Varianta 1

TOATE problemele trebuie însoţite de demonstraţia scrisă a corectitudinii şi complexităţii

1. **Acoperire** - Se dau un interval închis [a,b] si o mulţime de alte n intervale închise [ai,bi], $1 \le i \le N$. Scrieţi un program care calculează si afişează o submulţime de cardinal minim de intervale închise din mulţimea data cu proprietatea ca prin reuniunea acestora se obţine un interval care include pe [a,b]. Dacă prin reuniunea tuturor intervalelor nu putem obţine un interval care sa includă intervalul [a,b], se va afişa -1. Exemplu: [A,B]=[10,20], N= 5 si intervalele [10,12], [4,15], [5, 11], [14,18], [14,21] soluţia are cardinal 2: [4,15], [14,21] - **O(n logn)** **http://varena.ro/problema/acoperire** **(1p)**

Indicaţie demonstraţie corectitudine – v. pb spectacolelor curs + [1]

date.in	date.out
10 20 7 10 12 4 15 3 16 5 11 13 18 14 20 15 21	3 16 14 20

2. Planificare cu minimizarea întârzierii maxime – Se consideră o mulţime de n activităţi care trebuie planificate pentru a folosi o aceeaşi resursă. Această resursă poate fi folosită de o singură activitate la un moment dat. Pentru fiecare activitate i se cunosc durata l_i şi termenul limită până la care se poate executa t_i (raportat la ora de început 0). Dorim să planificăm aceste activităţi astfel încât întârzierea fiecărei activităţi să fie cât mai mică. Mai exact, pentru o planificare a acestor activităţi astfel încât activitatea *i* este programată în intervalul de timp $[s_i, f_i)$, definim întârzierea activităţii *i* ca fiind durata cu care a depăşit termenul limită: $\boldsymbol{p_i = \max\{0, f_i - t_i\}}$.

Întârzierea planificării se defineşte ca fiind **maximul întârzierilor activităţilor**:

Exemplu. Pentru n = 3 şi $l_1 = 1$, $t_1 = 3$ / $l_2 = 2$, $t_2 = 2$ / $l_3 = 3$, $t_3 = 3$

o soluţie optimă se obţine dacă planificăm activităţile în ordinea 2, 3, 1; astfel:

- activitatea 2 în intervalul [0, 2) – întârziere 0
- activitatea 3 în intervalul [2, 5) – întârziere $5 - t_3 = 5 - 3 = 2$
- activitatea 1 în intervalul [5, 6) – întârziere $6 - t_1 = 6 - 3 = 3$

Întârzierea planificării este $\max\{0, 2, 3\}=3$ $\boldsymbol{P = \max\{p_1, p_2, ..., p_n\}}$

a) Să se determine o planificare a activităţilor date care să aibă întârzierea P minimă. Se vor afişa pentru fiecare activitate intervalul de desfăşurare şi întârzierea – **O(n log n)**

date.in	date.out
3 1 3 2 2 3 3	5 6 3 0 2 0 2 5 2 Propunere: 2: 0 2 0 3: 5 6 3 1: 2 5 2 Intarziere 3

b) Este corect un algoritm greedy bazat pe următoarea idee: planificăm activităţile în ordine crescătoare în raport l_i (adică în raport cu durata)? Justificaţi.

c) Este corect un algoritm greedy bazat pe următoarea idee: planificăm activităţile în ordine crescătoare în raport cu diferenţa t_i-l_i (adică în raport cu timpul maxim la care trebuie să înceapă activitatea i pentru a respecta termenul limită)? Justificaţi. **(2p)**

3. **Descompunere în subşiruri descrescătoare**: Se dă un şir de numere naturale. Să se descompună şirul de numere într-un număr minim de subşiruri descrescătoare (nestrict) şi să se afişeze aceste subşiruri. Exemplu: şirul 11, 13, 10, 15, 12, 7 se poate descompune în 3 subşiruri descrescătoare, spre exemplu: 11, 10, 7 / 13, 12 / 15

a) Implementaţi un algoritm – **O(n log n)** pentru rezolvarea acestei probleme. (Indicaţie pentru demonstrarea corectitudinii: Numărul optim de subşiruri este egal cu lungimea maximă a unui subşir crescător al şirului iniţial).

date.in	date.out
6 11 13 10 15 12 7	11 10 7 13 12 15

b) Este corect următorul algoritm greedy pentru rezolvarea acestei probleme? Justificați

Cât timp mai sunt elemente în șir repetă:

- construim un subșir descrescător care începe cu primul element din șir, adăugând pe rând la subșir următorul element din șir care se poate adăuga (mai mic sau egal cu ultimul element adăugat la subșir)
- afișează acest subșir și elimină-l din șirul inițial

c) Implementaţi Patience Sorting – o metodă de sortare bazată pe împărţirea unui şir în subşiruri descrescătoare, memorate ca stive (după această împărţire se repetă de n ori următoarea operaţie: mută cea mai mică valoare din topul unei stive în vectorul rezultat, iniţial vid) - **O(n log n).** Pentru mai multe detalii şi imagini sugestive pentru înțelegerea problemei, dar şi legătura cu problema determinării celui mai lung subşir crescător al unui şir – pe care o vom studia la programare dinamică – se pot consulta spre exemplu:

https://en.wikipedia.org/wiki/Patience_sorting

https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf **(3p)**

Varianta 2

TOATE problemele trebuie însoţite de demonstraţia scrisă a corectitudinii şi complexităţii

