Metoda Divide et Impera

Explicații complexități

Exemplu –maximul elementelor unui vector

```
function DivImp(p,u)
    if u==p
        r ← a[p]
    else
        m ← ⌊(p+u)/2⌋
        r1 ← DivImp(p,m)
        r2 ← DivImp(m+1,u)
        if r1>r2
            r ← r1
        else
            r ← r2
    return r
```

Apel: `DivImp(0,n-1)`

- `function DivImp(p,u)` ← Problema de dimensiune n
- `if u==p` → subproblema de dimensiune 1 – o rezolv direct in O(1) => T(1) = 1
- `r1 ← DivImp(p,m)` ← Subproblema de dimensiune n/2
- `r2 ← DivImp(m+1,u)` ← Subproblema de dimensiune n/2
- `if r1>r2 … r ← r2` ← Combinarea rezultatelor celor două subprobleme – timp constant c (O(1))
- `DivImp(0,n-1)` → T(n) = 2T(n/2) + O(1) => T(n) = 2T(n/2) + 1 (putem considera c=1)

Metoda Divide et Impera

- **Complexitate – relație de recurență**

• Suficient să presupunem că $n = 2^k$ și $c = 1$

$$T(n) = \begin{cases} 1, & \text{pentru } n = 1 \\ 2T(n/2) + 1, & \text{pentru } n > 1 \end{cases}$$

Rezolvăm recurența $T(n) = 2T(n/2) + 1$ pentru $n=2^k$

=> **$k=\log_2(n)$**

Putem înlocui de la început n cu 2^k sau lucra cu n, n/2, $n/2^2$...
Prezentăm ambele variante:

