1.

Se consideră fişierul **BAC.TXT** ce conţine un şir **crescător** cu cel mult un milion de numere naturale de cel mult nouă cifre fiecare, separate prin câte un spaţiu.

a) Să se scrie un program C/C++ care, folosind un algoritm eficient din punct de vedere al memoriei utilizate şi al timpului de executare, citeşte din fişier toţi termenii şirului şi afişează pe ecran, pe o singură linie, fiecare termen distinct al şirului urmat de numărul de apariţii ale acestuia în şir. Valorile afişate sunt separate prin câte un spaţiu.

Exemplu: dacă fişierul **BAC.TXT** are următorul conţinut:

```
1 1 1 5 5 5 5 9 9 11 20 20 20
```

programul va afişa:

```
1 3 5 4 9 2 11 1 20 3
```

deoarece 1 apare de 3 ori, 5 apare de 4 ori, etc. **(6p.)**

b) Descrieţi succint, în limbaj natural, metoda de rezolvare folosită, explicând în ce constă eficienţa ei (3 – 4 rânduri). **(4p.)**

2.

Se consideră două tablouri unidimensionale **a** şi **b** fiecare având numere naturale de maximum patru cifre, **ordonate crescător**. Tabloul **a** conţine **n** (1<n<100) numere pare, iar tabloul **b** conţine **m** (1<m<100) numere impare.

a) Scrieţi un program C/C++ care citeşte de la tastatură valoarea lui **n** şi cele **n** elemente ale tabloului **a**, apoi valoarea lui **m** şi cele **m** elemente ale tabloului **b** după care scrie în fişierul **BAC.TXT** un număr maxim de elemente ale tablourilor date, numerele fiiind scrise în ordine crescătoare, separate prin câte un spaţiu, iar cele aflate pe poziţii consecutive fiind de paritate diferită. Programul va utiliza un algoritm eficient din punct de vedere al timpului de executare.

Exemplu: pentru n=6, m=5 şi tablourile a=(2,4,8,10,14,16) şi b=(3,5,7,11,15) fişierul **BAC.TXT** va avea următorul conţinut : 2 3 4 5 8 11 14 15 16 **(6p.)**

b) Descrieţi succint, în limbaj natural, algoritmul pe baza căruia a fost scris programul de la punctul **a)**, explicând în ce constă eficienţa metodei utilizate. **(4p.)**

3.

În fişierul **numere.txt** sunt memorate pe mai multe linii, numere întregi (cel mult 100), numerele de pe aceeaşi linie fiind despărţite prin câte un spaţiu, fiecare număr având cel mult 9 cifre. Să se determine cele mai mici două valori având **exact** două cifre fiecare, memorate în fişier şi să se afişeze pe ecran aceste valori, despărţite prin câte un spaţiu.

a) Descrieţi în limbaj natural o metodă eficientă de rezolvare din punct de vedere al gestionării memoriei şi timpului de executare. **(4p.)**

b) Scrieţi programul C/C++ corespunzător metodei descrise la punctul **a.** **(6p.)**

Exemplu: dacă fişierul **numere.txt** are conţinutul alăturat, se va afişa pe ecran

```
-77 şi 10
```

```
5 10
3 -77 20
50 5 0 12 18 30
```

4.

a) Subprogramul **max_cif** primeşte prin parametrul **x** un tablou unidimensional, cu cel mult 100 de elemente, numere întregi cu cel mult 4 cifre fiecare, iar prin parametrul **n** un număr natural ce reprezintă dimensiunea tabloului **x** (**n≤100**). Scrieți definiția completă a subprogramului **max_cif** care returnează cel mai mare număr de trei cifre al tabloului **x**. Dacă tabloul nu conține nicio valoare de trei cifre, subprogramul va returna 0. **(6p.)**

b) În fişierul **numere.txt** se află memorat pe prima linie un număr natural **n** (**n≤100**), iar pe următoarele **n** linii, câte **n** numere întregi despărțite prin câte un spațiu. Scrieți în limbajul C/C++, un algoritm eficient din punct de vedere al gestionării memoriei care citeşte din fişier datele existente şi, folosind apeluri utile ale subprogramului **max_cif**, determină şi afişează cel mai mare număr de trei cifre memorat în fişier. Dacă în fişier nu există niciun număr de trei cifre se va afişa 0. **(10p.)**

c) Explicați în limbaj natural metoda utilizată justificând eficiența acesteia. **(4p.)**

Exemplu: dacă fişierul **numere.txt** are conținutul alăturat, se va afişa valoarea: 345.

```
5
112  333   1 18  345
 -1   95   7 97 -12
45 -806   0  7   89
1    5   17 197 -102
45 -86   0  7   9
```

5. Fişierul text **numere.txt** conține pe prima linie un număr natural **n** (**0<n<100000**) iar pe doua linie **n** numere naturale, formate dintr-o singură cifră, separate prin câte un spațiu.

a) Scrieți un program C/C++ care determină în mod eficient, din punct de vedere al timpului de executare, cea mai mare cifră dintre cele situate pe a doua linie a fişierului, precum şi numărul de apariții ale acesteia. Cele două numere vor fi afişate pe o singură linie a ecranului, separate printr-un spațiu.

Exemplu: daca fişierul **numere.txt** are următorul conținut:

```
7
3 5 2 1 5 3 1
```

atunci pe ecran se va afişa: 5 2. **(6p.)**

6. Fişierul text **numere.txt** conține pe prima linie un număr natural **n** (**0<n<100000**) iar pe doua linie, separate prin câte un spațiu, **n** numere naturale formate din cel mult 2 cifre fiecare.

a) Scrieți un program C/C++ care determină în mod eficient, din punct de vedere al timpului de executare, numerele ce apar o singură dată în a doua linie a fişierului. Aceste numere vor fi afişate pe ecran în ordine crescătoare, separate prin câte un spațiu.

Exemplu: dacă fişierul **numere.txt** are următorul conținut:

```
7
3 5 2 1 5 23 1
```

7. atunci pe ecran se va afişa: 2 3 23. **(6p.)**

8.

Fişierul text **numere.txt** conţine pe prima linie un număr natural **n** (0<n<100000) iar pe a doua linie **n** cifre, separate prin câte un spaţiu.

a) Scrieţi un program C/C++ care determină în mod eficient, din punct de vedere al timpului de executare, cel mai mare număr ce se poate forma cu toate cifrele conţinute de a doua linie a fişierului **numere.txt**. Numărul determinat se va afişa pe ecran.

Exemplu: daca fişierul **numere.txt** are următorul conţinut:

```
7
2 5 3 1 5 8 9
```

atunci pe ecran se va afişa: `9855321`. **(6p.)**

9.

Fişierul text **numere.txt** conţine pe prima linie un număr natural **n** (0<n<100000) iar pe a doua linie **n** numere naturale, formate din cel mult 4 cifre, separate prin câte un spaţiu.

a) Scrieţi un program C/C++ care determină în mod eficient, din punct de vedere al timpului de executare, cifrele ce apar în scrierea numerelor situate pe a doua linie a fişierului. Programul va afişa pe ecran aceste cifre in ordine crescătoare, separate prin câte un spaţiu.

Exemplu: daca fişierul **numere.txt** are următorul conţinut:

```
7
243 32 545 74 12 1344 90
```

atunci pe ecran se va afişa: `0 1 2 3 4 5 7 9` **(6p.)**

10.

