Metoda Divide Et Impera

PROBLEMA 1

Aplicatie MergeSort: Numarul de inversiuni dintr-un vector

O inversiune intr-un vector $v = (v_1, \dots, v_n)$ perechile (v_i, v_j) cu $i < j$ si $v_i > v_j$.
Sa se numere cate inversiuni contine vectorul v.

v = (9 , 6, 12, 3, 5, 8, 4, 7, 1, 2).
Numarul total de inversiuni din v este 8 + 5 + 7 + 2 + 3 + 4 + 2 + 2 + 0 + 0 = 33

Solutie complexitate timp O(n log n): T(n) = 2T(n/2) + O(n)

Pasul 1 impartirea problemei initiale in subprobleme:
Se imparte vectorul in doi vectori $vs = (v_1, \dots, v_m)$ si $vd = (v_{m+1}, \dots, v_n)$, unde $m = \lfloor n/2 \rfloor$.

vs = (9 , 6, 12, 3, 5)
vs = (8 , 4, 7, 1, 2)

Pasul 2 rezolvarea subproblemelor
Se obtin nrs si nrd numarul de inversiuni din vs respectiv din vd. Pentru exemplul de mai sus
nrs = 3 + 2 + 2 + 0 + 0 = 7
nrd = 4 + 2 + 2 + 0 + 0 = 8

Pasul 3 recombinarea rezultatelor Numarul total de inversiuni este $nrs + nrd + nrm$, unde nm este numarul de inversiuni care se formeaza cu perechi (a, b) unde $a \in vs$ si $b \in vd$.

Cum calculam eficient nrm?
O(n) – costul interclasarii lui vs si vd (presupunand ca acestea au fost sortate)

Este suficient sa numaram in cate inversiuni apare fiecare element din vs.
Se observa ca 9 este mai mare decat toate cele 5 elemente ale lui vd.

vs = (9 , 6, 12, 3, 5)
vd = (8 , 4, 7, 1, 2)

(9,8) (9,4) … => 5 inversiuni

7 este mai mic decat 2 valori din vs. Am obtine acelasi numar de inversiuni in care apare 7 indiferent de pozitia unde ar fi acesta sau valorile 9,12 in vs respectiv in vd.

vs = (….., 9, ….. 12 , …..)
vd = (….., 7, ……) => 2 inversiuni.

Obs: In plus fata de calculul valorilor nrs si nrd la rezolvarea subproblemelor presupune si sortarea vectorilor vs si vd.

Numarul de inversiuni in care apare un element x din vd, se calculeaza in timpul interclasarii lui vs cu vd, atunci cand elementul x este adaugat in vectorul rezultat dupa cum se arata in cele ce urmeaza.

Presupunem ca se ajunge la pasul in care se compara un element x de pe pozitia i din vs cu un element y de pe pozitia j din vd.

vs = (A x B)

vd = (C y D)

Daca y < x, y este adaugat in vectorul rezultat iar j = j + 1. Vectorul sortat va contine dupa acest pas (interclasare(A, C) , y).

Elementul y nu a fost adaugat pana la acest pas deoarece are valoarea mai mare decat toate valorile din subvectorul A si mai mica decat toate elementele din subvectorul B. Astfel ca sa numaram toate inversiunile in care apare y e suficient sa numaram cate elemente are subvectorul D (adica j - 1).

vs = (3, 5, 6, 9, 10)
vs = (1, 2, 4, 7, 8)

Dupa 6 pasi ai algoritmului de interclasare

V = (1 , 2 , 3 , 4, 5, 6) iar i = 4 (vs[i] = 9) si j=4 (vs[j] = 7).
La acest pas este adaugat 7 in vectorul final si vom numara inversiunile in care apare acesta
(m – i + 1= 2 inversiuni)

Alternativ se pot numara inversiunile in care apare fiecare element din subvectorul stang.

