Programarea algoritmilor
– SEMINAR NR. 6 –

Metoda Greedy

1. Se consideră n șiruri de numere sortate crescător. Știind faptul că interclasarea a două șiruri de lungimi p și q are complexitatea O(p+q), să se determine o modalitate de interclasare a celor n șiruri astfel încât complexitatea totală să fie minimă.
   Exemplu: Fie 4 șiruri sortate crescător având lungimile 30, 20, 10 și 25. Interclasarea primelor două șiruri necesită 30+20=50 de operații elementare și vom obține un nou șir de lungime 50, deci mai trebuie să interclasăm 3 șiruri cu lungimile 50, 10 și 30. Interclasarea primelor două șiruri necesită 50+10=60 de operații elementare și numărul total de operații elementare devine 50+60=110, după care vom obține un nou șir de lungime 60, deci mai trebuie să interclasăm două șiruri cu lungimile 60 și 25. Interclasarea celor două șiruri necesită 60+25=85 de operații elementare, iar numărul total de operații elementare devine 110+85=195, și vom obține șirul final de lungime 85.

2. **Planificarea unor proiecte cu profit maxim –** complexitate O(n*log(n))
   Se consideră n proiecte, pentru fiecare proiect cunoscându-se profitul, un termen limită (sub forma unei zi din lună) și faptul că implementarea sa durează exact o zi. Să se găsească o modalitate de planificare a unor proiecte astfel încât profitul total să fie maxim.

   Exemplu:

proiecte.in			**proiecte.out**
BlackFace	2	800	Ziua 1: BestJob 900.0
Test2Test	5	700	Ziua 2: FileSeeker 950.0
Java4All	1	150	Ziua 3: JustDoIt 1000.0
BestJob	2	900	Ziua 5: Test2Test 700.0
NiceTry	1	850	
JustDoIt	3	1000	Profit maxim: 3550.0
FileSeeker	3	950	
OzWizard	2	900	

Metoda Divide et Impera

3. Se consideră un șir ls format din n valori egale cu 0, urmate de valori egale cu 1 (este posibil ca în șir să nu existe nicio valoare egală cu 0 sau nicio valoare egală cu 1). Scrieți o funcție care să furnizeze, cu o complexitate minimă, poziția primului 1 din șirul ls sau valoarea -1 dacă nu există nicioi valoare egală cu 1 în șir.
   Exemple:
   ls = [0,0,0,0,1,1,1] => 4
   ls = [0,0] => -1
   ls = [1,1,1] => 0

   ***Indicație de rezolvare*:** Se va folosi o variantă modificată a algoritmului de căutare binară.

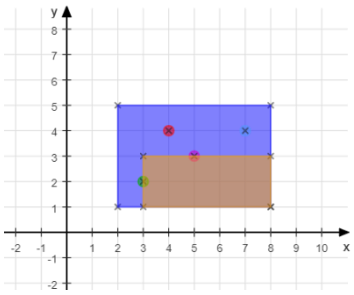
TEMĂ: Având un șir format din n numere întregi sortate crescător, să se găsească numărul de apariții în șir ale unei valori v date.
Exemplu: s = [1,1,2,2,2,2,3,4,4,4,4,5] , v = 2 => numărul de apariții = 4

Indicație de rezolvare: Scriem două funcții bazate pe căutarea binară, una pentru a determina prima poziție pe care apare valoarea v în șirul dat și una pentru a determina ultima poziție pe care apare valoarea v în șir. Apelăm prima funcție și, dacă valoarea v apare în șirul dat, apelăm și a doua funcție, după care calculăm diferența dintre cele două poziții.

Algoritm de complexitate O(log(n)): https://www.geeksforgeeks.org/count-number-of-occurrences-or-frequency-in-a-sorted-array/

4. **Problema debitării**
Se consideră o placă de tablă de formă dreptunghiulară având colțul stânga-jos în punctul (xst, yst) și colțul dreapta-sus în punctul (xdr, ydr). Placa are pe suprafața sa n găuri cu coordonate numere întregi. Știind că sunt permise doar tăieturi orizontale sau verticale complete, se cere să se decupeze din placă o bucată de arie maximă fără găuri.

Exemplu:

placa.in	placa.out	Explicație
2 1 8 5 3 2 4 4 5 3 7 4	Dreptunghiul: 3 1 8 3 Aria maximă: 10	Placa de tablă este un dreptunghi având colțul stânga-jos de coordonate (2,1) și colțul dreapta-sus de coordonate (8,5). În placă sunt date 4 găuri, având coordonatele (3,2), (4,4), (5,3) și (7,4). Dreptunghiul cu suprafața maximă de 10 și care nu conține nicio gaură are coordonatele (3,1) pentru colțul stânga-jos și (8,3) pentru colțul dreapta-sus. 

Observație: Pentru rezolvarea completă vezi laborator 6.

- Backtracking -

5. Descompunerea unui număr natural ca sumă de numere naturale nenule.

Variante (exemple pentru n = 4)**:**
a) toate descompunerile posibile (1+1+1+1, 1+1+2, 1+2+1, 1+3, 2+1+1, 2+2, 3+1, 4)
b) descompuneri distincte, respectiv nu au aceiași termeni în altă ordine (1+1+1+1, 1+1+2, 1+3, 2+2, 4) => elementele soluției vor fi păstrate în ordine crescătoare => vom inițializa elementul curent sol[k] al soluției cu valoarea elementului anterior sol[k-1]
c) descompuneri distincte cu termeni distincți (1+3, 4) => elementele soluției vor fi păstrate în ordine strict crescătoare:
d) descompuneri cu termeni distincți (1+3, 3+1, 4) => verificăm $X[k] \notin X[0:k]$ (la fel ca în problema generării permutărilor):
e) soluție cu lungime dată: $\leq p, == p, \geq p$ (pentru $p = 3$: 1+1+2, 1+2+1, 2+1+1)

6. Se consideră n spectacole pentru care se cunosc intervalele de desfășurare. Să se găsească toate planificările cu număr maxim de spectacole care se pot efectua într-o singură sală astfel încât, în cadrul fiecărei planificări, spectacolele să nu se suprapună.

Exemplu:

spectacole.in	**spectacole.out**
10:00-11:20 Scufita Rosie 09:30-12:10 Punguta cu doi bani 08:20-09:50 Vrajitorul din Oz 11:30-14:00 Capra cu trei iezi 12:10-13:10 Micul Print 14:00-16:00 Povestea porcului 15:00-15:30 Frumoasa si Bestia	08:20-09:50 Vrajitorul din Oz 10:00-11:20 Scufita Rosie 12:10-13:10 Micul Print 15:00-15:30 Frumoasa si Bestia 08:20-09:50 Vrajitorul din Oz 10:00-11:20 Scufita Rosie 12:10-13:10 Micul Print 14:00-16:00 Povestea porcului 08:20-09:50 Vrajitorul din Oz 10:00-11:20 Scufita Rosie 11:30-14:00 Capra cu trei iezi 15:00-15:30 Frumoasa si Bestia 08:20-09:50 Vrajitorul din Oz 10:00-11:20 Scufita Rosie 11:30-14:00 Capra cu trei iezi 14:00-16:00 Povestea porcului

Indicație de rezolvare:

a) Folosind metoda Greedy, calculăm numărul maxim de spectacole (nms) care se pot planifica în sală fără suprapuneri.

b) Folosind metoda Backtracking, generăm aranjamente de n luate câte nms.