Fişierul text **date.in** conţine pe prima linie , separate prin câte un spaţiu, cel mult 1000 numere naturale, fiecare dintre ele având maximum 9 cifre.

a) Scrieţi un program C/C++ care citeşte numerele din fişierul **date.txt** şi determină cea mai lungă secvenţă ordonată strict descrescător, formată din valori citite consecutiv din fişier. Numerele din secvenţa găsită vor fi afişate pe ecran, pe o linie, separate prin câte un spaţiu. Dacă sunt mai multe secvenţe care respectă condiţia impusă, se va afişa doar prima dintre acestea. Alegeţi o metodă de rezolvare eficientă din punctul de vedere al timpului de executare.

Exemplu: dacă fişierul **date.in** conţine

```
5 2 9 4 3 6 3 2 1 0 8
```

pe ecran se afişează:

```
6 3 2 1 0
```

(6p.)

11.

Fişierul text **numere.txt** conţine pe prima linie un număr natural **n** (0<n<100000) iar pe a doua linie **n** numere naturale, formate din cel mult 2 cifre, separate prin câte un spaţiu.

a) Scrieţi un program C/C++, eficient atât din punct de vedere al timpului de executare, care afişează pe ecran toate numerele situate pe a doua linie a fişierului, în ordinea crescătoare a valorilor lor, separate prin câte un spaţiu.

Exemplu: dacă fişierul **numere.txt** are următorul conţinut:

```
7
12 21 22 11 9 12 3
```

atunci pe ecran se va afişa: `3 9 11 12 12 21 22` **(6p.)**

Subprogramul **verif** primeşte prin singurul său parametru, **x**, un număr natural nenul cu cel mult 9 cifre şi returnează valoarea 1 dacă numărul conține cel puțin o secvență de 3 cifre impare alăturate şi 0 în caz contrar.
Exemplu: la apelul `verif(7325972)` se va returna valoarea 1.

a) Scrieți definiția completă a subprogramului **verif**. **(10p.)**

b) Fişierul text **date.txt** conține pe prima linie un număr natural nenul **n** cu cel mult 4 cifre şi pe fiecare dintre următoarele **n** linii câte un număr natural, cu exact 6 cifre. Scrieți un program C/C++ care citeşte numerele din fişierul **date.txt** şi afişează pe ecran, separate prin câte un spațiu, acele numere care au primele 3 cifre impare. Se vor utiliza apeluri utile ale subprogramului **verif**. Dacă nu există niciun număr cu această proprietate, se va afişa mesajul **nu**. Alegeți o metodă eficientă din punctul de vedere al memoriei utilizate.

De exemplu: dacă fişierul **date.txt** conține	Pe ecran se afişează:
3 133579 345796 973314	133579 973314

12. Subprogramul **cifra** primeşte prin singurul său parametru **x**, un număr real nenul pozitiv şi furnizează prin parametrul y valoarea cifrei unităților părții întregi a lui **x**.
Exemplu: la apelul `cifra(34.567)` se va returna 4.

a) Scrieți definiția completă a subprogramului **cifra**. **(10p.)**

b) Fişierul text **medii.txt** conține cel mult 600 de linii. Pe fiecare linie se află, separate printr-un spațiu, două numere reale, cu cel mult două zecimale, din intervalul `[1,10]`, care reprezintă media pe semestrul 1 respectiv media pe semestrul al 2-lea, ale unui elev. În situațiile statistice pe care şcoala le realizează, fiecare medie este încadrată într-una dintre următoarele categorii de medii: `[3,3.99]`, `[4,4.99]`, `[5,5.99]`, `[6,6.99]`, `[7,7.99]`, `[8,8.99]`, `[9,10]`. Scrieți un program C/C++ care citeşte datele din fişier şi afişează pe ecran numărul elevilor care au media din semestrul al 2-lea în categoria imediat următoare categoriei căreia îi aparține media din semestrul 1. Ordinea categoriilor este cea din enumerarea de mai sus. În program se vor folosi apeluri utile ale subprogramului **cifra**. Se va utiliza un algoritm eficient din punctul de vedere al memoriei utilizate.

Exemplu: dacă fişierul **medii.txt** conține:	Pe ecran se afişează:
9.45 7.90 6.34 7.60 8.75 9.90	2

13.

Subprogram `sfx` primeşte prin singurul său parametru, `x`, un număr natural din intervalul `[100,2000000000]` şi returnează valoarea `1` dacă ultimele trei cifre ale numărului sunt în ordine strict descrescătoare sau valoarea `0` în caz contrar.
Exemplu: la apelul `sfx(24973)` se va returna valoarea `1`.

a) Scrieți definiția completă a subprogramului `sfx`. **(10p.)**

b) Fişierul text `date.in` conține cel mult `10000` de numere naturale de exact `6` cifre fiecare, separate prin câte un spațiu. Scrieți un program `C/C++` care citeşte toate numerele din fişier, determină şi afişează pe ecran câte dintre aceste numere au toate cifrele în ordine strict descrescătoare. Programul va folosi apeluri utile ale subprogramului `sfx`. Se va utiliza un algoritm eficient din punctul de vedere al memoriei utilizate. **(6p.)**

Exemplu: dacă fişierul `date.in` conține

```
236543
865210
976532
```

Pe ecran se afişează:

```
2
```

14. Fişierele text `A.TXT` şi `B.TXT` conțin cel mult `10000` de numere naturale cu cel mult `9` cifre fiecare, scrise fiecare pe câte o linie.

a) Scrieți un program `C/C++` care citeşte numerele din cele două fişiere şi, printr-o metodă eficientă din punct de vedere al timpului de executare şi al spațiului de memorie utilizat şi afişează pe ecran câte dintre numerele din fişierul `A.TXT` sunt strict mai mici decât toate numerele memorate în fişierul `B.TXT`. **(6p.)**

Exemplu: dacă fişierul `A.TXT` are conținutul alăturat

```
41111
81111
11111
91111
51111
111111
31111
431111
61111
201111
```

iar fişierul `B.TXT` are conținutul alăturat:

```
91111
91111
61111
91111
91111
81111
61111
91111
```

15. atunci programul va afişa valoarea `4` deoarece `41111`, `11111`, `51111`, `31111` sunt mai mici decât toate elementele din fişierul `B.TXT`

Fişierul text `NUMERE.TXT` conține pe prima linie un număr natural `n` (`1≤n≤10000`) şi pe a doua linie, un şir **crescător** de `n` numere naturale, fiecare având cel mult `9` cifre. Numerele de pe a doua linie sunt separate prin câte un spațiu.

a) Scrieți un program `C/C++` care utilizând o metodă eficientă din punct de vedere al timpului de executare şi al spațiului de memorie, afişează pe ecran elementele distincte ale şirului aflat pe a doua linie a fişierului. **(6p.)**

Exemplu: dacă fişierul `NUMERE.TXT` are conținutul alăturat

```
7
111 111 111 2111 4111 71111 71111
```

16. atunci programul va afişa pe ecran `111 2111 4111 71111`

17. Fişierul text **SIR.TXT** conține pe prima linie un număr natural **n** (1≤n≤10000) şi pe a doua linie, separate prin spații, un şir **crescător** de **n** numere naturale cu cel mult 9 cifre fiecare.
Numim platou într-un şir de valori, o secvență de elemente identice situate pe poziții alăturate. Lungimea unui platou este egală cu numărul de elemente care îl formează.