Implementare: merge sort, implementare alternativa: cu arbori de intervale (vezi laborator).

PROBLEMA 2

Aplicatie QuickSort: Mediana unui vector (al k-lea minim)

Dat un vector a de n numere şi un indice $k, 1 \leq k \leq n$, sa se determine al k-lea cel mai mic element din vector.

A i -a statistică de ordine a unei mulțimi de n elemente este al k-lea cel mai mic element.

Minimul = prima statistică de ordine
Maximul = a n-a statistică de ordine

Mediana = o valoare v cu proprietatea că numărul de elemente din mulțime mai mici decât v este egal cu numărul de elemente din mulțime mai mari decât v. Dacă n este impar, atunci mediana este a $\lfloor n/2 \rfloor$-a statistică de ordine, altfel, prin convenție mediana este media aritmetică dintre a $\lfloor n/2 \rfloor$-a statistică și a ($\lfloor n/2 \rfloor$+1)-a statistică de ordine.

Solutie complexitate timp mediu de executie O(n)

Pasul 1 impartirea problemei initiale in subprobleme:

Se alege un element *x* (pivot) se gaseste pozitia *m* acestuia in vectorul sortat:

- toate elementele aflate la stânga poziţiei *m* vor fi mai mici decât *x*;
- toate elementele aflate la dreapta poziţiei m vor fi mai mari decât *x*.

Astfel daca *m = k* pivotul ales este al k-lea minim.

Pasul 2 rezolvarea subproblemelor

Daca *m < k* al *k*-lea minim se afla intre elementele situate in dreapta pivotului, intre pozitiile [*m+1*,*n*]

Daca *m > k* al k-lea minim se afla intre elementele situate in stanga pivotului, intre pozitiile [*1*,*m-1*]

```
function kmin(p,u)
      m = pozRand(p,u);
      if (m = k) return a[m]
            if(m < k)
                  return kmin(m+1,u)
            else
                  return kmin(p,m-1)
      end
```

Timpul mediu de executare al acestui algoritm este O(n). Mai exact se poate demonstra prin inducție că pentru o constantă c suficient de mare avem T(n) < cn.

PROBLEMA 3

Mediana vectorului obtinut prin interclasarea a doi vectori sortati

Se dau doi vectori a și b de lungime n, cu elementele ordonate crescător. Să se determine mediana vectorului obținut prin interclasarea celor doi vectori.

Solutie complexitate timp O(log n) fara a interclasa vectorii

Nu este necesar sa efectuăm procedura de interclasare care are complexitatea: O(n).

Vom compara medianele celor doi vectori. În funcție de rezultat, problema se reduce la aceeași problemă pentru subvectori ai vectorilor inițiali.

Astfel, fie **m1** mediana vectorului a și **m2** mediana vectorului b

Observație. Mediana unui vector sortat cu n elemente se poate calcula în timp constant, fiind chiar elementul de pe poziția $\lfloor n/2 \rfloor$ (pozițiile sunt numerotate de la 0), dacă n este impar, sau media aritmetică a elemetelor de pe pozițiile $\lfloor n/2 \rfloor - 1$ și $\lfloor n/2 \rfloor$, dacă n este par

- Dacă m1 = m2 atunci această valoare este mediana
- Dacă m1 > m2 atunci mediana se află în unul din subvectorii a[0.. $\lfloor n/2 \rfloor$], b[$\lfloor (n-1)/2 \rfloor \,..\, n-1$]
- Dacă m1 < m2 atunci mediana se află în unul din subvectorii a[$\lfloor n/2 \rfloor + 1, .. \, n-1$], b[$0 \,..\, \lfloor (n-1)/2 \rfloor - 1$]

Mediana noii probleme = mediana problemei iniţiale deoarece se elimina (n-1)/2 elemente mai mici decât mediana şi tot (n-1)/2 elemente mai mari decât mediana.

Dacă vectorii au cel mult două elemente, atunci problema se poate rezolva direct.

