`, algoritmul afișează 12 13 14 23 24 34

C. Apelul `algo(q, 1, arr, z)` afișează toate aranjamentele de `q` luate câte `z`

D. Apelurile `algo(n, 1, arr, k)` și `algo(n, 1, arr, n - k)` afișează aceleași valori întotdeauna, unde `k` este un număr natural nenul ($1 \leq k \leq n$)

641. Care dintre algoritmii următori pot fi implementați în așa fel încât să aibă complexitate de timp liniară ($O(n)$)?

A. Algoritmul care returnează suma elementelor dintr-un vector, folosind Divide et Impera

B. Algoritmul de descompunere în factori primi a unui număr

C. Algoritmul de sortare a unui vector cu elemente doar cifre folosind metoda numărării

D. Algoritmul de căutare a unui element într-un vector sortat folosind metoda căutării binare

642. Se consideră algoritmul `ceFace(arr, n, k)`, unde n și k sunt numere naturale ($1 \leq n, k \leq 10^4$), iar `arr` este un vector de n numere naturale ($x[1], x[2], \ldots, x[n], 1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, \ldots, n$).

```
Algorithm ceFace(arr, n, k)
   If k = 0 then
      Return 1
   EndIf
```

```
    If n ≤ 0 then
        Return 0
    EndIf
    t1 ← ceFace(arr, n - 1, k)
    If arr[n] mod 2 = 0 then
        Return t1+ ceFace(arr, n - 3, k - arr[n] * 2)
    EndIf
    Return t1+ ceFace(arr, n - 3, k - arr[n] * 3)
EndAlgorithm
```

Precizați care afirmații sunt adevărate referitor la apelul algoritmului `ceFace([8, 15, 11, 3, 3, 7, 6, 10, 12], 9, k)`.

A. Pentru $k = 45$, algoritmul returnează 5

B. Pentru $k = 33$, algoritmul returnează 3

C. Pentru $k = 33$, algoritmul returnează 4

D. Pentru $k = 29$, algoritmul returnează 2

643. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^6$). ✓ ?

```
Algorithm ceFace(n)
    k ← 0
    For i ← 1, n execute
        p ← i
        While p > 0 execute
            k ← k + p MOD 2
            p ← p DIV 2
        EndWhile
    EndFor
    Return k
EndAlgorithm
```

Precizați ce se afișează în urma apelului `ceFace(2046)`.

A. 11242

B. 11257

C. 11249

D. 11253

Testul 3

644. Se consideră algoritmul `ceFace(a, n)`, unde a este o matrice pătratică de dimensiune n $\left(1 \leq n \leq 10^3\right)$ cu elemente întregi $(-10^9 \leq a[i][j] \leq 10^9)$. ✓ ?

```
Algorithm ceFace(a, n)
    m ← −∞
    For i ← 1, n execute
        x ← 0, y ← 0
        For j ← 1, n execute
            x ← x + a[i][j]
            y ← y + a[i][n − j + 1]
        EndFor
        m ← max(m, |x − y|)
    EndFor
    Return m
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul returnează valoarea 0 dacă si numai daca toate elementele matricei sunt egale

B. Numărul minim de comparații necesare pentru a găsi valoarea maximă este n

C. Valoarea inițială a lui m poate sa fie 1

D. Algoritmul are complexitatea de timp $O(n^2)$ datorită parcurgerii complete a matricei

645. Se consideră algoritmul `ceFace(n, k)`, unde `n` și `k` sunt numere naturale $(1 \leq n \leq 100, 0 \leq k \leq n)$. ✓ ?

```
1: Algorithm ceFace(n, k)
2:     If k = 0 or k = n then
3:         Return 1
4:     EndIf
5:     Return _____ + _____
6: EndAlgorithm
```

Cu ce apeluri ar trebui înlocuite spațiile libere astfel încât algoritmul să returneze rezultatul C_n^k?

A. `ceFace(n - 1, k - 1); ceFace(n - 1, k)`

B. `ceFace(n - 1, k); ceFace(n - 1, n - k)`

C. `ceFace(n - 1, n - k); ceFace(n - 1, n - k - 1)`

D. `ceFace(n - 1, k); ceFace(n, k - 1)`

646. Se consideră algoritmul `ceFace(v, n, k)`, unde `v` este un vector de `n` numere întregi $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `k` este un număr natural nenul $(1 \leq k \leq n)$. ✓ ?

```
1: Algorithm ceFace(v, n, k)
2:     m ← −∞, s ← 0
3:     For i ← 1, n execute
4:         s ← s + v[i]
5:         If i ≥ k then
6:             m ← max(m, s)
7:             s ← s − v[i − k + 1]
8:         EndIf
9:     EndFor
10:    Return m
11: EndAlgorithm
```

În care dintre cazurile de mai jos rezultatul algoritmului `ceFace(v, n, k)` va fi diferit de rezultatul actual?

A. Inițializarea lui m cu 0 în loc de $-\infty$

B. Schimbarea condiției de la $i \geq k$ la $i > k$

C. Eliminarea liniei $s \leftarrow s - v[i - k + 1]$

D. Interschimbarea liniilor 6 și 7

647. Se consideră algoritmul `F(n, k)`, unde n și k sunt numere naturale ($1 \leq n \leq 10^9, 0 \leq k \leq n$). ✓ ?

```
Algorithm F(n, k)
    If k > (n DIV 2) then
        k ← n − k
    EndIf
    r ← 1
    For i ← 0, k − 1 execute
        r ← r · (n − i) DIV (i + 1)
    EndFor
    Return r
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul calculează numărul submulțimilor ordonate cu k elemente dintr-o mulțime cu n elemente

B. Algoritmul efectuează 3 înmulțiri pentru apelul `ceFace(100, 97)`

C. Algoritmul optimizează calculul coeficientului binomial aplicând proprietatea de simetrie $C(n, k) = C(n, n - k)$

D. Algoritmul efectuează 97 înmulțiri pentru apelul `ceFace(100, 97)`

648. Se consideră algoritmul `ceFace(v, n, t)`, unde v este un vector sortat de n numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar t este un număr întreg. ✓ ?

```
Algorithm CEFACE(v, n, t)
    a ← 1, b ← n, r ← −1
    While a ≤ b execute
        m ← a + (b − a) DIV 2
        If v[m] = t then
            r ← m
            b ← m − 1
        Else If v[m] < t then
            a ← m + 1
        Else
            b ← m − 1
        EndIf
    EndWhile
    Return r
EndAlgorithm
```

Care dintre următoarele afirmații nu au sens din punct de vedere logic în acest algoritm?

A. Calculul lui m cu formula $a + (b - a)$DIV2

B. Algoritmul returnează -1 când găsește valoarea t, în loc să returneze poziția

C. Pentru $n = 2^{31} - 1$, algoritmul intră în buclă infinită din cauza overflow-ului la calculul lui m

D. Condiția $a \leq b$ din `while` permite accesarea unor elemente din afara vectorului de numere

649. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `n` este un număr natural nenul ($1 \leq n \leq 10^5$). ✓ ?

```
Algorithm CEFACE(v, n)
    For i ← 1, n execute
        x ← v[i]
        For j ← i + 1, n execute
            If v[j] < x then
                x ← v[j]
            EndIf
        EndFor
        r[i] ← x
    EndFor
    Return r
EndAlgorithm
```

Care dintre următoarele afirmații descriu corect comportamentul algoritmului?

A. Pentru $v = [5, 3, 4, 2, 6]$, rezultatul este $[2, 2, 2, 2, 6]$

B. Dacă v este sortat strict crescător, rezultatul va fi v însuși

C. Numărul de atribuiri ale variabilei x este $O(n^2)$ în cel mai rău caz

D. Complexitatea temporală poate fi redusă la $O(n)$ prin stocarea minimului curent într-o parcurgere inversă

650. Se consideră algoritmul `ceFace(v, n, x)`, unde `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `x` este un număr întreg. ✓ ?

```
Algorithm ceFace(v, n, x)
    a ← −1, b ← −1
    For i ← 1, n execute
        If v[i] = x then
            If a = −1 then
                a ← i
            EndIf
            b ← i
        EndIf
    EndFor
    Return b − a + 1
EndAlgorithm
```

Pentru care dintre următoarele date de intrare se returnează valoarea 5, unde $n = 7$?

A. $v = [1, 42, 42, 42, 42, 42, 1], x = 42$

B. $v = [7, 1, 7, 7, 7, 2, 7], x = 7$

C. $v = [3, 7, 7, 7, 7, 7, 1], x = 7$

D. $v = [1, 2, 42, 42, 1, 42, 2], x = 42$

651. Se consideră algoritmul `X(v, n)`, unde `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `n` este un număr natural nenul ($1 \leq n \leq 10^5$). ✓ ?

```
Algorithm X(v, n)
    For i ← 2, n execute
        v[i] ← v[i] · v[i − 1]
    EndFor
    Return v[n]
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul calculează produsul tuturor elementelor vectorului

B. După execuție, vectorul de numere va conține la fiecare index i produsul tuturor numerelor pana la poziția respectivă

C. Complexitatea algoritmului este $O(n)$

D. Pentru $v = [2, 3, 4, 5]$, vectorul final va fi $[2, 6, 24, 120]$

652. Se consideră algoritmul `f(v, n)`, unde `v` este un vector de `n` numere întregi ($-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9$), iar `n` este un număr natural nenul ($1 \leq n \leq 10^5$). ✓ ?

```
Algorithm f(v, n)
    m ← v[1]
    For i ← 2, n execute
        If v[i] < m then
            v[i − 1] ← m
            m ← v[i]
        EndIf
    EndFor
    v[n] ← m
    Return m
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Algoritmul modifică vectorul original înlocuind elementele cu minimul precedent

B. Returnează valoarea minimă din vectorul de numere

C. Complexitatea algoritmului este $O(n)$

D. Ultimul element al vectorului rămâne nemodificat

653. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi ✓ ? $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `n` $(1 \leq n \leq 10^6)$. Care dintre următoarele instrucțiuni pot înlocui liniile `5` și `6` astfel încât la finalul execuției algoritmului, pentru $v = [4, 1, 5, 2, 3, 7, 6]$, $r = [3, 0, 4, 1, 2, 6, 5]$?

```
1: Algorithm ceFace(v, n)
2:     For i ← 1, n execute
3:         c ← 0
4:         For j ← 1, n execute
5:             If _______ then
6:                 _______
7:             EndIf
8:         EndFor
9:         r[i] ← c
10:    EndFor
11:    Return r
12: EndAlgorithm
```

A. `5:` `v[j] < v[i]`
   `6:` `c ← c + 1`

B. `5:` `v[i] > v[j]`
   `6:` `c ← c + 2`

C. `5:` `v[j] ≥ v[i]`
   `6:` `c ← c + 2`

D. `5:` `v[i] ≥ v[i]`
   `6:` `c ← c + 1`

654. Se consideră algoritmul `ceFace(s, t)`, unde `s` și `t` sunt vectori de caractere cu ✓ ? lungimile n și m $(1 \leq n, m \leq 10^3)$.

```
Algorithm ceFace(s, t, n, m)
    For i ← 1, n + 1 execute
        a[i][1] ← 0
    EndFor
    For j ← 1, m + 1 execute
        a[1][j] ← 0
    EndFor
    For i ← 2, n + 1 execute
        For j ← 2, m + 1 execute
            If s[i − 1] = t[j − 1] then
                a[i][j] ← a[i − 1][j − 1] + 1
            Else
                a[i][j] ←
                    max(a[i − 1][j], a[i][j − 1])
            EndIf
        EndFor
    EndFor
    Return a[n + 1][m + 1]
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Valoarea returnată pentru vectorii `"abcde"` și `"ace"` este 3

B. Funcția construiește o matrice pentru a determina lungimea subsecvenței comune maxime

C. Funcția calculează numărul de operații de inserare necesare pentru a transforma `s` în `t`

D. Funcția necesită $O(n + m)$ spațiu de memorie

655. Se consideră algoritmul `ceFace(v, n, target)`, unde v este un vector sortat de n ✓ ? numere întregi $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar *target* este un număr natural.

```
Algorithm ceFace(v, n, target)
    a ← 1, b ← n, pos ← −1
    While a ≤ b execute
        m ← (a + b) DIV 2
        If v[m] ≤ target then
            pos ← m
            a ← m + 1
        Else
            b ← m − 1
        EndIf
```

```
    EndWhile
    Return pos
EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Algoritmul returnează poziția celui mai mare număr din vectorul v mai mic sau egal decât `target`

B. Complexitatea algoritmului este $O(n \log n)$

C. Pentru $target < 0$ algoritmul returnează mereu `-1`

D. Algoritmul returnează mereu indexul primei apariții a lui `target`

656. Fie două liste de elemente Q și S. Pentru lista de elemente Q se definesc următoarele operații: ✓ ?

(a) `Put(Q, a)` - inserează variabila a la sfârșitul listei.

(b) `Remove(Q)` - returnează și extrage din listă prima valoare.

(c) `First(Q)` - returnează prima valoare din listă fără să o extragă.

Pentru lista de elemente S se definesc operațiile:

(a) `Push(S, a)` - inserează variabila a înaintea primului element din listă.

(b) `Pop(S)` - returnează și extrage valoarea primului element din listă.

(c) `Top(S)` - returnează valoarea primului element fără să îl elimine din listă.

Dacă lista S este goală, de câte ori se execută bucla `while` în secvența de cod de mai jos, pentru $Q = [16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$?

```
While Q execute
    If !S OR Top(S) ≤ First(Q) then
        x ← Remove(Q)
        Push(S, x)
    Else
        x ← Pop(S)
        Put(Q, x)
    EndIf
EndWhile
```

A. 128 C. 64

B. 256 D. 32

657. Se consideră algoritmul `ceFace(v, n)`, unde `v` este un vector de `n` numere întregi $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `n` este un număr natural nenul $(1 \leq n \leq 10^6)$. ✓ ?

```
Algorithm CEFACE(v, n)
    For i ← 1, n execute
        p[i] ← p[i − 1] + v[i]
    EndFor
    Return p[n]
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul permite obținerea fără parcurgere a sumei unui subșir consecutiv

B. Algoritmul egalează vectorul p cu v după finalizarea algoritmului

C. Valoarea returnată este suma tuturor elementelor din vectorul `v`

D. Algoritmul efectuează n^2 operații pentru calcularea sumei finale

658. O navă spațială trebuie să se deplaseze din punctul de plecare (sx, sy) către punctul de destinație (dx, dy), însă poate efectua doar două tipuri de mișcări: din orice poziție (x, y), nava poate ajunge fie în $(x, x + y)$, fie în $(x + y, y)$. Funcția de mai jos returnează 1 dacă nava poate ajunge de la plecare la destinație și 0 în caz contrar. Toate coordonatele sunt pozitive. ✓ ?

```
Algorithm PATH(sx, sy, dx, dy)
   If (A) then
      Return 0
   EndIf
   If sx = dx AND sy = dy then
      Return 1
   EndIf
   Return (B)
EndAlgorithm
```

Alegeți variantele corecte pentru A și B:

A. $A : sx > dx$ `OR` $sy > dy$
B : `Path`$(sx + sy, sy, dx, dy)$ `AND` `Path`$(sx, sy + sx, dx, dy)$.

B. $A : sx > dx$ `AND` $sy > dy$
B : `Path`$(sx + sy, sy, dx, dy)$ `AND` `Path`$(sx, sy + sx, dx, dy)$

C. $A : sx > dx$ `OR` $sy > dy$
B : `Path`$(sx + sy, sy, dx, dy)$ `OR` `Path`$(sx, sy + sx, dx, dy)$

D. $A : sx > dx$ `AND` $sy > dy$
B : `Path`$(sx + sy, sy, dx, dy)$ `OR` `Path`$(sx, sy + sx, dx, dy)$

659. Ce valoare va returna apelul `F(35)` pentru funcțiile `F` și `G` definite mai jos? ✓ ?

```
Algorithm G(x)
   If x < 10 then
      Return 2 * x
   Else
      Return 2 * F(x DIV 2)
   EndIf
EndAlgorithm
```

```
Algorithm F(x)
   If x < 8 then
      Return (2 + x)
   Else
      Return 2 + G(x - 2)
   EndIf
EndAlgorithm
```

A. 34 B. 42 C. 43 D. 35

660. Se consideră algoritmul `f(v, st, dr)`, unde `v` este un vector de `n` numere întregi $(-10^9 \leq v[1], v[2], \ldots, v[n] \leq 10^9)$, iar `st` și `dr` reprezintă limitele intervalului de căutare. ✓ ?

```
1: Algorithm F(v, st, dr)
2:     If st = dr then
3:         Return v[st]
4:     EndIf
5:     m ← (st + dr) DIV 2
6:     x ← f(v, st, m)
7:     y ← f(v, m + 1, dr)
8:     Return max(x, y)
9: EndAlgorithm
```

Care dintre următoarele afirmații sunt false?

A. Algoritmul face suma elementelor din vector

B. Algoritmul este echivalent dacă schimbam linia 5 cu $m \leftarrow st + (st + dr)$DIV2

C. vectorul trebuie să fie sortat pentru ca algoritmul sa parcurgă toate elementele

D. $st = dr = 1$ se va întâmpla măcar o dată pentru orice vector care respecta condițiile din enunț

661. Se consideră algoritmul `F(x)`, unde `v` este un vector de 10 elemente, astfel încât $v = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, iar x este un număr natural cuprins în intervalul $[1, 11]$. ✓ ?

```
Algorithm F(x)
   If x ≤ 10 then
      If v[x] MOD 2 ≠ 0 then
         Afișează v[x], ' '
      EndIf
      If x ≤ 8 then
         v[x] ← 0
         F(v[x + 2])
         Afișează x
      EndIf
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate pentru `F(1)`?

A. Algoritmul afișează 1 3 5 7 9 7531

B. După finalizarea algoritmului vectorul va fi $v = [0, 2, 0, 4, 0, 6, 0, 8, 0, 10]$

C. Algoritmul afișează 2 4 6 8 10 7531

D. După finalizarea algoritmului vectorul va fi $v = [1, 0, 3, 0, 5, 0, 7, 0, 9, 0]$

662. Următoarele subprograme determină maximul elementelor unui vector v care conține numerele $v = \{13, 113, 97, 6, -1, 69, 74, 31, 111, 97\}$. ✓ ?

```
Algorithm Maxim(v[], n)
   i ← 1
   j ← n
   While E execute
      If v[i] ≤ v[j] then
         i ← i + 1
      Else
         j ← j − 1
      EndIf
   EndWhile
   Return v[i]
EndAlgorithm
```

```
Algorithm estePrim(x)
   If x ≤ 1 then
      Return false
   EndIf
   For d ← 2, √x execute
      If x MOD d = 0 then
         Return false
      EndIf
   EndFor
   Return true
EndAlgorithm
```

Care este condiția cu care trebuie înlocuit E din algoritmul Maxim astfel încât sa fie corect conform cerinței ?

A. $j = i$

B. $j \geq i$ `AND estePrim(v[i])`

C. $j \neq i$

D. $i < j - 1$ `OR estePrim(v[i])`

663. Se consideră algoritmul `D(n)`, unde `n` este un număr natural nenul ($1 \leq n \leq 10^9$). ✓ ?

```
Algorithm D(n)
   suma ← 0
   For d ← 1, √n execute
      If n MOD d = 0 then
         suma ← suma + d
         If d ≠ n DIV d then
```

```
            suma ← suma − (n DIV d)
          EndIf
        EndIf
    EndFor
    Return suma
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul calculează suma tuturor divizorilor lui $2 * n$

B. Rezultatul este mereu negativ sau nul pentru $3 \leq n$

C. Algoritmul calculează rădăcina numărului dacă este pătrat perfect

D. Complexitatea algoritmului este $O(\sqrt{n})$

664. Se consideră algoritmul `functieRecursiva(n, k)`, unde `n` și `k` sunt numere naturale nenule ($1 \leq n, k \leq 10^3$). ✓ ?

```
Algorithm FUNCTIERECURSIVA(n, k)
   If n ≤ 1 then
      Return 1
   EndIf
   If k = 0 then
      Return 1
   EndIf
   Return functieRecursiva(n-1, k)
+ functieRecursiva(n-1, k-1) ×n
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 5$ și $k = 2$ avem 13 auto-apeluri

B. Soluția obținută poate fi optimizată prin memorarea soluțiilor calculate cel puțin o dată

C. Algoritmul funcționează corect și pentru $n < k$

D. Complexitatea de timp poate fi aproximata la $O(2^n)$

665. Se consideră algoritmul `f(x, n)`, unde `x` este un vector de `n` valori de tip `bool` ($0 \leq n \leq 1000$). ✓ ?

```
1: Algorithm F(x, n)
2:    rezultat ← true
3:    For i ← 1, n − 1 execute
4:       For j ← i + 1, n execute
5:          rezultat ← rezultat AND (x[i] OR  NOT x[j])
6:       EndFor
7:    EndFor
8:    Return rezultat
9: EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru orice doi indici i și j ($i < j$), algoritmul verifică dacă $x[i] = 1$ sau $x[j] = 0$

B. Formula $x[i]$ **OR NOT** $x[j]$ poate fi redusă la $x[i]$ prin principiul absorbției

C. Schimbarea semnului **AND** în **OR** ar rezulta în returnarea valorii true mereu

D. Dacă inițializăm rezultat cu 0, iar linia 5 ar fi $rezultat \leftarrow rezultat + 1$. Algoritmul al calcula numărul de comparații în sortarea prin selecție

666. Fie subalgoritmul f. Dacă $m > n$, $m, n \in \mathrm{Z}$, care este complexitatea cea mai compactă (fără factori sau termeni care pot fi eliminați) a acestui subalgoritm? ✓ ?

```
Algorithm F(n, m)
    s ← 100
    For i ← 1, n execute
        For j ← i + 1, m execute
            s ← s + j DIV 2
        EndFor
    EndFor
    For i ← n, m execute
        s ← s + 1
    EndFor
EndAlgorithm
```

A. $O(n^2)$

B. $O(n \cdot (m - n))$

C. $O(n \cdot m)$

D. $O(m - n)$

667. Pe o tablă sunt scrise numerele de la 1 la N. La fiecare pas, se aleg două numere x și y și se înlocuiesc cu numărul $(x + y)$ **MOD** 10. Procesul continuă până când rămâne un singur număr. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Rezultatul final va fi același indiferent de ordinea în care numerele sunt alese

B. Pentru $n = 93$, rezultatul va fi 1

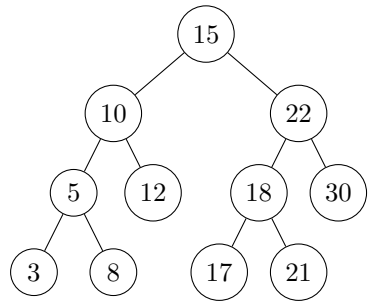
C. Pentru $n = 432$, rezultatul va fi 6

D. Dacă alegem doar perechi de numere consecutive, rezultatul fiecărei operații efectuate va fi un număr natural par

Testul 4

668. Se consideră algoritmul `Tree(node)`, unde `node` reprezintă un nod al arborelui, și arborele binar: ✓ ?

```
Algorithm TREE(node)
   If node = null then
      Return 0
   EndIf
   s ← node.val
   If node.st ≠ null AND node.dr ≠ null then
      temp ← node.st
      While temp.dr ≠ null execute
         temp ← temp.dr
      EndWhile
      s ← s + temp.val
      temp ← node.dr
      While temp.st ≠ null execute
         temp ← temp.st
      EndWhile
      s ← s + temp.val
   EndIf
   Return s + TREE(node.st) + TREE(node.dr)
EndAlgorithm
```



Pentru apelul `Tree(node)`, ce va returna algoritmul, dacă valoarea transmisă ca parametru este nodul rădăcină al arborelui ?

A. 390 B. 250 C. 310 D. 176

669. Fie expresia $E = AB_{(16)} + 120_{(3)} - 120_{(4)} + 2 * 44_{(5)}$, unde x_a reprezintă valoarea numărului x în baza de numerație a. Care dintre următoarele valori este egală cu valoarea expresiei E? ✓ ?

A. $322_{(8)}$ B. $162_{(10)}$ C. $1000101_{(2)}$ D. $278_{(8)}$

670. Se consideră algoritmul `sortare(v, n)`, unde `n` este un număr natural iar v este un vector de `n` numere naturale $(v[1], v[2], \ldots, v[n])$: ✓ ?

```
Algorithm SORTARE(v, n)
   For i ← 1, n − 1 execute
      For j ← i + 1, n execute
         u1 ← v[i] MOD 10
         u2 ← v[j] MOD 10
         check ← False
         If u1 > u2 then
            check ← True
         Else If u1 = u2
              AND v[i] > v[j] then
            check ← True
```

```
            EndIf
            If check then
                aux ← v[i]
                v[i] ← v[j]
                v[j] ← aux
            EndIf
        EndFor
    EndFor
    Return v
EndAlgorithm
```

A. Algoritmul sortează elementele vectorului în ordine descrescătoare după ultima cifră a fiecărui element

B. Dacă două elemente au aceeași ultimă cifră, algoritmul le sortează în ordine crescătoare

C. Algoritmul utilizează metoda de sortare prin selecție pentru a ordona elementele vectorului

D. Complexitatea timp a algoritmului este $O(n \log n)$

671. Se consideră algoritmul `Process(v, n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^5$), iar `v` este un vector cu n elemente numere naturale, și $1 \leq v[i] \leq 10^9$ pentru orice $i = 1, 2, \ldots, n$: ✓ ?

```
Algorithm Process(v, n)
    r ← 0
    smax ← 1
    lcurr ← 1
    s ← v[1]
    For i ← 2, n execute
        If Check(v[i-1], v[i]) then
            lcurr ← lcurr + 1
            s ← s + v[i]
        Else
            If lcurr > smax then
                smax ← lcurr
                r ← s
            Else If lcurr = smax  AND
                    s > r then
                r ← s
            EndIf
            lcurr ← 1
            s ← v[i]
        EndIf
    EndFor
    If lcurr > smax then
        r ← s
    Else If lcurr = smax  AND s > r then
        r ← s
    EndIf
    Return r
EndAlgorithm
```

```
Algorithm Check(a, b)
    suma ← 0
    sumb ← 0
    x ← a
    y ← b
    While x > 0 execute
        suma ← suma + (x MOD 10)
        x ← x DIV 10
    EndWhile
    While y > 0 execute
        sumb ← sumb + (y MOD 10)
        y ← y DIV 10
    EndWhile
    Return |suma − sumb| ≤ 2
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul găsește suma celei mai lungi secvențe de elemente consecutive cu proprietatea că diferența dintre sumele cifrelor a oricare două elemente consecutive este cel mult 2

B. Algoritmul returnează suma uneia dintre cele mai lungi secvențe de elemente consecutive cu proprietatea că diferența dintre sumele cifrelor oricăror două elemente din secvență este mai mică decât 3

C. Pentru orice vector în care toate elementele au aceeași sumă a cifrelor, rezultatul va fi suma tuturor elementelor

D. Dacă există mai multe secvențe de lungime maximă, algoritmul returnează suma cea mai mare dintre sumele acestora

672. Se consideră algoritmul `ceFace(v, st, dr, x)`, unde v este un vector de numere naturale $(v[1], v[2], \ldots, v[n])$, iar `st`, `dr` și `x` sunt numere naturale, cel mult 10^5: ✓ ?

```
Algorithm ceFace(v, st, dr, x)
    If st = dr then
        Return v[st]
    EndIf
    mid ← (st + dr) DIV 2
    sumSt ← 0, sumDr ← 0
    For i ← st, mid execute
        sumSt ← sumSt + v[i]
    EndFor
    For i ← mid + 1, dr execute
        sumDr ← sumDr + v[i]
    EndFor
    If sumSt < sumDr then
        For i ← 1, (dr − mid) execute
            tmp ← v[st + i − 1]
            v[st + i − 1] ← v[mid + i]
            v[mid + i] ← tmp
        EndFor
    EndIf
    lVal ← ceFace(v, st, mid, 1 - x)
    rVal ← ceFace(v, mid + 1, dr, 1 - x)
    If x = 1 then
        Return lVal + rVal
    EndIf
    If lVal ≠ 0  AND rVal ≠ 0 then
        Return lVal * rVal
    Else
        Return lVal + rVal
    EndIf
EndAlgorithm
```

Precizați care dintre afirmațiile de mai jos sunt adevărate:

A. Algoritmul reorganizează secvențele din vector comparând suma elementelor din prima jumătate cu cea din a doua. Dacă suma elementelor din segmentul stâng este mai mică decât cea din segmentul drept, atunci se interschimbă elementele corespunzătoare ale celor două segmente

B. Algoritmul utilizează parametrul x pentru a determina ordinea reorganizării segmentelor, comparând sumele elementelor din jumătățile vectorului

C. Pentru apelul `ceFace([3, 1, 4, 1, 5, 9], 1, 6, 1)` algoritmul va returna valoarea 30

D. Pentru apelul `ceFace([7, 5, 8, 3, 6, 2], 1, 6, 1)` algoritmul va returna valoarea 115

673. Se consideră algoritmul `Build(n, k, ant)`, unde `n` este un număr natural ($1 \leq n \leq 10^5$), `k` reprezintă un număr natural și `ant` este un număr natural sau -1: ✓ ?

```
Algorithm BUILD(n, k, ant)
   If k = 0 then
      Return 1
   EndIf
   If n = 0 then
      Return 0
   EndIf
   s ← 0
   ult ← n MOD 10
   rest ← n DIV 10
   v ← 1
   If ant ≥ 0 then
      If ult > ant then
         v ← ((ult − ant) MOD 2 = 1)
      Else
         v ← ((ant − ult) MOD 2 = 1)
      EndIf
   EndIf
   If v then
      s ← s+ BUILD(rest, k − 1, ult)
   EndIf
   s ← s+ BUILD(rest, k, ant)
   Return s
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru apelul `Build(12345, 3, -1)` algoritmul va returna valoarea 5

B. Pentru apelul `Build(65678, 4, -1)` algoritmul va returna valoarea 3

C. Pentru apelul `Build(356438, 2, -1)` subsecvența cu numărul șapte este 34

D. Pentru apelul `Build(97546, 2, -1)` algoritmul va returna valoarea 6

674. Se consideră subalgoritmul `ceFace(n, k, a, b)`, unde n și k sunt numere naturale ✓ ? $(a \leq n, k \leq 10^9)$, iar a și b sunt doi vectori cu câte n elemente. Vectorul b este inițializat cu 0.

```
Algorithm CEFACE(n, k, a, b)
   For i ← 1, n execute
      b[i] ← b[i − 1] + a[i]
   EndFor
   st ← 0
   dr ← 0
   c ← 0
   For i ← k, n execute
      If b[i] − b[i − k] > c then
         c ← b[i] − b[i − k]
         st ← i − k + 1
         dr ← i
      EndIf
   EndFor
   For i ← st, dr execute
      Write a[i], " "
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Subalgoritmul afișează secvența de lungime maximă din șirul a aflată în ordine crescătoare.

B. Subalgoritmul afișează secvența de lungime și sumă maximă din șirul a.

C. Subalgoritmul afișează secvența de lungime k din șirul a cu suma elementelor maximă.

D. Subalgoritmul afișează secvența de lungime k din șirul a cu suma elementelor exact egală cu k (dacă există, altfel $st > dr$, deci nu se va afișa nimic).

675. Se consideră algoritmul `ceFace(n)`, unde `N` este un număr natural nenul $(1 \leq n \leq$ ✓ ? $10^5)$.

```
Algorithm ceFace(n)
   count ← 0
   For i ← 1, n execute
      x ← i
      While x < n execute
         count ← count + 1
         x ← x * 2
      EndWhile
   EndFor
   Return count
EndAlgorithm
```

Care este complexitatea de timp a algoritmului `ceFace(n)`?

A. $O(n)$

B. $O(n * \log_2(n))$

C. $O(n^2)$

D. $O(n * \sqrt{n})$

676. Se consideră algoritmul `Prime(n)`, unde `n` este un număr natural nenul ($1 \leq n \leq 10^9$). ✓ ?

```
Algorithm Prime(n)
   r ← 0
   p ← 1
   d ← 2
   While n > 1 execute
      c ← 0
      While n MOD d = 0 execute
         c ← c + 1
         n ← n DIV d
      EndWhile
      If c > 0 then
         If c > 1 then
            r ← r + d * p
         Else
            r ← r + d * d * p
         EndIf
         p ← p * 10
      EndIf
      If d * d > n AND n > 1 then
         d ← n
      Else
         d ← d + 1
      EndIf
   EndWhile
   Return r
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 72$, algoritmul returnează 423

B. Pentru $n = 100$, algoritmul returnează 425

C. Algoritmul construiește un număr în care fiecare factor prim p care apare de k ori în descompunere contribuie cu p^2 dacă $k = 1$, respectiv cu p dacă $k > 1$

D. Pentru orice număr prim p, algoritmul va returna p^2

677. Se consideră algoritmul `Base(n)`, unde `n` este un număr natural nenul ($1 \leq n \leq 10^9$). ✓ ?

```
Algorithm BASE(n)
    r ← 0
    For b ← 2, 9 execute
        nr ← n
        s ← 0
        cifp ← 0
        While nr > 0 execute
            c ← nr MOD b
            s ← s + c
            If c MOD 2 = 0  AND c > 0 then
                cifp ← cifp + 1
            EndIf
            nr ← nr DIV b
        EndWhile
        If s MOD b = 0 then
            If cifp > 0 then
                r ← r + b * 10 + cifp
            Else
                r ← r + s
            EndIf
        EndIf
    EndFor
    Return r
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru fiecare bază de la 2 la 9, dacă suma cifrelor numărului în acea bază este divizibilă cu baza, atunci se adaugă la rezultat valoarea formată din baza înmulțită cu 10 și adunată cu numărul de cifre pare nenule, dacă există astfel de cifre. În caz contrar, se adaugă doar suma cifrelor

B. Pentru apelul `Base(16)`, algoritmul returnează 62

C. Pentru apelul `Base(15)`, algoritmul returnează 21

D. Pentru orice putere a lui 2, rezultatul va conține cel puțin o cifră impară

678. Într-o bibliotecă, toate cele m cărți sunt aranjate pe un raft, fiecare carte având un loc specific atribuit conform unui sistem de organizare. Fiecare carte i are poziția sa i pe raft. Deoarece bibliotecarii nu sunt stricți cu ordinea, atunci când rearanjează cărțile, acestea sunt plasate în primul spațiu disponibil fără a respecta exact pozițiile inițiale. Care este numărul posibilităților de rearanjare a cărților astfel încât cel mult două cărți să nu fie pe locul lor inițial? ✓ ?

A. $\frac{m(m-1)(m-2)}{6} + \frac{m(m-1)}{2} + 1$

B. $\frac{m(m-1)}{2} + 1$

C. $\frac{m!}{m-2} + 1$

D. Nicio variantă

679. Se consideră algoritmul `Search(v, n)`, unde `v` este un vector sortat crescător cu `n` elemente numere naturale $(1 \leq n \leq 10^5), (1 \leq v[i] \leq 10^9)$. ✓ ?

```
Algorithm SEARCH(v, n)
    c ← 0
    For i ← 1, n execute
        s ← ALGO(v[i])
        st ← i + 1
        dr ← n
        While st ≤ dr execute
            m ← (st + dr) DIV 2
            If v[m] = v[i] * s then
                c ← c + 1
                Break
            Else If v[m] < v[i] * s
then
                st ← m + 1
            Else
                dr ← m − 1
            EndIf
        EndWhile
    EndFor
    Return c
EndAlgorithm
```

```
Algorithm ALGO(x)
    s ← 0
    While x > 0 execute
        s ← s + (x MOD 10)
        x ← x DIV 10
    EndWhile
    Return s
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul numără câte elemente din vector au proprietatea că există un element ulterior egal cu produsul dintre elementul curent și suma cifrelor sale

B. Pentru apelul `Search([2, 4, 8, 16, 32, 64], 6)`, algoritmul va returna valoarea 3

C. Complexitatea algoritmului este $O(n * \log n)$

D. Pentru orice vector care conține doar puteri ale lui 2, rezultatul va fi 0

680. Se consideră algoritmul `ceFace(A, n, m)` unde `A` este o matrice de dimensiune n * m, iar n și m sunt numere naturale nenule ($1 \leq n, m \leq 10^5$) și elementele matricei sunt numere naturale astfel încât ($1 \leq A[i][j] \leq 10^5$). Se consideră următorul algoritm: ✓ ?

```
Algorithm CEFACE(A, n, m)
    r ← 0
    For i ← 1, n execute
        maxl ← A[i][1]
        p ← 1
        For j ← 2, m execute
            If A[i][j] > maxl then
                maxl ← A[i][j]
                p ← j
            EndIf
        EndFor
        If CHECK(A, n, p, i, maxl)
then
            r ← r + maxl
        EndIf
    EndFor
    Return r
EndAlgorithm
```

```
Algorithm CHECK(A, n, c, exlin, val)
    For i ← 1, n execute
        If i ≠ exlin AND A[i][c] ≥ val
then
            Return False
        EndIf
    EndFor
    Return True
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul calculează suma elementelor care au valoarea maximă atât pe linia cât și pe coloana lor

B. Pentru matricea: $\begin{bmatrix} 3 & 7 & 2 \\ 1 & 5 & 8 \\ 4 & 6 & 9 \end{bmatrix}$ și apelul algoritmului `Algo(A, 3, 3)`, acesta returnează valoarea 16

C. Complexitatea algoritmului este $O(n * m)$

D. Pentru orice matrice în care toate elementele de pe diagonala principală sunt strict mai mari decât restul elementelor de pe coloanele lor, acestea vor fi incluse în sumă

681. Se consideră algoritmul `Cool(v, n)`, unde n este un număr natural nenul ($1 \leq n \leq 10^5$), iar `v` este un vector cu n elemente numere naturale $(v[1], v[2], \ldots, v[n])$. ✓ ?

```
Algorithm COOL(v, n)
    r ← 0
    For i ← 1, n − 1 execute
        If v[i] < v[i+1] then
            temp ← ALGO(v[i])
            If temp = ALGO(v[i+1]) then
                r ← r + 1
            EndIf
        EndIf
    EndFor
    Return r
EndAlgorithm
```

```
Algorithm ALGO(x)
    s ← 0
    While x > 0 execute
        s ← s + (x MOD 10)
        x ← x DIV 10
    EndWhile
    Return s
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul numără câte perechi de elemente consecutive din vector sunt în ordine strict crescătoare și au aceeași sumă a cifrelor

B. Pentru apelul `Algo([12, 21, 13, 31, 15], 5)`, algoritmul returnează 2, adică perechile (12,21) și (13,31)

C. Pentru un vector sortat crescător în care toate elementele au aceeași sumă a cifrelor, rezultatul va fi n-1

D. Pentru apelul `Algo([21, 30, 15, 24, 42, 51], 6)`, algoritmul returnează valoarea 4

682. Se consideră algoritmul `ceFace()` definit alăturat. $a[][]$ este un tablou bidimensional care reprezintă matricea de adiacență a unui graf ceFacerientat cu n noduri, $n \leq 1000$ (așadar, dacă $a[i][j] = 1$ înseamnă că există o muchie de la nodul i la nodul j). $v[]$ este un tablou unidimensional cu n elemente, inițial toate 0. Metoda `d(i, a, v, n, c)` apelată de algoritmul `ceFace()` este, de asemenea, definită alăturat. ✓ ?

```
Algorithm ceFace(a, n)
    c ← 0
    For i ← 1, n execute
        If v[i] = 0 then
            d(i, a, v, n, c)
            c ← c + 1
        EndIf
    EndFor
EndAlgorithm
```

```
Algorithm d(i, a, v, n, c)
    v[i] ← c
    For j ← 1, n execute
        If a[i][j] = 1 & v[j] = 0 then
            d(j, a, v, n, c)
        EndIf
    EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul `ceFace(a,n)` verifică dacă graful dat este conex;

B. Algoritmul `d(i, a, v, n, c)` realizează o parcurgere în adâncime a grafului, notând în șirul v un număr diferit pentru fiecare două noduri (i, j) care nu se află în aceeași componentă conexă;

C. Algoritmul `ceFace(a,n)` verifică dacă graful dat este bipartit;

D. La finalul execuției algoritmului `ceFace(a,n)`, pentru orice nod i, $v[i]$ va conține numărul de ordine al componentei conexe în care i se află în graf.

683. Se consideră două șiruri, A și B, de lungime n. Fiecare două elemente aflate pe aceeași poziție formează o pereche. Notăm cu d diferența dintre numărul de perechi în care elementul din șirul A este mai mare decât elementul din șirul B și numărul de perechi în care elementul din șirul B este mai mare decât elementul din șirul A. Elementele celor două șiruri se rearanjează, astfel încât valoare lui d să fie cât mai mare. ✓ ?

Pentru care din următoarele seturi de date, valoarea finală a lui d este corectă?

A. n=5, A=[5 3 8 6 2], B=[7 4 5 9 3], d=2.

B. n=5, A=[5 3 8 6 2], B=[7 4 5 9 3], d=1.

C. n=10, A=[12 4 18 7 13 6 15 3 9 10], B=[8 14 11 6 16 5 17 2 10 7], d=8.

D. n=15, A=[50 10 25 35 5 40 15 60 75 55 30 65 70 45 20], B=[60 25 80 15 10 45 75 30 35 55 50 20 65 70 5], d=11.

684. Se consideră algoritmul `Verifica(n, a)`, unde n este un număr natural ($1 \leq n \leq 10^3$) și a este un vector cu n elemente numere întregi $(a[1], a[2], ..., a[n])$, unde $-100 \leq a[i] \leq 100$, pentru $i = 1, 2, ..., n$: ✓ ?

```
Algorithm VERIFICA(n, a)
    m ← n DIV 2
    If n MOD 2 = 0 then
        k ← 1
    Else
        k ← 2
    EndIf
    For i ← m + k,n execute
        j ← n − (i − m) + k
        If i < j then
            aux ← a[i]
            a[i] ← a[j]
            a[j] ← aux
        EndIf
    EndFor
    For i ← 1,m execute
        If a[i] ≠ a[n − i + 1] then
            Return False
        EndIf
    EndFor
    Return True
EndAlgorithm
```

Se cere să se determine pentru ce valori ale vectorului a, apelul `Verifica(n, a)` returnează **True**.

A. $a = [1, 2, 3, 4, 3, 2, 1]$;

B. $a = [1, 2, 3, 3, 3, 1, 2, 1]$;

C. $a = [1, 2, 3, 1, 2]$;

D. $a = [3, 6, 3, 6, 3, 6, 3, 6]$.

685. Se consideră algoritmul `ceFace(n, p)`, unde n este un număr natural nenul ($1 \leq n \leq 10^5$), iar k este un număr natural nenul: ✓ ?

```
Algorithm CEFACE(n, p)
    c ← 0
    For i ← 1, n execute
        ndiv ← COUNTER(i)
        If ndiv ≥ p then
            c ← c + 1
        EndIf
    EndFor
    Return c
EndAlgorithm
```

```
Algorithm COUNTER(n)
    t ← 0
    i ← 1
    While i ∗ i ≤ n execute
        If n MOD i = 0 then
            If i ∗ i ≠ n then
                t ← t + 2
            Else
                t ← t + 1
            EndIf
        EndIf
        i ← i + 1
    EndWhile
    Return t
EndAlgorithm
```

Precizați care dintre afirmațiile următoare referitoare la algoritmul `ceFace(n, prag)` sunt adevărate:

A. Algoritmul numără câte numere din intervalul $[1, n]$ au cel puțin `prag` divizori

B. Algoritmul `ceFace(n, p)` determină numărul cu cei mai mulți divizori

C. Pentru apelul `ceFace(10, 4)`, algoritmul returnează 3

D. Pentru apelul `ceFace(16, 2)`, algoritmul returnează 15

686. Se consideră algoritmul `Magic(n, p)`, unde n este un număr natural nenul ($1 \leq n \leq 10^6$), iar k este un număr natural nenul: ✓ ?

```
Algorithm MAGIC(n, p)
    r ← 0
    For i ← 1, n execute
        d ← FIND(i)
        If d MOD p = 0 then
            r ← r + 1
        EndIf
    EndFor
    Return r
EndAlgorithm
```

```
Algorithm FIND(n)
    s ← 1
    i ← 2
    While i * i ≤ n execute
        If n MOD i = 0 then
            If i * i ≠ n then
                s ← s + i + (n DIV i)
            Else
                s ← s + i
            EndIf
        EndIf
        i ← i + 1
    EndWhile
    Return s
EndAlgorithm
```

Ce valoare va fi returnată în urma apelului `Magic(45, 2)` ?

A. 18
B. 20
C. 17
D. 47

687. Se consideră algoritmul `X(v, n, k)`, unde n este un număr natural ($1 \leq n \leq 10^4$), ✓ ? v este un vector cu n elemente numere întregi $(v[1], v[2], \ldots, v[n])$, iar k este un număr natural:

```
Algorithm X(v, n, k)
    s ← BUILD(v, n)
    c ← 0
    For i ← 1, n execute
        For j ← 1, i execute
            If s[i] − s[j − 1] = k then
                c ← c + 1
            EndIf
        EndFor
    EndFor
    Return c
EndAlgorithm
```

```
Algorithm BUILD(v, n)
    s[1] ← 0
    For i ← 2, n execute
        s[i] ← s[i − 1] + v[i]
    EndFor
    Return s
EndAlgorithm
```

Care dintre următoarele afirmații referitoare la algoritmul `X(v, n, k)` sunt adevărate?

A. Pentru apelul `X([1, 2, 3, 4, 5], 5, 2)`, algoritmul returnează 2
B. Algoritmul numără câte subsecvențe ale vectorului `v` au suma exact k
C. Pentru apelul `X([2, -1 ,3, 5, -2, 4], 6, 4)`, algoritmul returnează 2
D. Algoritmul are o complexitate de timp de $O(n^2)$

688. Se consideră algoritmul `ceFace(v, n, idx, sumc, k, c)`, unde n este un număr ✓ ? natural ($1 \leq n \leq 10^4$), v este un vector cu n elemente numere întregi $(v[1], v[2], \ldots, v[n])$, iar k, *sumc*, c și *idx* sunt numere naturale:

```
Algorithm ceFace(v, n, idx, sumc, k, c)
   If sumc = k then
      c ← c + 1
      Return
   EndIf
   If idx ≥ n  OR sumc > k then
      Return
   EndIf
   ceFace(v, n, idx + 1, sumc + v[idx], k,
c)
   ceFace(v, n, idx + 1, sumc, k, c)
EndAlgorithm
```

```
Algorithm Start(v, n, k)
   c ← 0
   ceFace(v, n, 0, 0, k, c)
   Return c
EndAlgorithm
```

Precizați care dintre afirmațiile următoare sunt adevărate:

A. Algoritmul `Start(v, n, k)` returnează numărul de subseturi ale vectorului `v` a căror sumă este exact `k`

B. Complexitatea de timp a algoritmului este $O(2^n)$

C. Algoritmul funcționează corect doar dacă toate elementele vectorului `v` sunt numere pozitive

D. Pentru apelul `Start([2, 5, 4, 6, 7, 9], 6, 15)`, algoritmul returnează 4

689. Prietenul nostru Luca a învățat la ora de informatică despre backtracking și cum poate fi utilizat pentru a genera toate subseturile unui vector de numere întregi. Luca trebuie să determine toate subseturile unui vector ale căror sume se încadrează într-o limită maximă m și care dintre acești algoritmi funcționează corect și returnează toate subseturile valide. Variabilele p, s, t, m sunt numere naturale. Vectorul **v** conține un număr de **n** elemente numere naturale și reprezintă mulțimea numerelor din care se pot forma subseturile, iar vectorul **sol** este un vector binar, utilizat pentru a marca elementele incluse într-un subset. De asemenea, indexările vectorilor încep de la 1. Care dintre algoritmii de mai jos este cel care identifică corect și eficient toate subseturile care respectă condițiile date?

A.

```
Algorithm Find(p, s, t, m)
   If p > n then
      If s > 0  AND s ≤ m then
         For i ← 0, n − 1 execute
            If sol[i] = 1 then
               Write v[i] " "
            EndIf
         EndFor
         Write newLine
         t ← t + 1
      EndIf
      Return
   EndIf
   sol[p] ← 1
   Find(p + 1, s + v[p], t, m)
   sol[p] ← 0
   Find(p + 1, s, t, m)
EndAlgorithm
```

B.

```
Algorithm Find(p, s, t, m)
   If p = n then
      If s > 0  AND s ≤ m then
         For i ← 1, n execute
            If sol[i] = 1 then
               Write v[i] " "
            EndIf
         EndFor
         Write newLine
         t ← t + 1
      EndIf
      Return
   EndIf
   sol[p] ← 1
   news ← s + v[p]
   If news ≤ m then
      Find(p + 1, news, t, m)
   EndIf
EndAlgorithm
```

C.

```
Algorithm FIND(p, s, t, m)
   If p = n then
      If s > 0  AND s ≤ m then
         For i ← 1, n execute
            If sol[i] = 1 then
               Write v[i]
            EndIf
         EndFor
         Write newLine
         t ← t + 1
      EndIf
      Return
   EndIf
   If s + v[p] ≤ m then
      sol[p] ← 1
      FIND(p + 1, s + v[p], t, m)
   EndIf
   sol[p] ← 0
   FIND(p + 1, s, t, m)
EndAlgorithm
```

D.

```
Algorithm FIND(p, s, t, m)
   If p = n then
      If s > 0  AND s ≤ m then
         For i ← 1, n execute
            If sol[i] = 1 then
               Write v[i]
            EndIf
         EndFor
         Write newLine
         t ← t + 1
      EndIf
      Return
   EndIf
   sol[p] ← 1
   FIND(p + 1, s + v[p], t, m)
   If s + v[p] ≤ m then
      For i ← 1, n execute
         If sol[i] = 1 then
            Write v[i]
         EndIf
      EndFor
      Write newLine
      t ← t + 1
   EndIf
   sol[p] ← 0
   FIND(p + 1, s, t, m)
EndAlgorithm
```

690. Se consideră algoritmul `func(x, y, d)` unde `x` și `y` reprezintă linia, respectiv coloana unui element dintr-o matrice `mat`. Se consideră că dacă pe poziția (x, y) din matrice se află elementul 0, atunci acesta reprezintă o cale validă. Dacă pe poziția (x, y) se află elementul cu valoarea 1, atunci acesta este un obstacol, iar dacă pe poziția (x, y) se află elementul cu valoarea 2, atunci aceasta reprezintă comoara ce trebuie găsită. Variabila d este un număr natural. De asemenea, se dă o matrice adițională `v` cu aceeași dimensiune ca matricea inițială. Se consideră următorul algoritm: ✓ ?

```
Algorithm FUNC(x, y, d)
   If x < 0 or y < 0 or x ≥ n or y ≥ m or mat[x][y] = 1 or v[x][y] then
      Return
   EndIf
   If mat[x][y] = 2 then
      If d < m then
         m ← d
      EndIf
      Return
   EndIf
   v[x][y] ← True
   FUNC(x + 1, y, d + 1)
   FUNC(x - 1, y, d + 1)
   FUNC(x, y + 1, d + 1)
   FUNC(x, y - 1, d + 1)
   v[x][y] ← false
EndAlgorithm
```

Care dintre următoarele afirmații referitoare la algoritmul `func(x, y, d)` sunt adevărate?

A. Algoritmul `func(x, y, d)` găsește cel mai lung drum posibil în matrice pentru a ajunge la comoară

B. Pentru matricea: $\begin{bmatrix} 0 & 0 & 1 & 0 & 2 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ și apelul algoritmului `func(3, 1, 0)`, valoarea d va fi 6

C. Algoritmul `func(x, y, d)` găsește cel mai scurt drum posibil în matrice pentru a ajunge la comoară

D. Algoritmul `func(x, y, d)` compară pe rând fiecare element din matrice, și salvează în d cea mai mică valoare din matrice

691. Se consideră algoritmul `ceFace(n, x, len, last)`, unde n este un număr natural ($1 \leq n \leq 10^4$) și x, *len*, *last* sunt, de asemenea, numere naturale: ✓ ?

```
Algorithm ceFace(n, x, len, last)
   If len = n then
      For i ← 1, n execute
         Write sol[i]
      EndFor
      Write newLine
      Return 1
   EndIf
   count ← 0
   For d ← 1, x execute
      If x MOD d = 0  AND (len = 0  OR last * d < x) then
         sol[len] ← d
         count ← count+ ceFace(n, x, len + 1, d)
      EndIf
   EndFor
   Return count
EndAlgorithm
```

Care dintre următoarele afirmații referitoare la algoritmul `ceFace(n, x, len, last)` sunt adevărate?

A. Pentru apelul `ceFace(4, 9, 0, 1)`, algoritmul returnează valoarea 9

B. Algoritmul `ceFace(n, x, len, last)` determină toate șirurile de n elemente ale mulțimii divizorilor lui x, ordonate crescător, pentru care suma oricăror două elemente consecutive este mai mică decât x

C. În urma apelului `ceFace(5, 10, 0, 1)`, șirul al 8-lea care va fi generat este 11212

D. Algoritmul `ceFace(n, x, len, last)` determină toate secvențele de lungime n formate din divizori ai lui x, în care produsul oricăror două elemente consecutive este mai mic decât x

Testul 5

692. Fie variabilele `x = 14`, `y = 5` şi `z = 2`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big((x+1)\ \texttt{MOD}\ (y-1)+z\Big)\ \texttt{DIV}\ \Big((x+1)\ \texttt{DIV}\ (z+1)\Big)+((y-1)\ \texttt{MOD}\ z)$$

A. 2 B. 1 C. 4 D. 5

693. Se consideră un arbore cu 12 noduri reprezentat prin vectorul de taţi `t = (5, 1, 5, 3, 0, 4, 1, 7, 11, 7, 3, 4)`. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

A. Există un lanţ între nodurile 6 şi 9 care nu conţine rădăcina arborelui

B. Nodurile 2 şi 10 se află în subarbori diferiţi ai rădăcinii arborelui

C. Suma valorilor cu care sunt marcate frunzele arborelui este 47

D. Lungimea lanţului dintre nodurile 6 şi 10 este 5

694. Se consideră expresia logică: $(E_1$ **AND** $E_2)$ **OR** (**NOT** $(E_3$ **OR** $E_4))$, unde E_1, E_2, E_3, E_4 sunt expresii logice cu valorile adevărat (1) sau fals (0). Pentru ce valori ale E_1, E_2, E_3, E_4 de mai jos, expresia este evaluată ca fiind adevărată? ✓ ?

A. $E_1 = 1, E_2 = 0, E_3 = 1, E_4 = 0$ C. $E_1 = 1, E_2 = 1, E_3 = 1, E_4 = 1$

B. $E_1 = 1, E_2 = 1, E_3 = 0, E_4 = 0$ D. $E_1 = 1, E_2 = 0, E_3 = 1, E_4 = 1$

695. Se consideră algoritmul `What(a, n, i, j)`, unde n este lungimea unui vector a format din n elemente întregi $(a[1], a[2], ..., a[n])$. ✓ ?

```
Algorithm WHAT(a, n, i, j)
    aux[1] ← a[1]
    For k ← 2,n execute
        aux[k] ← aux[k − 1] + a[k]
    EndFor
    If i = 1 then
        Return aux[j]
    Else
        Return aux[j] − aux[i − 1]
    EndIf
EndAlgorithm
```

Ce face algoritmul?

A. Calculează suma elementelor din intervalul $[i, j]$

B. Determină numărul de elemente impare din intervalul $[i, j]$

C. Găsește valoarea maximă dintre elementele din intervalul $[i, j]$

D. Inversează ordinea elementelor din vectorul inițial

696. Se consideră algoritmul `Maze(m, n)`, unde n este dimensiunea unei matrice pătratice m de dimensiune n x n, iar m este o matrice binară $(m[1][1], m[1][2], ..., m[n][n])$ în care `1` reprezintă drum și `0` reprezintă obstacol. ✓ ?

```
Algorithm MAZE(m, n, i, j)
    If i = n and j = n then
        Return True
    EndIf
    If i ≤ n and j ≤ n and m[i][j] = 1 then
        If MAZE(m, n, i + 1, j) or MAZE(m, n, i, j + 1) then
```

```
            Return True
        EndIf
    EndIf
    Return False
EndAlgorithm
```

Ce face algoritmul `Maze`?

A. Determină dacă există un drum de la $(1,1)$ la (n,n) doar prin deplasări spre dreapta și în jos

B. Determină dacă există un drum de la (n,n) la $(1,1)$ doar prin deplasări spre stânga și în sus

C. Calculează numărul de drumuri posibile până la ieșire

D. Afișează drumul optim găsit în matrice

697. Se consideră o matrice pătratică de dimensiune $n \times n$ și o funcție g care primește ca parametru un număr și returnează 1 dacă acel număr este par și 0 altfel. Care dintre următoarele abordări determină dacă matricea este formată numai din numere pare? ✓ ?

A. Verificăm dacă funcția g, aplicată pe fiecare element al matricei, returnează întotdeauna 1

B. Verificăm dacă suma valorilor returnate de g, aplicată pe fiecare element al matricei, este egală cu numărul total de elemente ale matricei

C. Verificăm dacă funcția g, aplicată pe fiecare element al matricei, returnează cel puțin o dată 1

D. Aplicăm funcția g pe elemente alese aleatoriu din matrice până când suma valorilor egale cu 1 este mai mică decât numărul total de elemente ale matricei

698. Se dă următorul algoritm *algorithm*(n), cu n număr natural nenul. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

```
Algorithm ALGORITHM(n)
    For i ← 1, n execute
        p[i] ← 0
    EndFor
    p[1] ← 1
    For i ← 2, √n execute
        If p[i] = 0 then
            For j ← 2, n DIV i execute
                p[i · j] ← 1
            EndFor
        EndIf
    EndFor
EndAlgorithm
```

A. Algoritmul marchează în vectorul p valorile prime cu 0, iar pe cele compuse cu 1

B. Algoritmul determină numărul de divizori ai fiecărui număr mai mic sau egal cu n

C. Dacă $n = 20$, atunci $p[2] = 1$

D. Dacă $n = 20$, atunci $p[10] = 1$

699. Se dă următorul algoritm $J(n, m)$, n şi m numere naturale nenule. Care dintre următoarele este complexitatea ca timp de execuţie a acestuia? ✓ ?

```
Algorithm J(n, m)
   If n ≤ 1 then
      Return 1
   Else
      n ← n DIV m
      J(n, m)
      J(n, m)
      J(n, m)
   EndIf
EndAlgorithm
```

A. $O(m^{log_3 n})$

B. $O(3^{log_m n})$

C. $O(3^{log_n m})$

D. $O(n^{log_m 3})$

700. Se consideră următorii doi algoritmi : `A(a, n)` și `B(a, n)`, unde n este lungimea unui vector a format din n elemente întregi $(a[1], a[2], ..., a[n])$. ✓ ?

```
Algorithm A(a, n)
   For i ← 1, n execute
      For j ← 1, n − i execute
         If a[j]≥a[j + 1] then
            swap(a[j], a[j + 1])
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

```
Algorithm B(a, n)
   For i ← 1, n execute
      min ← i
      For j ← i + 1, n execute
         If a[j] < a[min] then
            min ← j
         EndIf
      EndFor
      If min ≠ i then
         swap(a[i], a[min])
      EndIf
   EndFor
EndAlgorithm
```

Ce instrucțiune ar trebui schimbată în algoritmul `A(a, n)` astfel încât ambii algoritmi să aibă același efect asupra vectorului a?

A. Înlocuirea $a[j]\geq a[j+1]$ cu $a[j] > a[j+1]$

B. Înlocuirea $a[j]\geq a[j+1]$ cu $a[j] < a[j+1]$

C. Nu trebuie înlocuit nimic, algoritmii sunt deja echivalenți

D. Înlocuirea $a[j]\geq a[j+1]$ cu $a[j] \neq a[j+1]$

701. Se dă următorul algoritm $P(k, n, a, f)$, unde k este un număr natural ($k \leq 20$) cu valoarea iniţială egală cu 1, n număr natural ($n \leq 20$), iar a şi f sunt 2 vectori de numere naturale de lungime n, f având elementele iniţializate cu 0. Numerotarea indicilor vectorilor începe de la 1. ✓ ?

```
Algorithm P(k, n, a, f)
   If k = n + 1 then
      For i ← 1, n execute
         Write a[i], ' '
      EndFor
      Write newline
   Else
      For i ← 1, n execute
         If f[i] = 0 AND i ≠ k then
            a[k] ← i
            f[i] ← 1
```

```
            P(k + 1, n, a, f)
            f[i] ← 0
         EndIf
      EndFor
   EndIf
EndAlgorithm
```

Care dintre următoarele afirmaţii sunt adevărate?

A. Algoritmul va afişa 2 soluţii pentru $n = 3$

B. Algoritmul va afişa 24 de soluţii pentru $n = 4$

C. Algoritmul generează permutările şirului numerelor de la 1 la n $\{1, 2, ..., n\}$, pentru care valoarea elementului este diferită de poziţia pe care se află aceasta

D. Algoritmul generează permutările şirului numerelor de la 1 la n $\{1, 2, ..., n\}$, pentru care valoarea fiecărui element este mai mare decât poziția sa

702. Se consideră algoritmul `Counter(a, n)`, care primește ca parametri un șir a de n elemente și numărul natural n $(1 \leq n \leq 10000)$. ✓ ?

```
Algorithm COUNTER(a, n)
   For i ← 1, n execute
      For j ← 1, n − i execute
         If a[j] > a[j + 1] then
            swap(a[j], a[j + 1])
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Dacă toate elementele sunt egale, se vor efectua 0 interschimbări

B. Dacă șirul este sortat crescător, se vor efectua 0 interschimbări

C. Dacă șirul este sortat descrescător, se vor efectua $n(n-1)$ interschimbări

D. Nicio variantă nu este adevărată

703. Care dintre următoarele afirmații sunt adevărate despre algoritmul `Sky(a, n)` prezentat mai jos, care primește ca parametri un șir a de n elemente și numărul natural n $(1 \leq n \leq 10000)$? ✓ ?

```
Algorithm SKY(a, n)
   For i ← 1, n − 1 execute
      min ← i
      For j ← i + 1, n execute
         If a[j] < a[min] then
            min ← j
         EndIf
      EndFor
      If min ≠ i then
         swap(a[i], a[min])
      EndIf
   EndFor
EndAlgorithm
```

A. Complexitatea în cel mai rău caz este $O(n^2)$

B. Complexitatea în cel mai bun caz este $O(n \log n)$

C. Complexitatea în cel mai bun caz este $O(n^2)$

D. Algoritmul face întotdeauna același număr de comparații, indiferent de ordinea elementelor

Problemele **704.** și **705.** se referă la următorul algoritm: `Order(a, n)`, în care n este un număr natural, $n = 2^k, k$ număr natural, și a este un vector de lungime n, indexat de la 0, transmis prin referință. Fiecare element din a are o valoare diferită.

704. Care dintre următoarele afirmații sunt adevărate? ✓ ?

```
Algorithm Order(a, n)
   If n = 1 then
      Return
   EndIf
   For i ← 0, n DIV 2 − 1 execute
      b[i] ← a[2 · i]
      c[i] ← a[2 · i + 1]
   EndFor
   Order(b, n DIV 2)
   Order(c, n DIV 2)
   For i ← 0, n DIV 2 − 1 execute
      a[i] ← b[i]
      a[n DIV 2 + i] ← c[i]
   EndFor
EndAlgorithm
```

A. Pentru $n = 16$, elementul inițial aflat la indexul 7 în a se va afla la final la indexul 6

B. Pentru $n = 512$, elementul inițial aflat la indexul 124 în a se va afla la final tot la indexul 124

C. Numărul de poziții care vor avea la final același element este egal cu $2^{k/2}$

D. Numărul de poziții care vor avea la final același element este egal cu $2^{(k+1)/2}$

705. Care este complexitatea algoritmului `Order`? ✓ ?

A. $O(n \log n)$ B. $O(n^2)$ C. $O(nk)$ D. $O(n)$

706. Se dă algoritmul `ceFace(n)`, în care n este un număr natural. Vectorul s conține n elemente, fiecare fiind inițializat cu valoarea 0. ✓ ?

```
Algorithm ceFace(n)
   For i ← 1, n execute
      For j ← i, n, i execute
         s[j] ← s[j] + i
      EndFor
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 10000, s[518] = 394$

B. Pentru $n = 10000, s[518] = 912$

C. Pentru $n = 10000, s[12] = 28$

D. Pentru $n = 12132, s[7129] = 7130$

707. Fie algoritmul `Repeta(n)`, unde n este un număr natural. ✓ ?

```
1: Algorithm Repeta(n)
2:    If n < 10 then
3:       Return n
4:    EndIf
5:    s ← Repeta(n DIV 10)
6:          +n MOD 10
7:    If s ≥ 10 then
8:       Return Repeta(s)
9:    EndIf
10: EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

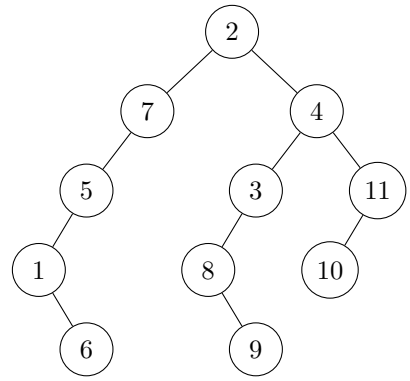
A. Pentru $n = 178342$, algoritmul returnează valoarea 7

B. Pentru $n = 178342$, algoritmul returnează valoarea 6

C. Pentru $n = 22138124382131243984312 23$, algoritmul returnează valoarea 1

D. Algoritmul returnează valoare lui $n\%9$

708. Se dă următorul arbore binar. Care dintre următoarele şiruri de noduri reprezintă traversarea acestuia în inordine? ✓ ?



A. $6, 1, 5, 7, 2, 8, 9, 3, 4, 10, 11$

B. $2, 7, 5, 1, 6, 4, 3, 8, 9, 11, 10$

C. $1, 6, 5, 7, 2, 8, 9, 3, 4, 10, 11$

D. $1, 6, 5, 7, 2, 9, 8, 3, 4, 10, 11$

Problemele **709.** și **710.** se referă la următorul scenariu. Oamenii de știință au construit un labirint special pentru șoricelul Algernon, reprezentat sub forma unei matrice de dimensiuni $n \times n$. Fiecare celulă a matricei are dimensiunea 1×1, iar poziția unei celule este identificată prin coordonatele (i, j), unde i este linia, iar j este coloana.

Algernon începe traseul din celula $(1, 1)$ (colțul stânga-sus) și se deplasează conform următoarelor reguli:

(a) Se deplasează spre **dreapta** cât timp nu iese din matrice sau ajunge într-o celulă deja vizitată

(b) La finalul liniei, schimbă direcția și se deplasează **în jos**

(c) Când atinge marginea de jos sau o celulă deja vizitată, continuă spre **stânga**

(d) La finalul liniei, schimbă direcția și se deplasează **în sus**, cât timp nu întâlnește o celulă deja vizitată

Algernon continuă acest traseu în spirală, schimbând direcția după regulile de mai sus, până când ajunge la ultima celulă nevizitată.

Observație: Algernon nu poate trece de două ori prin aceeași celulă.

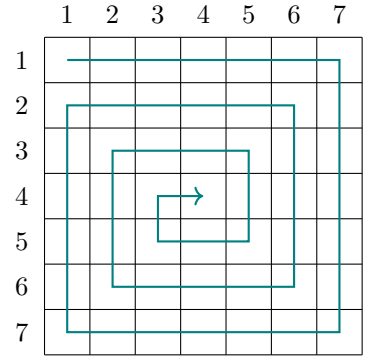


Figura 13.1 Traseul într-un labirint 7×7

709. Care dintre următoarele afirmații este adevărată?: ✓ ?

A. Pentru $n = 7$, ultima celulă vizitată de Algernon este situată la coordonatele $(4, 4)$

B. Pentru $n = 7$, ultima celulă vizitată de Algernon este situată la coordonatele $(3, 3)$

C. Pentru $n = 6$, ultima celulă vizitată de Algernon este situată la coordonatele $(4, 3)$

D. Pentru $n = 6$, ultima celulă vizitată de Algernon este situată la coordonatele $(3, 4)$

710. Pentru $n = 50$, a 1710-a celulă vizitată de Algernon este situată la coordonatele: ✓ ?

A. $(19, 5)$ B. $(12, 14)$ C. $(18, 11)$ D. $(14, 23)$

Problemele **711.** și **712.** se referă la următorii algoritmi: `First(a, n, x)` și `Next(n)` Șirul a este indexat de la 1, având lungimea n, iar x este un număr natural. Pentru fiecare i de la 1 la n, $a[i] = i$.

```
Algorithm NEXT(n)
   first ← 1
   While first < n execute
      first ← first ×2
   EndWhile
   Return first
EndAlgorithm

Algorithm FIND(a, n, x)
   poz ← 0
   i ← NEXT(n)
   While i ≠ 0 execute
      If a[poz + i] ≤ x then
         poz ← poz + i
      EndIf
      i ← i / 2
   EndWhile
   If a[poz] = x then
      Return poz
   EndIf
   Return -1
EndAlgorithm
```

711. Complexitatea algoritmului *find* este: ✓ ?

A. $O(\log n)$ C. $O(n \log n)$

B. $O(n)$ D. $O(n^2)$

712. Pentru $n = 13254, x = 8342$, de câte ori se va schimba valoarea lui *poz* în algoritmul *find* (inițializarea variabilei nu se consideră schimbare)? ✓ ?

A. 5 B. 6 C. 7 D. 8

713. Se dă următorul algoritm $F(n)$, n număr natural. Care dintre următoarele afirmații sunt adevărate? ✓ ?

```
Algorithm F(n)
   If n ≤ 2 then
      Return 1
   Else
      s ← 0
      If (n − 1) MOD 3 ≠ 0 then
         s ← s + F(n − 1)
      EndIf
      If (n − 2) MOD 3 ≠ 0 then
         s ← s + F(n − 2)
      EndIf
      If (n − 3) MOD 3 ≠ 0 then
         s ← s + F(n − 3)
      EndIf
      Return s
   EndIf
EndAlgorithm
```

A. Şirul de valori generate este identic cu cel al lui Fibonacci

B. Complexitatea ca timp de execuţie a algoritmului este $O(2^n)$

C. F(5) = 3

D. F(7) = 8

714. Se dă următorul algoritm *alt(n)*, n număr natural. Care va fi rezultatul afişat în urma apelului $alt(123456)$? ✓ ?

```
Algorithm ALT(n)
   If n ≠ 0 then
      If n MOD 2 = 0 then
         alt(n DIV 10)
         Write (n MOD 10 + 1)
      Else
         Write (n MOD 10 − 1)
         alt(n DIV 10)
      EndIf
   EndIf
EndAlgorithm
```

A. 024357

B. 357420

C. 420357

D. 420753

715. Se consideră un număr natural n şi n matrice $M_{p,q}$ cu p linii şi q coloane, $p, q \in \mathbb{N}^*$. Fie produsul matricelor $P_{p_1,q_n} = M_{p_1,q_1} \times M_{p_2,q_2} \times ... \times M_{p_n,q_n}$, cu $q_1 = p_2, q_2 = p_3, ..., q_{n-1} = p_n$. Prin produs scalar, vom înţelege produsul dintre o linie a primei matrice şi o coloană a celei de a doua matrice. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

A. Pentru $n = 4$ şi matricele $M_{6,5}, M_{5,4}, M_{4,3}$ şi $M_{3,2}$, numărul minim de produse scalare necesare pentru calcularea matricei $P_{6,2}$ este 124

B. Pentru $n = 3$ şi matricele $M_{1,2}, M_{2,3}$ şi $M_{3,2}$, numărul minim de produse scalare necesare pentru calcularea matricei $P_{1,2}$ este 16

C. Numărul total de moduri în care pot fi făcute înmulţirile a $n + 1$ matrice, prin parantetizarea expresiei produsului acestora, fără a schimba ordinea matricelor este $\frac{1}{n+1} C_{2n}^n$

D. Numărul total de moduri în care pot fi făcute înmulţirile a $n + 1$ matrice, prin parantetizarea expresiei produsului acestora, fără a schimba ordinea matricelor este C_{2n+1}^n

Testul 6

716. Fie variabilele `x = 15`, `y = 6` şi `z = 3`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big((x \texttt{ MOD } y + z\Big) \texttt{ DIV } \Big((x-2)+y\Big) + \Big((z+1) \texttt{ MOD } (y+2)\Big)$$

A. 2 B. 3 C. 4 D. 5

717. Se consideră un arbore cu 12 noduri reprezentat prin vectorul de tați `t = (2, 7, 7, 6, 1, 3, 0, 9, 6, 9, 2, 8)`. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Arborele dat are numărul frunzelor egal cu cel al nodurilor interioare

B. Nodurile 4 şi 9 sunt fraţi

C. Există un lanţ între nodurile 10 şi 11 care nu conţine rădăcina arborelui

D. Lungimea lanţului dintre nodurile 5 şi 9 este 6

718. Se consideră algoritmul `Star(a, n, i, j)`, unde n este lungimea unui vector a format din n elemente întregi $(a[1], a[2], ..., a[n])$. Șirul l are, de asemenea, n elemente întregi $(l[1], l[2], ..., l[n])$ ✓ ?

```
Algorithm STAR(a, n, i, j)
   If a[1] MOD 2 = 0 then
      l[1] ← a[1]
   Else
      l[1] ← 0
   EndIf
   For k ← 2 to n execute
      If a[k] MOD 2 = 0 then
         l[k] ← l[k − 1] + a[k]
      Else
         l[k] ← l[k − 1]
      EndIf
   EndFor
   If i = 1 then
      Return l[j]
   Else
      Return l[j] − l[i − 1]
   EndIf
EndAlgorithm
```

Ce face algoritmul?