a) Scrieți un program C/C++ care citeşte valorile din fişier şi, printr-o metodă eficientă din punct de vedere al timpului de executare şi al spațiului de memorie utilizat afişează pe ecran, separate prin câte un spațiu, lungimea maximă a unui platou, precum şi valoarea care formează platoul. În cazul în care sunt mai multe platouri de aceeaşi lungime se va afişa valoarea cea mai mare care formează unul dintre aceste platouri. **(6p.)**

Exemplu: dacă fişierul **SIR.TXT** are conținutul alăturat

```
10
111 2111 2111 2111 3111 4111 4111 51111 51111 51111
```

atunci programul va afişa pe ecran `3 51111`

18. Fişierul text **NUMERE.TXT** conține pe prima linie un număr natural **n** (1≤n≤10000) şi pe a doua linie, separate prin spații, **n** numere naturale cu cel mult 9 cifre fiecare. Aceste numere sunt dispuse în ordine **crescătoare** şi separate între ele printr-un spațiu.

a) Scrieți un program C/C++ care citeşte valorile din fişier şi, printr-o metodă eficientă din punct de vedere al timpului de executare şi al spațiului de memorie utilizat, afişează pe ecran separate printr-un spațiu, în ordine crescătoare, numerele pare de pe a doua linie a fişierului, urmate de cele impare în ordine descrescătoare. **(6p.)**

Exemplu: dacă fişierul **NUMERE.TXT** are conținutul alăturat

```
6
212 412 5111 71113 81112 101112
```

atunci programul va afişa pe ecran `212 412 81112 101112 71113 5111`

19. Fişierul text **NUMERE.TXT** conține pe prima linie un număr natural **n** (1≤n≤10000) şi pe a doua linie, **n** numere naturale cu cel mult 9 cifre fiecare, numere nu neapărat distincte. Aceste numere sunt dispuse în ordine **crescătoare** şi separate între ele printr-un spațiu.

a) Scrieți un program C/C++ care citeşte valorile din fişier şi, printr-o metodă eficientă din punct de vedere al timpului de executare şi al spațiului de memorie utilizat, afişează pe ecran, cu câte un spațiu între ele, valoarea care apare de cele mai multe ori în fişier şi de câte ori apare ea. Dacă există mai multe valori care apar de un număr maxim de ori, se va afişa cea mai mică dintre ele. **(6p.)**

Exemplu: dacă fişierul **NUMERE.TXT** are conținutul alăturat

```
8
711 711 711 11111 11111 11111 191111 231111
```

atunci programul va afişa pe ecran `711 3`

Fişierul text **NUMERE.IN** conține cel mult 100000 numere naturale de cel mult nouă cifre fiecare, numerele fiind despărțite prin câte un spațiu.

a) Scrieți programul C/C++ care citeşte numerele din fişierul **NUMERE.IN** şi, folosind apeluri utile ale subprogramului **Palindrom** definit la punctul **3**, determină în mod eficient, din punct de vedere al memoriei utilizate şi al timpului de executare, care este cel mai mare număr palindrom citit şi de câte ori apare el în fişierul **NUMERE.IN**. Programul scrie în fişierul text **NUMERE.OUT** numărul astfel determinat precum şi numărul de apariții ale acestuia, pe rânduri diferite.

Exemplu: dacă **NUMERE.IN** conține numerele:

```
23 565 78687 7887 7865 78687 7887 23 78687 98798
```

atunci **NUMERE.OUT** va conține:

```
78687
3
```

(6p.)

20.

Se consideră fişierul **BAC.TXT** ce conține cel mult un milion de numere naturale separate prin spatii, fiecare număr având cel mult nouă cifre.

a) Scrieți un program C/C++ care citeşte toate numerele din fişierul **BAC.TXT** şi determină, folosind un algoritm eficient din punct de vedere timpului de executare, cele mai mari două numere de trei cifre care nu se află în fişier. Dacă fişierul conține toate numerele de câte trei cifre atunci programul va afişa pe ecran valoarea 0.

Exemplu: dacă fişierul **BAC.TXT** conține numerele:

```
12 2345 123 67 989 6 999 123 67 989 999
```

atunci programul va afişa

```
998 997
```

(6p.)

21.

Evidența produselor vândute de o societate comercială este păstrată în fişierul **PRODUSE.TXT**. Pentru fiecare produs se cunoaşte tipul produsului (un număr natural de cel mult 4 cifre), cantitatea exprimată in kilograme (un număr natural mai mic sau egal cu 100) şi prețul unui kilogram (un număr natural mai mic sau egal cu 100).

Produsele de acelaşi tip pot fi vândute în cantități diferite, fiecare vânzare fiind înregistrată separat.

Fişierul **PRODUSE.TXT** are cel mult 200000 de linii şi fiecare linie conține trei numere naturale, separate prin câte un spațiu, ce reprezintă, în această ordine tipul, cantitatea şi prețul de vânzare al unui produs la un moment dat.

a) Să se scrie un program C/C++, care utilizând un algoritm eficient din punct de vedere al timpului de executare, determină pentru fiecare tip de produs vândut suma totală obținută în urma vânzărilor. Programul va afişa pe câte o linie a ecranului tipul produsului şi suma totală obținută, separate prin câte un spațiu, ca în exemplu.

Exemplu: dacă fişierul **PRODUSE.TXT** are conținutul alăturat, programul va afişa numerele următoare:

```
1 150
2 30
3 5
```

(6p.)

```
3 1 5
1 20 5
2 10 3
1 10 5
```

22.

Se consideră şirul s: 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2,... .
Pentru un număr natural k, 0<k≤100000, se cere să se determine valoarea elementului ce se află pe poziția k în şirul s.
Exemplu: pentru k=5 numărul cerut este 2.

a) Scrieți un program C/C++ care citeşte de la tastatură valoarea numărului natural k şi, prin apeluri utile ale funcției **Ecuație**, definite la punctul **3**, determină valoarea elementului ce se află pe poziția k în şirul s, folosind un algoritm eficient din punctul de vedere al spațiului de memorie alocat şi al timpului de executare. Valoarea astfel determinată se va scrie în fişierul text sir.out. **(6p.)**

23.

a) Scrieți definiția completă a subprogramului Cautare, cu 3 parametri, care primeşte prin parametrul n un număr natural (1≤n≤100), prin parametrul x un tablou unidimensional format din n componente (numere întregi de cel mult patru cifre fiecare: x_1, x_2, ..., x_n) memorate în ordine crescătoare şi prin parametrul val un număr întreg de cel mult patru cifre, diferit de oricare dintre elementele tabloului unidimensional x.
Subprogramul va căuta, în mod eficient din punct de vedere al timpului de executare, poziția pe care ar trebui inserată valoarea val în şirul x astfel încât să se obțină tot un şir ordonat şi returnează această poziție. **(6p.)**

24.

a) Se numeşte "număr mare" un număr care are mai mult de nouă cifre.

