1.(4p) Fie $x[1..n]$ un tablou cu n elemente. (a) Scrieţi un algoritm care returnează numărul de perechi de elemente $(x[i], x[j])$ cu proprietatea că $i < j$ şi $x[i] > x[j]$; (b) Scrieţi un algoritm care inversează ordinea elementelor din $x[1..n]$ (primul element este interschimbat cu ultimul, al doilea cu penultimul etc); (c) Stabiliţi ordinul de complexitate al algoritmului descris la pct. (a); (d) Scrieţi un algoritm care sorteaza descrescător tabloul $x[1..n]$ utilizând metoda interschimbării elementelor vecine.
2.(3p) Se consideră următorul algoritm recursiv:

```
alg(a, p)
if p = 0 then return 1
else if pMOD2 = 0
    then b ← alg(a, p/2)
        return b * b
    else
        b ← alg(a, (p − 1)/2)
        return b * b * a
    endif
endif
```

(a) Ce returnează algoritmul când este apelat pentru o valoare reală a şi o valoare naturală p? (b) Presupunând că $p = 2^k$ şi luând în considerare doar operaţia de înmulţire să se scrie relaţia de recurenţă care descrie evoluţia timpului de execuţie $T(k)$ (c) Să se rezolve relaţia de recurenţă şi să se determine ordinul de complexitate a algoritmului (în raport cu p).
3.(3p) (a) Scrieţi un algoritm de complexitate medie $\mathcal{O}(n)$ care determină al k-lea element în ordine crescătoare dintr-un tablou $x[1..n]$ care nu este neapărat ordonat crescător. De exemplu pentru $x = [3, -1, 6, 7, 2, 5, -4]$ şi $k = 3$ valoarea căutată este 2. (b) Analizaţi eficienţa algoritmului propus în cazul cel mai favorabil şi în cazul cel mai defavorabil.
4.(3p) Fie $x[1..m]$ şi $y[1..n]$ două şiruri. Se pune problema determinării, folosind tehnica programării dinamice, a lungimii celui mai lung subşir comun a celor două şiruri (lungimea unui subşir optim). (a) Scrieţi relaţia de recurenţă corespunzătoare lungimii unui subşir optim; (b) Scrieţi un algoritm care dezvoltă relaţia de recurenţă şi determinaţi lungimea unui subşir optim.

1.(4p) Fie $x[1..n]$ un tablou cu n elemente având valori în mulţimea $\{1, 2, \ldots, m\}$. Scrieţi un algoritm care construieşte tabelul frecvenţelor valorilor din x (în tabloul frecvenţelor, $f[1..m]$, elementul $f[i]$ conţine numărul de elemente din x care au valoarea i (b) Scrieţi un algoritm care determină numărul de elemente distincte din x (c) Stabiliţi ordinul de complexitate al algoritmului descris la pct. (a); (d) Scrieţi un algoritm care sortează crescător tabloul $x[1..n]$ folosind metoda sortării prin numărare.
2. (3p) Se consideră următorul algoritm recursiv (pentru care $x[1..n]$ si v sunt variabile globale; x este ordonat crescător):

```
alg(integer i, j)
if i ≤ j then
    m ← ⌊(i + j)/2⌋
    if x[m] = v then return m
        else if v < x[m] then return alg(i, m − 1)
        else return alg(m + 1, j)
endif endif endif
```

(a) Stabiliţi ce returnează algoritmul când este apelat pentru $i = 1$ şi $j = n$. (b) Considerând ca $n = 2^k$ şi că operaţia dominantă este comparaţia, scrieţi relaţia de recurenţă care descrie evoluţia timpului de execuţie $T(k)$ (numărul de comparaţii) în cazul cel mai defavorabil; (c) Dezvoltaţi relaţia de recurenţă şi determinaţi ordinul de complexitate a algoritmului (ordinul de complexitate se va exprima în raport cu n).
3. (3p) (a) Scrieţi un algoritm, folosind tehnica reducerii sau tehnica backtracking, care generează toate şirurile de n valori binare (pentru $n = 3$ acestea sunt: (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)). (b) Stabiliţi ordinul de complexitate al algoritmului propus.
4. (3p) Fie A_1, A_2, ... A_n un set de activităţi de durata egală cu 1. Fiecare activitate A_i este caracterizată de un profit p_i şi de un termen final de execuţie $1 \leq t_i \leq n$. Profitul se obţine doar daca activitatea este executată înainte de termenul final. O planificare a activităţilor este un tablou $s[1..n]$ unde $s[i]$ va conţine indicele activităţii planificate în intervalul $(i-1, i]$ (a) Scrieţi un algoritm, bazat pe tehnica greedy, care determină o planificare de profit maxim. (b) Stabiliţi ordinul de complexitate al algoritmului propus.
