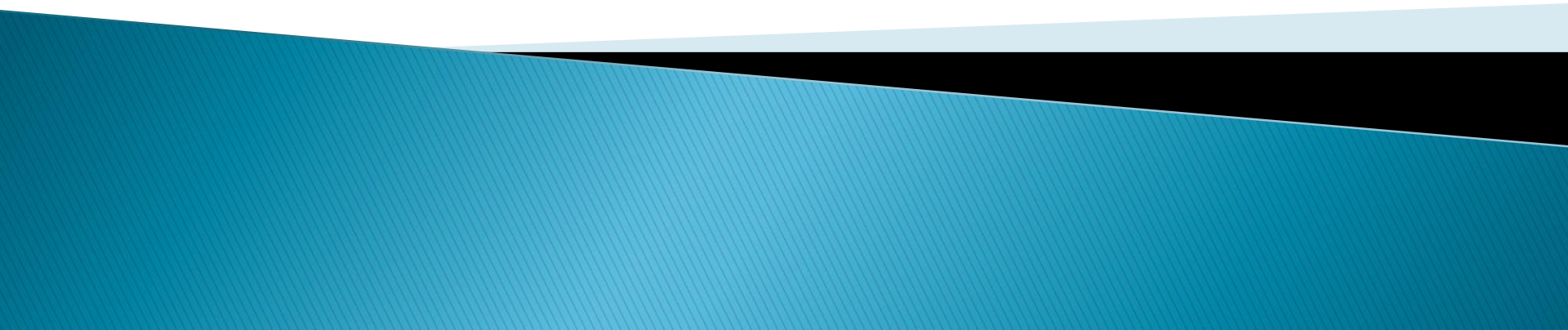


# Metoda Divide et Impera

desparte și stăpânește



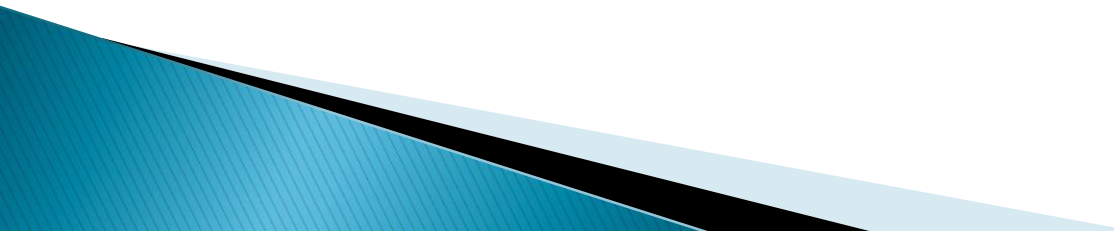
# Metoda Divide et Impera

- ▶ Constă în
  - **împărțirea** repetată a unei probleme de dimensiuni mari în mai multe subprobleme **de același tip**

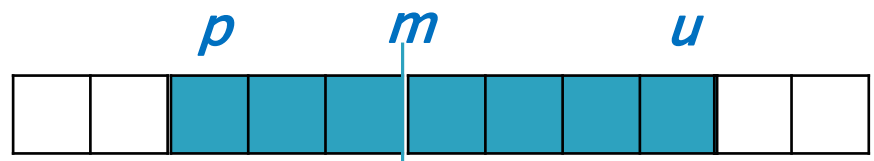
# Metoda Divide et Impera

- ▶ Constă în
  - **împărțirea** repetată a unei probleme de dimensiuni mari în mai multe subprobleme **de același tip**
  - **rezolvarea** acestor subprobleme (în același mod sau prin rezolvare directă, dacă au dimensiune mică)

# Metoda Divide et Impera

- ▶ Constă în
    - **împărțirea** repetată a unei probleme de dimensiuni mari în mai multe subprobleme **de același tip**
    - **rezolvarea** acestor subprobleme (în același mod sau prin rezolvare directă, dacă au dimensiune mică)
    - **combinarea rezultatelor** obținute pentru a determina rezultatul corespunzător problemei inițiale.
- 

# Schema posibilă



– pentru  $a[p..u]$

```
function DivImp(p, u)
```

```
    if  $u - p < \epsilon$ 
```

```
         $r \leftarrow \text{RezolvaDirect}(p, u)$ 
```

```
    else
```

```
         $m \leftarrow \text{Pozitie}(p, u)$  – de obicei mijlocul
```

```
         $r1 \leftarrow \text{DivImp}(p, m)$ 
```

```
         $r2 \leftarrow \text{DivImp}(m+1, u)$ 
```

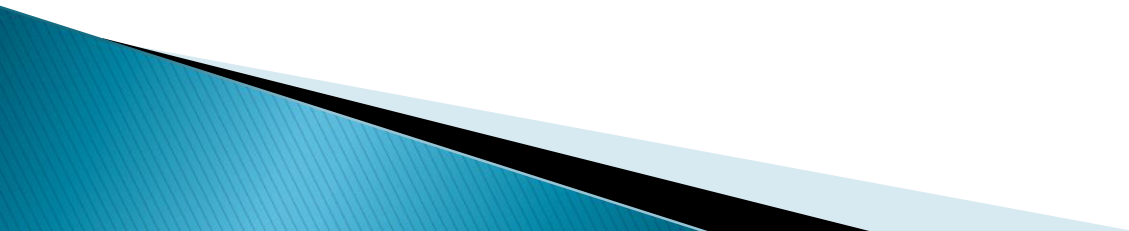
```
         $r \leftarrow \text{Combina}(r1, r2)$ 
```

```
    return r
```