Scrieți un program C/C++ care citeşte din fişierul text **NUMERE.IN** un număr natural n (10 <n < 1000) apoi de pe următoarea linie n numere naturale cu cel mult nouă cifre fiecare, dintre care cel puțin unul nenul, şi folosind apeluri ale funcției Cifra definită la punctul 3 construieşte în mod eficient din punct de vedere al memoriei folosite, cel mai mic "număr mare" cu exact n cifre. Fiecare dintre cifrele numărului mare reprezintă cifra de valoare maximă a unui număr citit din fişier şi nu există două cifre în "numărul mare" care să aparțină aceluiaşi număr citit. Scrieți în fişierul text **NUMERE.OUT** "numărul mare" obținut astfel. **(6p.)**

b) Descrieți succint în limbaj natural metoda de rezolvare folosită explicând în ce constă eficiența ei (3 – 4 rânduri) **(4p.)**

Exemplu:

NUMERE.IN	NUMERE.OUT
10 725 3695 423 0 7895 0 100 101 870 568	1001478899

25.

a) Scrieți definiția completă a funcției UltimaCifra care primeşte prin cei doi parametri a şi b câte un număr natural (0<a<1000000, 0<b<1000000), calculează în mod eficient din punct de vedere al timpului de executare şi returnează ultima cifră a numărului a^b (a la puterea b). **(6p.)**

26.

În fişierul **numere.txt** sunt memorate maximum 10000 de numere naturale cu cel mult 9 cifre fiecare. Fiecare linie a fişierului conține câte un număr. Se cere afişarea pe ecran, în ordine descrescătoare, a tuturor cifrelor care apar în numerele din fişier. Alegeți un algoritm de rezolvare eficient din punct de vedere al memoriei utilizate şi al timpului de execuție.
Exemplu: dacă fişierul numere.txt conține:
267
39628
79
se va tipări 9987766322.

27.

Fişierul **BAC.TXT** are pe prima linie două numere naturale **n** şi **m** (0<n<1000, 0<m<1000) separate prin câte un spaţiu, pe linia a doua **n** numere întregi ordonate strict crescător, iar pe linia a treia **m** numere naturale distincte. Să se scrie programul C/C++ care citeşte toate numerele din fişier şi afişează pe ecran, despărţite prin câte un spaţiu, toate numerele din a doua linie a fişierului care apar cel puţin o dată şi în linia a treia a acestuia

Exemplu: dacă fişierul are următorul conţinut:

```
7 5
2 3 4 5 8 9
4 5 2 11 8
```

atunci se va afişa: `5 2 8`, nu neapărat în această ordine.

a) Descrieţi în limbaj natural o metodă eficientă de rezolvare ca timp de executare. **(4p.)**

28. **b)** Scrieţi programul C/C++ corespunzător metodei descrise la punctul **a)**. **(6p.)**

Fişierul text **bac.in** conţine pe prima sa linie un număr natural **n** (0<n<10000), iar pe următoarea linie **n** numere naturale din intervalul [1,100] separate prin câte un spaţiu. Se cere să se citească din fişier toate numerele şi să se afişeze pe ecran numărul sau numerele care apar de cele mai multe ori printre numerele citite. Numerele afişate vor fi separate prin câte un spaţiu. Alegeţi un algoritm de rezolvare eficient atât din punctul de vedere al timpului de executare cât şi al gestionării memoriei. .

Exemplu: pentru **n=12** şi numerele `1 2 2 3 2 9 3 3 9 9 7 1` se vor afişa valorile 2, 3 şi 9.

a) Explicaţi în limbaj natural metoda utilizată justificând eficienţa acesteia (4-6 rânduri) **(4p.)**

b) Scrieţi programul C/C++ ce rezolvă problema enunţată, corespunzător metodei descrise la punctul a). **(6p.)**

29.

Fişierul text **bac.in** conţine pe prima sa linie un număr natural **n** (0<n<10000), iar pe următoarea linie **n** numere naturale din intervalul [1,100]. Se cere să se citească din fişier toate numerele şi să se afişeze pe ecran, în ordine descrescătoare, toate numerele care apar pe a doua linie a fişierului şi numărul de apariţii ale fiecăruia. Dacă un număr apare de mai multe ori, el va fi afişat o singură dată. Fiecare pereche „valoare - număr de apariţii" va fi afişată pe câte o linie a ecranului, numerele fiind separate printr-un spaţiu, ca în exemplu. Alegeţi un algoritm de rezolvare eficient din punctul de vedere al timpului de executare.

Exemplu: dacă se citeşte pentru **n** valoarea `12` şi apoi numerele `1 2 2 3 2 2 3 3 2 3 2 1` se va afişa:

```
3 4
2 6
1 2
```

a) Explicaţi în limbaj natural metoda utilizată justificând eficienţa acesteia (4-6 rânduri) **(4p.)**

b) Scrieţi programul C/C++ ce rezolvă problema enunţată, corespunzător metodei descrise

30. la punctul a). **(6p.)**

31. Se citeşte de pe prima linie a fişierului **numere.in** un număr natural **n** (0<n<10000) şi de pe a doua linie a fişierului **n** numere naturale din intervalul [1,100] şi se cere să se afişeze pe ecran, despărțite prin câte un spațiu, numărul sau numerele întregi din intervalul [1,100] care nu apar printre numerele citite. Dacă pe a doua linie a fişierului apar toate numerele din intervalul precizat se va afişa mesajul **NU LIPSESTE NICIUN NUMAR**. Alegeți un algoritm de rezolvare eficient atât din punctul de vedere al timpului de executare.

Exemplu: pentru fişierul **numere.in** cu următorul conținut

```
12
1 2 3 4 5 6 7 8 9 10 11 100
```

se vor afişa valorile 12 13 ... 99.

a) Explicați în limbaj natural metoda utilizată justificând eficiența acesteia (4-6 rânduri) **(4p.)**

b) Scrieți programul C/C++ ce rezolvă problema enunțată, corespunzător metodei descrise la punctul **a)**. **(6p.)**

32. Fişierul text **NUMERE.IN** conține, pe mai multe linii, cel mult 30000 de numere naturale nenule mai mici sau egale decât 500, despărțite prin câte un spațiu.

a) Scrieți programul C/C++ care, utilizând un algoritm eficient din punct de vedere al timpului de executare, afişează pe ecran, în ordine crescătoare, toate numerele care au apărut exact o singură dată din fişierul **NUMERE.IN**, despărțite prin câte un spațiu.

