Metoda Divide et Impera

Metoda Divide et Impera (Imparte si Stapaneste) este o metoda de programare care se aplica problemelor care pot fi descompuse in subprobleme independente, similare problemei initiale, de dimensiuni mai mici si care pot fi rezolvate foarte usor. Procesul se reia pana cand (in urma descompunerilor repetate) se ajunge la probleme care admit rezolvare imediata.

Metoda presupune:

- Descompunerea problemei **Pb** curente in subprobleme independente **$SubPb_i$.**
  - In cazul in care subproblemele **$SubPb_i$.** admit o rezolvare imediata se compun solutiile si se rezolva problema **Pb**
  - Altfel se descompun in mod similar si subproblemele **$SubPb_i$**

Evident, nu toate problemele pot fi rezolvate prin utilizarea acestei tehnici. Majoritatea algoritmilor de Divide et Impera presupun prelucrari pe tablouri (dar nu obligatoriu).

Aceasta metoda (tehnica) se poate implementa atat iterativ dar si recursiv. Dat fiind ca problemele se impart impart in subprobleme in mod recursiv, de obicei impartirea se realizeaza pana cand sirul obtinut este de lungime 1, caz in care rezolvarea subproblemei este foarte usoara, chiar banala.

Spre exemplu fie un vector $X=[x_1, x_2, x_3, \ldots x_i \ldots x_p \ldots x_j, \ldots x_n]$ asupra caruia se aplica o prelucrare. Pentru orice secventa din vector delimitata de indecsii i si j, i<j exista o valoare p astfel incat prin prelucrarea secventelor :

$$x_i, x_{i+1}, x_{i+2}, x_{i+3}, \ldots x_p \text{ si } x_{p+1}, x_{p+2}, x_{p+3}, \ldots x_j$$

se obtin solutiile corespunzatoare celor doua subsiruri care prin compunere conduc la obtinerea solutiei prelucrarii secventei:

$$x_i, x_{i+1}, x_{i+2}, x_{i+3}, \ldots x_j$$

Aplicatie:
Ne propunem sa determinam suma elementelor unui vector de intregi utilizand metoda **Divide et Impera:**

2	3	4	1	5	6	8	9	1	10	3	4	4	5	2	6

Se imparte vectorul in doi vectori:

2	3	4	1	5	6	8	9

1	10	3	4	4	5	2	6

Fiecare se dintre cei doi vectori se poate imparti in continuare in alti doi vectori:

2	3	4	1

5	6	8	9

1	10	3	4

4	5	2	6

La fel se procedeaza in continuare:

2	3

4	1

5	6

8	9

1	10

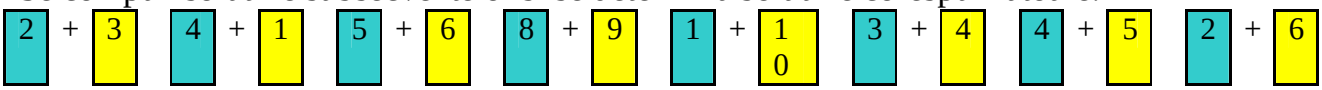
3	4

4	5

2	6

Apoi:

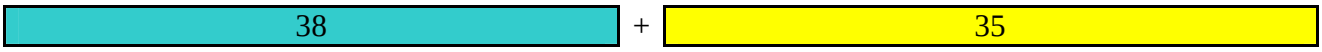
Dat fiind ca s-au obtinut secvente de vector de lungime 1, nu se mai realizeaza descompunerea.
Se compun solutiile subsecventelor si se determina solutiile corespunzatoare:



Si in continuare:

Apoi:

La fel:



La sfarsit:

Iata o solutie de implementare:

```
#include<iostream>
using namespace std;

int v[20],n;

int divide(int li, int ls) // functia primeste ca parametri extremitatile unei secvente din vector
{
int mij, d1 ,d2;  //mijlocul, d1 si d2 retin sumele pe extremitatea stanga respectiv dreapta
 if(li != ls)  //algoritmul se autoapeleaza daca secventele au lungime mai mare de 1
{
      mij = (li + ls) / 2;
     d1=divide(li, mij);
     d2=divide(mij+1, ls);
      return d1+d2;
     }
 else
   return v[li];
}
```

```
void main()
{
 cout<<"n=";
 cin>>n;
 for(int i=1; i <= n; i++)
 {
    cout<<"v["<<i<<"]=";
   cin>>v[i];}
cout<<"suma celor "<<n<<" elemente ale vectorului "<<divide(1,n);
return 0;
}
```

Maximul dintr-un vector

Algoritmul pentru determinarea maximului dintr-un vector (matrice unidimensională) este folosit pentru a calcula cel mai mare element dintr-un vector nesortat. Algoritmul inițializează o variabilă cu valoarea primului element, apoi compară restul elementelor cu acea variabilă. Dacă un element din vector e mai mare decât respectiva variabilă, ea preia noua valoare, până se epuizează toate elementele din vector.

Acesta este un algoritm care determină cel mai mare întreg dintr-un vector dat. Algoritmul functionează după acelasi principiu si în cazul unui vector cu elemente de alt tip (caractere, sau cuvinte) fiind necesare modificări la declarare, (și eventual la comparație).

```
int maxim (int n, int v[]) // n - numărul de elemente din vector, v[] - vectorul de n elemente
  {
     int max=v[0]; // Se inițializează variabila de tip întreg, max, cu primul element al vectorului v[].
     for(int i=0;i<n;i++) // Structura repetitivă de testare al maximului, începând cu primul element
până la ultimul.
          if(max<v[i]) // Dacă maximul precedent este mai mic decât elementul curent,
              max=v[i];  //maximul este înlocuit cu noul maxim găsit.
     return max; // Returnează în program maximul găsit.
  }
```

Probleme propuse:

1. Sa se determine produsul a n numere intregi
2. Sa se determine cel mai mare divizor comun a n valori dintr-un vector
3. Sa se caute o valoare intr-un vector. Daca se gaseste se va afisa pozitia pe care s-a gasit, altfel se va afisa un mesaj.
4. Sa se caute o valoare intr-un vector ordonat crescator
5. Sa se numere cate valori sunt egale cu x dintr-un sir de numere intregi citite.