$T(n) = 2T(n/2) + 1$

VARIANTA 1 – înlocuim de la început n cu 2^k

$T(n) = 2T(n/2) + 1$

$T(n) = T(2^k) = 2 \cdot T(2^{k-1}) + 1 =$

$$T(n) = 2T(n/2) + 1$$

$$T(n) = T(2^k) = 2 \cdot T(2^{k-1}) + 1 =$$

$$= 2 \cdot [2T(2^{k-2}) + 1] + 1 =$$

$T(n) = 2T(n/2) + 1$

$$T(n) = T(2^k) = 2 \cdot T(2^{k-1}) + 1 =$$

$$= 2 \cdot [2T(2^{k-2}) + 1] + 1 =$$

$$= 2^2 \cdot T(2^{k-2}) + (1+2) =$$

$$=$$

$T(n) = 2T(n/2) + 1$

$$T(n) = T(2^k) = 2 \cdot T(2^{k-1}) + 1 =$$

$$= 2 \cdot [2T(2^{k-2}) + 1] + 1 =$$

$$= 2^2 \cdot T(2^{k-2}) + (1+2) =$$

$$= 2^2 \cdot [2T(2^{k-3}) + 1] + (1+2) =$$

$$T(n) = 2T(n/2) + 1$$

$$T(n) = T(2^k) = 2 \cdot T(2^{k-1}) + 1 =$$

$$= 2 \cdot [2T(2^{k-2}) + 1] + 1 =$$

$$= 2^2 \cdot T(2^{k-2}) + (1+2) =$$

$$= 2^2 \cdot [2T(2^{k-3}) + 1\ ] + (1+2) =$$

$$= 2^3 \cdot T(2^{k-3}) + (1+2+2^2) = \ldots$$

$$=$$

$T(n) = 2T(n/2) + 1$

$$T(n) = T(2^k) = 2 \cdot T(2^{k-1}) + 1 =$$

$$= 2 \cdot [2T(2^{k-2}) + 1] + 1 =$$

$$= 2^2 \cdot T(2^{k-2}) + (1+2) =$$

$$= 2^2 \cdot [2T(2^{k-3}) + 1] + (1+2) =$$

$$= 2^3 \cdot T(2^{k-3}) + (1+2+2^2) = \ldots$$

$$= 2^k \cdot T(2^0) + (1+2+2^2+\ldots+2^{k-1}) =$$

$$T(n) = 2T(n/2) + 1$$

$$T(n) = T(2^k) = 2 \cdot T(2^{k-1}) + 1 =$$

$$= 2 \cdot [2T(2^{k-2}) + 1] + 1 =$$

$$= 2^2 \cdot T(2^{k-2}) + (1+2) =$$

$$= 2^2 \cdot [2T(2^{k-3}) + 1] + (1+2) =$$

$k=\log_2(n)$, $2^k=n$

$$= 2^3 \cdot T(2^{k-3}) + (1+2+2^2) = \ldots$$

$$= 2^k \cdot T(2^0) + (1+2+2^2+\ldots+2^{k-1}) =$$

$$= 2^k \cdot T(1) + (2^k-1) = n + (n-1) = (2n-1) \Rightarrow$$

$$\Rightarrow \mathbf{T(n) = O(n)}$$

$T(n) = 2T(n/2) + 1$

VARIANTA 2 – lucrăm cu n, $n/2$, $n/2^2$...

$T(n) = 2T(n/2) + 1$

$$
\begin{aligned}
T(n) &= 2 \cdot T(n/2) + 1 = \\
&= 2 \cdot [2T(n/2^2) + 1] + 1 = \\
&= 2^2 \cdot T(n/2^2) + (1+2) = \\
&= 2^2 \cdot [2T(n/2^3) + 1] + (1+2) = \\
&= 2^3 \cdot T(n/2^3) + (1+2+2^2) = \ldots \\
&= 2^k \cdot T(n/2^k) + (1+2+2^2+\ldots+2^{k-1}) = \\
&= 2^k \cdot T(1) + (2^k-1) = n + (n-1) = (2n-1) \Rightarrow
\end{aligned}
$$

$k=\log_2(n)$

$\Rightarrow$ **$T(n) = O(n)$**

Căutarea binară

Căutarea binară

```
def cautare_binara(x,ls,p,u):
    if p > u:
        return (False, u)
    else:
        mij = (p + u) // 2
        if x == ls[mij]:
            return (True,mij)
        elif x < ls[mij]:
            return cautare_binara(x,ls, p, mij-1)
        else:
            return cautare_binara(x,ls, mij+1, u)

def cautare(x,ls):
    n = len(ls)
    return cautare_binara(x,ls,0,n-1)
```

Problema de dimensiune n

Timp constant O(1)

O subproblema
de dimensiune n/2
(apelul recursiv se face doar pentru o jumatate, stanga sau dreapta dupa caz)

$\Rightarrow T(n) = T(n/2) + O(1)$

$T(n) = T(n/2) + c$ (putem pp c=1)

Căutarea binară

- **Complexitate:** $O(\log n)$
  - $n=2^k$
  - $T(n) = T(n/2)+c = [T(n/2^2)+c]+c = T(n/2^2)+2c =$

    $= \ldots = T(n/2^k)+kc = T(1) + (\log_2 n)c$

    $k = \log_2 n,\ T(n/2^k)=T(1)=1$

$\Rightarrow O(\log_2 n)$

Sortarea prin interclasare

Sortare prin interclasare

```
def sort_interclasare(v, p, u):
    if p == u:
        pass
    else:
        m = (p+u)//2
        sort_interclasare(v, p, m)
        sort_interclasare(v, m+1, u)
        interclaseaza(v, p, m, u)
```

Problema de dimensiune n

Timp constant O(1)

Două subprobleme de dimensiune n/2

O(n) (interclaseaza cele doua jumatati, adica doi vectori de dimensiune n/2)