Idee de rezolvare in cazul in care **dimensiunile celor doi vectori nu sunt egale: cautare binara**.

a = [4 5 6 8 9 10] n = 6

b = [1 3 5 7 12] m = 5

Daca a[4] = 8 (elementul situat la mijlocul lui a) ar fi mediana inseamna ar trebui sa fie mai mare decat (n + m)/2 = 5 elemente din vectorul obtinut prin interclasare. Este mai mare decat 3 = i-1 (i = 4, poz pe care se afla 8) elemente din a, inseamna ca ar trebui sa fie mai mare decat 2 elemente din b. Pentru a verifica acest fapt:

Comparam a[4] cu b[2] si b[3] timp constant: b[2] = b[(n+m)/2 – i + 1]

8 > 3, 8>6. deci exista mai mult de 5 valori mai mici decat 8, deci trebuie mediana este o valoare mai mica decat 8 si va fi cautata in subvectorul 4 5 6.

PROBLEMA 4

(**Cele mai apropiate puncte in plan**) Fiind date *n* puncte in plan, prin coordonatele lor (x_i, y_i) sa se gaseasca cea mai apropiata pereche de puncte.

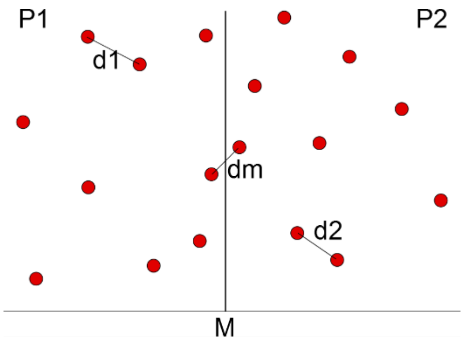
Solutie complexitate timp O(n log n): T(n) = 2T(n/2) + O(n)

Pasul 1 impartirea problemei initiale in subprobleme:

Se imparte multimea de puncte in doua submultimi P1 si P2 cu numar egal (n/2) de puncte. Se cauta mediana M. Dreapta verticala x = M va fi cea va delimita cele doua multimi de puncte. Complexitatea acestui pas este O(n).

Pasul 2 rezolvarea subproblemelor Se gaseste cea mai apropiata pereche de puncte din P1 intre care exista distanta d_1, cea mai apropiata pereche de puncte din P2, intre care exista distanta d_2. Problema initiala a fost impartita in doua subprobleme iar timpul de executie va fi

T(n) = 2T(n/2) + f(n), unde f(n) este timpul necesar recombinarii rezultatelor obtinute (timpul pentru pasul 3).



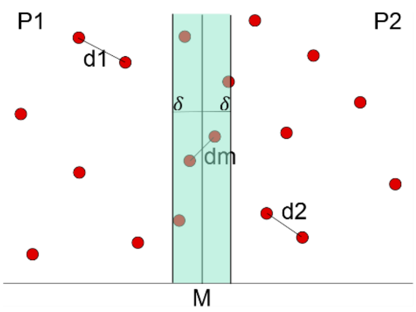
Pasul 3 recombinarea rezultatelor Se alege minimul dintre d_1, d_2 si d_m unde

d_m = min {d(A,B), pentru A ∈ P1, B ∈ P2}.

Pentru a obtine complexitatea O(n log n) vom rezolva pasul 3 in O(n) pasi (7n pasi).

Se va calcula cat mai eficient d_m (fara a se considera toate perechile de puncte (A,B) cu A ∈ P1, B ∈ P2)

Calculul distantei d_m**:**



Fie $\delta = \min(d_1, d_2)$. Este suficient sa consideram din P1 doar acele puncte $A(x_A, y_A)$ cu $M - x_A \leq \delta$ iar din P2 doar acele puncte $B(x_B, y_B)$ cu $x_B - M \leq \delta$. In caz contrar $d(A, B) \geq \delta$.

