1.(3p) Se consideră algoritmul:

alg($x[0..n-1]$)
  $k \leftarrow 0$; $i \leftarrow 0$
  while $i < n-1$ **do**
    $i \leftarrow i+1$
    if $x[i] > x[k]$ **then** $k \leftarrow i$ **endif**
  endwhile
  return k

(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi folosiţi acest invariant pentru a demonstra că algoritmul returnează indicele celui mai mare element din tabloul x.
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
2.(3p) (a) Descrieţi (pseudocod sau Python) un algoritm care pentru un vector $t[0..k]$ cu valori din $\{0,1,2\}$ returnează valoarea naturală care are pe t ca reprezentare în baza 3 ($t[0]$ este cifra cea mai semnificativă iar $t[k]$ cifra cea mai puţin semnificativă). De exemplu pentru $t = [1,2,0,1]$ va returna 46.
(b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
3. (3p) (a) Descrieţi (pseudocod sau Python) un algoritm care pentru o matrice $A[1..n, 1..n]$ returnează `True` dacă matricea este diagonală (toate elementele în afara celor de pe diagonala principală sunt nule) şi `False` în caz contrar. (b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

1.(3p) Se consideră algoritmul:

alg($a[0..n], x$)
  $s \leftarrow a[n]$; $i \leftarrow n$
  while $i > 0$ **do**
    $i \leftarrow i-1$; $s \leftarrow s * x + a[i]$
  endwhile
  return s

(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi folosiţi acest invariant pentru a demonstra că algoritmul returnează $\sum_{i=0}^{n} a[i] * x^i$.
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

2.(3p) (a) Se consideră un tablou $x[0..n-1]$ cu n elemente numere întregi. Descrieţi în pseudocod un algoritm care returnează indicele primei valori strict pozitive întâlnite la parcurgerea tabloului de la stânga la dreapta. Dacă o astfel de valoare nu există se returnează -1. De exemplu pentru $x = [-1,0,3,-1,0,1]$ se va returna 2.
(b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
3. (3p) (a) Descrieţi (pseudocod sau Python) un algoritm care pentru o matrice $A[1..n, 1..n]$ returnează `True` dacă matricea este inferior triunghiulară (toate elementele aflate deasupra diagonalei principale sunt nule) şi `False` în caz contrar. (b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

1.(3p) Se consideră algoritmul:
alg($x[0..n-1]$)
    $i \leftarrow 0$;
    while $i < n-1$ **do**
        if $x[i] < x[i+1]$ **then** $x[i] \leftrightarrow x[i+1]$ **endif**
        $i \leftarrow i+1$
    endwhile
    return $x[0..n-1]$
(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi folosiţi invariantul pentru a demonstra că algoritmul transformă un tablou x prin plasarea celei mai mici valori pe ultima poziţie.
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
2.(6p) (a) Doi vectori ($x[1..n]$, $y[1..n]$) sunt consideraţi ortogonali dacă produsul lor scalar ($\sum_{i=1}^{n} x[i] * y[i]$) este 0. Descrieţi (pseudocod sau Python) un algoritm care returnează 1 dacă vectorii primiţi ca parametri sunt ortogonali şi 0 în caz contrar.
(b) O matrice pătratică $A[1..n, 1..n]$ este ortogonală dacă oricare două linii din matrice sunt ortogonale (pentru verificarea acestei proprietăţi se va utiliza algoritmul descris la punctul (a)). Descrieţi un algoritm care returnează 1 dacă matricea pătratică primită ca parametru este ortogonală şi 0 în caz contrar. (c) Analizaţi eficienţa algoritmilor propuşi la punctele (a) şi (b) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

1.(3p) Se consideră algoritmul:
alg(n)
    $i \leftarrow 0$; $s \leftarrow 0$;
    while $i < n$ **do**
        $i \leftarrow i+1$
        $s \leftarrow s + 2 * i$
    endwhile
    return s

(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi demonstraţi că algoritmul returnează suma primelor n numere pare ($S = \sum_{i=1}^{n} 2i$).
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
2.(6p) (a) Descrieţi (pseudocod sau Python) un algoritm care, pentru o matrice $A[1..m, 1..n]$ cu elemente din $\{0, 1, \ldots, 255\}$ construieşte tabelul cu frecvenţele de apariţie în matrice ale valorilor din $\{0, 1, \ldots, 255\}$; (b) Descrieţi (pseudocod sau Python) un algoritm care construieşte un tablou cu cele mai frecvente valori din matrice (care corespund valorii maxime din tabelul de frecvenţe). (c) Analizaţi eficienţa algoritmilor propuşi la punctele (a) şi (b) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

1.(3p) Se consideră algoritmul:

alg($b[0..k]$) // $\{k \geq 1$, elementele lui b sunt din $\{0,1\}\}$
  $i \leftarrow k$
  $n \leftarrow b[k]$
  while $i >= 1$ **do**
    $n \leftarrow 2 * n + b[i-1]$
    $i \leftarrow i - 1$
  endwhile
  return n

(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi demonstraţi că returnează valoarea in baza 10 a numărului reprezentat în baza 2 prin $b_k b_{k-1} ... b_0$ (adică $n = \sum_{i=0}^{k} b_i 2^i$).
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