$\Rightarrow T(n) = 2T(n/2) + O(n)$

$T(n) = 2T(n/2) + n$

$T(n)=2T(n/2)+n,\ n=2^k$

Varianta 1 – inlocuim de la inceput n cu 2^k

$$T(n) = T(2^k) = 2\ T(2^{k-1}) + 2^k =$$

$$= 2\ [2T(2^{k-2}) + 2^{k-1}] + 2^k = 2^2T(2^{k-2}) + 2\cdot 2^k$$

$$= \ldots = 2^iT(2^{k-i}) + i\cdot 2^k =$$

$$= 2^kT(1) + k\cdot 2^k = n + n\cdot \log_2 n$$

$\Rightarrow$ **O(n log(n))**

$k=\log_2(n)$, $2^k=n$

$T(n)=2T(n/2)+n,\ n=2^k$

Varianta 2

$T(n) = 2\ T(n/2) + n =$

$= 2\ [2T(n/2^2) + n/2] + n =$

$= 2^2T(n/2^2) + 2 \cdot n/2 + n = 2^2T(n/2^2) + n + n =$

$= 2^2T(n/2^2) + 2 \cdot n =$

$= \ldots = 2^kT(n/2^k) + k \cdot n = 2^kT(1) + k \cdot n = n + n \cdot \log_2 n$

$\Rightarrow$ **$O(n \log(n))$**

$k=\log_2(n),\ 2^k=n$

Aplicație

Numărare inversiuni

Problemă

Se consideră un vector cu n elemente distincte.
Să de determine numărul de inversiuni din acest vector

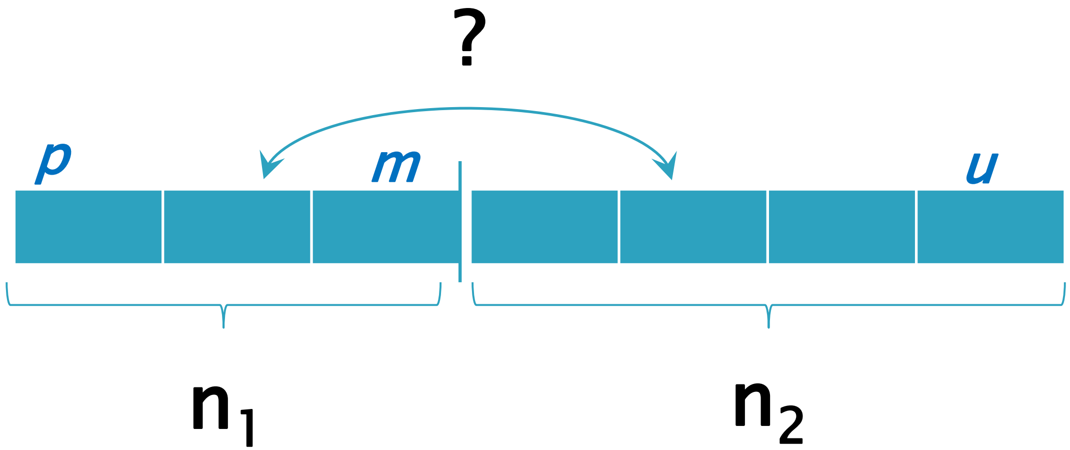
- **Inversiune** = pereche (i, j) cu proprietatea că $i < j$ şi $a_i > a_j$