A. Calculează suma tuturor elementelor din vector

B. Calculează suma elementelor pare din intervalul $[i, j]$

C. Găsește numărul maxim de elemente pare din interval

D. Sortează vectorul crescător

719. Se consideră numerele reale x, a și b, unde $a < b$. Care dintre următoarele expresii logice este adevărată dacă și numai dacă $x \in (a, b)$? ✓ ?

A. $(x > a)$ **ȘI** $(x < b)$
B. **NU** $((x > a)$ **ȘI** $(x < b))$
C. $(x > a)$ **SAU** $(x < b)$
D. **NU** $(x \leq a)$ **ȘI NU** $(x \geq b)$

720. Se consideră algoritmul `s(m, n)`, unde n este dimensiunea unei matrici pătratice m de dimensiune n x n, iar m este o matrice cu elemente întregi ($m[1][1], m[1][2], ...$, $m[n][n]$): ✓ ?

```
Algorithm s(m, n)
   For i ← 1 to n execute
      For j ← i + 1 to n execute
         temp ← m[i][j]
         m[i][j] ← m[j][i]
         m[j][i] ← temp
      EndFor
   EndFor
EndAlgorithm
```

Ce face apelul `s(m, n)`?

A. Transformă matricea prin interschimbarea liniilor cu coloanele, transpunând matricea
B. Inversează elementele matricei pe diagonală principală și secundară
C. Schimbă doar elementele de deasupra diagonalei principale cu cele de sub diagonală
D. Permută aleator elementele matricei fără a păstra vreo proprietate specifică

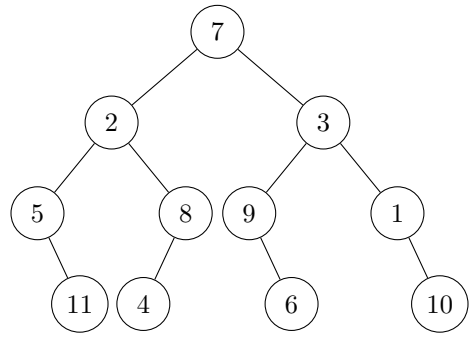
721. Se consideră algoritmul `Compute(A, B, C, n)`, unde A și B sunt două matrice de dimensiune n x n și C este matricea rezultat. ✓ ?

```
Algorithm Compute(A, B, C, n)
   For i ← 1 to n execute
      For j ← 1 to n execute
         C[i][j] ← 0
         For k ← 1 to n execute
            C[i][j] ← C[i][j] + A[i][k] · B[k][j]
         EndFor
      EndFor
   EndFor
EndAlgorithm
```

Ce face algoritmul `Compute`?

A. Sortează elementele din matricea A și B înainte de adunarea lor
B. Calculează suma elementelor din matricea A și matricea B
C. Înmulțește elementele corespunzătoare din A și B fără combinare
D. Înmulțește cele două matrice pătratice folosind metoda standard de înmulțire a matricelor

722. Se dă următorul arbore binar. Care dintre următoarele şiruri de noduri reprezintă traversarea acestuia în preordine? ✓ ?



A. $7, 2, 5, 11, 8, 4, 9, 6, 3, 1, 10$

B. $7, 2, 5, 11, 8, 4, 3, 9, 6, 1, 10$

C. $5, 11, 2, 4, 8, 7, 9, 6, 3, 1, 10$

D. $11, 5, 4, 8, 2, 6, 9, 10, 1, 3, 7$

723. Se consideră algoritmul `Complex(a, n)`, care primește ca parametri un șir a de n elemente și numărul natural n ($1 \leq n \leq 10000$). ✓ ?

```
Algorithm COMPLEX(a, n)
    done ← False
    m ← n
    While not done execute
        done ← True
        p ← m
        For i ← 1, p − 1 execute
            If a[i] > a[i + 1] then
                swap(a[i], a[i + 1])
                done ← False
                m ← i
            EndIf
        EndFor
    EndWhile
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul efectuează exact $n - 1$ comparații pentru un șir a ordonat crescător

B. Algoritmul efectuează același număr de comparații indiferent de valorile șirului a

C. Variabila *done* permite oprirea execuției mai devreme în cazul în care șirul devine sortat

D. Numărul total de comparații este același ca în varianta clasică a algoritmului `BubbleSort`, indiferent de valorile șirului

724. Se dă următorul algoritm skip(start, position, k, n, a), unde *start* este un număr natural cu valoarea inițială egală cu 1, *position* un număr natural cu valoarea inițială egală cu 1, k este un număr natural ($k \leq n \leq 20$), n număr natural ($n \leq 20$), iar a un vector de numere naturale de lungime n. Numerotarea indicilor vectorilor începe de la 1. ✓ ?

```
Algorithm SKIP(start, position, k, n, a)
    If position = k + 1 then
        For i ← 1, k execute
            Write a[i], ' '
        EndFor
        Write newline
    Else
```

```
        For i ← start, n execute
            a[position] ← i
            skip(i + 2, position + 1, k, n, a)
        EndFor
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmaţii sunt adevărate?

A. $\exists i, j \in \mathbb{N}, 1 \leq i < j \leq n, |a[i] - a[j]| = 1$

B. $\exists i, j \in \mathbb{N}, 1 \leq i < j \leq n, a[i] >= a[j]$

C. $\forall i, j \in \mathbb{N}, 1 \leq i < j \leq n, a[i] < a[j]$

D. $\forall i, j \in \mathbb{N}, 1 \leq i < j \leq n, a[i] + 2 = a[j]$

725. Se consideră algoritmul `Sort(a, n)`, care primește ca parametri un șir a de n ✓ ? elemente și numărul natural n ($1 \leq n \leq 10000$).

```
Algorithm SORT(a, n)
    For i ← 1, n − 1 execute
        For j ← 1, n − i execute
            If a[j] > a[j + 1] then
                swap(a[j], a[j + 1])
            EndIf
        EndFor
    EndFor
EndAlgorithm
```

Pentru care dintre următoarele șiruri, operația de interschimbare se efectuează cel mult o dată?

A. $a = [1, 0, 5, 7, 2, 3, 8]$

B. $a = [1, 0, 1, 1, 1]$

C. $a = [1, 2, 3, 4, 5, 6]$

D. $a = [1, 0, 1, 0, 1, 0, 1]$

726. Se consideră algoritmul `CeFace(a, n)`, care primește ca parametri un șir a de n ✓ ? elemente și numărul natural n ($1 \leq n \leq 10000$) și algoritmul `Maybe(x)`, care primește ca parametru un număr natural.

```
Algorithm MAYBE(x)
    If x < 2 then
        Return 0
    EndIf
    For i ← 2, x − 1 execute
        If x MOD i = 0 then
            Return 0
        EndIf
    EndFor
    Return 1
EndAlgorithm
Algorithm CEFACE(a, n)
    p ← 1
    For i ← 1, n execute
        If MAYBE(a[i]) = 1 then
            swap(a[i], a[p])
```

```
            p ← p + 1
        EndIf
    EndFor
    For i ← 1, p − 2 execute
        For j ← 1, p − i − 1 execute
            If a[j] > a[j + 1] then
                swap(a[j], a[j + 1])
            EndIf
        EndFor
    EndFor
    For i ← p, n execute
        key ← a[i]
        j ← i − 1
        While j ≥ p and a[j] < key execute
            a[j + 1] ← a[j]
            j ← j − 1
        EndWhile
        a[j + 1] ← key
    EndFor
EndAlgorithm
```

Ce face algoritmul `CeFace(a, n)`?

A. Algoritmul sortează întregul șir în ordine crescătoare, mutând numerele prime la sfârșitul șirului

B. Algoritmul sortează întregul șir în ordine crescătoare, mutând numerele prime la începutul șirului

C. Algoritmul verifică dacă în componența șirului există doar elemente prime și le sortează crescător

D. Algoritmul separă numerele prime de cele neprime, apoi sortează numerele prime în ordine crescătoare, iar numerele neprime în ordine descrescătoare

727. Se dă următorul algoritm $Z(n, m)$, $n, m \in \mathbb{N}^*, m > 1$. Care dintre următoarele este complexitatea ca timp de execuţie a acestuia? ✓ ?

```
Algorithm Z(n, m)
    If n ≤ 1 then
        Write 1
    Else
        n2 ← n DIV 2
        For i ← 2, m + 1 execute
            Z(n2, m)
        EndFor
    EndIf
EndAlgorithm
```

A. $O(n^m)$

B. $O(m^{\frac{n*(n+1)}{2}})$

C. $O(m^{logn})$

D. $O(log_m n)$

728. Se dă următorul algoritm `Compute(n, s)`, în care n este un număr natural. Vectorul s conține n elemente, fiecare fiind inițializat cu valoarea 0. ✓ ?

```
Algorithm COMPUTE(n, s)
   For i ← 2, n execute
      If s[i] == 0 then
         For j ← i, n, i execute
            s[j] ← s[j] + i
         EndFor
      EndIf
   EndFor
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $n = 10000, s[518] = 32$

B. Pentru $n = 10000, s[518] = 46$

C. Pentru $n = 10000, s[12] = 5$

D. Pentru $n = 12132, s[7129] = 7129$

729. Se dă următorul algoritm $algorithm(n)$, $n \in \mathbb{N}^*$. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

```
Algorithm ALGORITHM(n)
   For i ← 1, n execute
      p[i] ← 0
   EndFor
   p[1] ← 1
   For i ← 2, n execute
      If p[i] = 0 then
         p[i] ← i
         For j ← i * i, n, i execute
            If p[j] = 0 then
               p[j] ← i
            EndIf
         EndFor
      EndIf
   EndFor
EndAlgorithm
```

A. Algoritmul marchează în vectorul p fiecare număr prim cu valoarea acestuia şi numerele compuse cu 0, pentru $n \geq 2$

B. Algoritmul marchează în vectorul p fiecare număr cu valoarea celui mai mic factor prim al acestuia, pentru $n \geq 2$

C. Dacă $n = 25$, atunci $p[10] = 0$

D. Dacă $n = 25$, atunci $p[5] = 5$

730. Andra are o pungă cu n tipuri de buline, fiecare tip de bulină i aflându-se în cantitate a_i. Ea dorește să așeze bulinele pe care le are în formații. O formație reprezintă o formă similară cu un triunghi, in care, pe rândul k se află 2^{k-1} buline. O formație poate să conțină și un singur rând. Andra dorește să formeze formații, folosind toate bulinele pe care le are la dispoziție. ✓ ?

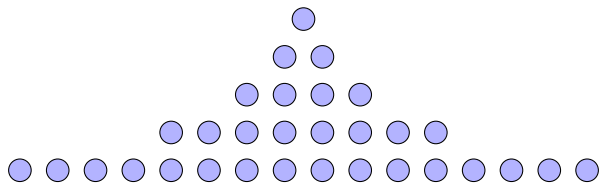


Figura 13.2 Aranjamentul bulinelor într-o formație triunghiulară cu 5 rânduri

Pentru $n = 8, a[] = \{9, 2, 45, 23, 5, 12, 54, 32\}$, care este numărul minim de formații pe care Andra le poate forma folosind toate bulinele pe care le are la dispoziție?

A. 4 B. 8 C. 6 D. 12

731. Se dă următorul algoritm `C(n, k)`, unde n, k sunt numere naturale. ✓ ?

```
Algorithm C(n, k)
   If n = k  or  k = 0 then
      Return 1
   EndIf
   Return C(n - 1, k - 1) + C(n - 1, k)
EndAlgorithm
```

Pentru $n = 10, k = 7$, de câte ori se autoapelează funcția `C`?

A. 120 B. 142 C. 238 D. 207

732. Într-un sistem de coordonate, se află punctul de start $S(0,0)$, și punctul de final $F(a,b)$, unde a, b sunt numere naturale. Dorim să ne deplasăm de la S la F folosind doar mișcări de tipul $(x,y) \to (x+1,y)$ sau $(x,y) \to (x,y+1)$. Două drumuri se consideră distincte dacă există cel puțin un punct în care acestea diferă. Câte drumuri distincte există de la S la F? ✓ ?

A. $a+b$ B. $a \cdot b$ C. C_{a+b}^{a} D. $(a+b)!$

733. Se defineşte şirul Fibonacci $\{F_n\}$, cu $F_1 = 1, F_2 = 1, F_n = F_{n-1} + F_{n-2}, n \geq 3$. Care dintre următoarele formule reprezintă suma $F_1 + F_2 + ... + F_n$? ✓ ?

A. $F_1 + F_2 + ... + F_n = F_{n+1} - 1$
B. $F_1 + F_2 + ... + F_n = F_{n+1}$
C. $F_1 + F_2 + ... + F_n = F_{n+2} - F_{n+1}$
D. $F_1 + F_2 + ... + F_n = F_{n+2} - 1$

Problemele **734.** şi **735.** se referă la următorul algoritm $T(n), n$ număr natural:

```
Algorithm T(n)
   r ← n
   For i ← 2, √n execute
      If n MOD i = 0 then
         r ← r · (i − 1) DIV i
         While n MOD i = 0 execute
            n ← n DIV i
         EndWhile
      EndIf
   EndFor
   If n > 1 then
      r ← r · (n − 1) DIV n
   EndIf
   Return r
EndAlgorithm
```

734. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

A. Algoritmul calculează numărul de factori primi ai lui n
B. Algoritmul calculează numărul de valori mai mici sau egale cu n care sunt prime cu n
C. Formula matematică folosită este $T(n) = n \cdot \prod_{p|n} \left(1 - \frac{1}{p}\right)$
D. $T(n) = n - 1$, pentru n număr prim

735. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

A. $T(10) = 4$ B. $T(7) = 2$ C. $T(1) = 1$ D. $T(25) = 15$

736. Se consideră algoritmul `Transform(i1, i2, j1, j2, cnt)`, unde *cnt* este un parametru transmis prin referință, având valoarea inițială 0. Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Pentru apelul `Transform(1, 2, 1, 2, cnt)`, *cnt* va avea valoarea 5

B. Pentru apelul `Transform(1, 9, 1, 9, cnt)`, *cnt* va avea valoarea 48

C. Pentru apelul `Transform(1, 7, 1, 7, cnt)`, *cnt* va avea valoarea 29

D. Pentru apelul `Transform(1, 13, 1, 13, cnt)`, *cnt* va avea valoarea 59

```
Algorithm TRANSFORM(i1, i2, j1, j2, cnt)
   If i2 − i1 = j2 − j1 then
      cnt ← cnt + 1
      If i1 ≠ i2 or j1 ≠ j2 then
         imid ← (i1 + i2) DIV 2
         jmid ← (j1 + j2) DIV 2
         If i1 = i2 then
            TRANSFORM(i1, i2, j1, jmid, cnt)
            TRANSFORM(i1, i2, jmid + 1, j2, cnt)
         Else If j1 = j2 then
            TRANSFORM(i1, imid, j1, j2, cnt)
            TRANSFORM(imid + 1, i2, j1, j2, cnt)
         Else
            TRANSFORM(i1, imid, j1, jmid, cnt)
            TRANSFORM(i1, imid, jmid + 1, j2, cnt)
            TRANSFORM(imid + 1, i2, j1, jmid, cnt)
            TRANSFORM(imid + 1, i2, jmid + 1, j2, cnt)
         EndIf
      EndIf
   EndIf
EndAlgorithm
```

737. Se consideră algoritmul `Split(s, l, r)`, unde s este un șir indexat de la 1. Fie n lungimea acestui șir. Pentru apelul `Split(s, 1, n)`, care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Algoritmul calculează suma valorilor din șirul s

B. Algoritmul are complexitatea de timp $O(n)$

C. Algoritmul are complexitatea de timp $O(n \log n)$

D. Algoritmul are complexitatea de timp $O(\log n)$

```
Algorithm SPLIT(s, l, r)
   If l > r then
      Return 0
   EndIf
   If l = r then
      Return s[l]
   EndIf
   mid ← ⌊(l + r)/2⌋
   Return SPLIT(s, l, mid) +
SPLIT(s, mid + 1, r)
EndAlgorithm
```

Problemele **738.** și **739.** se referă se referă la următorul scenariu. Într-o încăpere se află n becuri. Inițial, la momentul 0, toate becurile sunt stinse. Fiecare bec i pâlpâie tot la câte $s[i]$ secunde pentru o fracțiune de secundă, după care se stinge automat.

738. Pentru $n = 10$ și $s = [7, 6, 4, 2, 8, 9, 3, 10, 12, 14]$, la ce moment vor lumina toate becurile deodată? ✓ ?

A. 2520 B. 1020 C. 2040 D. 5040

739. Dacă fiecare bec luminează pentru 2 secunde din momentul aprinderii, după care se stinge, pentru $n = 6$ și $s = [7, 9, 4, 2, 8, 6]$, la ce moment vor lumina toate becurile deodată? ✓ ?

A. 3959 B. 108 C. 504 D. 2520

Testul 7

740. Fie variabilele `a = 12`, `b = 5` şi `c = 4`. Ce valoare va avea expresia următoare? ✓ ?

$$\Big(a \texttt{ DIV } (b-1)\Big) * \Big(c + ((a+1) \texttt{ MOD } (b-1))\Big) - \Big((c+1) \texttt{ MOD } (b-1)\Big)$$

A. 14 B. 15 C. 18 D. 20

741. Se consideră algoritmul `Trick1(c, ok, v, n, x, i, j)` și algoritmul `Trick2(v, x, i, j)` unde n este lungimea unui vector v format din n elemente întregi ($v[1], v[2], ..., v[n]$), iar c, ok, i, j și x sunt numere întregi. Fiecare algoritm este apelat de q ori pentru același vector v, dar cu valori diferite ale parametrilor i și j. ✓ ?

```
Algorithm MAGIC(a, n, x)
   construiește c[1...n]
   If a[1] = x then
      c[1] ← 1
   Else
      c[1] ← 0
   EndIf
   For k ← 2,n execute
      If a[k] = x then
         c[k] ← c[k − 1] + 1
      Else
         c[k] ← c[k − 1]
      EndIf
   EndFor
   Return c
EndAlgorithm
Algorithm TRICK1(c, ok, v, n, x, i,
j)
   If ok = 0 then
      c ← Magic(v,n,x)
      ok ← 1
   EndIf
   If i = 1 then
      Return c[j]
   Else
      Return c[j] − c[i − 1]
   EndIf
EndAlgorithm
```

```
Algorithm TRICK2(v, x, i, j)
   c ← 0
   For k ← i,j execute
      If v[k] = x then
         c ← c + 1
      EndIf
   EndFor
   Return c
EndAlgorithm
```

Ce concluzii putem trage în urma rulării celor doi algoritmi pentru q interogări?

A. Ambii algoritmi returnează același rezultat pentru o interogare

B. Algoritmul `Trick2` este mai eficient decât `Trick1` pentru orice valoare a lui q

C. Algoritmul `Trick1` este mai eficient decât `Trick2` pentru orice valoare a lui $q > 1$

D. `Trick1` și `Trick2` au aceeași complexitate și performanță, indiferent de numărul de interogări

742. Se consideră algoritmul `X(m, n)`, unde n este dimensiunea unei matrici pătratice m de dimensiune n x n, iar m este o matrice cu elemente întregi $(m[1][1], m[1][2], ..., m[n][n])$: ✓ ?

```
Algorithm X(m, n)
   For i ← 1,n execute
      For j ← 1,⌊n/2⌋ execute
         temp ← m[i][j]
         m[i][j] ← m[i][n − j + 1]
         m[i][n − j + 1] ← temp
      EndFor
   EndFor
EndAlgorithm
```

Ce face algoritmul `X`?

A. Inversează liniile matricei de sus în jos

B. Oglindește matricea față de axa verticală, schimbând coloanele între ele

C. Oglindește matricea față de diagonala principală

D. Rotunjește toate elementele matricei la cel mai apropiat multiplu de 10

743. Fie subalgoritmul `Fix(m, n, p)` unde n și p sunt dimensiunile unei matrice pătratice m de dimensiune $n \times p$, iar m este o matrice cu elemente întregi $(m[1][1], m[1][2], ..., m[n][p])$. ✓ ?

```
Algorithm Fix(m, n, p)
   s ← 0
   For i ← 1,n execute
      For j ← 1,p execute
         temp ← m[i][j]
         s ← s + (temp MOD 10)
      EndFor
   EndForReturn s
EndAlgorithm
```

```
Algorithm FixIT(m, n, p, i, j)
   If i ≤ n AND j ≤ p then
      Return _____
   EndIf
EndAlgorithm
```

Alegeți varianta care completează corect spațiul subliniat din subalgoritmul de mai jos astfel încât cei doi subalgoritmi să returneze mereu aceeași valoare, luând în considerare că subalgoritmul `FixIT(m, n, p, i, j)` se va apela cu valorile parametrilor i și j egale cu 1.

A. $(m[i][j]$ `MOD` $10) + FixIT(m, n, p, i, j$ `DIV` $1) + FixIT(m, n, p, i + 1, 1)$

B. $(m[i][j]$ `DIV` $10) + FixIT(m, n, p, i, j + 1) + FixIT(m, n, p, i + 1, 1)$

C. $(m[i][j]$ `MOD` $10) + FixIT(m, n, p, i, j + 1) + FixIT(m, n, p, i + 1, 1)$

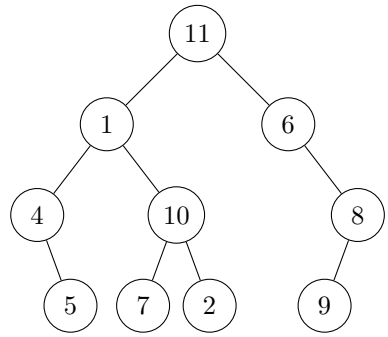
D. $(m[i][j]$ `DIV` $10) + FixIT(m, n, p, i, j + 1) + FixIT(m, n, p, i - 1, 1)$

744. Fie variabilele `a = 3`, `b = 5`, `c = 2` și `d = 8`. Ce valoare va avea expresia următoare? ✓ ?

$$(a > b\,?\,b : (c < d\,?\,d - c : a + b)) + (d < a\,?b - c : (c > a\,?\,c : a + d))$$

A. 15 B. 14 C. 17 D. 16

745. Se dă următorul arbore binar. Care dintre următoarele şiruri de noduri reprezintă traversarea acestuia în postordine? ✓ ?



A. $4, 5, 1, 7, 10, 2, 11, 6, 8, 9$

B. $11, 1, 4, 5, 10, 7, 2, 6, 8, 9$

C. $5, 4, 7, 2, 10, 1, 6, 8, 9, 11$

D. $5, 4, 7, 2, 10, 1, 9, 8, 6, 11$

746. Se dă următorul algoritm $algorithm(n), n \in \mathbb{N}^*$. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

```
Algorithm ALGORITHM(n)
   For i ← 1, n execute
      d[i] ← 0
   EndFor
   For i ← 1, n execute
      For j ← i, n, i execute
         d[j] ← d[j] + 1
      EndFor
   EndFor
EndAlgorithm
```

A. Algoritmul calculează numărul de divizori proprii ai numerelor mai mici sau egale decât n

B. Algoritmul calculează numărul de divizori ai numerelor mai mici sau egale decât n

C. Dacă $n = 25$, atunci $d[10] = 1$

D. Dacă $n = 25$, atunci $d[24] = 8$

747. Se dă următorul algoritm $Y(n, m), n, m \in \mathbb{N}^*, m > 1$. Care dintre următoarele este complexitatea ca timp de execuţie a acestuia? ✓ ?

```
Algorithm Y(n, m)
   If n ≤ 1 then
      Return m
   Else
      n ← n DIV m
      For i ← 1, m execute
         Y(n, m)
      EndFor
   EndIf
EndAlgorithm
```

A. $O(n^m)$

B. $O(m^{log_m n})$

C. $O(n)$

D. $O(nm)$

748. Se dă următorul algoritm `evenp(k, n, a, f)`, unde k este un număr natural ($k \leq 20$) cu valoarea iniţială egală cu 1, n număr natural ($n \leq 20$), iar a şi f, 2 vectori de numere naturale de lungime n, f având elementele iniţializate cu 0. Numerotarea indicilor vectorilor începe de la 1. ✓ ?

```
Algorithm EVENP(k, n, a, f)
   If k = n + 1 then
      For i ← 1, n execute
         Write a[i], ' '
      EndFor
      Write newline
   Else
      For i ← 1, n execute
```

```
            If f[i] = 0 AND (k + i) MOD 2 = 1 then
                a[k] ← i
                f[i] ← 1
                evenp(k + 1, n, a, f)
                f[i] ← 0
            EndIf
        EndFor
    EndIf
EndAlgorithm
```

Care dintre următoarele afirmaţii sunt adevărate?

A. Pentru orice număr natural n par, primul element al fiecărei soluţii va fi mereu un număr impar

B. Pentru orice număr natural n impar, nu există nicio soluţie

C. Algoritmul generează permutările şirului format din numere naturale de la 1 la n, în care paritatea unei valori este diferită de ceea a poziţiei pe care se află

D. Pentru orice număr natural n impar, primul element al fiecărei soluţii va fi mereu un număr impar

749. Se consideră arborele binar reprezentat prin vectorul de taţi `t = {4, 1, 1, 6, 4, 0, 8, 6, 8, 9, 10}`. Care dintre următoarele afirmaţii sunt adevărate? ✓ ?

A. Rădăcina arborelui este nodul marcat cu valoarea 6

B. Arborele dat are numărul de frunze egal cu cel de noduri interioare

C. Toate frunzele arborelui sunt marcate cu numere prime

D. Înălţimea arborelui este 3

750. Se consideră un poligon regulat cu 12 laturi, reprezentând traseul zilnic al curierului Jimmy pentru livrarea coletelor. Pornind dintr-un vârf fixat ca *depozit* (originea), Jimmy trebuie să parcurgă, în ordine, adresele dintr-un șir dat. Mai exact, dacă valorile a_1, a_2, a_3, ..., a_k determină distanțele (în număr de laturi) dintre adrese, atunci: ✓ ?

$$\begin{aligned}
\text{Prima adresă:} &\quad a_1 \text{ laturi de origine,}\\
\text{A doua adresă:} &\quad a_1 + a_2 \text{ laturi de origine,}\\
\text{A treia adresă:} &\quad a_1 + a_2 + a_3 \text{ laturi de origine,}\\
&\quad \vdots\\
\text{A } k\text{-a adresă:} &\quad a_1 + a_2 + \cdots + a_k \text{ laturi de origine.}
\end{aligned}$$

După fiecare deplasare către o adresă, Jimmy se întoarce la depozit.

Mod de calcul al energiei consumate:

(a) **Deplasarea de la depozit către o adresă.** Să considerăm o deplasare de n laturi parcurse în sensul acelor de ceasornic. Energia consumată este determinată prin "scrierea" unui octet (8 biți, numerotați de la 7 la 0) după cum urmează:

- **Dacă** $n \leq 8$: se setează n biți consecutivi cu valoarea 1, *începând de la stânga* (adică de la poziția 7). Valoarea în baza 10 a octetului obținut reprezintă energia consumată.

Exemplu: pentru $n = 3$, octetul este

Poziție	7	6	5	4	3	2	1	0
Bit	1	1	1	0	0	0	0	0

care corespunde valorii 224 în baza 10.

- **Dacă** $n > 8$: se procedează astfel:
  a) Pentru primele 8 laturi se setează toți cei 8 biți, obținându-se octetul 11111111, ce corespunde valorii 255.
  b) Imediat după parcurgerea acestor 8 laturi se efectuează o *pauză* ce consumă 128 unități de energie.
  c) Pentru cele $n-8$ laturi suplimentare, se setează câte un bit de 1 *începând de la dreapta* (poziția 0), iar energia suplimentară este egală cu valoarea obținută, adică:
  $$2^{n-8} - 1.$$

  Astfel, energia consumată pentru deplasarea spre adresă, când $n > 8$, este:
  $$E_{\rightarrow} = 255 + 128 + \left(2^{n-8} - 1\right).$$

(b) **Întoarcerea la depozit.** Pentru revenirea la depozit se alege drumul cel mai scurt (fie în sensul acelor de ceasornic, fie în sens invers). Dacă drumul ales presupune parcurgerea a n laturi, se "scrie" un octet în care se setează n biți consecutivi cu valoarea 1, *începând de la dreapta* (de la poziția 0). Valoarea în baza 10 a acestui octet reprezintă energia consumată la întoarcere.

Exemplu: pentru $n = 3$, octetul este

Poziție	7	6	5	4	3	2	1	0
Bit	0	0	0	0	0	1	1	1

ceea ce corespunde valorii 7 în baza 10.

Câtă energie va consuma azi (valoare în baza 10) Jimmy dacă are pachete de livrat la adresele $1, 2, 3$ și 9?

A. 1076 B. 810 C. 1183 D. 1192

751. Se dă următorul algoritm `C(n, k)`, unde n, k sunt numere naturale. ✓ ?

```
Algorithm C(n, k)
   If n = k  or  k = 0 then
      Return 1
   EndIf
   Return C(n - 1, k - 1) + C(n - 1, k)
EndAlgorithm
```

Pentru $n = 10, k = 7$, care este valoarea returnată de către funcția `C`?

A. 120 B. 142 C. 238 D. 207

752. Fie următorul algoritm, având ca parametru numărul natural nenul n și care returnează un număr natural. ✓ ?

```
Algorithm G(n)
   k ← n
   While k > 1 execute
      a ← 1
      While a ≤ n^3 execute
         a ← 3 * a
      EndWhile
      k ← k DIV 7
   EndWhile
   Return k
EndAlgorithm
```

În care dintre următoarele clase de complexitate se încadrează complexitatea timp a algoritmului?

A. $O(\log_3 n^3)$ C. $O(\log_7^2 n)$

B. $O(\log_3^2 n^2)$ D. $O(\log_3 \log_3 n)$

753. Se consideră algoritmul `New(n, k)`, unde n și k sunt numere naturale ($1 \leq n, k \leq$ 1000000). ✓ ?

```
Algorithm Zone(n)
   If n < 2 then
      Return 0
   EndIf
   For i ← 2, n − 1 execute
      If n MOD i = 0 then
         Return 0
      EndIf
   EndFor
   Return 1
EndAlgorithm

Algorithm New(n, k)
   s ← 0
   p ← 1
   While n > 99 și k > 0 execute
      If Zone(n MOD 10) then
         s ← s+
           p * (n DIV 10 MOD 100)
         p ← p * 100
      Else
         k ← k − 1
      EndIf
      n ← n DIV 10
   EndWhile
   Return s
EndAlgorithm
```

Care dintre următoarele perechi de apeluri returnează valori identice?

A. `New(32345, 3)` și `New(321458, 7)`

B. `New(321458, 1)` și `New(318468, 7)`

C. `New(2314, 3)` și `New(321358, 3)`

D. `New(32145, 3)` și `New(321458, 7)`

754. Se dă următorul algoritm *M(n)*, n număr natural. Care dintre următoarele afirmaţii sunt adevărate?

```
Algorithm M(n)
   If n > 100 then
      Return n − 10
   Else
      Return M(M(n + 11))
   EndIf
EndAlgorithm
```

A. Algoritmul returnează aceeaşi valoare pentru orice număr natural n, $n \leq 101$.

B. Algoritmul returnează n, pentru orice număr natural n, $n > 101$.

C. $M(523) = 513$

D. $M(102) = 102$

755. Se dă următorul algoritm *F(n)*, unde n este un număr natural. Care dintre următoarele afirmații sunt adevărate?

```
Algorithm F(n)
   If n = 0 then
      Return 1
   Else
      Return F(F(n − 1) − 1) · n
   EndIf
EndAlgorithm
```

A. $F(4) = 8$

B. $F(5) = 13$

C. $F(2) = 3$

D. Nu este posibilă calcularea valorilor pentru $F(n), n \geq 5$

756. Fie șirul $P = \{7, 4, 6, 5, 10, 3, 8, 1, 2, 9\}$. Care dintre următoarele afirmații sunt adevărate pentru apelul `Transform(P, 10)`?

```
Algorithm Transform(p, n)
   cnt ← 0
   ok ← True
   For i ← 1 to n execute
      q[i] ← p[i]
   EndFor
   While ok execute
      ok ← false
      cnt ← cnt + 1
      For i ← 1 to n execute
         q[i] ← p[q[i]]
      EndFor
      For i ← 1 to n execute
         If q[i] ≠ p[i] then
            ok ← true
         EndIf
      EndFor
   EndWhile
   Return cnt
EndAlgorithm
```

A. Algoritmul returnează valoarea 30

B. După bucla în care *cnt* devine 8, $q = \{7, 2, 3, 4, 5, 6, 8, 1, 9, 10\}$

C. După bucla în care *cnt* devine 17, $q = \{1, 10, 3, 9, 2, 6, 7, 8, 5, 4\}$

D. După bucla în care *cnt* devine 31, $q = \{8, 5, 3, 10, 9, 6, 1, 7, 4\}$

757. Se consideră o suprafață de dimensiune $n \times m$, unde $n, m \in \mathbb{N}$. Această suprafață este acoperită cu parchet, fiecare placă având dimensiunea $d \times d$, unde $d \in \mathbb{N}$. Dacă o placă nu încape, atunci aceasta va fi tăiată la dimensiunea necesară pentru a se potrivi. O placă care a fost tăiată nu se poate refolosi pentru a acoperi o altă porțiune a suprafeței. Câte plăci sunt necesare? ✓ ?

A. $\left[\frac{n \cdot m}{d^2}\right]$

B. $\left[\frac{n}{d}\right] \cdot \left[\frac{m}{d}\right]$

C. $\left[\frac{n+d-1}{d}\right] \cdot \left[\frac{m+d-1}{d}\right]$

D. $\left[\frac{n}{d}+1\right] \cdot \left[\frac{m}{d}+1\right]$

758. Se consideră o urnă, care conţine bile albe şi bile negre. Folosind următoarele 2 reguli, câte 2 bile sunt extrase pe rând până în momentul în care în interiorul urnei se află o singură bilă: ✓ ?

(a) dacă bilele extrase sunt de aceeaşi culoare, se introduce o bilă de culoare neagră

(b) dacă bilele extrase sunt de culori diferite, se introduce o bilă de culoare albă.

Care dintre următoarele afirmaţii sunt adevărate?

A. Oricare ar fi distribuția inițială de bile albe și negre, nu putem ști cu certitudine culoarea ultimei bile

B. Dacă la început se află 3 bile albe şi 3 bile negre în urnă, atunci ultima bilă va fi albă

C. Dacă numărul iniţial de bile albe este par, atunci ultima bilă va fi mereu neagră

D. Dacă numărul iniţial de bile albe este impar, atunci ultima bilă va fi mereu neagră

759. Într-un oraș se organizează un concurs de ghicit un număr, aflat în intervalul $[1, n]$. În cadrul concursului, participanții pot forma echipe, pentru a ghici numărul. Inițial, fiecare participant dintr-o echipă completează un bilețel pe care scrie oricâte numere dorește, iar dacă numărul câștigător se află pe bilețelul participantului, atunci participantul va fi informat că a selectat numărul câștigător, altfel, va fi informat că nu a ghicit numărul. Fiecare participant poate să completeze un singur bilețel. La final, echipa trebuie să ghicească numărul câștigător. Care este numărul minim de participanți pe care o echipă trebuie să îi aibă, pentru a fi sigură că va ghici numărul câștigător, indiferent care ar fi acesta? ✓ ?

A. n

B. $n-1$

C. $[\log_2 (n-1)] + 1$

D. $[\log_2 n]$

760. Un lac înconjurat de două maluri are n stânci, fiecare dintre ele aflându-se într-o linie și la distanța 1 față de cealaltă, sau de mal. Broscuța Broski dorește să sară de pe o stâncă pe alta, până ajunge de la un mal la celălalt. Inițial, Broski sare de pe mal pe stânca 1, efectuând un salt de o unitate. La următoarele sărituri, Broski poate fie să sară o unitate, fie dublul distanței pe care a sărit-o anterior. Broski trebuie neapărat să ajungă pe celălalt mal, neputând să depășească printr-un salt capătul celălalt (spre exemplu, dacă mai are 7 stânci, nu va putea efectua un salt de lungime 16). Pentru 120 de stânci, care este numărul minim de sărituri pe care Broski trebuie să le facă pentru a ajunge pe celălalt mal? ✓ ?

A. 7 B. 8 C. 22 D. 21

761. Se consideră algoritmul `Split(s, l, r)`, unde s este un șir indexat de la 1, care conține doar numere pozitive. Fie n lungimea acestui șir. Pentru apelul `Split(s, 1, n)`, care dintre următoarele afirmații sunt adevărate?

A. Algoritmul determină valoarea maximă din șirul s

B. Algoritmul are complexitatea timp $O(n)$

C. Algoritmul are complexitatea timp $O(n \log n)$

D. Algoritmul are complexitatea timp $O(\log n)$

```
Algorithm SPLIT(s, l, r)
   If l > r then
      Return -1
   EndIf
   If l = r then
      Return s[l]
   EndIf
   mid ← (l + r) DIV 2
   left ← SPLIT(s, l, mid)
   right ← SPLIT(s, mid + 1, r)
   If left > right then
      Return left
   Else
      Return right
   EndIf
EndAlgorithm
```

✓ ?

762. Într-un sistem solar, n planete se rotesc în jurul unei stele, în sens trigonometric. Fiecare planetă are orbita în formă de cerc, având centrul în jurul stelei. Planeta i are perioada orbitală de $z[i]$ zile. În ziua 0, toate planetele sunt aliniate pe aceeași dreaptă. ✓ ?

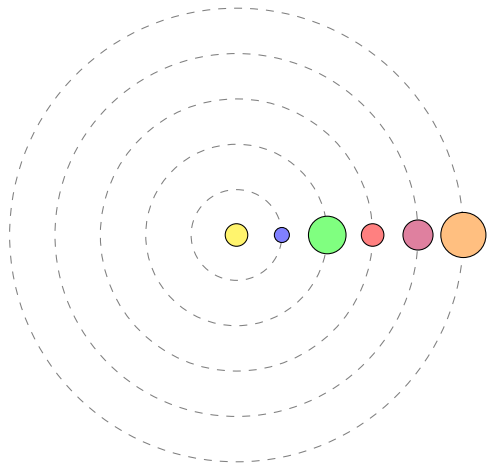


Figura: Planete orbitând în jurul stelei, aliniate în ziua 0.

Figura 13.3 Planetele sunt aliniate pe aceeași dreaptă în ziua 0.

Pentru $n = 8, z = \{3, 4, 6, 12, 8, 5, 20, 9\}$, după câte zile se vor afla planetele din nou aliniate în poziția inițială?

A. 9 B. 720 C. 360 D. 180

763. Se consideră algoritmii `NDiv(n)` și `Compute(m)`, unde m este un număr natural. Pentru $m = 153$, care dintre afirmații sunt adevărate?

A. 25 face parte din rezultatul funcției `Compute`.

B. 144 face parte din rezultatul funcției `Compute`.

C. Algoritmul `Compute` returnează toate numere pătrate perfecte din intervalul $[2, m]$.

D. 169 face parte din rezultatul funcției `Compute`.

✓ ?

```
Algorithm NDiv(n)
   cnt ← 0
   For i ← 1, √n execute
      If n MOD i = 0 then
         cnt ← cnt +2
         If i × i = n then
            cnt ← cnt −1
         EndIf
      EndIf
   EndFor
   Return cnt
EndAlgorithm

Algorithm Compute(m)
   idx ← 1
   For i ← 2,m execute
      If NDiv(i) = 3 then
         res[idx] ← i
         idx ← idx + 1
      EndIf
   EndFor
   Return res
EndAlgorithm
```

Testul 8

764. Se consideră algoritmul `f(x, n)`, unde x și n sunt două numere naturale ($x, n \leq 10^4$) ✓ ?

```
Algorithm F(x, n)
   If n = 0 then
      Return 1
   EndIf
   m ← n DIV 2
   p ← f(x, m)
   If n MOD 2 = 0 then
      Return p * p
   EndIf
   Return x * p * p
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Apelul `f(2, 5)` va returna valoarea 10

B. Apelul `f(3, 4)` va returna valoarea 81

C. Algoritmul calculează valoarea expresiei x^n

D. Algoritmul are complexitatea timp $O(\log_2 n)$

765. Se consideră o linie dreaptă cu orașe poziționate la coordonate $x_1 < x_2 < \cdots < x_n$. Un satelit poate acoperi un interval continuu de lungime fixă L. Care strategie minimizează numărul de sateliți necesari pentru a acoperi toate orașele? ✓ ?

A. Se plasează sateliții începând de la est, acoperind orașul cel mai vestic neatins

B. Se plasează sateliții astfel încât să acopere intervalul maxim posibil începând de la primul oraș neatins

C. Se ignoră pozițiile orașelor și se plasează sateliții la fiecare L kilometri

D. Se acoperă mai întâi intervalele cu cele mai multe orașe

766. Se dă subalgoritmul `X(n)`, unde $n \geq 1$. ✓ ?

```
Algorithm X(n)
   Return Y(1, [], n)
EndAlgorithm

Algorithm Y(k, d, m)
   If k > m then
      Return 1
   EndIf
   t ← 0
   For i ← 1, m execute
      found ← 0
      For j ← 1, len(d) execute
         If d[j] = i then
            found ← 1
         EndIf
      EndFor
      If found = 0  AND  i MOD k = 0 then
         t ← t + Y(k + 1, append(d, i), m)
      EndIf
   EndFor
   Return t
EndAlgorithm
```

```
Algorithm LEN(l)
   count ← 0
   For x ∈ l execute
      count ← count + 1
   EndFor
   Return count
EndAlgorithm

Algorithm APPEND(l, x)
   r ← []
   For i ← 1, LEN(l) execute
      r[i] ← l[i]
   EndFor
   r[LEN(l) + 1] ← x
   Return r
EndAlgorithm
```

Ce calculează acest subalgoritm?

A. Numărul de permutări ale mulțimii $\{1, 2, \ldots, n\}$ unde elementul de pe poziția i este divizibil cu i

B. Numărul de permutări unde poziția i este divizibilă cu elementul aflat pe ea

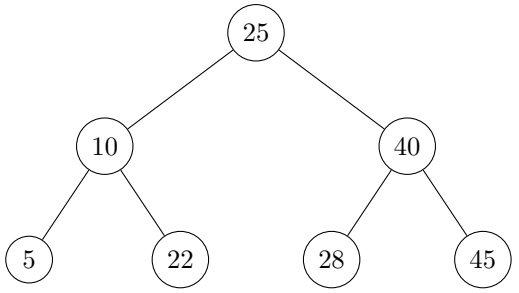
C. Numărul de submulțimi nevide cu elemente divizibile cu indicele lor

D. Numărul de partiții ale lui n în sume de numere distincte

767. Care este înălțimea unui arbore binar complet cu 100 de noduri? ✓ ?

A. 6 B. 7 C. 8 D. 9

768. În arborele binar de căutare alăturat, care este succesorul în traversarea inordine al nodului cu valoarea 28? ✓ ?



A. 22

B. 40

C. 45

D. 25

769. Se dă subalgoritmul `f(N)`, unde $N \geq 1$. ✓ ?

```
Algorithm F(N)
   For i ← −N, N execute
      For j ← −N, N execute
         If max(|i|, |j|) = N then
            Write '*'
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $n = 3$ subalgoritmul afișează 24 steluțe

B. Pentru $n = 3$ subalgoritmul afișează 36 steluțe

C. Pentru ca subalgoritmul să afișeze 5784 de steluțe, valoarea lui n trebuie să fie 723

D. Pentru ca subalgoritmul să afișeze 2116 de steluțe, valoarea lui n trebuie să fie 23

770. Se consideră expresia logică $(A \oplus B) \mod 5$, unde $\oplus$ reprezintă operația XOR pe biți (exemplu: $3 \oplus 5 = 6$, deoarece $011_2 \oplus 101_2 = 110_2$). A este numărul de biți setați (1) în reprezentarea binară a lui 14. B este numărul de zerouri finale în reprezentarea binară a lui 28. ✓ ?

Care este valoarea expresiei?

A. 1 B. 3 C. 4 D. 0

771. Se dă subalgoritmul `X(N)`, unde $N \geq 1$, N număr natural. ✓ ?

```
Algorithm X(N)
    s ← 0
    m ← 0
    t ← N
    While t > 0 execute
        c ← t MOD 10
        t ← t DIV 10
        s ← s + c
        k ← 1
        tmp ← s
        While tmp ≥ 10 execute
            tmp ← tmp DIV 10
            k ← k + 1
        EndWhile
        m ← m × 10^k + s
    EndWhile
    total ← 0
    While m > 0 execute
        total ← total + (m MOD 10)
        m ← m DIV 10
    EndWhile
    Return total
EndAlgorithm
```

Ce calculează acest subalgoritm pentru un N dat?

A. Suma cifrelor lui N

B. Suma sumelor parțiale ale cifrelor lui N

C. Suma cifrelor numărului format din sumele parțiale ale cifrelor lui N

D. Produsul cifrelor lui N

772. Care este complexitatea de timp a următorului algoritm? ✓ ?

```
Algorithm Z(n)
    If n ≤ 1 then
        Return
    EndIf
    s ← 0
    For i ← 1, log_2 n execute
        For k ← 1, n execute
            s ← k · 2
        EndFor
        For j ← 1, log_2 n execute
            s ← s + s MOD 10
        EndFor
    EndFor
    Z(n/2)
    Z(n/2)
EndAlgorithm
```

A. $O(n(\log n)^2)$

B. $O(n \log(n+1) \log n)$

C. $O(n^2)$

D. $O(n(\log n + \log\log n))$

773. Se dă subalgoritmul `F(M, T)`, unde T este un număr întreg și M este o matrice $n \times n$ cu următoarele proprietăți: Fiecare linie este sortată în ordine strict crescătoare; Primul element al liniei $i+1$ este strict mai mare decât ultimul element al liniei i. ✓ ?

```
Algorithm F(M, T)
    a ← 0
    lb ← 1
    rb ← n * n
    While lb ≤ rb execute
        mid ← lb + ⌊(rb − lb)/2⌋
        row ← mid DIV n
        col ← mid MOD n
        If M[row][col] = T then
            a ← lb * rb
        Else If M[row][col] < T then
            lb ← mid + 1
        Else
            rb ← mid − 1
        EndIf
    EndWhile
    Return a > 0
EndAlgorithm
```

Ce determină acest subalgoritm?

A. Verifică existența lui T în matrice

B. Găsește prima apariție a lui T în matrice

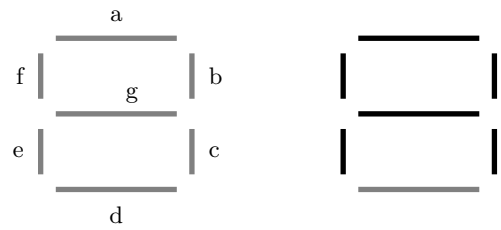
C. Calculează numărul de apariții ale lui T

D. Verifică dacă există două numere a și b astfel încât $a \times b = T$

774. S-a notat cu x_a numărul x în baza de numerație a. Valoarea în baza 10 a calculului $BD_{16} + 215_6 + 4_5 + 21_3$ este: ✓ ?

A. 280 B. 281 C. 282 D. 283

775. Se consideră un afișaj digital cu 7 segmente. Cu ajutorul acestuia, se generează un cod prin transformarea fiecărui caracter dintr-un cuvânt într-un număr întreg. De exemplu, cuvântul **acasa** a fost transformat în 119—78—119—109—119. Care dintre următoarele variante de răspuns corespund transformării cuvântului **concurs**? ✓ ?



Display gol Litera A

Figura 13.4 Fiecare literă (a,b,c,d,e,f,g) corespunde cu un bit în reprezentarea în baza 2 (de la stânga la dreapta, în ordinea dată). A conține **a,b,c,e,f,g**, astfel transformarea va genera: $1110111_2 = 119_{10}$

A. 78—126—118—78—62—119—109
B. 78—126—118—78—62—127—119
C. 78—126—118—78—62—127—109
D. 78—126—118—78—62—119—119

776. Se dă subalgoritmul `ceFace(n, k)`, unde $n, k \geq 0$. ✓ ?

```
Algorithm ceFace(n, k)
    If n = 0 then
```

```
      Return 1  If  k = 0  Else  0
   Else If k < 0 then
      Return 0
   Else
      Return ceFace(n − 1, k) + ceFace(n, k − n)
   EndIf
EndAlgorithm
```

Ce calculează acest subalgoritm?

A. Numărul de moduri de a scrie k ca sumă de cel mult n numere naturale

B. Numărul de partiții ale lui k în exact n termeni diferiți

C. Numărul de moduri de a descompune k într-o sumă de termeni din mulțimea $\{1, 2, \ldots, n\}$

D. Numărul de soluții pentru ecuația $x_1 + x_2 + \cdots + x_n = k$, unde $x_i \leq i$

777. Care ar putea fi elementele unui vector astfel încât, aplicând metoda de căutare binară modificată pentru valoarea 42, aceasta să fie comparată succesiv cu valorile 11, 53, 48? ✓ ?

A. $[48, 11, 53]$ B. $[53, 11, 48]$ C. $[11, 48, 53]$ D. $[11, 53, 48]$

778. Se dă subalgoritmul `Q(n)`, unde $n \geq 0$. ✓ ?

```
Algorithm Q(n)
   If n = 0 then
      Return 1
   Else
      s ← 0
      For k ← 1, n execute
         s ← s + Q(k − 1) ∗ Q(n − k)
      EndFor
      Return s
   EndIf
EndAlgorithm
```

Ce calculează acest subalgoritm?

A. Numărul de arbori binari de căutare cu n noduri

B. Numărul de modalități de a paranteza $n + 1$ operatori

C. Numărul de posibilități de a desena n coarde care nu se intersectează, pe un cerc care are $2 * n$ puncte

D. Numărul de partiții ale lui n în sume de numere prime

779. Care este lungimea maximă posibilă a unui subșir strict crescător într-un vector format doar din valori 0 și 1? ✓ ?

A. 1

B. 2

C. Numărul de elemente 1 din vector

D. Numărul de elemente 0 din vector

780. Se dă o matrice pătratică $A_{n \times n}$ ($n \geq 3$, impar) și subalgoritmul `Z(A, n)`, unde A este o matrice. ✓ ?

```
Algorithm Z(A, n)
   c ← (n + 1)DIV2
   For i ← 1, n execute
```

```
        For j ← 1, n execute
            If (j < c AND i + j < n + 1) OR (j > c AND i + j > n + 1) then
                Write A[i][j]
            EndIf
        EndFor
    EndFor
EndAlgorithm
```

Ce ordine vor avea elementele afișate, pentru $n = 5$?

A. Elementele de sub diagonala secundară și deasupra diagonalei principale, parcurse pe linii

B. Elementele din stânga centrului parcurse pe coloane de sus în jos, apoi cele din dreapta pe linii de la dreapta la stânga

C. Toate elementele din stânga diagonalei principale și dreapta diagonalei secundare, parcurse pe linii

D. Elementele din stânga centrului pe coloane, apoi cele din dreapta centrului pe linii, ambele de sus în jos

781. Se consideră algoritmul `Miracol(x,n)` unde x este un șir de cel mult 10^9 numere întregi, iar n este lungimea șirului x. ✓ ?

```
1: Algorithm MIRACOL(x, n)
2:     s ← False
3:     Do
4:         s ← True
5:         For i ← 1, n execute
6:             If x[i] < x[i − 1] then
7:                 s ← False
8:                 break
9:             EndIf
10:        EndFor
11:    While NOT (s)
12: EndAlgorithm
13:
14: Afișează Ok
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $x = [3, 2, 4, 1]$ și $n = 4$, la finalul algoritmului, $x = [4, 3, 2, 1]$

B. Pentru $x = [2, 0, 1, 5, 7]$ și $n = 5$, la finalul algoritmului, $x = [0, 1, 2, 5, 7]$

C. Algoritmul sortează crescător șirul x și afișează întotdeauna mesajul `Ok`

D. Instrucțiunea de la linia 14 nu este executată niciodată, indiferent de x

782. Se dă algoritmul `f(a, b)` unde a și b sunt două numere naturale ($1 \leq a, b \leq 10, b \leq a$). Cu ce instrucțiune trebuie completată zona punctată pentru ca algoritmul să calculeze corect numărul de aranjamente de a luate câte b? ✓ ?

```
Algorithm F(a, b)
    If b = 0 then
        Return 1
    EndIf
    If b > a then
        Return 0
    EndIf
    Return f(a − 1, b) + ...
EndAlgorithm
```

A. $a * f(a-1, b-1)$

B. $a * f(a-1, b)$

C. $b * f(a-1, b-1)$

D. $b * f(a-1, b)$

783. Se consideră expresia $E(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + a_4 \cdot x^4 + a_5 \cdot x^5$, unde $a_0, a_1, a_2, a_3, a_4, a_5$ și x sunt numere reale nenule. Numărul minim de înmulțiri necesare pentru a calcula valoarea expresiei $E(x)$ este: ✓ ?

A. 5 B. 6 C. 20 D. 22

Problemele **784.** și **785.** se referă la algoritmul `g(x, n)`, unde x și x sunt numere naturale, $x > 0$:

```
1: Algorithm g(x, n)
2:     If n = 0 then
3:         Return 1
4:     EndIf
5:     m ← n DIV 3
6:     p ← g(x, m)
7:     If n mod 3 = 0 then
8:         Return p · p · p
9:     EndIf
10:    If n mod 3 = 1 then
11:        Return x · p · p · p
12:    EndIf
13:    Return x · x · p · p · p
14: EndAlgorithm
```

784. Precizați care dintre următoarele afirmații sunt adevărate: ✓ ?

A. Algoritmul returnează x^n efectuând aproximativ n apeluri recursive

B. Algoritmul returnează x^n efectuând aproximativ $\log_3 n$ apeluri recursive

C. Algoritmul returnează x^n doar dacă n este multiplu de 3

D. Algoritmul returnează x^{n+1} atunci când $n \bmod 3 = 2$

785. Considerăm că linia 13 este înlocuită cu: ✓ ?

```
13.  return x + x * p * p * p
```

Precizați care dintre următoarele afirmații sunt adevărate pentru noua versiune a algoritmului:

A. Algoritmul nu mai returnează x^n

B. Algoritmul returnează tot x^n, dar cu o creștere constantă

C. Algoritmul efectuează aproximativ n^2 apeluri recursive

D. Algoritmul returnează o valoare mai mare decât x^n (pentru $n > 0$)

786. Folosind metoda Backtracking, se generează, pe rând, toate numerele pare de 4 cifre, cifre care iau valori din șirul $[1, 6, 2, 7, 4, 5]$, în ordinea dată. Știind că primele trei soluții sunt $1116, 1112, 1114$, care va fi cea de-a 8-a soluție generată? ✓ ?

A. 1126 B. 1122 C. 1221 D. 1124

787. Se dă subalgoritmul `F(n, k)`, unde $n \geq 0$, $k \geq 2$, și $\text{Fib}(k)$ este al k-lea termen din șirul Fibonacci ($\text{Fib}(1) = 1, \text{Fib}(2) = 1, \text{Fib}(3) = 2, \ldots$). ✓ ?

```
Algorithm F(n, k)
   If n = 0 then
      Return 1
   Else If k < 2 then
      Return 0
   Else
      fib ← Fib(k)
      If fib > n then
         Return F(n, k - 1)
      Else
         Return F(n - fib, k - 2)
             + F(n, k - 1)
      EndIf
   EndIf
EndAlgorithm
```

Ce **nu** calculează acest subalgoritm pentru $k = \lfloor \log_\phi(\sqrt{5}n) \rfloor + 2$, unde $\phi = \frac{1+\sqrt{5}}{2}$?

A. Numărul de moduri de a scrie n ca sumă de numere Fibonacci distincte și neconsecutive

B. Numărul de partiții ale lui n în termeni Fibonacci neconsecutivi

C. Numărul de șiruri binare de lungime k fără doi de 1 consecutivi

D. Numărul de descompuneri ale lui n în sume de puteri ale lui 2

Testul 9

788. Funcția `transform(cuvant)` ia ca parametru un singur cuvânt de cel mult 255 de caractere și îi atribuie acestuia o valoare întreagă, după o regulă necunoscută. Totuși, se cunoaște faptul că apelul `transform("CEAS")` returnează valoarea 28, apelul `transform("MASA")` returnează valoarea 34, iar `transform("LUNA")` valoarea 48. Pentru care dintre cuvintele de mai jos, la apelarea funcției `transform()`, este posibil să fie returnată valoarea 28?

A. CASE B. SECA C. CASA D. CEAI

789. Se dă subalgoritmul `Build(A)`, unde A este un șir de numere, care returnează o pereche de 3 numere (r, lb, rb). Funcția `len(X)` returnează lungimea șirului X.

Indexarea de tipul $A[i \dots j]$ va returna șirul A, conținând toate elementele dintre pozițiile i și j (inclusiv).

```
Algorithm BUILD(A)
    If len(A) = 0 then
        Return ∅
    EndIf
    r ← A[1]
    i ← 2
    While i ≤ len(A)
        AND A[i] ≤ A[1] execute
        i ← i + 1
    EndWhile
    lb ← BUILD(A[2 ... i − 1])
    rb ← BUILD(A[i ... len(A)])
    Return (r, lb, rb)
EndAlgorithm
```

Ce reprezintă numărul de frunze ale arborelui construit pentru $A = \langle 5, 3, 1, 4, 8, 7, 9 \rangle$?

A. Numărul de elemente din A mai mici decât 5

B. Numărul de elemente din A care apar pe pozițiile $2k - 1, 2k$ pentru $\forall k > 1$

C. Numărul de elemente din A care apar pe pozițiile $2k, 2k + 1$ pentru $\forall k > 1$

D. Numărul de elemente din A care sunt frunze în arborele binar de căutare construit

790. Care este numărul minim de înmulțiri necesare pentru a calcula valoarea expresiei $P(x) = a_5 * x^5 + a_4 * x^4 + a_2 * x + a_0$ pentru un x dat, unde a_0, a_2, a_4, a_5 sunt numere reale nenule?

A. 5 B. 4 C. 6 D. 3

791. Considerăm tabelul T unde fiecare celulă $T[i][j]$ este definită prin:

$$T[i][j] = \begin{cases} i \times j & \text{dacă } i = j \\ T[i][j-1] \oplus T[i-1][j] & \text{dacă } i + j \text{ este par} \\ 0, & \text{dacă } i < 1 \text{ sau } j < 1 \\ T[i-1][j] \times 2 & \text{altfel} \end{cases}$$

unde $\oplus$ reprezintă XOR pe biți ($4_{10} \oplus 3_{10} = 100_2 \oplus 011_2 = 111_2 = 7_{10}$). Inițial, $T[1][1] = 1$. Care este valoarea lui $T[3][3] + T[2][3]$?

A. 9 B. 13 C. 6 D. 18

792. Se dă un număr natural N și subalgoritmul `X(N)`. ✓ ?

```
Algorithm X(N)
   t ← 0
   While N > 0 execute
      d ← N MOD 10
      t ← t + Y(d)
      N ← N DIV 10
   EndWhile
   Return t
EndAlgorithm

Algorithm Y(x)
   c ← 0
   While x > 0 execute
      c ← c + (x MOD 2)
      x ← x DIV 2
   EndWhile
   Return c
EndAlgorithm
```

Pentru $N = 475$, care este rezultatul final?

A. 6 B. 7 C. 5 D. 8

793. Se consideră algoritmul `F(N)`, unde N este un număr natural. ✓ ?

```
1: Algorithm F(n)
2:    j ← n
3:    While j > 1 execute
4:       i ← 1
5:       While i ≤ n execute
6:          i ← 2 * i
7:       EndWhile
8:       j ← j DIV 3
9:    EndWhile
10:   Return j
11: EndAlgorithm
```

Din care din următoarele clase de complexitate face parte algoritmul descris?

A. $O(\log_2(n))$

B. $O(\log_2^2(n))$

C. $O(\log_3^2(n))$

D. $O(\log_2(\log_3(n)))$

794. Se dă subalgoritmul `ceFace(n, d)`, unde $n \geq 1$, $d \geq 1$. ✓ ?

```
Algorithm ceFace(n, d)
   If n = 1 then
      Return 1
   EndIf
   total ← 0
   For k ← d, n execute
      If n MOD k = 0 then
         total ← total + ceFace(n DIV k, k)
      EndIf
   EndFor
   Return total
EndAlgorithm
```

Ce calculează apelul `ceFace(n, 2)` pentru $n \geq 2$?

A. Numărul de descompuneri ale lui n în produs de factori primi distincți

B. Numărul de partiții multiplicative ordonate ale lui n cu factori $\geq d$

C. Numărul de moduri de a scrie n ca sumă de numere prime

D. Numărul de divizori proprii ai lui n

795. Care dintre următoarele expresii logice nu sunt echivalente cu `( NOT (A OR B) OR NOT(C AND NOT A) )`? ✓ ?

A. `( NOT A AND (NOT B OR (A AND NOT C)) )`
B. `( (NOT A AND NOT B) OR (NOT C OR A) )`
C. `( NOT (A AND B) AND (A OR NOT C) )`
D. `( NOT ( (NOT A AND B) OR (C AND NOT A) ) )`

796. În câte moduri se pot așeza 3 perechi de prieteni în jurul unei mese circulare cu exact 6 locuri, astfel încât fiecare persoană să stea lângă prietenul său? ✓ ?

A. $2 \times 4!$ B. $(3-1)! \times 2^3$ C. $(2!)^3$ D. $4 \times 2!$

797. Se dă subalgoritmul `Z(n)`, unde $n \geq 1$. ✓ ?

```
Algorithm Z(n)
    a ← 0
    b ← 0
    While 2^b ≤ n execute
        b ← b + 1
    EndWhile
    r ← 0
    While a ≤ b execute
        c ← (a + b) DIV 2
        If 2^c ≤ n then
            r ← c
            a ← c + 1
        Else
            b ← c − 1
        EndIf
    EndWhile
    Return 2^r
EndAlgorithm
```

Ce returnează subalgoritmul pentru $n = 25$?

A. Cel mai mare număr k pentru care $2^k \leq n$
B. 5
C. Numărul de biți de 1 din reprezentarea binară a lui n
D. 16

798. Se dă un vector $A = \langle a_1, a_2, \ldots, a_n \rangle$. Un element a_i este "special" dacă a_i este divizibil cu $i +$ suma cifrelor lui a_i. Fie subalgoritmul `F(A, n)` unde n este un număr natural. ✓ ?

```
Algorithm F(A, n)
    cnt ← 0
    For i ← 1, n execute
        s ← 0
        x ← A[i]
        While x > 0 execute
            s ← s + (x MOD 10)
            x ← x DIV 10
        EndWhile
        If A[i] MOD (i + s) = 0 then
            cnt ← cnt + 1
        EndIf
    EndFor
    Return cnt
EndAlgorithm
```

Pentru $A = \langle 14, 22, 30, 41 \rangle$ și $n = 4$, care este rezultatul?

A. 1 B. 2 C. 3 D. 0

799. Se consideră N puncte pe un cerc, poziționate la unghiuri distincte. Se dorește acoperirea tuturor punctelor cu un număr minim de arce de cerc de lungime fixă L (măsurată în grade). Care strategie asigură acoperirea cu cele mai puține arce? ✓ ?

A. Se sortează punctele în ordine crescătoare. Se alege întotdeauna arcul care începe la primul punct neacoperit și acoperă maximum de puncte

B. Se desface cercul într-o linie, se alege întotdeauna arcul care începe la primul punct neacoperit și acoperă cât mai multe puncte posibile fără a depăși L apoi se ajustează pentru suprapunerea la capete

C. Se rotește cercul astfel încât primul arc să acopere punctul cu cea mai mare densitate, apoi se repetă pentru punctele rămase

D. Se alege întotdeauna arcul care acoperă punctul cel mai îndepărtat de orice alt arc existent

800. Se consideră operația $(2^7 + 2^7 - 1)$ efectuată în binar pe 8 biți. Care sunt valorile corecte ale rezultatului? ✓ ?

A. 255 B. -1 C. 127 D. 0

Andrei și Maria se joacă împreună jocul "Bolț", care se desfășoară astfel: cei doi încep să enumere cu voce tare, pe rând, numerele naturale începând cu 1, iar de fiecare dată când ajung la un multiplu de 7 sau la un număr care conține cifra 7, în loc să spună numărul, vor spune "Bolț!". Jocul începe astfel: Andrei spune: 1, Maria: 2, $\cdots$, Maria: 6, Andrei "Bolț!" etc. Problemele **801.** și **802.** se referă la acest joc.

801. În locul cărui număr se va striga a 100-a oară cuvântul "Bolț!"? ✓ ?

A. 336 B. 343 C. 347 D. 350

802. Cei doi decid să oprească jocul atunci când ajung la numărul 1000. Precizați care dintre cei doi a strigat de mai multe ori cuvântul "Bolț!" și de câte ori mai mult în comparație cu celălalt. ✓ ?

A. Andrei, 68 B. Maria, 64 C. Andrei, 64 D. Maria, 68

803. Se consideră algoritmul `ceFace(a)`, unde a, b, r reprezintă tablouri bidimensionale de dimensiune 2×2. Toate valorile din cele două tablouri sunt $\leq 10^5$, iar b și r au inițial toate elementele nule. ✓ ?

```
Algorithm ceFace(a)
    b[1][1] ← a[1][1] * a[1][1] + a[1][2] * a[2][1]
    b[1][2] ← a[1][1] * a[1][2] + a[1][2] * a[2][2]
    b[2][1] ← a[2][1] * a[1][1] + a[2][2] * a[2][1]
    b[2][2] ← a[2][1] * a[1][2] + a[2][2] * a[2][2]
    r[1][1] ← b[1][1] − (a[1][1] + a[2][2]) * a[1][1] + (a[1][1] * a[2][2] − a[1][2] * a[2][1])
    r[1][2] ← b[1][2] − (a[1][1] + a[2][2]) * a[1][2]
    r[2][1] ← b[2][1] − (a[1][1] + a[2][2]) * a[2][1]
    r[2][2] ← b[2][2] − (a[1][1] + a[2][2]) * a[2][2] + (a[1][1] * a[2][2] − a[1][2] * a[2][1])
EndAlgorithm
```

La finalul execuției algoritmului `ceFace(a)`, care dintre următoarele afirmații sunt adevărate despre elementele tabloului r?

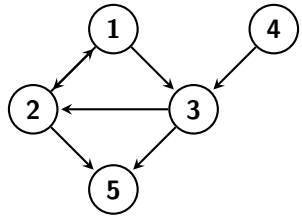
A. Dacă $a \leftarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, atunci $r = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

B. Dacă $a \leftarrow \begin{bmatrix} 10 & 19 \\ 25 & 36 \end{bmatrix}$, atunci $r = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

C. Pentru orice valori $a_{i,j}$ în matricea a, $r = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

D. Dacă $r = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, atunci cu siguranță $\det(a) = 0$

804. Se consideră următorul graf orientat cu 5 noduri. ✓ ?



Câte componente tare conexe are acest graf?

A. 2 B. 3 C. 1 D. 6

805. Se consideră algoritmul `AlexB(k)`, unde $k \leq 10$. Algoritmul `(afis)` afișează pe ecran toate valorile $s[x[i]-1]$, $i = \overline{1,n}$ și caracterul `newline`. s este un șir de cel mult 10 caractere, n este lungimea sa, iar x, P sunt șiruri de numere întregi, cu cel mult 10 elemente. ✓ ?

```
Algorithm AlExB(k)
   For i ← 1, n execute
      If !P[i] then
         x[k] ← i
         P[i] ← 1
         If k < n then
            AlexB(k+1, s x, P)
         Else
            afis(s, x)
         EndIf
         P[i] ← 0
      EndIf
   EndFor
EndAlgorithm
```

```
Algorithm F(s, x, p)
   For i ← 0, n − 1 execute
      For j ← i + 1, n − 1 execute
         If s[j] < s[i] then
            swap(s[j], s[i])
         EndIf
      EndFor
   EndFor
   AlexB(1, s, x, P)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate pentru apelul `f(s, x, p)`?

A. Pentru $s =$ `cal`, pe ecran vor fi afișate 6 șiruri de caractere distincte

B. Algoritmul afișează, pe câte o linie, toate caracterele din șirul s, de 2 ori

C. Algoritmul afișează toate anagramele cuvântului s. Se numește anagramă a lui s, un alt cuvânt ce conține toate literele din s, eventual în altă ordine

D. La finalul execuției algoritmului, întotdeauna vor fi afișate $2 \cdot n$ șiruri de caractere pe ecran

806. Se dă subalgoritmul `X(n)`, unde n este un număr natural pozitiv ($1 \leq n \leq 10^{18}$). ✓ ?

```
Algorithm X(n)
    If n ≤ 2 then
        Return 1
    Else
        M ← ((1, 1), (1, 0))
        M_putere ← P(M, n − 2)
        Return M_putere[0][0] + M_putere[0][1]
    EndIf
EndAlgorithm

Algorithm MULTIPLY(A, B)
    a ← A[0][0] · B[0][0] + A[0][1] · B[1][0]
    b ← A[0][0] · B[0][1] + A[0][1] · B[1][1]
    c ← A[1][0] · B[0][0] + A[1][1] · B[1][0]
    d ← A[1][0] · B[0][1] + A[1][1] · B[1][1]
    Return ((a, b), (c, d))
EndAlgorithm
```

```
Algorithm P(M, k)
    If k = 0 then
        Return ((1, 0), (0, 1))
    Else If k = 1 then
        Return M
    Else If k MOD 2 = 0 then
        H ← P(M, k DIV 2)
        Return MULTIPLY(H, H)
    Else
        H ← P(M, (k − 1) DIV 2)
        Return
    MULTIPLY(multiply(H, H), M)
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Complexitatea temporală a algoritmului este $O(n)$

B. Numărul de înmulțiri de matrice efectuate de funcția `p` este $O(\log k)$

C. Pentru $n = 6$, algoritmul calculează matricea $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^4$

D. Algoritmul calculează corect valoarea lui celui de a-l n-lea termen Fibbonaci pentru orice $n \geq 1$

807. Care este numărul minim de biți pentru a reprezenta numărul 97_{10} în baza 2? ✓ ?

A. 7 biți în reprezentarea fără semn

B. 16 biți în reprezentarea cu semn

C. 8 biți în reprezentarea cu semn

D. 6 biți în reprezentarea fără semn

808. Se consideră algoritmii `ceFace1(v)` și `ceFace2(x)`, unde v este un tablou unidimensional cu n elemente, iar x este un număr natural, cel mult 10^5. ✓ ?

```
Algorithm ceFace1(v, n)
   For i ← 1, n execute
      d[v[i]] ← ceFace2(v[i])
   EndFor
   For i ← 1, n execute
      For j ← i + 1, n execute
         If div[v[i]] < div[v[j]] then
            swap(v[i], v[j])
         EndIf
         If div[v[i]] = div[v[i]] then
            If v[i] > v[j] then
               swap(v[i], v[j])
            EndIf
         EndIf
      EndFor
   EndFor
EndAlgorithm
```

```
Algorithm ceFace2(x)
   If x = 1 then
      Return 1
   EndIf
   nrd ← 2
   For i ← 1, x execute
      If x MOD i = 0 then
         nrd ← nrd + 1
      EndIf
      If i * i = x then
         nrd ← nrd + 1
      EndIf
   EndFor
   Return nrd
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $v = [12, 20, 4, 100, 13]$, în urma apelului `ceFace1(v, 5)`, $v = [100, 12, 20, 4, 13]$

B. Pentru $v = [12, 20, 4, 100, 13]$, în urma apelului `ceFace1(v, 5)`, $v = [13, 4, 20, 12, 100]$

C. Algoritmul `ceFace2(x)` returnează numărul de divizori proprii ai numărului x

D. Algoritmul `ceFace1(v, n)` ordonează descrescător șirul v după numărul de divizori al fiecărui element

809. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural de exact 4 cifre. ✓ ? Algoritmii `sortC(n)` și `sortD(n)` iau ca parametru numărul n și sortează crescător, respectiv descrescător cifrele numărului n. În cazul în care numărul n conține cifra 0, aceasta este păstrată. De exemplu, la apelul `sortC(1012)`, se va returna 0112.