2.(6p) (a) Descrieţi (pseudocod sau Python) un algoritm care pentru un tablou $x[1..n]$ cu elemente din $\{0, 1\}$ returnează 1 dacă toate elementele sunt egale cu 1, 0 dacă toate elementele sunt egale cu 0 şi -1 în toate celelalte situaţii. (b) Descrieţi (pseudocod sau Python) un algoritm care pentru o matrice $A[1..m, 1..n]$ cu elemente din $\{0, 1\}$ determină numărul de linii care au toate elementele egale cu 0 şi numărul de coloane care au toate elementele egale cu 1. Se va folosi algoritmul de la punctul (a). (c) Analizaţi eficienţa algoritmilor propuşi la punctele (a) şi (b) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

1.(3p) Se consideră algoritmul:

`alg`(n)
  $i \leftarrow 0$; $P \leftarrow n$ MOD 10; $n \leftarrow n$ DIV 10
  while $n > 0$ **do**
    $i \leftarrow i + 1$
    $P \leftarrow P * (n$ MOD $10)$
    $n \leftarrow n$ DIV 10
  endwhile
  return P

(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi demonstraţi că algoritmul returnează produsul cifrelor lui n (dacă $n = c_k c_{k-1} ... c_0$ atunci algoritmul returnează $c_k * c_{k-1} * ... * c_0$).
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

2.(3p) (a) Descrieţi (pseudocod sau Python) un algoritm care pentru un tablou $x[1..n]$, primit ca parametru, determină câte dintre elementele aflate la începutul tabloului sunt în ordine strict descrescătoare. De exemplu, pentru $x = [7, 3, 1, 4]$ se returnează 3, pentru $x = [1, 7, 3, 5]$ se returnează 1. (b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

3.(3p) (a) Descrieţi (pseudocod sau Python) un algoritm care pentru două tablouri $x[1..m]$ şi $y[1..n]$ ordonate crescător construieşte un tablou ce conţine valorile comune din cele două tablouri. De exemplu pentru $x = [2, 5, 7, 8, 9]$ şi $y = [1, 2, 4, 7]$ se va obţine $z = [2, 7]$. (b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

1.(3p) Se consideră algoritmul:

alg$(x[0..n-1])$
  $i \leftarrow 0$;
  while $i < n-1$ **do**
    if $x[i] > x[i+1]$ **then** $x[i] \leftrightarrow x[i+1]$ **endif**
    $i \leftarrow i+1$
  endwhile
  return $x[0..n-1]$

(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi demonstraţi că algoritmul transformă un tablou x prin plasarea celei mai mari valori pe ultima poziţie.
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
2.(6p) (a) Se consideră un tablou $x[1..n]$ cu n elemente reale. Descrieţi (pseudocod sau Python) un algoritm care calculează abaterea standard a elementelor din tabloul x folosind formula $s = \sqrt{\sum_{i=1}^{n}(x[i]-m)^2/n}$, unde $m = \sum_{i=1}^{n} x[i]/n$; (b) Descrieţi (pseudocod sau Python) un algoritm care determină liniile dintr-o matrice pentru care abaterea standard a elementelor este maximă (se va utiliza algoritmul de la punctul (a)); (c) Analizaţi eficienţa algoritmului propus la punctele (a) şi (b) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.

1.(3p) Se consideră algoritmul:

`alg`(n)
  $i \leftarrow 0$; $S \leftarrow n$ MOD 10; $n \leftarrow n$ DIV 10
  while $n > 0$ **do**
    $i \leftarrow i+1$
    $S \leftarrow S + (n$ MOD $10)$
    $n \leftarrow n$ DIV 10
  endwhile
  return S

(a) Propuneţi un invariant pentru prelucrarea repetitivă din algoritmul `alg` descris mai sus şi demonstraţi că algoritmul returnează suma cifrelor lui n (dacă $n = c_k c_{k-1}...c_0$ atunci algoritmul returnează $\sum_{i=0}^{k} c_i$).
(b) Analizaţi eficienţa algoritmului `alg` parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
2.(3p) (a) Descrieţi în pseudocod un algoritm care pentru un tablou $x[1..n]$, primit ca parametru, determină câte dintre elementele aflate la sfârşitul tabloului sunt în ordine strict crescătoare (când parcurgem tabloul de la dreapta la stânga). De exemplu, pentru $x = [7,3,4,2]$ se returnează 2, pentru $x = [1,7,6,5]$ se returnează 3.
(b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.
3.(3p) (a) Descrieţi (pseudocod sau Python) un algoritm care pentru două tablouri $x[1..m]$ şi $y[1..n]$ ordonate strict crescător construieşte un tablou ce conţine valorile care se află în x dar nu se află în y. De exemplu pentru $x = [2,5,7,8,9]$ şi $y = [1,2,4,7]$ se va obţine $z = [5,8,9]$. (b) Analizaţi eficienţa algoritmului propus la punctul (a) parcurgând următoarele etape: stabiliţi dimensiunea problemei, alegeţi operaţia dominantă, estimaţi timpul de execuţie şi stabiliţi ordinul de complexitate.