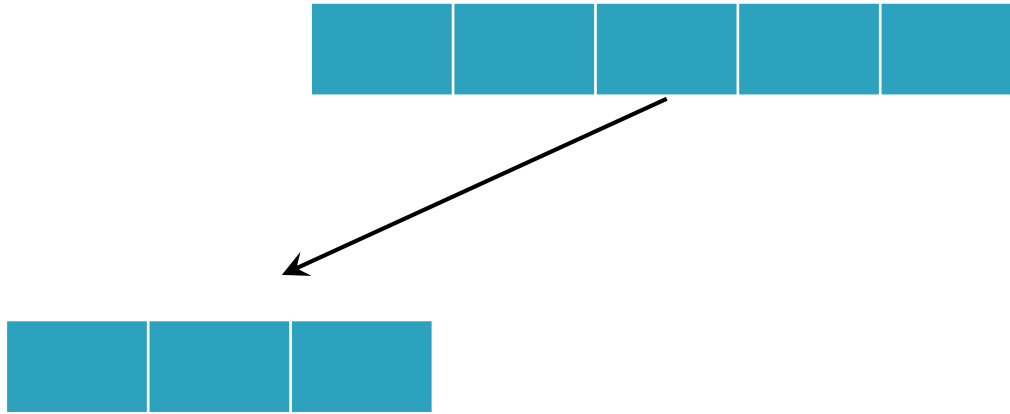
```
end
```

► **Apel:**  $\text{DivImp}(1, n)$

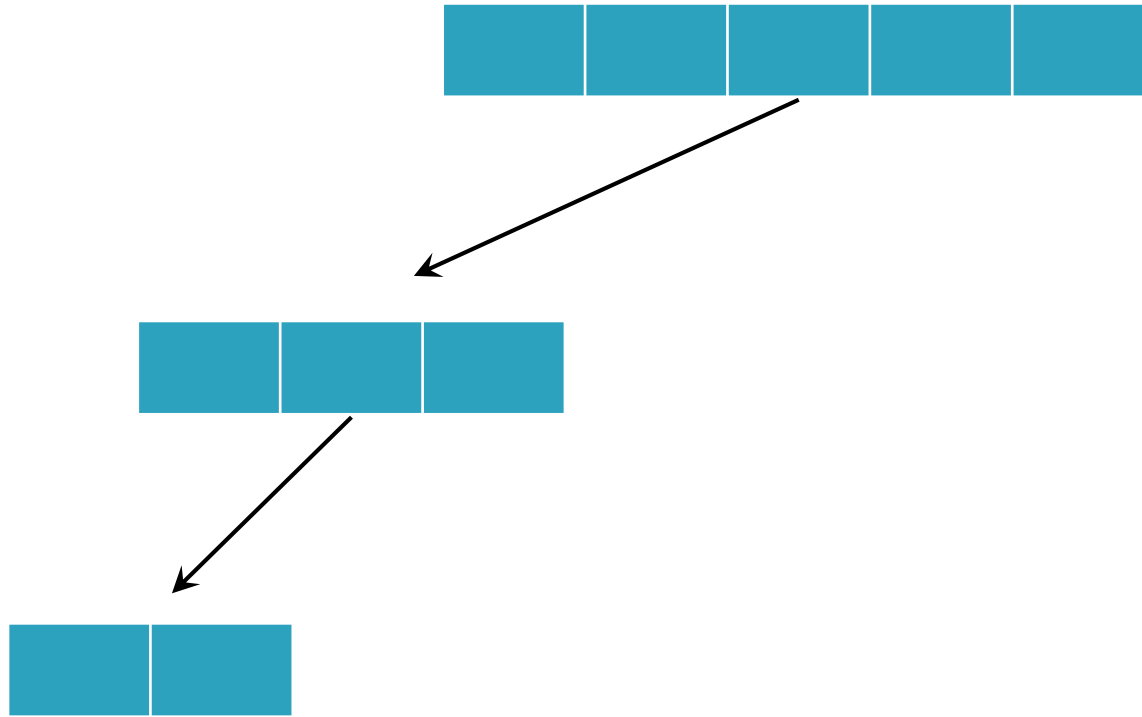
# Metoda Divide et Impera



# Metoda Divide et Impera

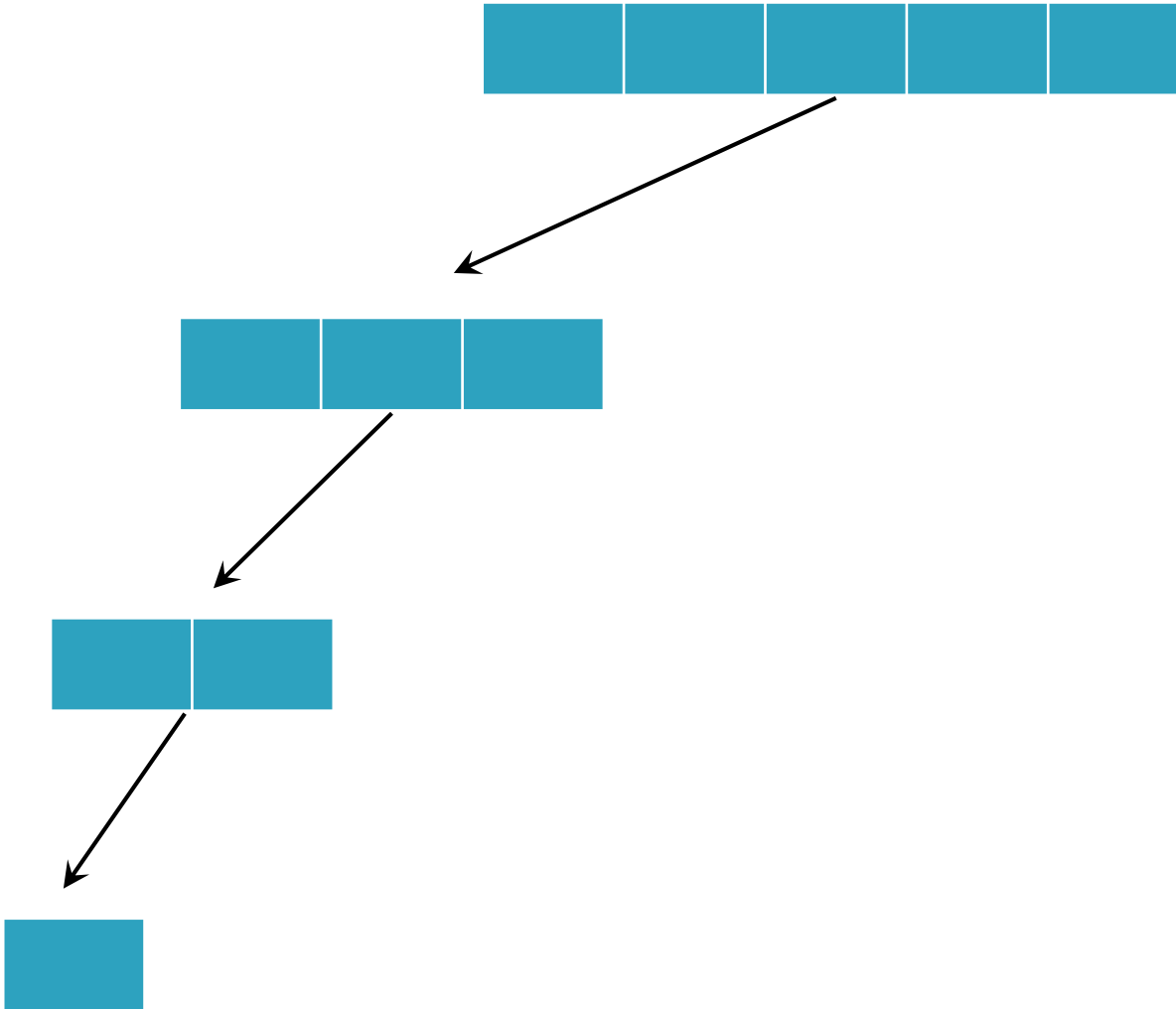


# Metoda Divide et Impera

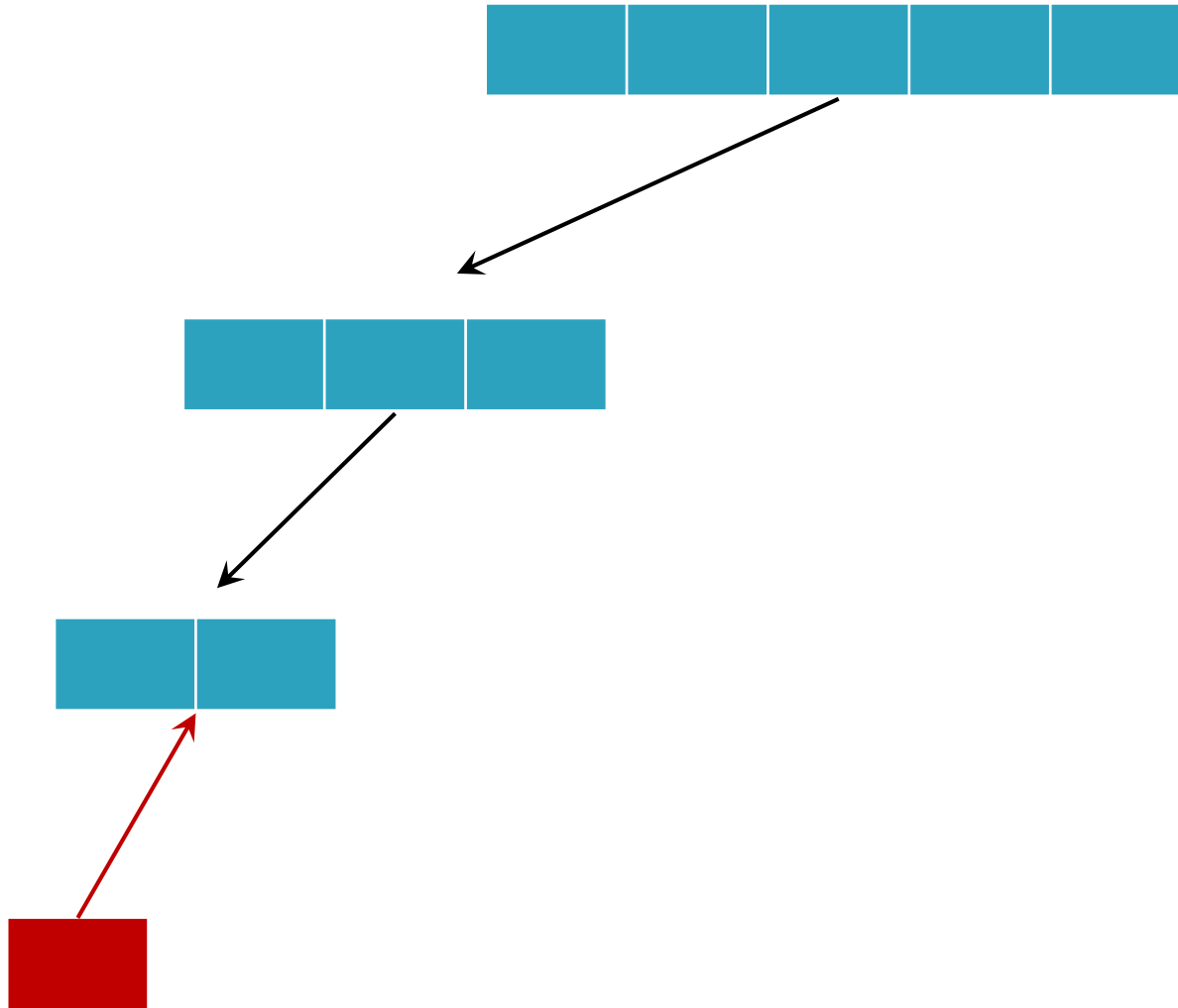