Exemplu: dacă fişierul **NUMERE.IN** conține numerele scrise alăturat, se vor afişa valorile următoare: 3 4 5 6 34 **(6p.)**

```
2 23 34 3
8 9 9 23
6 8 9 2
4 5 23 9
```

b) Descrieți succint, în limbaj natural, metoda de rezolvare folosită la punctul **a)**, explicând în ce constă eficiența ei (3 – 4 rânduri). **(4p.)**

33. Fişierul text **NUMERE.IN** conține, pe mai multe linii, cel mult 30000 de numere naturale nenule mai mici sau egale decât 500, despărțite prin câte un spațiu.

a) Scrieți programul C/C++ care citeşte toate numerele din fişierul **NUMERE.IN** şi creează fişierul text **NUMERE.OUT** care să conțină pe prima linie cel mai mare număr de două cifre din fişierul **NUMERE.IN**, şi de câte ori apare el în acest fişier, iar pe a doua linie, cel mai mic număr de două cifre din fişierul **NUMERE.IN** şi de câte ori apare el în acest fişier. Alegeți o metodă de rezolvare eficientă din punct de vedere al memoriei utilizate şi al timpului de executare. **(6p.)**

b) Descrieți succint, în limbaj natural, metoda de rezolvare folosită la punctul **a)**, explicând în ce constă eficiența ei (3 – 4 rânduri). **(4p.)**

Exemplu: dacă fişierul **NUMERE.IN** are conținutul alăturat:

```
2 253 34 3
6 88 9 2 3
4 54 34 88
```

atunci fişierul **NUMERE.OUT** va avea următorul conținut:

```
88 2
34 2
```

Fişierul text **NUMERE.IN** conţine, pe fiecare linie, câte un şir de numere naturale nenule mai mici sau egale decât 30000, despărţite prin câte un spaţiu; fiecare linie se termină cu numarul 0 (care se consideră că nu face parte din şirul aflat pe linia respectivă).

a) Scrieţi programul C/C++ care afişează pe ecran valoarea maximă din şirul care conţine cele mai puţine numere. În cazul în care există mai multe şiruri cu acelaşi număr minim de numere, se va afişa cea mai mare valoare care apare în unul dintre aceste şiruri. Alegeţi o metodă de rezolvare eficientă din punct de vedere al memoriei utilizate şi al timpului de executare. **(6p.)**

b) Descrieţi succint, în limbaj natural, metoda de rezolvare folosită la punctul **a)**, explicând în ce constă eficienţa ei (3 – 4 rânduri). **(4p.)**

Exemplu: dacă fişierul **NUMERE.IN** are conţinutul alăturat, atunci pe ecran se va afişa numărul 253.

```
2 253 34 3 0
6 88 9 3 0
4 54 88 12345 98 234 546 0
```

34. Fişierul **bac.txt** conţine cel mult 1000 de numere distincte, dintre care cel puţin două sunt pare. Numerele sunt separate prin câte un spaţiu şi fiecare dintre ele are cel mult 9 cifre.

a) Scrieţi un program C/C++ care determină cele mai mari două numere pare din fişier, utilizând un algoritm eficient din punct de vedere al timpului de executare şi al spaţiului de memorie utilizat. Cele două numere vor fi afişate pe ecran, în ordine descrescătoare, separate printr-un spaţiu.
Exemplu: dacă fişierul conţine numerele: 5123 8 6 12 3 se va afişa: 12 8 **(6p.)**

b) Descrieţi succint, în limbaj natural, algoritmul utilizat, justificând eficienţa acestuia. **(4p.)**

35. Fişierul **bac.txt** conţine un şir de cel mult 2008 numere naturale, cu cel mult **nouă** cifre fiecare, pe mai multe rânduri, separate printr-un spaţiu.

a) Scrieţi un program C/C++ care afişează pe ecran cel mai mic număr din fişier la care suma cifrelor pare este egală cu suma cifrelor impare, precum şi numărul de apariţii în fişier ale acestui număr, folosind o metodă eficientă din punctul de vedere al timpului de executare. Cele două valori vor fi afişate pe o linie a ecranului, separate printr-un spaţiu.
Exemplu: dacă în fişier avem numerele 22031 9021 22031 1021 2011 10012 1021 457008 99882 atunci pe ecran se vor afişa numerele: 1021 2. **(6p.)**

b) Descrieţi succint, în limbaj natural, algoritmul utilizat, justificând eficienţa acestuia. **(4p.)**

36. Fişierul **bac.txt** conţine cel mult 10000 de numere din intervalul închis [0, 99], separate prin spaţiu, pe mai multe rânduri.

a) Scrieţi un program C/C++ care determină şi afişează pe ecran cel mai mare număr **prim** care apare în fişier şi numărul de apariţii ale acestuia, utilizând un algoritm eficient din punct de vedere al timpului de executare şi al spaţiului de memorie utilizat. Programul afişează pe ecran cele două valori determinate separate printr-un spaţiu.
Exemplu: dacă fişierul conţine numerele: 5 8 99 5 1 1 2 2 se va afişa 5 2. **(6p.)**

b) Descrieţi succint, în limbaj natural, algoritmul utilizat, justificând eficienţa acestuia. **(4p.)**

37.

Fişierul **bac.txt** conţine pe mai multe rânduri cel mult 50000 de numere din intervalul închis [0, 99], separate prin câte un spaţiu.

a) Scrieţi un program C/C++ care afişează pe ecran **în ordine descrescătoare** acele numere din fişier care sunt mai mari decât un număr natural **k**, citit de la tastatură, utilizând un algoritm eficient din punct de vedere al timpului de executare. Dacă un număr apare de mai multe ori, şi este mai mare decât **k**, se va afişa o singură dată. Numerele vor fi afişate câte 20 pe fiecare linie (cu excepţia ultimei linii care poate să conţină mai puţine valori), separate prin câte un spaţiu.
Exemplu: dacă fişierul conţine numerele: **15 8 99 15 1 37 1 24 2**, iar pentru **k** se citeşte valoarea 7, se vor afişa numerele **99 37 24 15 8**. **(6p.)**

38. **b)** Descrieţi succint, în limbaj natural, algoritmul utilizat, justificând eficienţa acestuia. **(4p.)**

Fişierul **bac.txt** conţine pe mai multe rânduri cel mult 1000 de numere întregi separate prin spaţiu, între care există cel puţin două numere pozitive, neseparate de alt număr. Fiecare număr are cel mult 9 cifre.

a) Scrieţi un program C/C++ care determină cele mai mari două numere pozitive, aflate unul după altul în fişier, a căror sumă este maximă, utilizând un algoritm eficient din punct de vedere al timpului de executare şi al spaţiului de memorie utilizat. Dacă există mai multe soluţii, să se reţină doar aceea pereche pentru care diferenţa dintre cele două numere este maximă. Numerele vor fi afişate pe ecran în ordine descrescătoare, separate printr-un spaţiu.
Exemplu: dacă fişierul conţine numerele: **-2 11 4 16 -1 25 -2 8 12 0 10 10** se vor afişa numerele **16 4**, în această ordine, cu un spaţiu între ele. **(6p.)**

39. Pe prima linie a fişierului text **DATE.TXT** se găseşte o valoare naturală **k** ($k \leq 1000000$).

a) Scrieţi un program C/C++ care citeşte din fişierul **DATE.TXT** valoarea **k** şi afişează, pe ecran, toate perechile de numere naturale nenule **x**, **y** ($x \leq y$) cu proprietatea că $x^2+y^2=k$. Fiecare pereche va fi afişată pe câte o linie, numerele fiind despărţite printr-un spaţiu. Alegeţi o metodă de rezolvare eficientă din punctul de vedere al timpului de executare.
Exemplu: dacă fişierul **DATE.TXT** conţine numărul **1000000**, pe ecran se vor afişa, nu neapărat în această ordine, perechile alăturate. **(6p.)**

```
280 960
352 936
600 800
```

40. Fişierul **numere.in** conţine cel mult 5000 de numere reale, câte unul pe fiecare linie. Se cere să se scrie un program care să citească toate numerele din fişier şi să afişeze pe ecran numărul de ordine al primei, respectiv al ultimei linii pe care se află cel mai mare număr din fişier. Cele două numere vor fi separate printr-un spaţiu. Alegeţi o metodă de rezolvare eficientă din punct de vedere al spaţiului de memorare şi al timpului de executare.
Exemplu: dacă fişierul are conţinutul alăturat, pe ecran se vor afişa numerele **2 6**.

a) Descrieţi succint, în limbaj natural, metoda de rezolvare aleasă, explicând în ce constă eficienţa ei. **(4p.)**

b) Scrieţi programul C/C++ corespunzător metodei descrise. **(6p.)**

```
3.5
7
-4
7
2
7
6.3
5
```

41.