- **Exemplu** **`1,2,11,9,4,6`** $\Rightarrow$ **`5 inversiuni`**

  `((11,9), (11,4), (9,4), (11,6), (9,6))`

Numărare inversiuni

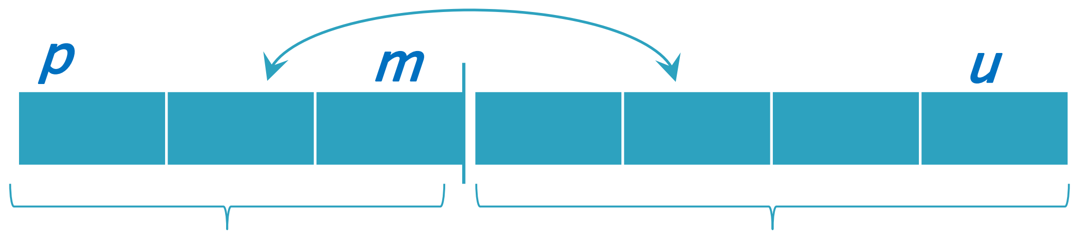
- Algoritm Divide et Impera



$$n_1+n_2+?$$

$$n_1+n_2+n_3$$

$$n_3$$



n_1

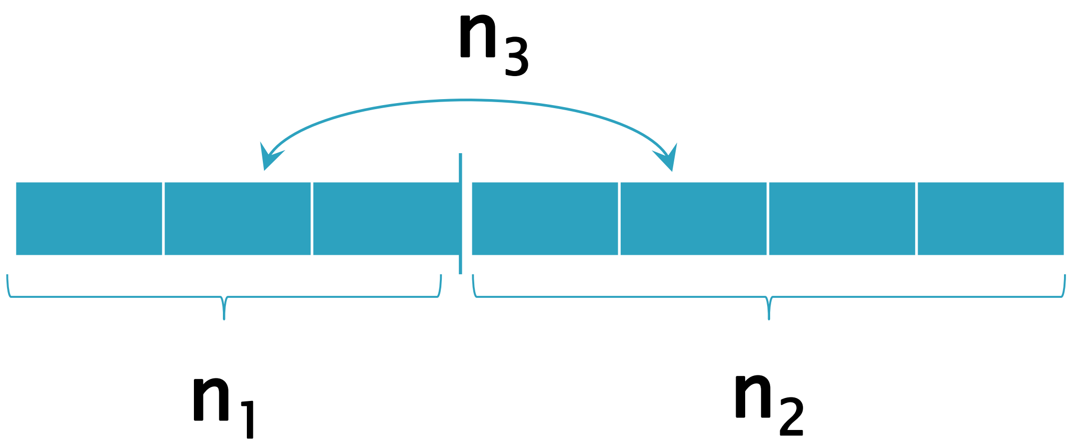
numarul de inversiuni din "jumatatea" stanga
=> T(n/2) pentru calcul n1

n_2

numarul de inversiuni din "jumatatea" stanga
=> T(n/2) pentru calcul n2

Cum calculăm eficient n_3?

$$T(n) = 2T(n/2) + O(\text{calcul } n_3)$$



Cum calculăm eficient n_3?

- **Încercare**: Considerăm fiecare pereche (i,j) cu i în subvectorul stâng și j în cel drept

$$T(n) = 2T(n/2) + \mathbf{O(n^2)}$$

```
for i in jumatatea din stanga
    for j in jumatatea din dreapta
        if v[i]>v[j]: n3 +=1
```