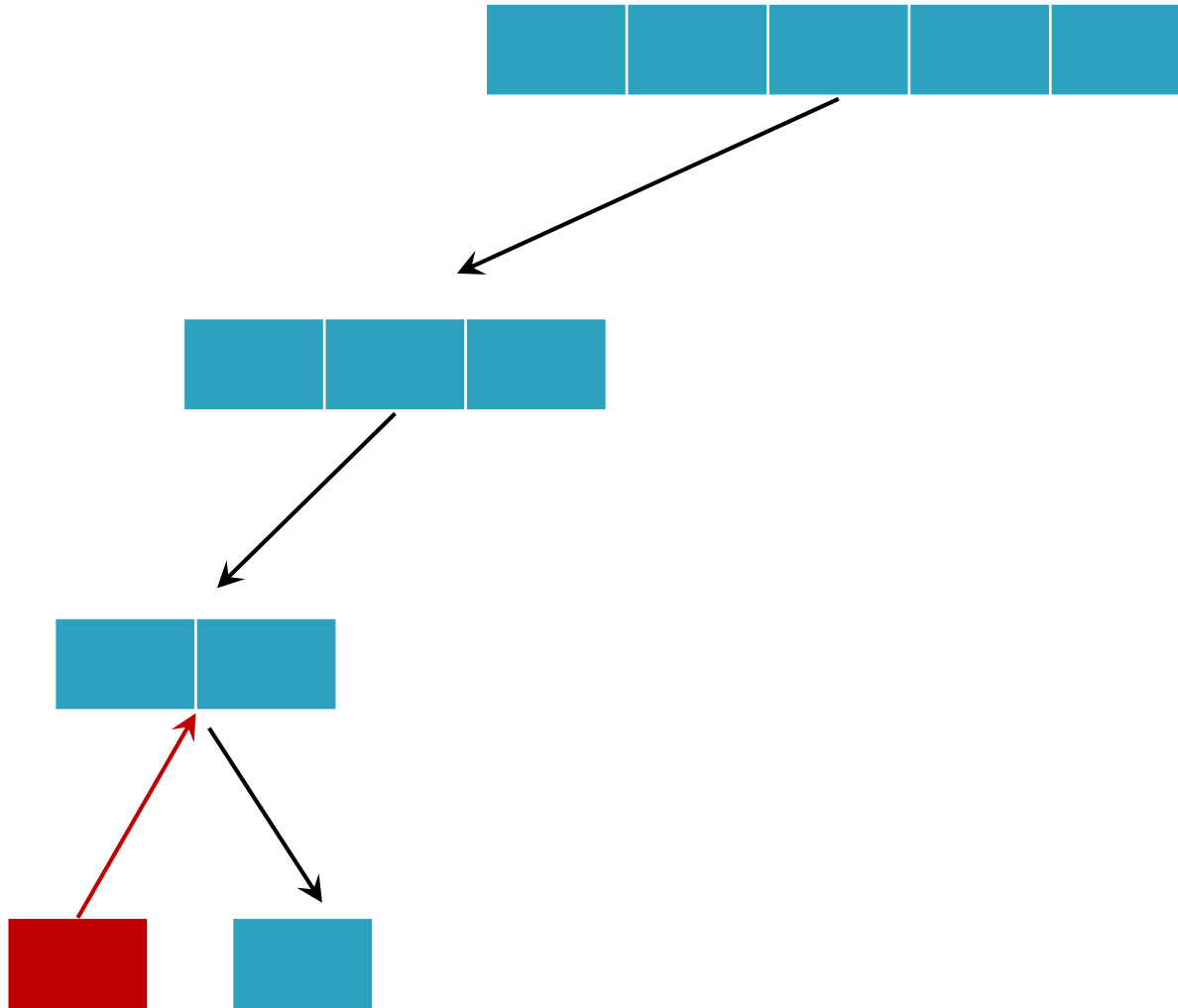
# Metoda Divide et Impera



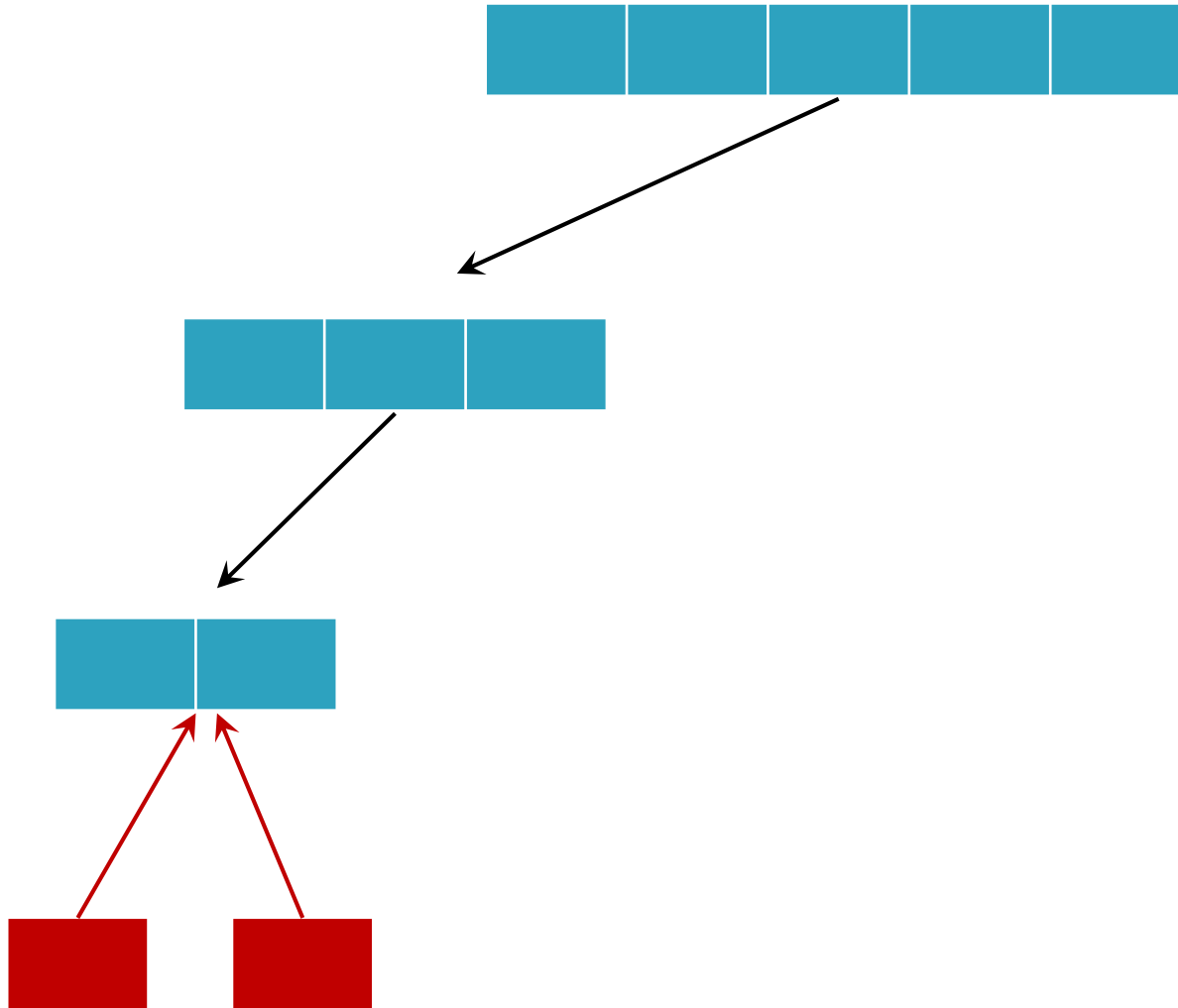
# Metoda Divide et Impera



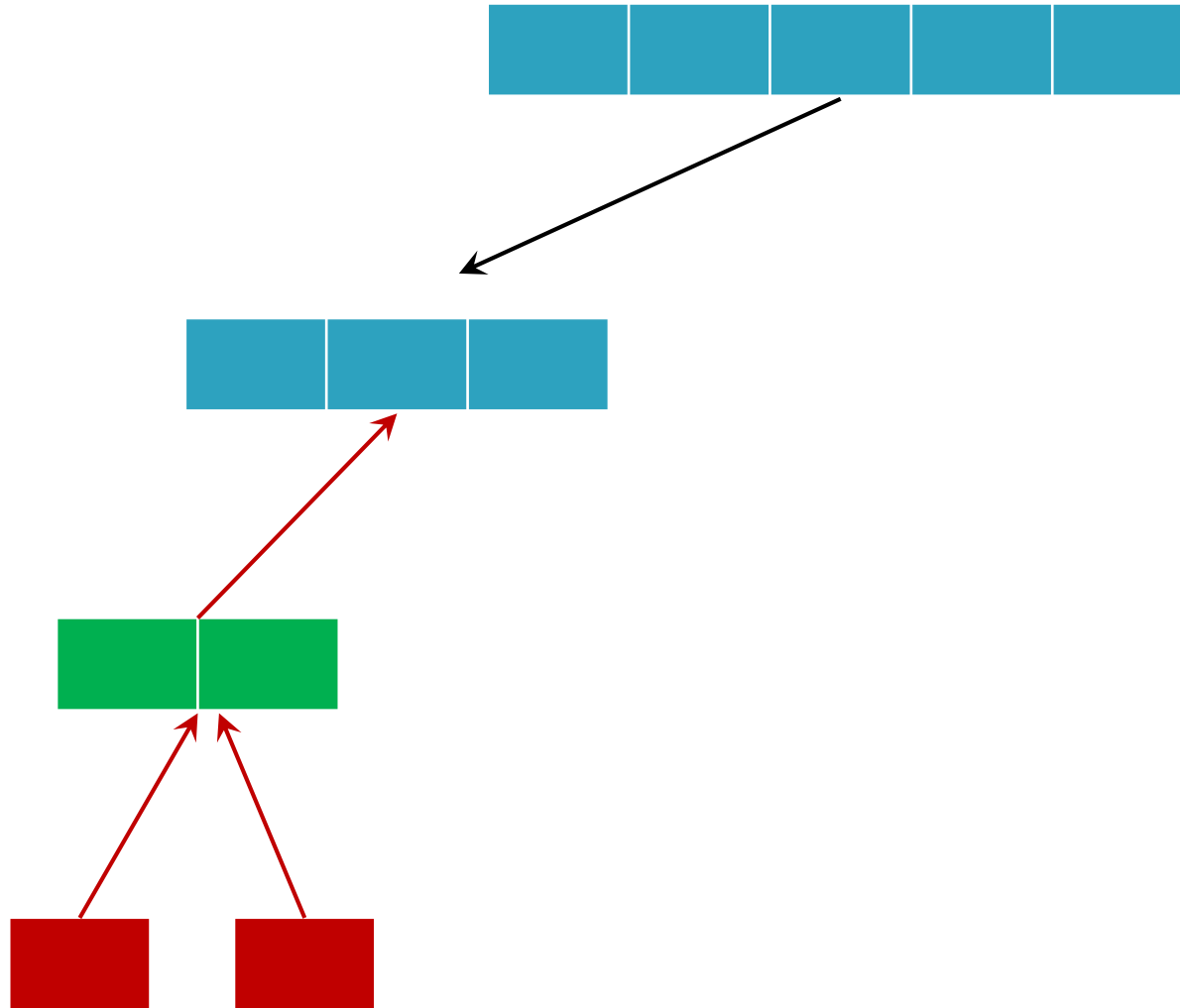
# Metoda Divide et Impera



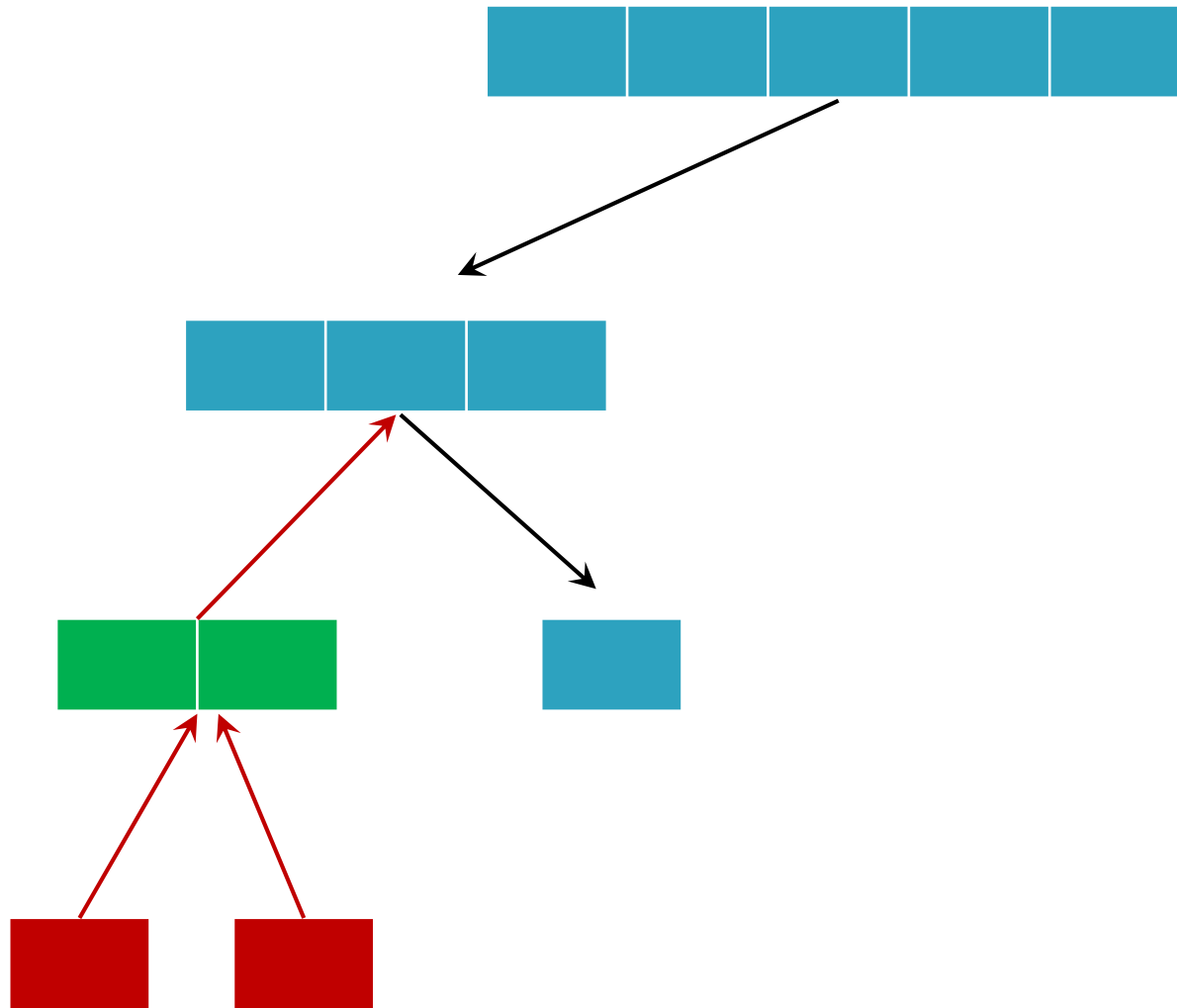