```
Algorithm ceFace(n)
   s ← 0
   While s ≤ 7 execute
      d ← sortD(n)
      a ← sortC(n)
      n ← d − a
      s ← s + 1
   EndWhile
   Return n
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. În urma apelului `ceFace(4215)`, se va returna 6174

B. În urma apelului `ceFace(5445)`, se va returna 7164

C. Indiferent de valoarea lui n, la finalul execuției algoritmului, se va returna aceeași valoare

D. La finalul execuției algoritmului, se va returna mereu aceeași valoare doar dacă n este multiplu de 5 și de 3

810. Se consideră algoritmul `F(n)`, unde n este un număr natural nenul, $n \geq 2$. ✓ ?

```
Algorithm F(n)
   s ← 0
   For k ← 1, n execute
      p ← 1
      For i ← 1, k execute
         p ← p * i
      EndFor
      s ← s + (k * k) DIV p
   EndFor
   Return s
EndAlgorithm
```

Care este valoarea returnată de algoritm?

A. 2 C. $n + 1$

B. $2 * n$ D. $n + 2$

811. Se consideră algoritmul `ceFace(n)`, unde n este un număr natural nenul. ✓ ?

```
Algorithm ceFace(n)
   If n ≤ 2 then
      Return 1
   EndIf
   c ← c + 1
   If n MOD 2 = 0 then
      Return ceFace(n DIV 2)
   Else
      Return ceFace(⌊√n⌋)
   EndIf
EndAlgorithm
```

Care este complexitatea de timp a algoritmului `ceFace(n)`?

A. $O(n)$

B. $O(\log n)$

C. $O(2^{\sqrt{n}})$

D. $O(2^n)$

Testul 10

812. Pe o insulă cu 4 locuitori (A, B, C, D), fiecare este fie *oracol* (spune adevărul despre viitor), fie *haos* (minte despre viitor). Un explorator întreabă pe A: „Q1: Dacă mâine aș întreba pe C dacă D este haos, va răspunde ≪Da≫?". Răspunsul (R1) e ambiguu. Apoi întreabă pe B: „Q2: Dacă mâine aș întreba pe D dacă cel puțin doi dintre voi sunteți oracoli, va răspunde ≪Nu≫?". Exploratorul deduce tipurile după acest răspuns. ✓ ?

Ce au răspuns A și B știind că exact doi sunt oracoli?

A. R1: Da, R2: Da
B. R1: Da, R2: Nu
C. R1: Nu, R2: Da
D. R1: Nu, R2: Nu

813. Care formulă calculează corect numărul de numere naturale din intervalul $[1, N]$ care NU sunt divizibile cu 3, 5 sau 7? ✓ ?

A. $N - \lfloor \frac{N}{3} \rfloor - \lfloor \frac{N}{5} \rfloor - \lfloor \frac{N}{7} \rfloor$
B. $N - \lfloor \frac{N}{3} \rfloor - \lfloor \frac{N}{5} \rfloor - \lfloor \frac{N}{7} \rfloor + \lfloor \frac{N}{15} \rfloor + \lfloor \frac{N}{21} \rfloor + \lfloor \frac{N}{35} \rfloor - \lfloor \frac{N}{105} \rfloor$
C. $N - \left(\lfloor \frac{N}{3} \rfloor + \lfloor \frac{N}{5} \rfloor + \lfloor \frac{N}{7} \rfloor\right) + \left(\lfloor \frac{N}{15} \rfloor + \lfloor \frac{N}{21} \rfloor + \lfloor \frac{N}{35} \rfloor\right)$
D. $\lfloor \frac{N}{15} \rfloor + \lfloor \frac{N}{21} \rfloor + \lfloor \frac{N}{35} \rfloor - \lfloor \frac{N}{105} \rfloor$

814. Se dă subalgoritmul `F(A, p, k)`, unde A este un șir de numere naturale, p este poziția curentă, iar k este dimensiunea curentă a submulțimii. ✓ ?

```
Algorithm F(A, p, k)
   If p > len(A) then
      Return 1  If  k ≥ 1
           Else  0
   EndIf
   t ← 0
   If A[p] MOD (k + 1) = 0
then
      t ← t+F(A, p+1, k+1)
   EndIf
   t ← t + F(A, p + 1, k)
   Return t
EndAlgorithm
```

Ce nu calculează acest subalgoritm când este apelat cu `F(A, 1, 0)`?

A. Numărul de submulțimi nevide cu toate elementele pare
B. Numărul de submulțimi nevide unde fiecare element este divizibil cu poziția sa în submulțime
C. Numărul de submulțimi cu suma elementelor egală cu dimensiunea lor
D. Numărul de partiții ale lui A în submulțimi de dimensiune k

815. Se dă subalgoritmul `F(P, i, S)`, unde P este o listă de puncte, i este indicele curent, iar S este o variabilă care păstrează suma. Operatorul "/" reprezintă împărțirea pe numere reale ($\in$ R) ✓ ?

```
Algorithm F(P, i, S)
   If i > len(P) then
      Return |S|/2
   Else
      x_i, y_i ← P[i]
      x_{i+1}, y_{i+1} ← P[(i MOD len(P)) + 1]
      Return F(P, i + 1, S + (x_i * y_{i+1} − x_{i+1} * y_i))
   EndIf
EndAlgorithm
```

Ce calculează apelul `F(P, 1, 0)`?

A. Aria cu semn a poligonului definit de punctele P, în ordinea dată

B. Aria fără semn a poligonului determinat de punctele P

C. Suma $\sum_{i=1}^{n} x_i y_{i+1}$

D. Jumătate din valoarea absolută a sumei $\sum_{i=1}^{n} (x_i y_{i+1} - x_{i+1} y_i)$

816. Fie N un număr natural. Care afirmații sunt întotdeauna adevărate? ✓ ?

A. Dacă N este palindrom în baza 4, atunci este palindrom și în baza 2

B. Numărul de cifre în baza 4 este cel mult $\lfloor \frac{\text{cifre\_b2}}{2} \rfloor + 1$, unde *cifre_b2* este numărul de cifre în baza 2

C. Dacă N este divizibil cu 5 în baza 10, ultima sa cifră în baza 4 este 1

D. Suma cifrelor în baza 4 este mai mare sau egală cu numărul de biți de 1 în reprezentarea binară

817. Se dă subalgoritmul `X(a, b, m)`, unde $a, b, m \in \mathbb{N}$. ✓ ?

```
Algorithm X(a, b, m)
    If b = 0 then
        Return 1
    Else If b MOD 2 = 0 then
        t ← X(a, b DIV 2, m)
        Return (t * t) MOD m
    Else
        t ← X(a, (b − 1)/2, m)
        Return (a * t * t) MOD m
    EndIf
EndAlgorithm
```

Care afirmații sunt adevărate?

A. Complexitatea temporală este de $O(\log b)$ operații aritmetice

B. Pentru $a = 3, b = 5, m = 7$, rezultatul este 5

C. Numărul de apeluri recursive este $\lfloor \log_2 b \rfloor +$ număr de biți de 1 în reprezentarea binară a lui b

D. Dacă $m = 1$, funcția returnează 0

818. Fie expresia ✓ ?

$$E(a,b) = \begin{cases} (a * (b \texttt{ DIV } 2) + (a \texttt{ MOD } 3)) & \texttt{dacă } (a \texttt{ MOD } b \neq 0 \texttt{ AND } (a+b) \texttt{ MOD } 2 = 0) \\ (b * (a \texttt{ DIV } 4) - (b \texttt{ MOD } 5)) & \texttt{altfel} \end{cases}$$

Care dintre următoarele afirmații sunt adevărate?

A. Pentru $a = 7, b = 3$, $E(a, b) = 8$

B. Expresia întoarce întotdeauna un număr par când $a + b$ este impar

C. Dacă $a = 10, b = 4$, rezultatul este 10

D. $E(a, b)$ este divizibil cu 3 atunci când a **MOD** $3 = 0$

819. Se dă subalgoritmul `G(n, k)`, unde $n, k \geq 1$. ✓ ?

```
Algorithm G(n, k)
   If k = 1 then
      Return 1 If n ≥ 2 Else 0
   EndIf
   s ← 0
   For i ← 2, n execute
      If G(i − 1, 1) = 1
          AND G(n − i, k − 1) ≥ 1 then
         s ← s + 1
      EndIf
   EndFor
   Return s
EndAlgorithm
```

Ce calculează acest subalgoritm?

A. Numărul de moduri de a scrie n ca sumă de k numere prime

B. Numărul de moduri de a diviza n în k numere naturale ≥ 2

C. Numărul de descompuneri ale lui n în k termeni primi distincți

D. Numărul de moduri de a plasa $k-1$ numere prime între 2 și n

820. Se dă subalgoritmul `M(A)`, unde A este un șir de numere. ✓ ?

```
Algorithm M(A)
   c ← −1, t ← 0
   For i ← 1, len(A) execute
      x ← A[i]
      If t = 0 then
         c ← x
         t ← 1
      Else
         If x = c then
            t ← t + 1
         Else
            t ← t − 1
         EndIf
      EndIf
   EndFor
   Return c
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Returnează elementul majoritar (apărut $> [n/2]$ ori) dacă există

B. Funcționează corect numai dacă șirul este sortat

C. Complexitatea sa este $O(n)$ timp și $O(1)$ spațiu

D. Returnează întotdeauna elementul cu cea mai mare frecvență

821. Se consideră suma returnată de subalgoritmul `S(x)`, unde x este un număr natural. Care este valoarea acestei sume? ✓ ?

```
Algorithm S(x)
   r ← 0, p ← 1
   For i ← 1, x execute
      p ← p ∗ (i + 1)
      r ← r + i/p
   EndFor
   Return r
EndAlgorithm
```

A. $1 - \dfrac{1}{x!}$ C. $\dfrac{1}{x+1}$

B. $1 - \dfrac{1}{(x+1)!}$ D. $\dfrac{x}{(x+1)!}$

822. Se dă subalgoritmul `E(s, t)`, unde s și t sunt șiruri de caractere majuscule. Considerăm algoritmul `cod(c)` care returnează codul indicele caracterului c în alfabet. ✓ ?

```
Algorithm E(s, t)
   If len(s) ≠ len(t) then
      Return false
   EndIf
   For i ← 1, 26 execute
      v[i] ← 0
   EndFor
   For i ← 1, len(s) execute
      v[cod(s[i])] ← v[cod(s[i])] + 1
      v[cod(t[i])] ← v[cod(t[i])] − 1
   EndFor
   For j ← 1, 26 execute
      If v[j] ≠ 0 then
         Return FALSE
      EndIf
   EndFor
   Return TRUE
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Verifică dacă s și t sunt anagrame

B. Verifică dacă s și t au aceleași caractere, indiferent de frecvență

C. Verifică dacă toate caracterele din s există în t și reciproc

D. Verifică dacă diferența individuală a frecvențelor caracterelor este zero

823. Care din următorii algoritmi calculează corect $F(n)$, definită prin relația de recurență ✓ ?
$F(n) = 3F(n-1) + 2F(n-3)$, cu condițiile inițiale $F(0) = 1, F(1) = 2, F(2) = 5$, în complexitatea de timp specificată? Se presupune că operațiile aritmetice sunt $O(1)$.

A. Complexitate $O(n)$

```
1: Algorithm F(n)
2:    If n = 0 then
3:       Return 1
4:    Else If n = 1 then
5:       Return 2
6:    Else If n = 2 then
7:       Return 5
8:    Else
9:       Return 3 * F(n − 1)
10:         +2 * F(n − 3)
11:   EndIf
12: EndAlgorithm
```

B. Complexitate $O(n)$

```
1: Algorithm F(n)
2:    If n ≤ 2 then
3:       v ← {1, 2, 5}
4:       Return v[n]
5:    EndIf
6:    v ← {1, 2, 5}
7:    j ← 0
8:    For i ← 1, n − 2 execute
9:       x ← 3 * v[(j + 2) MOD 3]+
10:       2 * v[j MOD 3]
11:       v[j MOD 3] ← x
12:       j ← j + 1
13:   EndFor
14:   Return v[(j + 2) MOD 3]
15: EndAlgorithm
```

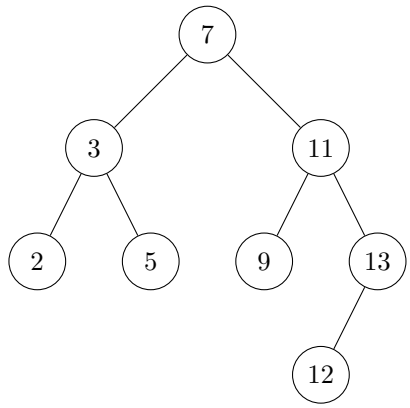
C. Complexitate $O(\log n)$

```
1: Algorithm F(n)
2:     If n ≤ 2 then
3:         v ← {1, 2, 5}
4:         Return v[n]
5:     EndIf
6:     a, b, c ← 1, 2, 5
7:     For i ← 3, n execute
8:         new ← 3c + 2a
9:         a, b, c ← b, c, new
10:    EndFor
11:    Return c
12: EndAlgorithm
```

D. Complexitate $O(1)$

```
1: Algorithm F(n)
2:     b = −3, c = 0, d = −2
3:     D = q_1^2/4 + p_1^3/27
4:     p_1 = c − b^2/3
5:     q_1 = 2b^3/27 − bc/3 + d
6:     u = ∛(−q_1/2 + √D)
7:     v = ∛(−q_1/2 − √D)
8:     r_1 ← u + v − b/3
9:     Return ⌊r_1^n * 100⌋
10: EndAlgorithm
```

824. Se consideră următorul arbore binar de căutare construit prin inserarea succesivă a elementelor într-un arbore (inițial gol): ✓ ?



✓ ?

Care dintre următoarele secvențe de inserare **NU POT** genera acest arbore?

A. 7, 3, 11, 2, 5, 9, 13, 12
B. 7, 11, 3, 12, 13, 5, 2, 9
C. 7, 3, 2, 5, 11, 9, 13, 12
D. 7, 11, 3, 9, 13, 12, 5, 2

825. Un tablou de 7 elemente întregi este sortat folosind algoritmul de sortare prin inserție. După primele patru iterații ale algoritmului, care dintre configurațiile următoare este posibilă? ✓ ?

A. $[2, 3, 5, 7, 4, 6, 1]$
B. $[3, 6, 4, 8, 5, 1, 9]$
C. $[1, 4, 6, 8, 3, 5, 2]$
D. $[2, 5, 7, 8, 6, 3, 1]$

826. Se consideră următorul algoritm de căutare binară într-un șir de cuvinte sortate alfabetic, unde `cuvant` și ℓ sunt șiruri de caractere: ✓ ?

```
Algorithm CAUTA(cuvant, dictionar, ℓ)
    stanga ← 1
    dreapta ← len(dictionar)
```

```
    While stanga ≤ dreapta execute
        mijloc ← ⌊stanga + (dreapta − stanga)/2⌋
        If dictionar[mijloc] = cuvant then
            Return TRUE
        Else If dictionar[mijloc] < cuvant then
            If dictionar[mijloc] = ℓ  then
                stanga ← mijloc + 1
            Else
                stanga ← mijloc + 2
            EndIf
        Else
            dreapta ← mijloc − 1
        EndIf
    EndWhile
    Return FALSE
EndAlgorithm
```

Șirul conține cuvintele: ["castan", "fag", "larice", "plop", "salcie", "stejar", "tei"]. Care dintre următoarele secvențe de comparații **NU POT** apărea în timpul căutării cuvântului "tei", când ℓ = stejar?

A. "plop" → "stejar" → "tei"

B. "plop" → "salcie" → "stejar" → "tei"

C. "plop" → "tei"

D. "plop" → "salcie" → "tei"

827. Se dă subalgoritmul ceFace(n, a), unde n și a sunt numere naturale pozitive ✓ ? $(2 \leq n \leq 10^9, 2 \leq a \leq n-1)$. Algoritmul GCD(a, b) returnează cel mai mare divizor comun al numerelor a, b, unde $a, b \in \mathbb{N}$.

```
Algorithm ceFace(n, a)
    If gcd(a, n) ≠ 1 then
        Return false
    EndIf
    result ← putere(a, n-1, n)
    If result ≠ 1 then
        Return false
    EndIf
    Return true
EndAlgorithm
```

```
Algorithm putere(b, e, mod)
    If e = 0 then
        Return 1
    Else If e MOD 2 = 0 then
        H ← putere(b, e / 2, mod)
        Return (H * H) MOD mod
    Else
        H ← putere(b, (e - 1) / 2, mod)
        Return (b * H * H) MOD mod
    EndIf
EndAlgorithm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Pentru $n = 341$ și $a = 2$, algoritmul returnează **true**

B. Dacă algoritmul returnează **true**, atunci n este obligatoriu compus

C. Numărul de apeluri recursive ale funcției putere este $O(\log e)$

D. Pentru $n = 91$ și $a = 3$, algoritmul returnează **false**

828. Se consideră următorul algoritm: ceFace(arr, n, i) unde arr este un vector de ✓ ? numere întregi, iar n, i sunt numere întregi, unde n reprezintă lungimea șirului arr. Funcția print(arr) afișează șirul de numere arr.

```
Algorithm ceFace(arr, n, i)
   If _________ then
      print(arr)
      Return
   EndIf
   For j ← i, n execute
      swap(arr[i], arr[j])
      ceFace(arr, n, i + 1)
      _________
   EndFor
EndAlgorithm
```

Care dintre următoarele completări corectează cele două linii lipsă, astfel încât algoritmul să genereze toate permutările vectorului *arr*?

A. Linia 1: `i == n`
Linia 2: `swap(arr[j], arr[i])`

B. Alt răspuns

C. Linia 1: `i > n`
Linia 2: `swap(arr[j], arr[i])`

D. Linia 1: `i == n`
Linia 2: `swap(arr[i], arr[j])`

829. Prin experiment, se determină că sortarea prin selecție efectuează 5000 de comparații pentru sortarea unui vector de dimensiune k. Dacă dimensiunea vectorului se dublează la $2k$, câte comparații aproximativ se vor efectua? ✓ ?

A. 10000 B. 15000 C. 20000 D. 40000

830. Se consideră funcția `check(n)` care verifică dacă un număr întreg n satisface: ✓ ?

1. Este par **SAU** divizibil cu 3, **ȘI** nu este negativ

```
Algorithm check(n)
   Return _____________________
EndAlgorithm
```

Care dintre următoarele expresii completează corect funcția?

A. `(n MOD 2 == 0 OR n MOD 3 == 0) AND n >= 0`

B. `n >= 0 AND (n MOD 2 == 0 OR n MOD 3 == 0)`

C. `!(n < 0) AND (n MOD 2 == 0 OR n MOD 3 == 0)`

D. `n MOD 2 == 0 OR n MOD 3 == 0 AND n >= 0`

831. Se dă subalgoritmul `Phi(n)`, unde n este un număr natural. Fie funcția `gcd(a, b)` care returnează cel mai mare divizor comun dintre a și b. ✓ ?

```
Algorithm Phi(n)
   If n = 1 then
      Return 1
   EndIf
   count ← 0
   For i ← 1, n execute
      If gcd(i, n) = 1 then
         count ← count + 1
      EndIf
   EndFor
   Return count
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Subalgoritmul calculează numărul de numerele întregi pozitive mai mici cu n și prime cu acesta

B. Rezultatul returnat include în calcul și numărul n însuși

C. Subalgoritmul returnează întotdeauna un rezultat mai mic decât n

D. `Phi(15) = 8`

832. Se consideră 9 bile identice, dintre care una are o greutate ușor diferită (mai grea sau mai ușoară). Folosind o balanță cu brațe egale, trebuie să identificați bila diferită în cel mult 3 cântăriri. Fie următoarele strategii: ✓ ?

i. Prima cântărire constă în compararea a două grupuri de câte 3 bile. Dacă balanța este echilibrată, bila diferită se află în cele 3 bile rămase. În caz contrar, se analizează direcția dezechilibrului.

ii. Prima cântărire compară două grupuri de câte 4 bile. Dacă balanța este echilibrată, bila diferită este cea rămasă. În caz contrar, se continuă cu cele 4 bile de pe talerul mai greu.

iii. După o primă dezechilibrare, se mută 2 bile de pe talerul greu pe cel ușor și se adaugă o bilă necântărită. Se analizează schimbarea direcției.

iv. Se utilizează o strategie de eliminare bazată pe paritatea numărului de bile cântărite pe fiecare taler în fiecare etapă.

Care dintre următoarele afirmații sunt adevărate?

A. Metoda i. permite identificarea bilei diferite indiferent dacă aceasta este mai grea sau mai ușoară

B. Metoda ii. necesită exact 3 cântăriri pentru a determina bila diferită

C. Metoda iii. folosește principiul transpunerii bilelor între cântăriri pentru a deduce natura diferenței

D. Metoda iv. garantează că numărul total de bile de pe fiecare taler este mereu par în fiecare etapă

Alex și Diana joacă „Șarpele numeric", un joc în care construiesc alternativ un șarpe din numere naturale. La fiecare tură, jucătorul trebuie să adauge un număr natural x care să respecte următoarele reguli:

i. x trebuie să fie strict mai mare decât ultimul număr din șarpe (coada șarpelui);

ii. $x +$ (coada șarpelui) trebuie să fie multiplu de 3.

Alex începe jocul cu numărul 1. Diana pune următorul număr, apoi Alex și tot așa, alternativ. După primele 8 mutări (4 ale lui Alex și 4 ale Dianei), s-a ajuns la un anumit ultim număr din șarpe. Problemele **833.** și **834.** se referă la acest joc.

833. Care este cel mai mic număr pe care Alex îl poate pune în a 5-a lui mutare (mutarea a 9-a din joc) dintre opțiunile de mai jos, astfel încât regula de mai sus ($x +$ coada multiplu de 3 și x mai mare decât coada) să fie respectată? ✓ ?

A. 15 B. 16 C. 17 D. 18

834. Jucătorii decid ca, într-o nouă partidă a jocului „Șarpele numeric", regula să se modifice astfel încât, la fiecare tură, suma lui x cu numărul din coada șarpelui (ultimul număr pus) să fie multiplu de 4. Alex pornește jocul cu 1, iar apoi Diana și Alex vor continua alternând mutările, fiecare fiind obligat să pună un număr strict mai mare decât coada șarpelui și care să îndeplinească noua condiție (suma multiplu de 4). Care dintre următoarele afirmații sunt adevărate? ✓ ?

A. Doar Alex poate câștiga

B. Diana va fi câștigătoare întotdeauna

C. Jocul va continua la infinit, fără câștigător

D. Jocul se va bloca întotdeauna după maximum 10 ture

835. Se consideră algoritmul `afiș(x)`, unde x este un număr natural ($1 \leq x \leq 10^4$). ✓ ?

```
Algorithm AFIȘ(x)
   If x < 8000 then
      Write x, " "
      afiș(4 * x)
      Write x, " "
   EndIf
EndAlgorithm
```

Ce se afișează pentru apelul `afiș(1000)`?

A. 1000 4000 4000 4000 1000

B. 1000 4000 4000 1000

C. 1000 4000 8000 4000 1000

D. 1000 4000 8000

Partea

III

Răspunsuri și indicații

Răspunsuri

1. B C
2. B
3. B D
4. A D
5. B D
6. D
7. D
8. A
9. A
10. C
11. C
12. B
13. A
14. B
15. A B C
16. B C
17. A B
18. A C
19. B D
20. A C
21. C
22. B
23. B
24. C
25. C
26. A
27. C
28. C
29. D
30. B
31. A
32. B
33. A
34. D
35. B
36. A
37. B
38. D
39. C
40. C
41. A
42. A
43. B
44. A
45. A
46. A D
47. A B D
48. B D
49. A
50. A B C
51. B C
52. A B
53. B C
54. A C
55. A C D
56. A C
57. B
58. A B
59. D
60. A B
61. A C
62. D
63. B D
64. C
65. A B
66. B C
67. B
68. A
69. B C D
70. B D
71. C D
72. A B D
73. D
74. A D
75. A
76. B
77. B
78. A B D
79. C
80. A B D
81. A B D
82. B C
83. A D
84. A D
85. A B D
86. A B D
87. A C
88. D
89. A B D
90. B
91. C
92. D
93. A D
94. D
95. C
96. C
97. A
98. A C
99. C D
100. A D
101. B C
102. B D
103. A D
104. A B
105. A D
106. A C
107. A B C
108. C
109. D
110. C
111. A C D
112. A
113. A C
114. C
115. B C D
116. B C D
117. B C
118. C
119. A B
120. A C

121. B C D
122. A C
123. A B C
124. B C
125. A D
126. A D
127. D
128. D
129. C
130. A C D
131. A D
132. D
133. C
134. A
135. A B C
136. A B D
137. A B C
138. A B C
139. A C
140. B C D
141. B
142. A C
143. B C
144. A B D
145. B C
146. A B C
147. A C D
148. A C D
149. A B C
150. A B
151. A B C
152. A C
153. B
154. C
155. B D
156. A C
157. B
158. A C
159. B
160. A
161. A B
162. C
163. A C D
164. A D
165. C
166. C
167. C
168. B C
169. B
170. A C
171. C
172. C D
173. A D
174. A B D
175. B
176. A C
177. B C D
178. A B C
179. A B D
180. A B C
181. A B
182. A B
183. C D
184. B
185. B
186. C
187. D
188. D
189. B C
190. A B
191. B C
192. B
193. A B D
194. A C D
195. B C D
196. A B
197. B C
198. B D
199. A C
200. B C
201. A D
202. A C D
203. A B D
204. A C
205. A B D
206. B
207. A
208. B C
209. D
210. D
211. C
212. A
213. A
214. B
215. C
216. A B
217. B D
218. B
219. D
220. A
221. A B
222. A
223. B
224. D
225. D
226. B C
227. B C
228. C D
229. A C
230. A
231. B C
232. A D
233. A B D
234. A B D
235. A C
236. A B
237. C
238. D
239. B
240. D
241. C
242. A D
243. B
244. C
245. B
246. A
247. B
248. A
249. B
250. C
251. A
252. D

253. C
254. B
255. C D
256. A D
257. B C
258. A B C
259. B C
260. D
261. C
262. D
263. B
264. B C D
265. B C D
266. A B D
267. A B D
268. A D
269. C
270. A
271. B
272. A B
273. B C D
274. A C
275. A D
276. B D
277. A
278. C
279. B C
280. B
281. C D
282. B
283. A B
284. B C D
285. A
286. B D
287. B
288. A C D
289. D
290. A
291. B
292. A B
293. A D
294. B
295. A B C
296. A C D
297. B D
298. A D
299. B
300. B D
301. A D
302. C
303. A B D
304. A
305. A
306. A C
307. A B
308. A B D
309. D
310. B
311. C D
312. A D
313. A D
314. C
315. B D
316. A B
317. A D
318. B C D
319. A B D
320. B
321. A D
322. D
323. C
324. B C D
325. C
326. A
327. B
328. A B
329. B D
330. A
331. A D
332. A B
333. A B C
334. A B C
335. C
336. A C
337. B C D
338. B
339. A C D
340. C
341. B C
342. A
343. C D
344. A B
345. C
346. B
347. A C
348. B
349. A B D
350. B
351. C
352. A B C
353. A
354. A D
355. D
356. C
357. A B C
358. A B
359. A
360. B
361. A C
362. B C
363. A
364. A
365. B D
366. C D
367. C
368. A B
369. A B
370. A B
371. C
372. B
373. A D
374. C
375. D
376. D
377. A
378. A
379. B
380. A C
381. A C D
382. D
383. C
384. B D

385. B
386. A B D
387. B D
388. B C
389. D
390. C
391. B C
392. D
393. A C D
394. A D
395. A B C
396. A B
397. A B
398. B
399. A
400. B C
401. D
402. C D
403. A
404. B D
405. B
406. B C
407. B
408. C
409. D
410. C
411. B
412. A
413. D
414. B
415. B
416. C
417. B C D
418. C
419. A B C
420. B
421. A C D
422. A C D
423. A C
424. A B D
425. A
426. C D
427. C D
428. C
429. B
430. A
431. B D
432. B
433. D
434. A B C
435. B
436. D
437. C
438. A C
439. A
440. A B
441. A C D
442. A B
443. A
444. B D
445. B D
446. C
447. C D
448. A
449. B D
450. B C
451. C
452. B C
453. A D
454. A
455. B C
456. B C
457. A
458. A C
459. B
460. C
461. D
462. D
463. B C
464. A
465. B C
466. A B D
467. B C
468. C D
469. C
470. D
471. A B C D
472. B
473. B
474. C D
475. A C D
476. A B C D
477. A C
478. B D
479. C
480. B D
481. D
482. B D
483. A
484. B
485. D
486. B C
487. B C
488. B D
489. A B
490. C
491. B C
492. C
493. A C
494. A
495. B C
496. B C D
497. A B
498. B C D
499. C
500. A C
501. B C
502. A D
503. A C
504. A B
505. C
506. B
507. A C
508. C
509. C
510. D
511. C
512. A
513. A
514. A D
515. A B C
516. A B C

517. C
518. A C
519. A
520. B
521. C D
522. B D
523. D
524. A B
525. A C
526. B D
527. A D
528. A C
529. A C
530. B C D
531. B D
532. A C
533. A D
534. B C
535. B D
536. C
537. A B D
538. A D
539. C
540. B
541. D
542. A B D
543. C
544. B
545. C
546. C
547. A C D
548. C
549. B
550. B D
551. B
552. C D
553. A C
554. B D
555. B
556. D
557. A C
558. A D
559. A
560. A
561. B
562. D
563. C
564. B C
565. A D
566. A B
567. B
568. A
569. C
570. B
571. C
572. D
573. B
574. A B C
575. D
576. B C D
577. D
578. B
579. D
580. A C
581. C
582. B
583. A
584. B
585. A C
586. B D
587. B
588. C
589. B D
590. A B D
591. A D
592. B D
593. B
594. C
595. B
596. A B
597. B
598. B
599. B D
600. A D
601. C
602. C D
603. A B D
604. C D
605. A B
606. B
607. C D
608. A C D
609. B C D
610. A C D
611. A C
612. A B D
613. C D
614. B D
615. A D
616. B D
617. A B C
618. B
619. C D
620. A B D
621. A B D
622. B C
623. A B
624. B C D
625. D
626. A B D
627. A B
628. A B C
629. A
630. C
631. B C
632. D
633. B C D
634. B C D
635. A
636. A C
637. C D
638. A C
639. A C
640. A C
641. A B C
642. A B D
643. D
644. D
645. A B C
646. A B C
647. B C
648. B

649. A C D
650. B D
651. B C D
652. D
653. A
654. A B
655. B C D
656. B
657. A C
658. C
659. B
660. A B C
661. A
662. C
663. B D
664. A B D
665. A C D
666. C
667. A B
668. C
669. A
670. B
671. A C D
672. A C
673. A C D
674. C
675. B
676. C D
677. A C
678. B
679. A B C
680. A B C
681. A B D
682. B D
683. A C D
684. C D
685. A C D
686. C
687. B C D
688. A B D
689. C
690. B C
691. C D
692. B
693. A C
694. B C
695. A
696. A
697. A B
698. A D
699. B D
700. A
701. A C
702. A B
703. A C D
704. B D
705. A C
706. B C D
707. A C
708. C
709. A C
710. C
711. A
712. A
713. B C
714. C
715. A C
716. C
717. B D
718. B
719. A D
720. A
721. D
722. B
723. A C
724. C
725. B C
726. D
727. C
728. B C D
729. B D
730. C
731. C
732. C
733. D
734. B C D
735. A C
736. A C
737. A B
738. A D
739. A
740. A
741. A C
742. B
743. C
744. C
745. D
746. B D
747. B C
748. B C
749. A C
750. D
751. A
752. B C
753. B D
754. A C
755. A D
756. A C
757. C
758. B C
759. C
760. C
761. A B
762. B C
763. A
764. B C D
765. B
766. A
767. A
768. B
769. A C
770. A
771. C
772. A B
773. A
774. D
775. A
776. C
777. D
778. A B C
779. B
780. D

781. D
782. C
783. A
784. B
785. A
786. B
787. B C D
788. A B
789. B D
790. A
791. A
792. A
793. B C
794. B
795. A C D
796. B
797. D
798. A
799. B
800. A B
801. A
802. A
803. B C
804. B
805. A C
806. B C D
807. A C
808. A D
809. A C
810. A
811. B
812. B
813. B
814. A C D
815. B D
816. B D
817. A B
818. A
819. B
820. A C
821. B
822. A D
823. B
824. B
825. A C D
826. B C D
827. A C
828. D
829. C
830. A B C
831. A D
832. A C
833. B
834. C
835. B

15

Rezolvări

1. Algoritmul pare să interschimbe elementele a și b, dar nu funcționează corect pe toate cazurile, deoarece după prima operație a va reține doar partea întreagă din a DIV b. A. Fals, $a = 18$ și $b = 20$, a va deveni 0, b va deveni 0, iar apoi a se calculează ca 0 DIV 0, unde va apărea o eroare. B. Adevărat, $a = 28$ și $b = 10$, a va deveni 2, b va deveni 20, iar apoi a se calculează ca 20 DIV 2, și se va afișa 10 20. C. Adevărat, $a = 20$ și $b = 10$, se vor interschimba corect elementele, deoarece a este multiplu de b și se va afișa 10 20. D. Fals, $a = 10$ și $b = 20$, a va deveni 0, b va deveni 0, iar apoi a se calculează ca 0 DIV 0, unde va apărea o eroare.

2. Analizăm fiecare algoritm: - `swap1`: Realizează greșit interschimbarea valorilor, din cauza liniei $a \leftarrow b - a$. - `swap2`: Realizează corect interschimbarea valorilor folosind metoda prin înmulțiri. - `swap3`: Realizează greșit interschimbarea valorilor, se poate pierde restul, folosind DIV. - `swap4`: Realizează greșit interschimbarea valorilor, din cauza liniilor $b \leftarrow a - b$, $a \leftarrow b + a$. Astfel, răspunsul corect este B.

3. Numărul total de valori dintr-un interval $[a, b]$, atunci când pasul este k, se calculează folosind formula $\frac{b-a}{k} + 1$. În algoritmul dat, dacă $a > b$, se face o interschimbare pe biți folosind operatorul XOR, iar pasul nu este k, ci $2k$, deoarece, în fiecare iterație, a se incrementează cu k, iar b se decrementează cu k. Prin urmare, numărul total de valori va fi $\frac{b-a}{2k} + 1$.

La punctul B, se afișează 12 valori, aplicând formula. La punctul C, se afișează corect valorile. La punctul D, ultima valoare care se afișeaza este mai mare decât $\frac{a+b}{2}$. Așadar, afirmațiile care nu sunt adevărate sunt B și D.

4. Pentru ca algoritmul `Bad(a, b)` să interschimbe corect valorile lui a și b, prin metoda scăderilor, linia trebuie completată cu $b \leftarrow a + b$, care este varianta A. De asemenea, varianta D, adusă la o formă simplificată, reprezintă același lucru. Astfel, răspunsul corect este A, D.

5. Algoritmul `container(n)` returnează cel mai mare număr k astfel încât 2^k să fie mai

mic sau egal cu n. La punctul A, pentru $n = 1024$, algoritmul returnează 10, deoarece $1024 = 2^{10}$. La punctul B, algoritmul returnează 8, deoarece cel mai mare număr putere de 2 mai mic decât 336 este $256 = 2^8$. La punctul C, algoritmul nu returnează o putere de 2, ci doar exponentul k. La punctul D, descrierea reflectă exact funcționarea algoritmului. Astfel, variantele corecte sunt B și D.

6. Se determină numărul total de numere de 1, 2 sau 3 cifre care au suma cifrelor egală cu 9.

7. Pentru ca un număr să fie divizibil atât cu 5 cât și cu 9, este necesar ca restul împărțirii la 5 să fie 0 ($n \bmod 5 = 0$) și restul împărțirii la 9 să fie 0 ($n \bmod 9 = 0$). Nicio altă expresie nu respectă simultan aceste condiții.

8. Pentru ca expresia să fie `True`, trebuie să fie îndeplinite simultan condițiile $y > 10$ și y `MOD` $4 \neq 0$, ceea ce face varianta `A` corectă. Restul opțiunilor nu respectă complet cerințele problemei.

9. Pentru ca expresia să fie `True`, este necesar ca `k` să fie un număr par (k `MOD` $2 = 0$) și simultan mai mic decât 20 ($k < 20$). Celelalte opțiuni nu respectă complet aceste condiții.

10. Pentru ca expresia să fie `True`, trebuie ca `m` să fie un număr negativ ($m < 0$) și divizibil cu 8 (m `MOD` $8 = 0$). Celelalte opțiuni fie contrazic cerința, fie nu verifică simultan ambele condiții.

11. Pentru $\frac{0+8}{4-2} + 17 \bmod 3 = \frac{8}{2} + 2 = 4 + 2 = 6$, unde am calculat mai întâi $16 \bmod 4 = 0$, $4 * 2 = 8$, $16 \div 4 = 4$, iar apoi $17 \bmod 3 = 2$.

12. Pentru $(((21 \bmod 7)+3)*((19 \div 3))-(7 \bmod 2) = (0+3)*6-1 = 3*6-1 = 18-1 = 17$, unde $21 \bmod 7 = 0$ (rest 0), $19 \div 3 = 6$ (cât 6) și $7 \bmod 2 = 1$ (rest 1).

13. Analizăm expresia pas cu pas: $(b * c) = 5 \times 8 = 40$; $(c$ **MOD** $(b+2)) = 8$ **MOD** $7 = 1$; 40 **DIV** $1 = 40$; Partea stângă: $10 + 40 = 50$; $((c-1) * (b+2)) = 7 \times 7 = 49$; Rezultatul final: $50 - 49 = 1$, deci valoarea lui `a` va fi 1.

14. Analizăm expresia pas cu pas: $(x$ **MOD** $(y-1)) = 8$ **MOD** $14 = 8$; $(z * (x+1)) = 5 * 9 = 45$; Suma din paranteză: $(8+45) = 53$; $((y-1)$ **MOD** $(z-1)) = 14$ **MOD** $4 = 2$; Împărţirea finală: 53 **DIV** $2 = 26$.

15. A. Expresia are valoarea True. B. Expresia are valoarea False. C. Expresia are

valoarea True. Expresia are valoarea False.

16. A. Expresia are valoarea False. B. Expresia are valoarea True. C. Expresia are valoarea True. Expresia are valoarea False.

17. A. Expresia are valoarea True. B. Expresia are valoarea True. C. Expresia are valoarea False. Expresia are valoarea False.

18. A. Expresia are valoarea True. B. Expresia are valoarea False. C. Expresia are valoarea True. Expresia are valoarea False.

19. A. Expresia are valoarea False. B. Expresia are valoarea True. C. Expresia are valoarea False. Expresia are valoarea True.

20. A. Expresia are valoarea False. B. Expresia are valoarea True. C. Expresia are valoarea False. Expresia are valoarea True.

21. $((p+1)$ **MOD** $q) = 13$ **MOD** $3 = 1$; $(r * (p+1)) = 4 * 13 = 52$; Prima parte: $1 + 52 = 53$; $(q$ **MOD** $(r-1)) = 3$ **MOD** $2 = 1$; A doua parte: $53 - 0 = 53$; $((p+1)$ **DIV** $(q+2)) = 13$ **DIV** $5 = 2$; Rezultatul final: $53 + 2 = 55$.

22. Analizăm expresia pas cu pas: $((a-1)$ **MOD** $b) = 8$ **MOD** $2 = 0$; $(b * (c+1)) = 2 * 6 = 12$; $((a-1)$ **DIV** $b) = 8$ **DIV** $2 = 4$; Rezultatul final: $0 + 12 - 4 = 8$.

23. $(x$ **MOD** $y) = 25$ **MOD** $4 = 1$; $((y+1) * z) = 5 * 2 = 10$; $(x$ **DIV** $(y+1)) = 25$ **DIV** $5 = 5$; Rezultatul final: $1 + 10 - 5 = 6$.

24. Analizăm expresia pas cu pas: $(a$ **DIV** $b) = 30$ **DIV** $6 = 5$; Prima parte: $5 - 2 = 3$; $((b-1)$ **MOD** $(c+1)) = 5$ **MOD** $3 = 2$; $(a$ **MOD** $(b-1)) = 30$ **MOD** $5 = 0$; Rezultatul final: $3 + 2 + 0 = 5$.

25. Analizăm expresia pas cu pas: $((m+1)$ **MOD** $n) = 11$ **MOD** $3 = 2$; $((n-1)*p) = 2 * 3 = 6$; $((m+1)$ **DIV** $p) = 11$ **DIV** $3 = 3$; Rezultatul final: $2 + 6 - 3 = 5$; Deci `m` va avea valoarea 5.

26. $(a$ **MOD** $(b-1)) = 14$ **MOD** $6 = 2$; $((a+1)$ **DIV** $(b-1)) = 15$ **DIV** $6 = 2$; $2 + c = 2 + 2 = 4$; $(2$ **MOD** $(b-1)) = 2$ **MOD** $6 = 2$; Rezultatul final: $2 + 4 * 2 = 10$.

27. $(x*(y+1)) = 4*3 = 12$; $((y+1)$ **DIV** $(z+1)) = 3$ **DIV** $2 = 1$; $((x+1)$ **MOD** $y) = 5$ **MOD** $2 = 1$; Rezultatul final: $12 - 1 + 1 = 12$, deci `x` va avea valoarea 12.

28. $(a$ **DIV** $(b-1)) = 6$ **DIV** $2 = 3$; $(c$ **MOD** $(a+1)) = 2$ **MOD** $7 = 2$; $((b-1)*2) =$

$2 * 2 = 4$; $(a$ **MOD** $(c + 1)) = 6$ **MOD** $3 = 0$; Rezultatul final: $3 + 4 - 0 = 7$.

29. $((u - 1) * v) = 11 * 4 = 44$; $((u - 1)$ **MOD** $w) = 11$ **MOD** $3 = 2$; $(44$ **DIV** $2) = 22$; $((v - 1)$ **MOD** $(w - 1)) = 3$ **MOD** $2 = 1$; Rezultatul final: $22 + 1 = 23$.

30. $(x$ **MOD** $y) = 5$ **MOD** $3 = 2$; $(z$ **DIV** $(y+1)) = 10$ **DIV** $4 = 2$; $((z-1)$ **MOD** $(x-1)) = 9$ **MOD** $4 = 1$; Rezultatul final: $2 + 2 - 1 = 3$.

31. $(x$ **MOD** $(y - 1)) = 7$ **MOD** $3 = 1$; $(z * (x + 2)) = 2 * 9 = 18$; Prima parte: $1 + 18 = 19$; $((x + 2)$ **DIV** $z) = 9$ **DIV** $2 = 4$; Rezultatul final: $19 - 4 = 15$.

32. $(a$ **DIV** $b) = 12$ **DIV** $5 = 2$; $(2 > 3)$ este fals; $((b - 1) < c) = (4 < 3)$ este fals; Prima parte: fals AND fals = fals; $(a$ **MOD** $(b - 1)) = 12$ **MOD** $4 = 0$; $(0 = 2)$ este fals; Rezultatul final: fals OR fals = fals.

33. `signed char` pe 8 biți reprezintă intervalul $[-2^7, 2^7 - 1] = [-128, 127]$.

34. $((k + 2)$ **MOD** $(k + 2)) = 5$ **MOD** $5 = 0$; $((k + 1)$ **DIV** $2) = 4$ **DIV** $2 = 2$; Rezultatul final: $3 + 0 + 2 = 5$.

35. `unsigned long` este reprezentat pe 32 biți **fără** semn, deci maximul e $2^{32} - 1$, corespunzător variantei B.

36. $(x > 5)$ și $(x + y) < 20$ sunt adevărate, deci vor returna True; $(y$ **MOD** $2 = 1) \Rightarrow (5$ **MOD** $2 = 1)$ va returna True, deci (True OR True) $\rightarrow$ True.

37. `signed char` poate stoca valori din intervalul $[-128, 127]$ (fiind tip de dată **cu semn**). Răspunsul este, așadar, nu.

38. $((a+1)$ **DIV** $b) = 9$ **DIV** $3 = 3$; $(b$ **MOD** $c) = 3$ **MOD** $1 = 0$; $((a+1)$ **MOD** $b) = 9$ **MOD** $3 = 0$; Rezultatul final: $3 - 0 + 0 = 3$.

39. `signed short` acceptă valori din intervalul [-32768, 32767] (fiind reprezentare cu semn), deci 200 este valid. Varianta C este cea corectă.

40. $(m$ **MOD** $(n+p)) = 15$ **MOD** $(4+2) = 15$ **MOD** $6 = 3$; $(n$ **DIV** $p) = 4$ **DIV** $2 = 2$; Rezultatul final: $3 + 2 = 5$.

41. Pentru `long long` reprezentat pe minim 64 de biți și cu semn, intervalul va fi $[-2^{63}, 2^{63} - 1]$. Evident, în cazul unei reprezentări fără semn, limita inferioară ar fi fost 0.

42. $(x$ **MOD** $y) = (13$ **MOD** $3) = 1$; $(z * y) = (5 * 3) = 15, (15 > 13)$, deci vom

avea `True` **AND** `True` = `True`; $(x < (z + y))$ => $(13 < (5 + 3)) = (13 < 8) \rightarrow$ `False`; (`True` **OR** `False`) $\rightarrow$ `True`.

43. (b **DIV** a) = 2 **DIV** 1 = 2; $(2 > 0)$ este adevărat; $((c - 1) < (b + 1)) = (2 < 3)$ este adevărat; $(a == 2)$ este fals; Rezultatul final: `True OR (True AND False)` = `True OR False` = `True`.

44. $(x + y) = 16, 16 \leq 15$; $(y - x) = 4, (4 \geq 4)$, deci avem `False` **AND** `True`) $\rightarrow$ `False`; $(x * y) = 60, (60$ **MOD** $2) = 0$. În final, (`False` **OR** `True`) $\rightarrow$ `True`.

45. Expresiile corecte sunt cele care verifică simultan ca x să fie multiplu de 5 și să aparțină intervalului $(a, b]$. Expresia **A** realizează această verificare utilizând negația și operatorii logici pentru a exclude valorile din afara intervalului.

46. Codul verifică dacă un șir este palindrom. Variantele B, C și D nu îndeplinesc condițiile.

47. Codul returnează lungimea maximă a unei subsecvențe a șirului cu suma egală cu 0.

48. Codul returnează numărul de subsecvențe cu mai mult de 1 element, elementele în ordine strict crescătoare a căror sumă este număr prim.

49. Codul compară două metode de rotire a unui vector la stânga cu k poziții. Algoritmul `A` folosește un vector temporar, iar `B` efectuează k rotații succesive folosind funcția Helper. Algoritmul `A` este mai eficient.

50. Codul calculează suma maximă a unei subsecvențe consecutive din vector folosind algoritmul lui Kadane.

51. Codul sortează vectorul folosind funcția `A` și afișează elementele care nu apar de un număr de ori multiplu de k în vector.

52. Codul sortează vectorul folosind funcția `A` și afișează elementele care nu apar de un număr de ori multiplu de k în vector.

53. Codul verifică dacă vectorul este de tip "munte", adică are o secvență strict crescătoare urmată de o secvență strict descrescătoare.

54. Codul verifică dacă vectorul este de tip văle, adică are o secvență strict descrescătoare urmată de o secvență strict crescătoare.

55. Codul determină lungimea celui mai lung prefix comun al celor două șiruri, comparând

caracterele succesive până când întâlnește caractere diferite sau ajunge la sfârșitul unuia dintre șiruri.

56. Codul determină lungimea celui mai lung prefix comun al celor două șiruri, comparând caracterele succesive până când întâlnește caractere diferite sau ajunge la sfârșitul unuia dintre șiruri.

57. Codul calculează produsul dintre produsul elementelor de pe diagonala principală și pe cea secundară. Diagonala principală conține elementele $m[i][i]$, iar diagonala secundară conține elementele $m[i][n-i+1]$.

58. Codul calculează diferența dintre suma elementelor de pe liniile pare și suma elementelor de pe liniile impare din matrice folosind un algoritm recursiv. Indexarea liniilor începe de la 1, astfel încât linia 1 este considerată impară, linia 2 este considerată pară etc.

59. Codul calculează diferența dintre suma elementelor de pe liniile pare și suma elementelor de pe liniile impare din matrice folosind un algoritm recursiv. Indexarea liniilor începe de la 1, astfel încât linia 1 este considerată impară, linia 2 este considerată pară etc.

60. Codul inversează ultimele două linii și ultimele două coloane dintr-o matrice de dimensiuni arbitrare. Algoritmul funcționează prin înlocuirea directă a elementelor, păstrând matricea originală modificată.

61. Codul utilizează un vector de prefix sum pentru a calcula eficient suma elementelor dintr-un interval $[i, j]$ al unui șir de numere întregi.

62. Algoritmul parcurge elementele de deasupra diagonalei principale și verifică dacă fiecare element este egal cu elementul său simetric față de diagonală. Dacă se găsește cel puțin o pereche de elemente care nu respectă condiția $m[i][j] = m[j][i]$, algoritmul returnează **False**. În caz contrar, returnează **True**.

63. Algoritmul returnează numărul de perechi de 2 elemente consecutive de 1 din vectorul `arr`. A. Fals, sunt 7 perechi de 2 elemente consecutive de 1, pe pozițiile (2, 3), (8, 9), (9, 10), (10, 11), (11, 12), (16, 17), (19, 20). Pentru a afla numărul de șiruri binare pentru care se returnează 0, aplicăm următoarea metodă. Pentru $n = 1$, avem 2 posibilități, 0

și 1. Pentru $n = 2$, avem 3 posibilități, 00, 01, 10. Pentru $n = 3$, avem 5 posibilități, 000, 001, 010, 100, 101. Pentru $n = 4$, avem 8 posibilități, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010. Pe caz general, notăm cu $F(n)$ numărul de șiruri binare pentru care se returnează 0, astfel $F(n) = F(n-1) + F(n-2)$, unde $F(1) = 2$, $F(2) = 3$. $F(1) = 2$, $F(2) = 3$, $F(3) = 5$, $F(4) = 8$, $F(5) = 13$, $F(6) = 21$, $F(7) = 34$, $F(8) = 55$, $F(9) = 89$, $F(10) = 144$, $F(11) = 233$. B. Adevărat, pentru $n = 11$, există 233 de șiruri binare distincte pentru care se returnează 0. C. Fals, pentru $n = 6$, există 21 de șiruri binare distincte pentru care se returnează 0. D. Adevarat, pentru $n = 8$, există 55 de șiruri binare distincte pentru care se returnează 0 și $2^8 - 55 = 256 - 55 = 201$ șiruri binare pentru care se returnează o valoare diferită de 0.

64. Analizam expresia pas cu pas: $v[2*v[2*v[3]-1]+1]+v[2*v[7-5]-1]+v[3-v[2*v[2]-3]+6] = v[2*v[2*2-1]+1]+v[2*v[2]-1]+v[3-v[2*2-3]+6] = v[2*v[3]+1]+v[2*2-1]+v[3-v[1]+6] = v[2*2+1]+v[3]+v[3-5+6] = v[5]+v[3]+v[4] = 9+2+7 = 18$

65. Algoritmul `Algo` construiește o matrice auxiliară `x`, în care păstrează doar numerele prime din matricea a, înlocuind restul elementelor cu 0, parcurge toate submatricele $k \times k$ posibile din `x` pentru a calcula suma elementelor acestora și returnează suma maximă dintre toate aceste submatrice.

66. Analizam expresia pas cu pas: $v[2*v[4]$ **MOD** $3]+v[v[7]$ **DIV** $2-1]*v[3-v[2]+1] = v[2*6$ **MOD** $3]+v[5$ **DIV** $2-1]*v[3-1+1] = v[0]+v[2-1]*v[3] = 4+7*9 = 67$ A. Fals. B. Adevărat. C.Adevărat, $v[v[9]*v[4]$ **MOD** $v[5]]+v[v[8]$ **DIV** $v[9]-v[2]]*v[3-v[2]+v[2]] = v[2*6$ **MOD** $3]+v[5$ **DIV** $2-1]*v[3-1+1] = v[0]+v[1]*v[3] = 4+7*9 = 67$. D. Fals, $v[v[8]$ **MOD** $v[5]*3]*v[3]-v[v[9]]-v[0]-v[v[2]$ **DIV** $3]*v[4-v[6]] = v[5$ **MOD** $3*3]*9-v[2]-4-v[1$ **DIV** $3]*v[4-3] = v[2*3]*9-1-4-v[0]*v[1] = 8*9-1-4-4*7 = 72-1-4-28 = 72-33 = 39$

67. Transformăm numărul binar 1010101100 din baza 2 în baza 10, obținem $684_{(10)}$, apoi convertim $684_{(10)}$ în baza 8, rezultând $1254_{(8)}$.

68. Calculăm expresia $2^4 + 2^6 + 2^{10} - 3$ în baza 10, obținem $1101_{(10)}$, apoi convertim $1101_{(10)}$ în baza 2, rezultând $10001001101_{(2)}$.

69. Se transformă termenii expresiei E în baza 10, se calculează rezultatul expresiei, apoi

se fac conversiile în bazele indicate în variantele de răspuns.

70. Algoritmul returnează elementul minim din șirul arr.

71. Algoritmul `ceFace2(x)` este cunoscut ca fiind alternativa de calcula cifrei de control a unui număr, după algoritmul `ceFace1(x)`. În consecință, variantele corecte sunt C și D.

72. La punctul A se face transformarea din baza 2 în baza 10, iar pentru punctele B, C, D, se folosesc conversiile rapide pentru a verifica echivalențele dintre numere.

73. Algoritmul returnează `True` dacă numărul este prim sau 0, altfel `False`. A: Fals. 21 nu este prim. B: Fals. Returnează `True` pentru 0. C: Fals. Returnează `True` pentru 0. D: Adevărat. Returnează `True` pentru $\{0, 2, 3, 5, 7\}$. Răspuns corect: D.

74. Algoritmul calculează numărul de cifre 0 din reprezentarea în baza 6 a numărului n.

75. Calculăm valorile în baza 10: $x = 429_{10} = 429$. $y = 1AD_{16} = 1 \times 16^2 + 10 \times 16 + 13 = 256 + 160 + 13 = 429$. $z = 110101100_2 = 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 256 + 128 + 0 + 32 + 0 + 8 + 4 + 0 + 0 = 428$. Deci, $x = 429, y = 429, z = 428$, de unde avem $x = y \neq z$ și afirmația A este adevărată.

76. Se determină mai întâi, prin parcurgerea din stânga spre dreapta, vectorul `sp`, unde `sp[i]` reprezintă suma maximă a unui subşir neîntrerupt din intervalul $[0, i]$. Apoi, printr-o parcurgere inversă (de la dreapta la stânga), calculează un subşir cu sumă maximă în partea dreaptă. În final, încearcă să combine aceste rezultate pentru a obţine suma maximă a două subşiruri fără elemente comune. Deci, se identifică două secvențe disjuncte cu suma totală maximă, de unde rezultă că varianta B este corectă.

77. Algoritmul `ceFace` este un algoritm recursiv care returnează valoarea maximă între poziția `i` a apelului inițial și poziția `n`, efectuând comparările pe revenirea recursivității. La punctul A, pentru $1 \leq i \leq 8$, maximul este 12. La punctul B, apelul este același ca la A, iar maximul este 12. La punctul C, pentru $2 \leq i \leq 5$, elementul maxim este 3, deoarece elementul 5 nu este luat în considerare, pornind de la poziția 2. La punctul D, afirmația este falsă, deoarece algoritmul returnează maximul doar din secvența de la `i` până la `n`, nu neapărat din întregul vector.

78. Criteriul de divizibilitate cu 7: Dacă din numărul n fără ultima cifră se scade dublul

ultimei cifre, iar numărul rezultat este divizibil cu 7, atunci numărul inițial n este divizibil cu 7.Algoritmul calculează și returnează numărul de numere divizibile cu k din vectorul v, doar dacă $k = 7$. În caz contrar, algoritmul returnează câte elemente din vectorul v respectă proprietatea respectivă.

79. Algoritmul taste(arr, n) returnează suma numerelor neprime din vectorul arr.

80. Algoritmul afișează resturile succesive ale împărțirii lui n la 2 în ordinea lor, dar pentru a obține reprezentarea binară corectă, aceste resturi trebuie citite în ordine inversă. Astfel, afirmațiile A și B sunt corecte. D este corect deoarece complexitatea algoritmului este $\mathcal{O}(\log n)$.

81. Algoritmul returnează suma dintre cel mai mic element mai mare decât primul element din vector și cel mai mare element mai mic decât primul element, dacă acestea există; altfel, returnează primul element. La A, se returnează primul element. La B, $-1 + 32 = 31$. La C, cu $n = 4$, rezultatul este $20 + 6 = 26$. La D, $14 + 17 = 31$. Afirmațiile A, B, D sunt corecte.

82. Algoritmul determină și afişează cel mai mic număr din intervalul $[a, b]$ cu număr maxim de divizori, numărul de divizori al acestuia și numărul de numere cu această proprietate. A: Se afisează 200 12 1. B: Numarul maxim de divizori este 4 pentru numerele $\{6, 8, 10\}$.

83. Pentru a verifica succesiv valorile `36`, `45`, și `64`, vectorul trebuie să fie sortat în ordine crescătoare, iar valoarea căutată trebuie să se afle la poziția corectă pentru ca metoda căutării binare să urmeze acest traseu al comparațiilor. Vectorii A și D respectă aceste condiții.

84. Convertim în baza 10: AB_{16} în baza 10: $A = 10_{10}$, $B = 11_{10}$, deci $AB_{16} = 10 \times 16 + 11 = 171_{10}$. 120_3 în baza 10: $120_3 = 1 \times 3^2 + 2 \times 3^1 + 0 \times 3^0 = 9 + 6 + 0 = 15_{10}$. 120_4 în baza 10: $120_4 = 1 \times 4^2 + 2 \times 4^1 + 0 \times 4^0 = 16 + 8 + 0 = 24_{10}$. Avem deci $E = 171_{10} + 15_{10} - 24_{10} = 171 + 15 - 24 = 186 - 24 = 162_{10}$. Cum 242_8 în baza 10 este $2 \times 8^2 + 4 \times 8 + 2 = 2 \times 64 + 32 + 2 = 128 + 32 + 2 = 162_{10}$, deci variantele corecte sunt A și D.

85. Algoritmul calculează suma tuturor valorilor de 1 din reprezentarea în baza 2. Pentru

$n = 8 = 2^3$:

Număr	Reprezentare binară
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Grupăm reprezentările după formatul: primul cu ultimul, al doilea cu penultimul, etc.

0 și $7 = 000, 111 \quad \Rightarrow \quad$ 3 biți de 1

1 și $6 = 001, 110 \quad \Rightarrow \quad$ 3 biți de 1

2 și $5 = 010, 101 \quad \Rightarrow \quad$ 3 biți de 1

3 și $4 = 011, 100 \quad \Rightarrow \quad$ 3 biți de 1

Observăm că se obțin 4 grupe a câte 3 biți de 1, adică:

$4 \times 3 = 12$ biți de 1pentru toate numerele naturale până la 7. Mai trebuie să adăugăm numărul de biți din reprezentarea lui 8, adică:

$8 = 1000_{(2)} \quad \Rightarrow \quad 12+1 = 13$ biți de 1 $\quad \Rightarrow \quad$ pentru $n = 8 \quad \Rightarrow \quad$ 13. Pe caz general, trebuie să găsim cea mai apropiată putere de 2 de numărul nostru și să aplicăm regula. Pentru $n = 2^k \Rightarrow \frac{n}{2} \times k + 1$.

86. Algoritmul `indigo(n)` calculează și returnează numărul de factori primi distincți din factorizarea numărului n.

87. Algoritmul returnează cifra cu cel mai mare rest la împărțirea la 4 din n. Dacă există mai multe cifre cu același rest, este selectată cifra cea mai semnificativă. A: Corect, pentru $n = 12569$, algoritmul returnează 2. B: Fals, pentru $n = 783031$, algoritmul returnează 7. C: Corect, pentru $n = 2024$ și $n = 2025$, algoritmul returnează 2. D: Fals, algoritmul nu returnează prima cifră divizibilă cu 4.

88. Algoritmul returnează cifra maximă din numărul n care este divizibilă cu 3 și nu

este divizibilă cu 2. Dacă nu există o astfel de cifră, algoritmul returnează -1. Singura afirmație falsă este D, deoarece cifra 3 îndeplinește condițiile algoritmului.

89. Algoritmul switch(a, b) returnează CMMDC dintre a și b, iar case(a, b) returnează CMMMC dintre a și b. Două numere prime între ele au CMMDC egal cu 1. Astfel, variantele corecte sunt A, B, D.

90. Algoritmul returnează maximul din vectorul arr, ducându-l pe prima poziție din vector, efectuând o singură iterație a algoritmului de sortare bubble sort.

91. Algoritmul calculează CMMDC al numerelor din secvența de la poziția i până la n. A: CMMDC(30, 60, 12, 48, 72) = 6. B: CMMDC(14, 42) = 14. C: CMMDC(20, 80, 50) = 10. Complexitatea algoritmului este $O(n \cdot \log m)$, unde n este numărul de elemente, iar m este valoarea maximă din vector. Astfel, varianta corecta este C.

92. Algoritmul `Algo(v, n)` inițializează un vector cu valori descrescătoare pornind de la n până la 1, iar apoi calculează suma valorilor de pe pozițiile impare ale vectorului, afișând rezultatul.

93. Algoritmul `ceFace`(v, n, i, fl, sl) afișează perechea (a, b), unde a și b sunt cele mai mari două elemente distincte din șirul v, cu $a > b$, exceptând cazul în care șirul conține toate elementele nule. Indexarea vectorului se face de la 0. Algoritmul `ceFace`(v, n) afișează toate perechile de elemente cu proprietatea respectivă pentru toate subsecvențele de la i până la n, unde $i \in [0, n]$. A. Adevărat: Apelul este cu $i = 1$. B. Fals: Se afișează și $(3, 0)$ la final. C. Fals: Dacă vectorul are toate elementele nule, algoritmul afișează $(0, 0)$. D. Adevărat: Se ia doar secvența {2, 1, 3, 1}, apelul este cu $i = 3$. Răspuns corect: A, D.

94. Algoritmul verifică dacă numărul n conține doar cifrele 0 și 1 în reprezentarea sa în baza 4. A. Corect. Algoritmul returnează True dacă n are doar cifrele 0 și 1 în baza 4. B. Corect. Un număr cu cifrele 0 și 1 în orice bază b poate fi scris ca sumă de puteri distincte ale lui b. C. Corect. Pentru $n = 806$, $806_4 = 30212_4$, care nu conține doar 0 și 1, deci algoritmul returnează False. D. Fals. Pentru $n = 81$, $81_{10} = 1101_4$, care nu este multiplu de 4, dar algoritmul returnează True. Astfel varianta falsă este D.

95. Algoritmul `Algo(v)` modifică elementele vectorului `v` iterând descendent de la 8 la 1. Pentru fiecare element, dacă indexul este impar, valoarea este incrementată cu 1, iar dacă

este par, valoarea este redusă cu jumătatea indexului. Vectorul rezultat reflectă aceste modificări.

96. Algoritmul `Algo(v, n, e)` caută elementul e în vectorul v și, dacă acesta există, mută toate elementele de după e la începutul vectorului, păstrând ordinea inițială a celorlalte elemente. Dacă elementul nu este găsit, algoritmul returnează `False` și nu modifică vectorul.

Varianta `C` este corectă deoarece implementează corect mutarea elementelor, utilizând un vector temporar pentru a salva elementele din partea finală a vectorului și le reintroduce în pozițiile corecte după ajustare.

În varianta `A` apare o eroare în gestionarea mutării elementelor, ceea ce poate duce la suprascriere incorectă. Varianta `B` introduce o verificare redundantă și nu gestionează corect repoziționarea elementelor. Varianta `D` utilizează un vector temporar, dar mută greșit elementele, ceea ce duce la un vector final incorect.

Astfel, varianta corectă este `C`, deoarece respectă specificațiile problemei și mută elementele în mod corect.

97. Algoritmul `algo(v, n, k)` verifică dacă vectorul v poate fi împărțit în k subsecvențe de lungime egală, fiecare dintre acestea fiind strict crescătoare. Inițial, algoritmul verifică dacă n este divizibil cu k, deoarece doar în acest caz vectorul poate fi împărțit în k secțiuni de lungime egală. Dacă această condiție nu este îndeplinită, algoritmul returnează `False`. Dacă n este divizibil cu k, algoritmul determină lungimea fiecărei subsecvențe, notată cu s, și parcurge fiecare dintre cele k subsecvențe. În cadrul fiecărei subsecvențe, compară fiecare element cu următorul pentru a verifica dacă acestea sunt ordonate strict crescător. Dacă există cel puțin o pereche de elemente care nu respectă această ordine, algoritmul returnează `False`.

Astfel, algoritmul verifică dacă vectorul poate fi împărțit în k subsecvențe egale ca lungime, fiecare dintre acestea fiind strict crescătoare, ceea ce corespunde variantei de răspuns `A`. Varianta `B` este incorectă deoarece enunțul sugerează doar o verificare a ordonării crescătoare, fără a impune ca elementele să fie strict crescătoare. Variantele `C` și `D` sunt incorecte deoarece apelurile indicate nu respectă cerințele algoritmului.

Așadar, singura variantă corectă este `A`.

98. Algoritmul `algo` calculează un număr format printr-o secvență de operații asupra cifrelor din n, ținând cont de condiția specificată și decrementând k când condiția nu este îndeplinită.

99. Algoritmul `transform` calculează un nou vector pe baza produselor elementelor vecine, în funcție de valoarea pozitivă sau negativă a lui m.

100. Algoritmul `f(v, n, t)` determină tripletul de elemente din vectorul v a cărui sumă este cea mai apropiată de valoarea t. Pentru a eficientiza căutarea, vectorul este mai întâi sortat folosind funcția `g(v, n)`. Apoi, algoritmul parcurge vectorul și, pentru fiecare element, folosește doi indici – unul la stânga și unul la dreapta – pentru a găsi suma optimă prin ajustarea pozițiilor acestora.

Dacă suma curentă este mai apropiată de t decât cea salvată anterior, tripletul este actualizat. Dacă suma este prea mică, indicele stâng este crescut, iar dacă este prea mare, indicele drept este redus. Dacă suma exactă este găsită, algoritmul returnează imediat tripletul.

Varianta `A` este corectă deoarece algoritmul caută suma cea mai apropiată de t. Varianta `D` este corectă, deoarece pentru apelul `f([-1, 7, 1, 5, -8], 5, 6)`, vectorul sortat devine $[-8, -1, 1, 5, 7]$, iar tripletul optim este $[-1, 1, 5]$.

Astfel, răspunsurile corecte sunt `A` și `D`.

101. Algoritmul `one(nr)` returnează numărul de divizori ai lui `nr`, iar `two(v, n)` calculează numărul de perechi de elemente din `v` care îndeplinesc una din următoarele două condiții: ambele elemente au număr par de divizori sau unul are număr par de divizori și celălalt are număr impar de divizori. Se returnează `p - cnt - 1`. **A.** Adevărat. 2027 este prim, deci are 2 divizori. **B.** Fals. Se returnează 9 perechi care îndeplinesc una din cele 2 condiții. **C.** Fals. **D.** Adevărat. Un număr care are număr impar de divizori este pătrat perfect.

102. Algoritmul returnează 1 dacă și numai dacă toate cifrele lui n din reprezentarea în baza b sunt impare. A. Fals, $42 = 222_{(4)}$. B. Adevărat, $502 = 1315_{(7)}$.

103. A. Adevărat. B. Fals, instrucțiunea "$k \leftarrow k*b$" trebuie executată după instrucțiunea

"$rez \leftarrow rez + (nr \mod 10) * k$". C. Fals, apelul recursiv trebuie înmulțit cu b nu parametrul b din apel. D. Adevărat.

104. Algoritmul calculează diferența dintre divizorii pari și cei impari ai lui n. A. Adevărat pentru $n = 12$. B. Adevărat, acesta este scopul algoritmului. C. Fals, pentru $n = 2$ returnează 0. D. Fals, 72 are 3 divizori impari și 9 divizori pari.

105. Algoritmul verifică dacă vectorul `x` este periodic cu perioada `p` și dacă lungimea `n` este un multiplu al perioadei `p`. În acest caz, opțiunile A și D sunt corecte.

106. Algoritmul numără aparițiile cifrei d și verifică dacă aceasta apare de mai multe ori decât jumătate din numărul total de cifre ale lui n.

107. Algoritmul convertește numărul n în baza b, parcurge fiecare cifră, și verifică dacă aceasta este divizibilă cu $b-1$. Incrementarea variabilei `count` numără câte astfel de cifre există.

108. Algoritmul parcurge perechile de elemente adiacente din vector, calculează diferența dintre cifrele lor, și ajustează valoarea elementelor conform regulilor date. La final, adună toate elementele rămase și actualizează ultima poziție din vector cu restul împărțirii sumei totale la 10.

109. Algoritmul calculează suma cifrelor pare ale numărului `n`. Dacă numărul nu conține cifre pare, algoritmul returnează `-1`. Pentru `algo(6356)`, suma cifrelor pare (`6` și `6`) este `12`, deci rezultatul final este `12`.

110. Algoritmul combină cifrele numerelor `a` și `b` pe baza regulilor specificate, adunând sau scăzând valorile în funcție de paritatea sumei cifrelor, cu ajustări suplimentare. Valoarea returnată pentru apelul `algo(387, 2349)` este `97`.

111. Algoritmul `f(v, n)` determină suma maximă a unui subset de elemente din vectorul v astfel încât niciun element consecutiv să nu fie inclus în subset. Variantele A, C și D sunt corecte, deoarece acestea respectă condițiile date.

112. Algoritmul `ceFace` verifică dacă toate elementele vectorului sunt pare.

113. Algoritmul modifică vectorul `arr` pe baza formulei date și returnează valoarea de pe poziția y.

114. Algoritmul calculează suma divizorilor numărului n folosind factorizarea. Comple-

xitatea de timp a algoritmului este $O(\sqrt{n})$, datorită faptului că mergem cu k de la 2 la $\sqrt{n}$ cu pasul 1.

115. Algoritmul calculează suma divizorilor numărului n folosind factorizarea. A. Adevarat, suma divizorilor lui 12 este $1 + 2 + 3 + 4 + 6 + 12 = 28$. B. Fals, pentru apelul ceFace

116. Algoritmul verifică folosind căutarea binară dacă numărul n este cub perfect. A. Fals, $n = 8$, $2^3 = 8$, se returnează True. B. Adevărat, $n = 5832$, $18^3 = 5832$, se returnează True. C. Adevărat, în intervalul $(1, 30]$ cuburile perfecte sunt 8 și 27. În interval sunt 29 de valori, deci se returnează Fals pentru $29 - 2 = 27$ valori. D. Adevărat, în intervalul $[9, 513)$ cuburile perfecte sunt: 27, 64, 125, 216, 343, 512. Sunt exact 6 valori în interval pentru care se returnează True.

117. Funcția rearanjează cifrele numărului n astfel: cifrele impare apar în ordine directă, iar cifrele pare în ordine inversă, păstrându-și pozițiile relative.

118. Algoritm calculează cel mai mic multiplu comun (CMMMC) al celor două numere naturale x și y, fără a utiliza formule matematice directe sau diviziuni.

119. Algoritmul `ceFace` construiește numere formate din cifrele lui n repetate și verifică dacă sunt palindromuri. Dacă da, acestea sunt adăugate la suma numerelor speciale și produsul cifrelor speciale este actualizat. Rezultatul este restul împărțirii sumei la produsul cifrelor speciale.

120. Algoritmul `ceFace(n, p)` calculează numărul total de factori de p din factorizarea lui $n!$.

121. Funcția `Cifra(n)` verifică dacă toate cifrele lui n sunt distincte și dacă suma divizorilor săi este egală cu n. Dacă suma divizorilor diferă de n, funcția reduce suma la o singură cifră și returnează această valoare.

122. Funcția `Algo(n)` construiește o permutare a numerelor $0, 1, ..., n - 1$ pe baza unei reguli definite prin operații binare. Funcția `ceFace (n)` verifică fiecare element al permutării și numără câte elemente respectă condiția $(p[i]$ **AND** $i) = i$.

123. Algoritmul verifică dacă n este pătrat perfect folosind metoda diferențelor consecutive. Un număr n este pătrat perfect dacă diferența dintre orice două pătrate perfecte

consecutive este constantă și egală cu $2d + 1$, unde d reprezintă numărul pătrat. De exemplu: $1 = 1$, $4 = 1 + 3$, $9 = 1 + 3 + 5$, $16 = 1 + 3 + 5 + 7$, $25 = 1 + 3 + 5 + 7 + 9$, etc. A. Adevărat. B. Adevărat. C. Adevărat, cel mai mic număr pătrat perfect din intervalul $[71, 734]$ este 9^2, iar cel mai mare este 27^2, deci există $27 - 9 + 1 = 19$ valori pentru care se returnează True. D. Fals, în mulțimea $\{73, 9, 442, 841, 576, 962\}$, numerele pătrate perfecte sunt $9 = 3^2$, $841 = 29^2$, $576 = 24^2$, deci se returnează True pentru 3 valori și False pentru 3 valori.