1. Se dau n cuburi cu laturile **diferite două câte două**. Fiecare cub are o culoare, codificată cu un număr de la 1 la p (p dat). Să se construiască un turn de înălţime maximă de cuburi în care un cub nu poate fi aşezat pe un cub de aceeaşi culoare sau cu latură mai mică decât a sa – **O(n logn).** Justificaţi corectitudinea algoritmului propus. În cazul în care lungimile laturilor cuburilor nu erau diferite mai este valabilă metoda propusă? Justificaţi. Exemplu: pentru n=4, p=2 şi cuburi cu caracteristicile latură/culoare: 5/1, 10/1, 9/1, 8/2, o soluţie este turnul cu cuburile 10/1, 8/2, 5/1. Se aleg 3 cuburi, cuburile 2 4 și 1 (în ordinea datelor de intrare) și se obține înălțimea 23. **(1p)**

date.in	date.out
4 2 5 1 10 1 9 1 8 2	3 23 2 4 1

2. Memorarea textelor pe banda – variante

a) n texte cu lungimile L(1),...,L(n) urmează a fi aşezate pe o bandă. Pentru a citi textul de pe poziţia k, trebuie citite textele de pe poziţiile 1,2,...,k. Pentru fiecare text i se cunoaşte şi frecvenţa f(i) cu care acesta este citit. Să se determine o modalitate de aşezare a textelor pe bandă astfel încât timpul total de acces să fie minimizat. Timpul total de acces pentru o aşezare a textelor corespunzătoare unei permutări p se defineşte ca fiind $\sum_{i=1}^{n} f(p_i)[L(p_1)+...+L(p_i)]$ **O(n log n)**

Problemă echivalentă - programarea taskurilor cu priorităţi diferite. n activităţi cu duratele L(1),...,L(n) urmează a fi programate pentru executare (secvenţială) folosind o aceeaşi resursă. Fiecare activitate i are asociată o prioritate w(i). O ordine în care se execută aceste activităţi se poate reprezenta printr-o permutare $p \in S_n$. În raport cu această ordine, a i-a activitate executată are timpul de terminare $L(p_1)+...+L(p_i)$. Să se determine o ordine în care trebuie planificate activităţile astfel încât să fie minimizată suma ponderată a timpilor de finalizare a acţiunilor, dată de formula $\sum_{i=1}^{n} w(p_i)[L(p_1)+...+L(p_i)]$

b) n texte cu lungimile L(1),...,L(n) urmează a fi aşezate pe p benzi (p dat). Pentru a citi textul de pe poziţia k de pe o banda, trebuie citite textele de pe poziţiile 1,2,...,k de pe aceea bandă. Fiecare text are aceeaşi frecvenţă de citire. Să se determine o modalitate de aşezare a textelor pe cele p benzi astfel încât timpul total de acces să fie minimizat (=suma timpului total de acces pentru fiecare din cele p benzi). **O(n log n) (2p)**

3. Maximizarea profitului cu respectarea termenelor limită Se consideră o mulţime de n activităţi care trebuie planificate pentru a folosi o aceeaşi resursă. Această resursă poate fi folosită de o singură activitate la un moment dat. Toate activităţile au aceeaşi durată (să presupunem 1). Pentru fiecare activitate i se cunosc termenul limită până la care se poate executa t_i (raportat la ora de început 0, $1 \leq t_i \leq n$) şi profitul p_i care se primeşte dacă activitatea i se execută la timp (cu respectarea termenului limită). Se cere să se determine o submulţime de activităţi care se pot planifica astfel încât profitul total obţinut să fie maxim.

Exemplu. Pentru n = 4 şi

$p_1 = 4,\ t_1 = 3$

$p_2 = 1,\ t_2 = 1$

$p_3 = 2,\ t_3 = 1$

$p_4 = 5,\ t_4 = 3$

o soluţie optimă se obţine dacă planificăm activităţile în ordinea 3, 4, 1, profitul obţinut fiind

$p_3 + p_4 + p_1 = 2 + 5 + 4 = 11$. **O(n logn) (3p)**

date.in	date.out
4 4 3 1 1 2 1 5 3	11 3 4 1

Varianta 3

TOATE problemele trebuie însoţite de demonstraţia scrisă a corectitudinii şi complexităţii