# Metoda Divide et Impera



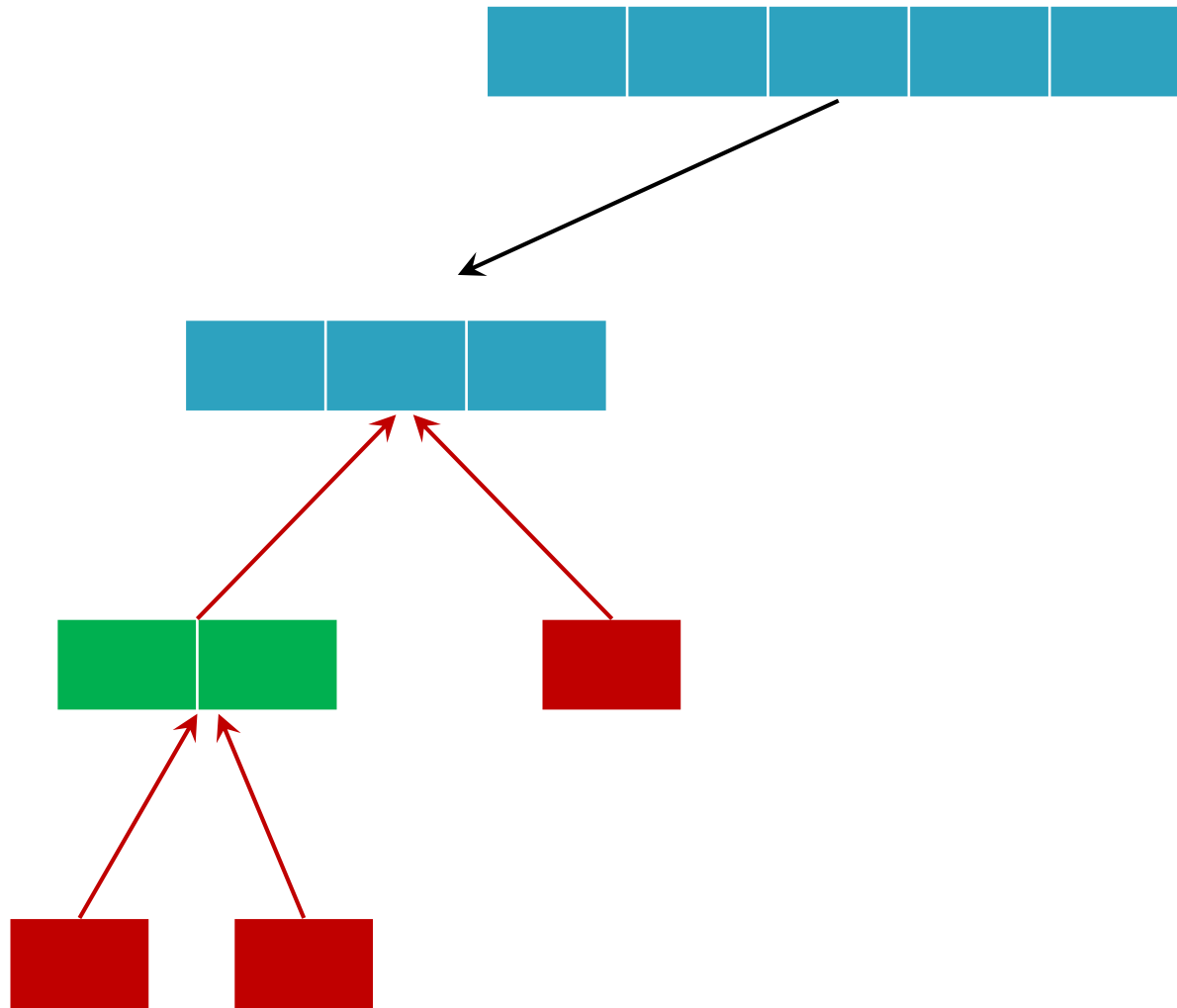
# Metoda Divide et Impera



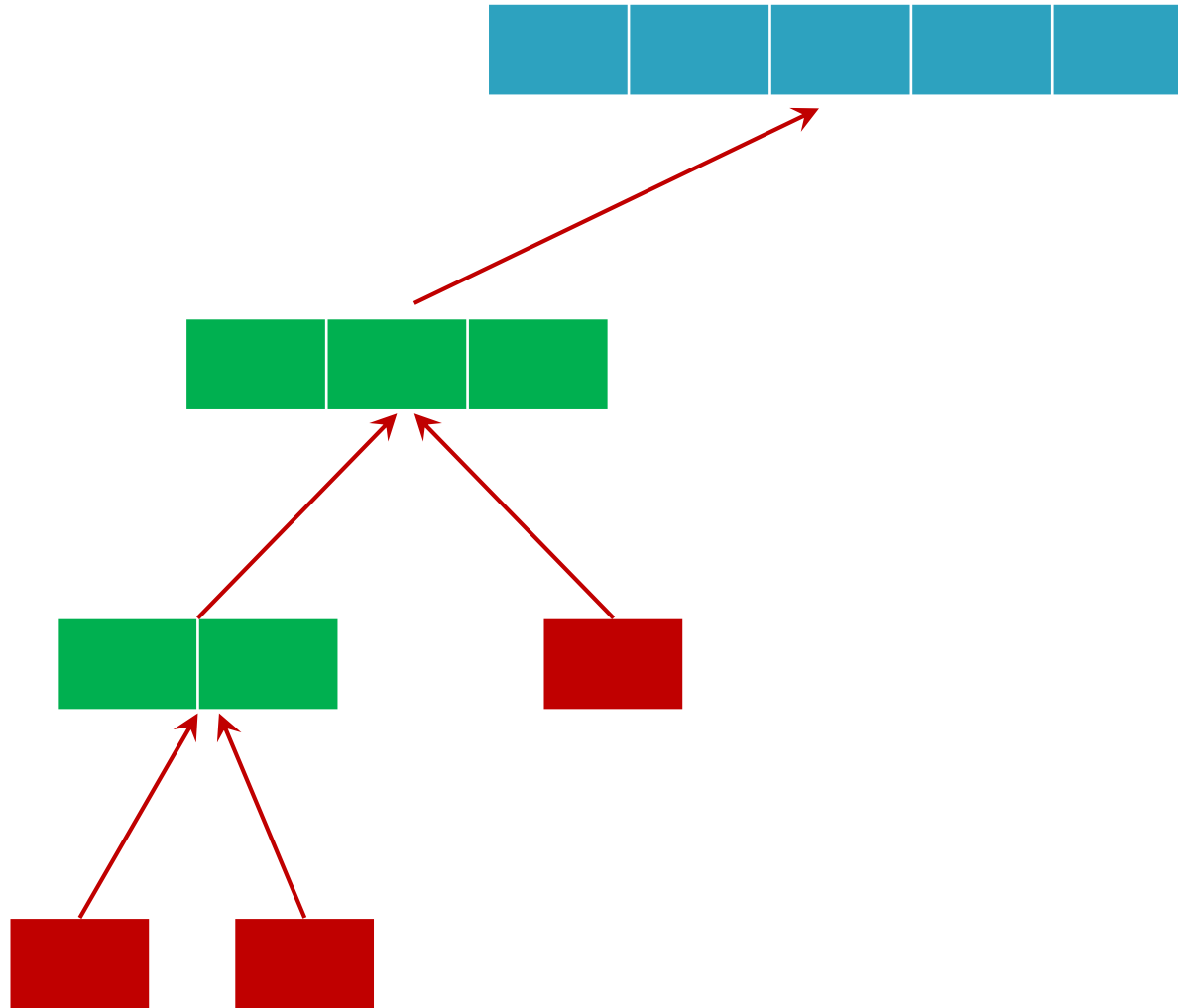
# Metoda Divide et Impera



# Metoda Divide et Impera

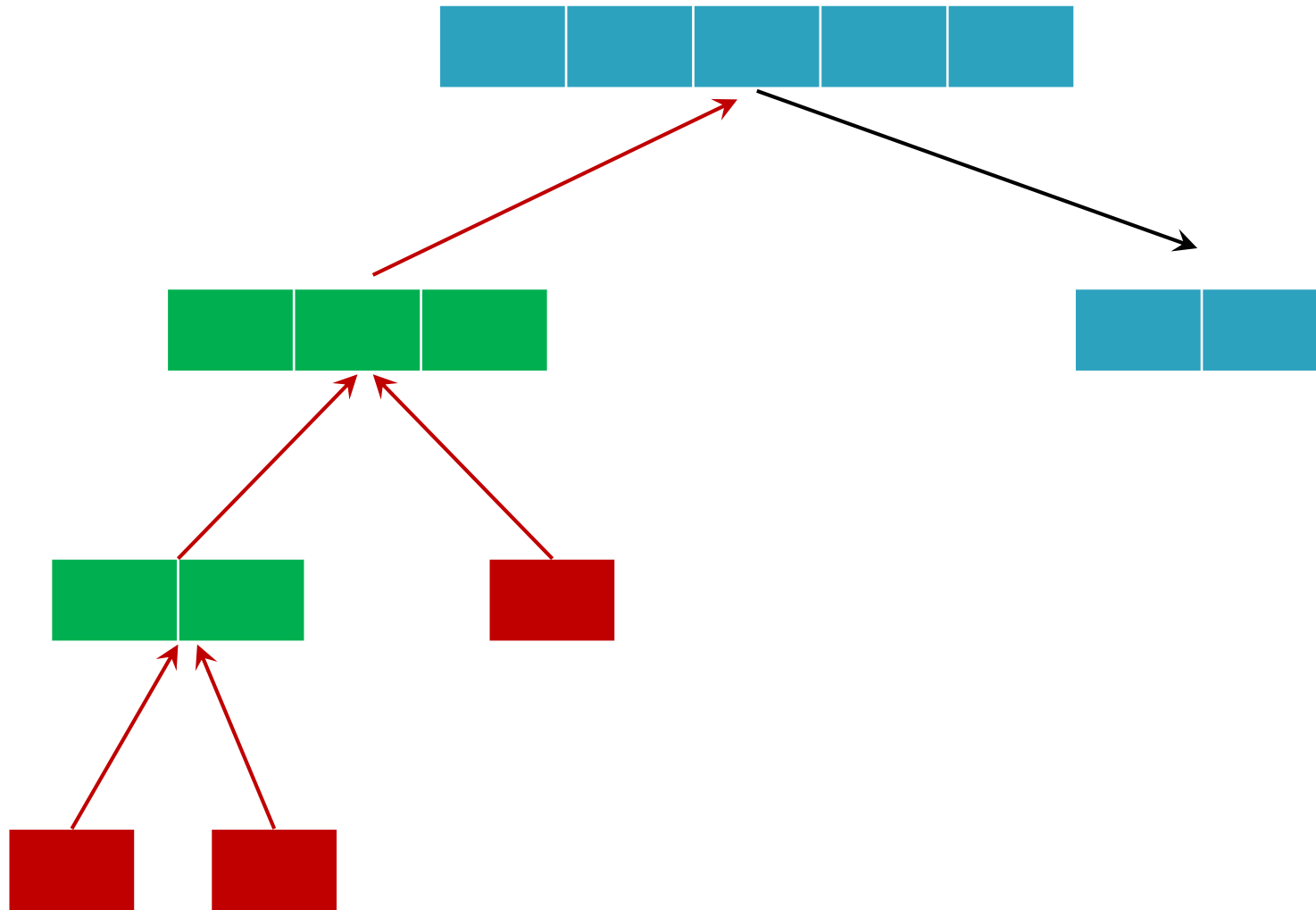


# Metoda Divide et Impera

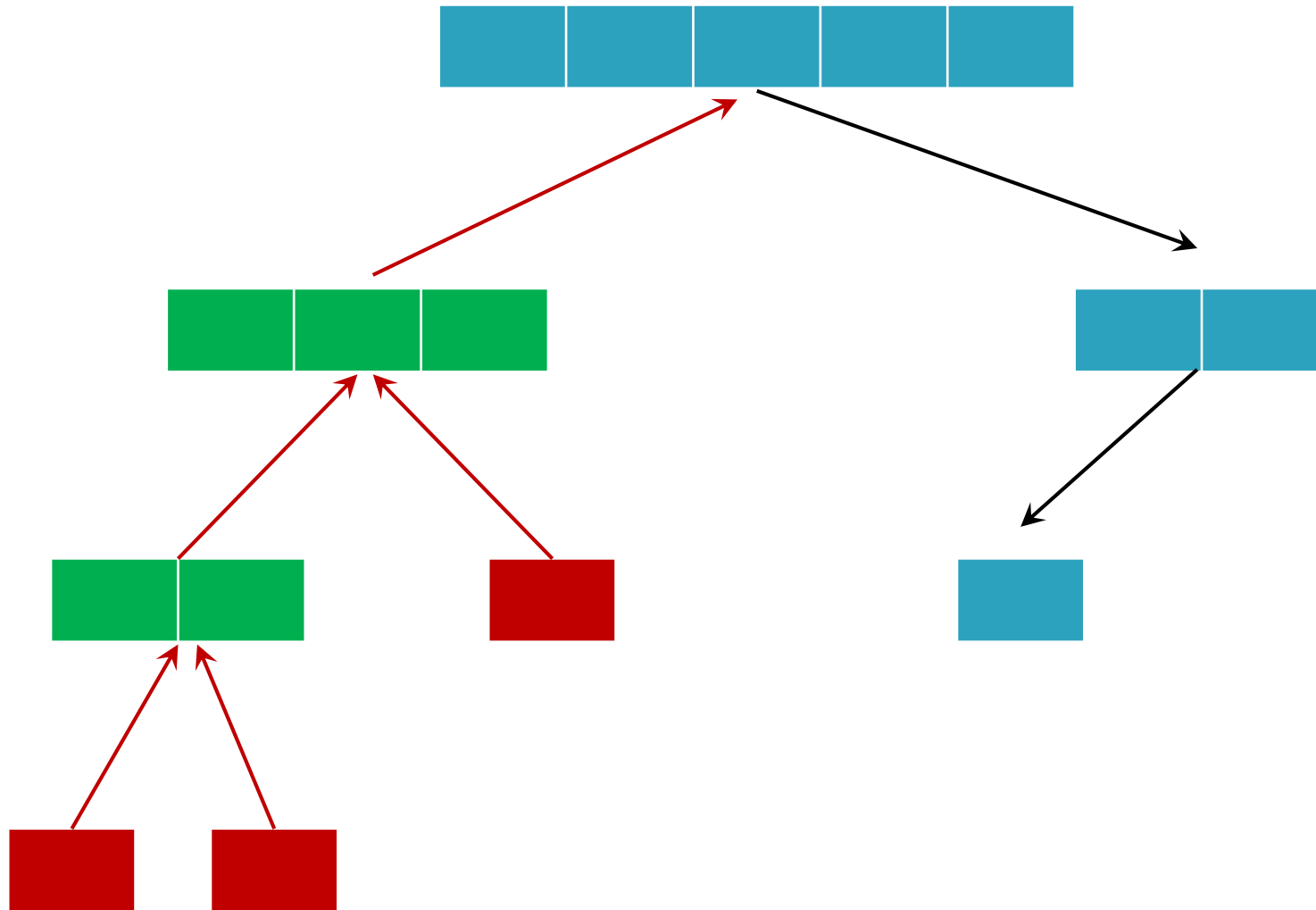