124. Algoritmul calculează numărul de secvențe distincte de lungime k dintr-un număr n. El parcurge fiecare secvență de lungime k, verifică dacă toate cifrele sunt unice, și le numără dacă îndeplinesc această condiție.

125. Algoritmul returnează numărul de operații posibile pe care le poate efectua asupra lui n prin utilizarea operațiilor pe biți (`AND` și `SHL`), ținând cont de constrângerile impuse de mască și de poziționarea biților. Pentru $n = 12$, algoritmul identifică 9 astfel de operații valide. Varianta A este corectă, deoarece descrie corect obiectivul algoritmului, iar varianta C este corectă, deoarece rezultatul calculat pentru $n = 12$ este 9.

126. Algoritmul determină CMMMC al numerelor a și b. A: Corect, CMMMC(24, 36) = 72. B: Fals, pentru $a = 52$ și $b = 14$, rezultatul este 364. C: Fals, algoritmul returnează CMMMC, nu CMMDC. D: Corect, modificarea ar face ca algoritmul să calculeze CMMDC deoarece $CMMDC \cdot CMMMC = a \cdot b$.

127. Bucla interioară rulează de $O(\log_3 n)$ ori, determinată de numărul de puteri ale lui 3 mai mici decât n. Bucla exterioară rulează de aproximativ $\frac{2n}{3}$ ori, adică $O(n)$, datorită constantelor care nu se iau în calcul. Complexitatea totală este $O(n \cdot \log_3 n)$.

128. Algoritmul sortează tabloul v și apoi, pentru fiecare pereche (i, j), caută binar un k care satisface condiția $v[i] + v[j] > v[k]$, deci tripletele i, j, k reprezintă indicii lungimilor de laturi care pot forma triunghiuri. Algoritmul are complexitatea $O(n^2 \log n)$, deci singura variantă corectă este D.

129. $T(n) = \sum_{k=0}^{n} 3^k \Rightarrow T(n) = \frac{3^{n+1}-1}{3-1} = \frac{3^{n+1}-1}{2} \Rightarrow T(n) = O(3^n)$

130. $T(n) = 2 \cdot T(n-1) + 2 \Rightarrow T(n-1) = 2 \cdot T(n-2) + 2 \Rightarrow T(n) = 2 \cdot (2 \cdot T(n-2) + 2) + 2 \Rightarrow T(n) = 4 \cdot T(n-2) + 4 + 2 \Rightarrow T(n) = 4 \cdot T(n-2) + 6$ Continuând în acest mod, observăm că:

$T(n) = 2^k \cdot T(n-k) + 2 \cdot (2^{k-1} - 1)$. Când $(k = n-1) : T(n) = 2^{n-1} \cdot T(1) + 2 \cdot (2^{n-2} - 1) \Rightarrow$ $T(n) = 2 \cdot (2^{n-1} - 1) \Rightarrow T(n) = 2^n - 2 \Rightarrow T(n) = O(2^n)$

131. Dacă $n = 1$, algoritmul returnează 1. Altfel, calculează m ca fiind n împărțit la 2 și verifică paritatea lui i. Dacă $n = 1, T(1) = O(1)$. Dacă n $\neq$ 1, algoritmul face un apel recursiv cu n înjumătățit și efectuează o operație de adunare sau scădere: $T(n) = T\left(\frac{n}{2}\right) + O(1)$. Această relație de recurență poate fi rezolvată folosind metoda substituției: $T(n) = T\left(\frac{n}{2}\right) + O(1)$, $T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + O(1)$, $T(n) = T\left(\frac{n}{4}\right) + O(1) + O(1)$, $T(n) = T\left(\frac{n}{8}\right) + O(1) + O(1) + O(1)$. Continuând în acest mod, observăm că $T(n) = T\left(\frac{n}{2^k}\right) + k \cdot O(1)$. Când $\frac{n}{2^k} = 1$, $k = \log_2 n$: $T(n) = T(1) + \log_2 n \cdot O(1)$, $T(n) = O(1) + O(\log n)$, $T(n) = O(\log n)$. Prin urmare, complexitatea de timp a algoritmului este $T(n) = O(\log n)$.

132. Complexitatea de timp a algoritmului este $O(\log n)$, datorită instrucțiunii "$ind \leftarrow ind \,\text{DIV}\, 5$" care este pusă în while-ul interior.

133. Algoritmul `CountSort` este cel mai eficient în acest caz, deoarece valorile posibile sunt doar cifre, iar algoritmul funcționează în timp liniar $O(n)$.

134. Dacă șirul a este sortat, putem construi un tablou de sume parțiale în $O(n)$. Apoi, pentru fiecare q, folosind căutarea binară (complexitate $O(\log n)$), putem găsi cel mai mare index i pentru care suma primelor i elemente nu depășește q. Astfel, răspunsul pentru fiecare interogare se poate obține în $O(\log n)$, ceea ce face ca subprogramul de căutare să aibă complexitatea $O(\log n)$.

135. Funcția parcurge vectorul `arr` într-un singur ciclu și compară fiecare element cu `y`. Astfel, complexitatea este $O(n)$, iar funcția returnează numărul elementelor mai mari decât `y`. Dacă toate elementele sunt mai mici decât $y + 1$, rezultatul este 0. Optimizarea la $O(\log n)$ nu este posibilă deoarece fiecare element trebuie evaluat.

136. Algoritmul rotește vectorul `v` la dreapta cu $k\%n$ poziții, are complexitate $O(n)$, folosește un vector auxiliar pentru a stoca rotația temporară și modifică direct vectorul original `v`.

137. Algoritmul implementează o singură iterație din bubble sort, parcurgând vectorul de la stânga la dreapta și efectuând interschimbări între elementele adiacente dacă nu sunt

în ordine crescătoare. Complexitatea este $O(n)$, iar numărul returnat reprezintă câte interschimbări au fost efectuate. Totuși, nu garantează sortarea completă a vectorului după o singură iterație.

138. Algoritmul implementează metoda Kadane pentru găsirea sumei maxime a unei subsecvențe continue dintr-un vector. Are complexitate $O(n)$, deoarece vectorul este parcurs o singură dată. Dacă toate elementele sunt negative, funcția va returna cel mai mare element din vector. Algoritmul nu modifică vectorul original.

139. Algoritmul determină numărul total de divizori ai unui număr `n`, iterând până la radicalul său, ceea ce asigură o complexitate $O(\sqrt{n})$. Pentru $n = 1$, există un singur divizor, iar pentru un număr prim, există exact doi divizori: 1 și numărul însuși.

140. Algoritmul este derivat din calcularea factorialului, însă condiția `i % k == 0` impune constrângerea de a înmulți doar multiplii lui `k` din intervalul $[1, n]$. Dacă linia `if(i % k == 0)` ar lipsi, funcția ar calcula factorialul lui `n`. De asemenea, rezultatul este divizibil cu `k` dacă și numai dacă `n` $\geq$ `k`.

141. Algoritmul verifică dacă un număr este prim, parcurgând divizorii posibili până la `n / 2`, ceea ce îl face să aibă complexitate $O(n)$. O implementare mai eficientă ar folosi un algoritm cu complexitate $O(\sqrt{n})$, iterând până la rădăcina pătrată a numărului.

142. Algoritmul folosește tehnica sumelor parțiale: mai întâi transformă vectorul astfel încât fiecare element $v[i]$ să devină suma primelor i elemente din vectorul original, apoi calculează suma elementelor dintre indicii $[k+1, n]$ prin diferența $v[n] - v[k]$. Rezultatul poate fi negativ dacă vectorul conține numere negative. Complexitatea algoritmului este $O(n)$. Afirmația D este falsă deoarece chiar și pentru un vector sortat crescător cu elemente pozitive, suma $v[n] - v[k]$ ar putea fi zero dacă toate elementele de la poziția $k+1$ la n sunt zero.

143. Algoritmul calculează valoarea lui $n!$, stocând cifrele acestui număr în tabloul a. Fiind vorba de un număr foarte mare (1000! având 2568 cifre), operațiile sunt realizate pe tabloul a, la fiecare pas ținându-se cont de variabila cr, variabilă de carry.

144. Algoritmul identifică vârfurile din vector, adică punctele unde o secvență crescătoare este urmată de una descrescătoare. Complexitatea este $O(n)$, deoarece parcurge vectorul

o singură dată. Pentru un vector constant, algoritmul nu găsește vârfuri și returnează 0. Pentru un vector strict crescător, rezultatul este 1.

145. Algoritmul determină toți divizorii lui `n` și calculează suma cifrelor acestora. Pentru `n = 1`, singurul divizor este 1, deci suma returnată este 1. Complexitatea nu este optimizată pentru că parcurge toate numerele de la 1 la `n`, iar pentru numere prime rezultatul nu este garantat să fie `suma cifrelor lui n` + 1.

146. Algoritmul determină lungimea celei mai lungi secvențe de elemente consecutive din vector care formează o progresie aritmetică, având complexitate $O(n)$. Pentru un vector constant(valori), lungimea progresiei este egală cu dimensiunea vectorului. Pentru un vector cu elemente distincte, nu este garantat să returneze 2, deoarece pot exista progresii mai lungi.

147. Algoritmul parcurge toate perechile posibile de elemente pentru a găsi distanța maximă între două elemente egale, având o complexitate de $O(n^2)$. Pentru un vector cu elemente distincte, nu există elemente egale, deci rezultatul este 0. Pentru un vector constant(valori), rezultatul este $n - 1$, deoarece toate elementele sunt egale.

148. Algoritmul folosește tehnica ferestrei glisante pentru a calcula eficient suma maximă a `k` elemente consecutive. Complexitatea algoritmului este $O(n)$, iar pentru `k = 1` returnează maximul din vector, iar pentru `k = n` returnează suma tuturor elementelor.

149. Algoritmul extrage cifrele pare dintr-un număr și le recompune în ordinea inițială folosind două inversări succesive. Pentru un număr care conține doar cifre impare, rezultatul este 0. Complexitatea este $O(\log n)$, deoarece numărul de operații este proporțional cu numărul de cifre din `n`.

150. Algoritmul parcurge vectorul o singură dată, având complexitate $O(n)$, și determină lungimea celei mai lungi secvențe de numere pozitive consecutive. Dacă vectorul conține doar elemente negative, funcția returnează 0. Pentru un vector cu toate elementele pozitive, funcția returnează n, nu $n/2$.

151. Algoritmul afișează "săriturile" lui n, adunând valoarea lui d, daca suma lor este deasupra varfului p, sau scăzând valoarea lui d în caz contrar.

152. Algoritmul parcurge recursiv vectorul de biți b, iar când se întâlnește un 1, se fac

două apeluri recursive modificând ℓ prin $\ell + 3$ și $\ell \cdot 2$, combinând rezultatele cu $c(x, y)$ (care dă produsul dacă x este impar și suma dacă este par).

Vectorii $[0, 0, 1, 1]$ și $[1, 1, 0, 0]$ produc 76 deoarece în primul caz se construiește arborele doar pentru ultimele două poziții iar în al doilea caz pentru primele două poziții, dar ambele cazuri duc la aceeași valoare finală prin diferite căi de combinare a operațiilor de adunare și înmulțire.

153. Algoritmul `g(x)` afișează suma cifrelor lui x, urmată recursiv de aceleași operații pentru *x DIV 2*, intercalate cu valoarea *x + 3*.

154. Algoritmul procesează un șir prin împărțirea sa recursivă în două jumătăți, verificând dacă mijlocul curent este o vocală; dacă da, concatenează rezultatele obținute din procesarea recursivă a jumătăților cu un separator "*". Varianta corectă pentru a procesa partea dreaptă este `comp(s, a + m, l - m)`, deoarece partea dreaptă începe de la indicele $a + m$ și are lungimea $l - m$.

155. Algoritmul `algo(v, n)` determină suma maximă posibilă a unei subsecvențe de elemente neadiacente dintr-un vector, utilizând funcția auxiliară `g(a, b)` pentru a obține maximul dintre două valori.

156. Funcția `f` calculează suma maximă a unui traseu de la stânga-sus la dreapta-jos, alegând între deplasările spre dreapta și în jos.

157. Algoritmul este recursiv și determină secvența de caractere cbazată pe condițiile date.

158. Algoritmul calculează pentru fiecare apel valoarea corespunzătoare pe baza condițiilor impuse.

159. Algoritmul `f` calculează rezultatul pe baza împărțirii intervalului în două subintervale și combină rezultatele folosind operații condiționate de paritatea lungimii intervalului.

160. Algoritmul generează o matrice în care fiecare element este calculat în funcție de poziția anterioară sau de elementul ultim din rândul anterior.

161. Algoritmul `f` calculează diferența dintre suma elementelor de pe pozițiile pare și suma elementelor de pe pozițiile impare din vectorul `v`.

162. Algoritmul `algo` parcurge fiecare cifră a numărului și adaugă la rezultat doar cifrele

impare, ignorând pe cele pare, ceea ce face varianta `C` corectă.

163. Observăm că la fiecare apel se afișează 3 valori și se fac 2 autoapeluri, iar când n devine 0 algoritmul se oprește. De aici putem deduce că oricare ar fi valoarea lui n, numărul de valori afișate va fi un multiplu de 3. Pentru a afla numărul de valori afișate nu ne interesează care sunt valorile doar numărul lor. D. Adevărat, `ceFace(2)` = 1 `ceFace(1)` 3 `ceFace(1)` 2 = 1 0 `ceFace(0)` 2 `ceFace(0)` 1 3 0 `ceFace(0)` 2 `ceFace(0)` 1 2 = 1 0 2 1 3 0 2 1 2.

`ceFace(1)` = 3 valori

`ceFace(2)` = 9 valori

`ceFace(3)` = 3 valori `ceFace(1)` `ceFace(2)` = 3 + 3 + 9 = 15 valori

`ceFace(4)` = 3 valori `ceFace(2)` `ceFace(3)` = 3 + 9 + 15 = 27 valori

`ceFace(5)` = 3 valori `ceFace(2)` `ceFace(4)` = 3 + 9 + 27 = 39 valori

`ceFace(6)` = 3 valori `ceFace(3)` `ceFace(5)` = 3 + 15 + 39 = 57 valori

Deducem că pentru `ceFace(6)` se vor afișa 57 de valori și pentru `ceFace(5)` se vor afișa 39 de valori.

164. Algoritmul începe prin a afișa ultima cifră, ultima cifră împărțită la 3, și tot așa pana când ajunge la 1. După aceea algoritmul generează o secvență de tipul [6661332444222666333], atât pentru 3003, cât și pentru 9009, respectiv [3344888296148444667], pentru 6006. Ultima cifră nu afectează ultimul număr afișat, ci doar primele numere.

165. Opțiunea C este corectă deoarece pentru $x = 23015$ avem $23015 \mod 5 = 0$, deci $g(23015) = 2 \cdot g(230)$ și, întrucât $g(230) = 2 \cdot g(2) = 4$, rezultă $g(23015) = 2 \cdot 4 = 8$. Algoritmul procesează recursiv numărul x: dacă $x = 0$ returnează 1, dacă $x < 10$ returnează x, iar pentru $x \geq 10$ verifică dacă x este divizibil cu 5 pentru a decide între a returna $2 \cdot g(x \,\texttt{DIV}\, 100)$ sau $g(x \,\texttt{DIV}\, 100) + g(x \,\texttt{MOD}\, 10)$.

166. Algoritmul `f` parcurge recursiv valorile `i` și `j`, și scrie caracterele `A`, `B`, sau `C` în funcție de condițiile impuse asupra lui `n`, `i` și `j`.

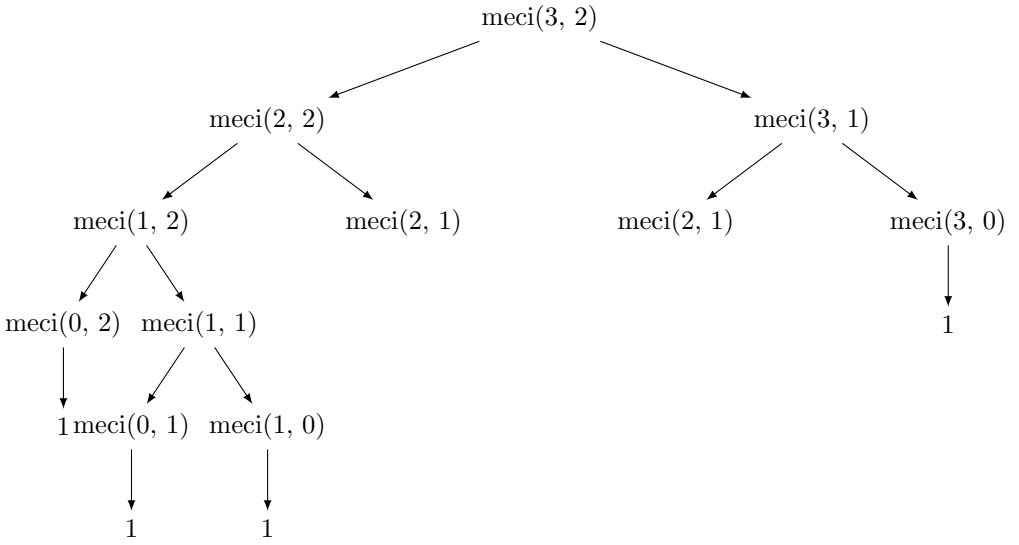
167. Algoritmul `f` calculează valorile bazate pe relații recursive, alternând între adunare și înmulțire în funcție de paritatea lui `n`.

168. Algoritmul `Y(a, b)` returnează a^b, în complexitate liniară, deoarece nu se salvează

apelul `Y(a, b DIV 2)` într-o variabilă și se apelează de 2 ori. Algoritmul `X(n, i, j)` returnează un număr construit fără nicio logică. A. Fals, pentru `X(369, 3, 5)`, $p = 100$ și se returnează $369+9\cdot100+3\cdot100+5 = 369+900+300+5 = 1574$. B. Adevărat, pentru `X(369, 2, 3)`, $p = 10$ și returnează $369 + 9 \cdot 10 + 36 \cdot 10 + 3 = 369 + 90 + 360 + 3 = 822$. C. Adevărat, $6^3 = 216$. D. Fals, returnează a^b, dar complexitatea este liniară.

169. Algoritmul calculează un rezultat bazat pe operații recursive, alternând între adunare, scădere, înmulțire și resturi, în funcție de anumite condiții. Pentru apelul `f([2, 8, 7, 9, 2, 8], 6)`, valoarea finală returnată este `4`.

170. Algoritmul returnează numărul de moduri distincte de a ajunge de la scorul 0-0 la scorul $a - b$ într-un meci de fotbal.



Din acest arbore pe revenire, însumăm rezultatele obținute, iar în momentul în care vedem că un apel se repetă de mai multe ori îl calculăm o singură dată și ne folosim pe revenire de rezultatul obținut pentru același apel identic. Astfel obținem că $\text{meci}(1,1) = 2$, $\text{meci}(1,2) = 3$, $\text{meci}(2,1) = 3$, $\text{meci}(2,2) = 6$, $\text{meci}(3,0) = 1$, $\text{meci}(3,1) = 4$, $\text{meci}(3,2) = 10$. Aplicăm aceeași metodă și pentru $\text{meci}(5,5)$ și vom obține 252. Complexitatea algoritmului este $O(2^{a+b})$.

171. Pentru n = 7 se afiseaza: 25140201172414020116130300512030041402011.

Pentru n = 6 se afiseaza: 2414020116130300512030041402011.

Pentru n = 5 se afiseaza: 130300512030041402011.

172. Algoritmul `sun(i)` are o complexitate timp $T(i) = O(\log i)$, datorită faptului că valoarea lui i este împărțită la 2 la fiecare apel recursiv.Algoritmul `moon(n)` are o complexitate de timp $T(n) = O(\log n \cdot \log n)$, datorită faptului că bucla `while` se execută de $O(\log n)$ ori, iar în fiecare iterație se apelează funcția `sun(j)`, care are o complexitate de $O(\log j)$. Deoarece j crește exponențial, complexitatea totală a algoritmului `moon(n)` este $O(\log n \cdot \log n)$.

173. A. Adevărat, `sun(40)` returnează valoarea 5. B. Fals, `moon(128)` returnează valoarea 21. C. Fals, `sun(7)` returnează valoarea 2 și `moon(7)` returnează valoarea 3. D. Adevărat, `moon(2048)` returnează valoarea 55.

174. Algoritmul implementează o metodă recursivă pentru a calcula al n-lea termen din șirul lui Fibonacci, având complexitate exponențială $O(2^n)$. Rezultatul pentru $n = 5$ este 8. Totuși, algoritmul poate fi optimizat utilizând programare dinamică pentru a reduce complexitatea la $O(n)$.

175. Algoritmul are complexitatea: $T(n) = O(4^{\log_2(n)}) = O(2^{2\log_2(n)}) = O(2^{(\log_2(n))^2}) = O(n^2)$.

176. Algoritmul `S` parcurge recursiv cifrele lui n și aplică reguli diferite în funcție de restul împărțirii fiecărei cifre la 3: adaugă pătratul cifrei dacă este divizibilă cu 3, scade cifra dacă restul este 1, și adaugă dublul cifrei dacă restul este 2.

177. Algoritmul `ceFace(n, b)` rearanjează cifrele lui n în baza b astfel încât cifrele impare apar în față, iar cifrele pare apar la final, păstrând poziția relativă.

178. Algoritmul `Exp` calculează cel mai mare divizor comun (CMMDC) al numerelor a și b folosind metoda binară, cunoscută și sub numele de Algoritmul lui Exp. Această metodă oferă o alternativă eficientă la algoritmul clasic al lui Euclid, eliminând necesitatea operațiilor de împărțire și utilizând doar operații mai rapide precum scăderea, împărțirea la 2 și înmulțirea.

Dacă unul dintre numere este zero, algoritmul returnează celălalt număr, deoarece orice număr nenul are drept divizor comun cu zero propria sa valoare. Dacă ambele numere sunt pare, se aplică regula conform căreia CMMDC-ul acestora este de două ori CMMDC-ul valorilor lor înjumătățite. Dacă doar unul dintre numere este par, se elimină factorul

de două al acestuia prin împărțirea la 2, fără a modifica celălalt număr. În cazul în care ambele numere sunt impare, se efectuează o scădere și o diviziune la 2 pentru a aduce problema la un caz mai simplu.

Pentru exemplul `Exp(48, 36)`, se aplică regulile algoritmului pas cu pas. Deoarece ambele numere sunt pare, se împarte fiecare la 2 și se multiplică rezultatul final cu 2. Prin aplicări succesive ale regulilor, algoritmul ajunge la 12, care este CMMDC-ul lui 48 și 36. Algoritmul este echivalent cu cel al lui Euclid, garantând același rezultat pentru orice pereche de numere. În schimb, pentru apelul `Exp(28, 7)`, aplicând metoda, rezultatul corect nu este 4, ci 7, ceea ce face afirmația D incorectă.

179. Algoritmul numără recursiv elementele pare din vector, având complexitate $O(n)$ de timp, însa $O(2 * n)$ este echivalent în notația Big-O. Pentru un vector vid returnează 0, iar algoritmul poate fi rescris iterativ, păstrând aceeași complexitate $O(n)$.

180. Algoritmul calculează suma maximă pe un drum de la $(0, 0)$ la $(n-1, n-1)$, explorând toate căile posibile și având o complexitate exponențială $O(2^n)$. Este posibilă optimizarea sa folosind programare dinamică pentru a evita recalculările. Permite doar deplasări în jos și la dreapta, conform constrângerilor algoritmului.

181. Algoritmul calculează numărul de factori primi ai unui număr `n` folosind o abordare recursivă eficientă. Complexitatea este $O(\sqrt{n})$, deoarece divizorii sunt testați doar până la rădăcina pătrată a lui `n`. Pentru numere prime, algoritmul returnează 1.

182. Algoritmul implementează o metodă recursivă de căutare binară, având complexitate $O(\log n)$. Funcționează corect pe vectori sortați crescător (elementele se pot repeta). Returnează o valoare booleană care indică dacă elementul `x` există în vector, nu returnează poziția acestuia.

183. Algoritmul calculează minimul dintre trei numere. Funcția `combine(x,y)` calculează minimul dintre două numere folosind formula $(a + b - |a - b|)/2$. Pentru valorile date $(8, 5, 6)$, `combine(8,5)` returnează 5, iar apoi `combine(5,6)` returnează tot 5. Astfel, orice configurație de parametrii (a,b,c); (b,a,c), etc. va returna același rezultat, astfel răspunsul $\boxed{\text{C}}$ este fals. Dacă $|a| = |b|$, atunci b ar putea fi mai mic decât a, (ex: $a = 5, b = -5, c = 10$) astfel răspunsul $\boxed{\text{D}}$ este fals și el.

184. Algoritmul afișează valoarea ultimelor două cifre ale numărului n (n **MOD** 100), apelează recursiv funcția pentru partea întreagă a împărțirii lui n la 10 (n **DIV** 10), iar apoi afișează din nou ultimele două cifre ale fiecărui număr obținut.

185. Algoritmul afișează în ordine descrescătoare valorile calculate ca $n - 2$ până la 2, printr-o parcurgere recursivă, și apoi, în ordine crescătoare, valorile calculate ca $n + 2$ corespunzătoare fiecărei etape a recursiei.

186. Algoritmul `spirala` combină prima și ultima cifră ale unui număr în funcție de direcția specificată de parametrul `sens` și aplică recursiv aceeași operație pe restul cifrelor.

187. Algoritmul este o funcție recursivă care afișează pentru fiecare apel valoarea dublului parametrului de intrare urmat de un simbol "!", apoi apelează recursiv funcția pentru fiecare număr de la 1 la $n - 1$, adăugând simbolul "*" după fiecare apel. Structura rezultată reflectă procesul de recursivitate și ordinea apelurilor.

188. Algoritmul `count` este utilizat pentru a determina câte combinații de monede dintr-un set dat pot forma suma dorită. Metoda folosită este recursivă, având două posibilități pentru fiecare monedă: fie este utilizată în formarea sumei, fie nu este utilizată.

Varianta `D` este cea corectă deoarece respectă structura clasică a problemei de numărare a combinațiilor de monede. Dacă suma devine 0, înseamnă că am găsit o combinație validă și returnăm 1. Dacă numărul de monede disponibile devine 0 sau suma devine negativă, returnăm 0, deoarece nu se mai poate forma suma dorită. Algoritmul explorează două opțiuni: folosirea monedei curente în sumă (scăzând valoarea acesteia din `s`) și neutilizarea acesteia (trecând la următoarea monedă). Aceste două posibilități sunt combinate pentru a obține numărul total de combinații posibile.

189. Algoritmul `g(v, n, t)` verifică dacă există o submulțime a vectorului `v` care să aibă suma exact egală cu `t`. Acesta folosește recursivitatea pentru a explora toate combinațiile posibile ale elementelor vectorului, având două opțiuni pentru fiecare element: fie este inclus în submulțimea curentă, fie este exclus.

Dacă suma dorită `t` devine 0, atunci s-a găsit un subset valid și funcția returnează `True`. Dacă dimensiunea vectorului devine 0 înainte ca suma să fie atinsă, înseamnă că nu mai

există elemente disponibile pentru a construi suma, deci funcția returnează `False`. În fiecare pas, algoritmul încearcă să includă ultimul element al vectorului și să verifice dacă se poate forma suma scăzând această valoare din `t`. În paralel, se verifică și cazul în care elementul este exclus, menținând suma dorită neschimbată.

Varianta de răspuns `B` este corectă deoarece descrie exact funcționalitatea algoritmului, confirmând că acesta determină existența unui subset care atinge suma `t`. Varianta `C` este de asemenea corectă, deoarece apelul `g([3, 34, 4, 12, 5, 2], 6, 9)` returnează `True`, existând o combinație validă de elemente care formează suma 9.

190. Algoritmul `neon(x, k)` afişează primele k elemente ale vectorului x cu spaţiu între ele şi un `NewLine`. Algoritmul `xenon(x, n, p)`, dacă $x[0] = 0$ şi $p = 1$ la apelul iniţial, atunci va afişa toate soluţiile submulţimilor cu elemente de la 1 până la n. Iniţializarea lui i cu $i \leftarrow x[p-1]+1$ asigură că ordinea elementelor în soluţie să fie crescătoare. Se va afişa soluţia la fiecare atribuire a lui i în vectorul x pe poziţia p astfel se vor afişa soluţii de lungime 1, 2, ..., n. A. Adevărat, dacă $x[0]$ este iniţializat cu valoarea 2 înseamnă că posibilităţile de alegere a elementelor sunt de la 3 la n, astfel vor fi $2^{(n-2)} - 1$ soluţii afişate. B. Adevărat, pentru $n = 5$, primele 12 soluţii vor fi: 1, 1 2, 1 2 3, 1 2 3 4, 1 2 3 4 5, 1 2 3 5, 1 2 4, 1 2 4 5, 1 2 5, 1 3, 1 3 4, 1 3 4 5. C. Fals, pentru $n = 4$, primele 6 soluţii vor fi: 1, 1 2, 1 2 3, 1 2 3 4, 1 2 4, 1 3. D. Fals, pentru $n = 3$, algoritmul `xenon(x, 3, 1)` se va autoapela de 6 ori, nu se ia în considerare apelul iniţial.

191. Algoritmul generează toate permutările cu condiția ca niciun element din soluție să nu fie egal cu poziția pe care se află. Pentru apelul `ceFace(arr, 1, 4)`, soluțiile afișate sunt:

1. 2 1 4 3∗,	4. 3 1 4 2∗,	7. 4 1 2 3∗,
2. 2 3 4 1∗,	5. 3 4 1 2∗,	8. 4 3 1 2∗,
3. 2 4 1 3∗,	6. 3 4 2 1∗,	9. 4 3 2 1∗,

Pentru apelul `ceFace(arr, 1, 5)`, primele zece linii afișate sunt:

1. 2 1 4 5 3∗,	4. 2 3 4 5 1∗,	7. 2 4 5 1 3∗,	10. 2 5 4 1 3 ∗ .
2. 2 1 5 3 4∗,	5. 2 3 5 1 4∗,	8. 2 4 5 3 1∗,	
3. 2 3 1 5 4∗,	6. 2 4 1 5 3∗,	9. 2 5 1 3 4∗,	

192. Algoritmul explorează toate submulțimile posibile utilizând backtracking, verificând suma elementelor și generând submulțimile care respectă condițiile impuse. A 8-a submulțime generată este $\{5, 24, 11\}$.

193. Algoritmul `four(c, l, n)`, la apelul `four(0, 0, n)` generează și afișează toate numerele de n cifre care sunt formate doar din cifre prime. A. Adevărat, există 4 cifre prime, 2, 3, 5, 7, iar lungimea numărului format este n, deci se vor afișa 4^n valori. B. Adevărat, cifrele 2, 3, 5, 7 se regăsesc în mulțimea {2, 3, 5, 7, 9}, nu trebuie să se folosească și 9 ca afirmația să fie adevărată. C. Fals, nu afișează numere prime, ci numere formate din cifre prime. D. Adevărat, algoritmul afișează numere de n cifre, iar cifrele prime au exact 2 divizori (1 și cifra însăși).

194. Algoritmul `f(e)` determină numărul de 1 din reprezentarea binară a unui număr și returnează adevărat dacă acest număr este impar. Algoritmul `g(arr, a, b)` returnează true dacă toate numerele din vectorul arr au număr impar de biți 1 în reprezentarea lor în baza 2, false altfel.

195. Algoritmul verifică dacă toate elementele din intervalul indicilor $[ts, td]$ au număr par de cifre, folosind Divide et Impera. A: Adevarat, toate elementele au 2 cifre. B: Fals, Toate elementele au 3 cifre, 3 e impar. C: Fals, Se parcurge doar secvența de la poziția 2 până la poziția 6, doar elemente cu 2 și 4 cifre, returnează True. D: Fals, verifică dacă au număr de cifre par, nu dacă sunt pare.

196. Algoritmul `star(m, n)` returnează cmmdc dintre m și n. Algoritmul `sky(x, y, arr)` folosește Divide et Impera pentru a calcula cmmdc dintre elementele din intervalul indicilor $[x, y]$ din vectorul arr. A: Adevărat. B: Adevărat, se ia doar secvența $[10, 20, 30, 40]$. C: Fals, complexitatea este $O(n \cdot \log(n))$. D: Fals, algoritmul returnează cmmdc dintre m și n, chiar dacă m și n sunt prime.

197. Algoritmul returnează suma elementelor din vectorul `arr`, folosind Divide Et Impera. A. Fals, suma este 15. B. Adevărat. C. Adevărat. D. Fals, complexitatea de timp este $O(n)$.

198. Algoritmul returnează suma elementelor de pe coloana a doua din matrice, folosind Divide Et Impera. A. Fals, $23 + 17 + 8 = 48$. B. Adevărat, $22 + 9 = 31$. C. Fals, $13 + 4 + 6 = 23$. D. Adevărat, $18 + 5 + 3 + 1 + 4 = 31$.

199. Algoritmul returnează `True` dacă există în secvența determinată de indicii a și b un număr care conține un număr par de cifre (sau 0), iar `False` în caz contrar.

200. Algoritmul folosește tehnica Divide Et Impera și verifică dacă toate elementele din secvența determinată de indicii $[il, rl]$ au suma cifrelor cu proprietatea X impară. Proprietatea X: Pentru un număr x se verifică la fiecare pas dacă numărul este divizibil cu 3, dacă da, ultima cifră se adaugă la o sumă, apoi se renunță la ultima cifră și tot așa. Atenție, nu se verifică ca cifra să fie divizibilă cu 3 ci tot numărul la momentul respectiv.

201. Algoritmul împarte matricea în 4 submatrice pentru a găsi maximul folosind Divide et Impera. Complexitatea este $O(nm)$, însă, din cauza apelurilor recursive, algoritmul este mai încet decât unul iterativ făcând parte din aceeași clasă de complexitate.

202. Algoritmul utilizează Divide et Impera pentru a număra perechile de elemente cu sumă dată `k`, funcționând corect doar pe vectori sortați crescător. Complexitatea este mai mare decât $O(n)$ din cauza recursivității.

203. Algoritmul implementează o metodă eficientă pentru determinarea minimului și maximului dintr-un vector folosind Divide et Impera. Optimizarea constă în reducerea numărului de comparații, având complexitate $O(n)$. Dacă toate elementele vectorului sunt egale, algoritmul returnează aceeași valoare pentru minim și maxim.

204. Algoritmul implementează o metodă de tip Divide et Impera pentru a găsi maximul dintr-un vector. Complexitatea este $O(n)$, deoarece fiecare element este evaluat o singură dată. Spațiul utilizat pe stivă este $O(\log n)$ datorită adâncimii maxime a apelurilor recursive. Pentru un vector cu un singur element, funcția returnează acel element.

205. Variantele `A` și`B` nu țin cont de cazul în care nu sunt suficiente elemente negative, deci sunt false.

Varianta `D` nu respectă condiția ca cele k elemente să fie aflate pe poziții distincte din șirul dat, deci este falsă.

206. Singura variantă corectă este B. Pentru simplicitate, putem folosi exemplul $a = [9, 3, 30, 90]$.

Varianta A va compara mereu numere de aceeași dimensiune, dar nu va produce întotdeauna soluția optimă deoarece nu ține cont și de valorile (cifrele) alăturate. Pentru exemplul dat, va returna $[9, 90, 30, 3]$.

Varianta B concatenează numerele și va produce mereu versiunea optimă. Pentru exemplul dat, va returna $[9, 90, 3, 30]$.

Varianta C sortează șirul descrescător. Pentru exemplul dat, va returna $[90, 30, 9, 3]$.

Varianta D sortează doar după ultima cifră. Pentru exemplul dat, va returna $[9, 3, 30, 90]$.

207. Fiecare bețișor trebuie adus la aceeași lungime cu lungimea minimă a unui bețișor până la el. Dacă cumva bețișorul curent are lungimea mai mare decât minimul, atunci vom adăuga la total diferența dintre lungimea acestuia, și minim. Altfel, actualizăm valoarea minimului. Răspunsul final va fi reprezentat de sumă. Varianta corectă este `A`.

208. Costul minim pentru matricea dată este 31, atins prin parcurgerea celulelor astfel: $S \to 7 \to 3 \to 2 \to 4 \to 2 \to 3 \to 5 \to 1 \to 4 \to F$. Acest drum este atins prin alegerea direcției cu suma minimă pană la destinație (deși nu asigura o soluție optimă global, deoarece ar ignora unele drumuri care inițial par mai costisitoare).

209. Strategia D este corectă deoarece sortează artiștii în ordine crescătoare a raportului d_i **DIV** f_i, ceea ce minimizează suma ponderată a timpilor de așteptare.

210. Sortând prezentările după ora de sfârșit și aplicând algoritmul greedy, se selectează intervalele $(1, 4)$, $(5, 7)$ și $(8, 11)$, deci se pot organiza maxim 3 prezentări fără suprapuneri. Deoarece sunt 10 prezentări în total, numărul minim de prezentări care trebuie reprogramate este $10 - 3 = 7$.

211. O permutare de lungime i are cel mult $\frac{i \cdot (i-1)}{2}$ inversiuni, caz în care toate numerele sunt în ordine descrescătoare.

Dacă K este de forma $\frac{M \cdot (M-1)}{2}$, permutarea minimă lexicografic cu K inversiuni va fi:

$$1, 2, 3, \ldots, N-M, N, N-1, N-2, \ldots, N-M+1$$

Cele K inversiuni apar în ultimele M elemente. Dacă în această permutare mutăm un element $N-x$ imediat înaintea lui N, numărul de inversiuni scade cu x. Astfel, dacă $K > \frac{M \cdot (M-1)}{2}$, construim permutarea:

$$1, 2, 3, \ldots, N-M-1, N, N-1, N-2, \ldots, N-M$$

(care are $\frac{(M+1) \cdot M}{2}$ inversiuni) și mutăm elementul $N - \left(\frac{(M+1) \cdot M}{2} - K\right)$ imediat înaintea lui N, astfel reducând numărul de inversiuni la K. Este evident că permutarea astfel construită este minimă lexicografic. Algoritmul descris are complexitate $O(N)$.

212. Se sortează obiectele descrescător după profit – raportul dintre valoare și greutate. Alegem în ordine obiectele cu profitul cel mai mare. Când obiectul curent nu mai încape în rucsac, vom lua doar o parte din acesta. Pentru acest set de date, vom selecta obiectele 2 și 4, iar din obiectul 1 vom lua jumătate, având astfel în total câștigul maxim 220.

213. La fiecare pas, se vor selecta cele mai mici 2 lungimi existente, se va face suma lor, suma se adaugă la total, iar după este adăugată din nou în șir. Procedeul se repetă de 6 ori. În total obținem 173.

214. Afirmația `C` este cheia rezolvării acestei grile. Fiecare înșiruire reprezintă de fapt o permutare a primelor n numere naturale, care poate să fie reprezentată sub forma unui graf. Observăm că, dacă construim graful, putem obține componente conexe, ce conțin cicluri. Pentru permutările în care $p[i] = i$ (denumit ca punct fix al unei permutări), vom avea o componentă conexă cu un singur nod, care nu va trebui mutat niciodată, așadar formula $n + c$ **nu** este validă. Aceasta ar fi adevărată dacă c ar reprezenta numărul de componente conexe cu mai mult de 1 element. Deoarece fiecare configurație poate fi transformată într-un graf, opțiunea `A` nu este corectă. Mutare unui palton deja aflat pe poziția corectă poate fi anulată prin aplicarea mutării anterioare în mod opus (exemplu (1, 5), (5, 1)), deci Victor tot va putea obține configurația finală corectă, dar nu într-un număr minim de pași. Simulând mutările de la `B`, obținem configurația finală corectă.

215. Pentru fiecare dreptunghi selectat, deoarece avem același număr de pătrățele albe și pătrățele negre, vom avea un număr par de pătrățele. Pentru a avea un număr par, vom avea o latură a acestui dreptunghi de două pătrățele. Ca să maximizăm numărul de dreptunghiuri, vom avea suma laturilor a două dreptunghiuri aflate pe același rând egală cu n, adică vom avea dreptunghiuri de dimensiune 1×2, $(n-1) \times 2$, 2×2, $(n-2) \times 2$, 3×2, $(n-3) \times 2$, etc. Pe un singur rând vom avea un singur dreptungi de dimensiune $n \times 2$, deci numărul maxim de dreptunghiuri este $n-1$.

216. Afirmația **A** este falsă deoarece sortarea doar după deadline nu ia în considerare timpul de procesare și profitul, deci nu garantează soluția optimă. **B** este falsă; prin calcularea corectă, profitul maxim pentru setul dat este 21, nu 19. **C** este adevărată; aceasta descrie soluția optimă. **D** este adevărată; cu deadline-uri egale, sortarea după raportul profit/timp maximizează profitul total posibil.

217. - Varianta `B` nu ia în considerare corect intersecțiile intermediare și poate sări peste intersecții valide. - Varianta `D` este greșită deoarece sortarea după punctul de sfârșit descrescător și intersecționarea de la capete spre interior nu garantează găsirea intersecției maxime între toate intervalele.

218. Numărul de permutări în care exact două persoane nu sunt pe locul lor este echivalent cu alegerea a două poziții pe care să le schimbăm și restul elementelor să fie fixate. Numărul de moduri în care putem alege două persoane care să își schimbe locurile este egal cu: $C_m^2 = \frac{m(m-1)}{2}$.

219. Sortarea influențează complexitatea algoritmului, deci varianta C este falsă. Algoritmul nu returnează permutări, ci va returna un număr la final, așadar varianta A este falsă. Legat de opțiunea B, pentru $n = 4, a = [1, 2, 3, 4]$, avem 2 permutări care respectă condiția impusă în enunț, însă algoritmul va returna 4, deci rămâne doar D ca variantă corectă.

220. Soluțiile de numărare a voturilor sunt: AABAABAB, AABAABBA, AABABAAB, AABABABA, AABABBAA, AABBAABA, AABBABAA, ABAABAAB, ABAABABA, ABAABBAA, ABABAABA, ABABABAA $\longrightarrow$ 12 secvențe valide.

221. Pentru $n = 2$, există o singură secvență validă "UD", pentru $n = 4$ există două

secvențe valide "UUDD", "UDUD", pentru $n = 6$ sunt 5 secvențe valide "UDUDUD", "UDUUDD", "UUDDUD", "UUDUDD", "UUUDDD", iar pt $n = 8$ exista 14 secvențe valide. Deși nu este necesar pentru rezolvarea acestui exercițiu, în specialitate, numărul total de secvențe valide de acest tip este dat de către numerele lui Catalan, ce reprezintă un șir de numere determinat de formula $C_n = \frac{1}{n+1}\binom{2n}{n}$, unde C_n reprezintă al n-lea număr Catalan, iar cu $\binom{2n}{n}$ se notează numărul de combinări de $2n$ luate câte n. Totodată, fiecare termen al șirului poate să fie determinat și de relația de recurență $C_0 = 1, C_1 = 1, C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-1-i}, n \geq 2$.

222. Algoritmul explorează recursiv mișcările posibile (dreapta sau jos) din matrice pentru a găsi un traseu valid. După ce găsește primul traseu valid, îl afișează și se oprește.

223. Algoritmul utilizează formula combinatorică recursivă pentru a calcula numărul de submulțimi de dimensiune k: $C(n, k) = C(n-1, k-1) + C(n-1, k)$. Cazurile de bază sunt atunci când $k = 0$ sau $k = n$, rezultând câte o singură submulțime.

224. Algoritmul afișează toate permutările mulțimii $\{1, 2, \ldots n\}$, în ordine invers lexicografică, deoarece parcurgerea în funcția `back` se realizează începând de la n, nu de la 1. Complexitatea este, de asemenea, $O(n! \cdot n)$ - $n!$ pentru numărul de permutări și n pentru verificarea realizată de subalgoritmul `ok`. Deci, varianta corectă este D.

225. Formula de calcul este formula permutărilor fără puncte fixe: $D(n) = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$.

226. Algoritmul `old(a, b)` calculează și returnează aranjamente de a luate câte b.A. Greșit, formula de recurență utilizată este pentru combinări. B. Corect. C. Corect. D. B. Greșit.

227. Algoritmul `old(a, b)` calculează și returnează aranjamente de a luate câte b. A. Adevărat, se calculează aranjamentele de a luate câte b. B. Fals, se returnează 3024. C. Fals, complexitatea de timp este $O(b)$. D. Adevărat.

228. Algoritmul generează toate combinările de k elemente din `arr`, verificând pentru fiecare dacă produsul lor este un pătrat perfect utilizând funcția `G`. Combinările sunt afișate doar dacă această condiție este îndeplinită.

229. Algoritmul generează toate permutările vectorului `arr` și validează fiecare permutare, verificând dacă diferența absolută dintre primul și ultimul element este un număr

prim folosind funcția `g`.

230. Se dă factor comun la fiecare pas: $a_0 + x(a_1 + x(a_2 + \cdots + x(a_{2024})))$. Astfel, sunt necesare 2024 adunări și 2024 înmulțiri, adică 4048 operații în total.

231. Algoritmul `Algo(n, k, idx, s, part)` generează toate partițiile numărului n în k numere naturale, fiecare mai mic decât n, printr-o abordare recursivă. Condițiile impuse în bucla `for` asigură că suma elementelor din partiție este exact n, iar verificarea $k > n$ elimină posibilitatea generării soluțiilor invalide.

232. Algoritmul generează toate permutările posibile ale vectorului `arr`, verificând pentru fiecare dacă suma pozițiilor pare depășește suma pozițiilor impare. Doar permutările care respectă această condiție sunt afișate. Variantele A și D sunt corecte.

233. Algoritmul generează secvențe strict crescătoare cu proprietatea că numerele consecutive sunt prime între ele. Pentru secvența (1,2,5), avem gcd(1,2)=1 și gcd(2,5)=1, deci este o soluție validă. Complexitatea include factorul $\log n$ de la calculul GCD-ului.

234. Algoritmul $f(n, k)$ returnează numărul combinărilor de n luate câte k. $C_n^k = \frac{n!}{k!\cdot(n-k)!} = \frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{k!}$. A. Adevărat, formula clasică iterativă. B. Adevărat, recurența triunghiului lui Pascal. C. Fals, trebuie ca împărțirea (DIV) să fie făcută după înmulțire, dacă nu se va pierde restul împărțirii. D. Adevărat, formula clasică iterativă.

235. Algoritmul `f(n, k)` calculează combinări de n luate câte k, $C_n^k = \frac{n!}{k!(n-k)!}$. A. Fals, $C_{13}^7 = 1716$, nu 1715. B. Adevărat, $C_n^k = C_n^{n-k}$, conform formulei. C. Fals, algoritmul calculează combinări, nu aranjamente. D. Adevărat. Răspunsurile false sunt A și C.

236. Algoritmul trebuie să calculeze toate permutările fără puncte fixe. A. Corect, formula pentru deranjamente este: $D(n) = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$. B. Corect, recurența pentru deranjamente este: $D(n) = (n-1) * (D(n-1) + D(n-2))$. C. Greșit, nu funcționează pentru n impar, semnul este greșit. D. Greșit, suma trebuie să înceapă de la $k = 0$, nu de la 1, pentru a calcula corect deranjamentele.

237. Fie o mulţime M cu n elemente şi un element fix $x \in M$. Fiecare submulţime a lui M fie conţine x, fie nu, iar submulţimile care conţin x corespund exact tuturor submulţimilor lui $M \setminus \{x\}$, care sunt în număr de 2^{n-1}. Singura variantă corectă este C.

238. Fixând cele k elemente ale mulţimii T, acestea trebuie să fie prezente în toate

submulţimile considerate. Rămân astfel $n-k$ elemente din $S \setminus T$, fiecare dintre acestea putând fi inclus sau nu într-o submulţime. Prin urmare, numărul total de submulţimi care conţin toate elementele lui T este 2^{n-k}. Singura variantă corectă este D.

239. Mulţimile A_k şi B_k sunt complementare şi există C_n^k perechi de astfel de mulţimi (A_k, B_k). Se deduce faptul că, pentru un element $x \in M$, numărul de submulţimi A_k în care nu se află acesta este C_{n-1}^k. Aşadar, x apare in C_{n-1}^k în submulţimi de tipul B_k. În concluzie, suma tuturor elementelor din toate submulţimile B_k este $C_{n-1}^k \cdot \sum_{x \in M} x$. Singura variantă corectă este B.

240. Traversarea în inordine parcurge mai întâi subarborele stâng, apoi nodul curent, și la final subarborele drept. Astfel, ordinea corectă este: A, E, B, F, X, T, H, K, M, R.

241. Un arbore binar plin cu înălțimea h conține $2^{h+1} - 1$ noduri.

242. Procesul de inserare a nodurilor într-un arbore binar de căutare trebuie să respecte următoarele proprietăți fundamentale: toate nodurile din subarborele stâng trebuie să aibă valori mai mici decât decât cea a rădăcinii, iar toate nodurile din subarborele drept trebuie să aibă valori mai mari decât cea a rădăcinii. În timpul inserării, valoare elementului, care se dorește a fi adăugat, este comparată cu valoarea nodului curent din arbore pentru a verifica respectarea condițiilor. Dacă elementul este mai mic decât nodul curent, procesul de inserare continuă în subarborele stâng, iar dacă este mai mare, procesul se continuă în subarborele drept.

243. Traversarea în preordine parcurge arborele vizitând mai întâi rădăcina, apoi subarborele stâng, și în final subarborele drept. Ordinea corectă este: 6, 7, 9, 10, 12, 5, 3, 8, 4.

244. Traversarea în postordine parcurge mai întâi subarborele stâng, apoi subarborele drept și la final rădăcina. Ordinea corectă este: B, C, A, F, Y, T, G, R, Z, X.

245. Traversarea inordine a unui arbore binar de căutare vizitează nodurile astfel încât valorile sunt parcurse în ordinea crescătoare, deoarece pentru orice nod, toate valorile din subarborele stâng sunt mai mici, iar cele din subarborele drept sunt mai mari.

246. Un arbore binar complet cu n noduri are înălțimea h dată de formula $h = \lfloor \log_2(n) \rfloor$. Pentru $n = 31$, înălțimea este $\lfloor \log_2(31) \rfloor = 4$.

247. Succesorul în inordine al unui nod dintr-un arbore binar de căutare este cel mai mic

nod din subarborele drept al acestuia. Pentru nodul 20, subarborele drept este format doar din nodul 25. Astfel, succesorul în inordine este 25.

248. Structura arborelui se determină din traversările preordine și inordine: - Preordine indică ordinea nodurilor vizitate, pornind de la rădăcină. - Inordine definește ordinea nodurilor din subarborele stâng, rădăcină, apoi subarborele drept.

249. Numărul maxim de noduri într-un arbore binar complet de înălțime h este dat de formula $2^{h+1} - 1$. Aplicând formula pentru $h = 4$ observam ca numărul maxim de noduri este 31 , iar pentru $h = 5$ observam ca numărul maxim de noduri este 63 . Deci, înălțimea arborelui trebuie sa fie minim 5. Pentru ca numărul de niveluri este $h + 1$, avem răspunsul: numărul minim de niveluri este 6.

250. Structura arborelui reflectă ordinea operațiilor. Rădăcina arborelui este operatorul $+$, care combină rezultatele celor două subarbori principali. Subarborele stâng reprezintă produsul $(a + b + c) * (d * e)$, iar subarborele drept este operandul f. Expresia finală corespunzătoare arborelui este $(a + b + c) * (d * e) + f$.

251. Pentru a evalua expresia, nodurile trebuie procesate în ordinea dată de traversarea în postordine. Parcurgând arborele: 1. Procesăm subarborele stâng $(a*b)$, rezultând $a\,b*$. 2. Procesăm subarborele drept: - Subarborele stâng al acestuia reprezintă $((c + d) * e)$, rezultând $c\,d\,+\,e*$. - Subarborele drept reprezintă $(f * g)$, rezultând $f\,g*$. - Aplicăm operatorul $*$ pe aceste două componente, rezultând $c\,d\,+\,e\,*\,f\,g\,*\,*$. 3. Combinația finală este obținută aplicând $+$: $a\,b\,*\,c\,d\,+\,e\,*\,f\,g\,*\,*+$.

252. Algoritmul calculează înălțimea unui arbore binar în mod recursiv. Pentru fiecare nod, determină înălțimea subarborelui stâng și a celui drept, selectează valoarea maximă dintre ele și adaugă 1 pentru a include nivelul curent. În cazul dat, înălţimea este 4, deci singura variantă corectă este D.

253. Algoritmul dat calculează numărul de descendenţi stângi ai unui arbore, prin explorarea în întregime a acestuia, în mod recursiv. Pentru fiecare descendent stâng pe care îl întâlneşte, incrementează contorul. În cazul dat, numărul de descendenţi stângi este 7 (nodurile b, d, h, l, f, j, n).

254. Algoritmul dat calculează numărul de noduri ai arborelui dat, care au un singur

descendent. Acesta verifică recursiv fiecare nod și, dacă acesta are exact un fiu (stâng sau drept, dar nu ambii), incrementează contorul. În cazul dat, numărul de noduri cu un singur descendent este 4 (nodurile c, h, i, j), deci singura variantă corectă este B.

255. Algoritmul dat verifică dacă arborele dat este arbore binar de căutare. Prin apelul $algoritm(node.left, inf, node.value)$, este impusă condiţia ca descendentul stâng, împreună cu toţi descendenţii lui (dacă aceştia există) să fie mai mici sau egali cu nodul curent şi mai mari sau egali cu cel mai apropiat ascendent al nodului curent, care conţine nodul curent sau ascendenţi ai acestuia în subarborele drept, în cazul în care ascendentul are cel puţin un descendent drept. Prin apelul $algoritm(node.right, node.value,$
$sup)$, este impusă condiţia ca descendentul drept, împreună cu toţi descendenţii lui (dacă aceştia există) să fie mai mari sau egali cu nodul curent şi mai mici sau egali cu cel mai apropiat ascendent al nodului curent, care conţine nodul curent sau ascendenţi ai acestuia în subarborele stâng, în cazul în care ascendentul are cel puţin un descendent stâng. Varianta A este incorectă, deoarece algoritmul verifică relaţia dintre valorile asociate nodurile, în timp ce un arbore binar este complet daca toate nivelurile sunt ocupate în întregime, cu excepţia ultimului, condiţie independentă de valorile nodurilor. Varianta B este incorectă, deoarece condiţia impusă descendenţilor stângi este incorectă. Variantele C şi D sunt corecte.

256. Numărul ciclurilor hamiltoniene distincte într-un graf complet cu n noduri este: $\frac{(n-1)!}{2}$.

257. Algoritmul prezentat este Algoritmul lui Kosaraju, folosit pentru determinarea componentelor tare conexe dintr-un graf orientat. Astfel, conform algoritmului, metoda $d1$ realizează o parcurgere în adâncime asupra grafului inițial, iar $d2$ tot o parcurgere în adâncime, însă asupra grafului transpus. Deci, variantele corecte de răspuns sunt B și C

258. A: Corect. Parcurgerea în preordine vizitează nodul curent, subarborele stâng și apoi subarborele drept. B: Corect. Parcurgerea în inordine vizitează subarborele stâng, nodul curent și apoi subarborele drept. C: Corect. Parcurgerea în postordine vizitează subarborele stâng, subarborele drept și apoi nodul curent. D: Fals. Arborele este complet ultimul nivel k, contine eventual mai putin de 2^k noduri.

259. Dacă arborele are 1023 de noduri, înseamnă că are 10 nivele, numerotate de la 0 la 9 ($2^{10} - 1 = 1023$), deci afirmația A este falsă și B adevărată (pe ultimul nivel vor fi (2^9 noduri). Cum pe ultimul nivel se află nodurile numerotate de la 512 la 1023, 1000 este pe acest nivel și este și fiu stâng al unui alt nod, deoarece este număr par. Verificând strămoșii lui 768, obținem $768 \to 384 \to 192 \to 96 \to 48 \to 24 \to 12 \to 6 \to 3 \to 1$, deci D este, de asemenea, falsă.

260. Algoritmul `Transform(node, k, s, c)` parcurge arborele binar în postordine, calculând suma și numărul de noduri din subarborii stâng și drept, iar pentru fiecare nod verifică dacă suma acestor valori este divizibilă cu k, ajustând valoarea nodului în funcție de această condiție.

261. Algoritmul parcurge recursiv arborele binar de căutare și verifică pentru fiecare nod dacă diferența dintre valorile maxime și minime din subarborii stâng și drept este egală cu valoarea nodului curent și returnează numărul de noduri care îndeplinesc aceste condiții.

262. Nu există nicio componentă tare conexă cu 2 noduri în acest caz, deoarece nu există drum între 2 noduri distincte.

263. Algoritmul parcurge arborele binar în postordine pentru a calcula adâncimile subarborilor stâng și drept ale fiecărui nod, verificând dacă acestea sunt egale. Dacă adâncimile sunt egale și rezultatul adâncimii plus unu este o putere a lui doi, valoarea nodului este dublată. Arborele rezultat trebuie mai apoi traversat în preordine pentru a afla răspunsul corect.

264. Algoritmul traversează arborele binar în postordine, calculând pentru fiecare nod suma și numărul de noduri din subarborii săi. Dacă suma descendenților este un număr prim, valoarea nodului devine produsul numerelor de noduri din subarbori. Altfel, devine suma acestora. După execuție, rădăcina arborelui are valoarea 14, iar toate nodurile interne au fost actualizate condițiilor algoritmului.

265. Algoritmul parcurge arborele binar în postordine, determinând valorile returnate de subarborii stâng și drept pentru fiecare nod intern. Dacă produsul acestor valori % k este egal cu valoarea nodului curent % k, nodul își modifică valoarea la suma acestor valori, iar contorul total este incrementat. Astfel, algoritmul determină câte noduri interne își

schimbă valoarea conform condiției impuse.

266. Algoritmul `Mister` parcurge arborele binar în postordine și calculează suma valorilor nodurilor și numărul de noduri din arbore. Pentru fiecare nod intern, dacă suma valorilor din subarbori este divizibilă cu p, valoarea nodului devine produsul numărului de noduri din subarbori. Dacă valoarea nodului este divizibilă cu p, valoarea acestuia devine suma numărului de noduri din subarbori. Rezultatele finale sunt acumulate în variabilele `sum` și `cnt`.

267. Algoritmul `Algo` parcurge recursiv arborele binar și decide valoarea fiecărui nod intern pe baza valorilor returnate de subarborii săi. Dacă diferența dintre valorile subarborilor este mai mică sau egală cu `dist`, nodul preia valoarea copilului cu adâncimea maximă. Dacă diferența este mai mare, nodul preia valoarea minimă dintre cele două. În urma execuției algoritmului pentru `dist = 5`, valoarea nodului rădăcină devine 6, iar un nod care își modifică valoarea o preia întotdeauna din valorile returnate de subarbori.

268. Arborele este reprezentat printr-un vector de tați `t`, unde fiecare poziție indică părintele nodului corespunzător. Nodul cu valoarea 0 în acest vector este rădăcina arborelui. Analizând vectorul de tați `t = (2, 3, 0, 1, 3, 1, 10, 5, 6, 5)`, observăm că nodul **3** este rădăcina, deoarece este singurul care are valoarea 0.

Pentru afirmația **A**, un nod frunză este un nod care nu are descendenți. Analizând structura arborelui, nodurile **4**, **7**, **8** și **9** nu apar ca părinți pentru alte noduri, ceea ce înseamnă că sunt frunze. Astfel, afirmația **A** este adevărată.

Afirmația **B** este falsă deoarece rădăcina arborelui este nodul **3**, nu nodul **2**.

Afirmația **C** este falsă, deoarece nodul **10** are un descendent, și anume nodul **7**, conform vectorului de tați.

Pentru afirmația **D**, nodurile sunt frați dacă au același părinte. Observăm că nodurile **8** și **10** au ca părinte nodul **5**, ceea ce înseamnă că sunt frați. Prin urmare, afirmația **D** este adevărată.

Astfel, variantele corecte de răspuns sunt **A D**.

269. Traversarea arborelui în inordine presupune parcurgerea subarborelui stâng, apoi procesarea rădăcinii, iar apoi parcurgerea subarborelui drept.

270. Traversarea arborelui în postordine presupune parcurgerea subarborelui stâng, apoi a subarborelui drept, iar în final procesarea rădăcinii.

271. Traversarea arborelui în preordine presupune procesarea rădăcinii, apoi parcurgerea subarborelui stâng, iar apoi a subarborelui drept.

272. Variabila b se inițializează cu True și devine False doar dacă există un element în șir mai mare decât următorul element din șir, adică în momentul în care valorile șirului nu mai sunt în ordine strict crescătoare. Algoritmul returnează True pentru orice șir strict crescător.

273. Algoritmul nu parcurge pur și simplu vectorul de la stânga la dreapta, ci selectează elemente fie din partea stângă, fie din partea dreaptă, în funcție de comparația dintre ele, fiind afișată valoarea mai mică. Afirmația A este greșită deoarece, într-un șir sortat crescător, de la prima comparație se va afișa primul element din șir, cel mai mic, astfel valorile se vor afișa în ordine crescătoare. Pentru un vector sortat descrescător, la prima comparație, se afișează valoarea minimă, iar procesul va continua, alegând întotdeauna cel mai mic dintre $a[i]$ și $a[j]$, iar ultimul element rămas va fi maximul. De asemenea, dacă elementul maxim se află pe prima poziție, în urma comparațiilor, va fi afișat mereu elementul din capătul drept al șirului, rezultând afișarea valorilor în ordine inversă.

274. Scriem numerele în baza 10: $X = 6543_{(8)} = 3427_{(10)}$, $Y = CEF_{(16)} = 3311_{(10)} \rightarrow X \geq Y, X > Y$.

275. Algoritmul nu va executa niciodată instrucțiunile de pe ramura z **MOD** $2 = 0$ pentru că nu x și y nu ajung la aceeași paritate, pentru orice n din intervalul precizat, deci algoritmul va returna mereu aceeași valoare. Dacă am schimba instrucțiunea de pe linia 10 cu $x \leftarrow x-1$, și cea de pe linia 11 cu $y \leftarrow y+1$ algoritmul returnează aceeași valoare ca în varianta originală pentru orice număr natural $1 \leq n \leq 15$, afirmație adevărată datorită faptului că valorile nu pot ajunge la aceeași paritate, singura condiție care ar schimba rezultatul.

276. Algoritmul verifică dacă numărul valorilor cu ultima cifră pară din șir este egal cu numărul valorilor cu ultima cifră impară din șir.

277. Algoritmul returnează lungimea celei mai lungi subsecvențe formate din numere

consecutive crescătoare din vectorul v. Variabila a are valoarea lungimii maxime, fiind actualizată dacă este găsită o lungime mai mare decât maximul curent, iar variabila b reprezintă lungimea secvenței curente parcurse ce respectă proprietatea, valoarea lui b crescând cu o unitate în momentul în care elementul curent respectă proprietatea, sau fiind actaulizată cu valoarea 1, marcând începutul unei noi subsecvențe.

278. Traversarea în postordine a unui arbore binar este o metodă de parcurgere a nodurilor sale în care fiecare nod este procesat după ce au fost procesate toate nodurile din subarborele său stâng și toate nodurile din subarborele său drept, varianta corectă fiind varianta C.

279. Algoritmul afișează un șir de m - 1 valori, variabila $j \in [2, m]$. Deși există și o structură repetitivă bazată pe valoarea variabilei i, algoritmul nu mai afișează nimic după cele m - 1 valori deoarece condiția $j \leq m$ nu va mai fi îndeplinită. De asemenea, dacă m este par, se afișează un șir de valori în care valoarea 0 alternează cu valori care reprezintă pătrate perfecte pare, iar prima și ultima valoare sunt 0, idee ce reiese din $x[i][j] \leftarrow k * k$, pentru k par.

280. Algoritmul încearcă să implementeze operația de comparare element cu element de la sfârșit (os) între doi vectori de biți, returnând 1 când elementele sunt egale și 0 când sunt diferite, păstrând elementele nepereche de la începutul vectorului mai lung, dar implementarea actuală compară elementele de la început în loc de sfârșit.

281. Valoarea variabilei *ok* reprezintă îndeplinirea sau nu a cerinței, devenind False dacă proprietatea de prefix nu este respectată, așa că, varianta D este corectă. De asemenea, dacă se parcurge tot șirul x fără a se schimba starea lui *ok*, variabila i va ajunge la valoarea $m+1$, ceea ce înseamnă că este îndeplinită proprietatea de prefix, varianta C fiind corectă.

282. Varianta A este greșită deoarece se folosește n, numărul de coloane, în loc de m care reprezintă numărul liniilor. Varianta B respectă distincția dintre linii și coloane și calculează corect suma cerută. Varianta C adună numerele de pe linia k, în loc de coloana k, și se produce din nou eroarea în folosirea greșită a variabilei m. Varianta D reinițializează și modifică k, ducând la erori.

283. Deoarece `F(n)` necesită apeluri recursive și reduce progresiv dimensiunea lui n, nu poate avea complexitate constantă, varianta D fiind falsă. Algoritmul reduce dimensiunea problemei la fiecare apel recursiv. Algoritmul este definit prin relația: $F(n) = 1 + F\left(\left\lfloor \frac{n}{\sqrt{n}} \right\rfloor\right)$ cu cazul de bază: $F(1) = 1$. Observăm că noua valoare a lui n este: $n' = \left\lfloor \frac{n}{\sqrt{n}} \right\rfloor = \lfloor \sqrt{n} \rfloor$ Astfel, fiecare apel reduce n aproximativ la rădăcina pătrată a lui n.Fie $T(n)$ numărul de apeluri recursive.

Putem scrie procesul de reducere astfel: $n \to \sqrt{n} \to \sqrt{\sqrt{n}} \to \sqrt{\sqrt{\sqrt{n}}} \to \cdots \to 1$ Pentru a afla câți pași sunt necesari, notăm: $n_{k+1} = \sqrt{n_k}$ unde $n_0 = n$, iar procesul continuă până când $n_k = 1$.

Rezolvând ecuația: $\log n_k = \frac{1}{2} \log n_{k-1}$ Și iterând de mai multe ori, ajungem la: $k \approx \log \log n$. `F(200)`=`F(250)`=4.

284. Algoritmul verifică proprietatea de "vale" a unui șir de elemente, adică există un indice $p \in [2, n-1]$, astfel încât $x[i] < x[i-1], i \in [2, p]$ și $x[j] < x[j+1], i \in [p+1, n-1]$, cu $n \geq 3$, variantele B și C fiind corecte. Dacă vectorul x este ordonat descrescător și are cel puțin 3 elemente, algoritmul returnează `False` deoarece algoritmul `select(n, x)` returnează `n`. Dacă vectorul x este ordonat strict crescător și are cel puțin 3 elemente, algoritmul returnează `True` deoarece algoritmul `select(n, x)` returnează `0` și verificările următoare din algoritmul `check(n, x)` sunt ignorate.

285. Algoritmul determină cele mai mari două valori din șirul x folosind recursivitatea. Varianta A este corectă.

286. Algoritmul returnează `False` pentru un vector cu număr par de elemente. Se parcurge vectorul de la început și se numără elementele mai mici decât e. Când întâlnim primul element mai mare sau egal cu e, numărătoarea se oprește. Se verifică dacă numărul de elemente mai mici decât e este exact egal cu numărul de elemente rămase după poziția lui e (inclusiv el). Aceasta înseamnă că pentru ca algoritmul să returneze `True`, șirul trebuie să îndeplonească următoarele proprietăți: vectorul trebuie să aibă un număr impar de elemente, elementul e trebuie să fie situat în mijlocul vectorului (la poziția $\frac{n}{2} + 1$), toate elementele din stânga lui e trebuie să fie mai mici, toate elementele din dreapta lui e trebuie să fie mai mari.

287. Varianta A este greșită deoarece afișează în ordine inversă cifrele dorite. Varianta B este corectă, afișând restul doar la întoarcerea din recursie. Varianta C afișează cifrele în ordine inversă, iar varianta D nu afișează cifrele individual, ci încearcă să reconstruiască numărul în baza 10.

288. Algoritmul procesează recursiv două numere până când devin egale, modificându-le în funcție de relația dintre ele (divizibilitate și paritatea câtului), și afișează caractere speciale (`*` sau `#`) la ieșirea din recursivitate, plus cuvântul `start:` când numerele devin egale.

289. Algoritmul folosește doi indici, $left$ și $right$ pentru a găsi două elemente din șir cu suma egală cu t. Pe baza actualizării celor doi indici și a comparațiilor, șirul trebuie să fie ordonat crescător pentru a respecta cerința. De asemenea, trebuie să existe în șir și cele două valori cu suma t.

290. Variabila nr reprezintă numărul de factori primi (nu și distincți) comuni ai numerelor x și y. Pentru afișarea valorilor 1 7 11, numerele din pereche trebuie să aibă un singur factor comun, iar, după împărțirea cu acel factor, x devine 7, iar y devine 11. Varianta A este corectă cu factorii comuni următori: $(14, 22) = 2$, $(21, 33) = 3$, $(35, 55) = 5$, $(49, 77) = 7$. Varianta B este greșită deoarece $(7, 11)$ nu au factori comuni, variabila nr rămâne la valoarea 0. Varianta C este greșită din același motiv, $(1, 7)$ nu au factori primi comuni. Varianta D este greșită deoarece, după împărțire, toate numerele din perechi ajung la valoarea 1.

291. Algoritmul găsește cel mai mic număr natural care nu apare în vector folosind o metodă de marcare în care valorile existente sunt identificate prin adăugarea lui n la pozițiile corespunzătoare din vector, iar apoi se returnează primul index unde nu s-a făcut această marcare.

292. Algoritmul verifică dacă un șir poate fi împărțit în două submulțimi cu sume egale, dacă suma tuturor elementelor este număr par. Șirul $[11, 5, 6, 22, 0, 7, 6, 13]$ cu suma 70, poate fi împărțit în $[22, 13, 0]$ și $[11, 5, 6, 7, 6]$, deci returnează `True`. Varianta B este corectă deoarece se returnează `False`, suma elementelor fiind impară. Varianta C este greșită deoarece algoritmul `auxiliar(arr, n, sum)` are condiții de oprire $sum = 0$,

$n = 1$ **AND** $sum \neq 0$, ce opresc recursia. Varianta D este greșită deoarece suma trebuie să fie egală, nu media.

293. Algoritmul returnează cel mai mare divizor comun al șirului. Varianta A este corectă. Suma elementelor înainte de transformarea valorilor în cel mai mare divizor comun nu este mereu strict mai mare deoarece, în cazul unui șir cu valori egale, sumele vor fi egale, deci varianta B este greșită. Varianta C este greșită deoarece nu sunt modificate corespunzător toate valorile. De exemplu, pentru șirul $[30, 12, 18]$, în varianta modificată valoarea primului element nu va fi schimbată niciodată, deci vectorii nu vor avea același conținut. Varianta D este corectă deoarece, pentru orice șir cu valori egale, complexitatea algoritmului `g(a, b)` este $O(1)$, iar cea a algoritmului `f(n, x)` este de $O(n)$.

294. Varianta B este cea corectă deoarece nu putem adăuga în șir mai multe paranteze deschise decât $\frac{n}{2}$, iar nicio paranteză închisă nu poate fi adăugată înaintea unei paranteze deschise corespunzătoare, condiția `inc < desc` asigurând faptul că toate parantezele închise au pereche înainte de adăugare.

295. Varianta A folosește un vector auxiliar pentru a verifica pozițiile curente și face modificările pe șirul inițial până când nu mai sunt mutări de făcut, abordare corectă deoarece evită, prin verificarea valorilor din *aux* schimbarea greșită a pozițiilor. Varianta B efectuează modificările direct pe șirul inițial, dar înaintează cu doi pași dacă a făcut o schimbare pentru a nu modifica invers o pereche nou formată, varianta fiind corectă. Varianta C folosește două variabile pentru a contoriza câți copii trebuie să se întoarcă pe baza întâlnirilor dintre grupurile de 'd' și 's', fără a modifica efectiv șirul, varianta fiind corectă. Varianta D este greșită deoarece actualizează incorect variabilele dr și st, decrementându-le.

296. Algoritmul `ceFace` implementează o variantă de sortare prin selecție, care parcurge vectorul de la stânga la dreapta și selectează cel mai mic element dintre cele rămase, plasându-l pe poziția corectă. Dacă $n = m$, atunci întreaga secvență va fi sortată crescător. Dacă $n < m$, doar primele n elemente vor fi în ordine corectă, iar restul pot rămâne nesortate.

297. Algoritmul `h` parcurge vectorul de la final spre început și modifică elementele sale în

funcție de relația dintre valorile adiacente. În fiecare pas, comparațiile determină suma sau diferența valorilor pentru a obține un nou vector redus. Returnarea valorii 1 se poate întâmpla doar pentru anumite configurații inițiale ale vectorului.