42. Fişierul text `permut.txt` conţine pe prima linie o valoare naturală `n` (`0<n≤9`), iar pe fiecare dintre următoarele linii câte un număr natural format din exact `n` cifre nenule distincte, cifre care aparţin mulţimii `{1,2,...,n}`. Fişierul conţine în ordine strict descrescătoare, toate numerele care îndeplinesc aceste proprietăţi,
Scrieţi un program eficient atât din punctul de vedere al vitezei de executare cât şi al spaţiului de memorie utilizat care citeşte de la tastatură un număr natural nenul, `x`, şi verifică, utilizând apeluri utile ale subprogramului `verif`, dacă `x` apare printre numerele scrise, începând cu a doua linie, în fişierul `permut.txt`. În caz afirmativ, programul va afişa pe ecran mesajul `Apare pe linia` urmat de numărul liniei în care apare valoarea `x` (se consideră ca prima linie din fişier are numărul `1`). În cazul în care `x` nu apare printre numerele din fişier, programul va afişa pe ecran mesajul `Nu apare`.

Exemplu: dacă fişierul `permut.txt` are conţinutul alăturat, iar de la tastatură se citeşte valoarea `213`, programul va afişa următoarele: `Apare pe linia 5`
Dacă pentru acelaşi conţinut al fişierului, de la tastatură se citeşte oricare dintre valorile `211`, `243`, `12` sau `301` programul va afişa mesajul `Nu apare`

```
3
321
312
231
213
132
123
```

a) Descrieţi succint, în limbaj natural, metoda de rezolvare folosită, explicând în ce constă eficienţa ei (3 – 4 rânduri). **(4p.)**

b) Scrieţi un program `C/C++` care rezolvă problema conform metodei descrise. **(6p.)**

43. **a)** Scrieţi un program `C/C++` care citeşte de la tastatură un număr natural nenul, `S`, şi printr-o metodă eficientă din punct de vedere al timpului de executare, determină şi afişează pe ecran trei valori naturale a căror sumă este egală cu `S`, şi al căror produs este maxim. Cele trei valori vor fi scrise în ordine crescătoare pe prima linie a fişierului `rez.dat`, separate prin câte un spaţiu.
Exemplu: dacă se citeşte valoarea `5`, fişierul `rez.dat` va avea o linie cu conţinutul `1 2 2`. **(6p.)**

44. Fişierul `BAC.DAT` conţine pe prima linie, separate printr-un spaţiu, două valori naturale `n` şi `m` (`2≤n≤1000`, `2≤m≤1000`), pe a doua linie `n` valori întregi şi pe a treia linie `m` valori întregi. Valorile de pe a doua şi de pe a treia linie apar în fişier în ordine strict crescătoare, sunt separate prin câte un spaţiu şi au cel mult `4` cifre fiecare.
Se cere afişarea pe ecran a două valori, dintre cele aflate în poziţii consecutive pe a treia linie a fişierului, care determină intervalul închis în care se află un număr maxim de valori de pe a doua linie a fişierului. Se va utiliza o metodă eficientă din punct de vedere al timpului de executare şi al spaţiului de memorie utilizat.

Exemplu: dacă fişierul `BAC.DAT` are conţinutul alăturat, programul va afişa: `1 9`

```
10 4
-1 1 3 4 5 6 10 15 16 117
0 1 9 20
```

Explicaţie: cele patru numere de pe a treia linie a fişierului determină trei intervale: `[0,1]`, `[1,9]`, `[9,20]`; în intervalul `[1,9]` se află `5` valori de pe a doua linie a fişierului, acesta fiind numărul maxim de valori aflate în unul dintre cele trei intervale.

a) Descrieţi succint, în limbaj natural, metoda de rezolvare folosită, explicând în ce constă eficienţa ei (3 – 4 rânduri) **(4p.)**

b) Scrieţi un program **C/C++** care să rezolve problema conform metodei descrise. **(6p.)**

45. Pe prima linie a fişierului text **DATE.TXT** se află două numere naturale nenule **n** şi **m** (**n≤3000, m≤3000**), pe a doua linie un şir de **n** numere naturale, ordonate crescător, având fiecare cel mult 9 cifre, iar pe linia a treia un şir de **m** numere naturale, ordonate descrescător, având fiecare cel mult 9 cifre. Numerele sunt despărțite, în cadrul liniilor, prin câte un spațiu.

a) Scrieți programul C/C++ care citeşte numerele din fişier şi afişează, pe ecran, doar numerele pare din cele două şiruri, ordonate crescător. Alegeți o metodă de rezolvare eficientă ca timp de executare.
Exemplu: dacă fişierul are conținutul alăturat, pe ecran se va afişa: `2 4 4 32 42 42 88 88` **(6p.)**

```
5 8
2 4 7 37 42
88 88 67 45 42 32 4 1
```

b) Descrieți succint, în limbaj natural, metoda utilizată, justificând eficienta acesteia. **(4p.)**

46. **a)** Fişierul **date.in** conține un şir de cel mult **10000** numere naturale cu cel mult **2** cifre fiecare, separate prin câte un spațiu. Scrieți un program C/C++ care citeşte numerele din fişierul **date.in** şi scrie în fişierul text **date.out**, valorile distincte citite, separate prin câte un spațiu, respectându-se regula: pe prima linie vor fi scrise numerele impare în ordine crescătoare, iar pe linia a doua numerele pare, în ordine descrescătoare. Alegeți o metodă eficientă din punctul de vedere al timpului de executare. **(6p.)**
Exemplu: dacă pe prima linie a fişierului **date.in** se află numerele:
`75 12 3 3 18 75 1 3`
atunci fişierul **date.out** va conține:
`1 3 75`
`18 12`

47. Pe prima linie a fişierului text **DATE.TXT** se află un şir de cel mult **10000** de numere întregi, având cel mult **4** cifre fiecare. Numerele sunt despărțite prin câte un spațiu.

a) Scrieți un program C/C++ care citeşte numerele din fişier şi afişează pe ecran lungimea maximă a unei secvențe de numere din şir, cu proprietatea că oricare două numere din secvență, aflate pe poziții consecutive, au parități diferite. Pe a doua linie a ecranului, programul va afişa o secvență de lungime maximă, valorile fiind despărțite prin câte un spațiu. Alegeți o metodă de rezolvare eficientă ca timp de executare.
Exemplu: dacă fişierul conține, în ordine, numerele `2 4 3 2 7 4 6 2 7 8 12`, se va afişa:
`5`
`4 3 2 7 4` **(6p.)**

48. Se consideră şirul **1, 2,1, 3,2,1, 4,3,2,1, ...**
construit astfel: prima grupă este formată din numărul **1**, a doua grupă este formată din numerele **2** şi **1**, iar grupa a **k**-a, este formată din numerele **k, k-1,..., 1.**
Se cere să se citească de la tastatură un număr natural **n** (**n≤1000**) şi să se afişeze pe ecran cel de al **n**-lea termen al şirului dat.

a) Descrieți un algoritm de rezolvare a acestei probleme, eficient din punct de vedere al timpului de executare şi al spațiului de memorie, explicând în ce constă eficiența acestuia. **(4p.)**

b) Scrieți programul C/C++ corespunzător algoritmului descris **(6p.)**

Se consideră un şir **s** format după regula alăturată, unde s-a notat cu **aΘb** numărul obținut prin concatenarea cifrelor lui **a** şi **b**, în această ordine.

$$s_n=\begin{cases} x & \text{dacă } n=1 \\ x+1 & \text{dacă } n=2 \\ s_{n-1}\Theta s_{n-2} & \text{dacă } n>2 \end{cases}$$

Exemplu: pentru **x=2** se obține şirul:

2, 3, 32, 323, 32332,....