Fiecare jumatate are n/2 elemente => $(n/2)*(n/2) \Rightarrow O(n^2)$

Complexitate

$T(n) = 2T(n/2) + n^2 =$

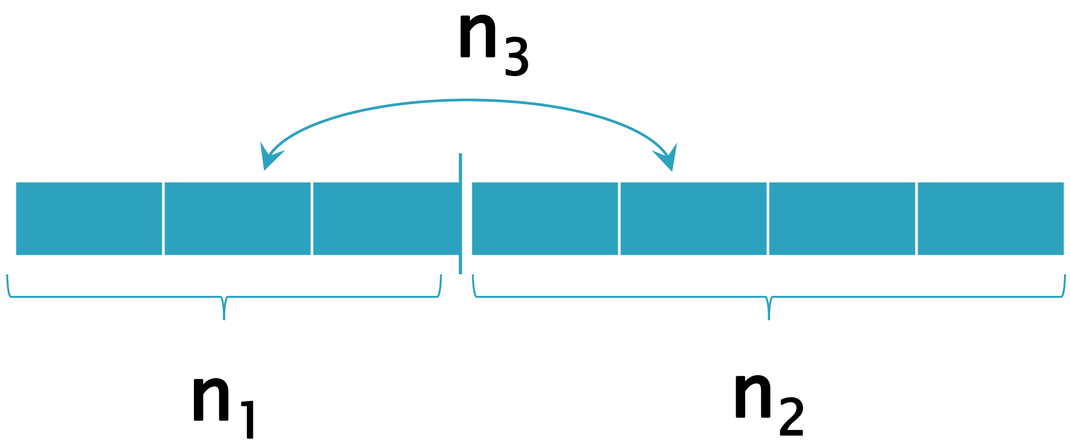
$= 2[2T(n/2^2) + (n/2)^2] + n^2 =$

$= 2^2T(n/2^2) + n^2(1+1/2)$

$= \dots =$

$= 2^kT(n/2^k) + n^2(1+1/2+\dots+1/2^{k-1}) =$

$$= n + n^2\left(\underbrace{1+\frac{1}{2}+\frac{1}{2^2}..+\frac{1}{2^{k-1}}}_{\leq 2 \text{ (suma de progresie geometrica)}}\right) \Rightarrow O(n^2)$$



Cum calculăm eficient n_3?

- **Mai bine** Numărăm inversiunile la interclasare

$$T(n) = 2T(n/2) + \color{red}{O(n)}$$

complexitatea interclasare

```
def nr_inversiuni(v, p, u):
    if p==u:
        return 0
    else:
        m = (p+u)//2
        n1 = nr_inversiuni(v, p, m)
        n2 = nr_inversiuni(v, m+1, u)
        return n1+n2+interclaseaza(v, p, m, u)
```

Problema de dimensiune n

Timp constant O(1)

Două subprobleme de dimensiune n/2

O(n) (interclasare)