298. Expresia logică dată este evaluată folosind operatori de prioritate precum `OR` și `NOT`. Evaluând pentru valorile $x = 10$ și $y = 41$, expresia devine adevărată deoarece subexpresia (x **MOD** $3 = 0$) este falsă, dar partea dreaptă a expresiei poate deveni adevărată datorită operatorilor de comparare și negare.

299. Algoritmul `prim` verifică primalitatea unui număr natural folosind o abordare optimizată prin eliminarea multiplilor de 2 și testarea divizibilității doar pentru numerele impare până la rădăcina numărului. Acesta funcționează corect și eficient pentru numere impare mai mari de 2, însă returnează fals pentru numere pare mai mari de 2.

300. Algoritmul `f` verifică dacă un vector este strict descrescător, comparând succesiv fiecare element cu următorul. Dacă toate elementele respectă această condiție, returnează adevărat. În caz contrar, returnează fals, detectând prima încălcare a ordinii.

301. Se efectuează conversii din bazele 16, 3 și 4 în baza 10, iar apoi se efectuează operații aritmetice asupra valorilor convertite. Este esențial să se interpreteze corect fiecare bază pentru a obține rezultatul corect.

302. Algoritmul recursiv `f` folosește o abordare unica, reducând valoarea parametrului până când se ajunge la o condiție de oprire. Prin repetarea apelurilor recurente și ajustarea valorilor, se determină punctul în care rezultatul devine negativ.

303. Algoritmul `compute` determină numărul de biți de 1 din reprezentarea binară a unui număr, ceea ce corespunde numărului de biți activați (Hamming weight). Parcurge fiecare bit folosind operații de diviziune și modulo pentru a verifica paritatea.

304. Algoritmul `f` execută un ciclu controlat de condiții logice și modifică valorile variabilelor pe parcurs. În cazul apelului specificat, variabilele sunt alterate succesiv, determinând afișarea unor valori fixe sau intrarea în buclă infinită.

305. Parcurgerea în preordine a unui arbore binar implică vizitarea rădăcinii înaintea subarborilor stâng și drept. Această strategie de explorare produce un șir de noduri în ordinea întâlnirii lor.

306. Algoritmul `mark` utilizează o abordare de propagare a valorilor prin verificări de adiacență într-un graf reprezentat prin vectori. Valorile sunt modificate în funcție de anumite condiții de conectivitate date de funcția auxiliară `tuple`.

307. Algoritmul `matrice` umple o matrice $n \times n$ cu valori care alternează semnul la fiecare poziție. Produsul sau suma elementelor din anumite linii și coloane sunt influențate de acest model de alternare a semnelor.

308. Algoritmul `modifica` rearanjează vectorul astfel încât toate elementele mai mici sau egale decât ultimul element să fie plasate înaintea acestuia, similar unui pas de partitionare din algoritmul quicksort.

309. Algoritmul parcurge vectorul și calculează suma elementelor care sunt multipli de 3, stocând numărul acestora pentru a efectua o împărțire la final. Returnează o valoare bazată pe numărul multiplilor de 3 detectați.

310. Algoritmul de generare a submulțimilor utilizează recursivitatea pentru a explora toate combinațiile posibile ale elementelor setului dat, afișând fiecare subset pe măsură ce este generat.

311. Algoritmul aplică o sumă cumulativă asupra elementelor vectorului și returnează valoarea de la poziția specificată după efectuarea tuturor adunărilor succesive.

312. Algoritmul determină ultimele două cifre ale unui număr și verifică divizibilitatea acestora pentru a decide validitatea condiției impuse.

313. Algoritmul recursiv determină numărul de moduri în care un număr poate fi reprezentat ca sumă de numere consecutive, utilizând apeluri succesive pentru a explora toate posibilitățile.

314. Algoritmul analizează cifrele vectorului de intrare și construiește cel mai mare număr posibil din cifrele care nu apar în vector, sortându-le descrescător.

315. Algoritmul numără insulele de 1 într-o matrice prin parcurgerea recursivă a elementelor vecine pe orizontală și verticală pentru a marca regiunile conectate.

316. Algoritmul verifică dacă un șir de caractere poate fi obținut prin rotirea unui alt șir, folosind concatenarea și compararea secvențelor rezultate.

317. Algoritmul analizează subsecvențe de lungime 3 în vector și identifică pozițiile în

care există mai mult de un număr palindrom, determinând secvența cu cea mai mare frecvență.

318. Algoritmul schimbă valorile în doi vectori conform unor reguli specifice de interschimbare, rezultând modificări bazate pe poziții și relațiile dintre ele.

319. Algoritmul utilizează recursivitate pentru a determina dacă există o submulțime a vectorului care are suma egală cu o valoare specificată, explorând toate combinațiile posibile.

320. Algoritmul va calcula oglinditul lui b, în variabila c. Pentru apelul `ceFace(a, a)`, va calcula oglinditul lui a. În caz ca numerele nu sunt egale, acesta se auto-apelează, decrementând valoarea inițială, până când se întâlnește o valoare pentru care oglinditul său este egal cu numărul inițial (palindrom).

321. Algoritmul va afișa o valoare pentru fiecare pereche $n * m$. În cazul în care k este par, valoarea afișată va fi 0 (valoarea inițială), altfel se va afișa k^2. Astfel, șirul afișat va alterna între valori de 0 și $(2k+1)^2, \forall k \in \mathrm{N} : [0, 1, 0, 9, \ldots]$.

322. Algoritmul calculează numărul de cifre pentru fiecare număr al șirului.

323. Condiția $((d \textbf{ DIV } a[i]) * a[i] = d)$ **AND** $((d \textbf{ DIV } v) * v = d)$ poate fi tradusă ca și "cel mai mic număr (mai mare sau egal decât 3) care e divizibil cu $[i]$ și v concomitent". Astfel, algoritmul găsește, pe rând $v = 15$, pentru $a[1]$, $v = 60$, pentru $a[2]$.

324. Algoritmul compară numărul de cifre pentru perechile de numere egal depărtate de mijlocul șirului (nr. cifre din primul număr - nr. cifre din ultimul număr șamd.) Observăm ca vectorii de la $\boxed{\text{B}}$ și $\boxed{\text{D}}$ îndeplinesc această condiție, iar dacă elementele vectorului au același număr de cifre ($\boxed{\text{C}}$), condiția este îndeplinită de asemenea.

325. Algoritmul se auto-apelează până la cazul de bază, adăugând cifra curentă la rezultat, doar dacă aceasta este pară.

326. Urmărind algoritmul, $f(10)$ va afișa $f(5)$ 5 $f(2)$. Unde $f(5)$ afișează $f(2)2f(1)$, iar $f(2)$ afișează $f(1)1$. Înlocuind fiecare apel cu rezultatul returnat, obținem 0120501, pentru $f(10)$.

327. Algoritmul parcurge matricea M sub forma unei spirale. Fiecare for din interiorul while-ului parcurge elementele pe cele patru direcții (dreapta, jos, stânga, sus), în această

ordine. Astfel, se va afișa 123456789.

328. Algoritmul verifică dacă un număr 'a' poate fi redus la 1 prin împărțiri repetate cu 'b'. Este similar cu verificarea dacă un număr poate fi exprimat ca o putere a lui b, dar în sens invers - se împarte în jos în loc să se înmulțească în sus.

329. Algoritmul este format din două părți: `decide(n)` mai întâi inversează un număr (ex. 123 devine 321) și verifică dacă acest număr inversat este divizibil cu 3, returnând 1 dacă este divizibil și -1 dacă nu este, în timp ce `compute(m)` folosește această funcție pentru a procesa fiecare număr de la 0 până la $m-1$, adunând toți acești 1 și -1 pentru a produce o sumă finală. Doar 99 și 101 obțin valoarea finală -33.

330. Algoritmul convertește și afișează recursiv reprezentarea numărului n în baza x, începând cu cea mai semnificativă cifră.

331. Algoritmul parcurge recursiv cifrele numărului și returnează cea mai mare cifră pară găsită, sau -1 dacă nu există nicio cifră pară.

332. Algoritmul verifică dacă elementele vectorului sunt în ordine strict crescătoare, parcurgând perechile consecutive de elemente și returnând True doar dacă fiecare element este mai mic decât următorul.

333. Expresia evaluează două condiții: fie împărțirea la 6 a produsului numerelor plus 3 dă 10, fie produsul este divizibil cu 6 și suma lor este divizibilă cu 4. Folosind perechile (2,57) și (4,30), unde primele numere sunt puteri ale lui 2 iar al doilea sunt multipli de 3, găsim că expresia poate fi adevărată pentru perechi diferite, prima satisfăcând prima condiție iar a doua satisfăcând a doua condiție. Cu perechea (4,6) demonstrăm că expresia poate fi și falsă, deci afirmațiile A, B și C sunt adevărate, iar D este falsă.

334. Algoritmul verifică dacă un șir poate fi obținut ca o subsecvență a altui șir, și acest lucru este implementat corect în trei moduri diferite: varianta A face verificarea recursiv pornind de la sfârșit și când găsește caractere identice le elimină din ambele șiruri altfel doar elimină din șirul sursă, varianta B parcurge iterativ de la stânga și avansează în primul șir doar când găsește potriviri iar în al doilea mereu, iar varianta C face același lucru dar pornind de la dreapta la stânga și decrementând indecșii, toate trei menținând astfel ordinea relativă a caracterelor.

335. Varianta C este corectă deoarece atunci când găsește elementul căutat, folosește un al doilea while pentru a muta fiecare element din față cu o poziție la dreapta ($x[index2] = x[index2 - 1]$), creând astfel spațiu la începutul vectorului pentru elementul găsit, și păstrând ordinea celorlalte elemente, totul în complexitate O(n) pentru că deplasarea se face o singură dată.

336. Algoritmul calculează suma dintre pătratul fiecărui număr de la 1 la x, plus produsul dintre x și pătratul lui y, plus z, folosind recursivitate pentru a acumula aceste valori prin adăugarea la fiecare pas a expresiei $x^2 + y^2 + z$.

Expresiile A și C sunt corecte deoarece algoritmul produce același rezultat în două moduri diferite de interpretare a recursivității: în varianta A, când apelăm `expresie(x,y,z)`, acumulăm $x^2 + y^2 + z$ la fiecare pas până ajungem la $x = 0$, ceea ce dă suma pătratelor până la x plus suma de xy repetată de y ori plus z, iar în varianta C obținem același rezultat pentru că y^2 adăugat de x ori este echivalent cu xy^2, deci ambele formule sunt reprezentări matematice diferite ale aceluiași calcul recursiv.

337. Algoritmul folosește căutarea binară pentru a găsi poziția primului 1 într-un vector sortat de 0 și 1, iar apoi returnează numărul total de elemente de 1 din vector (care, datorită sortării, sunt toate consecutive la final), făcând acest lucru prin împărțirea recursivă a intervalului în două jumătăți și sumând rezultatele parțiale.

338. Pentru a găsi un șir minim care conține toate secvențele binare posibile de lungime 4, trebuie să: numărăm câte secvențe diferite avem ($2^4 = 16$ secvențe, de exemplu 0000, 0001, 0010, ..., 1111), apoi construim un șir în care fiecare grup de 4 cifre consecutive să fie una din aceste secvențe diferite, iar pentru a putea citi ultima secvență completă avem nevoie de 19 cifre deoarece dacă am avea 18, ultimele 4 cifre ar forma doar o secvență, dar ne-ar lipsi altele.

339. Algoritmul afișează caractere 'c' în funcție de raportul dintre x și y, folosind recursivitate cu trei cazuri diferite: când x se divide cu y crește x și scade y, când câtul împărțirii lui x la y este impar scade x și crește y afișând un 'c', iar când câtul este par scade ambele numere și afișează 'cc', terminându-se când x devine mai mic sau egal cu y și afișează caracterul q.

340. Algoritmul calculează h-index-ul unui vector prin parcurgerea acestuia și menținerea primelor h elemente sortate descrescător, verificând la fiecare pas dacă există cel puțin h elemente mai mari sau egale cu h.

341. Algoritmul generează toate combinațiile de p numere din primele n numere naturale, unde fiecare număr este mai mare decât toate numerele precedente din combinație.

342. Algoritmul generează recursiv toate secvențele posibile de numere -1 și 1 care au lungimea $2p$, având p numere de -1 și p numere de 1, prin adăugarea alternativă a numerelor -1 (când $s < p$) și 1 (când $s > d$), până când ambii contori s și d ajung la p.

343. Algoritmul găsește suma maximă a unei subsecvențe continue din vector folosind o abordare divide-et-impera, unde pentru fiecare poziție de mijloc calculează trei posibilități: suma maximă din stânga, suma maximă din dreapta, și suma maximă care traversează poziția de mijloc.

344. Algoritmul numără de câte ori se poate împărți numărul la 10 până ajunge la 0, adăugând +1 de fiecare dată când numărul este divizibil cu 3.

345. Algoritmul calculează oglinditul numărului a, în variabila p, pe care o compară cu b. Astfel, dacă a este oglinditul lui b se returnează **True**.

346. Urmărind instrucțiunile din interiorul for-ului, observăm ca primul număr adăugat la suma este $frac12 \cdot 3$. Cum i merge până la n, valoarea finală adaugată la suma este $\frac{n}{(n+1)(n+2)}$.

347. Algoritmul verifică dacă vectorul dat are un aspect de "vale", adică numerele scad până la un anumit punct, iar apoi cresc. Doar vectorii de la $\boxed{\text{A}}$ și $\boxed{\text{C}}$ respectă această condiție.

348. Algoritmul implementează metoda lui Euclid prin împărțiri succesive, între două numere (ab) și (cd) pentru a găsi CMMDC-ul lor.

349. Algoritmul interclasează elementele din vectorii a și b (iar dacă un vector devine consumat, restul de elemente din celălalt șir nu vor fi adăugate la șirul final). De aceea valoarea returnată de $nc = na + nb$, dacă și numai dacă șirurile au lungime egală.

350. Algoritmul calculează suma primelor $a[i]$ elemente, din șirul b. În cazul nostru, $a[i]$ va lua valorile: 1, pentru care $s = 2$, și 4, pentru care $s = 2 + 4 + 6 + 8 = 20$, deci suma

finală este 22.

351. Algoritmul implementează căutarea binară pentru a verifica dacă numărul n este pătrat perfect, căutând un număr între $p1$ și $p2$ al cărui pătrat este egal cu n.

352. Algoritmul verifică dacă un număr este pătrat perfect folosind proprietatea matematică conform căreia suma primelor n numere impare este întotdeauna egală cu n^2, deoarece adună numere impare consecutive începând cu 1 $(1, 3, 5, 7, ...)$ până când suma fie devine egală cu numărul dat (caz în care numărul este un pătrat perfect), fie îl depășește (caz în care nu este pătrat perfect).

353. Algoritmul găsește cel mai mic număr palindrom care este mai mare sau egal cu numărul dat a, prin verificarea succesivă a numerelor începând cu a până găsește primul palindrom.

354. Algoritmul verifică dacă primul element este par și apoi dacă restul elementelor alternează între impar și par în această ordine, returnând True doar dacă această alternanță este respectată.

355. Algoritmul implementează metoda Kadane pentru a găsi suma maximă a unei subsecvențe continue din vector, resetând suma curentă la zero când aceasta devine negativă și păstrând suma maximă găsită până în acel moment.

356. Doar algoritmul C transformă poziția unui element din matricea originală în poziția sa corespunzătoare din matricea redimensionată, prin convertirea indicilor (i,j) într-un index liniar după scăderea lui 1 din indici, apoi împărțind acest index la numărul de coloane al matricei noi pentru a obține linia și folosind restul împărțirii pentru coloană, adăugând 1 la final pentru a reveni la numerotarea de la 1.

357. Algoritmul completează elementele matricei aflate pe poziții cu indici impari și a căror sumă este mai mică decât n cu numerele din șirul lui Fibonacci, parcurgând matricea pe coloane.

358. Algoritmii implementează căutarea unui element într-un vector sortat, unde variantele A și B folosesc căutare binară (complexitate logaritmică) în mod recursiv și iterativ, în timp ce variantele C și D folosesc căutare liniară.

359. Algoritmul compară valoarea absolută a diferenței dintre n și m cu diferența directă

dintre n și m, și în funcție de această comparație calculează fie restul împărțirii lui n la m, fie restul împărțirii lui $(m+2)$ la n. Algoritmul returnează 0 doar pentru $m = 1$ și $m = n$ deoarece doar în aceste cazuri diferența dintre $abs(m-n)$ și $(n-m)$ este 0, iar operația $(nMODm)$ are întotdeauna rezultatul 0, indiferent de valoarea lui n.

360. După ce epuizează toate combinațiile cu 4 la sute și 4,3,8 la zeci (cu cifre impare 3,5,7 la unități), algoritmul ajunge la numărul 453 deoarece 5 este următoarea cifră disponibilă pentru poziția zecilor conform șirului dat [4,3,8,5,7,6].

361. Algoritmul numără recursiv câte numere din vector au exact k divizori proprii (excluzând 1 și numărul însuși).

362. Algoritmul verifică dacă un număr poate fi reprezentat în baza 3 folosind doar cifrele 0 și 1 (condiția `n MOD 3` $\geq$ `1`, ceea ce este echivalent cu verificarea dacă numărul poate fi scris ca sumă de puteri distincte ale lui 3.

363. A este corect deoarece transformă corect numerele din sala I în coordonate pentru sala II prin: ajustarea numerelor impare (adăugând 1) înainte de calcule pentru a păstra consecvența, calcularea corectă a rândului folosind împărțirea la $2K$, și determinarea precisă a poziției în rând prin raportare la începutul rândului curent.

B este greșit deoarece nu ajustează numerele impare înainte de calcule (nu adaugă 1), ceea ce duce la calcularea incorectă a poziției în rând pentru locurile din partea dreaptă.

C este greșit deoarece deși ajustează numerele impare, nu verifică cazul special când numărul locului este multiplu de $2K$, ducând la calcularea incorectă a rândului în aceste situații.

D este greșit deoarece adaugă 1 în plus la calculul poziției în rând ("+1" în ultima linie), ceea ce face ca toate locurile să fie deplasate cu o poziție față de culoar, rezultând în coordonate invalide.

364. Varianta A este corectă deoarece implementează corect backtracking-ul prin decrementarea lui k când se ajunge la finalul unei încercări ($k > 1$) și setează final pe True doar când k ajunge la 1, permițând astfel generarea completă și unică a tuturor permutărilor posibile prin revenirea sistematică la pozițiile anterioare pentru a încerca toate combinațiile posibile; celelalte variante eșuează: B verifică $k > 0$ în loc de $k > 1$ ceea ce

ar opri algoritmul prea devreme, C setează final = True imediat și ar genera doar prima permutare, iar D setează final = True simultan cu decrementarea lui k, împiedicând generarea tuturor permutărilor

365. Algoritmul `problema` conține trei for-uri imbricate, fiecare de la 0 la n, de unde rezultă ca vom avea $(n+1)^3$ pași. Însă, rezultatul crește doar dacă p este par (jumătate din cazuri), deci rezultatul final va fi $(n+1)^2 \times \lceil \frac{n+1}{2} \rceil$, unde $\lceil \frac{n+1}{2} \rceil$ reprezintă partea întreagă superioara (adică numărul de numere pare). Algoritmul `calcul` calculează suma pentru fiecare astfel de cifră din intervalul $[a, b]$. Calculul final este $\sum_{k=a}^{b} (k+1)^2 \times \lceil \frac{k+1}{2} \rceil$.

Observație, pentru $\lceil \frac{n}{2} \rceil = \frac{n}{2} \Rightarrow n^2 \lceil \frac{n}{2} \rceil = \frac{n^3}{2}$, când n este par, și $\lceil \frac{n}{2} \rceil = \frac{n+1}{2} \Rightarrow n^2 \lceil \frac{n}{2} \rceil = \frac{n^3+n^2}{2}$.

Astfel, suma finală poate fi scrisă ca $S = \frac{1}{2}\Big[\sum_{n=M}^{N} n^3 + \sum_{n=M,\, n \text{ impar}}^{N} n^2 \Big]$, unde $M = a+1$, $N = b+1$.

366. Algoritmul implementĕază un automat finit cu două stări (1 şi 2) care verifică dacă o secvenţă de tranziţii este validă conform tabelului t(stare_curentă, input, stare_următoare) – unde din starea 1 putem rămâne în 1 cu input 0, merge în 2 cu input 1, iar din starea 2 putem doar rămâne în 2 cu input 1 – astfel că $[0, 0, 1, 1]$ cu $f = 2$ şi $[0, 0, 0, 0]$ cu $f = 1$ sunt singurele cazuri valide deoarece în primul caz ajungem corect în starea finală 2 prin tranziţii 1→1→1→2→2, iar în al doilea caz rămânem constant în starea 1 care este şi starea finală dorită.

367. Algoritmul folosește un sistem de tip stivă prin vectorul b, unde putem doar adăuga elementele din a în ordinea lor inițială (0,1,2,3...) și le putem afișa doar în ordine inversă față de cum au fost adăugate, făcând imposibilă secvența '2 4 6 5 3 7 0 1 9 8' deoarece pentru a afișa 6 înaintea lui 5 și 3, ar trebui să avem 6 deasupra acestora în stivă, dar cifrele pot fi adăugate doar în ordinea lor inițială.

368. Algoritmul `decide(n, x)` verifică dacă vectorul x este strict crescător. Inițial, variabilei b îi este atribuită valoarea *True*, iar vectorul este parcurs element cu element. Dacă la un moment dat un element $x[i]$ este mai mare sau egal cu $x[i+1]$, atunci b devine *False* și algoritmul se oprește, returnând această valoare.

Dacă toate elementele sunt în ordine strict crescătoare, algoritmul parcurge întregul vector și returnează *True*. Astfel, afirmațiile **A** și **B** sunt corecte, deoarece un vector de forma $1, 2, 3, \ldots, 10$ este strict crescător, iar orice vector strict crescător va face ca algoritmul să returneze întotdeauna valoarea *True*.

369. Un număr este palindrom dacă citit de la stânga la dreapta are aceeași valoare ca atunci când este citit de la dreapta la stânga. Algoritmii care verifică această proprietate trebuie să compare elementele corespunzătoare din șirul a, de la început și sfârșit, avansând spre centru.

Algoritmul `palindrom1` compară succesiv prima și ultima cifră, apoi avansează spre interior, verificând egalitatea până când toate perechile sunt validate sau se găsește o diferență. Dacă toate verificările sunt valide, atunci acesta returnează *True*, confirmând astfel că funcția este corectă.

Algoritmul `palindrom2` utilizează o abordare recursivă. Compară prima și ultima cifră și, dacă sunt egale, elimină aceste două cifre prin translatare și apelează funcția pe secțiunea rămasă. Când ajunge la un șir de lungime 0 sau 1, returnează *True*, ceea ce confirmă că funcționează corect.

Algoritmul `palindrom3` este incorect, deoarece compară sumele cifrelor din prima și a doua jumătate, ceea ce nu este suficient pentru a verifica dacă numărul este palindrom. De exemplu, pentru numărul 12321, sumele pot fi egale fără ca pozițiile cifrelor să fie corecte.

Algoritmul `palindrom4` introduce o condiție suplimentară nejustificată, verificând doar pozițiile pare din șir, ceea ce poate duce la rezultate eronate în multe cazuri.

370. Algoritmul `F(n)` parcurge recursiv cifrele numărului n de la stânga la dreapta și returnează ultima cifră care respectă o anumită condiție legată de resturile împărțirii la 5.

Dacă n are o singură cifră, algoritmul returnează acea cifră. În caz contrar, separă ultima cifră u și apelează recursiv funcția pentru restul numărului p. După revenirea din apelurile recursive, algoritmul compară u **MOD** 5 cu p **MOD** 5. Dacă restul lui u la împărțirea la 5 este mai mic sau egal decât cel al lui p, returnează u, altfel returnează p.

Pentru $n = 812376$, cifrele sunt analizate în ordine: 8, 1, 2, 3, 7, 6. Comparând resturile împărțirii la 5, algoritmul ajunge la cifra 6, ceea ce confirmă afirmația **A**.

Pentru $n = 8237631$, analiza cifrelor duce la rezultatul final 1, confirmând afirmația **B**.

371. Algoritmul `f(n)` determină cifra cu cel mai mare număr de apariții în numărul n. Pentru fiecare cifră c de la 0 la 9, algoritmul parcurge numărul n și numără de câte ori apare această cifră. Dacă o anumită cifră apare mai frecvent decât cele anterioare, aceasta este salvată în variabila z, iar frecvența acesteia este memorată în variabila v.

Afirmația **C** este corectă, deoarece algoritmul returnează una dintre cifrele cu cel mai mare număr de apariții în n.

Afirmația **A** este falsă, deoarece algoritmul nu returnează numărul total de cifre ale lui n, ci doar cea mai frecventă cifră.

Afirmația **B** este falsă, deoarece algoritmul nu returnează numărul de apariții al celei mai frecvente cifre, ci cifra însăși.

Afirmația **D** este falsă, deoarece algoritmul nu returnează numărul cifrelor care au cea mai mare frecvență.

Prin urmare, singurul răspuns corect este **C**.

372. Reprezentarea binară a unui număr se poate obține prin utilizarea împărțirilor succesive la 2 și memorarea resturilor în ordine inversă. Algoritmul corect trebuie să apeleze recursiv funcția utilizând partea întreagă rezultată în urma împărțirii x **DIV** 2 și apoi să afișeze restul x **MOD** 2. Aceasta asigură că bitul cel mai semnificativ este afișat primul.

Varianta **B** este corectă, deoarece verifică dacă $x \neq 0$ și apelează recursiv funcția pentru x **DIV** 2, iar apoi afișează restul, obținând astfel reprezentarea binară corectă.

Varianta **A** este incorectă deoarece condiția $x = 0$ nu permite apelul recursiv corect.

Varianta **C** utilizează **DIV** în loc de **MOD** pentru reprezentarea restului, ceea ce duce la afișarea incorectă a bitului.

Varianta **D** nu modifică valoarea x la apelul recursiv, având ca rezultat apeluri recursive infinite.

373. Afirmația **A** este corectă, deoarece algoritmul din varianta A are ca și condiție

$x = 0$, ceea ce înseamnă că funcția nu se va apela recursiv pentru valori valide ale variabliei x, iar astfel nu se va afișa nimic.

Afirmația **B** este falsă, deoarece algoritmul din varianta B se va apela recursiv pentru orice valoare validă a lui x, împărțindu-l succesiv la 2 și afișând corect resturile.

Afirmația **C** este falsă, deoarece schimbarea condiției de la $x = 0$ la $x \neq 0$ în varianta C nu ar modifica cu nimic corectitudinea algoritmului, întrucât algoritmul utilizează **DIV** în loc de **MOD** pentru calculul restului, ceea ce generează o reprezentare binară incorectă.

Afirmația **D** este corectă, deoarece dacă în varianta D se înlocuiește apelul imp(x) cu imp(x **DIV** 2), atunci algoritmul va funcționa corect, aplicând împărțirea succesivă la 2 și afișând bitii în ordinea corectă.

Prin urmare, afirmațiile corecte sunt **A** și **D**.

374. Expresia **NOT** $((a > 0)$ **AND** $(b > 0))$ este adevărată atunci când cel puțin unul dintre numerele a sau b nu este strict pozitiv. Aceasta înseamnă că expresia este echivalentă cu cazul în care cel puțin unul dintre a sau b nu îndeplinește condiția > 0, adică $a \leq 0$ sau $b \leq 0$.

Varianta **C** este corectă, deoarece afirmă că cel puțin unul dintre a sau b nu este strict pozitiv, iar acest lucru determina echivalența celor două expresii. Varianta **A** este incorectă deoarece **NOT** $(a < 0)$ este echivalent cu $a \geq 0$, iar expresia rezultată verifică dacă ambele numere sunt pozitive, lucru care nu este echivalent cu expresia inițială. Varianta **B** este incorectă deoarece verifică dacă ambele numere sunt negative sau zero, ceea ce nu respectă condiția expresiei inițiale. Varianta **D** este incorectă deoarece schimbă structura logică și nu reprezintă echivalentul expresiei inițiale.

Așadar, varianta corectă de răspuns este varianta **C**.

375. Algoritmul `s(n)` calculează o sumă de termeni fracționari în care fiecare termen este de forma $\frac{1}{k!}$, unde $k!$ reprezintă factorialul lui k. Factorialul este calculat iterativ prin acumulare valorilor în variabila p, iar suma este actualizată la fiecare pas.

Baza de numerotare a buclei începe de la $k = 0$ și merge până la $k = n - 1$, ceea ce înseamnă că suma finală este:$\sum_{k=0}^{n-1} \frac{1}{k!}$

Varianta **C** este corectă, deoarece formula sumei reflectă ceea ce calculează algoritmul.

Varianta **A** este incorectă, deoarece suma calculată de algoritm nu include termenul $\frac{1}{n!}$. Varianta **B** este incorectă, deoarece algoritmul nu calculează suma inverselor numerelor naturale, ci ale factorialelor acestora. Varianta **D** este incorectă, deoarece suma începe de la $k = 0$ și nu de la $k = 1$.

În concluzie, răspunsul corect este varianta **C**.

376. Algoritmul `ceFace(n)` calculează o sumă de resturi succesive ale împărțirii numărului n la puteri crescătoare ale lui 10. Inițial, variabilei p îi este atribuită valoarea 10 și aceasta se mărește exponențial cât timp rămâne mai mică decât n. La fiecare iterație, algoritmul determină restul împărțirii lui n la p și adaugă această valoare la m. Singura variantă corectă este varianta **D**, deoarece pentru $n = 340$, algoritmul returnează valoarea 40.

377. Algoritmul `f(v, n)` determină numărul divizorilor primi distincți ai tuturor numerelor din vectorul v.

În prima parte al acestuia, algoritmul parcurge fiecare element $v[i]$ al vectorului și determină toți divizorii săi primi, înmulțindu-i într-o variabilă x. Așadar, x conține produsul tuturor divizorilor primi ai elementelor din v.

În a doua parte, algoritmul numără divizorii primi distincți ai lui x, ceea ce echivalează cu numărarea divizorilor primi unici ai tuturor numerelor din v. Aceasta confirmă că afirmația **A** este corectă. Varianta **B** este incorectă, deoarece algoritmul returnează numărul divizorilor primi distincți și nu produsul lor. Varianta **C** este incorectă, deoarece algoritmul nu verifică dacă numerele din vector sunt prime. Varianta **D** este incorectă, deoarece algoritmul nu numără toți divizorii fiecărui număr, ci doar divizorii primi unici. Așadar, singura variantă corectă de răspuns este varianta **A**.

378. Algoritmul `f(n)` construiește cel mai mare număr care poate fi obținut folosind cifrele lui n, ordonându-le descrescător.

Prima parte a algoritmului extrage cifrele lui n și le salvează într-un vector v. A doua parte parcurge vectorul și selectează iterativ cea mai mare cifră rămasă și construiește un nou număr x prin concatenarea acestora în ordine descrescătoare.

Varianta **A** este corectă, deoarece algoritmul sortează cifrele lui n în ordine descrescătoare

și le combină într-un nou număr și astfel rezultă cel mai mare număr posibil format din cifrele lui n.

379. Algoritmul `f(n)` parcurge cifrele numărului n și înlocuiește fiecare cifră care este divizibilă cu 3 cu $9 - c$, construind astfel un nou număr. Acest proces este realizat prin utilizarea unei variabile z, care inițial este 0, și a unei variabile p, folosită pentru poziționarea corectă a cifrelor în noul număr.

Pentru $n = 103456$, descompunem cifrele și verificăm divizibilitatea cu 3:

- $1 \rightarrow$ nu este divizibil cu 3 și nu se modifică
- $0 \rightarrow$ este divizibil cu 3 și este înlocuit cu $9 - 0 = 9$
- $3 \rightarrow$ este divizibil cu 3 și înlocuit cu $9 - 3 = 6$
- $4 \rightarrow$ nu este divizibil cu 3 și nu se modifică
- $5 \rightarrow$ nu este divizibil cu 3 și nu se modifică
- $6 \rightarrow$ este divizibil cu 3 și este înlocuit cu $9 - 6 = 3$

Construind numărul doar cu cifrele înlocuite, obținem 963, ceea ce corespunde variantei de răspuns **B**.

380. Algoritmul `f(n)` determină cifrele lui n care sunt divizibile cu 3 și le înlocuiește cu $9 - c$, apoi construiește un nou număr din aceste valori. Pentru a verifica care dintre variante returnează exact valoarea 3, este necesar să analizăm cifrele fiecărui număr și să vedem dacă înlocuirea lor generează rezultatul corect.

În varianta **A**, numerele sunt 61, 65 și 67. Dintre cifrele numerelor, doar cifra 6 este divizibilă cu 3 și se transformă în $9 - 6 = 3$, iar celelalte cifre nu contribuie la rezultat. Astfel, pentru toate numerele din această variantă, algoritmul returnează 3.

În varianta **B**, numerele sunt 62, 66 și 68. Pentru numărul 66, ambele cifre sunt divizibile cu 3, ceea ce duce la două transformări și generează două cifre de 3.

În varianta **C**, numerele sunt 16, 56 și 76. Cifra 6 este divizibilă cu 3 și devine 3, iar celelalte cifre nu contribuie la rezultat. Astfel, algoritmul returnează 3 pentru toate numerele din acest șir

În varianta **D**, numerele sunt 26, 66 și 86. Similar variantei B, numărul 66 conține două cifre divizibile cu 3, ceea ce duce la două transformări și generează două cifre de 3.

Așadar, răspunsurile corecte sunt variantele **A** și **C**.

381. Algoritmul `ceFace(a, b)` identifică toți divizorii pari ai numărului a și verifică care dintre aceștia sunt și divizori ai numărului b. Dacă un astfel de divizor comun este găsit, acesta este afișat.

Pentru $a = 600$, trebuie mai întâi să determinăm toți divizorii săi pari. Numărul 600 are următorii divizori pari: $2, 4, 6, 10, 12, 20, 30, 40, 50, 60, 100, 150, 200, 300, 600$. Algoritmul îi afișează doar dacă sunt și divizori ai lui b. Pentru a se afișa exact patru numere, trebuie să identificăm acele valori ale lui b care au exact patru divizori comuni cu a.

Pentru $b = 20$, divizorii comuni sunt $2, 4, 10, 20$, ceea ce înseamnă că vor fi afișate patru numere. Pentru $b = 50$, divizorii comuni sunt $2, 10, 50$, ceea ce înseamnă că vor fi afișate doar trei numere, astfel această variantă este incorectă. Pentru $b = 12$, divizorii comuni sunt $2, 4, 6, 12$, ceea ce înseamnă că vor fi afișate patru numere. Pentru $b = 90$, divizorii comuni sunt $2, 6, 10, 30$, ceea ce înseamnă că vor fi afișate patru numere.

Așadar, răspunsurile corecte sunt variantele **A**, **C** și **D**.

382. Algoritmul `ceFace(a, b)` parcurge numerele pare de la 2 până la a, verificând care dintre acestea sunt divizori ai lui a. Pentru fiecare astfel de divizor, verifică dacă este și divizor al lui b și, în caz afirmativ, îl afișează. Acest proces garantează că sunt afișați doar divizorii comuni ai numerelor a și b care sunt pari.

Afirmația **D** este corectă, deoarece algoritmul selectează și afișează doar divizorii pari comuni ai numerelor a și b.

383. Pentru a determina numărul generat imediat înainte și cel generat imediat după această secvență, trebuie să analizăm permutările în ordine lexicografică.

Numărul generat imediat înainte de 34256 este ultima permutare lexicografic mai mică, ceea ce înseamnă că trebuie să găsim cea mai mare permutare posibilă care precedă această secvență. Aceasta este 32654.

Numărul generat imediat după 34562 este prima permutare lexicografic mai mare, ceea ce înseamnă că trebuie să găsim cea mai mică permutare posibilă care urmează acestei

secvențe. Aceasta este 34625.

Prin urmare, răspunsul corect este varianta **C**.

384. În șirul observat se poate determina un model după care apar numerele și numărul lor de apariții. Astfel, numărul 1 apare de două ori, numărul 2 apare de patru ori, numărul 3 apare de șase ori, numărul 4 apare de opt ori și așa mai departe. Pentru a determina pozițiile în care apare doar valoarea 11, trebuie să identificăm unde începe și unde se termină secvența de 11 apariții consecutive.

Analizând fiecare variantă, subsecvențele $x[113], \ldots, x[120]$ și $x[123], \ldots, x[132]$ sunt complet cuprinse în domeniul de apariție al valorii 11, ceea ce le face corecte.

Prin urmare, răspunsul corect este varianta **B D**.

385. Pentru a determina câte dintre primele 100 de elemente ale șirului x sunt numere prime, trebuie mai întâi să identificăm structura șirului și să număram câte dintre elementele sale sunt numere prime.

Șirul x este definit astfel încât fiecare număr k apare de $2k$ ori. Distribuția valorilor în primele 100 de poziții este:

- 1 apare de 2 ori (pozițiile 1–2)
- 2 apare de 4 ori (pozițiile 3–6)
- 3 apare de 6 ori (pozițiile 7–12)
- 4 apare de 8 ori (pozițiile 13–20)
- 5 apare de 10 ori (pozițiile 21–30)
- 6 apare de 12 ori (pozițiile 31–42)
- 7 apare de 14 ori (pozițiile 43–56)
- 8 apare de 16 ori (pozițiile 57–72)
- 9 apare de 18 ori (pozițiile 73–90)
- 10 apare de 20 ori (pozițiile 91–110, dar luăm doar până la 100).

Acum trebuie verificat câte dintre aceste valori sunt numere prime: 2, 3, 5, și 7. Numărăm câte dintre primele 100 de elemente sunt formate din aceste valori:

- $2 \rightarrow 4$ apariții
- $3 \rightarrow 6$ apariții
- $5 \rightarrow 10$ apariții
- $7 \rightarrow 14$ apariții

Totalul numerelor prime este $4 + 6 + 10 + 14 = 34$.

Prin urmare, răspunsul corect este varianta **B**.

386. Algoritmii `one` și `two` determină poziția p unde ar trebui inserată valoarea a în vectorul V, dar folosesc abordări diferite. `one` parcurge V și se oprește la primul element care nu respectă $a > V[p]$, în timp ce `two` parcurge tot vectorul și numără câte elemente sunt mai mici decât a.

Pentru ca cei doi algoritmi să returneze aceeași valoare, V trebuie să fie fie constant, fie sortat crescător. Dacă toate elementele sunt egale (**A**), ambele funcții returnează aceeași poziție. Dacă elementele sunt distincte și sortate crescător (**B**) sau dacă sunt sortate crescător dar nu neapărat distincte (**D**), ordinea de parcurgere nu afectează rezultatul.

Prin urmare, răspunsurile corecte de răspuns sunt variantele **A**, **B** și **D**.

387. Algoritmul `suma(n)` este recursiv și adună la rezultatul apelului anterior valoarea n **DIV** $(n+1) + (n+1)$ **DIV** n. Această expresie returnează 1 pentru orice $n > 0$, ceea ce înseamnă că funcția returnează $n + 1$, confirmând că varianta **A** este adevărată.

Apelul `suma(1)` returnează 2, deoarece contribuția sa este 1 **DIV** $2+2$ **DIV** $1 = 0+2 = 2$, ceea ce face ca varianta **C** să fie adevărată.

Varianta **B** este falsă, deoarece algoritmul nu calculează suma divizorilor proprii ai lui n, ci doar efectuează un calcul bazat pe împărțiri succesive. De asemenea, varianta **D** este falsă, deoarece algoritmul nu calculează dublul părții întregi a mediei aritmetice a primelor n numere naturale.

Prin urmare, răspunsurile corecte sunt variantele **B** și **D**.

388. Algoritmul `ceFace(a, b)` reprezintă o implementare recursivă a algoritmului lui

Euclid pentru determinarea celui mai mare divizor comun (**cmmdc**) a două numere. Se repetă operația a **MOD** b sau b **MOD** a până când unul dintre numere devine 0, moment în care se returnează celălalt număr.

Varianta **A** este incorectă, deoarece algoritmul nu returnează suma numerelor a și b, ci **cmmdc**. Varianta **B** este corectă, deoarece dacă unul dintre parametri este 0, algoritmul returnează celălalt număr, ceea ce este în conformitate cu definiția **cmmdc**. Varianta **C** este corectă, deoarece algoritmul aplică exact pașii metodei lui Euclid pentru a găsi **cmmdc**. Varianta **D** este incorectă, deoarece algoritmul nu calculează puterea unui număr.

Așadar, răspunsurile corecte sunt variantele **B** și **C**.

389. Algoritmul `afișare(n)` generează perechi de numere $(i, j-i)$ sau $(j-i, i)$ pe baza unei condiții legate de diferența $j-i$ și de n **DIV** 2. Pentru fiecare pereche (i, j), dacă diferența $j-i$ este mai mică decât n **DIV** 2, se afișează perechea $(i, j-i)$. În caz contrar, dacă diferența nu este n **DIV** 2, se afișează perechea $(j-i, i)$.

Numărul total de perechi generate și afișate în acest caz este 17. Calculul exact al perechilor poate fi verificat prin numărarea efectivă a cazurilor valide.

Așadar, răspunsul corect este varianta **D**.

390. Algoritmul utilizează două bucle imbricate pentru a determina de câte ori se afișează șirul de caractere `UBB`. Variabila n este definită ca $n = 3^k$, unde k este un număr natural.

Buclele funcționează în următorul mod: variabila j începe de la n și este împărțită la 3 la fiecare iterație a primei bucle `while`, rulând astfel de k ori, iar în fiecare iterație a primei bucle, a doua buclă `while` rulează pornind de la $i = 1$ și crește i prin înmulțire cu 3, până când $i > n$. Aceasta rulează exact $k+1$ ori pentru fiecare valoare a j.

Astfel, numărul total de afișări ale șirului `UBB` este $k \times (k+1)$.

Varianta corectă de răspuns este varianta **C**.

391. Secvențele `S1` și `S2` sunt două bucle imbricate care afișează caractere în funcție de valorile lui a și b. În ambele cazuri, bucla exterioară controlează câte linii se vor scrie, iar bucla interioară controlează câte caractere `'*'` se vor afișa pe fiecare linie.

Complexitatea timp pentru ambele secvențe este aceeași, deoarece fiecare conține două bucle imbricate. Astfel, varianta **B** este corectă.

În `S1`, variabila i rulează de la 1 la $b-1$, iar j rulează de la 1 la $a-1$, ceea ce înseamnă că numărul total de caractere afișate este $(a-1) \times (b-1)$, confirmând că varianta **C** este corectă.

392. Algoritmul `ceFace(nr)` analizează fiecare grup de trei cifre consecutive din numărul nr și verifică dacă acestea formează un număr prim folosind funcția `testProprietateNr`. Dacă acest număr este prim, suma cifrelor grupului respectiv este adăugată la rezultatul final.

Pentru a obține rezultatul corect, prima oară se identifică grupurile de trei cifre consecutive. Mai apoi se verifică dacă acestea sunt numere prime și în cazul în care un număr este prim, atunci suma cifrelor acestuia este adăugată la rezultatul final. La final se obține suma rezultată. Prin urmare, răspunsul corect este varianta **D**.

393. Fiecare algoritm încearcă să determine rădăcina pătrată a numărului n, rotunjită în jos la cel mai apropiat întreg.

Algoritmul `radical_A` folosește o metodă bazată pe suma numerelor impare succesive, ceea ce corespunde proprietății matematice conform căreia suma primelor k numere impare este egală cu k^2. Acesta este un mod corect de a calcula rădăcina pătrată, deci varianta **A** este corectă.

Algoritmul `radical_B` utilizează o metodă de căutare binară pentru a găsi rădăcina pătrată, dar are o eroare în ajustarea finală a valorii returnate. Deoarece poate returna o valoare greșită în anumite cazuri, varianta **B** este incorectă.

Algoritmul `radical_C` aplică metoda lui Newton pentru determinarea rădăcinii pătrate, folosind recursivitatea și o aproximație succesivă a rezultatului. Această metodă este corectă, iar partea întreagă a rezultatului este determinată corect, astfel încât varianta **C** este corectă.

Algoritmul `radical_D` utilizează o secvență de creșteri succesive pentru a găsi pătratul perfect corespunzător unui număr, ceea ce duce la o determinare corectă a rădăcinii pătrate. Astfel, varianta **D** este corectă.

Prin urmare, răspunsurile corecte sunt variantele **A**, **C** și **D**.

394. Problema cere să identificăm expresiile care sunt **True** dacă și numai dacă x este un număr par și **nu** aparține intervalului deschis $(10, 20)$.

Varianta **A** folosește **NOT** pentru a exclude intervalul $(10, 20)$, iar condiția **NOT** (x **MOD** $2 = 1$) asigură că x este par. Astfel, această expresie este corectă.

Varianta **B** poate părea corectă la prima vedere, dar expresia ($x < 10$) **OR** ($x > 20$) nu exclude numerele impare. De exemplu, $x = 7$ ar satisface condiția, dar nu este număr par, deci această variantă este incorectă.

Varianta **C** este greșită deoarece condiția ($x > 10$) **AND** ($x < 20$) asigură că x este **în interiorul** intervalului, ceea ce contrazice cerința problemei.

Varianta **D** folosește o expresie echivalentă pentru a verifica dacă x este par. Se exclud numerele impare prin verificarea modulo 4 (x **MOD** $4 = 1$ sau x **MOD** $4 = 3$), iar condiția **NOT** ($x > 10$ **AND** $x < 20$) elimină numerele din interval. Aceasta este o variantă corectă.

Așadar, variantele corecte de răspuns sunt variantele **A** și **D**.

395. Algoritmul `rearanjare(a, n)` trebuie să rearanjeze elementele unui șir strict crescător astfel încât să maximizeze numărul de vârfuri locale. Un vârf local este un element mai mare decât vecinii săi direcți.

Varianta **A** distribuie elementele din partea finală a șirului pe pozițiile pare și continuă cu elementele rămase pe pozițiile impare. Această soluție este corectă.

Varianta **B** plasează elementele din finalul șirului pe pozițiile pare și imediat după fiecare poziție pară plasează următorul element din șir. Această strategie asigură un număr maxim de vârfuri locale. Dacă n este impar, ultimul element este plasat corespunzător. Astfel, varianta este corectă.

Varianta **C** este similară cu varianta **A**, dar plasează elementele pentru pozițiile pare în ordine descrescătoare și pentru pozițiile impare în ordine crescătoare. Aceasta asigură alternanța necesară pentru vârfuri locale, fiind o soluție corectă.

Varianta **D** utilizează distribuie elementele în grupe de câte trei, ceea ce nu garantează întotdeauna un număr maxim de vârfuri locale, făcând-o incorectă.

În concluzie, răspunsurile corecte sunt variantele **A**, **B** și **C**.

396. Algoritmul `f(n, p1, p2)` calculează suma câturilor succesive obținute prin împărțirea lui n la puterile succesive ale lui $p1$, fiecare înmulțită cu $p2$.

Varianta **A** este corectă. Dacă $n = p1 = p2$, atunci la prima iterație c devine n **DIV** $n = 1$, iar în iterația următoare $p1$ devine mai mare decât n, ceea ce încheie bucla, rezultatul final fiind 1.

Varianta **B** este corectă. Dacă $p1 = 5$ și $p2 = 5$, algoritmul calculează suma câturilor succesive obținute prin împărțirea lui n la $5, 5^2, 5^3$, etc. Aceasta reprezintă metoda de a calcula numărul de zerouri terminale din $n!$, deoarece un zero terminal în factorial provine din factori de 10, adică factori de 2 și 5, iar numărul de factori de 5 determină numărul respectiv de zerouri.

Varianta **C** este greșită. Chiar dacă $p1 = p2$, algoritmul nu returnează $\lfloor \log_{p1} n \rfloor$, ci suma câturilor succesive, care poate fi mai mare.

Așadar, singurele variante de răspuns corecte sunt variantele **A** și **B**.

397. Un număr natural n este sumativ dacă n^2 se poate scrie ca sumă a n numere naturale nenule consecutive. Aceasta echivalează cu faptul că n trebuie să fie impar, deoarece suma a n termeni consecutivi este întotdeauna divizibilă cu n dacă și numai dacă n este impar.

Varianta **A** este corectă. Algoritmul parcurge intervalul $[a, b]$ și verifică pentru fiecare i dacă este impar. Numărul total de astfel de valori este incrementat și returnat corect.

Varianta **B** este corectă. Aceasta calculează direct numărul de numere impare din interval, folosind formule aritmetice pentru a determina câte astfel de valori există

Varianta **C** este incorectă. Algoritmul încearcă să verifice dacă n^2 poate fi scris ca sumă de n termeni consecutivi, dar condiția folosită în verificare este eronată, ceea ce îl face incorect.

Varianta **D** este incorectă. Deși încearcă să determine dacă n^2 este suma a n termeni consecutivi, bucla internă parcurge un interval greșit, ceea ce face ca algoritmul să nu funcționeze corect.

398. Algoritmul `ceFace(a, b)` compară cifrele numerelor a și b de la dreapta la stânga.

Atâta timp cât ultimele cifre sunt egale, acestea sunt eliminate prin împărțirea întreagă la 10. Dacă ambele numere ajung la 0 simultan, înseamnă că sunt identice și se returnează *True*. În caz contrar, se returnează *False*. Prin urmare, algoritmul verifică egalitatea completă a numerelor, ceea ce corespunde variantei corecte **B**.

399. Algoritmul `f(a, n)` construiește un vector *b*, unde fiecare element *b[i]* reprezintă suma primelor *i* elemente din vectorul *a*. Inițial, *b[1]* este egal cu *a[1]*, iar apoi fiecare element *b[i]* se obține adăugând *a[i]* la suma anterioară. La final, se returnează *b[n]*, care conține suma tuturor elementelor din *a*. Acest lucru corespunde variantei corecte **A**.

400. Algoritmul trebuie să determine numărul factorilor primi distincți ai unui număr *n*. Un factor prim distinct este un număr prim care divide *n* cel puțin o dată.

Algoritmul `nrFactoriPrimi_B` verifică recursiv divizibilitatea lui *n* cu fiecare divizor prim *d*, eliminând complet multiplii acestuia din *n* și incrementând numărul de factori primi distincți. Se avansează printre divizori, crescând cu 1 după 2, apoi cu 2.

Algoritmul `nrFactoriPrimi_C` parcurge toți divizorii *d* de la 2 la *n*, verificând dacă *d* divide *n*. Dacă da, crește numărul de factori primi distincți și elimină toți multiplii acestuia.

401. Algoritmul `ceFace(n, m)` inversează cifrele numărului *n* printr-un proces recursiv. La fiecare apel, ultima cifră a lui *n* (obținută cu **MOD** 10) este adăugată la sfârșitul lui *m*, iar *n* este redus prin împărțirea întreagă la 10 (**DIV** 10). Acest proces continuă până când *n* devine 0, moment în care se returnează *m*, care conține numărul *n* oglindit. Prin urmare, rezultatul apelului `ceFace(n, 0)` este numărul *n* inversat, ceea ce corespunde variantei corecte **D**.

402. Algoritmul `f(x, n)` parcurge șirul *x* și compară fiecare element cu următorul. Dacă două elemente consecutive sunt egale, algoritmul returnează *False*. În caz contrar, dacă toate elementele consecutive sunt distincte, returnează *True*. Astfel, algoritmul verifică egalitatea elementelor consecutive.

Prin urmare, afirmația **C** este corectă, deoarece algoritmul returnează *False* doar când două elemente consecutive sunt egale. De asemenea, afirmația **D** este corectă, deoarece dacă primele două elemente sunt egale, condiția este îndeplinită și algoritmul returnează

False.

403. Algoritmul `f(x, n)` calculează x la puterea n folosind algoritmul de exponențiere rapidă prin ridicare la pătrat. Dacă n este 0, returnează 1, conform definiției exponențierii. În caz contrar, împarte n la 2 și calculează recursiv x la puterea m. Dacă n este par, rezultatul este p la pătrat, iar dacă este impar, se înmulțește suplimentar cu x. Aceasta confirmă că algoritmul returnează x la puterea n, ceea ce corespunde variantei corecte **A**.

404. Algoritmul `f(x, n)` folosește algoritmul de exponențiere rapidă, care reduce exponenții prin împărțire la 2 la fiecare apel recursiv. Astfel, numărul total de apeluri recursive este proporțional cu numărul de împărțiri succesive la 2, ceea ce înseamnă că complexitatea sa este $O(\log n)$.

Parametrul x este utilizat doar în înmulțirile finale și nu afectează numărul de apeluri recursive, deci complexitatea nu depinde de x. Prin urmare, afirmațiile corecte sunt **B**, deoarece timpul de execuție nu este influențat de x, și **D**, deoarece complexitatea este logaritmică în raport cu n.

405. Algoritmul `afișare(n)` utilizează recursivitatea pentru a afișa numere într-un anumit format. Dacă n este cel mult 4000, acesta este afișat, apoi funcția se apelează recursiv cu *2 * n*, iar după revenirea din apel se afișează din nou n. La apelul `afișare(1000)`, se afișează mai întâi 1000, apoi se apelează recursiv pentru 2000, care afișează 2000 și apelează recursiv pentru 4000. Deoarece 4000 este în limita permisă, acesta se afișează, iar execuția revine la apelurile anterioare, afișând din nou valorile în ordinea inversă. Astfel, ieșirea rezultată este 1000 2000 4000 4000 2000 1000, ceea ce corespunde variantei corecte **B**.

406. Căutarea binară funcționează prin selectarea repetată a elementului din mijlocul unui vector sortat și compararea acestuia cu valoarea căutată. Dacă valoarea elementului căutat este mai mică, căutarea continuă în jumătatea stângă, iar dacă aceasta este mai mare, continuă în jumătatea dreaptă. Pentru ca valoarea 36 să fie comparată succesiv cu 12, 24 și 36, vectorul trebuie să fie astfel structurat încât acești trei pași să apară în procesul de împărțire al vectorului. În cazul vectorilor **B** și **C**, selecțiile din mijlocul vectorului urmează această secvență, ceea ce confirmă că răspunsul corect este **B** și **C**.

407. Operația **MOD** returnează restul împărțirii întregi a lui x la y. Formula echivalentă care exprimă această operație este $x - (y * (x \textbf{ DIV } y))$, deoarece **DIV** calculează câtul împărțirii, iar înmulțirea acestuia cu y oferă cel mai mare multiplu al lui y care nu depășește x. Scăzând acest multiplu din x, obținem restul împărțirii, care este exact valoarea returnată de **MOD**. Astfel, expresia echivalentă cu x **MOD** y corespunde variantei de răspuns **B**.

408. Un număr este divizibil simultan cu 2 și 3 dacă și numai dacă este par și restul împărțirii sale la 3 este 0. Condiția ca un număr să fie par este ca n **MOD** 2 să fie diferit de 1 sau să fie egal cu 0, adică restul împărțirii la 2 să nu fie 1. De asemenea, pentru a fi divizibil cu 3, restul împărțirii sale la 3 trebuie să fie egal cu 0. Expresia care verifică aceste două condiții este $(n \textbf{ MOD } 2 \neq 1) \textbf{ AND } (n \textbf{ MOD } 3 = 0)$, ceea ce corespunde variantei de răspuns **C**.

409. Un număr este divizibil simultan cu 2 și cu 3 dacă și numai dacă restul împărțirii sale la 2 este 0 și restul împărțirii sale la 3 este tot 0. Expresia care verifică aceste două condiții trebuie să returneze *True* doar atunci când ambele resturi sunt 0.

În expresia $(n \textbf{ MOD } 2) + (n \textbf{ MOD } 3) = 0$, suma resturilor este egală cu 0 doar dacă fiecare termen este 0, ceea ce înseamnă că n este divizibil atât cu 2, cât și cu 3. Prin urmare, varianta corectă este **D**.

410. Algoritmul `f(n)` calculează un produs de termeni fracționari, unde fiecare termen are în numitor suma primelor i numere naturale. Variabila s acumulează suma numerelor de la 1 la i, iar la fiecare iterație a buclei p este înmulțit cu $1/s$.

Astfel, expresia evaluată de algoritm este $1/1*1/(1+2)*1/(1+2+3)*...*1/(1+2+3+...+n)$, ceea ce corespunde variantei de răspuns **C**.

411. Algoritmul `prelucrare(s1, lung1, s2, lung2)` compară frecvențele caracterelor din cele două șiruri de caractere *s1* și *s2*. Se inițializează un vector x de frecvență pentru caracterele ASCII din intervalul [1, 125] cu valori nule. Apoi, fiecare caracter din *s1* incrementează corespunzător vectorul de frecvență, iar fiecare caracter din *s2* îl decrementează. La final, se verifică dacă toate valorile din vectorul x sunt zero. Dacă da, atunci înseamnă că cele două șiruri conțin aceleași caractere cu aceleași frecvențe, indife-

rent de ordinea lor, și algoritmul returnează *True*. În caz contrar, returnează *False*.

412. Transformarea unui număr din baza 2 în baza 10 se realizează prin interpretarea sa ca o sumă de puteri ale lui 2. Fiecare cifră binară contribuie la valoarea finală prin înmulțirea cu 2^p, unde p este poziția cifrei, aceasta fiind numărată de la dreapta la stânga și începând de la 0. Astfel, un număr binar de forma $b_k b_{k-1}...b_1 b_0$ se convertește folosind formula:

$$b_k \cdot 2^k + b_{k-1} \cdot 2^{k-1} + ... + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Aplicând această metodă, se obține echivalentul în baza 10 al numărului 1001011001111_2.

413. Algoritmul trebuie să determine prima apariție a valorii x în vectorul a și să afișeze indicele corespunzător sau -1 dacă x nu se găsește în vector. Se parcurge vectorul de la primul la ultimul element, oprindu-se la prima apariție a lui x. Dacă această valoare este găsită, se afișează poziția curentă, iar dacă nu, se afișează -1. Varianta corectă este **D**, deoarece utilizează condiția $a[i] \neq x$ în buclă pentru a căuta prima apariție a lui x, iar la final verifică dacă s-a depășit limita vectorului pentru a decide afișarea lui -1.

414. Algoritmul `f(x)` numără cifrele lui x care sunt divizibile cu 3. Funcția se apelează recursiv, eliminând ultima cifră a lui x la fiecare pas prin împărțirea întreagă la 10. Dacă ultima cifră a lui x este divizibilă cu 3, valoarea returnată se incrementează cu 1, altfel rămâne neschimbată. Procesul continuă până când x devine 0, moment în care funcția returnează 0.

415. Algoritmul `f(n, i, j)` utilizează recursivitatea și condiții bazate pe divizibilitate și împărțirea întreagă pentru a determina ce caractere sunt afișate. Dacă $i > j$, se afișează caracterul '*', iar execuția se oprește. În caz contrar, dacă n este divizibil cu i, funcția se apelează recursiv cu i decrementat. Dacă împărțirea întreagă n **DIV** i este diferită de j, funcția se apelează recursiv modificând i și j, iar după revenirea din apel se afișează '0'. În cazul în care împărțirea întreagă este egală cu j, funcția se apelează cu un pas mai mare și după revenire se afișează '#'.

Pentru apelul `f(15, 3, 10)`, execuția duce la afișarea secvenței '*0#000', ceea ce corespunde variantei corecte **B**.

416. Algoritmul `ceFace(n, x)` inversează ordinea elementelor din vectorul x. Parcurge

prima jumătate a vectorului și interschimbă fiecare element cu elementul corespunzător din partea opusă, utilizând o variabilă auxiliară c pentru stocarea temporară a valorii.

417. Algoritmul `what(n)` verifică dacă toate cifrele numărului n sunt fie 3, fie 7. Acesta analizează cifra unităților utilizând n **MOD** 10. Dacă cifra este 3 sau 7, elimină ultima cifră și se apelează recursiv pentru restul numărului. Dacă găsește orice altă cifră, returnează *False*. Dacă toate cifrele sunt 3 sau 7, în final se ajunge la $n = 0$, caz în care returnează *True*.

Această logică confirmă că algoritmul returnează *False* dacă și numai dacă n conține cel puțin o cifră diferită de 3 și 7, ceea ce validează afirmația **C**. De asemenea, orice cifră pară nu poate fi 3 sau 7, astfel încât prezența unei cifre pare duce automat la *False*, validând afirmația **B**. În plus, algoritmul returnează *True* doar dacă n nu conține cifrele 0, 1, 2, 4, 5, 6, 8, 9, ceea ce confirmă și afirmația **D**.

418. Algoritmul `calcul(x, n)` determină numărul aranjamentelor de n elemente luate câte x. Prima buclă calculează factorialul lui $(n - x)$, iar a doua buclă completează produsul cu factorii de la $(n - x + 1)$ până la n. Prin urmare, algoritmul returnează numărul aranjamentelor de n elemente luate câte x, ceea ce confirmă varianta corectă **C**.

419. Problema determină numărul de găini și iepuri din fermă pe baza a două constrângeri: suma totală a capetelor trebuie să fie n, iar suma totală a picioarelor trebuie să fie m. Fie i numărul găinilor și j numărul iepurilor, atunci următoarele două relații trebuie să fie îndeplinite:$i + j = n$ și $2i + 4j = m$.

Astfel pentru ca un algoritm să determine corect soluția trebuie să parcurgă valori posibile pentru i și j astfel încât să respecte ambele ecuații și afișează toate soluțiile valide. Varianta **A** determină j din ecuația $j = n - i$ și verifică condiția picioarelor. Varianta **B** explorează toate combinațiile posibile de i și j, iar varianta **C** optimizează căutarea limitând j la intervalul $[0, n - i]$, ceea ce face ca acestea să fie variantele corecte. Varianta **D** este incorectă deoarece bucla pentru j se oprește la i, ceea ce duce la excluderea unor soluții valide.

420. Restul împărțirii produsului $n = a \cdot b \cdot c$ la d poate fi calculat folosind proprietățile operației **MOD**. O proprietate importantă este că restul unui produs **MOD** d poate fi

determinat din resturile fiecărui factor:

$$(a \cdot b \cdot c) \textbf{ MOD } d = ((a \textbf{ MOD } d) \cdot (b \textbf{ MOD } d) \cdot (c \textbf{ MOD } d)) \textbf{ MOD } d$$

Varianta **B** respectă proprietatea prezentată, în timp ce celelalte variante nu iau în considerare efectul propagării resturilor sau folosesc împărțirea întreagă (**DIV**), care nu păstrează informațiile necesare despre resturi. Prin urmare, varianta corectă de răspuns este **B**.

421. Algoritmul `det(a, n, m)` verifică dacă există o pereche de elemente în șirul a a căror sumă este egală cu m. Inițial, acesta sortează șirul a în ordine crescătoare utilizând algoritmul de sortare prin selecție (**Bubble Sort**), ceea ce este implementat în liniile 2-10.

După sortare, se folosesc doi indici: unul la începutul șirului și unul la sfârșit. Se verifică suma valorilor acestor doi indici. Dacă suma este mai mică decât m, indicele inferior este incrementat. În cazul în care suma este mai mare, indicele superior este decrementat. Dacă se găsește o pereche care satisface condiția, algoritmul returnează *True*, altfel continuă căutarea. Dacă n este 0, adică șirul este vid, nu poate exista nicio pereche care să satisfacă condiția, iar algoritmul returnează *False*.

Prin urmare, răspunsurile corecte sunt **A**, deoarece algoritmul verifică existența unei perechi cu suma m, **C**, deoarece returnează *False* pentru $n = 0$, și **D**, deoarece prima parte a algoritmului sortează șirul în ordine crescătoare.

422. Algoritmul `magic(n, a)` verifică dacă există două elemente consecutive egale în vectorul a. Dacă n este mai mic decât 2, acesta returnează *False*, deoarece în acest caz nu pot exista elemente duplicate. În caz contrar, parcurge vectorul de la al doilea element până la ultimul și compară fiecare element cu cel anterior. Dacă găsește o astfel de pereche, returnează *True*. Dacă acesta parcurge tot vectorul fără să găsească elemente consecutive egale, returnează *False*. Prin urmare, afirmațiile corecte sunt **A**, **C** și **D**.

423. Algoritmul `f(n, a, b, c)` utilizează recursivitatea pentru a calcula o valoare bazată pe două apeluri recursive și o incrementare cu 1 la fiecare nivel. Când $n = 0$, returnează 1. În caz contrar, efectuează două apeluri către $f(n-1, \dots)$, iar rezultatul

final este suma celor două apeluri recursive plus 1.

Această structură recursivă duce la o creștere exponențială a valorii returnate. Se poate demonstra prin inducție că funcția returnează $2^{n+1} - 1$, ceea ce corespunde variantei **A**. Alternativ, această sumă poate fi rescrisă ca $2^0 + 2^1 + ... + 2^n$, ceea ce corespunde variantei **C**.

Prin urmare, răspunsurile corecte sunt **A** și **C**.

424. Algoritmul `g(n)` verifică dacă un număr este prim prin încercarea tuturor divizorilor de la 2 până la $\sqrt{n}$. Dacă găsește un divizor, returnează *False*, altfel returnează *True*. Aceasta confirmă că afirmația **A** este corectă.

Algoritmul `f(n, p)` determină numărul de moduri în care n poate fi scris ca o sumă de numere prime distincte, luate în ordine strict crescătoare. Acesta parcurge toate valorile posibile pentru p, iar dacă p este prim, îl scade din n și apelează recursiv funcția pentru restul sumei, asigurând că fiecare termen este ales în ordine crescătoare. Astfel, algoritmul returnează numărul de descompuneri ale lui n în sume de numere prime distincte, ceea ce face ca afirmația **B** să fie corectă.

Deoarece funcția `f(n, p)` impune $p \geq 2$ și începe căutarea de la 2, rezultatul apelului `f(n, 2)` este același ca și în cazul `f(n, 1)`, deoarece 1 nu este un număr prim și nu influențează soluțiile. Prin urmare, afirmația **D** este corectă.

425. Algoritmul `AlexB(value, n, k, p)` generează și afișează toate permutările posibile ale numerelor de la 1 la n. Fiecare poziție din șir este completată recursiv cu o valoare incrementată a lui p, iar la fiecare pas se caută prima poziție liberă ($value[i] = 0$) pentru a continua generarea permutării. Când $p = n$, se afișează permutarea generată.

După ce o permutare este afișată, algoritmul revine la pasul anterior și încearcă alte variante, resetând pozițiile deja completate. Aceasta corespunde unui algoritm de generare recursivă a permutărilor.

Pentru $n = 5$, șirul de permutări este generat în ordine lexicografică. Pe a zecea linie de afișare, permutarea este 1 5 2 3 4, ceea ce corespunde variantei corecte **A**.

426. Algoritmul `f(n)` determină numărul de biți de 1 din reprezentarea binară a lui n. Acest lucru se realizează prin utilizarea repetată a operației $n = n\&(n-1)$, care elimină

ultimul bit 1 din n la fiecare iterație. Numărul total de astfel de operații până când n devine 0 este exact numărul de biți de 1 din reprezentarea binară a lui n.

Afirmația **C** este corectă, deoarece numărul returnat de algoritm este exact numărul de numere pare mai mici strict decât n în reprezentarea binară. Similar, afirmația **D** este corectă, deoarece numărul returnat de algoritm corespunde numărului de numere impare mai mici decât n în reprezentarea binară.

427. Algoritmul `calcul(v, n)` determină cea mai lungă secvență de elemente egale din a doua jumătate a șirului v. Se inițializează indici i și j pentru a parcurge elementele din a doua jumătate a șirului și se compară elementele consecutive. Dacă două elemente consecutive sunt egale, atunci secvența curentă este extinsă. La fiecare nouă secvență găsită, se actualizează pozițiile și lungimea celei mai lungi secvențe identificate.

Pentru cazul în care $n = 2$ și elementele sunt consecutive, algoritmul returnează întotdeauna $1, 2$, confirmând afirmația **C**. De asemenea, unul dintre numerele returnate reprezintă lungimea celei mai lungi secvențe de valori egale din a doua jumătate a șirului, ceea ce face ca afirmația **D** să fie corectă.

428. Variabila nr calculează numărul de divizori ai numărului n și se verifică dacă numărul are doi divizori, adică dacă este prim, sau nu. Subalgoritimul returnează *adevărat* dacă numărul n este prim.

429. Pentru ca numărul memorat în t să NU aparțină intervalului (x, y), acesta trebuie să respecte condiția $(t \leq x)$ **SAU** $(t \geq y)$.

430. Al doilea algoritm trebuie să reprezinte varianta recursivă a primului algoritm, așa că, returnând $(n$ **MOD** $2) * (n$ **MOD** $10) +$ `fr`$(n$ **DIV** $10)$, apelul `fr`(n **DIV** 10) merge în recursie, îndepărtând ultima cifră a numărului n procesat.

431. Se observă că numărul stelușelor poate fi descompus în următorul mod: pentru $i = 1 \rightarrow *, i = 2 \rightarrow (1 + 2)*, i = 3 \rightarrow (2 + 3) * ...$ de unde rezultă că numărul stelușelor este egal cu $2 \cdot S_{n-1} + n$, unde S_{n-1} este suma numerelor de la 1 la $n - 1$. Simplificând, ajungem la n^2. $3^2 = 9, 34^2 = 1156, 17^2 = 289$.

432. Variabila a păstrează suma elementelor pare, iar variabila b numărul lor, returnându-se media aritmetică a numerelor pare prin a/b.

433. Se poate observa din instrucțiunile algoritmului că variabila *ok* își schimbă valoarea în momentul în care se găsește o cifră pară în componența elementului curent procesat, astfel elementul nu mai este adăugat la suma curentă, de unde rezultă că algoritmul returnează suma elementelor din vectorul v care au în componența lor doar cifre impare.

434. Algoritmul variantei de răspuns D este vizibil greșit, returnându-se pentru o valoare pozitivă a lui n, opusul său, un număr negativ, ceea ce este incorect. Pentru varianta A, expresia logică $n < 0$ are valoarea 1 dacă n este negativ și 0 dacă n este pozitiv sau zero; când $n < 0$ este adevărat (adică 1), expresia devine $1 * -2 + 1 = -1$, astfel încât rezultatul este $n * (-1)$, ceea ce face ca n să devină pozitiv; când $n < 0$ este fals (adică 0), expresia devine $-2*0+1 = 1$, deci rezultatul rămâne n, algoritmul returnând întotdeauna valoare absolută a lui n. Variantele B și C sunt echivalente. Dacă n este negativ, este returnată valoarea $n * (-1)$, ceea ce îl transformă în pozitiv. Dacă n este pozitiv sau zero, se returnează direct n. Această abordare este o implementare clasică a funcției de modul și este corectă pentru toate cazurile.

435. Expresia este evaluată prin rezultatul a trei expresii independente legate prin **ȘI** , de unde rezultă faptul că este suficient ca o singură expresie din cele trei să fie falsă pentru a evalua întreaga expresie la **fals** . (y **MOD** $2 = 0$) este falsă, 17 fiind un număr impar, așa că rezultatul este **fals** .

436. Algoritmul `ceFace(n, i)` calculează suma tuturor divizorilor numărului n, atât proprii, cât și improprii. Apelul recursiv parcurge descrescător valorile de la n la 1, adăugând la rezultat valorile pentru care n este divizibil cu i. Când $i = 1$, algoritmul returnează valoarea 1, astfel incluzând toți divizorii lui n. Prin urmare, afirmația corectă este D, deoarece algoritmul returnează suma tuturor divizorilor numărului n, inclusiv pe el însuși.

437. Algoritmul `magic(s, n)` parcurge jumătate din șir și compară fiecare caracter de la început cu caracterul corespunzător de la sfârșit. Dacă toate perechile sunt egale, variabila f rămâne 1, ceea ce indică faptul că șirul este un palindrom. În caz contrar, dacă există cel puțin o nepotrivire, f devine 0 și algoritmul returnează această valoare, indicând că șirul nu este un palindrom. Astfel, afirmația corectă este C, deoarece algoritmul verifică

dacă șirul s este un palindrom.

438. Varianta A verifică dacă valoarea absolută a lui x este impară și că x este negativ. Aceasta este condiția exactă căutată, deci varianta este corectă. Varianta B verifică dacă x NU este simultan par și pozitiv. Aceasta include și numere pare negative, deci nu este o condiție suficientă pentru numere impare negative și este incorectă. Varianta C neagă faptul că x este par sau pozitiv. Dacă x este impar și negativ, expresia devine adevărată, deci varianta este corectă. Varianta D include și numere impare pozitive, deoarece verifică doar dacă x este impar sau negativ, nu neapărat ambele condiții simultan. Aceasta este incorectă.

439. Algoritmul calculează și returnează suma cifrelor impare ale numărului n.

440. Pentru a verifica dacă șirul de caractere conține numai cifre este necesară verificarea fiecărui caracter. Dacă numărul de caractere cifră este egal cu lungimea șirului, înseamnă că fiecare caracter este o cifră, variantele de răspuns A și B fiind corecte.