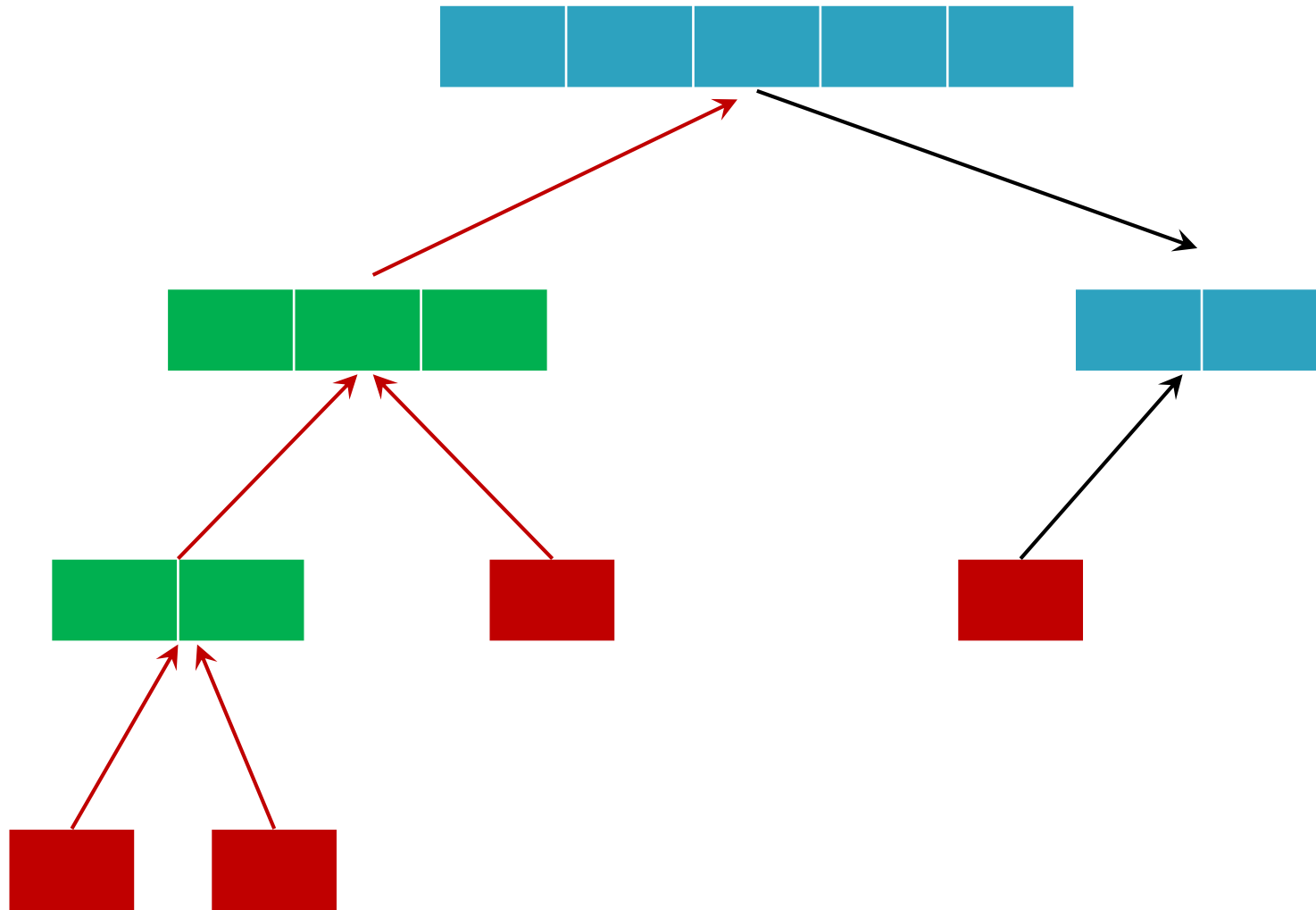
# Metoda Divide et Impera



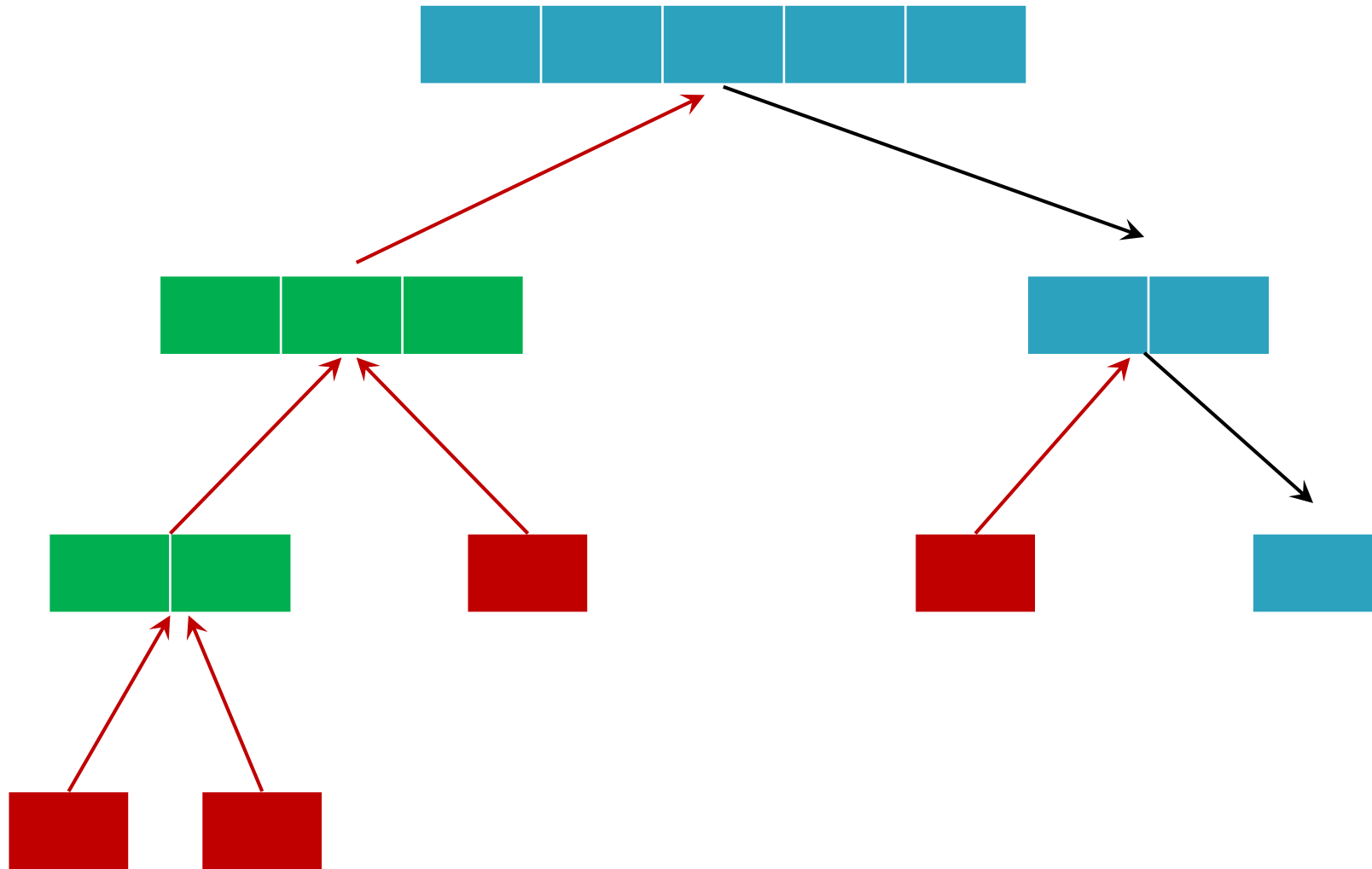
# Metoda Divide et Impera



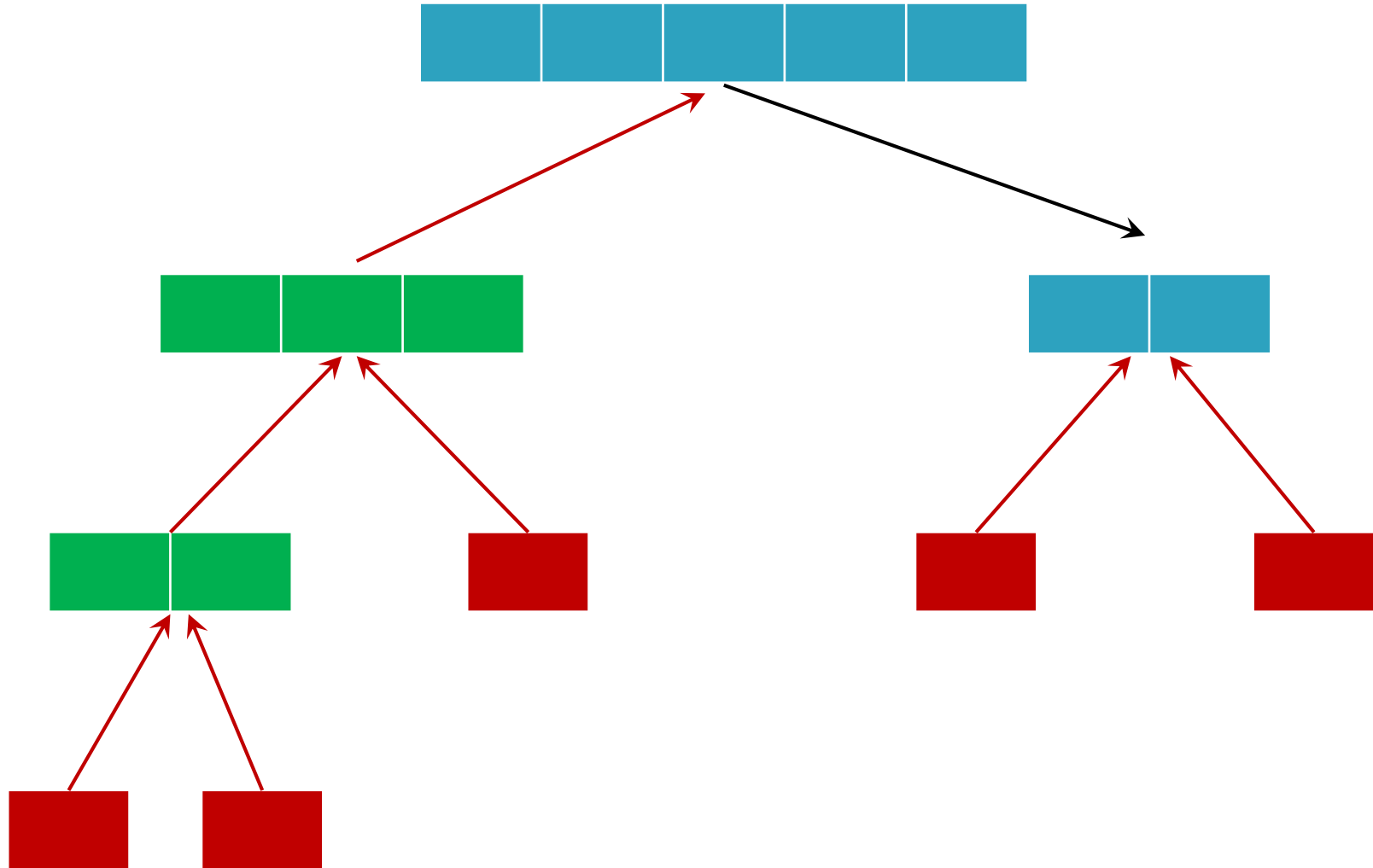
# Metoda Divide et Impera



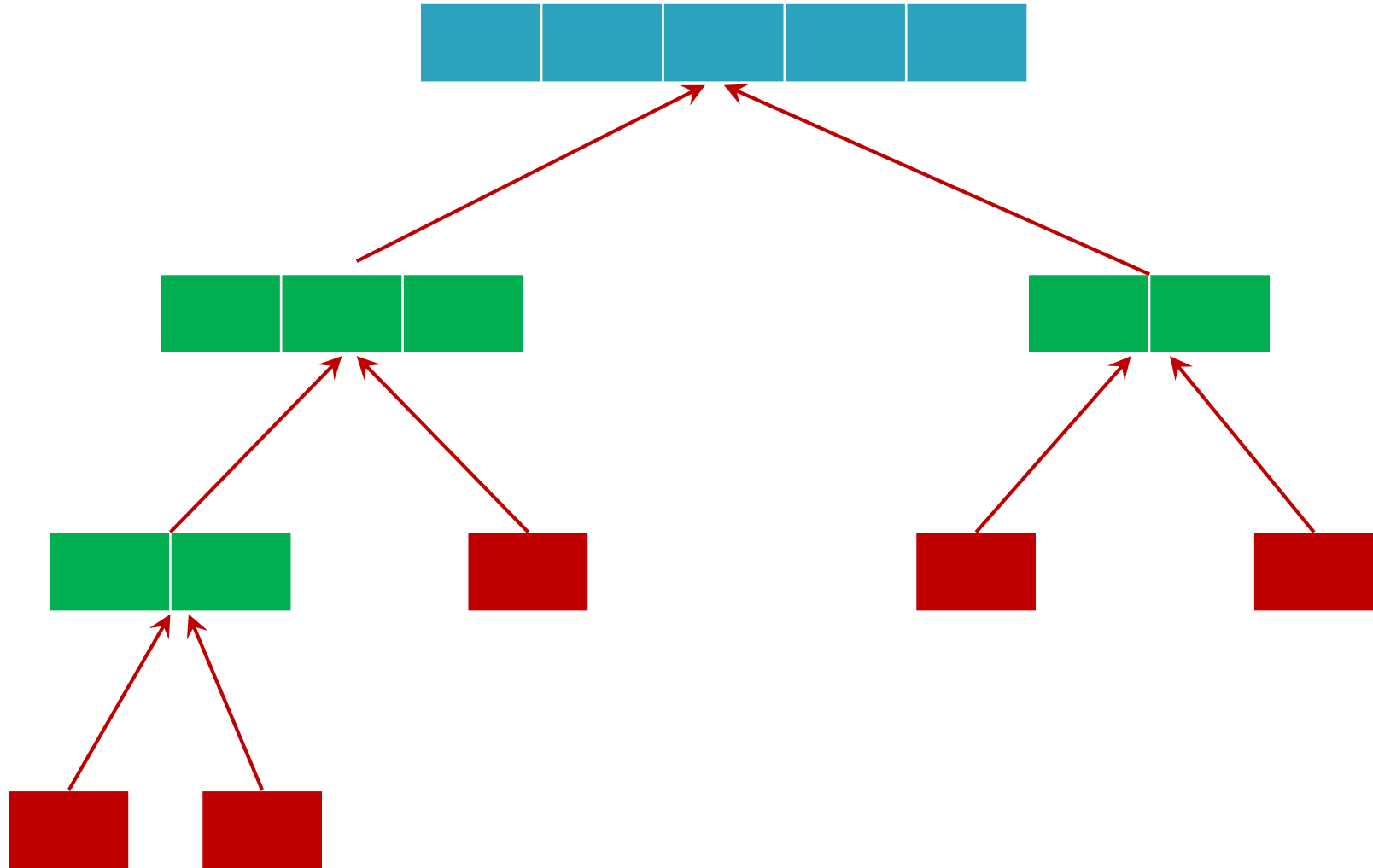