1. Considerăm următorul **joc pentru două persoane.** Tabla de joc este o secvenţă de n numere întregi pozitive, iar cei doi jucători mută alternativ. Când un jucător mută, el selectează un număr ori de la stânga ori de la dreapta secvenţei. Numărul selectat este şters de pe tablă. Jocul se termină când toate numerele au fost selectate. Primul jucător câştigă dacă suma numerelor pe care le-a selectat este **cel puţin egală** cu suma selectată de al doilea jucător. Al doilea jucător joacă cât de bine poate. Primul jucător începe jocul. Ştim că tablă se află la început un număr **par** de elemente n. Să se scrie un program astfel încât, indiferent cum va juca al doilea jucător, primul jucător câştigă. Scrieţi programul astfel încât primul jucător să mute cu ajutorul programului, iar calculatorul să mute aleator de la stânga sau de la dreapta. La ieşire se va scrie suma obţinută de primul jucător, suma obţinută de cea de al doilea şi secvenţele de mutări sub forma unor şiruri cu caracterele S pentru stânga şi D pentru dreapta. **Exemplu**: pentru tabla cu numerele 2 1 4 3 o soluţie câştigătoare este următoarea:

   Pasul 1 - primul jucător alege S (valoarea 2); rămân pe tablă 1 4 3

   Pasul 2 - calculatorul are două posibilităţi: S (valoarea 1) sau D (valoarea 3).

   Pasul 3 – dacă la pasul 2 calculatorul a ales S, atunci primul jucător alege S (valoarea 4);
   dacă la pasul 2 calculatorul a ales D, atunci primul jucător alege D (valoarea 4);

   Pasul 4 – pe tablă a mai rămas doar o valoare (3 respectiv 1, în funcţie de alegerea de la pasul 2), pe care o alege calculatorul

Astfel, primul jucător a adunat suma 2+4, iar calculatorul suma 1+3 (respective 3+1), deci primul jucător a câştigat - **O(n) (1p)**

2. Dat un arbore cu n vârfuri, să se determine o mulţime de vârfuri neadiacente de cardinal maxim (o submulţime independentă maximă a mulţimii vârfurilor). **Exemplu**: Pentru un arbore cu 8 vârfuri şi muchiile {(1,2),(1,3),(2,4),(2,5),(3,6),(3,7),(5,8)} se va afişa mulţimea {1,4,6,7,8}, nu neapărat în această ordine. Pentru un graf oarecare mai este valabilă metoda? Justificaţi. – **O(n) (2p)**

date.in	date.out
8 7 1 2 1 3 2 4 2 5 3 6 3 7 5 8	5 1 4 6 7 8

3. Problema partiţionării intervalelor – Se consideră n intervale închise (interpretare: n cursuri, pentru care se cunosc ora de început şi ora de sfârşit). Se cere să se împartă (partiţioneze) această mulţime de intervale într-un **număr minim de submulţimi** cu proprietatea că oricare două intervale dintr-o submulţime nu se intersectează şi să se afişeze aceste submulţimi (interpretare: să se determine numărul minim de săli necesare pentru a putea programa aceste cursuri în aceeaşi zi şi afişaţi o astfel de programare). Care dintre următorii algoritmi Greedy sunt corecţi pentru a rezolva această problemă? Pentru fiecare algoritm (subpunct) justificaţi corectitudinea sau daţi un contraexemplu:

a) Se sortează intervalele crescător după extremitatea iniţială Pentru fiecare interval I în această ordine execută: se adaugă I la o submulţime deja construită, dacă se poate (nu se intersectează cu niciun interval din ea), altfel se creează o nouă submulţime cu intervalul I.

b) Se sortează intervalele crescător după extremitatea finală. Pentru fiecare interval I în această ordine execută: se adaugă I la o submulţime deja construită, dacă se poate, altfel se creează o nouă submulţime cu intervalul I.

c) Se sortează intervalele crescător după extremitatea finală Pentru fiecare interval I în această ordine execută: se adaugă I la o mulţime deja construită, dacă se poate, altfel se creează o nouă mulţime cu intervalul I. Dacă există mai multe mulţimi la care se poate adăuga I, se alege acea submulţime care conţine intervalul cu extremitatea finală cea mai mare (care se termină cât mai aproape de începutul intervalului I)

d) Cât timp mai sunt intervale nedistribuite în submulţimi repetă: construieşte o submulţime de cardinal maxim de intervale disjuncte din mulţimea de intervale (folosind algoritmul de la problema spectacolelor discutată la curs) şi se elimină aceste intervale din mulţimea de intervale (altfel spus, se construieşte prima submulţime folosind un număr maxim de intervale, apoi a doua submulţime folosind un număr maxim de intervale din cele rămase etc).

e) Implementaţi un algoritm **O(n log n)** pentru rezolvarea acestei probleme.

Exemplu: pentru n=3 cursuri, care trebuie să se desfăşoare în intervalele: [10, 14], [12, 16], respectiv [15, 18], sunt necesare 2 săli, o programare optimă fiind:

- **Sala 1**: [10, 14] – activitatea 1, [15, 18] – activitatea 3
- **Sala 2**: [12, 16] – activitatea 2 **(3p)**

date.in	date.out
4 10 14 12 16 15 18	2 1 3 2