441. Căutarea secvențială parcurge elementele vectorului unul câte unul până găsește elementul căutat sau ajunge la sfârșitul vectorului. În cel mai rău caz, verifică toate cele n elemente, ceea ce duce la o complexitate de timp de $O(n)$. Sortarea prin inserție are complexitatea de $O(n^2)$ în cazul general, deoarece fiecare element trebuie comparat și plasat în poziția corectă în secvența ordonată. În cel mai bun caz (când vectorul este deja sortat), complexitatea este $O(n)$, dar în cazul general nu poate fi garantată o complexitate liniară. Varianta C presupune parcurgerea întregului vector și actualizarea valorii maxime găsite, având o complexitate de $O(n)$. Suma elementelor de pe diagonala principală presupune accesarea exact a n elemente (cele de forma $m[i][i]$). Deoarece se efectuează doar n operații, complexitatea este $O(n)$.

442. Corpul buclei **While** se va executa cel mult o dată pentru apelurile în care b este multiplu de a sau în care m necesita o singură creștere pentru a ajunge la o valoare ce divide b.

443. Algoritmul `f(a, b)` realizează adunarea celor două numere a și b cifră cu cifră, asemenea metodei utilizate în aritmetica manuală, incluzând și transportul dintre poziții.

444. Algoritmul `afisare(M, n)` afișează toate submulțimile mulțimii date M. Pentru

aceasta, utilizează o tehnică bazată pe reprezentarea în sistem binar a numerelor de la 0 la $2^n - 1$. Fiecare bit din reprezentarea binară a unui număr indică prezența sau absența unui element în submulțimea curentă. De asemenea, afișarea submulțimilor este echivalentă cu afișarea tuturor combinărilor elementelor mulțimii M luate câte i, $i = 0, 1, ..., n$.

445. Analiza cazului $a = b$ și $a < c$: cazul de bază: Dacă a **SAU** b **SAU** b) ajunge la 1, rezultatul devine 1. Recursivitate pentru $a = b$:

Când $a = b$, apelul recursiv devine $c * s(a-1, b-1, c-1)$. Aceasta corespunde unei forme de calcul al produsului descrescător de la c până la $(c - a + 1)$, adică: $c * (c-1) * (c-2) \ldots (c-a+1)$. Aceasta este formula pentru $\frac{c!}{(c-a+1)!}$. Aceasta corespunde numărului de aranjamente ale unui set de c elemente luate câte $a - 1$.

446. Algoritmul `h(A, n)` este un subalgoritm recursiv care parcurge șirul de la dreapta la stânga și efectuează un calcul bazat pe paritatea fiecărui element. Dacă elementul $A[n]$ este impar, este scăzut din sumă, iar dacă este par, este adunat la suma curentă. Subalgoritmul returnează diferența dintre suma elementelor impare și suma elementelor pare din șirul.

447. Pe prima pagină sunt copiate $r1 * c = 24$ înregistrări și rămân $5883 - 24 = 5859$. Pe restul paginilor sunt copiate $r * c = 92$ de înregistrări pe pagină, de unde rezultă $\frac{5859}{92} = 63$ de pagini complete. Mai rămân 63 de înregistrări de copiat pe ultima pagină și trebuie distribuite în cele două coloane. Le putem distribui astfel: 32 de înregistrări în prima coloană și 31 de înregistrări în cea de a doua coloană, ultima înregistrare găsindu-se pe rândul 31, sau 31 de înregistrări în prima coloană și 32 de înregistrări în cea de a doua coloană, ultima înregistrare găsindu-se pe rândul 32.

448. Algoritmul `prelucreaza` analizează cifrele numerelor a și b în reprezentarea lor în baza c. Se verifică dacă ultima cifră a ambelor numere este egală cu d, caz în care se face apel recursiv fără a modifica parametrul e. Dacă doar una dintre cifre este egală cu d, parametrul e este incrementat sau decrementat corespunzător. Apelul inițial se face cu `prelucreaza(a, b, c, d, 0)`, ceea ce înseamnă că la fiecare pas valoarea lui e măsoară diferența dintre numărul aparițiilor cifrei d în cele două numere. La final, dacă e rămâne 0, înseamnă că cifra d apare de un număr egal de ori în reprezentările numerelor a și b,

iar subalgoritmul returnează 1. În caz contrar, returnează 0, varianta corectă fiind A.

449. Funcția `val(p, s, i, n, x)` prelucrează coeficienții polinomului în mod recursiv folosind următoarea abordare: Dacă $s + i > n$, se returnează coeficientul curent $p[s]$. Altfel, algoritmul trebuie să combine termenii recursiv, astfel încât să împartă polinomul în două părți echilibrate. Varianta A este incorectă deoarece creșterea dublă a indicelui i împreună cu $x * x$ determină o suprapunere incorectă a termenilor. Varianta B descompune recursiv polinomul în două părți: una care tratează termenii la puterile inferioare, alta care se ocupă de termenii superiori, asigură creșterea corespunzătoare a puterii lui x prin $x * x$, reduce dimensiunea prin $n - i$ care asigură un calcul eficient, varianta fiind corectă. Varianta C este incorectă deoarece ordinea apelurilor nu respectă logica evaluării polinomului. Varianta D este corectă deoarece calculează coeficientul curent $p[s]$, apelul recursiv avansează în șirul de coeficienți cu $s + i$, multiplică rezultatul următor cu x, menținând progresia corectă a termenilor.

450. Varianta A nu funcționează corect din cauza condiției structurii repetitive $d \leftarrow 3, [\sqrt{a}] - 1, 2$. Ca aceasta să funcționeze, variabila d ar trebui să ajungă până la valoarea $[\sqrt{a}]$ inclusiv (algoritmul nu funcționează pentru pătrate perfecte impare precum 9). Varianta B și varianta C sunt corecte, ambele aflând în mod corect dacă numărul are divizori proprii. Varianta D este incorectă deoarece, în cazul unui număr prim, algoritmul nu va returna $fals$ deoarece variabila d va opri execuția structurii repetitive în momentul în care devine 1 și imparte exact valoarea a, dar în cerință este specificat faptul că d trebuie să fie strict mai mare decât 1.

451. A Fals $A \cdot \frac{A^n-1}{A-1} = \sum_{i=1}^{n} A^i$, deci funcția calculează corect din punct de vedere matematic suma cerută. Însă spațiul de reprezentare pe 32 de biți nu va permite calculul corect al sumei pentru orice n din intervalul specificat.

B Fals Spațiul de reprezentare nu permite calculul puterilor pentru orice valori ale lui n.

C Adevărat La fiecare 2 apeluri recursive consecutive ale funcției E1, cel puțin unul dintre apeluri va înjumătăți pe n. Deci complexitatea algoritmului este mărginită de $2\log_2(n)$ și aparține lui $O(\log(n))$. La fiecare apel, funcția E1 returnează valori mod 2022, deci spațiul de reprezentare nu este depășit. Din punct de vedere al corectitudinii matematice,

algoritmul descompune suma astfel:

$$\sum_{i=1}^{n} A^i = (A^k + 1)(\sum_{i=1}^{k} A^i), \text{ dacă } n = 2k$$

$$\sum_{i=1}^{n} A^i = (A^n) + \sum_{i=1}^{n-1} A^i, \text{ dacă } n = 2k + 1$$

Cum funcția mod este distributivă la adunare și înmulțire, putem să o aplicăm pe fiecare termen al descompunerii.

D Fals Similar cu varianta A, matematic este corect, însă spațiul de reprezentare nu permite calculul puterilor pentru orice valori ale lui n.

452. Aceasta problemă este una clasică. Procesul de colorare urmează un model de aritmetică modulară: pornim de la 1 și adăugăm k la fiecare pas, utilizând operația modulo 1000, astfel încât: $urmatorulnr. = (nr.curent + k)$ **MOD** 1000. Dacă 1000 și k sunt prime între ele, toate numerele vor fi colorate, colorându-se un ciclu complet de 1000 de numere. Altfel, procesul va parcurge un subciclu mai mic, determinat de cel mai mare divizor comun $gcd(k, 1000)$, găsindu-se un număr deja colorat în $\frac{1000}{gcd(k,1000)}$ de pași. $\frac{1000}{gcd(15,1000)} = \frac{1000}{5} = 200 \neq 300$. $\frac{1000}{gcd(45,1000)} = \frac{1000}{5} = 200$. $\frac{1000}{gcd(25,1000)} = \frac{1000}{25} = 40$. $\frac{1000}{gcd(30,1000)} = \frac{1000}{10} = 100 \neq 150$.

453. Algoritmul construiește un număr prin extragerea în ordine inversă a cifrelor din numărul n care sunt urmate de o cifră impară până când numărul n ajunge la 0 sau până când sunt întâlnite k cifre pare de la dreapta la stânga.

454. Algoritmul caută recursiv cel mai mare produs între un număr format din ultima cifră a primului parametru concatenată cu cifrele celui de-al doilea parametru și același calcul după eliminarea ultimei cifre din primul parametru.

455. Algoritmul verifică dacă un șir a are aspect de munte, adică $\exists k$, $(1 < k < n)$, astfel încât $a[1] < a[2] < ... < a[k] > a[k+1] > ... > a[n]$. Dacă șirul este crescător, nu îndeplinește proprietatea.

456. Structura repetitivă externă se execută de $\log_2(n)$ ori, iar cea internă de $\log_4(n^4)$ ori. Complexitatea algoritmului poate fi scrisă ca $O(\log_2 n * \log_4 n^4) = O(\log_4 n * 4 * \log_4 n) = O(\log_4^2 n) = O(2 * \log_2^2 n) = O(\log_2^2 n)$, pentru că, în calculul complexităților, constantele

pot fi ignorate.

457. Algoritmul găsește cel mai lung sufix palindromic prin construirea a două valori, hf și hb. Variabila hf se construiește în mod asemănător procesului de construire a unui număr adăugând valori de la dreapta spre stânga, iar hb se construiește asemănător adăugând valori de la stânga spre dreapta. În loc de înmulțirea valoarea 10, aceasta se face cu 2021 și puteri ale acestei valori, dar acest detaliu nu schimbă rezultatul. În momentul în care se găsesc valorile hf și hb egale, se actualizează lungimea maximă, varianta A fiind corectă. Varianta B este greșită deoarece se folosește valoarea 3 în construirea valorilor, ceea ce ar putea cauza erori, iar în varianta C variabilele sunt actualizate eronat.

458. Algoritmul calculează $1! - 2! + 3! - 4! + \cdots + (-1)^{n+1} \cdot n!$, se observă clar din instrucțiunea $P = (-1) \cdot P \cdot i$, că P-ul reprezintă $i!$, dar cu semnul din față alternant. Se observă că semnul din față alternant este $(-1)^{n+1}$, deoarece 2! vine cu '-', înseamnă că toate factorialele numerelor pare vin scăzute, înseamnă că pentru n par, -1 trebuie ridicat la o putere impară, care este $n + 1$, deci se va calcula suma tuturor factorialelor de la 1 la n, cu semnul alternant, adică: $1! - 2! + 3! - 4! + \cdots + (-1)^{n+1} \cdot n!$

459. Determinăm numărul de pagini complet scrise dinainte. Prima pagină conține doar 24 de înregistrări, rămân 3221 de înregistrări. Fiecare pagină va stoca câte 92 de înregistrări, vom mai umple încă $3221 \div 92 = 35$ pagini, iar numărul de înregistrări rămase pentru ultima pagină va fi egal cu $3221 \text{ MOD } 92 = 1$. Deci avem 36 de pagini completate complet, noi suntem pe a 37-a pagină, iar pe această pagină înregistrarea cu număr de ordine 3245 va fi prima. Deci, în concluzie, înregistrarea noastră este pe **Pagina** 37, **Rândul** 1, **Coloana** 1.

460. Algoritmul `ceFace(m)` returnează întotdeauna cifra de control a numărului m. Cifra de control a unui număr se calculează adunând cifrele sale, apoi repetând procesul cu cifrele sumei obținute, până când rezultatul este o singură cifră, care devine cifra de control. Algoritmul calculează cifra de control folosind o proprietate matematică. Toate numerele divizibile cu 9, mai puțin 0, au cifra de control 9, altfel cifra de control este egală cu restul împărțirii numărului la 9. Exemplu: $123456789 \rightarrow 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45 \rightarrow 4 + 5 = 9$

461. Se observă că se folosește metoda `Backtracking` pentru a genera numerele cu n cifre. Dacă numerele vor fi în ordine crescătoare, următorul număr valid va fi 2020.

462. Algoritmul returnează numărul de cifre de 0 din numărul m.

463. Algoritmul returnează 0 dacă șirul nu conține niciun element, returnează -1 dacă variabila p este o poziție invalidă în șirul a și returnează $a[p]$ dacă p este o poziție validă în șirul a.

464. Pentru exemplul $x = [\prime a\prime, \prime b\prime, \prime c\prime, \prime *\prime]$, în i ar trebui să fie valoarea 4 la final. A. Corect, i va avea valoarea 4. B. Greșit, i va avea valoarea 0. C. Greșit, i va avea valoarea 5. D. Greșit, i va avea valoarea 3.

465. Bucla exterioară se execută de $\log_3(n)$ ori, iar bucla interioară se execută de $\log_2(n)$ ori, deci complexitatea algoritmului este $O(\log_3(n) \cdot \log_2(n))$. Se aplică formula schimbării de bază a logaritmului: $\log_a b = \frac{\log_c b}{\log_c a}$, unde c este baza la care se schimbă logaritmul. $O(\log_3(n) \cdot \log_2(n)) = O\left(\frac{\log_2 n}{\log_2 3} \cdot \log_2(n)\right) = O\left((\log_2 n)^2\right)$, $\log_2 3$ nu se mai ia în considerare, fiind constant. $O(\log_3(n) \cdot \log_2(n)) = O\left(\frac{\log_3 n \cdot \log_3 n}{\log_3 2}\right) = O\left((\log_3 n)^2\right)$, $\log_3 2$ nu se mai ia în considerare, fiind constant.

466. Algoritmul calculează câte numere sunt divizibile cu 2 și nu sunt divizibile cu 3 în intervalul $[n, m]$.Cea mai rapidă modalitate de rezolvare este să aflăm numărul de multipli de 2 din intervalul $[n, m]$ din care să scădem numărul de multipli de 6 din intervalul $[n, m]$. $\text{cate}(4, 21) \to M2 - M6 = 6$, $M2 = [4, 21] \Leftrightarrow [2, 10] \Rightarrow 9$, $M6 = [4, 21] \Leftrightarrow [1, 3] \Rightarrow 3$; $\text{cate}(7, 120) \to M2 - M6 = 38$, $M2 = [4, 60] \Rightarrow 57$, $M6 = [2, 20] \Rightarrow 19$; $\text{cate}(1, 215) \to M2 - M6 = 72$, $M2 = [1, 107] = 107$, $M6 = [1, 35] = 35$

467. Algoritmul verifică dacă n conține doar cifre de 0 și 1 în reprezentarea sa în baza 3. Dacă un număr este reprezentat în baza b doar cu cifre de 0 și 1, atunci acesta poate fi scris ca sumă de puteri distincte ale lui b. Vezi Transformarea din baza b în baza 10.

468. Se va trata fiecare variantă în parte: A. Greșit, dacă ultima cifră a lui nr nu este 0, ar trebui să se scadă 1, nu să se împartă la 10. B. Greșit, k nu scade nicăieri. C. Corect. D. Corect, verifică pentru exemplul din cerință.

469. Algoritmul returnează elementul maxim din șirul $[100, x[2], x[3], \ldots, x[n]]$

470. Algoritmul calculează și returnează diferența dintre suma elementelor pare de pe

poziții pare și suma celorlalte elemente din șir.

471. Algoritmul afișează în ordine crescătoare poziția + 1 a cifrelor de 1 din scrierea în baza 2 a numărului n. Numerotarea pozițiilor începe de la 0 de la dreapta la stânga. A. Corect, $31 = 11111_2 \Rightarrow 1\ 2\ 3\ 4\ 5$. B. Corect, $14 = 1110_2 \Rightarrow 2\ 3\ 4$. C. Corect, orice număr impar are cel mai din dreapta bit cu valoarea 1. D. Corect, orice număr de formă 2^k este reprezentat în baza 2: $1000\ldots0$, unde 1 este pe poziția k.

472. Algoritm care presupune metoda Greedy. Problema identică cu problema spectacolelor, dar sub alte cuvinte. Trebuie neapărat ca intervalele să se sorteze după capătul drept, astfel încât să se poată alege intervalele care se termină cel mai devreme. Se alege primul interval care se termină cel mai devreme, apoi se alege următorul interval care începe după capătul drept al intervalului ales anterior. Se repetă procedeul până când nu mai sunt intervale disponibile.

473. Algoritmul afișează pe ecran numerele din intervalul $[a, b]$ care au numărul maxim de divizori.

474. Expresia trebuie să fie a DIV b, dacă a MOD $b = 0$, altfel a DIV $b+1$. Se presupune că a este de forma $b \cdot k + r$, unde r este restul împărțirii lui a la b și $r \in \{0, 1, 2, \ldots, b-1\}$. $(b \cdot k + r) \div b = k$, dacă $r = 0$, altfel $(b \cdot k + r)$ DIV $b = k + 1$. Astfel, clar varianta corectă este C. $\frac{a+b-1}{b} = \frac{a+b-1}{b} + \frac{b}{b} - \frac{b}{b} = \frac{a+2b-1}{b} - 1$. Astfel, varianta D este corectă, de asemenea.

475. Algoritmul folosește Căutarea Binara și caută poziția primei valori dintr-un vector ordonat care este mai mare sau egală cu a.

476. Algoritmul returnează Aranjamente de n luate câte x, $A_n^x = \frac{n!}{(n-x)!}$. După primul For, b va fi egal cu $(n-x)!$ și după al doilea For, a va fi egal cu $n!$. Se va returna a DIV b, adică $A_n^x = \frac{n!}{(n-x)!}$. Se vor efectua exact n operații de înmulțire, deși în $n!$ sunt $n-1$ înmulțiri, primul For înmulțește la primul pas 1 cu 1, astfel se va efectua o înmulțire în plus. Mulțimea de la varianta C reprezintă soluțiile aranjamentelor, adică toate aranjamentele de lungime x, cu termeni distincți, dintr-o mulțime de n elemente, nu neapărat crescători.

477. Algoritmul va afișa pe ecran n valori, reprezentând alipirea șirurilor $1\ 2\ 3\ \ldots\ k\ 1\ 2\ 3$

... k 1 2 3 ..., ultimul șir terminându-se în momentul în care s-au afișat n valori. La punctul C, valoarea va fi calculată prin $5 \times 36 + 2$, adică 36 de șiruri de 5 valori, după care mai pune 1 și 2, astfel se vor afișa 37 valori de 2.

478. A. Fals, algoritmul intră în buclă infinită, contraexemplu: $a = 12$, $b = 18$, $c = 42$. B. Adevărat, se calculează C.M.M.D.C folosind algoritmul lui Euclid prin scăderi, pentru a și b, iar rezultatul cu c. C. Fals, dacă a, b și c sunt egale la apelul inițial, atunci se returnează x, iar x nu va fi inițializat. D. Adevărat, se calculează C.M.M.D.C folosind algoritmul lui Euclid prin împărțiri, pentru a și b, iar rezultatul cu c.

479. Dacă n este par, atunci algoritmul calculează și returnează suma numerelor impare mai mici decât n, altfel calculează și returnează suma numerelor pare mai mici decât n.

480. Se va crea un tabel care va conține pentru fiecare d posibil, cifrele adăugate în b și în c.

d	b	c
0	0	0
1	0	1
2	1	1
3	1	2
4	1	3
5	2	3
6	3	3
7	1	6
8	1	7
9	1	8

Se parcurg ambele numere în paralel, luând perechea formată din ultima cifră din b și ultima cifră din c și verificăm dacă apare în tabel. Dacă o pereche nu apare, înseamnă că perechea respectivă de numere nu poate fi afișată niciodată. A. Fals, toate perechile apar. B. Adevărat, perechea $0, 8$ nu apare. C. Fals, toate perechile apar. D. Adevărat, perechea $4, 5$ nu apare.

481. Algoritmul $f(n, c)$ returnează frecvența cifrei c în numărul n. Algoritmul $g(n, c)$ returnează numărul de cifre distincte din intervalul $[1, c]$ care apar în numărul n.

482. Cel mai scurt cod posibil este: 1, 2, 5, 6, 9, 4, 4, 3, 0, 7, 2, 1. A. Fals, Este posibil să nu se fi generat poziția lui 8. B. Adevărat. C. Fals, Se poate forma doar cu 2 valori de 2. D. Adevărat. Suma cifrelor este 44.

483. Algoritmul calculează și returnează x^n, folosind exponentierea rapidă.

484. Algoritmul f calculează suma maximă a elementelor aflate pe poziții ne-consecutive din șirul a, exceptând suma de la ultimul element. Se vor calcula toate sumele de elemente ne-consecutive și pe revenire se va alege suma maximă. Pentru șirul dat suma maximă este: $10 + 4 + 12 + 11 = 37$.

485. A. Pentru $n = 176$, algoritmul ar trebui să returneze adevărat, deoarece $7 = 1 + 6$, dar returnează fals din cauza că după ce se verifică cifra 7, se va verifica cifra 1 și variabila r va lua valoarea fals. Deci r poate lua adevărat, iar apoi se poate face înapoi fals. B. Pentru $n = 77$, algoritmul ar trebui să returneze adevărat, deoarece $7 = 7$, dar returnează fals. C. Pentru $n = 77$, algoritmul ar trebui să returneze adevărat, deoarece $7 = 7$, dar returnează fals. D. Corect, nici o afirmație nu este adevărată.

486. Algoritmul returnează șirul obținut din eliminarea tuturor duplicatelor unor elemente din șirul x, iar în locul primei apariții a elementului e în șirul x se va insera șirul y în urma eliminării tuturor duplicatelor din acesta, iar în șirul a, restul aparițiilor ale lui e vor fi șterse.

487. Algoritmul calculează recursiv inversul combinărilor de b luate câte a $(1/C_b^a)$ când $a+c = b$ deoarece în acest caz, recursivitatea va genera exact factorii necesari din formula $\frac{a!(b-a!)}{b!}$ prin împărțiri succesive la b până când $a = b$, moment în care va înmulți cu c și va reduce toate numerele, iar când c este diferența $b-a$ (cazul C), algoritmul va genera aceeași formulă printr-o succesiune similară de operații recursive.

488. Algoritmul verifică dacă vectorul de numere pare citit de la stânga la dreapta este identic cu cel citit de la dreapta la stânga cu câteva condiții care influențează modul în care se deplasează indexii, iar acest lucru este adevărat doar în anumite condiții de simetrie.

489. Algoritmul implementează metoda lui Euclid pentru calculul celui mai mare divizor comun (CMMD). Dacă $a = b$, atunci valoarea returnată este direct a. Dacă $a = 0$,

algoritmul se apelează o singură dată și returnează b.

490. Se numără vârfurile care nu au ieșiri (grad extern nul). Se observă că doar un nod nu are muchii care ies din el, ceea ce duce la răspunsul corect.

491. Se evaluează expresia logică în funcție de valorile date pentru $x = 12$ și $y = 23$. Expresia inițială se reduce la evaluarea condițiilor conform priorității operatorilor logici.

492. Algoritmul determină cifra cu cea mai mare frecvență dintr-un număr dat. Parcurge fiecare cifră și calculează frecvența acesteia, returnând una dintre cifrele cu frecvența maximă.

493. Algoritmul returnează numărul total de divizori ai lui n, luând în considerare doar divizorii pozitivi și folosind optimizarea pentru divizorii până la $\sqrt{n}$. Dacă $n < 0$, se consideră valoarea absolută.

494. Algoritmul numără elementele pare dintr-un interval. Rezultatul returnat va fi 0 doar în cazul în care toate elementele din interval sunt impare.

495. Algoritmul afișează pătratele numerelor naturale, utilizând proprietatea sumelor succesive de numere impare.

496. Se verifică dacă cifrele unui număr apar și în celălalt număr în aceeași frecvență. Algoritmul utilizează o metodă de verificare pentru fiecare cifră.

497. Se verifică dacă cifrele numărului sunt în ordine strict descrescătoare. Se compară succesiv cifrele de la dreapta la stânga și dacă toate îndeplinesc condiția, se returnează `True`.

498. Algoritmul calculează media aritmetică a numerelor pare, împărțind suma numerelor pare la numărul elementelor impare.

499. Căutarea se face eficient folosind o combinație de căutare binară pentru fiecare linie a matricei.

500. Algoritmul corect realizează rotația matricei cu 90 de grade în sens trigonometric, păstrând pozițiile corecte.

501. Algoritmul efectuează o rearanjare similară unui pas de partiționare din QuickSort. Elementele mai mici decât pivotul sunt mutate în stânga.

502. Se determină cel mai mare divizor comun al elementelor vectorului printr-o abordare

similară metodei lui Euclid, prin interschimbarea elementelor.

503. Algoritmul convertește numărul în reprezentarea sa în baza 3, returnând un șir de caractere corespunzător.

504. Algoritmul determină dacă un element apare de mai multe ori în vector, comparând fiecare element cu valoarea dominantă găsită.

505. Numără perechile de elemente consecutive diferite printre numerele citite, incrementând contorul de fiecare dată când două elemente consecutive diferă.

506. Algoritmul determină numărul de operații necesare pentru a reduce valoarea lui a la 1, folosind înmulțiri succesive cu 3.

507. Algoritmul determină triplete de numere a căror sumă este egală cu un al treilea număr din vector.

508. Algoritmul utilizează metoda recursivă pentru a verifica toate combinațiile posibile de subseturi care dau suma necesară.

509. Algoritmul implementează o căutare binară modificată pentru a găsi poziția unui element într-un vector parțial ordonat.

510. Calculul combinatoric corect pentru determinarea numerelor strict mai mari formate din cifrele lui n.

511. Transformarea numerotării scaunelor din sala I în sala II respectă regulile specificate în enunț și returnează poziția corectă a scaunului. Cel mai simplu mod de a rezolva aceasta problema este sa desenați modelul sălilor asa cum este descris în enunț și sa va stabiliți o formula înainte de a va uita la răspunsuri.

512. Algoritmul calculează cel mai mare divizor comun al numerelor naturale a și b folosind scăderi repetate în loc de împărțiri (metoda prin scăderi repetate a lui Euclid pentru CMMDC). Cum 2000 și 21 sunt prime între ele, rezultatul este 1.

513. Algoritmul implementează sortarea prin selecție, căutând la fiecare pas cel mai mic element din partea neordonată a vectorului și plasându-l în poziția corectă în partea ordonată. O proprietate a acestei metode de sortare este că după fiecare pas din bucla exterioară, primele i elemente sunt mai mici decât oricare alte elemente din restul vectorului.

514. Algoritmul returnează n, dacă n este impar, sau $2 * n - 1$, dacă acesta este par.

515. Algoritmul returnează 1 dacă, după ce transformă numărul valoarea sa absolută și aplică repetat operația de scădere a dublului ultimei cifre din restul numărului, ajunge la 0 sau 7.

516. Algoritmul afișează recursiv ultima cifră pentru numerele pare după ce elimină ultimele două cifre, iar pentru numerele impare continuă procesul după eliminarea unei singure cifre, până când ajunge la un număr mai mic sau egal cu 9.

517. Algoritmul afișează recursiv numărul curent, apoi înmulțește cu 3 și afișează din nou numărul inițial la întoarcerea din recursivitate, oprind procesul când valoarea depășește 9000. Rezultatul, la mod general, pentru un a dat este: a, $3 * a$, $9 * a$, $27 * a \ldots 27 * a, 9 * a, 3 * a, a$.

518. Algoritmul verifică dacă un vector în care fiecare element începând cu al treilea este suma celor două elemente precedente.

519. In baza 2, numărul $2^{10} - 2^5 - 1$ se obține prin calcularea expresiei 1111111111 $(2^1 0 - 1) - 100000(2^5) = 1111011111$.

520. Algoritmul `two(n, m)` afișează numerele prime din intervalul $[n, m]$ deoarece funcția `one(a,b)` calculează suma divizorilor lui a și b, iar un număr este prim dacă și numai dacă suma divizorilor săi este egală cu numărul plus 1 (care apare de două ori în sumă când $a = b$).

521. Algoritmul verifică dacă un vector este palindrom, comparând elementele din capete spre centru până când indicii se intersectează sau găsește o pereche de elemente diferite.

522. Algoritmul calculează recursiv a^b, înmulțind numărul a cu rezultatul apelului recursiv care scade exponențial b până ajunge la cazul de bază când $b = 0$, care returnează 1.

523. Algoritmul descompune recursiv numărul a în factori primi, căutând cel mai mic număr prim b care îl divide, împărțind apoi a la b și continuând procesul până când ajunge la un număr prim sau la un număr mai mic decât b.

524. Algoritmul construiește un număr nou păstrând doar cifrele pare ale numărului inițial, procesate de la dreapta la stânga, cu excepția cazului când ultima cifră rămasă

este impară, caz în care se returnează numărul construit până în acel moment.

525. Algoritmul caută recursiv ultima cifră impara a numărului dat, returnând -1 dacă nu există o asemenea cifră sau dacă n este 0.

526. Algoritmul construiește un nou număr comparând cifrele de la dreapta la stânga ale celor două numere: pune cifra comună când cifrele sunt egale, sau jumătatea diferenței dintre ele când sunt diferite, multiplicând fiecare rezultat cu puterea corespunzătoare a lui 10.

527. Algoritmul `ceva` verifică dacă două numere au același număr de cifre, iar algoritmul `altceva` adaugă cifre de 1 la sfârșitul lui m până când acesta ajunge să aibă același număr de cifre cu n.

528. Algoritmul împarte recursiv intervalul $[s, d]$ în două jumătăți și verifică pentru fiecare număr din vector aflat pe o poziție din interval dacă cifrele sale, parcurse de la dreapta la stânga, formează o progresie aritmetică cu rația 2, returnând suma numerelor de poziții care satisfac această condiție.

529. Algoritmul implementează o variantă modificată a căutării binare care poate intra în ciclu infinit când elementul căutat x nu există în vector și toate elementele din vector sunt egale, sau când x este un număr impar într-un vector care conține doar numere pare.

530. Algoritmul construiește recursiv un nou număr păstrând doar cifrele pare din numărul inițial în aceeași ordine în care apar, iar dacă numărul are o singură cifră, returnează acea cifră dacă e pară sau 0 dacă e impară.

531. A1(k) generează secvența:

Grupa 1: 1

Grupa 2: $1, 2$

Grupa 3: $1, 2, 3$

Grupa 4: $1, 2, 3, 4$

Grupa 5: $1, 2, 3, 4, 5$

Care concatenată devine: $1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, ...$

A2(k) generează secvența:

Grupa 1: 1

Grupa 2: 2, 1

Grupa 3: 3, 2, 1

Grupa 4: 4, 3, 2, 1

Grupa 5: 5, 4, 3, 2, 1

Care concatenată devine: $1, 2, 1, 3, 2, 1, 4, 3, 2, 1, 5, 4, 3, 2, 1, ...$

Pentru a găsi grupul care conține poziția k, rezolvăm ecuația $n^2 - n - 2k < 0$ care vine din observația că după n grupe avem $\frac{n(n+1)}{2}$ elemente totale, obținând formula $n = \frac{-1+\sqrt{1+8k}}{2}$. După aflarea grupului, calculăm poziția exactă în acel grup folosind $p - (k - \frac{p(p-1)}{2} - 1)$.

532. Algoritmul implementează problema sumei maxime a unei subsecvențe continue (algoritmul lui Kadane), menținând suma curentă în $value2$ și suma maximă găsită în $value1$, resetând suma curentă la 0 când aceasta devine negativă.

533. Algoritmul caută binar prima poziție din vector pentru care $sir[poz] \neq poz$.

534. Algoritmul folosește paritatea lui k pentru a alterna între cele două șiruri de caractere, iar pentru a asigura alternarea corectă, la fiecare pas trebuie să modificăm k cu o valoare impară pentru a-i schimba paritatea și a forța intrarea pe cealaltă ramură a instrucțiunii `If` la următorul apel recursiv.

535. Se consideră numerele:

$$S(0) = 0, \quad S(1) = x[1], \quad S(2) = x[1] + x[2], \quad \ldots, \quad S(n) = x[1] + x[2] + \cdots + x[n].$$

Observăm că avem $(n + 1)$ sume, iar resturile lor la împărțirea la n pot lua cel mult n valori (0, 1, 2, ..., $n - 1$).

A: Dacă $n = 2$ și vectorul $x = [1, 2]$. Atunci: - $S(1) = 1 \equiv 1 \pmod 2$, - $S(2) = 1 + 2 = 3 \equiv 1 \pmod 2$.

Niciun $S(k)$ cu $1 \leq k \leq 2$ nu este 0 modulo 2, deci afirmația A e falsă în acest caz.

B: Prin principiul pușculiței (pigeonhole principle), două dintre aceste sume, $S(i)$ și $S(j)$ cu $0 \leq i < j \leq n$, au același rest modulo n, adică: $S(j) - S(i) \equiv 0 \pmod n$. Dar

$S(j) - S(i) = x[i+1] + x[i+2] + \cdots + x[j]$. Astfel, afirmația B este adevărată.

D: Dacă notăm $k = j - i$, rezultă că suma elementelor din pozițiile $i + 1$ până la j formează exact o subsecvență de k elemente (consecutive) și, după argumentul pentru B, suma aceasta este divizibilă cu n. Deci D este adevărată.

536. Algoritmul `magic(x)` implementează o căutare binară pentru a verifica dacă numărul x este un pătrat perfect. Se inițializează două variabile, `st` și `dr`, care definesc intervalul de căutare între 1 și x. La fiecare iterație a buclei, se calculează mijlocul intervalului (`mj`) și se verifică dacă pătratul acestuia este egal cu x. Dacă da, se returnează **True**, indicând că x este un pătrat perfect. Dacă pătratul lui `mj` este mai mic decât x, se restrânge intervalul la partea superioară, iar dacă este mai mare, la partea inferioară. Dacă bucla se termină fără a găsi un pătrat perfect, se returnează **False**.

Afirmația **A** este falsă, deoarece pentru valori mai mici decât 10, algoritmul returnează **True** pentru 1, 4 și 9, care sunt pătrate perfecte.

Afirmația **B** este falsă, deoarece algoritmul nu descompune numărul x în factori primi, ci doar verifică dacă este pătrat perfect.

Afirmația **C** este adevărată, deoarece algoritmul returnează **True** dacă x este un pătrat perfect.

Afirmația **D** este falsă, deoarece algoritmul returnează **True** pentru pătratele perfecte valide

Așadar, răspunsul corect este **C**.

537. Algoritmul `calculeaza(a, b)` calculează o valoare bazată pe multiplicări succesive ale lui a, păstrând doar ultima cifră la fiecare pas. Variabila x este inițializată cu 1 și apoi, într-o buclă care rulează de b ori, este actualizată ca produsul dintre ultima cifră a lui x și a. La final, algoritmul returnează valoarea lui x.

Afirmația **A** este adevărată. Dacă $a = 107$ și $b = 101$, atunci fiecare multiplicare păstrează doar ultima cifră a produsului, ceea ce face ca rezultatul final să fie 107.

Afirmația **B** este adevărată. Dacă $a = 1001$ și $b = 101$, algoritmul păstrează doar ultima cifră, iar pentru orice putere a lui 1001, rezultatul rămâne 1001.

Afirmația **C** este falsă. Deși pentru anumite valori ale lui a rezultatul poate fi a, acest

lucru nu este general valabil pentru toate valorile între 1 și 10000.

Afirmația **D** este adevărată. Deoarece 1001 are ultima cifră 1, multiplicările succesive vor duce întotdeauna la păstrarea lui 1001, indiferent de valoarea lui b.

Astfel, răspunsul corect este **A, B, D**.

538. Algoritmul `afis(n)` utilizează recursivitatea pentru a afişa valorile lui n și ale subîmpărțirilor sale prin diviziunea întreagă cu 2. Inițial, se afișează valoarea lui n, apoi algoritmul este apelat recursiv pentru n **DIV** 2, iar după revenirea din recursivitate, valoarea n este afișată din nou.

Această metodă produce un șir de numere în care primul număr este n, apoi urmează secvența obținută prin împărțiri succesive la 2, până la 0. După ce se ajunge la 0, valorile sunt afișate din nou, dar în ordinea inversă a apelurilor recursive.

Afirmația **A** este adevărată. Datorită modului în care funcționează recursivitatea, primul element afișat este și ultimul, al doilea este și penultimul, și așa mai departe, cu excepția elementului din mijloc, care apare o singură dată.

Afirmația **B** este falsă. Algoritmul nu afișează doar numere pare, ci și numere impare în funcție de valoarea inițială a lui n.

Afirmația **C** este falsă. Numerele nu sunt afișate mai întâi în ordine crescătoare și apoi în ordine descrescătoare.

Afirmația **D** este adevărată. Secvența rezultată este în ordine descrescătoare în prima parte și apoi în ordine crescătoare după revenirea din recursivitate.

Astfel, variantele corecte de răspuns sunt **A** și **D**.

539. Algoritmul `cauta(n, b)` determină și returnează numărul de cifre 0 din reprezentarea numărului n în baza b.

Pentru a face acest lucru, algoritmul inițializează o variabilă v cu 0, care va conta câte cifre 0 există. Dacă $n = 0$, atunci algoritmul returnează direct 1, deoarece 0 are o singură cifră în orice bază.

Dacă $n \neq 0$, se copiază valoarea lui n în m și se iterează prin reprezentarea sa în baza b. În fiecare iterație, se verifică dacă ultima cifră obținută prin m **MOD** b este 0. Dacă da, se incrementează v. Apoi, m este împărțit la b, continuând procesul până când m devine

0.

Afirmația **A** este falsă. Algoritmul nu returnează numărul de cifre al lui n în baza b, ci doar numără câte dintre aceste cifre sunt egale cu 0.

Afirmația **B** este falsă. Algoritmul nu verifică dacă n este o putere a lui b.

Afirmația **C** este adevărată. Algoritmul returnează numărul de cifre 0 din reprezentarea numărului n în baza b.

Afirmația **D** este falsă. Algoritmul nu verifică dacă n se termină cu cifra b, ci numără doar aparițiile cifrei 0 în reprezentarea sa.

Așadar, varianta corectă de răspuns este varianta **C**.

540. Algoritmul `abc(a, n, p)` verifică dacă parametrul p se află într-un interval valid și returnează o valoare corespunzătoare în funcție de această verificare.

Dacă $n < 1$, algoritmul returnează -1. Dacă p se află în intervalul valid $[1, n]$, returnează elementul de pe poziția p din șirul a. În caz contrar, returnează 0.

Afirmația **B** este singura adevărată, deoarece dacă p este strict mai mare decât 0 și mai mic sau egal cu n, atunci algoritmul returnează $a[p]$, adică elementul de pe poziția p.

541. Algoritmul generează numere cu n cifre utilizând doar cifrele 0, 6 și 7, în ordine crescătoare. Pentru a determina numărul care urmează după 6767, observăm ordinea generării numerelor.

Lista numerelor generate pentru $n = 4$ începe astfel:

$$6000, 6006, 6007, 6060, 6066, 6067, 6070, 6076, 6077, 6600, 6606, \ldots 6767, 6770, 6776, 6777, \ldots$$

După numărul 6767, următorul număr generat conform regulii este 6770. Deoarece această valoare nu se regăsește printre variantele de răspuns oferite, răspunsul corect este `D`.

542. Algoritmul aplică în mod repetat o operație de decrementare definită astfel: dacă ultima cifră a numărului nr nu este 0, se scade 1 din nr; altfel, numărul este împărțit la 10 și se păstrează doar partea întreagă.

Varianta `A` este o implementare recursivă corectă. Se verifică dacă $k = 0$, caz în care se returnează direct nr. Altfel, se aplică regula decrementării și se apelează recursiv funcția. Aceasta menține ordinea corectă a operațiilor și oferă rezultatul corect.

Varianta `B` implementează același proces folosind o structură repetitivă `while`, iterând de k ori și aplicând decrementarea conform regulilor enunțate. Aceasta este o variantă iterativă echivalentă cu `A`, fiind astfel corectă.

Varianta `C` conține o eroare de logică, deoarece schimbă ordinea operațiilor: dacă ultima cifră este mai mare ca 0, în loc să scadă 1, face o împărțire la 10, ceea ce duce la rezultate incorecte în anumite cazuri. Aceasta nu este o soluție corectă.

Varianta `D` oferă o soluție recursivă optimizată, reducând numărul de apeluri recursive prin verificarea directă a ultimei cifre și aplicând eliminarea grupată a zerourilor. Deoarece această implementare respectă regulile problemei și este mai eficientă decât varianta simplă recursivă, este de asemenea corectă.

Așadar, algoritmii corecți care rezolvă problema sunt `A`, `B` și `D`.

543. Algoritmul `fn(v, n)` parcurge vectorul v și verifică fiecare element pentru a determina dacă toate cifrele sale sunt pare.

Pentru fiecare număr *v[i]*, se inițializează variabila `ok` cu valoarea **True**, iar numărul este verificat cifră cu cifră prin împărțiri succesive la 10. Dacă o cifră impară este găsită, `ok` devine **False**, iar numărul nu este considerat valid. În final, se contorizează toate numerele care conțin doar cifre pare.

Varianta `A` este incorectă deoarece algoritmul nu verifică paritatea întregului număr, ci a fiecărei cifre în parte.

Varianta `B` este incorectă deoarece puterile lui 2 nu sunt identificate corect prin verificarea cifrelor pare; de exemplu, 8 este o putere a lui 2 dar conține doar o cifră.

Varianta `C` este corectă, deoarece algoritmul identifică și numără elementele din vector care sunt formate doar din cifre pare.

Varianta `D` este incorectă, deoarece algoritmul verifică doar existența unei cifre impare, dar nu confirmă că toate cifrele sunt impare.

Așadar, varianta corectă de răspuns este varianta `C`.

544. Algoritmul `magic(s, n)` verifică dacă șirul s este un palindrom, adică dacă se citește la fel de la dreapta la stânga și de la stânga la dreapta.

Algoritmul compară fiecare caracter *s[i]* cu caracterul simetric său față de mijlocul șirului,

s[n - i + 1]. Dacă cel puțin o pereche de caractere nu coincide, algoritmul returnează 0. Dacă toate caracterele sunt egale cu omologii lor din partea opusă, algoritmul returnează 1.

Varianta `A` este incorectă, deoarece algoritmul nu verifică paritatea numărului de caractere, ci doar dacă șirul este un palindrom.

Varianta `B` este corectă, deoarece algoritmul compară simetric caracterele și returnează 1 doar dacă șirul este un palindrom.

Varianta `C` este incorectă, deoarece expresia $n - i + 1$ nu poate deveni negativă. Valoarea minimă a lui i este 1, iar în acest caz, expresia devine $n - 1 + 1 = n$, care este întotdeauna validă.

Varianta `D` este incorectă, deoarece algoritmul nu verifică dacă șirul conține doar caractere alfanumerice.

Așadar, varianta corectă este `B`.

545. Algoritmul citește un număr natural a și parcurge două bucle imbricate pentru a genera perechi de valori *(i, j)* conform unor condiții specifice.

Prima buclă for iterează variabila i de la 1 la $a - 1$, iar cea de-a doua buclă for parcurge variabila j de la $i + 2$ până la a. Pentru fiecare pereche *(i, j)*, algoritmul verifică dacă suma $i + j$ este mai mare decât $a - 1$. Dacă această condiție este îndeplinită, se afișează tripletul format din a, i și j, iar pe fiecare linie a ieșirii se va afișa un astfel de triplet.

Pentru $a = 9$, numărul total de soluții afișate este 19, ceea ce corespunde variantei `C`.

546. Algoritmul `ceFace(n)` calculează suma cifrelor pare ale numărului n.

Se inițializează variabila s cu valoarea 0. Apoi, într-o buclă `while`, algoritmul extrage ultima cifră a lui n folosind operația **MOD** 10 și verifică dacă este pară. Dacă da, o adaugă la sumă. După fiecare iterație, elimină ultima cifră din n printr-o împărțire la 10.

Pentru apelul `ceFace(9876)`, cifrele pare sunt 8, 6 și suma acestora este $8 + 6 = 14$. Prin urmare, algoritmul returnează valoarea `C`.

547. Algoritmul `generare(n)` calculează numărul de transformări succesive aplicate asupra lui n până când se ajunge la o valoare deja generată anterior.

Se folosește un vector `used` pentru a marca valorile întâlnite și se inițializează numărătorul

nr cu 0. Algoritmul transformă n iterativ prin înlocuirea lui cu suma cuburilor cifrelor sale, până când se ajunge la o valoare deja vizitată.

Astfel, variantele corecte de răspuns sunt `A`, `C` și `D`.

548. Algoritmul `f(a, b)` are o problemă de recursivitate infinită, deoarece nu există o condiție de oprire corect definită. Se observă că atunci când a este diferit de 0, apelul recursiv este făcut fie cu a împărțit la 2 și b înmulțit cu 2, fie cu a înmulțit cu 2 și b împărțit la 2. Prima variantă reduce treptat valoarea lui a și crește valoarea lui b, iar a doua variantă crește a și reduce b.

Analizând apelul `f(20, 10)`, se observă că în prima ramură a condiției, a se reduce la jumătate, iar b se dublează succesiv: `f(10, 20)`, `f(5, 40)`, `f(2, 80)`, `f(1, 160)`, `f(0, 320)`. Astfel, prima dată când a devine 0, valoarea lui b este 320.

Prin urmare, răspunsul corect este `C`.

549. Pentru ca numărul n să îndeplinească condițiile din enunț, acesta trebuie să fie divizibil cu 3 și să aibă ultima cifră 4 sau 6. Analizând fiecare variantă:

Varianta A este incorectă deoarece expresia (n **MOD** $10 = 4$) **AND** (n **MOD** $10 = 6$) este mereu falsă, întrucât un număr nu poate avea simultan ultima cifră 4 și 6.

Varianta B este corectă deoarece (n **MOD** $6 = 0$) asigură divizibilitatea cu 3 și 2, iar ((n **MOD** $10 = 4$) **OR** (n **MOD** $10 = 6$)) garantează că ultima cifră este 4 sau 6.

Varianta C este incorectă deoarece (n **MOD** $9 = 0$) asigură doar că suma cifrelor lui n este divizibilă cu 9, nu că n este divizibil cu 3. Astfel, condiția nu este suficientă pentru a garanta corectitudinea expresiei.

Varianta D este incorectă deoarece verifică dacă n este divizibil cu 3 și dacă ultima cifră este determinată printr-o combinație de modulo 2 și 5, ceea ce nu garantează că ultima cifră este exact 4 sau 6.

Prin urmare, răspunsul corect este `B`.

550. Pentru ca expresia logică (X **OR** Z) **AND** (X **OR** Y) să fie evaluată ca **True**, trebuie ca ambele paranteze (X **OR** Z) și (X **OR** Y) să fie evaluate ca **True**.

În varianta A, X și Y sunt **False**, iar Z este **True**. Evaluând expresia, (X **OR** Z) devine **True** deoarece Z este **True**, dar (X **OR** Y) devine **False** deoarece atât X, cât și Y sunt

False. Astfel, expresia finală este **False**, deci această variantă nu este corectă.

În varianta B, X este **True**, iar Y și Z sunt **False**. Evaluând expresia, (X **OR** Z) devine **True** deoarece X este **True**, iar (X **OR** Y) devine, de asemenea, **True** din același motiv. Astfel, expresia finală este **True**, ceea ce face această variantă corectă.

În varianta C, X și Z sunt **False**, iar Y este **True**. Evaluând expresia, (X **OR** Z) devine **False** deoarece ambele sunt **False**, iar (X **OR** Y) devine **True** deoarece Y este **True**. Deoarece una dintre paranteze este **False**, expresia finală este **False**, deci această variantă nu este corectă.

În varianta D, toate valorile X, Y și Z sunt **True**. Evaluând expresia, atât (X **OR** Z), cât și (X **OR** Y) devin **True**, ceea ce face expresia finală **True**. Astfel, această variantă este corectă.

Prin urmare, variantele corecte de răspuns sunt `B` și `D`.

551. Pentru a determina câte șiruri respectă condițiile problemei, analizăm mai întâi totalul de șiruri ordonate strict crescător și apoi verificăm câte dintre acestea au un număr impar de consoane.

Pentru $l = 1$, șirurile valide sunt formate dintr-o singură literă. Există 5 litere în total (a, b, c, d, e), dintre care 3 sunt consoane (b, c, d). Astfel, avem 3 șiruri valide de lungime 1.

Pentru $l = 2$, trebuie să alegem două litere distincte din cele 5, în ordine strict crescătoare. Numărul total de astfel de combinații este $\binom{5}{2} = 10$. Pentru ca un șir de lungime 2 să aibă un număr impar de consoane, trebuie să conțină exact una dintre cele 3 consoane disponibile. Alegând una dintre cele 3 consoane și o vocală, obținem $3 \times 2 = 6$ șiruri valide.

Pentru $l = 3$, trebuie să alegem trei litere distincte din cele 5, în ordine strict crescătoare. Numărul total de astfel de combinații este $\binom{5}{3} = 10$. Pentru ca un șir de lungime 3 să aibă un număr impar de consoane, trebuie să conțină exact o consoană sau toate cele trei consoane. Există $\binom{3}{1} \times \binom{2}{2} = 3 \times 1 = 3$ șiruri care conțin exact o consoană și 1 șir care conține toate consoanele. În total, avem 4 astfel de șiruri.

Adunând cazurile pentru fiecare lungime obținem: $3+6+4 = 13$. Astfel, varianta corectă

de răspuns este `B`.

552. Pentru a determina numărul de asteriscuri și puncte necesare pentru construirea pătratului cu diagonale, analizăm modelul general.

Un pătrat de latură n conține în total $n \times n$ caractere. Toate caracterele de pe margine și pe diagonale sunt asteriscuri, iar restul sunt puncte.

Liniile de margine, adică prima și ultima linie, conțin fiecare n asteriscuri. Liniile intermediare conțin două asteriscuri pe margine și două asteriscuri pe fiecare diagonală. Deoarece cele două diagonale se intersectează în mijloc atunci când n este impar, trebuie să avem grijă să nu dublăm numărarea acelui punct.

Pentru $n = 7$, avem $7 \times 7 = 49$ caractere în total. Numărul de asteriscuri de pe margine este $7+7+5+5 = 24$ (margini orizontale și verticale). Cele două diagonale au împreună $7 + 7 - 1 = 13$ asteriscuri , deoarece eliminăm punctul comun. Astfel, avem $24 + 9 = 33$ asteriscuri și $49 - 33 = 16$ puncte. Aceasta confirmă că varianta `C` este corectă.

Pentru $n = 18$, numărul total de caractere este $18 \times 18 = 324$. Marginea conține $18+18+16+16 = 68$ asteriscuri, iar diagonalele contribuie cu $18+18-1 = 35$ asteriscuri. Totalul de asteriscuri este 100, iar punctele rămase sunt $324 - 100 = 224$. Aceasta confirmă că varianta `D` este corectă.

Astfel, răspunsul final este reprezentat de variantele de răspuns `C` și `D`.

553. Algoritmul `ceFace(T, n, e)` verifică dacă valoarea e se află în șirul T și returnează poziția acesteia sau 0 dacă nu este găsită.

Dacă e este un număr par, algoritmul utilizează căutarea binară. Se inițializează două variabile, a și b, care definesc limitele intervalului de căutare. Algoritmul compară e cu elementul din mijlocul intervalului și îngustează căutarea la jumătatea corespunzătoare până când fie găsește valoarea căutată, fie determină că aceasta nu există în șir. În cazul în care e nu se află în șir, algoritmul returnează 0.

Dacă e este un număr impar, algoritmul utilizează căutarea secvențială. Începe de la prima poziție a șirului și verifică fiecare element până când găsește e sau ajunge la finalul șirului. Dacă valoarea este găsită, returnează poziția acesteia, altfel returnează 0.

Afirmația conform căreia algoritmul returnează 0 dacă e nu se află în șirul T este

adevărată. De asemenea, dacă e este impar și se află în șir, algoritmul utilizează căutarea secvențială pentru a determina poziția sa, ceea ce confirmă a doua afirmație corectă. Pe de altă parte, dacă e este impar, algoritmul nu folosește căutarea binară, iar dacă este par, există posibilitatea să nu fie găsit, ceea ce face ca celelalte afirmații să fie false.

Astfel, variantele corecte de răspuns sunt `A` și `C`.

554. Algoritmul `calcul(x, n)` calculează o valoare bazată pe două etape de adunare. Prima buclă for acumulează o sumă inițială în variabila b, adăugând toate valorile de la 1 până la $n - x$. A doua buclă for continuă adunarea în variabila a, adăugând valorile de la $n - x + 1$ până la n. La final, algoritmul returnează diferența dintre cele două sume.

Dacă $n = 5$ și $x = 2$, valorile adunate în prima buclă sunt $1 + 2 + 3$, iar în a doua buclă se adaugă $4 + 5$. Diferența dintre suma finală și suma inițială este 5, ceea ce confirmă varianta B. Deoarece algoritmul efectuează doar operații de adunare pe valori strict pozitive, rezultatul său va fi întotdeauna un număr strict mai mare decât 0, ceea ce confirmă varianta D.

Așadar, variantele corecte de răspuns sunt `B` și `D`.

555. Algoritmul `s(a, b, c)` este o funcție recursivă care efectuează operații de înmulțire pe parametrii de intrare în funcție de relațiile dintre aceștia. Dacă oricare dintre valori este 1, algoritmul returnează 1. În caz contrar, aplică recursiv înmulțiri succesive modificând valorile lui a, b și c până când una dintre acestea devine 1.

Pentru cazul în care $a = b$ și $a < c$, execuția algoritmului urmează ramura care apelează `s(a - 1, b - 1, c - 1)`, ceea ce duce la o înmulțire succesivă a valorilor lui c până la $c - a + 1$. Aceasta corespunde expresiei $c!/(c - a + 1)!$, ceea ce confirmă varianta B ca fiind corectă.

Astfel, varianta corectă de răspuns este `B`.

556. Pentru cazul în care $a = 3$, $b = 4$ și $c = 7$, algoritmul `s(a, b, c)` urmează diferite ramuri în funcție de relațiile dintre parametrii de intrare. Deoarece $a < b$, se execută apelul recursiv $s(3, 3, 7)$. Acum, cum $a = b$, se va executa ramura care returnează $c \times s(2, 2, 6)$. Continuând acest proces, se ajunge la $7 \times 6 \times 4$, ceea ce oferă rezultatul final 168. Astfel, varianta D este corectă.

557. Algoritmul `numere(a, b, c, d, e)` compară frecvența apariției cifrei d în reprezentările în baza c ale numerelor a și b. Pentru fiecare cifră extrasă din a care este egală cu d, variabila e este incrementată, iar pentru fiecare cifră extrasă din b care este egală cu d, variabila e este decremenată. La final, dacă e este 0, înseamnă că a și b conțin cifra d de același număr de ori, iar algoritmul returnează **True**, în caz contrar returnează **False**. De asemenea, algoritmul parcurge cifrele lui a și b în mod sincron, astfel încât schimbare lui a cu b în apel nu modifică rezultatul returnat.

Așadar, variantele corecte de răspuns sunt `A` și `C`.

558. Șirul s este generat conform unei reguli de concatenare, în care fiecare termen este format prin alăturarea cifrelor ultimilor doi termeni ai șirului. Primii doi termeni sunt x și $x+1$, iar toți ceilalți sunt construiți iterativ pe baza celor anteriori.

Numărul de cifre din fiecare termen crește exponențial, iar pentru a determina termenul precedent unui termen cu $k1$ cifre, trebuie să parcurgem secvențial termenii șirului și să verificăm numărul lor de cifre. În fiecare caz, identificăm primul termen care are cel puțin k cifre, determinăm termenul precedent și verificăm numărul său de cifre.

Aplicând această metodă, se obțin următoarele rezultate corecte: pentru $x = 15$ și $k = 8$, numărul cifrelor termenului căutat este 6; pentru $x = 5$ și $k = 12$, numărul cifrelor termenului căutat este 8.

Astfel, variantele corecte de răspuns sunt `A` și `C`.

559. Algoritmul `fibonacci(n)` este o implementare recursivă a secvenței Fibonacci, unde pentru fiecare apel cu $n > 1$, funcția se autoapelează de două ori: o dată cu $n - 1$ și o dată cu $n - 2$.

Mesajul "Aici" este afișat doar atunci când $n \leq 1$, ceea ce corespunde cazurilor de bază ale recursiei. Fiecare astfel de apel corespunde unei frunze din arborele recursiv de apeluri, iar numărul total de frunze într-un astfel de arbore pentru `fibonacci(n)` este exact `fibonacci(n)`.

Astfel, numărul total de afișări ale mesajului "Aici" este exact egal cu valoarea lui `fibonacci(n)`, ceea ce face ca afirmația A să fie corectă.

560. Pentru a calcula eficient valoarea expresiei $E(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3 + a_4 * x^4$,

trebuie să determinăm numărul minim de operații de înmulțire necesare.

O abordare directă presupune calculul separat al fiecărei puteri a lui x, ceea ce ar duce la patru înmulțiri pentru x^2, x^3, x^4. Ulterior, fiecare termen $a_i * x^i$ ar necesita încă patru înmulțiri, rezultând un total de șapte înmulțiri. Această abordare nu este optimă.

O metodă mai eficientă este aplicarea Schemei lui Horner, care permite factorizarea expresiei astfel:

$$E(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + xa_4)))$$

Prin această metodă, se efectuează doar patru înmulțiri: fiecare factor x este multiplicat succesiv în expresie. Aceasta este soluția optimă, ceea ce face ca răspunsul corect să fie A.

561. Algoritmul `f(x, n)` calculează valoarea lui x^n utilizând metoda exponențierii rapide, reducând exponența la jumătate la fiecare pas. În loc să efectueze n înmulțiri, algoritmul recurge la apeluri recursive care reduc n prin împărțirea la 2, ceea ce duce la un număr de aproximativ $\log_2 n$ apeluri recursive.

În fiecare apel, dacă n este par, algoritmul returnează pătratul valorii obținute pentru exponentul redus, iar dacă n este impar, acesta înmulțește rezultatul parțial cu x. Acest comportament este corect pentru orice valoare naturală a lui n, indiferent dacă este par sau nu.

Prin urmare, algoritmul returnează x^n în general, fără a fi necesar ca n să fie o putere a lui 2 sau să fie par. Deoarece timpul de execuție este de ordin $\mathcal{O}(\log_2 n)$ datorită împărțirii repetate a exponentului, răspunsul corect este varianta B.

562. În noua variantă a algoritmului, linia 10 a fost modificată astfel încât, în loc de `return x * p * p`, acum returnează `return x * f(x, n - 1)`. Această modificare elimină avantajul exponențierii rapide, transformând algoritmul într-o implementare recursivă directă a exponențierii.

În această nouă variantă, algoritmul reduce exponentul cu 1 la fiecare apel recursiv, ceea ce înseamnă că efectuează aproximativ n apeluri recursive, în loc de $\log_2 n$. Totuși, deoarece fiecare apel efectuează exact o înmulțire cu x, rezultatul final rămâne corect și algoritmul continuă să returneze x^n. Astfel, varianta corectă este D.

563. Algoritmul `f(arr, i, n, p)` determină suma maximă a unui subset de elemente din șirul *arr*, respectând condiția ca două elemente consecutive din subset să nu fie adiacente în șirul original.

Funcția recursivă explorează două opțiuni pentru fiecare element *arr[i]*: fie îl exclude și apelează recursiv pentru elementul următor (`n1`), fie îl include, dar doar dacă nu este imediat după ultimul element selectat (`n2`). Rezultatul final este determinat prin compararea celor două variante și alegerea maximului folosind funcția auxiliară `f2(a, b)`.

Pentru șirul $(10, 1, 5, 4, 7, 12, 1, 12, 6)$ și apelul `f(arr, 1, 9, -10)`, algoritmul calculează suma maximă de elemente neadiacente. În acest caz, suma maximă este 39, astfel încât răspunsul corect este C.

564. Algoritmul `f(n)` conține două bucle imbricate care influențează complexitatea sa temporală. Prima buclă exterioară are variabila j inițializată la n și se execută până când j devine 1, fiind împărțită la 2 în fiecare iterație. Aceasta determină un număr de iterații proporțional cu $\log_2 n$.

Cea de-a doua buclă, interioară, are variabila i inițializată la 1 și se execută până la n^4, dublându-se de fiecare dată. Numărul total de iterații este proporțional cu $\log_4(n^4)$, ceea ce duce la un număr de pași egal cu $O(\log_4^2 n)$.

Astfel, complexitatea timp a algoritmului se încadrează în clasele $O(\log_2^2 n)$ și $O(\log_4^2 n)$, ceea ce face ca răspunsul corect să fie B și C.

565. Algoritmul `f(n)` determină un model de afișare bazat pe două bucle imbricate. Bucla exterioară reduce valoarea j prin împărțirea la 2 în fiecare iterație, ceea ce determină un număr de pași proporțional cu $\log_2 n$.

În fiecare iterație a buclei exterioare, bucla interioară afișează un număr de steluțe, acesta fiind determinat de creșterea exponențială a variabilei i, de la 1 până la n^4, în pași multiplicați cu 4. Numărul total de steluțe afișate depinde de numărul de iterații ale acestei bucle.

În plus, dacă valoarea j **DIV** $2 > 1$, algoritmul afișează un spațiu între grupurile de steluțe. Acest lucru influențează modul în care sunt separate grupurile de caractere afișate.

Pentru $n = 10$, se confirmă că sunt afișate grupuri de câte 7 steluțe, separate prin câte un spațiu, ceea ce face afirmația A corectă. În cazul $n = 100$, algoritmul ajunge să afișeze 84 de steluțe și 5 caractere spațiu, ceea ce confirmă și afirmația D.

566. Expresia va avea valoare ADEVĂRAT dacă și numai dacă $a < 5$ și $a > 2$, deci $a \in \{3, 4\}$.

567. Algoritmul returnează numărul total de factori de 3 din toate elementele vectorului v.

568. In expresia dată se adună parităților a două numere consecutive. Întotdeauna unul este par și unul impar. Întotdeauna expresia va avea valoarea 1.

569. Algoritmul realizează sortarea crescătoare a elementelor vectorului x, folosind metoda Bubble Sort. Variabila p este inițializată cu 1 și este incrementată cu 1 de $n - 1$ ori, astfel încât p va avea valoarea n. La final, n va lua valoarea lui p, deci nu va fi decrementat.

570. Algoritmul returnează suma numerelor impare din vectorul a.

571. Algoritmul returnează numărul de numere impare din vectorul v.

572. Algoritmul returnează adevărat dacă și numai dacă numărul x este pătrat perfect, fals altfel. Se caută binar rădăcina pătrată a lui x în intervalul $[1, x]$.

573. Algoritmul calculează în b oglinditul lui a. Se verifică dacă a este palindrom; dacă da, se returnează adevărat, altfel fals.

574. A. Adevărat, pentru $a = 2021$, valoarea lui x va fi întotdeauna 2021, oricare ar fi b, deoarece x va fi întotdeauna 1×2021. B. Adevărat. C. Adevărat, se observă că x-ul se va repeta din 4 în 4, pentru b de forma $4k + 1$ se returnează 7777, pentru b de forma $4k + 2$ se returnează 7×7777, pentru b de forma $4k + 3$ se returnează 9×7777, pentru b de forma $4k$ se returnează 3×7777. $b = 2021$ este de forma $4k + 1$, astfel se returnează 7777. D. Fals, valoarea returnată este influențată de variabila a, nu de b.

575. Numărul de elemente de pe cele două diagonale ale unei matrice pătratice este $2 \times n - 1$, dacă n este impar și $2 \times n$ dacă n este par.

576. A. Fals, NU (((1 > 0) ȘI (0 < 1)) SAU (1 > 1)) ⇒ NU ((ADEVĂRAT ȘI ADEVĂRAT) SAU FALS) ⇒ NU (ADEVĂRAT SAU FALS) ⇒ NU (ADEVĂRAT)

⇒ FALS. B. Adevărat, ((0 > 0) ȘI (0 < 1)) SAU ((1 > 0) ȘI (1 < 2)) ⇒ (FALS ȘI ADEVĂRAT) SAU (ADEVĂRAT ȘI ADEVĂRAT) ⇒ FALS SAU ADEVĂRAT ⇒ ADEVĂRAT. C. Adevărat, (NU (1 > 0)) SAU (NU (0 > 0)) ⇒ (NU ADEVĂRAT) SAU (NU FALS) ⇒ FALS SAU ADEVĂRAT ⇒ ADEVĂRAT. D. Adevărat, (1 > 0) SAU ((0 > 0) ȘI (0 < 0)) SAU (1 < 1) ⇒ ADEVĂRAT SAU (FALS ȘI FALS) SAU FALS ⇒ ADEVĂRAT SAU FALS SAU FALS ⇒ ADEVĂRAT.

577. Algoritmii de la A, B, C calculează același lucru, suma elementelor de pe diagonala principală a matricei e. Rezultatul de la D este diferit, niciodată i nu va fi egal cu j, deoarece j pleacă de la $i + 1$ întotdeauna.

578. Algoritmul returnează cel mai mic divizor al lui b care este mai mare sau egal cu a.

579. afis(4) = 4 afis(3) 4, afis(3) = 3 afis(2) 3, afis(2) = 2 afis(1) 2, afis(1) = 1 afis(0) 1, ⇒ afis(0) = 0, afis(1) = 1 0 1, afis(2) = 2 1 0 1 2, afis(3) = 3 2 1 0 1 2 3, afis(4) = 4 3 2 1 0 1 2 3 4

580. Se realizează conversia numărului din baza 10 în baza x sau se realizează conversia numărului din baza x în baza 10. Aplicăm metoda de a transforma numărul din baza 10 în baza x. A. Adevarat, $67_{(10)} = 232_{(5)}$, $232 \leq 232$. B. Fals, Nu există $323_{(3)}$, 3 nu poate apărea în baza 3. C. Adevarat, $67_{(10)} = 1003_{(4)}$, $232 \leq 1003$. D. Fals, $67_{(10)} = 151_{(6)}$, $232 > 151$.

581. A. Fals, algoritmul intră în buclă infinită în momentul în care șirul a are cel puțin 2 elemente de 0 la apelul inițial. B. Algoritmul numără câte elemente de 0 sunt în șirul a, iar apoi suprascrie ultimele k elemente din șir cu 0, unde k reprezintă numărul de elemente de 0. Se pierd ultimele elemente. C. Adevărat. D. Fals, algoritmul nu ține cont de ordinea relativă a elementelor șirului a din apelul inițial.

582. Se aplică formula de la Suma Gauss, $\frac{n \cdot (n+1)}{2}$. Numerele de la 1 la 20 apar pe primele $\frac{20 \cdot 21}{2} = 210$ poziții. Prima poziție pe care apare valoarea 21 este 211, iar 21 apare de 21 ori, deci ultima poziție va fi 231. Pozițiile din intervalul $[211, 231]$.

583. $f(3, 2) \Rightarrow$ "FMI" $f(-1, 1) \Rightarrow$ "FMI" "FMI" $f(0, 0) \Rightarrow$ "FMI" "FMI" "FMI" 1, se scrie de 3 ori textul "FMI" și se returnează 1.

$f(2, 3) \Rightarrow$ "FMI" $f(-1, 1) \Rightarrow$ "FMI" "FMI" $f(0, 0) \Rightarrow$ "FMI" "FMI" "FMI" 1, se scrie

de 3 ori textul "FMI" și se returnează 1.

$f(f(3,2),f(2,3)) \Rightarrow f(1,1) \Rightarrow$ "FMI" $f(0,0) \Rightarrow$ "FMI" "FMI" 1, se scrie de 2 ori textul "FMI" și se returnează 1.

Pentru secvența de cod $f(f(3,2),f(2,3))$ se va scrie 8 ori textul "FMI".

584. Apelul `ceFace(n, 2)` returnează suma divizorilor proprii ai lui n, din cauza faptului că i se apelează cu valoarea 2. 1 și n nu se iau în calcul.

585. Dacă e este par, atunci algoritmul verifică dacă e apare în șirul T folosind căutare binară, altfel verifică dacă e apare în șirul T folosind căutare secvențială.

586. Numărul de asteriscuri este $3 \times (n-1)$, iar numărul total de caractere este $n^2 + \frac{n \times (n-1)}{2}$. Numărul de puncte = Numărul total de caractere - Numărul de asteriscuri, adică $n^2 + \frac{n \times (n-1)}{2} - 3 \times (n-1)$. A. Fals, pentru $n = 2$ avem 3 asteriscuri și 2 puncte. B. Adevarat, pentru $n = 7$ avem 18 asteriscuri și 52 de puncte. C. Fals. D. Adevarat, pentru $n = 15$ avem 42 asteriscuri și 288 de puncte.

587. A. Fals, pentru n par, stânga nu va ajunge egală cu dreapta, astfel se poate intra în bucla infinită. B. Adevarat. C. Fals, se returnează fals când $prim \neq ultim$ și când găsim două caractere diferite, nu trebuie să se returneze fals. D. Fals, se returnează adevărat când $prim \neq ultim$ și nu ajunge doar ca două caractere să fie diferite, ci trebuie ca toate.

588. A. Fals, contraexemplu: $n = 4$ și $a = [1,1,1,2,2,2,2,1]$ va rezulta $[1,1,1,2,1,2,2,2]$. B. Fals, ultima linie din interschimbare este incorectă, nu se interschimbă corect elementele. C. Adevarat. D. Fals, având "Dacă" nu va fi niciodată adevărat, nu se poate ca $a[j]$ să fie și par și impar în același timp.

589. Vom lua exemplul $n = 10$ și $k = 3$, iar indicii generați trebuie să fie 1, 4, 7. A. Pentru $j = 1$ se generează 2, pentru $j = 2$ se generează 5 și pentru $j = 3$ se generează 8. B. Pentru $j = 1$ se generează 1, pentru $j = 2$ se generează 4 și pentru $j = 3$ se generează 7. C. Pentru $j = 1$ se generează 0, pentru $j = 2$ se generează 3 și pentru $j = 3$ se generează 6. D. Pentru $j = 1$ se generează 1, pentru $j = 2$ se generează 4 și pentru $j = 3$ se generează 7. De asemenea, formulele de la variantele B și D sunt echivalente: $\left(\frac{(j-1)\cdot n}{k}\right) + 1 = \left(\frac{(j-1)\cdot n}{k}\right) + \frac{k}{k} = \frac{(j-1)\cdot n + k}{k}$.

590. Varianta C. se elimină, deoarece reprezentarea binară a lui 3 este 11, iar suma

cifrelor binare este 2, nedivizibil cu 3. Demonstrăm că afirmațiile A, B, D sunt adevărate prin următoarea afirmație mai generală: "Fie N un număr natural a cărui reprezentare în baza B este $b_n b_{n-1} \dots b_0$. Numărul N este divizibil cu $B+1$ dacă și numai dacă expresia $E = b_0 - b_1 + b_2 - b_3 + \dots + (-1)^n \cdot b_n$ este divizibilă cu $B+1$." Pentru a demonstra teorema de mai sus, vom demonstra mai întâi următoarea lemmă: "Fie P un număr natural. Atunci P^{2k} este de forma $X \cdot (P+1) + 1$, iar P^{2k+1} este de forma $Y \cdot (P+1) - 1$." Demonstrația lemei se face prin inducție. Mai întâi să observăm că $P^0 = 1 = 0 \cdot (P+1) + 1$, iar $P^1 = P = 1 \cdot (P+1) - 1$. Dacă $P^{2k} = X \cdot (P+1) + 1$, atunci $P^{2k+1} = P \cdot P^{2k} = P \cdot (X \cdot (P+1) + 1) = P \cdot X \cdot (P+1) + P = Y \cdot (P+1) - 1$. Similar, dacă $P^{2k+1} = Y \cdot (P+1) - 1$, atunci $P^{2k+2} = P \cdot P^{2k+1} = P \cdot (Y \cdot (P+1) - 1) = P \cdot Y \cdot (P+1) - P = X \cdot (P+1) + 1$. Demonstrația teoremei: Dacă $b_n b_{n-1} \dots b_0$ este reprezentarea în baza B a lui N, atunci $N = b_0 + b_1 \cdot B + b_2 \cdot B^2 + \dots + b_n \cdot B^n$. Deoarece puterile lui B sunt de forma precizată în lemmă, obținem: $N = b_0 \cdot ((B+1) \cdot f_0 + 1) + b_1 \cdot ((B+1) \cdot f_1 - 1) + b_2 \cdot ((B+1) \cdot f_2 + 1) + b_3 \cdot ((B+1) \cdot f_3 - 1) + \dots + b_n \cdot ((B+1) \cdot f_n + (-1)^n)$. Așadar, $N = (B+1) \cdot b_0 \cdot f_0 + b_0 + (B+1) \cdot b_1 \cdot f_1 - b_1 + (B+1) \cdot b_2 \cdot f_2 + b_2 + (B+1) \cdot b_3 \cdot f_3 - b_3 + \dots + (B+1) \cdot b_n \cdot f_n + (-1)^n \cdot b_n$. Putem rescrie această expresie ca: $N = (B+1) \cdot (b_0 \cdot f_0 + b_1 \cdot f_1 + b_2 \cdot f_2 + \dots + b_n \cdot f_n) + b_0 - b_1 + b_2 - b_3 + \dots + (-1)^n \cdot b_n$. Deoarece $(B+1) \cdot M$ este divizibil cu $B+1$, deducem că N este divizibil cu $B+1$ dacă și numai dacă $b_0 - b_1 + b_2 - b_3 + \dots + (-1)^n \cdot b_n$ este divizibil cu $B+1$. Mai sus este un caz particular al teoremei, în care $B = 2$. Expresia E poate fi scrisă și $E = (b_0 + b_2 + \dots) - (b_1 + b_3 + \dots)$ - diferența dintre suma cifrelor de rang par și suma cifrelor de rang impar. Dacă $B = 10$, proprietatea de mai sus este "Criteriul de divizibilitate cu 11".