# Metoda Divide et Impera



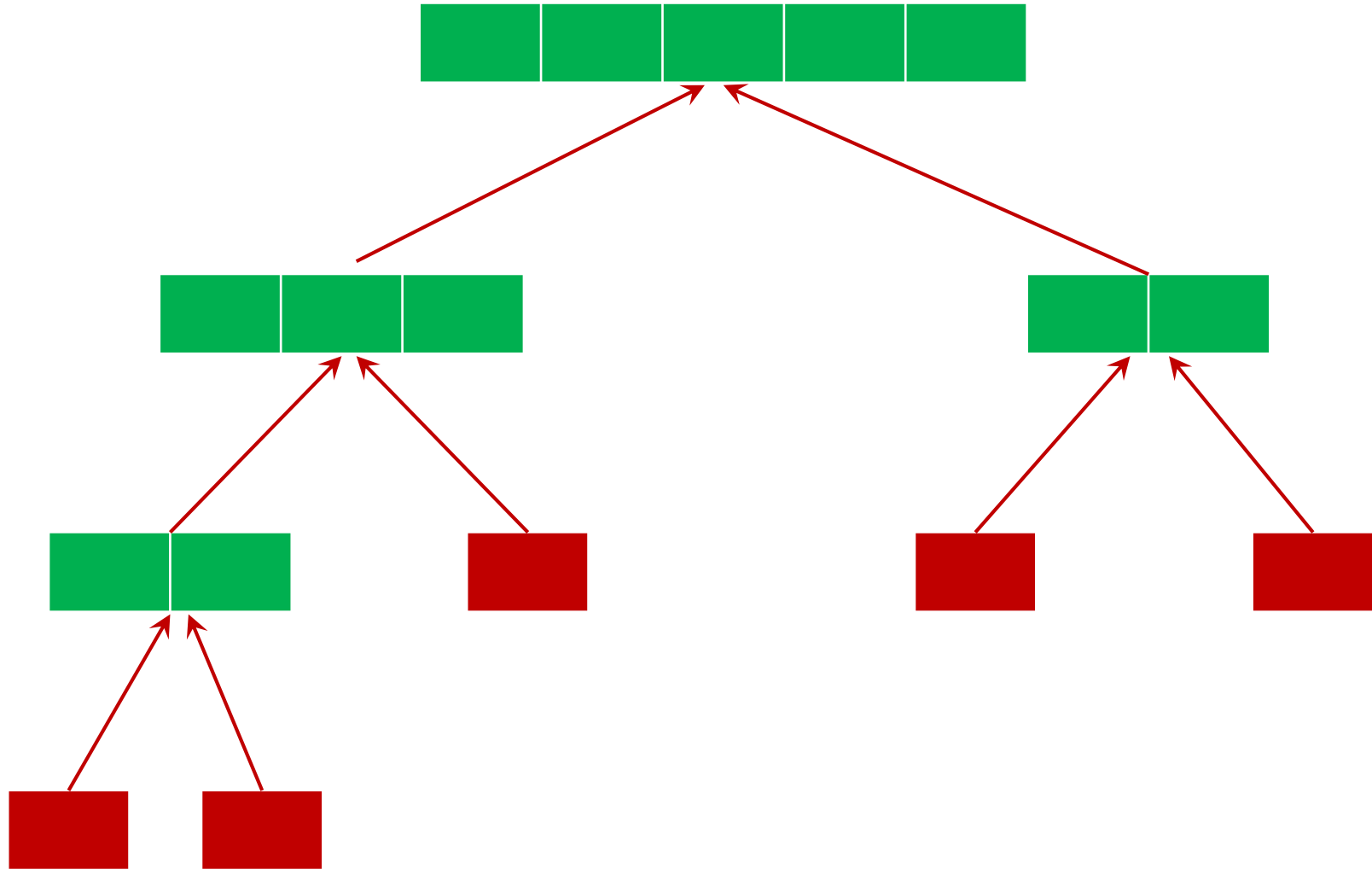
# Metoda Divide et Impera



# Metoda Divide et Impera



# Metoda Divide et Impera



# Exemplu –maximul elementelor unui vector

```
function DivImp(p,u)
    if u==p  $\longrightarrow$  subproblema de dimensiune 1:u-p<1
        r  $\leftarrow$  a[p]
    else
        m  $\leftarrow$   $\lfloor (p+u)/2 \rfloor$ 
        r1  $\leftarrow$  DivImp(p,m)
        r2  $\leftarrow$  DivImp(m+1,u)
        if r1>r2
            r  $\leftarrow$  r1
        else
            r  $\leftarrow$  r2
    return r
```