Fişierul text **SIR.TXT** conține pe prima linie două numere, **x** (**1≤x≤20**) şi **k** (**1≤k≤5000**), separate printr-un spațiu, iar pe a doua linie un număr format din exact **k** cifre, reprezentând un termen al şirului **s** (diferit de **x**). Cifrele numărului nu sunt separate prin spații.

a) Scrieți un program **C/C++** care, utilizând un algoritm eficient din punct de vedere al timpului de executare şi al memoriei utilizate, afişează pe ecran acel termen din şir care îl precede pe cel citit din fişier.

Exemplu: dacă fişierul conține valorile alăturate, se va afişa pe ecran numărul **323**. **(6p.)**

```
2 5
32332
```

49.

Fişierul text **numere.txt** conține pe prima sa linie un număr natural **n** (**n<30000**), iar pe a doua sa linie, **n** numere întregi, având maximum **4** cifre fiecare. Se cere să se afişeze pe ecran un şir de **n** numere întregi, cu proprietatea că valoarea termenului de pe poziția **i** (**i=1,2,…,n**) din acest şir este egală cu cea mai mare dintre primele **i** valori de pe a doua linie a fişierului **numere.txt**.

a) Descrieți pe scurt un algoritm de rezolvare, eficient din punct de vedere al timpului de executare şi al spațiului de memorie utilizat, explicând în ce constă eficiența sa. **(4p.)**

b) Scrieți programul **C/C++** corespunzător algoritmului descris. **(6p.)**

Exemplu: dacă fişierul **numere.txt** are conținutul alăturat, se afişează pe ecran numerele

```
12
4 6 3 7 8 1 6 2 7 9 10 8
```

```
4 6 6 7 8 8 8 8 8 9 10 10
```

50.

Fişierele text **NR1.TXT** şi **NR2.TXT** conțin, separate prin câte un spațiu, mai multe numere întregi de cel mult **9** cifre fiecare. Fiecare dintre fişiere conține cel mult **100** de valori şi numerele din fiecare fişier sunt ordonate strict crescător. Se cere să se afişeze pe ecran, în ordine crescătoare, numerele divizibile cu **5** care se găsesc doar în unul din cele două fişiere.

Exemplu: dacă fişierul **NR1.TXT** conține numerele **1 2 3 4 7 20 60**, iar fişierul **NR2.TXT** conține numerele **3 5 7 8 9 10 12 20 24**, atunci se vor afişa pe ecran valorile **5 10 60**.

a) Descrieți un algoritm de rezolvare a acestei probleme, eficient din punct de vedere al timpului de executare şi al spațiului de memorie utilizat, explicând în ce constă eficiența acestuia. **(4p.)**

b) Scrieți programul **C/C++** corespunzător algoritmului descris. **(6p.)**

51.

Se citeşte de la tastatură un număr natural **n** (**n≤500**) şi apoi **n** cifre. Se cere să se afişeze pe ecran cele **n** cifre citite, în ordine crescătoare, separate prin câte un spațiu.

Exemplu: pentru **n=19** şi cifrele **3 3 0 9 2 1 2 1 3 7 1 5 2 7 1 0 3 2 3** se va afişa pe ecran **0 0 1 1 1 1 2 2 2 2 3 3 3 3 3 5 7 7 9**.

a) Descrieți pe scurt un algoritm de rezolvare al problemei, eficient din punct de vedere al spațiului de memorie utilizat şi al timpului de executare, explicând în ce constă eficiența metodei alese. **(4p.)**

b) Scrieți programul **C/C++** corespunzător algoritmului descris. **(6p.)**

52.

In fişierul text **BAC.IN** se găsesc, pe o singură linie, separate prin câte un spaţiu, mai multe numere naturale de cel mult 6 cifre fiecare. Se cere să se determine şi să se afişeze pe ecran, separate printr-un spaţiu, ultimele **două** numere prime (nu neapărat distincte) din fişierul **BAC.IN**. Dacă în fişier se găseşte un singur număr prim sau niciun număr prim se va scrie pe ecran mesajul **Numere prime insuficiente**.

Exemplu: dacă fişierul **BAC.IN** conţine valorile: 12 5 68 13 8 17 9 31 42 se va afişa 17 31.

a) Descrieţi în limbaj natural un algoritm eficient, din punct de vedere al spaţiului de memorie şi al timpului de executare, pentru rezolvarea acestei probleme, explicând în ce constă eficienţa acestuia. **(4p.)**

53. **b)** Scrieţi programul C/C++ corespunzător algoritmului descris. **(6p.)**

În fişierul **numere.txt** pe prima linie este memorat un număr natural **n** (n≤10000), iar pe linia următoare un şir de **n** numere naturale distincte două câte două, separate prin câte un spaţiu, cu maximum 4 cifre fiecare. Se cere afişarea pe ecran a poziţiei pe care s-ar găsi primul element din şirul aflat pe linia a doua a fişierului, în cazul în care şirul ar fi ordonat crescător. Numerotarea poziţiilor elementelor în cadrul şirului este de la 1 la **n**. Alegeţi un algoritm de rezolvare eficient din punct de vedere al memoriei utilizate şi al timpului de execuţie.

Exemplu: dacă fişierul **numere.txt** conţine:

```
6
267 13 45 628 7 79
```

se va afişa 5, deoarece primul element din şirul iniţial, 267, s-ar găsi pe poziţia a cincea în şirul ordonat crescător.

a) Descrieţi succint, în limbaj natural, strategia de rezolvare şi justificaţi eficienţa algoritmului ales. **(4p.)**

54.

În fişierul **numere.txt** este memorat un şir de maximum 10000 numere naturale, distincte două câte două, cu maximum 4 cifre fiecare, separate prin câte un spaţiu. Pentru un număr **k** citit de la tastatură, se cere afişarea pe ecran a poziţiei pe care se va găsi acesta în şirul de numere din fişier, dacă şirul ar fi ordonat descrescător, sau mesajul **nu există**, dacă numărul **k** nu se află printre numerele din fişier. Alegeţi un algoritm eficient de rezolvare din punct de vedere al memoriei utilizate şi al timpului de execuţie.

Exemplu: dacă fişierul **numere.txt** conţine numerele 26 2 5 30 13 45 62 7 79, iar **k** are valoarea 13, se va afişa 6 deoarece 13 s-ar găsi pe poziţia a şasea în şirul ordonat descrescător.

a) Descrieţi succinct, în limbaj natural, strategia de rezolvare şi justificaţi eficienţa algoritmului ales. **(4p.)**

55. **b)** Scrieţi programul C/C++ corespunzător algoritmului ales. **(6p.)**

56.