591. Pentru $n = 12133121$, prefixul este 12 și are lungimea 2. La A. se returnează 2, la B. se returnează 1, la C. se returnează 3, la D. se returnează 2.

Pentru $n = 34534536$, prefixul este 3453 și are lungimea 4. La A. se returnează 4, la B. se returnează 1, la C. se returnează 5, la D. se returnează 4.

Pentru $n = 1223$, un astfel de prefix nu există (considerăm că are lungime 0).

La A. se returnează 0, la B. se returnează 1, la C. se returnează 0, la D. se returnează 0.

592. Pentru a calcula $X(i)$, se vor vizita celulele din celula i, apoi celulele care apar

în cele noi vizitate și tot așa până când ajungem la celule doar cu constante. La final, numărăm câte celule distincte am vizitat.

Pentru a calcula $Y(i)$, se vor vizita celulele unde apare celula i, apoi celulele unde apar cele noi vizitate și tot așa, până când nici o celulă nouă nu mai apare. La final, numărăm câte celule distincte am vizitat, adăugând și celula inițială i, iar din numărul total de celule scădem cele calculate.

Exemple:

$X(A2) \to B1, B2, B3, D3, D2 \quad \Rightarrow \quad X(A2) = 5$

$Y(A2) \to A2, A4, D1 \quad \Rightarrow \quad Y(A2) = 16 - 3 = 13$

$X(C4) \to A3, B1, D3, B3, D2 \quad \Rightarrow \quad X(C4) = 5$

$Y(B2) \to B2, A2, A4, D1 \quad \Rightarrow \quad Y(B2) = 16 - 4 = 12$

593. Algoritmul `functie(a, b)` trebuie să returneze CMMDC(a, b). A. Fals, returnează 1 întotdeauna. B. Adevărat, Algoritmul lui Euclid prin împărțiri funcționează și dacă $a = 0$ (a poate fi 0, în cerință nu se pune restricție pentru nr). C. Fals, Algoritmul lui Euclid prin scăderi nu funcționează dacă $a = 0$, va fi ciclu infinit. D. Fals, dacă $a = 0$, nu funcționează.

594. Fie $nr_t(n)$ numărul de afișări ale mesajului "Aici" pentru apelul `fibonacci(n)`.

n	1	2	3	4	5	6	7	8	9	10	...
$fibonacci(n)$	1	2	3	5	8	13	21	34	55	89	...
$nr_t(n)$	0	1	2	4	7	12	20	33	54	88	...

Din acest tabel rezultă că: $nr_t(n) = nr_t(n-1) + nr_t(n-2) + 1$, pentru $n \geq 2$.De asemenea, avem: $nr_t(n) = fibonacci(n) - 1$.

595. Se dă factor comun la fiecare pas și obținem $E(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + x \cdot (a_3 + x \cdot x \cdot a_5)))$. Numărul minim de înmulțiri necesare este 5.

596. Algoritmul folosește un indice i pentru a parcurge vectorul, resetându-l la zero când ajunge la final, și interschimbă elementele adiacente folosind operația XOR când acestea sunt în ordine descrescătoare, resetând contorul len la dimensiunea vectorului la fiecare interschimbare, până când nu mai sunt necesare interschimbări.

597. $ABCD_{(16)} = A \cdot 16^3 + B \cdot 16^2 + C \cdot 16^1 + D \cdot 16^0 = 10 \cdot 4096 + 11 \cdot 256 + 12 \cdot 16 + 13 = 43981; 132_{(8)} = 1 \cdot 8^2 + 3 \cdot 8^1 + 2 \cdot 8^0 = 64 + 24 + 2 = 90; 24_{(6)} = 2 \cdot 6^1 + 4 \cdot 6^0 = 12 + 4 = 16; A28B_{(14)} = A \cdot 14^3 + 2 \cdot 14^2 + 8 \cdot 14^1 + B \cdot 14^0 = 10 \cdot 2744 + 2 \cdot 196 + 8 \cdot 14 + 11 = 27955$. Deci, $E = 43981 + 90 + 16 + 27955 = 72042$.

598. Traversarea în preordine parcurge rădăcina, apoi subarborele stâng, iar apoi subarborele drept. În acest arbore, ordinea este: 1, 2, 4, 6, 7, 5, 3, 8, 9, 10. Deci, răspunsul corect este B.

599. Algoritmul calculează suma tuturor elementelor dintr-un pătrat de dimensiune kk. Prima parte construiește sume parțiale pe matrice unde fiecare element $a[i][j]$ conține suma tuturor elementelor din dreptunghiul $(1, 1)$ până la (i, j). A doua parte folosește această matrice pentru a găsi suma maximă dintr-un pătrat kk și numărul de astfel de pătrate, folosind formula $sum = a[i][j] + a[i-k][j-k] - a[i][j-k] - a[i-k][j]$ care calculează suma elementelor din pătratul $k \times k$ care are colțul dreapta-jos la poziția (i, j).

600. Algoritmul implementează o căutare binară într-un vector sortat pentru a găsi poziția maximă până la care suma a două laturi date poate forma un triunghi valid (prin verificarea inegalității triunghiului $a[i] + a[j] > a[k]$), permițând astfel determinarea tuturor combinațiilor posibile de laturi pentru triunghiuri valide, iar pentru exemplul dat cu vectorul $[3, 4, 5, 6, 7]$ și parametrii $(2, 9)$, funcția returnează corect poziția 4 deoarece aceasta este ultima poziție unde suma 9 (care reprezintă suma a două laturi) respectă inegalitatea triunghiului. Afirmația C ar fi adevărată dacă ar fi fost formulată ca "nu există niciun triunghi valid care să includă laturi aflate după poziția j", dar în forma actuală este falsă deoarece pot exista triunghiuri valide folosind chiar laturile selectate.

601. Algoritmul calculează numărul de cifre de 0 care apar la finalul lui $n!$. Cum un produs de 2 și 5 formează un zero, algoritmul numără câte perechi de 2 și 5 există în descompunerea în factori primi a lui $n!$.

602. Algoritmul generează toate numerele de 3 cifre, care au cifra sutelor pară, iar cifrele sale sunt in ordine crescătoare (de la cifra sutelor, la cea a zecilor și cea a unităților). Această constrângere impune faptul că primul număr generat va fi 234, iar ultimul număr generat va fi 689. Primele numere afișate sunt 234, 235, 236, 237, 238, 239, 245, 246, 247,

șamd.

603. Ambii algoritmi simulează exponențierea rapidă, deci rezultatul va fi, în ambele cazuri, a^b. Algoritmul `ceFace2` este mai eficient din punct de vedere al memoriei, deoarece folosește operații pe biți.

604. Pentru variantele C și D expresia logică devine True deoarece în ambele cazuri, deși prima parte a expresiei este False (din cauza XOR între valori identice pentru A și B), a doua parte devine True (prin XOR între False obținut din **NOT** (True AND True) și True obținut din (C OR D)), iar în final **OR** dintre False și True dă True.

605. Algoritmul prezentat este specific Ciurului lui Eratostene, care marchează cu 1 multiplii termenului curent i în șirul v. Complexitatea totală a algoritmului este $O(log(logn))$.

606. Algoritmul găsește poziția de start din care mașina poate parcurge toate benzinăriile. El folosește o abordare greedy, verificând pentru fiecare poziție dacă mașina poate ajunge la următoarea benzinărie. Dacă la un moment dat suma de combustibil devine negativă ($s < 0$), înseamnă că nu putem începe de la nicio poziție anterioară poziției curente, așa că actualizăm poziția de start (d) la următoarea poziție. Complexitatea este $O(n)$ deoarece parcurgem array-ul o singură dată. Exemplu: $gas = [1, 2, 3, 4, 5]$ $cost = [3, 4, 5, 1, 2]$ Rezultat: 3 (începând de la poziția 3 putem parcurge toate benzinăriile)

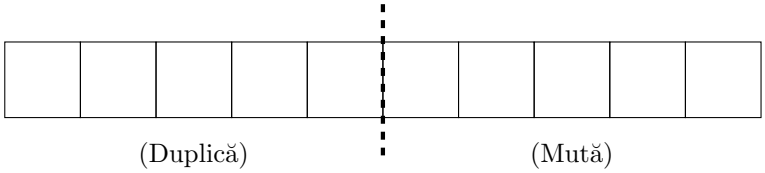
607. Algoritmul calculează recursiv cea mai mică cifră pară a numărului x și o returnează, iar în cazul numerelor care nu conțin cifre pare, algoritmul returnează -1. Deoarece în baza 10, un număr x are $\theta(\log_{10} x)$ cifre, complexitatea algoritmului este $O(\log x)$.

608. Funcția X calculează, de fapt, indicatorul lui Euler. Pentru $n = 12$ este 4, deoarece numerele relativ prime cu 12 sunt 1, 5, 7, 11. Pentru $n = 5$, care este prim, $X(5) = 4$, nu 5. Dacă p este prim, $X(p) = p - 1$ deoarece toate numerele mai mici decât p sunt relativ prime cu p. Proprietatea D este adevărată deoarece indicatorul lui Euler respectă principiul fundamental că pentru $\gcd(a, b) = 1$, dacă numărăm elementele din mulțimea $\{n : \gcd(n, ab) = 1\}$, aceasta este echivalentă cu produsul cardinalelor mulțimilor $\{n : \gcd(n, a) = 1\}$ și $\{n : \gcd(n, b) = 1\}$. Formal, dacă n este prim cu a și b, atunci n este prim și cu ab, păstrând astfel proprietatea multiplicativă $X(ab) = X(a)X(b)$.

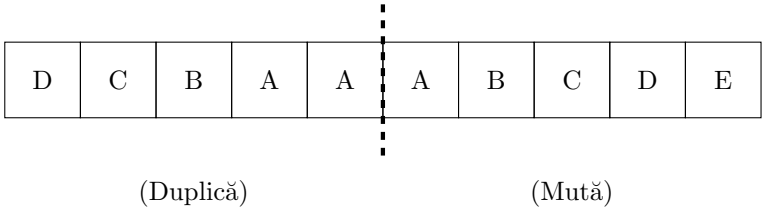
609. Algoritmul simulează o sortare prin numărare a elementelor din v. f este un vector

de frecvență, iar elementele sunt plasate în final la poziția corespunzătoare în varianta sortată a șirului, care se va reține în șirul o. Algoritmul mai este cunoscut drept "Counting Sort" și va avea complexitatea timp $O(n+m)$ în toate cele 3 cazuri.

610. Folosind o linie imaginară pentru prima literă inserată in l, putem delimita vectorul m în două părți: cel rezultat din operația **Duplică** (cel din stanga liniei) și cel rezultat din operația **Mută** (cel din dreapta liniei):



Astfel, încă din procesarea primelor elemente, observăm că elementele din partea dreapta sunt în ordine strict crescătoare, și mai exact, apar în aceeași configurație ca în șirul l. Elementele din partea stânga vor fi mereu descrescătoare, indiferent de câte ori a fost folosită operația **Duplică**. Șirul final va avea un aspect de "vale". De aceea, doar răspunsul $\boxed{\text{B}}$ este fals. (Figura de mai jos reprezintă răspunsul $\boxed{\text{A}}$).



611. Algoritmul calculează **determinantul** unei matrice folosind metoda Gauss. $\boxed{\text{A}}$ este corectă, deoarece matricea B este de fapt, transpusa lui A, deci au același determinant. $\boxed{\text{B}}$ este falsă, deoarece înmulțirea este multiplicativă, **doar dacă matricile au aceeași dimensiune**. $\boxed{\text{C}}$ este adevărată, determinantul unei matrici cu o coloana nulă va fi mereu 0. $\boxed{\text{D}}$ este falsă, deoarece formula ariei cu ajutorul determinantului are și un termen de $\frac{1}{2}$ înmulțit cu determinantul matricei.

612. Algoritmul A implementează "Stooge Sort", un algoritm care sortează un șir astfel: verifică dacă valoarea de la începutul șirului este mai mare decât cea de la finalul șirului, apoi, dacă sunt mai mult de 3 elemente în listă sortează, pe rând, prima 2/3 parte din șir, apoi ultima 2/3 parte din șir, iar apoi prima 2/3 parte din nou. Rezultatul este $[123, 456, 763, 998]$

Algoritmul B sortează șirul inițial crescător, după cifra zecilor. Configurația dată va rezulta astfel într-un șir sortat crescător. Rezultatul este $[123, 456, 763, 998]$

Algoritmul C încearcă să sorteze șirul, atât după cifra zecilor, cât și după rezultatul operației modulo 7. ($998 \mod 7 = 4$, $998 \div 100 = 9$, $763 \mod 7 = 0, 763/10 = 6$, etc.). Rezultatul este $[763, 123, 456, 998]$, deci este singurul răspuns diferit.

Algoritmul D implementează "Kirkpatrick–Reisch sort" care este un algoritm de sortare care combină selecția directă cu inserția, funcționând în două faze: mai întâi împarte șirul în două subsecvențe (una ordonată și alta neordonată) folosind selecția directă pentru a găsi elementul minim și a-l plasa la început, iar apoi folosește inserția pentru a muta eficient elementele rămase la pozițiile lor corecte în secvența ordonată, rezultând astfel un șir complet sortat.

613. Când $a+b = c$ și $a \neq b$, funcția calculează C_{c-1}^{a-1} prin adăugarea recursivă a căilor din triunghiul lui Pascal (varianta D). Varianta A este incorectă (partițiile nu sunt corelate), iar B e falsă deoarece $a > b$ nu forțează 0 decât dacă $a = c$ sau $b = c$.

614. Avem 2 cazuri pentru calculul complexității:

$$T(n) = \begin{cases} 1 & \text{n} \leq 1 \\ 2 + 2T(n/2) & n > 1 \end{cases}$$

Calculăm complexitatea apelului recursiv: $T(n) = 2(1+T(n/2)) = 2(1+2(1+T(n/4))) = \cdots = 2(1+2(1+\ldots 2(1+T(X))))$, unde $X \leq 1$. La fiecare apel recursiv, valoarea lui n se înjumătățește. La fiecare execuție, numărul de apeluri se dublează. Deci complexitatea algoritmului va fi: $T(n) = 2^{\log_2(n)} = n \in \Theta(n)$.

615. Pentru n pătrat impar și $m = 1$, funcția returnează 1 doar dacă $k = \sqrt{n}$ (varianta A), deoarece se scade k^2 din n, lăsând 0. Varianta D este corectă: apelul cu $k = 1$ și $m = 1$ verifică dacă n este pătrat impar (căutând $k = \sqrt{n}$). Varianta B e falsă ($n = 25$, $k = 3$ continuă până la $k = 5$).

616. Varianta **B** este adevărată: pentru $n = 2^m - 1$, jucătorul B poate replica mutările lui A pentru a-l forța să elimine ultima piatră.

Varianta **D** este adevărată: Numărul de biți de 1 în binar indică câte puteri ale lui 2 sunt

necesare pentru a descompune n, determinând mutări optime.

Varianta A este falsă: Dacă $n = 2^k$, jucătorul A poate câștiga imediat eliminând toate pietrele. Varianta C e falsă: $n = 11$ $(4*2+3)$ poate fi câștigat de A dacă elimină 4 pietre, lăsând 7 (poziție pierzătoare pentru B).

617. Algoritmul simulează o căutare binară pe șirul a și reține cel mai mare element mai mic sau egal cu a. Condiția de la linia 6 trebuie să aloce corect indicele mijlocului dintre capătul stâng (x) și capătul drept (y). Varianta A este, evident, corectă, iar variantele B și C sunt, de asemenea, corecte, deoarece calculează mijlocul șirului evitând scenariului de overflow.

618. Subalgoritmul implementează recurența $f(n) = f(n-1) + f(n-2)$ cu condițiile inițiale $f(0) = 1$, $f(1) = 2$. Aceasta corespunde numărului de șiruri binare de lungime n fără doi de 1 consecutive (opțiunea B). Opțiunea A implică o recurență diferită (pentru zerouri consecutive), C este incorectă ($F_0 = 0 \neq 1$), iar D nu are legătură cu recurența.

619. Varianta **C** este adevărată: Dacă toate elementele sunt pare, suma oricărui subșir nevid va fi întotdeauna pară (suma de numere pare rămâne pară).

Varianta **D** este adevărată: Conform principiului lui Dirichlet, există $2^n - 1$ subșiruri nevide și cel mult $\frac{n(n+1)}{2}$ sume distincte posibile. Pentru $n \geq 2$, $2^n - 1 > \frac{n(n+1)}{2}$, deci cel puțin două subșiruri au aceeași sumă.

Varianta A este falsă: Pentru $S = [1, 1, \ldots, 1]$ cu $n = 3$, nicio sumă de subșir nevid (1, 2 sau 3) nu este divizibilă cu 3.

Varianta B este falsă: Dacă $S = [1, 2]$, există 3 subșiruri cu sumă impară ([1], [1,2], [2] are sumă pară), dar $2^{2-1} = 2 \neq 3$.

620. Algoritmul realizează primele n iterații din metoda de sortare Bubble Sort crescător. Dacă $n = m$, vectorul va fi complet sortat crescător. Dacă $n < m$, doar ultimele $n + 1$ elemente vor fi sortate crescător, iar restul vectorului poate rămâne nesortat. În prima iterație, cel mai mare element ajunge pe ultima poziție. În a doua iterație, al doilea cel mai mare element ajunge pe penultima poziție. După n iterații, ultimele $n + 1$ elemente vor fi în ordine crescătoare, dar primele elemente pot rămâne nesortate.

621. Algoritmul calculează suma tuturor numerelor din intervalul $[x, y]$, dacă $x \leq y$, sau

din intervalul $[y, x]$, dacă $y < x$, care sunt divizibile cu 3 și nu sunt divizibile cu 2.

622. $E = 268 + 76 + 118 - 5 * 30 = 268 + 76 + 118 - 150 = 312$, $312_{(10)} = 624_{(7)}$, $312_{(10)} = 138(16)$

623. Algoritmul `call(arr, n)` returnează True atunci când vectorul `arr` începe cu o valoare mai mică decât următoarea valoare și, ulterior, apelând algoritmul `f`, acesta găsește un vârf în secvență, unde elementul curent este mai mare decât vecinii săi. C. Fals, dacă vectorul `arr` are toate elementele ordonate strict crescător, atunci nu se găsește un element mai mare decât ambii vecini ai săi. D. Fals, nu e neapărat să se respecte acel model în totalitate, trebuie să existe doar un singur element mai mare decât ambii vecini ai săi.

624. Algoritmul verifică dacă media aritmetică a cifrelor lui `n` este mai mare sau egală decât prima cifră a lui `n`. A. Fals, trebuie să fie "mai mare sau egală". B. Adevărat. C. Adevărat. D. Adevărat.

625. Algoritmul calculează și returnează cel mai mare divizor comun al numerelor `a` și `b`, folosind Algoritmul lui Euclid. Dacă `a` și `b` sunt prime între ele, atunci cel mai mare divizor comun este 1.

626. Parcurgerile sunt: Postordine(Stânga, Dreapta, Rădăcina): 4 7 9 8 5 2 6 3 1, Preordine(Rădăcina, Stânga, Dreapta): 1 2 4 5 7 8 9 3 6, Inordine(Stânga, Rădăcina, Dreapta): 4 2 7 5 8 9 1 3 6. Arborele binar nu este strict, deoarece nodurile 3 și 8 au un singur fiu.

627. Algoritmul `ceFace(n, p)` returnează un număr format din cifrele lui n, astfel încât după fiecare cifră pară se inserează jumătate din aceasta, iar cifrele impare se păstrează la fel. De exemplu, dacă $n = 122$, algoritmul va returna 1213425, deoarece pentru fiecare cifră pară, se adaugă jumătatea acesteia, iar cifrele impare sunt păstrate neschimbate.

628. A. Fals, deoarece pentru $x = False$, $y = False$, $z = False$ se vor afișa $True\ True$. B. Fals, există doar 2 perechi de valori $(False, True, False)$ și $(False, True, True)$ pentru care algoritmul va afișa `True`. C. Fals, algoritmul poate să nu afișeze niciun rezultat, în funcție de condițiile inițiale. D. Adevărat, pentru apelurile `sim(`$False, True, False$`)` și `sim(`$False, True, True$`)` algoritmul va afișa aceeași valoare.

629. Parcurge simultan cele două șiruri și compară elementele: dacă un element din a este mai mic decât cel curent din b, trece la următorul element din a; dacă sunt egale,

trece la următorul element din b; altfel, elementul din b nu există în a, așa că îl adaugă în c. La final, c conține toate elementele din b care nu apar în a.

630. Algoritmul `pg` verifică dacă un șir *arr* de lungime n formează o progresie aritmetică. Dacă n are mai puțin de 3 elemente, returnează `True`, deoarece oricare două numere formează o progresie aritmetică. Apoi, calculează diferența comună d dintre primele două elemente și parcurge restul șirului, verificând dacă fiecare diferență succesivă este egală cu d. Dacă găsește o abatere, returnează `False`; altfel, returnează `True`, confirmând că șirul este o progresie aritmetică. A. Fals, șirul *arr* formează o progresie geometrică, nu aritmetică. B. Fals, $a[2] - a[1] = 19$, iar $a[8] - a[7] = 18$. C. Adevărat. D. Fals.

631. Algoritmul calculează numărul de pătrate perfecte de numere prime mai mici decât n, folosindu-se de un algoritm în stilul Ciurului lui Eratostene. La fiecare pas, în cazul în care găsește un număr prim, toți multiplii, începând de la pătratul numărului, vor fi marcați ca fiind numere compuse (nu e nevoie să înceapă marcarea de la un număr mai mic, deoarece toți multiplii mai mic decât pătratul perfect vor fi deja marcați ca nefiind numere prime.

632. Primele 20 de cifre afișate sunt: 05105620573051056805, iar cifrele de pe pozițiile 16-20 sunt 56805.

633. Algoritmul actualizează elementul de la poziția idx + 1 din vector cu suma elementului curent arr[idx], elementului următor arr[idx + 1] și elementului următor următor arr[idx + 2] împărțit (DIV) la 3. Algoritmul continuă recursiv până când idx = $n - 1$, când returnează valoarea elementului de la poziția k din vector.

634. A. Algoritmul `divisors1` nu tratează cazul în care n este un pătrat perfect, astfel că rezultatul nu va fi corect pentru aceste cazuri. B. Algoritmul `divisors2` folosește descompunerea în factori primi și calculează corect numărul de divizori. C. Algoritmul `divisors3` folosește o abordare recursivă pentru a număra divizorii și tratează corect și cazul în care n este pătrat perfect. D. Algoritmul `divisors4` folosește o abordare similară cu `divisors2` și calculează corect numărul de divizori folosind descompunerea în factori primi. Algoritmul ia în considerare și cazul în care n rămâne mai mare de 1 la sfârșit și este corect în aceste condiții.

635. Algoritmul construiește matricea `A`, astfel încât fiecare element este calculat după formula $A[i][j] = k^{(j-1)} \cdot C_{j-1}^{i-1}$, unde $C_a^b = \frac{a!}{b!(a-b)!}$ reprezintă combinări de a luate câte b. Algoritmul returnează elementul $A[n-3][n-2]$, care este $3^6 \cdot C_6^5 = 3^6 \cdot 6 = 729 \cdot 6 = 4374$.

636. Algoritmul caută secvența de lungime maximă cu termeni din vector care sunt multipli de `k`, apoi numără câte secvențe de lungime maximă sunt în vector, dar nu ia în calcul ultima secvență, dacă vectorul se termină cu o secvență cu proprietatea respectivă.

637. Algoritmul calculează și returnează a^b. Complexitatea timp a algoritmului este: $T(b) = O(1)$, dacă $b = 0$, $T(b) = T(b/2) + O(1)$, dacă b este par, $T(b) = 2 \cdot T(b/2) + O(1)$, dacă b este impar. Se va lua complexitatea în cel mai rău caz, adică atunci când b este impar la fiecare apel, astfel încât $T(b) = 2 \cdot T(b/2) + O(1) = 2 \cdot (2 \cdot T(b/4) + O(1)) + O(1) = O(2^{\log_2 b}) = O(b)$. A. Fals, $2^{14} = 16384$, dar algoritmul se autoapelează de 15 ori, nu de 14. B. Fals, complexitatea algoritmului este $O(b)$. C. Adevărat. D. Adevărat, prin simulare.

638. Algoritmul numără câte numere din intervalul de indici $[a, b]$ din vectorul `arr` au toate cifrele diferite de c și verifică ca numărul să fie diferit de poziția pe care se află în vector. Algoritmul folosește tehnica divide et impera.

639. Algoritmul presupune căutarea celui mai mare element din vector, care este mai mic sau egal cu val, folosind căutarea binară. A. Adevărat. B. Fals. C. Adevărat. D. Fals.

640. Algoritmul `algo(q, w, arr, z)`, pentru apelul cu $w = 1$, va afişa toate combinările de q luate câte z cu elemente de la 1 până la n. For-ul "For $i \leftarrow arr[w-1]+1, q-z+w$ execute" asigură posibilităţile de unde vor putea lua valori elementele din soluţie. "arr[w-1]+1" reprezintă limita inferioară şi asigură că elementele să fie în ordine crescătoare, iar "q-z+w" reprezintă limita superioară, reprezentând diferenţa dintre valoarea maximă şi numărul de poziţii care trebuie completate de la poziţia curentă w, până la final. A. Fals, $C_8^3 = 56$, se vor afişa 56 de caractere de spaţiu. B. Adevărat, numărul soluţiilor pentru C_4^2 sunt 6, iar soluţiile sunt 12, 13, 14, 23, 24, 34. C. Fals, afişează toate combinările nu aranjamentele. D. Adevărat, $C_n^k = C_n^{n-k}$ întotdeauna, datorită proprietăţii de simetrie a coeficienţilor binomiali.

641. A. Adevărat. B. Adevărat, algoritmul de descompunere în factori primi a unui număr poate avea complexitate $O(n)$ sau $O(\sqrt{n})$. C. Adevărat, metoda numărării presupune numărarea frecvenței fiecărui element din vector, apoi plasarea fiecărui element în poziția corectă în funcție de frecvența sa. D. Fals, căutarea binară poate fi implementată doar în complexitate $O(\log n)$.

642. Algoritmul calculează numărul total de moduri în care îl putem scrie pe k ca sumă de elemente înmulțite cu 2 sau cu 3, în funcție de paritatea lor, din vector, astfel încât elementele să fie situate la cel puțin două poziții distanță unul de celălalt. De exemplu, dacă includem elementul de pe poziția 2, putem adăuga elemente începând cu poziția 5 pentru suma respectivă. Vom rescrie vectorul cu toate elementele pare înmulțite cu 2 și toate elementele impare înmulțite cu 3: $[8, 15, 11, 3, 3, 7, 6, 10, 12] \Rightarrow [16, 45, 33, 9, 9, 21, 12, 20, 24]$. A. Pentru $k = 45$, se găsesc următoarele modalități: 45, $24 + 21$, $12 + 33$, $16 + 9 + 20$, $16 + 9 + 20$, adică se va returna 5. B. Pentru $k = 33$, se găsesc următoarele modalități: 33, $24 + 9$, $24 + 9$, adică se va returna 3. D. Pentru $k = 29$, se găsesc următoarele modalități: $20 + 9$, $20 + 9$, adică se va returna 2.

643. Algoritmul calculează suma tuturor valorilor de 1 din reprezentarea în baza 2. Pentru $n = 8 = 2^3$:

Număr	Reprezentare binară
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Grupăm reprezentările după formatul: primul cu ultimul, al doilea cu penultimul, etc.

0 și $7 = 000, 111 \quad \Rightarrow \quad$ 3 biți de 1

1 și $6 = 001, 110 \quad \Rightarrow \quad$ 3 biți de 1

2 și $5 = 010,101 \quad \Rightarrow \quad$ 3 biți de 1

3 și $4 = 011,100 \quad \Rightarrow \quad$ 3 biți de 1

Observăm că se obțin 4 grupe a câte 3 biți de 1, adică:

$4 \times 3 = 12$ biți de 1pentru toate numerele naturale până la 7. Mai trebuie să adăugăm numărul de biți din reprezentarea lui 8, adică:

$8 = 1000_{(2)} \quad \Rightarrow \quad 12+1 = 13$ biți de 1 $\quad \Rightarrow \quad$ pentru $n = 8 \quad \Rightarrow \quad$ 13. Pe caz general, trebuie să găsim cea mai apropiată putere de 2 de numărul nostru și să aplicăm regula. Pentru $n = 2^k \Rightarrow \frac{n}{2} \times k + 1$. Pentru $n = 2046$, cea mai apropiată putere de 2 este $2048 = 2^{11} \Rightarrow 1024 \times 11 + 1 = 11265$. Atenție, aceasta este suma pentru toate numerele de la 1 până la 2048. Trebuie să scădem numărul de biți 1 din reprezentările 2047 și 2048, adică: $2048 = 100000000000_{(2)}$ și $2047 = 11111111111_{(2)} \quad \Rightarrow \quad 11265 - 1 - 11 = 11253$.

644. Algoritmul calculează diferența absolută dintre suma elementelor fiecărei linii și suma elementelor acelei linii în ordine inversă(adică în orice caz valoarea obținuta va fi 0). Complexitatea este determinată de cele două bucle imbricate, ceea ce duce la o complexitate de $O(n^2)$.

645. Pentru a calcula C_n^k, varianta A implementează recurența obișnuită $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$. Varianta C se folosește de faptul că se poate utiliza simetria astfel încât $C_{n-1}^k = C_{n-1}^{n-1-k}$, ceea ce este corect. Varianta B calculează C_n^{n-k} ceea ce este echivalent cu C_n^k. Varianta D va rezulta într-o recurență infinită.

646. Algoritmul utilizează tehnica ferestrei glisante (sliding window) pentru a determina suma maximă a unei subsecvențe de lungime fixă k din vectorul `v`. Complexitatea sa este $O(n)$, deoarece fiecare element al vectorului este adăugat și scăzut o singură dată din suma curentă. Algoritmul poate fi optimizat folosind sume parțiale, care ar permite calculul sumei oricărei subsecvențe în O(1) după o preprocesare în O(n). Numărul de actualizări ale variabilei s este direct proporțional cu n, deoarece pentru fiecare element facem maxim două operații (o adunare și eventual o scădere), rezultând aproximativ $2 \cdot n$ operații în total.

647. Algoritmul implementat calculează coeficientul binomial folosind relația combinatorială $C(n,k) = \frac{n!}{k!(n-k)!}$. Algoritmul ajustează k la $100 - 97 = 3$ deoarece $97 > 50$. Bucla

rulează 3 iterații, fiecare implicând o singură înmulțire.

648. Algoritmul implementează o variantă de căutare binară pentru a găsi prima apariție a valorii `t`. Dacă valoarea este găsită, se continuă căutarea în jumătatea stângă pentru a verifica dacă există o apariție anterioară. Algoritmul reduce în mod exponențial, având complexitatea $O(\log n)$.

Pentru afirmațiile A, C și D nu există inconsistențe: formula de calcul a lui m este validă și preferabilă pentru evitarea overflow-ului, algoritmul nu intră în buclă infinită pentru $n = 2^{31} - 1$ deoarece formula previne overflow-ul, iar deoarece a începe de la 1 și crește iar b începe de la n și scade, nu există posibilitatea accesării elementelor în afara vectorului. Afirmația B este falsă deoarece algoritmul nu returnează -1 când găsește valoarea t, ci returnează poziția corectă prin variabila r care primește valoarea lui m când $v[m] = t$

649. Algoritmul analizează fiecare element al vectorului și calculează valoarea minimă pentru toate subsecventele începând de la poziția curentă. Complexitatea algoritmului este $O(n^2)$, deoarece pentru fiecare element se parcurg toate elementele ulterioare. Algoritmul poate fi optimizat utilizând o parcurgere inversă și păstrând minimul curent, reducând astfel timpul de execuție la $O(n)$.

650. Funcția `ceFace()` parcurge întregul vector de numere pentru a determina prima și ultima apariție a elementului `x`, având complexitatea $O(n)$. Pentru vectorii de numere $[1, 2, 42, 42, 1, 42, 2]$ și $[7, 1, 7, 7, 7, 2, 7$ se returnează 4, respectiv 7.

651. Funcția `X()` transformă vectorul de numere astfel încât fiecare element devine produsul tuturor elementelor precedente, inclusiv el însuși. Algoritmul are complexitate $O(n)$, deoarece parcurge vectorul de numere o singură dată, efectuând multiplicări succesive. De exemplu pentru vectorul $[2, 3, 4, 5]$, va rezulta $[2, 6, 24, 120]$.

652. Funcția `f()` parcurge vectorul și înlocuiește fiecare element cu m-ul precedent(adică valoarea minima), returnând m-ul final. Complexitatea algoritmului este $O(n)$, deoarece se parcurge vectorul o singură dată. Ultimul element este modificat pentru a reflecta m-ul final(adică cel mai mic element).

653. Algoritmul numără pentru fiecare element din vectorul `v` câte numere mai mici decât el există în vector. Pentru $v =$ `[4, 1, 5, 2, 3, 7, 6]`, rezultatul `r = [3, 0, 4, 1,`

`2, 6, 5]` se obține deoarece 4 are trei numere mai mici (1, 2, 3), 1 are zero numere mai mici, 5 are patru numere mai mici (4, 1, 2, 3), 2 are un număr mai mic (1), 3 are două numere mai mici (1, 2), 7 are șase numere mai mici (toate în afară de el), iar 6 are cinci numere mai mici (4, 1, 5, 2, 3). Dintre variantele date, doar A este corectă: `v[j] < v[i]` cu incrementare `c ← c + 1`.

654. Funcția `ceFace()` calculează lungimea celei mai lungi subsecvențe comune (LCS) între două vectori. Complexitatea timp este $O(n \cdot m)$ și folosește o matrice pentru a stoca soluțiile subproblemelor. De exemplu, pentru `"abcde"` și `"ace"`, lungimea LCS este 3. Numărul de inserări necesare pentru a transforma s în t este $m - \text{LCS}$, unde m este lungimea lui t. Deoarece algoritmul calculează lungimea LCS, această valoare poate fi derivată folosind rezultatul.

655. Algoritmul parcurge vector sortat și returnează cea mai mare poziție pentru care $v[m] \leq target$. Dacă `target` nu se află în vector, returnează -1. Complexitatea sa este $O(\log n)$. Afirmația C este falsă, pentru că `target` este un număr natural.

656. În cazul cel mai defavorabil, fiecare element din Q poate fi mutat în S și înapoi, generând un număr de operații proporțional cu pătratul numărului de elemente. Astfel, se execută de 256 ori.

657. Funcția `ceFace()` construiește vectorul de sume parțiale, unde fiecare element reprezintă suma elementelor de la început până la poziția curentă. Complexitatea este $O(n)$, deoarece fiecare element este prelucrat o singură dată. După construirea vectorului, suma unui interval $[l, r]$ poate fi calculată în timp constant folosind formula $p[r] - p[l-1]$. Funcția calculează suma totală a vectorului, și nu returnează toate prefixele.

658. Dacă nava depășește destinația pe oricare dintre axe, nu mai poate ajunge la punctul dorit, ceea ce justifică condiția $sx > dx$ **OR** $sy > dy$. Pentru a verifica dacă există un drum valid, trebuie să apelăm recursiv funcția pentru ambele direcții posibile, utilizând operatorul **OR**, deoarece este suficient ca una dintre căi să ducă la destinație.

659. Pentru apelul `F(35)`, funcția `F` apelează `G(33)`. Acesta, fiind mai mare ca 10, apelează `F(16)`, iar procesul continuă recursiv până la baza de recursie. Evaluând pas cu pas apelurile, se obține rezultatul final 42.

660. Algoritmul `f()` folosește tehnica Divide et Impera pentru a determina valoarea maximă din vector, împărțindu-l în două jumătăți și comparând maximele subintervalelor. Algoritmul nu necesită ca vectorul să fie sortat inițial, iar recursivitatea continuă până la cele mai mici subintervale.

661. Algoritmul `f()` parcurge recursiv vectorul v și înlocuiește fiecare valoare de pe pozițiile în cazul lui F(1) impare. Afișând când se coboară în recursivitate doar numerele impare, iar la urcarea din recursivitate toate pozițiile peste care a trecut. Afirmația B este falsă deoarece vectorul rezultat, v, este $[0, 2, 0, 4, 0, 6, 0, 8, 9, 10]$.

662. Algoritmul `Maxim(v,n)` parcurge vectorul dat în funcție de valorile elementelor, iar noi trebuie să stabilim condiția de oprire, care după o parcurgere simplă, putem observa că este $j \neq i$. Algoritmul `EstePrim(x)` este dat doar pentru a îngreuna parcurgerea, dacă citim atent cerință nu se face referință la faptul că valoarea ar trebui să fie prima.

663. La fiecare pas, dacă găsește un divizor d, adaugă numărul și se scade perechea sa n/d, evitând parcurgerea întregului interval de la 1 la n. Dacă $d \times d = n$, divizorul este adăugat o singură dată. Complexitatea algoritmului este $O(\sqrt{n})$.

- Dacă n este prim, atunci singurii divizori sunt 1 și n. Pentru $d = 1$, algoritmul adaugă 1 și scade n, rezultând $1 - n$ (care este negativ pentru $n \geq 3$).
- Dacă n este compus, pentru fiecare $d < \sqrt{n}$ avem contribuții de forma $d - \frac{n}{d}$ cu $d < \frac{n}{d}$ (adică termen negativ), iar pentru $d = \sqrt{n}$ (cazul de pătrat perfect) se adaugă doar $\sqrt{n}$. Suma totală rămâne negativă sau cel mult zero.

664. Funcția `functieRecursiva()` realizează două apeluri recursive la fiecare nivel, ceea ce determină o creștere exponențială a numărului de apeluri. Rezultatul poate fi optimizat prin memorarea rezultatelor intermediare. Complexitatea este $O(2^n)$.

665. Algoritmul verifică toate perechile posibile de elemente din vector și aplică expresia logică conform principiului absorbției pentru operatorii `AND` și `OR`. Complexitatea algoritmului este $O(n^2)$ datorită celor două bucle imbricate care parcurg fiecare pereche posibilă. Numărul de comparații care s-ar număra dacă algoritmul ar suferi schimbările de la varianta D este numărul de comparații pentru sortarea prin selecție.

666. Prima buclă: pentru fiecare i de la 1 la n, bucla interioară parcurge variabila j de la $i+1$ la m. Numărul de iterații pentru un anumit i este: $m-i$ (deoarece j variază de la $i+1$ la m). Deci, numărul total de iterații este: $\sum_{i=1}^{n}(m-i) = n \cdot m - \frac{n(n+1)}{2}$. Această sumă este dominantă și, în notația Big-O, se poate scrie $O(n \cdot m)$ (deoarece termenul $\frac{n(n+1)}{2}$ este de ordin inferior atunci când m este semnificativ mai mare decât n; iar chiar și dacă m este puțin mai mare decât n, forma expresiei ca funcție de ambele variabile este $O(n \cdot m)$). A doua buclă: Parcurge i de la n la m, adică efectuează $m-n+1$ iterații, deci contribuie cu $O(m-n)$. În total, complexitatea este: $O(n \cdot m) + O(m-n)$. Observăm că $O(n \cdot m)$ este termenul dominant, deci complexitatea totală se poate scrie compact ca: $O(n \cdot m)$.

667. Rezultatul final nu depinde de ordinea în care se aleg perechile deoarece operația $(x + y) \mod 10$ conservă suma numerelor modulo 10, iar pentru $n = 93$ suma $1 + 2 + \cdots + 93 = 4371$ are ultima cifră 1, deci rezultatul final este 1.

668. Algoritmul `Tree(node)` parcurge recursiv arborele binar și calculează o sumă specifică pentru fiecare nod. Pentru fiecare nod, se adaugă valoarea sa proprie și, dacă are atât copil stânga, cât și copil dreapta, se adaugă și cea mai mare valoare din subarborele său stâng și cea mai mică valoare din subarborele său drept.

669. Pentru a calcula valoarea expresiei E, trebuie convertit fiecare termen în baza 10 și efectuate operațiile indicate mai jos.

Numărul $AB_{(16)}$ în baza 10 se determină astfel:

$$AB_{(16)} = A * 16^1 + B * 16^0 = 10 * 16 + 11 * 1 = 160 + 11 = 171$$

Numărul $120_{(3)}$ în baza 10 este:

$$120_{(3)} = 1 * 3^2 + 2 * 3^1 + 0 * 3^0 = 1 * 9 + 2 * 3 + 0 = 9 + 6 = 15$$

Numărul $120_{(4)}$ în baza 10 este:

$$120_{(4)} = 1 * 4^2 + 2 * 4^1 + 0 * 4^0 = 1 * 16 + 2 * 4 + 0 = 16 + 8 = 24$$

Numărul $44_{(5)}$ în baza 10 este:

$$44_{(5)} = 4 * 5^1 + 4 * 5^0 = 4 * 5 + 4 * 1 = 20 + 4 = 24$$

Acum se calculează expresia:

$$E = 171 + 15 - 24 + 2 * 24$$

$$E = 171 + 15 - 24 + 48 = 210$$

Se verifică răspunsurile date. Se convertește $322_{(8)}$ în baza 10:

$$322_{(8)} = 3 * 8^2 + 2 * 8^1 + 2 * 8^0 = 3 * 64 + 2 * 8 + 2 * 1 = 192 + 16 + 2 = 210$$

Astfel, valoarea expresiei este $322_{(8)}$, ceea ce corespunde variantei A.

670. Algoritmul `sortare(v, n)` sortează elementele vectorului în funcție de ultima lor cifră. Parcurge toate perechile de elemente și compară ultimele cifre ale acestora. Dacă ultima cifră a unui element este mai mare decât ultima cifră a altuia, cele două valori sunt interschimbate. Dacă ultimele cifre sunt egale, algoritmul compară valorile complete și le ordonează crescător. Complexitatea finală a algoritmului este $O(n^2)$, ceea ce face varianta `D` incorectă.

671. Algoritmul `Process` identifică cea mai lungă secvență de elemente consecutive din vector care respectă condiția ca diferența dintre sumele cifrelor a două elemente consecutive să fie cel mult 2. Se menține suma curentă a secvenței și se compară cu cea mai lungă secvență găsită anterior. Dacă există mai multe secvențe cu lungime maximă, se reține suma cea mai mare dintre ele.

672. Algoritmul `ceFace(v, st, dr, x)` împarte recursiv vectorul `v` în două segmente și reorganizează elementele în funcție de suma sau produsul elementelor, în funcție de parametrul `x`. Dacă `x` este 1, rezultatul final este suma valorilor calculate pentru cele două segmente, iar dacă `x` este 0, se utilizează produsul elementelor nenule pentru decizie.

673. Algoritmul `Build(n, k, ant)` determină numărul total de subsecvențe de lungime

`k` din cifrele numărului `n`, respectând condiția ca diferența dintre orice două cifre consecutive din subsecvență să fie impară. Soluția corectă este determinată printr-o combinație de explorare recursivă a cifrelor și verificare a validității condiției impuse. Pentru varianta de răspuns `A` subsecvențele găsite sunt: 345, 145, 125, 234, 123, ceea ce o face o variantă corectă de răspuns.

Pentru varianta de răspuns `C` subsecvențele găsite sunt: 38, 58, 38, 43, 63, 54, 34, 56, 36, ceea ce o face, de asemenea o variantă corectă.

Pentru varianta de răspuns `D` subsecvențele găsite sunt: 56, 76, 96, 54, 74, 94. Aceasta este, de asemenea o variantă corectă.

674. Funcția calculează sumele parțiale ale elementelor din a în vectorul b, iar apoi parcurge b de la indicele k la n pentru a determina subsecvența de lungime k cu suma maximă. Prin $b[i] - b[i-k]$ se obține suma subsecvenței $a[i-k+1], a[i-k+2], \ldots, a[i]$. Dacă această sumă este mai mare decât c (suma maximă curentă), se actualizează indicii st și dr. La final se afișează subsecvența de lungime k cu suma maximă, deci afirmația C este corectă.

675. Pentru fiecare (i) de la 1 la (n), bucla while dublează valoarea lui (x) până ajunge la (n). Numărul de dublări necesare pentru fiecare (i) este $\log_2(\frac{n}{i})$. Astfel, pentru fiecare (i), bucla while se execută de $O(\log_2(n))$ ori. Bucla exterioară rulează de (n) ori, deci complexitatea totală este $O(n \cdot \log_2(n))$.

676. Algoritmul `Prime(n)` determină factorii primi ai numărului `n` și construiește un număr rezultat pe baza unei reguli specifice. Se începe cu `d = 2`, iar pentru fiecare factor prim `d` al lui `n`, se contorizează de câte ori acesta apare în descompunere. Dacă un factor prim apare de mai multe ori, se adaugă la rezultat doar valoarea factorului prim. Dacă apare o singură dată, atunci se adaugă pătratul acestuia. Pe măsură ce noi factori primi sunt identificați, poziția acestora în numărul rezultat este ajustată prin înmulțirea cu 10 a variabilei `putere`, astfel încât fiecare factor nou să fie adăugat pe o poziție mai semnificativă.

Pentru varianta de răspuns `C`, algoritmul confirmă că pentru fiecare factor prim p care apare de k ori în descompunerea lui n, dacă $k > 1$, atunci p este adăugat ca atare în

numărul rezultat. Dacă $k = 1$, atunci se adaugă p^2. Astfel, afirmația `C` este corectă.

Pentru varianta de răspuns `D`, se analizează cazul în care n este un număr prim. Deoarece un număr prim nu are alți factori primi în afară de el însuși și este divizibil doar de 1 și de el însuși, contorul său de apariții în descompunere va fi exact 1. Conform regulii algoritmului, dacă un factor prim apare o singură dată, atunci se adaugă p^2 la rezultat. Astfel, pentru orice număr prim p, algoritmul returnează p^2, ceea ce face ca afirmația `D` să fie corectă.

Pe baza acestor observații, afirmațiile corecte sunt `C` și `D`.

677. Algoritmul `Base(n)` parcurge, pentru fiecare bază de la 2 la 9, reprezentarea numărului `n` în baza respectivă, calculând suma cifrelor și numărând câte dintre acestea sunt cifre pare nenule. Dacă suma obținută este divizibilă cu baza curentă, se procedează astfel: în cazul în care există cel puțin o cifră pară nenulă, se adaugă la rezultatul final valoarea formată din baza curentă urmată de numărul de cifre pare (prin înmulțirea bazei cu 10 și apoi adunarea numărului de cifre pare), iar în caz contrar, se adaugă suma cifrelor. Pentru exemplul în care $n = 15$, se analizează reprezentările lui 15 în diferitele baze. În anumite baze condiția de divizibilitate a sumei cifrelor cu baza este îndeplinită, iar contribuția calculată conform mecanismului descris conduce, în urma adunării tuturor contribuțiilor, la valoarea finală 21. Așadar, variantele corecte de răspuns sunt `A` și `C`.

678. Numărul de moduri în care cel mult două cărți sunt plasate greșit pe raft se determină astfel: Dacă nicio carte nu este plasată greșit, există exact o singură posibilitate (toate cărțile sunt la locul lor). Dacă exact o carte este plasată greșit, acest lucru nu este posibil, deoarece orice mutare afectează cel puțin o altă carte. Dacă exact două cărți își schimbă locurile, putem alege orice două cărți dintre cele m și le putem inversa. Numărul de astfel de perechi este $C_m^2 = \frac{m(m-1)}{2}$. Prin urmare, numărul total de aranjamente posibile în care cel mult două cărți sunt pe poziții greșite este $C_m^2 + 1 = \frac{m(m-1)}{2} + 1$.

679. Algoritmul `Search(v, n)` parcurge fiecare element al vectorului `v` și calculează suma cifrelor acestuia folosind funcția auxiliară `Algo(x)`. Apoi verifică dacă există un element ulterior în vector egal cu produsul dintre elementul curent și suma cifrelor sale, utilizând căutarea binară. Algoritmul returnează numărul total de astfel de perechi găsite.

Dacă se analizează varianta de răspuns `B`, se observă că algoritmul parcurge vectorul [2, 4, 8, 16, 32, 64] și, pentru fiecare element, calculează suma cifrelor folosind funcția auxiliară. De exemplu, pentru elementul 2 suma este 2, iar produsul 2*2 este 4, care se găsește ulterior în vector. Similar, pentru elementul 4 suma este 4, iar produsul 4*4 este 16, iar pentru elementul 8 suma este 8, iar produsul 8*8 este 64. Aceste trei situații determină incrementarea contorului, rezultând o valoare finală de 3, ceea ce demonstrează că varianta `B` este corectă.

680. Algoritmul `ceFace(A, n, m)` determină suma elementelor care au valoarea maximă atât pe linia cât și pe coloana lor. Pentru fiecare rând, se identifică valoarea maximă și poziția sa, apoi se verifică dacă aceasta este strict mai mare decât toate celelalte elemente din coloana respectivă folosind funcția `Check`. Dacă condiția este îndeplinită, valoarea este adăugată la sumă.

681. Algoritmul `Cool(v, n)` parcurge vectorul `v` și verifică fiecare pereche de elemente consecutive. O pereche este considerată validă dacă primul element este mai mic decât al doilea și dacă suma cifrelor celor două elemente este aceeași. Funcția auxiliară `Algo(x)` determină suma cifrelor unui număr prin extragerea fiecărei cifre și adunarea acestora.

Pentru fiecare pereche $(v[i], v[i+1])$, se calculează suma cifrelor fiecărui element și se compară. Dacă sunt egale și $v[i] < v[i+1]$, contorul r este incrementat. La final, se returnează numărul total de astfel de perechi.

Pentru apelul `Cool([12, 21, 13, 31, 15], 5)`, rezultatul este 2, deoarece perechile valide sunt (12,21) și (13,31). Dacă vectorul este sortat crescător și toate elementele au aceeași sumă a cifrelor, toate perechile consecutive sunt valide, iar rezultatul va fi $n-1$. De asemenea, pentru apelul `Cool([21, 30, 15, 24, 42, 51], 6)`, rezultatul obținut este 4, confirmând validitatea răspunsului.

Varianta de răspuns `C` nu este corectă, deoarece elementele consecutive trebuie să fie sortate strict crescător pentru ca această proprietate să fie îndeplinită.

Astfel, afirmațiile corecte sunt `A`, `B` și `D`.

682. Metoda `d(i, a, v, n, c)` este o parcurgere în adâncime în cadrul căreia, în șirul de noduri "vizitate" (aici, v) în loc să marcăm doar vizitarea nodului, marcăm cu valoarea

actuală a unui contor, iar astfel, $v[i]$ va reprezenta numărul de ordine al componentei conexe din care i face parte.

683. Vom trata cele două șirurui ca fiind mulțimi, și vom reconstrui șirurile, pentru a obține valoarea cea mai mare a lui d. Pentru fiecare poziție, vom extrage din fiecare mulțime cel mai mare element, cât și cel mai mic. Vom compara cel mai mare element din A cu cel mai mare element din B. În cazul în care elementul din A este mai mare decât elementul din B, atunci vom așeza pe poziția curentă cel mai mare element din fiecare mulțime. Altfel, dacă cel mai mic element din A este mai mare decât cel mai mare element din B, atunci vom așeza pe poziția curentă cel mai mic element din fiecare mulțime. Altfel, vom așeza în primul șir cel mai mic element din A, iar în a doua cel mai mare element din B. Primele două așezări vor rezulta în creșterea lui d, iar ultima va rezulta în scăderea acestuia.

684. Codul verifică dacă un șir este palindrom după inversarea elementelor din a doua jumătate a șirului. Variantele A și B nu îndeplinesc condițiile.

685. Algoritmul parcurge numerele de la 1 la n, numărând pentru fiecare număr câți divizori are și verificând dacă depășesc pragul dat.

686. Algoritmul parcurge numerele de la 1 la n, calculând suma divizorilor fiecărui număr și verificând dacă această sumă este divizibilă cu p. Funcția `Find` calculează suma tuturor divizorilor unui număr, inclusiv 1 și numărul însuși.

687. Algoritmul `X(v, n, k)` determină numărul de subsecvențe ale vectorului `v` a căror sumă este exact `k`. Funcția `Build` construiește un vector de sume parțiale care permite calculul rapid al sumei oricărei subsecvențe. Algoritmul parcurge toate perechile de indexi (i, j) și verifică dacă diferența dintre două sume parțiale este egală cu `k`. Această abordare are o complexitate de timp de $O(n^2)$, deoarece utilizează două bucle imbricate pentru a examina toate subsecvențele posibile.

688. Algoritmul `ceFace(v, n, idx, sumc, k, c)` explorează toate subseturile posibile ale vectorului `v` pentru a determina câte dintre acestea au suma elementelor egală cu `k`. Apelul inițial al funcției `Start(v, n, k)` inițializează numărul de soluții `c` cu 0 și pornește procesul de căutare recursivă. Algoritmul verifică fiecare subset posibil, incluzând sau

excluzând fiecare element din vector, ceea ce duce la o complexitate exponențială de $O(2^n)$. Această complexitate rezultă din faptul că fiecare element poate fi fie inclus, fie exclus, generând astfel toate subseturile posibile. Funcția este corectă indiferent de semnul elementelor din vector, deci afirmația conform căreia funcționează corect doar pentru numere pozitive este falsă.

689. Algoritmul `Find` utilizează backtracking pentru a genera toate subseturile unui vector `v` și verifică dacă suma acestora se încadrează în limita maximă `m`. Algoritmul explorează fiecare element din vector, având posibilitatea fie să îl includă în subsetul curent, fie să îl excludă. Aceasta permite generarea tuturor combinațiilor posibile.

Varianta corectă a algoritmului este algoritmul **C**, deoarece acesta verifică în mod corect dacă adăugarea unui element la suma curentă depășește limita `m` înainte de a efectua apelul recursiv. Mai mult, algoritmul **C** parcurge toate subseturile posibile și le afișează corect, respectând constrângerea impusă de `m`.

Algoritmul **A** nu funcționează corect deoarece permite apeluri recursive chiar și atunci când suma curentă depășește `m` și indexarea începe de la 0, ceea ce poate duce la rezultate incorecte. Algoritmul **B** nu generează toate subseturile posibile, deoarece exclude din start anumite combinații. Algoritmul **D** introduce o afișare redundantă după apelurile recursive, ceea ce rezultă în dublarea seturilor și poate duce la rezultate incorecte sau afișări multiple.

Prin urmare, algoritmul care identifică corect și eficient toate subseturile valide este **C**.

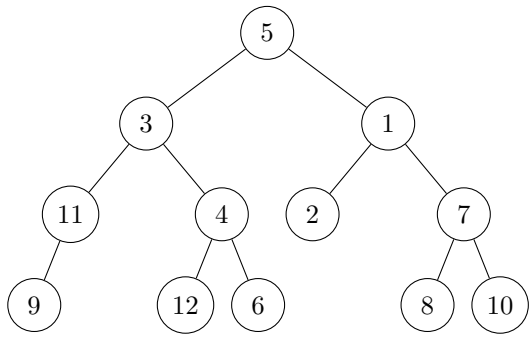
690. Algoritmul `func(x, y, d)` explorează matricea folosind o căutare recursivă pentru a găsi comoara, reprezentată de valoarea 2. Se deplasează în cele patru direcții posibile (sus, jos, stânga, dreapta), evitând obstacolele (valoarea 1) și celulele deja vizitate, marcate în matricea `v`. Dacă se ajunge la poziția comorii, valoarea `d` (care reprezintă distanța parcursă) este comparată cu `m` și, dacă este mai mică, `m` este actualizat.

691. Algoritmul `ceFace(n, x, len, last)` generează recursiv toate secvențele de lungime n formate din divizorii lui x, respectând condiția ca produsul oricăror două elemente consecutive să fie mai mic decât x. În apelul `ceFace(5, 10, 0, 1)` se construiesc secvențe din divizorii lui 10, iar restricția $last * d < x$ este verificată pentru a asigura

validitatea fiecărei tranziții între elemente. Pe măsură ce se explorează toate combinațiile posibile în ordinea recursivității, cea de-a opta secvență generată este 1 1 2 1 2. Această secvență îndeplinește exact condiția impusă de algoritm, confirmând astfel corectitudinea afirmației din varianta `C`.

692. $((x{+}1)$ **MOD** $(y{-}1)) = 15$ **MOD** $4 = 3$; Prima parte: $3{+}2 = 5$; $((x{+}1)$ **DIV** $(z{+}1)) = 15$ **DIV** $3 = 5$; Rezultatul primei părți: 5 **DIV** $5 = 1$; $((y-1)$ **MOD** $z) = 4$ **MOD** $2 = 0$; Rezultatul final: $1 + 0 = 1$.

693. Arborele dat are următoarea reprezentare:



Lanţul dintre nodurile 6 şi 9 este $6 \to 4 \to 3 \to 11 \to 9$, care nu conţine rădăcina, deci varianta A este corectă. Nodurile 2 şi 10 se află în subarborele drept al rădăcinii, deci varianta B este incorectă. Nodurile 9, 12, 6, 2, 8 şi 10 reprezintă frunzele arborelui, iar suma lor este 47, deci varianta C este corectă. Lungimea lanţului dintre nodurile 6 şi 10 este 6, deci varianta D este incorectă.

694. Varianta C este corectă: $(1$ **ȘI** $1)$ **SAU NU** $(1$ **SAU** $1) \to 1$ **SAU** $0 \to 1$; Varianta B este corectă:$(1$ **ȘI** $1)$ **SAU NU** $(0$ **SAU** $0) \to 1$ **SAU** $1 \to 1$.

695. Algoritmul construiește un șir de sume parțiale și îl folosește pentru a calcula eficient suma numerelor cu indicii în intervalul $[i, j]$

696. Algoritmul verifică dacă există un drum de la colțul stânga sus la colțul dreapta jos folosind Metoda Backtracking (generează toate drumurile posibile).

697. Pentru a verifica dacă întreaga matrice conține doar numere pare, trebuie să verificăm fiecare element și să ne asigurăm că toate sunt pare. Suma valorilor returnate de g trebuie să fie egală cu numărul total de elemente $n \times n$.

698. Folosind ,,Ciurul lui Eratostene", algoritmul marchează cu 0 numerele prime, iar cu 1 numerele compuse. La început, toate valorile din p sunt 0. Sunt parcurse toate numerele de la 2 până la radicalul lui n, iar pentru cele cu valoarea asociată lor din vectorul p egală cu 0 (acestea vor fi numerele prime), sunt marcaţi multiplii lor cu valoarea 1, semnificând faptul că aceştia sunt compuşi, deoarece sunt divizibili cu p. Prin urmare, varianta A este corectă, iar varianta B este incorectă. 2 este un număr prim, deci valoarea asociată în vectorul p va fi 0. Aşadar, varianta C este incorectă. 10 este un număr compus, deci valoarea asociată în vectorul p va fi 1. Aşadar, varianta D este corectă.

699. Pentru fiecare apel recursiv, sunt făcute alte 3 apeluri, ceea ce face ca numărul de paşi să crească exponenţial. Pentru fiecare nou apel, valoarea lui n este împărţită la m, ceea ce face ca valoarea lui n să descrească logaritmic. Ca urmare, complexitatea algoritmului dat este $O(3^{log_m n}) = O(n^{log_m 3})$. Calculul poate fi efectuat, de asemenea, folosind Teorema Master. Complexitatea algoritmului dat poate fi scrisă astfel: $T(n, m) = 3T(\frac{n}{m}, m)$. Sunt identificate constantele: $a = 3, b = m, k = 0$ şi $p = 0$. Datorită faptului că $a > b^k$, deoarece $3 > 1$, $T(n, m) = \Theta(n^{\log_b a}) = \Theta(n^{\log_m 3})$. În concluzie, variantele B şi D sunt corecte, iar variantele A şi C sunt incorecte.

700. Diferența dintre cei doi algoritmi constă în faptul că, în implementarea algoritmului `A(a, n)`, compararea se face folosind operatorul $\geq$, ceea ce poate duce la rezultate incorecte pentru șiruri care conțin elemente egale. Pentru a avea comportament echivalent, operatorul ar trebui schimbat în $>$, asigurând astfel stabilitatea și corectitudinea algoritmului. Algoritmul `A(a, n)` este un algoritm de Bubble Sort, iar `B(a, n)` Selection Sort.

701. Algoritmul generează permutările şirului numerelor de la 1 la n $\{1, 2, ..., n\}$, pentru care valoarea elementului este diferită de poziţia pe care se află aceasta. Aşadar, varianta C este corectă, în timp ce varianta D este greşită. Acest fapt presupune, spre exemplu, că valoarea 1 nu se va afla niciodată pe prima poziţie a unei permutări (numerotarea indicilor vectorilor începe de la 1). Această condiţie este impusă prin $i \neq k$, iar metoda folosită este backtracking, bazată pe un vector a, care memorează valorile unei permutări şi un vector de frecvenţă f, care asigură ca fiecare valoare să se regăsească exact o dată într-o soluţie. Pentru $n = 3$, soluţiile vor fi $\{2, 3, 1\}$ şi $\{3, 1, 2\}$, deci varianta A este corectă.

Varianta B este incorectă, 24 este numărul de permutări a unei mulţimi cu 4 elemente, fără vreo condiţie suplimentară.

702. Algoritmul simulează metoda de sortare "Bubble Sort" și efectuează interschimbări doar atunci când un element este mai mare decât elementul său următor. În cazul unui șir deja sortat sau cu toate elementele egale, nu sunt necesare interschimbări. În cazul unui șir sortat descrescător, algoritmul efectuează numărul maxim de interschimbări, adică $\frac{n(n-1)}{2}$.

703. Algoritmul de sortare prin selecție parcurge întotdeauna întregul șir pentru a găsi elementul minim, efectuând $O(n^2)$ comparații indiferent de ordinea inițială a elementelor. Complexitatea în cel mai bun și cel mai rău caz rămâne $O(n^2)$, iar numărul de comparații este constant pentru orice intrare.

704. La fiecare pas, algoritmul va separa elementele aflate pe pozițiile pare de cele care se află pe poziții impare, după care, recursiv se va autoapela câte o dată pentru fiecare șir în parte. După executare, în vectorul inițial se vor reașeza elementele din cei 2 vectori mai mici. Observăm că la fiecare pas, în primul vector se vor afla elemente ce au ultimul bit 0, iar în al doilea cele care au ultimul bit 1. Notăm cu *ogl* oglinditul unui număr i. Poziția finală este determinată de către oglinditul în baza a poziției inițiale. În cazul în care numărul are mai puțin de k biți, se vor adăuga biți de 0 la începutul acestuia, pentru a ajunge la k biți. Exemplu: $n = 16, i = 3 => i = 0011, ogl = 1100 => ogl = 12$, elementul aflat inițial la poziția 3 va ajunge la poziția 11. Pentru a determina numărul de poziții care vor avea la final același element, trebuie să determinăm numărul de poziții care au aceeași valoare atât pentru număr, cât și pentru oglindit în baza 2, care este egal cu $2^{(k+1)/2}$

705. Complexitatea unui apel este O(n), iar, algoritmul bazându-se pe divide et impera, va avea log apeluri, ajungând la complexitatea $O(nlogn)$, sau $O(nk), k = \log n$

706. Algoritmul calculează suma divizorilor unui număr.

707. Algoritmul calculează suma cifrelor unui număr, până la o cifră, cunoscută în specialitate ca și cifra de contor, și este egală cu $n\%9$, dacă $n\%9$ este diferit de 0, sau 9 dacă n este multiplu de 9.

708. Traversarea arborelui în inordine presupune parcurgerea subarborelui stâng, apoi procesarea rădăcinii, iar apoi parcurgerea subarborelui drept.

709. Dacă numărul de linii și coloane este impar, ultima celulă vizitată va fi situată pe linia și coloana mediană. Dacă numărul de linii și coloane este par, ultima celulă vizitată va fi situată pe linia mediană și coloana mediană - 1.

710. Analizăm traseul lui Algernon. Inițial, el se va vizita 50 de celule, schimbând după direcția. Apoi, vizitează 49 de celule, schimbă direcția, vizitează din nou 49 de celule, și schimbă direcția. Continuând simularea, observăm că Algernon va vizita același număr de celule pentru perechi de laturi ale labirintului (pentru laturile de jos și din dreapta, va vizita un număr impar de celule, iar pentru laturile de sus și din stânga, va vizita un număr par de celule). Așadar, numărul total de celule vizitate de Algernon va fi $50 + 2 * (49 + 48 + \cdots + 1) = 2500$. Pentru a determina care este poziția celulei noastre, vom căuta ultima pereche de laturi pe care Algernon a vizitat-o, după care vom continua cu parcurgerea celulelor de pe laturile rămase. Vom avea ecuația: $50 + 2 * (49 + 48 + \cdots + x) \leq 1710$. Rezolvând ecuația, obținem $x = 29$, vizitând în total astfel 1688 de celule. Pentru că 29 este impar, Algernon a vizitat ultima dată o pereche de forma (stânga, jos). Când a vizitat prima dată perechea (stânga, jos), a vizitat 49 de celule pe fiecare latură, ajungând la final la celula (50, 1). La următoarea vizită pentru această pereche, va vizita 47 de celule, ajungând la celula (49, 2). La următoarea vizită, va ajunge la celula (48, 3), și tot așa. Pentru $x = 29$, va ajunge la celula (40, 11). Lui Algernon i-au rămas de vizitat 22 de celule, care le va vizita pe laturile de sus, ajungând la linia $40 - 22 = 18$, deci la celula $(18, 11)$ la final.

711. i va fi inițializat cu prima putere a lui 2 mai mare decât n, calculată în complexitate $O(\log n)$. În interiorul buclei, i va fi împărțit la 2 de fiecare dată, indiferent de condiție, deci numărul de iterații va fi $\log n$. În total, complexitatea algoritmului va fi $O(\log n)$.

712. Funcția $find$ caută binar poziția pe care elementul x ar trebui să fie în șirul a. Orice număr natural se poate scrie ca și o sumă de puteri ale lui doi. Pentru orice număr natural k, $2^k \geq (2^1 + 2^2 + \cdots + 2^(k-1))$. Așadar, pornind de la cea mai mică putere a lui 2 mai mare sau egală cu n, se va împărți această putere la 2 de fiecare dată, până când se

va ajunge la 1. În cazul în care adăugarea acelei valori la posibilul rezultat nu depășește x, se va adăuga acea valoare la poziția rezultatului. În caz contrar, se va continua cu următoarea valoare. Astfel, pentru $n = 13254, x = 8342$, se va ajunge la poziția 8192, 8192 + 1024, 8192 + 1024 + 128, 8192 + 1024 + 128 + 8, 8192 + 1024 + 128 + 8 + 2, deci de 5 ori. În specialitate, această tehnică se numește căutare binară pe biți, numărul de modificări ale răspunsului corespunde numărului de biți de 1 din reprezentarea binară a lui *poz* (8342 = 10000010010110).

713. Algoritmul generează valorile din şirul lui Fibonacci, însă duplică valoarea $F(n)$, pentru $F(n+1)$, unde n MOD 3 = 0, deci varianta A este incorectă. Considerând 3 numere naturale consecutive $n-1$, $n-2$ şi $n-3$, exact unul dintre acestea va fi divizibil cu 3, aşa că apelul nu va avea loc pentru aceea valoare. Împreună cu faptul că n descreşte linear, complexitatea ca timp de execuţie va fi $O(2^n)$, aşadar varianta B este corectă. $F(5) = F(4)+F(2) = F(2)+F(1)+F(2) = 1+1+1 = 3$. Aşadar, varianta C este corectă. $F(7) = F(5)+F(4) = F(4)+F(2)+F(2)+F(1) = F(2)+F(1)+F(2)+F(2)+F(1) = 1+1+1+1+1 = 5$. Aşadar, varianta D este incorectă.

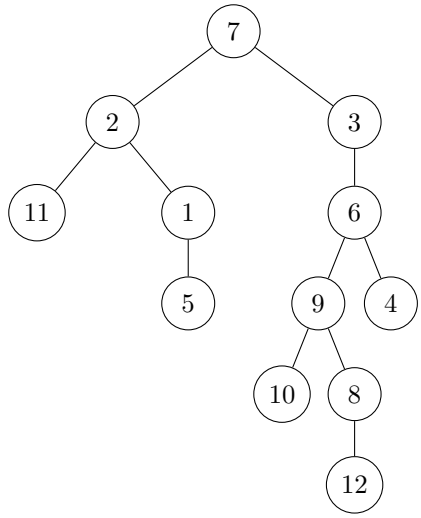
714. Pentru valorile n pare , prima dată va fi făcut apelul funţiei `alt` pentru n DIV 10, iar după finalizarea execuţiei acestuia, va fi afişată ultima cifră a lui n, incrementată. Pentru cele impare, afişarea va fi făcută înaintea apelului. Aşadar, algoritmul afişează mai întâi valorile $4, 2$ şi 0 pentru cifrele $5, 3$ şi 1, iar apoi, $3, 5$ şi 7 pentru cifrele $2, 4$ şi 6.

715. Inmulţirea matricelor este asociativă, iar ca urmare, înmulţirile necesare pentru calcularea produsului pot fi făcute în orice ordine. Fiecare parantetizare (fără modificarea ordinii matricelor) a expresiei produsului determină o ordine diferită de efectuare a înmulţirilor, fiecare necesitând un număr diferit de produse scalare între valorile matricelor. Pentru 2 matrice $M_{p1,q1}, N_{p2,q2}$, cu $q_1 = p_2$, produsul dintre acestea va necesita $p_1 \cdot q_1 \cdot q_2$ produse scalare. Pentru $n = 3$ şi matricele $M_{1,2}, M_{2,3}$ şi $M_{3,2}$, expresia $(M_{1,2} \times M_{2,3}) \times M_{3,2}$ necesită 12 înmulţiri scalare, în timp ce expresia $M_{1,2} \times (M_{2,3} \times M_{3,2})$ necesită 16 înmulţiri scalare. Aşadar, varianta B este corectă. Pentru $n = 4$, expresia $M_{6,5} \times (M_{5,4} \times (M_{4,3} \times M_{3,2}))$ va conduce la numărul minim de produse scalare, 124. Aşadar, varianta A este corectă. Numărul de moduri de a parantetiza expresia produsu-

lui a $n+1$ matrice este dat de al n-lea număr Catalan, $C_n = \frac{1}{n+1}C_{2n}^n$.

716. $(x$ **MOD** $y) = 15$ **MOD** $6 = 3$; Prima sumă: $3+3 = 6$; $((x-2)+y) = 13+6 = 19$; Prima parte: 6 **DIV** $19 = 0$; $((z+1)$ **MOD** $(y+2)) = 4$ **MOD** $8 = 4$; Rezultatul final: $0 + 4 = 4$.

717. Arborele dat are următoarea reprezentare:



Nodurile 11, 5, 10, 12 şi 4 reprezintă frunzele acestuia. Aşadar, arborele are 5 frunze şi 7 noduri interioare, iar varianta A este incorectă. Nodurile 9 şi 4 sunt descendenţii direcţii ai nodului 6, deci acestea sunt fraţi, iar varianta B este corectă. Lanţul dintre nodurile 11 şi 10 este $11 \to 2 \to 7 \to 3 \to 6 \to 9 \to 10$, care trece prin rădăcină, care este marcată cu 7, deci varianta C este incorectă. Lungimea lanţului dintre nodurile 5 şi 9 este 6, deci varianta D este corectă.

718. Algoritmul creează un vector de sume prefixate pentru elementele pare și îl folosește pentru a calcula suma rapidă a elementelor pare dintr-un interval.

719. Varianta A este corectă, deoarece ambele condiții trebuie să fie adevărate pentru ca x să fie în interval. Varianta D este echivalentă cu varianta A, deci este corectă. Varianta B este echivalentă cu faptul că x nu aparține intervalului $[a, b]$. Varianta C nu impune condițiile suficiente.

720. Algoritmul realizează transpunerea unei matrice pătratice prin interschimbarea elementelor de deasupra și sub diagonala principală, menținând elementele de pe diagonală

neschimbate.