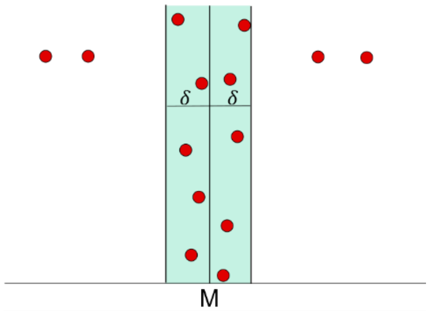
Astfel vom considera perechi de puncte cu abscisa

[M – δ, M + δ]

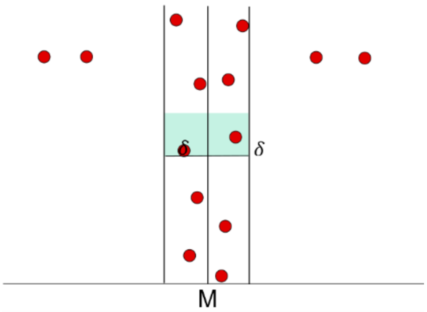
Vom nota multimea acestor puncte cu B_δ.

Daca vom considera toate perechile de puncte din multimea B_δ complexitatea obtinuta va fi O(n^2) deoarece $|B_\delta| \approx n$ (distanta dintre doua puncte pot fi mai mare decat δ chiar daca distanta dintre abscise este mai mica decat δ)

In constructia lui B_δ am luat in considerare doar distantele intre abscisele punctelor. Dar, spre exemplu, pentru un punct A cu ordonata 0 nu este necesar sa verificam decat distante $d(A, B)$ unde B este un punct cu ordonata $y_B \leq \delta$.



Este suficient sa consideram pentru fiecare punct $A(x_A, y_A)$ cu $|M - x_A| \leq \delta$ puncte $B(x_B, y_B)$ din dreptunghiul de dimensiuni [2δ x δ] centrat pe dreapta x = M.



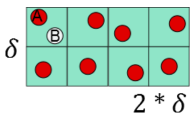
Deoarece $\delta = \min(d_1, d_2)$ intr-un astfel de dreptunghi pot fi maxim 8 puncte. Daca ar exista mai mult de 8 puncte cel putin 2 dintre ele ar fi la distanta mai mica decat δ.

Astfel pentru fiecare punct $A(x_A, y_A)$ cu $|M - x_A| \leq \delta$ este suficient sa consideram perechile formate cu urmataorele 7 puncte $B(x_B, y_B)$ sortate crescator dupa ordonata.

Presupunem ca ar exista doua puncte A, B in patratul din coltul din stanga sus din dreptunghiul de dimensiuni [2δ x δ] impartit in 8 patrate de latura $\delta/2$.

Atunci $d(A, B) \leq \delta$. Ceea ce ar contrazice alegerea lui δ ca fiind $\delta = \min(d_1, d_2)$. Ar insemna ca exista in P1 o pereche de puncte la distanta mai mica decat d_1.

Deci fiecare din cele 8 patrate contine maxim 1 punct.



Algoritm: X contine punctele sortate dupa abscisa, Y contine punctele sortate dupa ordonata

```
function dmin(X , Y, st, dr)

    daca |X| < 4
        d = min(perechi de elemente din X[st..dr])

    altfel
        mid=(st+dr)/2

        SY= mulțimea punctelor din Y ∩ X[st..mij]
        DY= mulțimea punctelor din Y ∩ X[mij+1..dr]
        d1=divimp(X[st..mij], SY, st, mij)
        d2=divimp(X[mij+1..dr], DY, mij+1,dr)
        d=min{d1, d2}

        LY= Y ∩ banda (cu abscisa la distanta < d de abscisa punctului X[mid])

        calculează d3 considerând punctele p din LY si perechile formate de p cu fiecare din
        cele 7 puncte care îi urmează în LY

        d=min{d, d3}

        return d
end
```