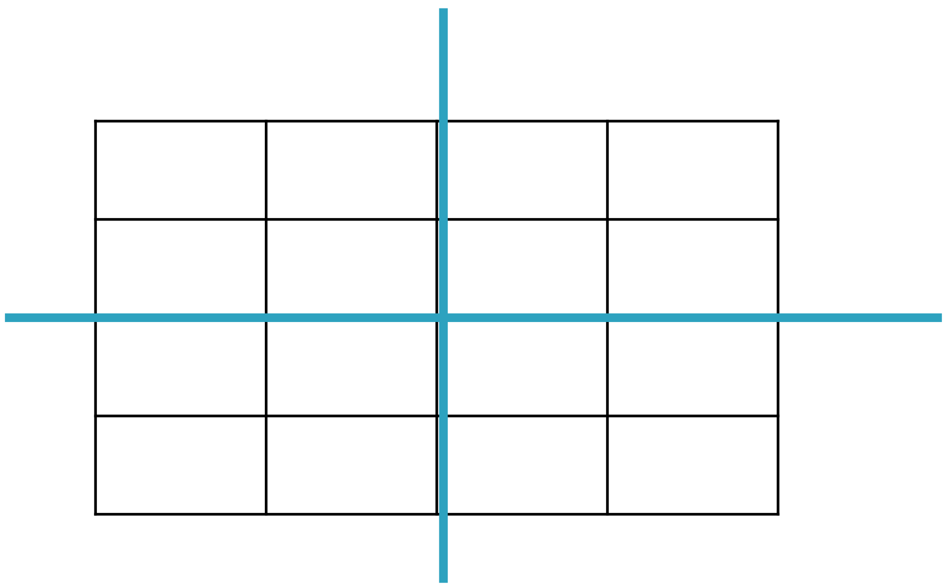
=> $T(n) = 2T(n/2) + O(n)$

$T(n) = 2T(n/2) + n$ – ca la sortarea prin interclasare

- **Apel:** `x = nr_inversiuni(v,0, len(v)-1)`

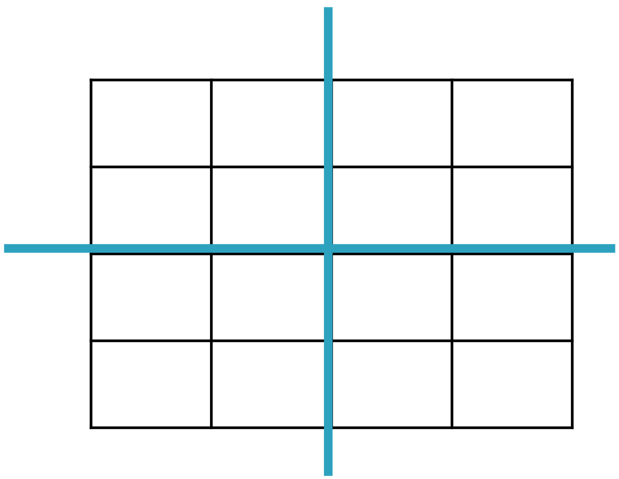
Divide et impera – Matrice

Să se calculeze suma elementelor unei matrice pătratice de dimensiune n unde n este putere a lui 2 folosind Divide et Impera (împărțind matricea succesiv în 4 subcadrane)



Divide et impera – Matrice

Să se calculeze suma elementelor unei matrice pătratice de dimensiune n unde n este putere a lui 2 folosind Divide et Impera (împărțind matricea succesiv în 4 subcadrane)



O subproblemă este identificată de:

- Două colțuri opuse ale submatricei

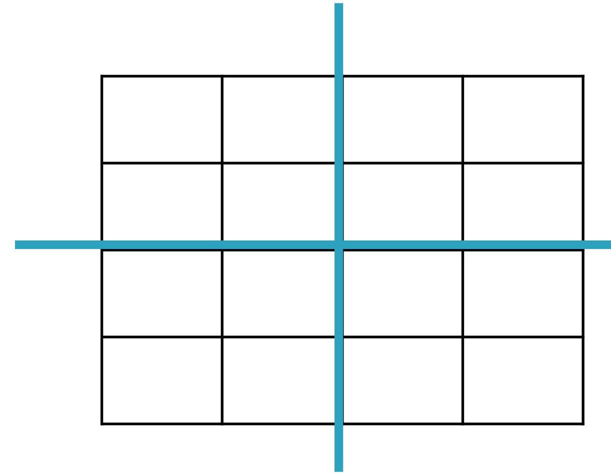
    `divide(x1, y1, x2, y2)`

sau

- Un colț al submatricei și dimensiunea ei

    `divide(x, y, dim)`

Divide et impera – Matrice



```python
def suma(m,x,y,n):
    print(x,y)
    if n==1:
        return m[x][y]

    s1 = suma(m, x, y, n//2)
    s2 = suma(m, x+n//2, y, n//2)
    s3 = suma(m, x, y+n//2, n//2)
    s4 = suma(m, x+n//2, y+n//2, n//2)
    return s1+s2+s3+s4

def suma_matrice(m):
    return suma(m,0,0,len(m))
```

Divide et impera – Matrice

Problema matrice de latura n

```
def suma(m,x,y,n):
    print(x,y)
    if n==1:
        return m[x][y]

    s1 = suma(m, x, y, n//2)
    s2 = suma(m, x+n//2, y, n//2)
    s3 = suma(m, x, y+n//2, n//2)
    s4 = suma(m, x+n//2, y+n//2, n//2)
    return s1+s2+s3+s4
```

Timp constant O(1)

4 subprobleme cu matrice de dimensiune n/2

O(1)

```
def suma_matrice(m):
    return suma(m,0,0,len(m))
```

=> $T(n) = 4T(n/2) + O(1)$

$T(n) = 4T(n/2) + 1$

$T(n)=4T(n/2)+1,\ n=2^k$

$$T(n) = 4\,T(n/2) + 1 =$$

$$= 4\,[4T(n/2^2) + 1] + 1 =$$

$$= 4^2T(n/2^2) + 4+1 = \quad =$$

$$= 4^2[4T(n/2^3)+1] + 4+1 =$$

$$= 4^3T(n/2^3) + 4^2 + 4+1 =$$

$T(n/2^k)=T(1)=1$

$$= \ldots = 4^kT(n/2^k) + 4^{k-1}+\ldots+4^2+4+1 =$$

$$= 4^k + 4^{k-1}+\ldots+4^2+4+1 = (4^{k+1}-1)/3 = (4n^2-1)/3$$

$$\Rightarrow O(n^2)$$

$n=2^k \Rightarrow 4^k=n$