În fiecare dintre fişierele **nr1.txt** şi **nr2.txt** este memorată pe prima linie câte o valoare naturală **n** de cel mult 8 cifre, iar pe linia următoare sunt memorate câte **n** numere naturale, cu maximum 4 cifre fiecare, ordonate strict crescător şi separate prin câte un spaţiu. Se cere afişarea pe ecran, separate prin câte un spaţiu, în ordine strict crescătoare, a tuturor numerelor aflate pe a a doua linie în cel puţin unul dintre cele două fişiere. În cazul în care un număr apare în ambele fişiere, el va fi afişat o singură dată. Alegeţi un algoritm de rezolvare eficient din punct de vedere al memoriei utilizate şi al timpului de execuţie.
Exemplu: pentru următoarele fişiere:

nr1.txt
```
5
3 6 8 9 12
```

nr2.txt
```
6
2 3 5 7 9 13
```

se va afişa **2 3 5 6 7 8 9 12 13**.

a) Descrieţi succinct, în limbaj natural, strategia de rezolvare şi justificaţi eficienţa algoritmului ales. **(4p.)**

b) Scrieţi programul C/C++ corespunzător algoritmului ales. **(6p.)**

57.

În fiecare dintre fişierele **nr1.txt** şi **nr2.txt** este memorată pe prima linie câte o valoare naturală **n** de cel mult 8 cifre, iar pe linia următoare sunt memorate câte **n** numere naturale, cu maximum 4 cifre fiecare, ordonate strict crescător şi separate prin câte un spaţiu. Se cere afişarea pe ecran, separate prin câte un spaţiu, în ordine strict crescătoare, a tuturor numerelor aflate pe a a doua linie atât în primul cât şi în al doilea fişier. Alegeţi un algoritm de rezolvare eficient din punct de vedere al memoriei utilizate şi al timpului de execuţie.
Exemplu: pentru următoarele fişiere:

nr1.txt
```
5
3 6 8 9 12
```

nr2.txt
```
6
2 3 5 7 9 13
```

se va afişa **3 9**.

a) Descrieţi succinct, în limbaj natural, strategia de rezolvare şi justificaţi eficienţa algoritmului ales. **(4p.)**

b) Scrieţi programul C/C++ corespunzător algoritmului ales. **(6p.)**

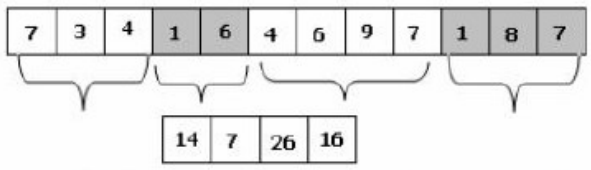
58.

Fişierul **NUMERE.IN** conţine pe prima linie un număr natural nenul **n** ($2 \le n \le 100$) şi pe următoarea linie **n** numere reale pozitive în ordine strict crescătoare separate prin câte un spaţiu.

a) Scrieţi un program C/C++ care, utilizând un algoritm eficient din punct de vedere al memoriei utilizate, determină şi afişează pe ecran cel mai mare număr natural **x** cu proprietatea că în orice interval deschis având capete oricare două dintre cele **n** numere aflate pe linia a doua în fişierul **NUMERE.IN** se găsesc cel puţin **x** numere întregi. Numărul astfel determinat se afişează pe ecran.
Exemplu: dacă fişierul **NUMERE.IN** are conţinutul:
```
6
3.5 5.1 9.2 16 20.33 100
```
atunci se afişează **2** pentru că în oricare dintre intervalele (3.5;5.1), (3.5;9.2), (3.5;16), (3.5;20.33), (3.5;100), (5.1;9.2), (5.1;16), (5.1;20.33) (5.1;100), (9.2;16), (9.2;20.33), (9.2;100), (16;20.33), (16;100), (20.33;100) există cel puţin două numere întregi. **(6p.)**

Se consideră două tablouri unidimensionale **A** şi **B** cu elemente numere naturale din intervalul `[1;10000]`. Spunem că tabloul **A** "**se poate reduce**" la tabloul **B** dacă există o împărțire pe secvențe de elemente aflate pe poziții consecutive în tabloul **A** astfel încât prin înlocuirea secvențelor cu suma elementelor acestora să se obțină, în ordine, elementele tabloului **B**.

De exemplu tabloul



se poate reduce la tabloul

Fişierul **NUMERE.IN** conține pe prima linie două numere naturale nenule **n** şi **m**(1≤n≤100), pe linia a doua **n** numere naturale din intervalul `[1;10000]` şi pe linia a treia alte **m** numere naturale din intervalul `[1;10000]`.Pe fiecare linie numerele sunt separate prin câte un spațiu.

a) Scrieți un program C/C++ care citeşte cele două numere naturale **n** şi **m** din fişierul **NUMERE.IN**, construieşte în memorie două tablouri unidimensionale **A** şi **B** cu elementele aflate în fişier pe a doua, respectiv a treia linie şi verifica, utilizând un algoritm eficient din punct de vedere al timpului de executare, dacă tabloul **A** se poate reduce la tabloul **B**. Programul afişează pe ecran mesajul **DA** în caz afirmativ şi mesajul **NU** în caz negativ. **(6p.)**

59. Fişierul **NUMERE.IN** conține pe prima linie un număr natural nenul **n**(1≤n≤100) şi pe următoarea linie **n** numere reale pozitive **ordonate crescător**, separate prin câte un spațiu.

a) Scrieți un program C/C++ care citeşte din fişierul **NUMERE.IN** numărul natural **n**, şi determină, utilizând un algoritm eficient din punct de vedere al timpului de executare şi al memoriei utilizate, numărul **minim** de intervale închise de forma `[x;x+1]`, cu **x** număr natural, a căror reuniune include toate numerele reale din fişier.

Exemplu: Dacă fişierul **NUMERE.IN** are conținutul:
6
`2.3 2.8 5.1 5.7 5.9 6.3` atunci se afişează `3` (intervalele `[2;3],[5;6],[6;7]` sunt cele **3** intervale de forma cerută care conțin numere din şir). **(6p.)**

60. În fişierul **numere.txt**, se află memorate, pe prima linie un număr natural **n** (1≤n≤100), iar pe fiecare dintre următoarele **n** linii câte două numere întregi **x,y** (-100≤x≤y≤100) reprezentând capetele câte unui segment `[x,y]` desenat pe axa **Ox** de coordonate.

a) Scrieți în limbajul C/C++ un program eficient din punct de vedere al timpului de executare şi al spațiului de memorare, care citeşte din fişier datele existente, determină segmentul rezultat în urma intersecției tuturor celor **n** segmente date şi afişează pe ecran două numere despărțite printr-un spațiu ce reprezintă capetele segmentului cerut. Dacă segmentele nu au nici un punct comun se va afişa pe ecran valoarea 0. **(6p.)**

b) Descrieți în limbaj natural algoritmul utilizat, justificând eficiența acestuia. **(4p.)**

Exemplu: dacă fişierul **numere.txt** are conținutul alăturat, se va afişa pe ecran
`3 5`

```
5
-7 10
3 20
-5 5
0 12
-8 30
```

61.