721. Algoritmul înmulțește două matrice pătratice de dimensiune $n \times n$ folosind formula clasică de înmulțire a matricelor.

722. Traversarea arborelui în preordine presupune procesarea rădăcinii, apoi parcurgerea subarborelui stâng, iar apoi a subarborelui drept.

723. Algoritmul verifică la fiecare pas dacă s-au realizat permutări. Dacă nu s-au realizat, înseamnă că șirul este deja sortat și se poate opri execuția, reducând numărul de pași ai algoritmului. Pentru un șir deja sortat, se efectuează doar $n - 1$ comparații, așa că, varianta B este falsă. Totuși, pentru cazul în care elementele șirului sunt ordonate strict descrescător, complexitatea rămâne $O(n^2)$, la fel ca în varianta clasică a algoritmului `BubbleSort`.

724. Algoritmul generează combinări de n luate câte k, pentru care elementele sunt în ordine crescătoare, iar diferenţa dintre 2 elemente alăturate este mai mare sau egală cu 2, folosind metoda backtracking. Variabila *start* indică valoare minimă pe care o va lua următorul element din combinare, *position* indică poziţia din combinare pe care va fi plasată valoarea curentă, k reprezintă numărul de valori dintr-o combinare validă, n este numărul total de elemente, iar a memorează pe rând soluţiile algoritmului. Variantele A şi D sunt incorecte, deoarece prin creşterea valorii *start* cu 2 faţă de valoarea curentă, elementele consecutive ale unei combinări vor avea diferenţa mai mare sau egală cu 2. Varianta B este incorectă, deoarece elementele combinărilor vor avea valori strict crescătoare. Varianta C este, aşadar, corectă.

725. Pentru șirul $a = [1, 0, 1, 1, 1]$ (varianta B), algoritmul de sortare va efectua exact o interschimbare între primul element și al doilea, rezultând ordinea corectă: $[0, 1, 1, 1, 1]$. Astfel, interschimbarea se efectuează o singură dată. Pentru șirul $a = [1, 2, 3, 4, 5, 6]$ (varianta C), algoritmul de sortare nu va efectua nicio interschimbare deoarece șirul este deja ordonat crescător. În schimb, pentru șirurile $a = [1, 0, 5, 7, 2, 3, 8]$ (varianta A) și $a = [1, 0, 1, 0, 1, 0, 1]$ (varianta D), algoritmul va efectua mai multe interschimbări pentru a obține o ordonare corectă.

726. Algoritmul `CeFace` funcționează în trei etape: identifică și mută toate numerele

prime la începutul șirului; sortează numerele prime în ordine crescătoare folosind Bubble Sort; sortează numerele neprime în ordine descrescătoare folosind Insertion Sort.

727. Pentru fiecare apel recursiv al algoritmului, sunt făcute alte m apeluri recursive, pentru valori $n \geq 1$, ceea ce face ca numărul de paşi să crească exponenţial. Pentru fiecare nou apel, valoarea lui n este împărţită la 2, ceea ce face ca valoarea lui n să descrească logaritmic. Ca urmare, complexitatea algoritmului dat este $O(m^{logn})$. Calculul poate fi efectuat, de asemenea, folosind Teorema Master. Complexitatea de timp a algoritmului poate fi descrisă astfel: $T(n,m) = mT\left(\frac{n}{2}, m\right)$. Sunt identificate constantele: $a = m, b = 2, k = 0$ şi $p = 0$. Datorită faptului că $a > b^k$, deoarece $m > 1$, $T(n,m) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 m}) = \Theta(m^{\log_2 n})$. În concluzie, varianta C este corectă, iar variantele A, B, D sunt incorecte.

728. Algoritmul calculează suma divizorilor primi ai unui număr.

729. Algoritmul parcurge numerele de la 2 la n şi marchează în vectorul p numerele prime cu valoarea acestora, iar pentru multiplii lor, care nu au fost procesaţi până în acel moment, setează valoarea corespunzătoare din p cu valoarea numărului prim găsit anterior. Aşadar, varianta A este incorectă, iar varianta B este corectă. Cel mai mic factor prim al lui 10 este 2, deci varianta C este incorectă, iar cel mai mic factor prim al lui 5 este 5, deci varianta D este corectă.

730. Vom face câteva observații cheie: O formație cu k rânduri conține $2^k - 1$ buline, iar adăugare unui nou rând la o formație deja existentă va folosi mai multe buline decât crearea unei noi formații identice cu cea deja existentă. Așadar, scopul nostru este să formăm piramide cu cât mai multe rânduri. Determinăm câte buline avem la dispoziție. Cât timp mai avem buline, căutăm cel mai apropiat număr $2^k - 1$ și formăm o formație cu k rânduri, după care scădem numărul de buline folosite. În total, avem 182 de buline. Vom avea: 1 formație cu 7 rânduri (127 de buline), o formație cu 5 rânduri (31 de buline), o formație cu 4 rânduri (15 buline), o formație cu 3 rânduri (7 buline) și două formații cu 2 rânduri (2 buline). În total, avem 6 formații.

731. Algoritmul calculează combinările de n luate câte k. Numărul de apeluri se poate determina prin construirea unei funcții similare, care însă, pe lângă suma celor două ape-

luri recursive, mai adaugă un 1 la fiecare apel, pentru a număra și apelul curent. Așadar, rezultatul final va fi $2 * (C_{10}^{7} - 1)$.

732. Indiferent de ordinea în care se fac mișcările, drumul va avea a mișcări de tipul $(x, y) \to (x + 1, y)$ și b mișcări de tipul $(x, y) \to (x, y + 1)$. Să notăm cu 0 fiecare mișcare de tipul $(x, y) \to (x + 1, y)$ și cu 1 fiecare mișcare de tipul $(x, y) \to (x, y + 1)$. Drumul va fi reprezentat de un șir de lungime $a + b$ format din a 0-uri și b 1-uri, deci vom avea de așezat a 0-uri într-un șir de lungime $a + b$. Cu alte cuvinte, trebuie să selectăm a numere din $a + b$ posibile, deci numărul de drumuri distincte este $\binom{a+b}{a}$.

733. $F_{k+2} - F_{k+1} = F_k \implies \sum_{k=1}^{n}(F_{k+2} - F_{k+1}) = \sum_{k=1}^{n} F_k \implies (F_3 - F_2) + (F_4 - F_3) + ... + (F_{n+2} - F_{n+1}) = \sum_{k=1}^{n} F_k \implies F_{n+2} - F_2 = \sum_{k=1}^{n} F_k \implies F_{n+2} - 1 = \sum_{k=1}^{n} F_k$. Aşadar, singura variantă corectă este D.

734. Algoritmul se numeşte "Indicatorul lui Euler" şi calculează numărul de valori mai mici sau egale cu n, care sunt prime cu n, folosind formula $T(n) = n \cdot \prod_{p|n}\left(1 - \frac{1}{p}\right)$. Aşadar, varianta A este incorectă, dar variantele B şi C sunt corecte. Pentru n prim, acesta este prim cu toate numerele mai mici decât acesta, deci $T(n) = n - 1$. În concluzie, varianta D este corectă.

735. Numerele naturale mai mici sau egale cu 10, care sunt prime cu 10 sunt $1, 3, 7, 9$, cele mai mici sau egale cu 7, prime cu 7 sunt $1, 2, 3, 4, 5, 6$, iar numărul mai mic sau egal cu 1, prim cu 1 este 1. Similar, $T(25) = 20$. Aşadar, varianta A este corectă, varianta B este incorectă, varianta C este corectă, iar varianta D este incorectă.

736. Să ne imaginăm o matrice pătratică de dimensiune $n \times n$. Fie $D(n)$ valoarea pe care *cnt* o ia pentru un anumit n dat. Observăm că, la fiecare pas, matricea noastră se împarte în 4 submatrice. Condiția inițială impune ca matricea pe care lucrăm să fie pătratică, pentur a exista modificări asupra contorului. Dacă matricea noastră are dimensiune impară, atunci două matrice nu vor respecta condiția inițială (exemplu: pentru o matrice de dimensiune 5×5, obținem matricele de dimensiune 3×3, 2×3, 3×2 și 2×2) Așadar,

avem relația de recurență:

$$D(n) = \begin{cases} 1, & \text{dacă } n = 1, \\ 4 * D(\frac{n}{2}) + 1, & \text{dacă } n \text{ este par} \\ D(\frac{n}{2}) + D(\frac{n}{2} + 1) + 1, & \text{dacă } n \text{ este impar} \end{cases}$$

737. Recursiv, algoritmul calculează suma elementelor dintr-un șir. Complexitatea acestuia este $O(n)$, deoarece fiecare element din șir va fi accesat o singură dată, când va ajunge la el în apelul recursiv.

738. Momentul la care toate becurile luminează deodată trebuie să fie un multiplu comun al tuturor elementelor din s. Dintre variante, 2520 și 5040 verifică această proprietate.

739. Problema se schimbă față de cea anterioară prin faptul că becurile luminează pentru o perioadă mai îndelungată. Primul bec va lumina, pe rând, între secundele cuprinse în intervalul [7, 9], după aceea între secundele [16, 18], după aceea între secundele [25, 27], și tot așa. Generalizând, un bec i va lumina între secundele $[s[i] * k - 2, s[i] * k]$. Momentul în care luminează toate becurile deodată trebuie să aparțină unui interval comun tuturor becurilor. Dintre variante, 3959 verifică această proprietate.

740. $(12 \texttt{ DIV } (5-1)) \cdot (4 + ((12+1) \texttt{ MOD } (5-1))) - ((4+1) \texttt{ MOD } (5-1)) = 14$

741. Algoritmul `Trick1` folosește sume prefixate pentru a eficientiza interogările după un pas inițial de preprocesare, făcându-l mai eficient decât `Trick2` pentru un număr mare de interogări. Ambii algoritmi returnează pentru o interogare, numărul de apariții al elementului x în intervalul $[i, j]$ din vectorul v, unde i, j sunt poziții în vector.

742. Algoritmul oglindește matricea față de axa verticală prin interschimbarea coloanelor.

743. Algoritmul recursiv trebuie să parcurgă matricea pe linii și coloane și să calculeze suma ultimelor cifre ale tuturor elementelor, trecând la următoarea coloană sau la următoarea linie dacă s-a ajuns la sfârșitul unei coloane.

744. Evaluăm expresia pas cu pas: Prima parte: $a > b \rightarrow 3 > 5$ este fals, deci evaluăm

ramura $(c < d?d - c : a + b)$. $c < d \to 2 < 8$ adevărat, rezultatul este $8 - 2 = 6$. A doua parte: $d < a \to 8 < 3$ este fals, evaluăm $(c > a?c : a + d)$. $c > a \to 2 > 3$ fals, rezultatul este $a + d = 3 + 8 = 11$.Rezultatul expresiei: $6 + 11 = 17$.

745. Traversarea arborelui în postordine presupune parcurgerea subarborelui stâng, apoi a subarborelui drept, iar în final procesarea rădăcinii.

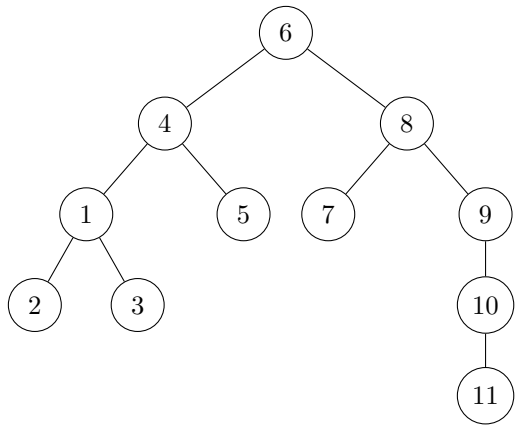
746. Algoritmul calculează numărul de divizori ai numerelor mai mici sau egale decât n. Numerele de la 1 la n sunt parcurse pe rând, iar pentru fiecare multiplu al fiecăruia, mai mic sau egal decât n, valoarea corespunzătoare acestuia din vectorul d este crescută cu 1, fapt care reprezintă găsirea unui nou divizor. Aşadar, varianta A este incorectă, deoarece omite divizorii improprii ai numerelor, însă variantă B este corectă. 10 are 4 divizori (1, 2, 5, 10), deci afirmaţia C este incorectă, iar 24 are 8 divizori (1, 2, 3, 4, 6, 8, 12, 24), deci afirmaţia D este corectă.

747. Ca rezultat al faptului că pentru fiecare apel recursiv, sunt apelate alte m apeluri recursive, numărul de paşi al algoritmului creşte exponenţial odată cu creşterea lui n. Valoarea lui n este împărţită în mod repetat la m, iar ca rezultat, m descreşte logaritmic. În concluzie, complexitatea va fi $O(m^{log_m n}) = O(n)$. Calculul poate fi efectuat, de asemenea, folosind Teorema Master. Complexitatea de timp a algoritmului poate fi descrisă astfel: $T(n, m) = mT\left(\frac{n}{m}, m\right)$. Sunt identificate constantele: $a = m, b = m, k = 0$ şi $p = 0$. Datorită faptului că $a > b^k$, deoarece $m > 1$, $T(n, m) = \Theta(n^{\log_b a}) = \Theta(n^{\log_m m}) = \Theta(m^{\log_m n}) = \Theta(n)$. În concluzie, variantele B şi C sunt corecte, iar variantele A şi D sunt incorecte.

748. Algoritmul generează permutările şirului format din numere naturale de la 1 la n, în care paritatea unei valori este diferită de ceea a poziţiei pe care se află, folosind metoda backtracking. k reprezintă poziţia curentă dintr-o permutare, n este numărul total de elemente ale unei permutări, a este vectorul care reţine pe rând fiecare soluţie a algoritmului, iar f este un vector de frecvenţă, care asigură că fiecare element apare exact o dată într-o soluţie. Aşadar, varianta C este corectă. Pentru n impar, numărul poziţiilor pare este mai mare decât numărul valorilor pare, în timp ce numărul valorilor impare este mai mare decât numărul poziţiilor impare. Aşadar, nu există soluţii în acest caz, iar varianta

B este corectă şi varianta D este incorectă. Pentru n par, este nevoie ca prima valoare să fie una pară, deci varianta A este incorectă.

749. Arborele dat are următoarea reprezentare:



Nodurile 2, 3, 5, 7 şi 11 reprezintă frunzele acestuia, deci varianta C este corectă. Nodul 6 este marcat cu valoarea 0 în vectorul de taţi, deci acesta este rădăcina, iar varianta A este corectă. Numărul de noduri interioare este 6, deci varianta B este incorectă. Înălţimea arborelui este 4, deci varianta D este incorectă.

750. Până la adresa 1, acesta a consumat $10000000_{(2)} = 128_{(10)}$ unități, iar înapoi $1_{(2)} = 1_{(10)}$, adică un total de $129_{(10)}$ de unități pentru prima livrare. Până la adresa 2, acesta a consumat $11100000_{(2)} = 224_{(10)}$ unități, iar înapoi $111_{(2)} = 7_{(10)}$, adică un total de $231_{(10)}$ de unități. Până la adresa 3, acesta a consumat $11111100_{(2)} = 252_{(10)}$ unități, iar înapoi $111111_{(2)} = 63_{(10)}$, adică un total de $315_{(10)}$ de unități. Până la adresa 4, el a parcurs 8 laturi, a luat o pauză și a mai parcurs încă 7 laturi pentru a ajunge la adresă, adică a consumat $255 + 128 + 127 = 510$, și decide să se întoarcă la depozit pe drumul cel mai scurt, în sens invers acelor de ceasornic, și să mai parcurgă 3 laturi, $111_{(2)} = 7_{(10)}$. În total, acesta a consumat 1192 de unități de energie.

751. Algoritmul calculează combinările de n luate câte k, folosindu-se de triunghiul lui Pascal.

752. Structura repetitivă externă se execută de $\log_7(n)$ ori, iar cea internă de $\log_3(n^3)$ ori. Complexitatea algoritmului poate fi scrisă ca $O(\log_7 n * \log_3 n^3) = O(\log_3 n * 3 * \log_3 n) =$

$O(\log_3^2 n) = O(\log_3^2 n^2) = O(\log_7^2 n)$, pentru că, în calculul complexităților, constantele pot fi ignorate.

753. Algoritmul construiește un număr din cifrele lui n după următoarea regulă: analizează fiecare cifră a numărului n și verifică dacă este un număr prim. Dacă este prim, adaugă la numărul final ultimele două cifre precedente cifrei analizate. Dacă nu, decrementează valoarea k.

754. Definiția subalgoritmului $M(n)$ este, de fapt: $M(n) = \begin{cases} 91, & n \leq 101 \\ n-10, & n > 101 \end{cases}$

Aşadar, varianta A este corectă, iar varianta B este incorectă. $M(523) = 513$, deoarece $523 > 101$, deci varianta C este corectă. $M(102) = 92$, deoarece $102 > 101$, deci varianta D este incorectă.

755. Algoritmul dat va returna următoarele valori pentru $n \in \{0, 1, 2, 3, 4\}$:

$$F(0) = 1,$$

$$F(1) = F(F(0) - 1) \cdot 1 = F(0) \cdot 1 = 1,$$

$$F(2) = F(F(1) - 1) \cdot 2 = F(0) \cdot 2 = 2,$$

$$F(3) = F(F(2) - 1) \cdot 3 = F(1) \cdot 3 = 3,$$

$$F(4) = F(F(3) - 1) \cdot 4 = F(2) \cdot 4 = 8.$$

Ca urmare, varianta A este corectă, iar varianta C este incorectă. $F(5) = F(F(4)-1)\cdot 5 = F(7)\cdot 5 = F(F(6)-1)\cdot 7\cdot 5 = F(F(F(5)-1)\cdot 6-1)\cdot 35$. Calculul recursiv este infinit, iar calcularea oricărui număr $n, n \geq 5$ este imposibilă. Varianta B este incorectă, iar varianta D este corectă.

756. P reprezintă o permutare a numerelor de la 1 la 10. Algoritmul `Transform` transformă permutarea P într-o altă permutare Q astfel încât $Q[i] = P[Q[i]]$ pentru orice i. La fiecare construcție a lui Q, observăm că se creează o legătură între indexul i și valoarea lui $P[i]$. De asemenea, observăm că, pentru anumite poziții, elementele vor ajunge în șirul Q în exact aceeași ordine în care se aflau în P. Ele vor forma cicluri: $7 \to 8 \to 1 \to 7, 4 \to 5 \to 10 \to 9 \to 2 \to 4, 6 \to 3 \to 6$. Pentru a determina care va fi valoarea lui cnt, va trebui să calculăm cel mai mic multiplu comun al lungimilor ciclurilor.

Pentru celelalte subpuncte, vom calcula pentru fiecare valoare a lui *cnt* restul împărțirii lui *cnt* la lungimea ciclului, și determinăm astfel ordinea elementelor în șirul Q. Chiar dacă pentru o posibilă valoarea a lui $cnt = 31$ permutarea obținută ar fi aceeași cu cea pentru $cnt = 1$, algoritmul se va opri când va găsi o valoare a lui *cnt* pentru care toate elementele sunt pe poziția lor inițială, adică 30.

757. Problema se învârte în jurul divizibilității lui n și m la d. Dacă n și m sunt divizibile la d, atunci numărul de plăci necesare este $[\frac{n}{d}] \cdot [\frac{m}{d}]$. Dacă n și m nu sunt divizibile la d, atunci numărul de plăci necesare crește cu 1 pentru fiecare latură. Însă, această creștere este condiționată de divizibilitate lui n și m la d, deci varianta D nu este întotdeauna corectă. Varianta C este corectă, deoarece se ocupă de toate cazurile posibile.

758. Culoarea ultimei bile este mereu dată de numărul iniţial de bile albe. În situaţia în care cele 2 bile extrase sunt de aceeaşi culoare, se introduce o bilă neagră, deci paritatea numărului celor albe este aceeaşi. În situaţia în care bilele extrase sunt de culori diferite, se introduce o bilă albă, ceea ce face ca numărul celor albe să aibă aceeaşi paritate ca înainte de extragere. În concluzie, ultima bilă va fi albă, dacă numărul iniţial de bile albe era impar (paritatea numărului de bile albe este păstrată), respectiv neagră, în caz contrar. Variantele A şi D sunt incorecte. Varianta C este corectă, deoarece este în concordanţă cu raţionamentul de mai sus, iar varianta B este corectă, deoarece aplică acelaşi raţionament pentru un număr impar de bile albe.

759. Vom analiza scrierea în baza 2 a fiecărui număr posibil. Pentru fiecare bit în parte, putem avea fie valoarea 0, fie valoarea 1. Dacă pe poziția i avea valoarea 1, atunci al i-lea participant va scrie numărul respectiv pe bilețel. Ce se întâmplă în cazul în care n este o putere a lui 2? În acest caz, niciun participant nu va avea numărul n pe biletul lui. Dacă n este câștigător, atunci, cum nimeni nu va avea numărul n pe bilet, echipa va ști automat că numărul câștigător este n. Așadar, numărul minim de participanți va fi $\log 2x$, unde x este cel mai mic număr mai mare sau egal decât n care este putere a lui 2. Din formulele noastre, cea corespunzătoare acestui răspuns este D.

760. Fiind 120 de stânci, Broski va avea de sărit 121 de unități. Cât timp nu depășește celălalt mal, Broski va sări dublul ultimei sărituri, iar doar când nu va mai putea sări

dublul distanței, va sări o unitate. Deci, săriturile lui Broski vor fi: $1 + 2 + 4 + 8 + 16 + 32 + 1 + 2 + 4 + 8 + 16 + 1 + 2 + 4 + 8 + 1 + 2 + 4 + 1 + 2 + 1 + 1 = 121$, adică 22 de sărituri. Fiecare secvență de sărituri duble va aduce în total o deplasare de forma $2^k - 1$ unități, unde k este numărul de sărituri din secvență. Rescris, $121 = 63 + 31 + 15 + 7 + 3 + 1 + 1$.

761. Recursiv, algoritmul determină valoarea maximă din șir. Complexitatea acestuia este $O(n)$, deoarece fiecare element din șir va fi accesat o singură dată, când va ajunge la el în apelul recursiv.

762. Căutăm cel mai mic multiplu comun al numerelor din vectorul z. Toți multiplii acestui număr reprezintă o soluție validă.

763. Singurele numere care au doar 3 divizori sunt pătratele perfecte ale unor numere prime. Dintre variantele expuse, 25 este pătrat perfect al unui număr prim, 144 nu este pătrat perfect al unui număr prim, iar, deși este pătratul lui 13, 169 este mai mare decât 153.

764. Algoritmul simulează o exponențiere rapidă și calculează mereu x^n, sub formă de apeluri recursive. Complexitatea timp a algoritmului va fi $O(\log_2 n)$ datorită apelurilor recursive.

765. Strategia corectă (B): Se sortează orașele. La fiecare pas, se plasează satelitul astfel încât să acopere de la primul oraș neatins până la $x_i + L$, maximizând acoperirea spre est. Varianta A ar lăsa goluri vestice, C ignoră datele, iar D poate crește inutil numărul de sateliți.

766. Subalgoritmul verifică dacă un element i plasat pe poziția k este divizibil cu k (condiția i **MOD** $k = 0$) și evită duplicarea elementelor. Doar varianta A este corectă. Varianta B inversează condiția. Pentru $n = 3$, singura permutare validă este $[1, 2, 3]$, deci rezultatul este 1.

767. Un arbore binar complet cu n noduri are înălțimea $\lfloor \log_2(n) \rfloor$. Pentru $n = 100$, $\lfloor \log_2(100) \rfloor = 6$. Prin urmare, înălțimea arborelui binar complet cu 100 de noduri este 6.

768. Succesorul în inordine al unui nod dintr-un arbore binar de căutare este fie cel mai

stâng nod din subarborele drept, dacă acesta există sau, altfel, primul strămoș la care nodul se află în subarborele stâng. În cazul de față, nodul 28 nu are subarbore drept, însă este copilul stâng al nodului 40, deci succesorul său în inordine este nodul părinte, adică 40.

769. Subalgoritmul calculează numărul de puncte (i, j) unde cel puțin una dintre coordonate este $\pm N$, formând un "pătrat". Fiecare latură a pătratului are $2N + 1$ puncte, dar colțurile sunt numărate o singură dată. Total: $4 \times (2N + 1) - 4 = 8N$.

770. Calculăm: - A: $14_{10} = 1110_2 \Rightarrow 3$ biți setați.

- B: $28_{10} = 11100_2 \Rightarrow 2$ zerouri finale.

- $A \oplus B = 3 \oplus 2 = 1$ (deoarece $11_2 \oplus 10_2 = 01_2$).

- $1 \mod 5 = 1$.

771. Subalgoritmul calculează sumele parțiale ale cifrelor extrase de la dreapta la stânga, formează un nou număr prin concatenarea acestor sume și returnează suma cifrelor noului număr. Exemplu: Pentru $N = 1234$, sumele parțiale sunt (de la dreapta la stânga) $4, 7, 9, 10$, iar numărul format este 47910. Suma cifrelor sale este $4 + 7 + 9 + 1 + 0 = 21$.

772. Complexitatea este determinată de recurența $T(n) = 2T(n/2) + O(n \log n)$. Pentru o recurență $T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$, dacă $f(n) = \Theta(n^{\log_b a} \log^k n)$, atunci $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$. (teorema master).

Aplicând teorema master: $a = 2, b = 2, f(n) = \Theta(n \log n)$.

Deoarece $f(n) = \Theta(n^{\log_b a} \log^k n)$ cu $k = 1$, avem $T(n) = \Theta(n \log^{k+1} n) = \Theta(n(\log n)^2)$.

Variantele A și B sunt corecte deoarece $\log(n + 1) = \Theta(\log n)$, deci $n \log(n + 1) \log n = \Theta(n(\log n)^2)$.

773. Subalgoritmul aplică o căutare binară pe matricea tratată ca un vector 1D sortat. Proprietățile matricei asigură că toate elementele sunt sortate global.

774. $BD_{16} = B \times 16^1 + D \times 16^0 = 11 \times 16 + 13 = 176 + 13 = 189; 215_6 = 2 \times 6^2 + 1 \times 6^1 + 5 \times 6^0 = 2 \times 36 + 1 \times 6 + 5 = 72 + 6 + 5 = 83; 4_5 = 4, 21_3 = 2 \times 3^1 + 1 \times 3^0 = 6 + 1 = 7$.

775. Convertind fiecare caracter obținem C = $1001110_2 = 78_{10}$ O = $1111110_2 = 126_{10}$ N = $1110110_2 = 118_{10}$ U = $0111110_2 = 62_{10}$ R = $1110111_2 = 119_{10}$ S = $1101100_2 = 109_{10}$.

776. Subalgoritmul calculează numărul de moduri de a descompune k ca sumă de termeni

din mulțimea $\{1, 2, \ldots, n\}$, permise să se repete. Recurența ceFace(n, k) = ceFace$(n-1, k)$+ceFace$(n, k-n)$ corespunde cu excluderea sau includerea termenului n. Varianta A nu restricționează termenii la $\leq n$, B necesită termeni distincți, iar D introduce condiții inegale ($x_i \leq i$). Pentru $n = 3, k = 4$, rezultatul este 4 (1+1+1+1, 1+1+2, 1+3, 2+2).

777. Vectorul trebuie să fie structurat ca un arbore binar de căutare (BST) în reprezentare pre-ordine. Căutarea lui 42 începe cu rădăcina 11, apoi merge la dreapta (53), apoi la stânga (48). Opțiunea A corespunde unui BST cu 11 ca rădăcină, 53 ca fiu drept, și 48 ca fiu stâng al lui 53. Opțiunile B și C nu respectă proprietatea BST, iar D (sortat) ar genera comparații diferite.

778. Subalgoritmul implementează recurența numerelor Catalan $Q(n) = \sum_{k=1}^{n} Q(k-1)Q(n-k)$. Acestea numără:

- Arbori binari de căutare cu n noduri,

- Numărul de modalități de a paranteza $n + 1$ factori,

- Numărul de posibilități de a desena n coarde care nu se intersectează, pe un cerc care are $2 * n$ puncte.

Varianta D este incorectă (nu ține de numerele Catalan). Toate celelalte trei sunt definiții echivalente ale numerelor Catalan.

779. Într-un vector binar, subșirul strict crescător maxim apare când există cel puțin un 0 urmat de un 1 (e.g., [0, 1]), având lungimea 2. Dacă toate elementele sunt identice (doar 0 sau doar 1), lungimea maximă este 1. Variantele C și D sunt greșite deoarece numărul de 0-uri sau 1-uri nu garantează o secvență crescătoare.

780. Pentru $n = 5$, centrul este la $c = 3$. Condiția selectează:

- Zona 2: $j < 3$ și $i + j < 6$ (stânga centrului, deasupra diagonalei secundare).

- Zona 3: $j > 3$ și $i + j > 6$ (dreapta centrului, sub diagonala secundară).

Elementele din zona 2 sunt parcurse pe coloane (de la $j = 1, 2$), iar zona 3 pe linii (de la $j = 4, 5$), ambele de sus în jos. Exemplu pentru $A_{5\times5}$:

Zona 2 (coloane): $A[1][1], A[2][1], A[1][2], A[2][2], A[3][2]$.

Zona 3 (linii): $A[3][4], A[4][4], A[5][4], A[3][5], A[4][5], A[5][5]$.

Ordinea finală este cea din varianta D.

781. Algoritmul simulează o sortare, dar, de fapt, nicio schimbare nu este realizată asupra șirului x, deci întreaga funcție este un loop infinit. Algoritmul mai este cunoscut și drept "Miracle Sort", iar singura afirmație adevărată este D.

782. $f(a,b) = \begin{cases} 1 & \text{dacă } b = 0 \\ 0 & \text{dacă } b > a \\ f(a-1,b) + b \times f(a-1,b-1) & \text{altfel} \end{cases}$

unde f(a,b) reprezintă aranjamente de a luate câte b.

783. Expresia se rescrie $E(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + x(a_4 + x \cdot a_5))))$, deci sunt necesare 5 înmulțiri pentru a evalua expresia.

784. Pentru $n = 0$, algoritmul returnează 1, adică x^0. Algoritmul împarte n prin 3 şi apelează recursiv $g(x, \lfloor n/3 \rfloor)$, deci numărul de apeluri va fi proporţional cu $\log_3(n)$. Rezultatul final se compune în funcţie de n mod 3:

$$\begin{cases} p^3, & \text{dacă } n \bmod 3 = 0, \\ x \cdot p^3, & \text{dacă } n \bmod 3 = 1, \\ x \cdot x \cdot p^3, & \text{dacă } n \bmod 3 = 2. \end{cases}$$

Având $p = x^{\lfloor n/3 \rfloor}$, deducem că algoritmul calculează exact x^n (pentru toate valorile lui n) cu aproximativ $\log_3(n)$ apeluri.

785. În acest caz, atunci când $n \bmod 3 = 2$, valoarea returnată de algoritm nu mai este $x \cdot x \cdot p^3 = x^n$, ci $x + x \cdot p^3$, care (pentru $x > 1$, $n > 2$) este în general diferită și, de obicei, mai mică decât x^n. Această modificare nu afectează însă structura recursivă (împărțirea la 3 rămâne), deci numărul de apeluri rămâne aproximativ $\log_3(n)$, nu devine n^2.

786. Soluțiile generate sunt: $1116, 1112, 1114, 1166, 1162, 1164, 1126, 1122$, cea de-a 8-a fiind 1122.

787. Subalgoritmul calculează numărul de moduri de a exprima n ca sumă de numere Fibonacci distincte și neconsecutive. $\phi = \frac{1+\sqrt{5}}{2}$ este secțiunea de aur (golden ratio). Iar $\log_\phi(\sqrt{5}n)$ calculează de câte numere Fibonacci averm nevoie pentru un anume n. Spre exemplu, pentru $n = 10$, $\log_\phi(\sqrt{5}n) \simeq 4$. k adaugă 2 la acest număr, pentru a ne asigura că

avem destule numere pentru sumă. Recurența $F(n,k) = F(n-\text{Fib}(k), k-2)+F(n,k-1)$ corespunde includerii/excluderii termenului Fib(k), evitând termenii consecutivi $(k-2)$. Pentru $n = 7$, soluțiile sunt $2+5$ și $1+2+5$, deci rezultatul este 2.

788. Observăm că o regulă simplă care verifică ipoteza este **suma pozițiilor literelor în alfabet** (unde A=1, B=2, ..., Z=26). Pentru variantele date: **CASE**: $C(3)+A(1)+S(19)+E(5) = 28$, **SECA**: $S(19)+E(5)+C(3)+A(1) = 28$, **CASA**: $C(3)+A(1)+S(19)+A(1) = 24$, **CEAI**: $C(3)+E(5)+A(1)+I(9) = 18$.

789. Subalgoritmul construiește un arbore binar de căutare (BST) din șirul A. Primul element este rădăcina, elementele $\geq$ rădăcină formează subarborele stâng, iar cele $>$ formează subarborele drept. Pentru $A = [5,3,1,4,8,7,9]$, structura BST este:

Rădăcină 5 (stânga: [3,1,4], dreapta: [8,7,9]).

Frunzele sunt nodurile 1, 4, 7, 9 → 4 frunze. Varianta B găsește aceleași numere (pozițiile 3,4 și 6,7 din șir).

790. Dând factor comun succesiv pe x și x^3 obținem $P(x) = x\cdot\big(x^3\cdot(a_5\cdot x+a_4)+a_2\big)+a_0$ Total: 5 înmulțiri. (două pentru x^3, iar restul pentru fiecare semn de înmulțire din expresia dată)

791. Știm că T[1][1] = 1 Pentru Rândul 1: T[1][1] = 1 (dat) T[1][2] = 0 (deoarece 1+2=3 este impar și nu este i=j) T[1][3] = 0 (deoarece 1+3=4 este par, XOR al valorilor precedente) Pentru Rândul 2: T[2][1] = 2 (T[1][1] * 2 deoarece 2+1=3 este impar) T[2][2] = 4 (deoarece i=j=2, deci 2×2=4) T[2][3] = 0 (deoarece 2+3=5 este impar, T[1][3] * 2 = 0) Pentru Rândul 3: T[2][1] = 2 (T[2][1] * 2 deoarece 3+1=4 este par) T[3][2] = 8 (T[2][2] * 2 deoarece 3+2=5 este impar) T[3][3] = 9 (deoarece i=j=3, deci 3×3=9) Prin urmare, T[3][3] + T[2][3] = 9 + 0 = 9

Răspunsul este 9.

792. Fiecare cifră a lui $N = 475$ este convertită în binar: - 4 → 100 (1 bit de 1) - 7 → 111 (3 biți de 1) - 5 → 101 (2 biți de 1) După care se adună numărul de biți de 1 la o sumă finală: $1+3+2 = 6$.

793. Complexitate timp: $3\log_2(n)\cdot\log_3(n)+4\log_3(n)+1$ Termenul dominant al funcției care descrie complexitatea algoritmului este $\log_2(n)\cdot\log_3(n)$. Vom folosi proprietatea

de schimbare a bazei a funcției logaritm: $\log_a(X) = \log_a b \cdot \log_b(X)$ Astfel, obținem: $\log_2(n) \cdot \log_3(n) = \log_2(3) \cdot \log_3^2(n) \in O(\log_3^2(n))$ Similar: $\log_2(n) \cdot \log_3(n) = \log_3(2) \cdot \log_2^2(n) \in O(\log_2^2(n))$

Pentru $n > 3$ are loc: $\log_2(n) \cdot \log_3(n) > \log_2(n)$, deci $T(n) \notin O(\log_2(n))$ Folosind inegalitatea $\log_a(n) < n$ avem: $\log_2(\log_3(n)) < \log_3(n) < \log_2(n) \cdot \log_3(n), \forall n > 2$, deci $T(n) \notin O(\log_2(\log_3(n)))$

794. Subalgoritmul calculează numărul de partiții multiplicative ordonate ale lui n cu factori $\geq d$. Pentru $d = 2$, acesta numără toate descompunerile ordonate ale lui n în produse de numere ≥ 2, unde fiecare factor ulterior este $\geq$ precedentul. Exemplu pentru $n = 8$: 8, 2×4, 4×2, $2 \times 2 \times 2 \rightarrow 4$ moduri.

795. Expresia dată este: NOT(A OR B) OR NOT(C AND NOT A)

Simplificăm folosind legile lui De Morgan: (NOT A AND NOT B) OR (NOT C OR A)

796. Considerând fiecare pereche ca o singură entitate, avem 3 entități care pot fi aranjate în cerc în $(3-1)! = 2!$ moduri. Fiecare pereche poate fi așezată în 2 moduri (prieten 1: stânga sau dreapta). Total: $2! \times 2^3 = 2 \times 8 = 16$.

797. Subalgoritmul efectuează o căutare binară pentru a găsi cea mai mare putere a lui 2 (2^k) care este $\leq n$. Pentru $n = 25$, cea mai mare astfel de putere este 16 (2^4).

798.

- $14 \div (1 + 1 + 4) = 14 \div 6 = 2$ rest $2 \rightarrow$ nu este divizibil.
- $22 \div (2 + 2 + 2) = 22 \div 6 = 3$ rest $4 \rightarrow$ nu este divizibil.
- $30 \div (3 + 3 + 0) = 30 \div 6 = 5$ rest $0 \rightarrow$ este divizibil.
- $41 \div (4 + 4 + 1) = 41 \div 9 = 4$ rest $5 \rightarrow$ nu este divizibil.

799. Strategia corectă (B): Cercul este "tăiat" într-o linie prin punctul cu cea mai mare distanță între două puncte consecutive. Se plasează arce de la stânga la dreapta, apoi se verifică suprapunerea la capete datorită circularității. Varianta A eșuează din cauza circularității (arcul final poate să nu se conecteze cu primul). Pentru $N = 8$ puncte echidistante și $L = 180°$, soluția optimă este 2 arce, obținută prin strategia B.

800. Calculăm $2^7 = 128$ (binar: `10000000`). Adunăm $128 + 128 = 256$, care în 8 biți devine `00000000`. Scădem 1: `00000000` - 1 = `11111111` (complementul lui 2 pentru -1).

- Fără sem: `11111111` = 255.

- Cu semn: `11111111` = -1.

801. Pe intervalul $[0, 100)$ sunt 14 multipli de 7, 19 numere care conțin cifra 7 și 3 numere care conțin atât cifra 7, dar sunt și divizibile cu 7, deci pe intervalul $[0, 100)$, se strigă "Bolț!" de exact 30 de ori. Analizând la fel intervalele $[100, 200)$ și $[200, 300)$, obținem că al 100-lea element pentru care se strigă "Bolț!" este 336.

802. Notăm cu M mulțimea multiplilor de 7 mai mici decât 1000, cu D mulțimea cifrelor care conțin cifra 7 până la 1000. Din cele 1000 de numere naturale, avem $|M| = 142, |D| = 271$ și $|M \cap D| = 39$, unde $|M|$ reprezintă cardinalul mulțimii M. Deci, $|S| = |M \cup D| = 142 + 271 - 39 = 374$. Observăm că Andrei rostește numerele de pe pozițiile impare (1, 3, 5, ...), iar Maria pe cele de pe pozițiile pare (2, 4, 6, ...), iar un „Bolț!" apare exact la numerele $n \in S$. Avem 221 de numere impare în S și 153 pare, lucru care rezultă din forma generală a multiplilor, respectiv a numerelor care conțin cifra 7. Așadar, Andrei a spus „Bolț!" de 221 ori, iar Maria de 153 ori, deci Andrei are cu $221 - 153 = 68$ mai multe strigări.

803. Algoritmul implementează Teorema Cayley - Hamilton pe matrici de dimensiuni 2×2: $A^2 - Tr(A) \cdot A + \det A \cdot I_2 = O_2$, deci la finalul execuției algoritmului, indiferent de valorile din a, în r toate elementele vor fi nule (se poate verifica și luând $a \leftarrow \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ și simulând algoritmul pas cu pas).

804. Componentele tare conexe sunt $\{1, 2, 3\}, \{4\}$ și $\{5\}$, deci varianta corectă este B.

805. `AlexB(k)` permută caracterele cuvântului s, astfel că, atunci când o permutare ajunge la dimensiunea n, este afișată. Având în vedere definiția anagramelor, varianta corectă de răspuns este C.

806. Algoritmul utilizează exponențierea rapidă pentru a calcula puterea matricei, având complexitate $O(\log n)$. Varianta A este incorectă deoarece complexitatea este logaritmică, nu liniară. Varianta B este corectă deoarece fiecare apel recursiv împarte exponentul la 2, rezultând $O(\log k)$ înmulțiri. Varianta C este corectă: pentru $n = 6$, exponentul este

$6-2=4$. Varianta D este corectă: algoritmul tratând cazurile de bază $n=1$ și $n=2$ prin returnarea valorii 1, iar pentru $n \geq 3$ calculul matriceal este corect conform proprietăților șirului lui Fibonacci.

807. $97_{10} = 1100001_2$, deci sunt necesari 7 biți pentru reprezentarea fără semn. Dacă reprezentarea se face cu semn, mai avem nevoie de un bit rezervat semnului, deci minimul va fi 8 biți.

808. `ceFace1(v, n)` ordonează descrescător șirul v după numărul de divizori al fiecărui element, iar `ceFace2(x)` returnează numărul total de divizori ai numărului x.

809. Proprietatea implementată de algoritm se referă la "Numărul magic" al lui Von Neumann. Practic, repetând acest procedeu de scădere a formei cu cifrele sortate crescător din cea cu cifrele sortate descrescător, indiferent de n, după cel mult 7 pași, vom ajunge la numărul 6174, care rămâne constant.

810. Suma este $F(n) = \sum_{k=1}^{n} \frac{k^2}{k!}$. Observăm că: $\frac{k^2}{k!} = \frac{k(k-1)+k}{k!} = \frac{k(k-1)}{k!} + \frac{k}{k!} = \frac{1}{(k-2)!} + \frac{1}{(k-1)!}$. Pentru $k=1: \frac{1}{0!} = 1$. Pentru $k=2: \frac{1}{0!} + \frac{1}{1!} = 1+1 = 2$. Pentru $k \geq 3$: termenii se anulează reciproc, deci suma va fi mereu 2, indiferent de valoarea lui $n \geq 2$.

811. Când n este par, se efectuează aproximativ $\log n$ apeluri recursive, iar când n este impar, aproximativ $\log(\log n)$ apeluri, deci complexitatea totală va fi $O(\log n)$.

812.

Avem 4 locuitori (A, B, C, D) cu exact doi oracoli (O) și doi haos (H). Fiecare răspunde despre viitor astfel:

- Un **oracol** spune întotdeauna adevărul.
- Un **haos** minte întotdeauna.

Întrebarea Q1 adresată lui A: „Dacă mâine aș întreba pe C dacă D este haos, va răspunde 'Da'?"

Notăm: T_1 = propoziția "C va răspunde "Da" la întrebarea „D e haos?"". Observăm că, analizând comportamentul lui C:

- Dacă C este oracol, răspunde veridic: "Da" dacă $D = H$, "Nu" dacă $D = O$.
- Dacă C este haos, răspunde opusul: "Da" dacă $D = O$, "Nu" dacă $D = H$.

Deci, T_1 = "Da" $\iff$ (C și D sunt de tipuri diferite); în caz contrar, T_1 este "Nu". Apoi, A va răspunde:

$$R_1 = \begin{cases} T_1, & \text{dacă } A = O, \\ \text{opusul lui } T_1, & \text{dacă } A = H. \end{cases}$$

Întrebarea Q2 adresată lui B: „Dacă mâine aș întreba pe D dacă cel puțin doi dintre voi sunteți oracoli, va răspunde 'Nu'?". Știm că, fiind exact doi oracoli, afirmația „cel puțin doi sunteți oracoli" este adevărată. Dacă D este:

- O: va spune adevărul, adică „Da" la întrebarea directă.
- H: va minți și va răspunde „Nu".

Notăm:

$$T_2 = \text{propoziția "}D\text{ va răspunde "Nu" la întrebarea directă"} = \begin{cases} \text{"Nu"} & \text{dacă } D = O, \\ \text{"Da"} & \text{dacă } D = H. \end{cases}$$

Iar B răspunde:

$$R_2 = \begin{cases} T_2, & \text{dacă } B = O, \\ \text{opusul lui } T_2, & \text{dacă } B = H. \end{cases}$$

Rezumatul cazurilor posibile (cu exact doi oracoli):

Caz	A	B	C	D	R_1	R_2
1	O	O	H	H	Nu	Da
2	O	H	O	H	Da	Nu
3	O	H	H	O	Da	Da
4	H	O	O	H	Nu	Da
5	H	O	H	O	Nu	Nu
6	H	H	O	O	Da	Da

Observații:

- Din R_1 singur (fie "Da", fie "Nu") se rămân 3 cazuri, deci răspunsul lui A este ambiguu.
- Pentru ca exploratorul să deducă tipurile după R_2, combinația (R_1, R_2) trebuie să corespundă unui singur caz.

Analizând:

- Dacă R_1 = "Da", cazurile posibile sunt 2, 3, 6. Observăm că singur **cazul 2** dă R_2 = "Nu".
- Dacă R_1 = "Nu", cazurile posibile sunt 1, 4, 5. Singur **cazul 5** dă R_2 = "Nu".

Deci am obține, din punct de vedere strict logic, două perechi unice:

$$\text{(Da, Nu)} \quad \text{și} \quad \text{(Nu, Nu)}.$$

Concluzie: Deși din punct de vedere matematic atât (Da, Nu) (cazul 2) cât și (Nu, Nu) (cazul 5) determină o configurație unică, cerința suplimentară ca răspunsul lui A să fie *ambiguu* (adică să nu indice de la sine tipul său) favorizează situația în care A este oracol. Prin urmare, varianta corectă este: $R_1 = \text{Da}, \quad R_2 = \text{Nu}$.

813. Folosim principiul includerii și excluderii:

1. Scădem numerele divizibile cu 3, 5 sau 7: $\lfloor \frac{N}{3} \rfloor + \lfloor \frac{N}{5} \rfloor + \lfloor \frac{N}{7} \rfloor$.
2. Adunăm intersecțiile pe perechi (divizibile cu 15, 21, 35): $\lfloor \frac{N}{15} \rfloor + \lfloor \frac{N}{21} \rfloor + \lfloor \frac{N}{35} \rfloor$.
3. Scădem intersecția tuturor trei (divizibile cu 105): $\lfloor \frac{N}{105} \rfloor$.

Formula corectă este $N - (A + B + C) + (AB + AC + BC) - ABC$.

814. Subalgoritmul calculează numărul de submulțimi nevide ale lui A în care fiecare element este divizibil cu poziția sa în submulțime. - Condiția $A[p]$ **MOD** $(k + 1) = 0$ verifică dacă elementul curent poate fi al $(k + 1)$-lea element al submulțimii (divizibil cu

$k+1$).

815. Subalgoritmul calculează aria unui poligon cu coordonatele x_i, y_i: $T_i = \frac{1}{2} \cdot \begin{vmatrix} 0 & 0 & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix} = \frac{1}{2} \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$. Formula finală devine $A = \sum_{i=1}^{n} T_i = \frac{1}{2} \cdot \sum_{i=1}^{n} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$.

816. **Varianta B**: Fiecare cifră în baza 4 corespunde la 2 biți. Pentru k biți în baza 2, numărul de cifre în baza 4 este $\lceil k/2 \rceil$, care este $\leq \lfloor k/2 \rfloor + 1$. Exemplu: $7_{10} = 111_2$ (3 biți) $\rightarrow 13_4$ (2 cifre; $\lfloor 3/2 \rfloor + 1 = 2$).

Varianta D: Valoarea unei cifre în baza 4 este mai mare sau egală cu numărul corespunzător de biți de 1 din baza 2 (0 -¿ 0 biți, 1 -¿ 1 bit, 2 -¿ 1 bit, 3 -¿ 2 biți), deci suma cifrelor va fi mai mare sau egală cu numărul de biți în baza 2. **Varianta A** este falsă: $10_{10} = 22_4$ (palindrom) dar 1010_2 nu este palindrom.

Varianta C este falsă: $15_{10} = 33_4$ (ultima cifră 3).

817. **Varianta A**: Algoritmul folosește exponențierea rapidă, reducând b la jumătate la fiecare pas $\Rightarrow O(\log b)$. **Varianta B**: $3^5 \mod 7 = 243 \mod 7 = 5$. **Varianta C** este falsă, pentru $b = 3 = 111_2$ avem 2 apeluri recursive, nu 3. **Varianta D** este falsă. $m = 1$ implică $x \mod 1 = 0, \forall x \in \mathrm{Z}$, dar pentru $b = 0$, funcția returnează 1.

818. **Varianta A**: Corect. Calcul: 7 **MOD** $3 = 1 \neq 0$, $7 + 3 = 10$, 10 **MOD** $2 = 0$. Aplică prima ramură: $7*(3$ **DIV** $2)+(7$ **MOD** $3) = 7*1+1 = 8$. **Varianta B**: Dacă $a+b$ este impar, a doua ramură poate returna impar (ex: $a = 10, b = 3$: $3*2-3$ **MOD** $5 = 3$). **Varianta C**: Pentru $a = 10, b = 4$: 10 **MOD** $4 = 2 \neq 0$, $10 + 4 = 14$, 14 **MOD** $2 = 0$. Prima ramură: $10 * (4$ **DIV** $2) + (10$ **MOD** $3) = 10 * 2 + 1 = 21$. **Varianta D**: a **MOD** $3 = 0$ nu implică divizibilitatea cu 3 (ex: $a = 3, b = 2$: prima ramură $3*1+0 = 3$, care e divizibil, dar $a = 6, b = 5$: $6 * 2 + 0 = 12$, divizibil cu 3; dar nu este "ori de câte ori").

819. Algoritmul calculează numărul de modalități de a partiționa n în k numere naturale, fiecare fiind cel puțin 2. Pentru ($k = 1$), funcția returnează 1 dacă $n \geq 2$, indicând o partiție validă cu un singur termen.. Pentru ($k > 1$), funcția iterează peste posibilele

divizări i (începând de la 2). Pentru fiecare i, verifică:

- Partea Stângă $(i-1)$: Trebuie să fie validă pentru $k=1$, adică $i-1 \geq 2$.
- Partea Dreaptă $(n-i)$: Trebuie să fie partiționată în $k-1$ termeni, fiecare ≥ 2.

820. Subalgoritmul identifică elementul majoritar, parcurgând șirul o singura dată ($O(n)$ timp), fără a folosi structuri de date auxiliare ($O(1)$ spațiu).

821. Suma este, de fapt, $S(x) = \sum_{k=1}^{x} \frac{k}{(k+1)!}$, care poate fi simplificată observând că fiecare termen se scrie ca: $\frac{k}{(k+1)!} = \frac{1}{k!} - \frac{1}{(k+1)!}$. Suma devine: $\left(\frac{1}{1!} - \frac{1}{2!}\right) + \left(\frac{1}{2!} - \frac{1}{3!}\right) + \cdots + \left(\frac{1}{n!} - \frac{1}{(n+1)!}\right)$. Toți termenii intermediari se reduc, rămânând $1 - \frac{1}{(n+1)!}$

822. Algoritmul compară frecvențele caracterelor (diferențele individuale ale frecvențelor sunt zero pentru fiecare caracter), verificând anagramele.

823. Varianta A calculează corect rezultatul, dar fiecare apel va calcula atât `F(n-1)`, cât și `F(n-3)`, deci complexitatea nu este $O(n)$. Varianta B folosește un vector circular, actualizând valorile pentru fiecare număr procesat, având complexitatea $O(n)$. Varianta C calculează corect rezultatul, dar complexitate finală este $O(n)$. Varianta D încearcă sa aproximeze rezultatul prin calcularea unei rădăcini de ordinul 3 a ecuației $x^3 - 3x^2 - 2 = 0$. Rezultatul final ar trebui sa ia în calcul fiecare rădăcină, până la n, nu doar prima.

824. In secvența B, când inserăm 12 înaintea lui 13, 12 ar deveni fiul drept al lui 11 (deoarece $12 > 11$). Apoi, când încercăm să inserăm 13, acesta ar trebui să devină fiul drept al lui 12, dar în arborele țintă, 13 este părintele lui 12, nu invers.

825. Varianta B este imposibilă: primele patru elemente (3, 6, 4, 8) nu sunt sortate crescător. Algoritmul de sortare prin inserție garantează că după fiecare iterație i, primele $i+1$ elemente sunt sortate. Opțiunile A, C și D au primele patru elemente sortate corect.

826. Varianta A este singura posibilă deoarece algoritmul compară mai întâi cu `"plop"` și sare cu +2 la `"stejar"`, apoi datorită cazului special pentru `"stejar"` sare doar cu +1 ajungând la `"tei"`, în timp ce celelalte secvențe sunt imposibile întrucât saltul de +2 după `"plop"` nu permite accesarea cuvântului `"salcie"`.

827. Algoritmul ceFace(n, a) verifică două condiții: (1) $gcd(a, n) = 1$ (a și n sunt coprime) și (2) $a^{n-1} \equiv 1 \pmod{n}$ (testul micii teoreme a lui Fermat) - dacă ambele sunt

îndeplinite, returnează true, reprezentând un test de pseudoprimalitate. Pentru $n = 341$ și $a = 2$ avem $341 = 11 \times 31$, $gcd(2, 341) = 1$ și $2^{340} \equiv 1 \pmod{341}$ (prin calculul $2^{10} \equiv 1 \pmod{341}$ și apoi $(2^{10})^{34} \equiv 1^{34} \equiv 1 \pmod{341}$), deci 341 este pseudoprim în baza 2, însă pentru $n = 91$ și $a = 3$ deși $91 = 7 \times 13$ și $gcd(3, 91) = 1$, calculul arată că $3^{90} \not\equiv 1 \pmod{91}$, deci 91 nu este pseudoprim în baza 3.

828. **Linia 1** necesită condiția de terminare `i == n` (când am completat toate pozițiile). **Linia 2** trebuie să anuleze swap-ul inițial ⇒ `swap(arr[i], arr[j])` este varianta corectă (echivalentă cu `swap(arr[j], arr[i])`).

829. Sortarea prin selecție are complexitate $O(n^2)$. Numărul de comparații este $\frac{n(n-1)}{2} \approx \frac{n^2}{2}$. Pentru k: $\frac{k^2}{2} = 5000 \Rightarrow k^2 = 10000$. Pentru $2k$: $\frac{(2k)^2}{2} = \frac{4k^2}{2} = 2k^2 = 20000$. Varianta C reflectă corect creșterea pătratică. Varianta D este incorectă deoarece ignoră împărțirea cu 2 din formulă.

830. Doar **D** este incorectă (prioritate operator: **AND** > **OR**, deci evaluează ca `n MOD 2 == 0 OR (n MOD 3 == 0 AND n >= 0)`).

831. **A. Adevărat** Algoritmul numără toate numerele naturale i mai mici decât n pentru care $\gcd(i, n) = 1$. Aceasta este definiția funcției Euler $\varphi(n)$.

B. Fals Iterația merge până la n, dar n nu este inclus în numărare deoarece $\gcd(n, n) = n \neq 1$. Singura excepție este cazul $n = 1$, unde se returnează 1.

C. Fals Pentru $n > 1$, se returnează un număr mai mic decât n, dar pentru $n = 1$, rezultatul este 1, care nu este strict mai mic decât 1. Prin urmare, afirmația nu este întotdeauna adevărată.

D. Adevărat Dacă n este prim, toate numerele $1, 2, \ldots, n-1$ sunt coprime cu n, deci algoritmul returnează corect $\varphi(n) = n - 1$. Algoritmul verifică fiecare număr și numără corect cele coprime.

832. (A) Nu permite identificarea bilei diferite independent de natura dezechilibrului. Strategia cu 3 grupe de 3 bile necesită urmărirea direcției dezechilibrului. **Falsă**.

(B) Compararea 4 vs 4 este deficientă: în caz de dezechilibru, cele 4 bile suspecte nu pot fi identificate în 3 cântăriri. **Falsă**.

(C) Transpunerea bilelor între talere și analiza modificărilor de direcție permite deducerea

naturii diferenței. **Adevărată**.

(D) Strategiile de paritate nu necesită obligatoriu număr par de bile (e.g., 3 vs 3 este valid). **Falsă**.

833. Considerăm că la fiecare mutare jucătorul alege cel mai mic număr valid posibil. Secvența primelor 8 mutări poate fi: 1(Alex), 2(Diana), 4(Alex), 5(Diana), 7(Alex), 8(Diana), 10(Alex), 11(Diana). Coada șarpelui este astfel 11. Pentru a 5-a mutare a lui Alex, el trebuie să aleagă $x > 11$ astfel încât $11 + x$ să fie multiplu de 3. Dintre variantele de răspuns, doar 16 este corectă, deoarece $11 + 16 = 27$ `MOD` $3 = 0$.

834. La început, Alex pune 1. Diana trebuie să aleagă un număr $n > 1$ astfel încât $n + 1$ să fie multiplu de 4. Astfel, n poate fi $3, 7, 11, 15, \ldots$ (numere de forma $4k - 1$). Presupunem, spre exemplu, că Diana pune 3. Atunci Alex va alege un număr $m > 3$ cu $m + 3$ multiplu de 4, adică $m = 5, 9, 13, 17, \ldots$ (forma $4k - 3$). Repetând raționamentul, observăm că, pentru orice număr x deja pus, există întotdeauna un y mai mare decât x de forma $4k - x$, suficient de mare încât $y + x$ să fie multiplu de 4 și $y > x$. Astfel, există un șir nelimitat de mutări valide, iar jocul nu se blochează niciodată.

835. Se realizează un singur apel recursiv: `afiș(1000)` $\rightarrow$ 1000 $\rightarrow$ `afiș(4000)` $\rightarrow$ 4000 $\rightarrow$ `afiș(16000)`. La întoarcerea pe stivă, se va mai afișa o dată 4000, respectiv 1000, de unde B este varianta corectă.

16

Propunători

1. Mihai Gheorgheș
2. Mihai Gheorgheș
3. Mihai Gheorgheș
4. Mihai Gheorgheș
5. Mihai Gheorgheș
6. Mihai Gheorgheș
7. Luca Tudor
8. Luca Tudor
9. Luca Tudor
10. Luca Tudor
11. Rareș Cotoi
12. Rareș Cotoi
13. Rareș Cotoi
14. Rareș Cotoi
15. Mihai Gheorgheș
16. Mihai Gheorgheș
17. Mihai Gheorgheș
18. Mihai Gheorgheș
19. Mihai Gheorgheș
20. Mihai Gheorgheș
21. Rareș Cotoi
22. Rareș Cotoi
23. Rareș Cotoi
24. Rareș Cotoi
25. Rareș Cotoi
26. Rareș Cotoi
27. Rareș Cotoi
28. Rareș Cotoi
29. Rareș Cotoi
30. Rareș Cotoi
31. Rareș Cotoi
32. Rareș Cotoi
33. Rareș Cotoi
34. Rareș Cotoi
35. Rareș Cotoi
36. Rareș Cotoi
37. Rareș Cotoi
38. Rareș Cotoi
39. Rareș Cotoi
40. Rareș Cotoi
41. Rareș Cotoi
42. Rareș Cotoi
43. Rareș Cotoi
44. Rareș Cotoi
45. Daniel Pop
46. Mara Ielciu
47. Mara Ielciu
48. Mara Ielciu
49. Mara Ielciu
50. Mara Ielciu
51. Mara Ielciu
52. Mara Ielciu
53. Mara Ielciu
54. Mara Ielciu
55. Mara Ielciu
56. Mara Ielciu
57. Mara Ielciu
58. Mara Ielciu
59. Mara Ielciu
60. Mara Ielciu
61. Mara Ielciu
62. Mara Ielciu
63. Mihai Gheorgheș
64. Mihai Gheorgheș
65. Luca Tudor
66. Mihai Gheorgheș
67. Mihai Gheorgheș
68. Mihai Gheorgheș
69. Mihai Gheorgheș
70. Mihai Gheorgheș
71. Rareș Cotoi
72. Mihai Gheorgheș
73. Mihai Gheorgheș
74. Mihai Gheorgheș
75. Rareș Cotoi
76. Rareș Cotoi
77. Mihai Gheorgheș
78. Mihai Gheorgheș
79. Mihai Gheorgheș
80. Mihai Gheorgheș
81. Mihai Gheorgheș
82. Mihai Gheorgheș
83. Luca Tudor
84. Rareș Cotoi
85. Mihai Gheorgheș
86. Mihai Gheorgheș
87. Mihai Gheorgheș
88. Mihai Gheorgheș
89. Mihai Gheorgheș
90. Mihai Gheorgheș
91. Mihai Gheorgheș
92. Daniel Pop
93. Mihai Gheorgheș
94. Mihai Gheorgheș
95. Daniel Pop
96. Luca Tudor
97. Luca Tudor
98. Luca Tudor
99. Luca Tudor
100. Luca Tudor
101. Mihai Gheorgheș
102. Mihai Gheorgheș
103. Mihai Gheorgheș
104. Mihai Gheorgheș
105. Luca Tudor
106. Luca Tudor
107. Luca Tudor
108. Luca Tudor
109. Luca Tudor
110. Luca Tudor
111. Luca Tudor
112. Mihai Gheorgheș
113. Mihai Gheorgheș
114. Mihai Gheorgheș
115. Mihai Gheorgheș
116. Mihai Gheorgheș
117. Luca Tudor
118. Daniel Pop
119. Luca Tudor
120. Mihai Gheorgheș
121. Luca Tudor
122. Luca Tudor
123. Mihai Gheorgheș
124. Luca Tudor
125. Luca Tudor
126. Mihai Gheorgheș
127. Mihai Gheorgheș
128. Rareș Cotoi
129. Mihai Gheorgheș
130. Mihai Gheorgheș
131. Mihai Gheorgheș
132. Mihai Gheorgheș
133. Mihai Gheorgheș
134. Rareș Cotoi
135. Paul Dobrescu
136. Paul Dobrescu
137. Paul Dobrescu
138. Paul Dobrescu
139. Paul Dobrescu
140. Paul Dobrescu
141. Paul Dobrescu
142. Paul Dobrescu
143. Rareș Cotoi
144. Paul Dobrescu
145. Paul Dobrescu
146. Paul Dobrescu
147. Paul Dobrescu
148. Paul Dobrescu
149. Paul Dobrescu
150. Paul Dobrescu
151. Cristian Crețu
152. Cristian Crețu
153. Luca Tudor
154. Cristian Crețu
155. Luca Tudor
156. Luca Tudor
157. Luca Tudor
158. Luca Tudor
159. Luca Tudor
160. Luca Tudor
161. Luca Tudor
162. Luca Tudor
163. Mihai Gheorgheș
164. Cristian Crețu
165. Cristian Crețu
166. Luca Tudor
167. Luca Tudor
168. Mihai Gheorgheș
169. Luca Tudor
170. Mihai Gheorgheș
171. Mihai Gheorgheș
172. Mihai Gheorgheș
173. Mihai Gheorgheș
174. Paul Dobrescu
175. Mihai Gheorgheș
176. Luca Tudor
177. Luca Tudor
178. Luca Tudor
179. Paul Dobrescu
180. Paul Dobrescu
181. Paul Dobrescu
182. Paul Dobrescu
183. Cristian Crețu
184. Daniel Pop

185. Daniel Pop
186. Luca Tudor
187. Daniel Pop
188. Luca Tudor
189. Luca Tudor
190. Mihai Gheorgheș
191. Mihai Gheorgheș
192. Luca Tudor
193. Mihai Gheorgheș
194. Mihai Gheorgheș
195. Mihai Gheorgheș
196. Mihai Gheorgheș
197. Mihai Gheorgheș
198. Mihai Gheorgheș
199. Mihai Gheorgheș
200. Mihai Gheorgheș
201. Paul Dobrescu
202. Paul Dobrescu
203. Paul Dobrescu
204. Paul Dobrescu
205. Cristian Crețu
206. Cristian Crețu
207. Mircea Măierean
208. Cristian Crețu
209. Cristian Crețu
210. Cristian Crețu
211. Mircea Măierean
212. Mircea Măierean
213. Mircea Măierean
214. Mircea Măierean
215. Mircea Măierean
216. Cristian Crețu
217. Cristian Crețu
218. Mara Ielciu
219. Mara Ielciu
220. Mara Ielciu
221. Mara Ielciu
222. Mara Ielciu
223. Mara Ielciu
224. Rareș Cotoi
225. Mihai Gheorgheș
226. Mihai Gheorgheș
227. Mihai Gheorgheș
228. Luca Tudor
229. Luca Tudor
230. Mihai Gheorgheș
231. Luca Tudor
232. Luca Tudor
233. Luca Tudor
234. Mihai Gheorgheș
235. Mihai Gheorgheș
236. Mihai Gheorgheș
237. Luca Crețu
238. Luca Crețu
239. Luca Crețu
240. Luca Tudor
241. Luca Tudor
242. Luca Tudor
243. Luca Tudor
244. Luca Tudor
245. Luca Tudor
246. Luca Tudor
247. Luca Tudor
248. Luca Tudor
249. Luca Tudor
250. Luca Tudor
251. Luca Tudor
252. Luca Crețu
253. Luca Crețu
254. Luca Crețu
255. Luca Crețu
256. Mihai Gheorgheș
258. Mihai Gheorgheș
259. Rareș Cotoi
260. Luca Tudor
261. Luca Tudor
262. Rareș Cotoi
263. Luca Tudor
264. Luca Tudor
265. Luca Tudor
266. Luca Tudor
267. Luca Tudor
268. Daniel Pop
269. Luca Crețu
270. Luca Crețu
271. Luca Crețu
272-595. Comisie
596. Cristian Crețu
597. Rareș Cotoi
598. Rareș Cotoi
599. Cristian Crețu
600. Cristian Crețu
601. Cristian Crețu
602. Cristian Crețu
603. Rareș Cotoi
604. Cristian Crețu
605. Rareș Cotoi
606. Cristian Crețu
607. Rareș Cotoi
608. Cristian Crețu
609. Rareș Cotoi
610. Cristian Crețu
611. Cristian Crețu
612. Cristian Crețu
613. Cristian Crețu
614. Horea Mureșan
615. Cristian Crețu
616. Cristian Crețu
617. Rareș Cotoi
618. Cristian Crețu
619. Cristian Crețu
620. Mihai Gheorgheș
621. Mihai Gheorgheș
622. Mihai Gheorgheș
623. Mihai Gheorgheș
624. Mihai Gheorgheș
625. Mihai Gheorgheș
626. Mihai Gheorgheș
627. Mihai Gheorgheș
628. Mihai Gheorgheș
629. Mihai Gheorgheș
630. Mihai Gheorgheș
631. Mihai Gheorgheș
632. Mihai Gheorgheș
633. Mihai Gheorgheș
635. Mihai Gheorgheș
636. Mihai Gheorgheș
637. Mihai Gheorgheș
638. Mihai Gheorgheș
639. Mihai Gheorgheș
640. Mihai Gheorgheș
641. Mihai Gheorgheș
642. Mihai Gheorgheș
643. Mihai Gheorgheș
644. Paul Dobrescu
645. Cristian Crețu
646. Paul Dobrescu
647. Paul Dobrescu
648. Paul Dobrescu
649. Paul Dobrescu
650. Paul Dobrescu
651. Paul Dobrescu
652. Paul Dobrescu
653. Paul Dobrescu
654. Paul Dobrescu
655. Paul Dobrescu
656. Paul Dobrescu
657. Paul Dobrescu
658. Paul Dobrescu
659. Paul Dobrescu
660. Paul Dobrescu
661. Paul Dobrescu
662. Paul Dobrescu
663. Paul Dobrescu
664. Paul Dobrescu
665. Paul Dobrescu
666. Paul Dobrescu
667. Paul Dobrescu
668. Luca Tudor
669. Luca Tudor
670. Luca Tudor
671. Luca Tudor
672. Luca Tudor
673. Luca Tudor
674. Rareș Cotoi
675. Luca Tudor
676. Luca Tudor
677. Luca Tudor
678. Mara Ielciu
679. Luca Tudor
680. Luca Tudor
681. Luca Tudor
682. Rareș Cotoi
683. Mircea Măierean
684. Mara Ielciu
685. Luca Tudor
686. Luca Tudor
687. Luca Tudor
688. Luca Tudor
689. Luca Tudor
690. Luca Tudor
691. Luca Tudor
692. Mara Ielciu
693. Luca Crețu
694. Mara Ielciu
695. Mara Ielciu
696. Mara Ielciu
697. Mara Ielciu
698. Luca Crețu
699. Luca Crețu
700. Mara Ielciu
701. Luca Crețu
702. Mara Ielciu
703. Mara Ielciu
704. Mircea Măierean
705. Mircea Măierean
706. Mircea Măierean
707. Mircea Măierean
708. Luca Crețu
709. Mircea Măierean
710. Mircea Măierean
711. Mircea Măierean
712. Mircea Măierean
713. Luca Crețu
714. Luca Crețu
715. Luca Crețu
716. Mara Ielciu
717. Luca Crețu
718. Mara Ielciu
719. Mara Ielciu
720. Mara Ielciu
721. Mara Ielciu
722. Luca Crețu
723. Mara Ielciu
724. Luca Crețu
725. Mara Ielciu

726. Mara Ielciu
727. Luca Crețu
728. Mircea Măierean
729. Luca Crețu
730. Mircea Măierean
731. Mircea Măierean
732. Mircea Măierean
733. Luca Crețu
734. Luca Crețu
735. Luca Crețu
736. Mircea Măierean
737. Mircea Măierean
738. Mircea Măierean
739. Mircea Măierean
740. Mara Ielciu
741. Mara Ielciu
742. Mara Ielciu
743. Mara Ielciu
744. Mara Ielciu
745. Luca Crețu
746. Luca Crețu
747. Luca Crețu
748. Luca Crețu
749. Luca Crețu
750. Mara Ielciu
751. Mircea Măierean
752. Mara Ielciu
753. Mara Ielciu
754. Luca Crețu
755. Luca Crețu
756. Mircea Măierean
757. Mircea Măierean
758. Luca Crețu
759. Mircea Măierean
760. Mircea Măierean
761. Mircea Măierean
762. Mircea Măierean
763. Mircea Măierean
764. Rareș Cotoi
765. Cristian Crețu
766. Cristian Crețu
767. Rareș Cotoi
768. Rareș Cotoi
769. Cristian Crețu
770. Cristian Crețu
771. Cristian Crețu
772. Cristian Crețu
773. Cristian Crețu
774. Rareș Cotoi
775. Cristian Crețu
776. Cristian Crețu
777. Cristian Crețu
778. Cristian Crețu
779. Cristian Crețu
780. Cristian Crețu
781. Rareș Cotoi
782. Cătălin Daniș
783. Rareș Cotoi
784. Rareș Cotoi
786. Rareș Cotoi
787. Cristian Crețu
788. Rareș Cotoi
789. Cristian Crețu
790. Cristian Crețu
791. Cristian Crețu
792. Cristian Crețu
793. Horea Mureșan
794. Cristian Crețu
795. Cristian Crețu
796. Cristian Crețu
797. Cristian Crețu
798. Cristian Crețu
799. Cristian Crețu
800. Cristian Crețu
801. Rareș Cotoi
802. Rareș Cotoi
803. Rareș Cotoi
804. Rareș Cotoi
805. Rareș Cotoi
806. Cristian Crețu
807. Rareș Cotoi
808. Rareș Cotoi
809. Rareș Cotoi
810. Rareș Cotoi
811. Rareș Cotoi
812. Cristian Crețu
813. Cristian Crețu
814. Cristian Crețu
815. Cristian Crețu
816. Cristian Crețu
817. Cristian Crețu
818. Cristian Crețu
819. Cristian Crețu
820. Cristian Crețu
821. Cristian Crețu
822. Cristian Crețu
823. Cristian Crețu
824. Cristian Crețu
825. Cristian Crețu
826. Cristian Crețu
827. Cristian Crețu
828. Cristian Crețu
829. Cristian Crețu
830. Cristian Crețu
831. Cristian Crețu
832. Cristian Crețu
833. Rareș Cotoi
834. Rareș Cotoi
835. Rareș Cotoi

Bibliografie

[1] *Arhiva Educațională, Infoarena*, URL: `https://www.infoarena.ro/arhiva-educationala`.

[2] Thomas H. Cormen et al., *Introduction to Algorithms*, 2009, URL: `https://mitpress.mit.edu/books/introduction-algorithms`.

[3] M. Frenţiu, H.F. Pop și G. Şerban, *Programming Fundamentals*, Cluj-Napoca: Presa Universitară Clujeană, 2006.

[4] *Informatică, clasa a IX-a, Pbinfo*, URL: `https://www.pbinfo.ro`.

[5] *Informatică, clasa a X-a, Pbinfo*, URL: `https://www.pbinfo.ro`.

[6] *Informatică, clasa a XI-a, Pbinfo*, URL: `https://www.pbinfo.ro`.

[7] *Manuale de informatică aprobate de Ministerul Educaţiei şi Cercetării.*

[8] D. Rancea, *Limbajul Pascal, Algoritmi fundamentali*, Ed. Computer Libris Agora, 1999.