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



# Metoda Divide et Impera

3

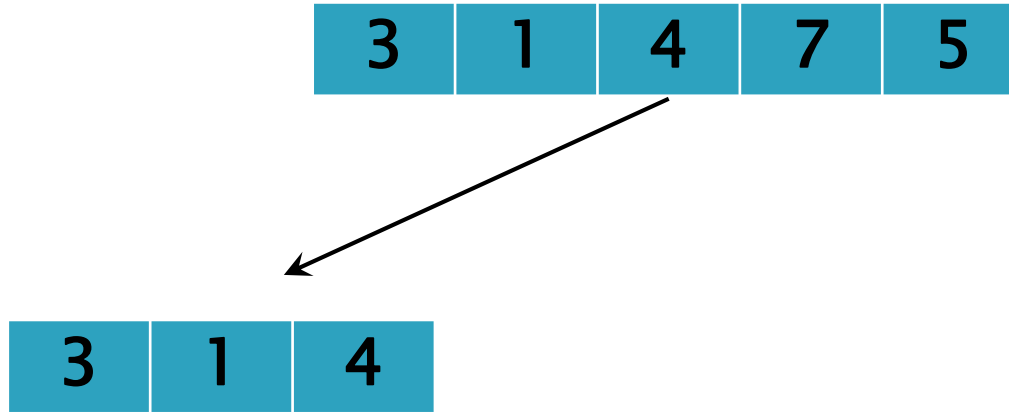
1

4

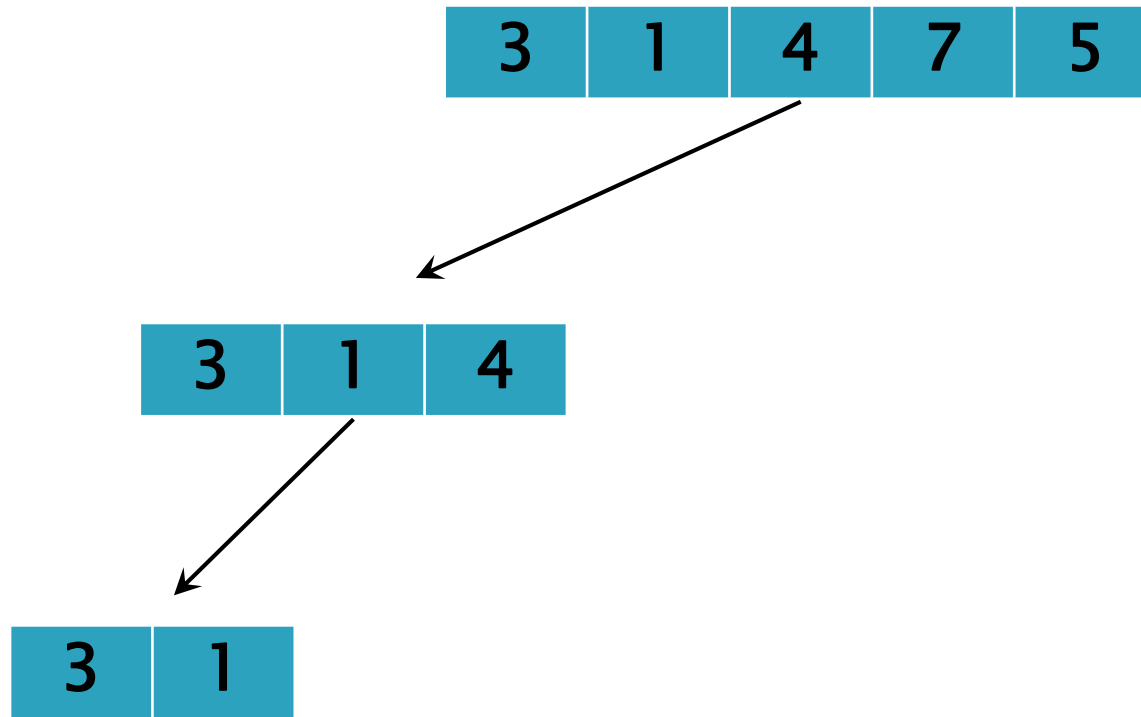
7

5

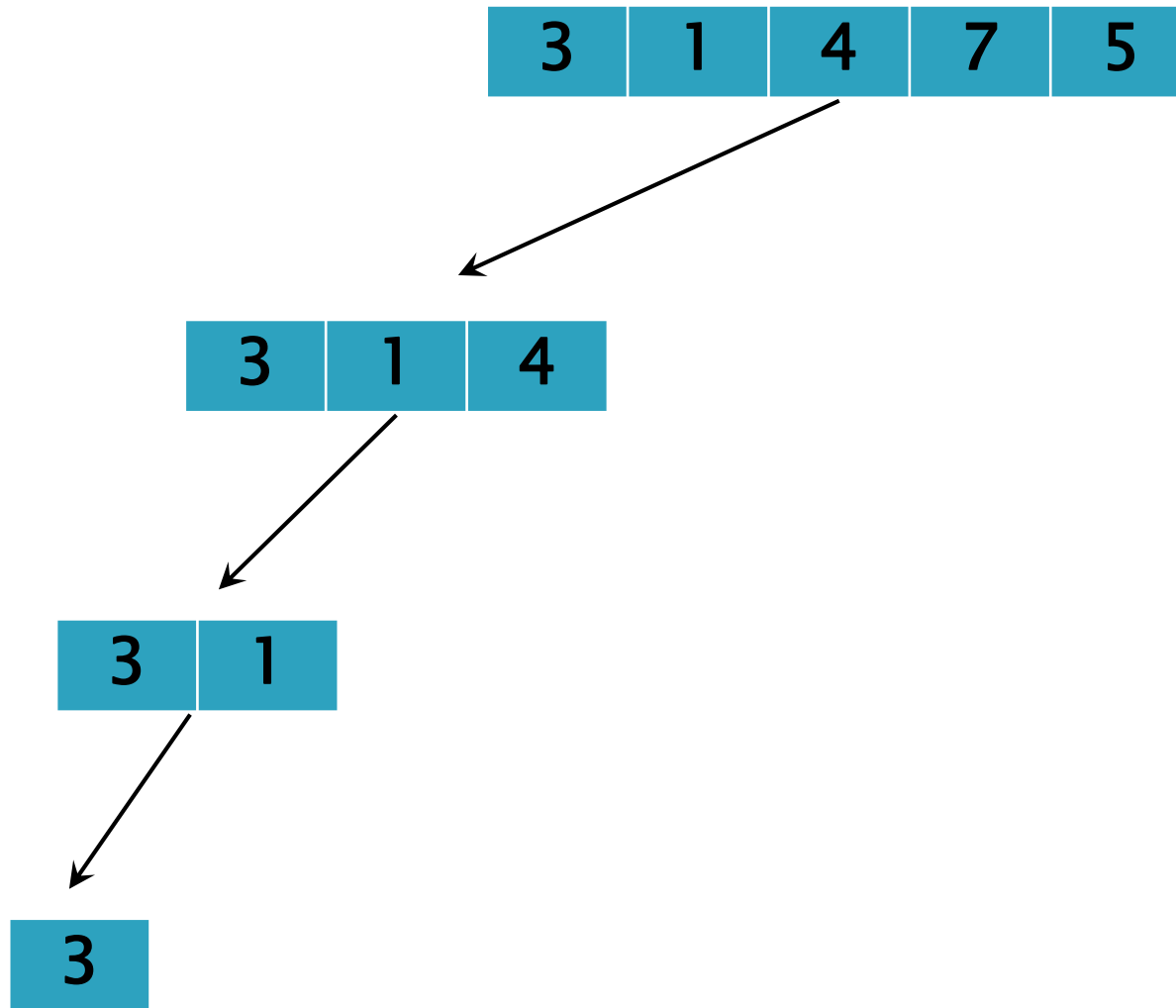
# Metoda Divide et Impera



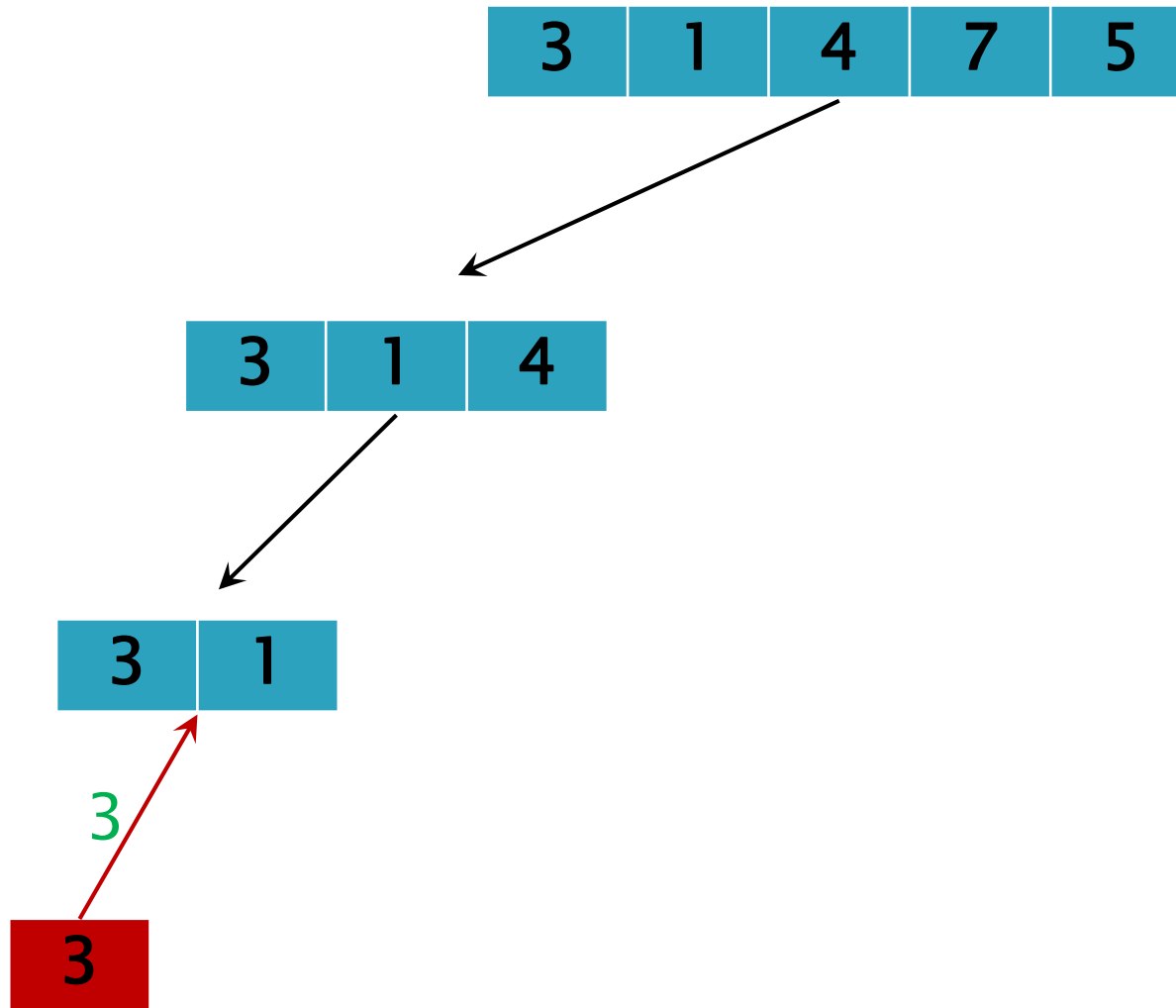
# Metoda Divide et Impera



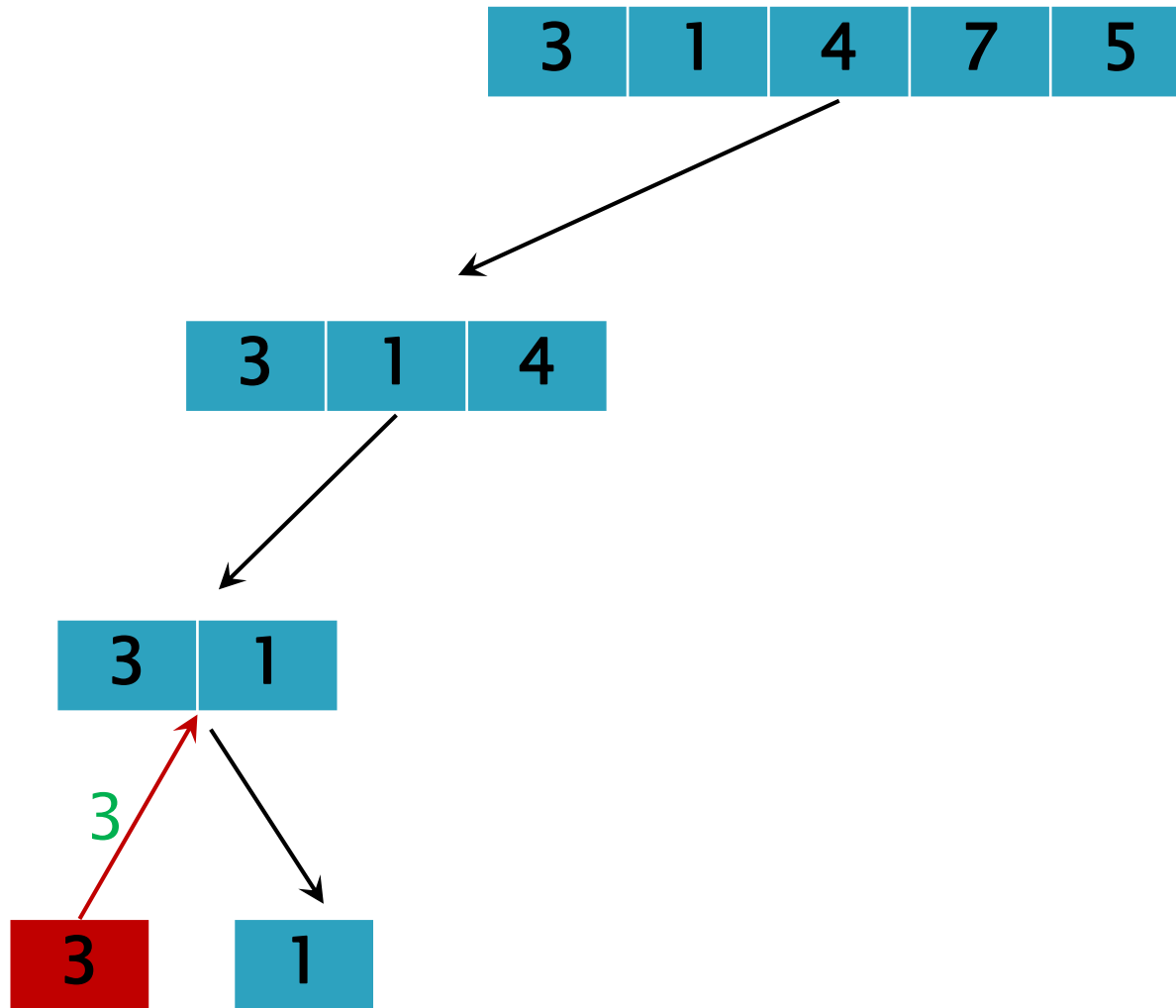
# Metoda Divide et Impera



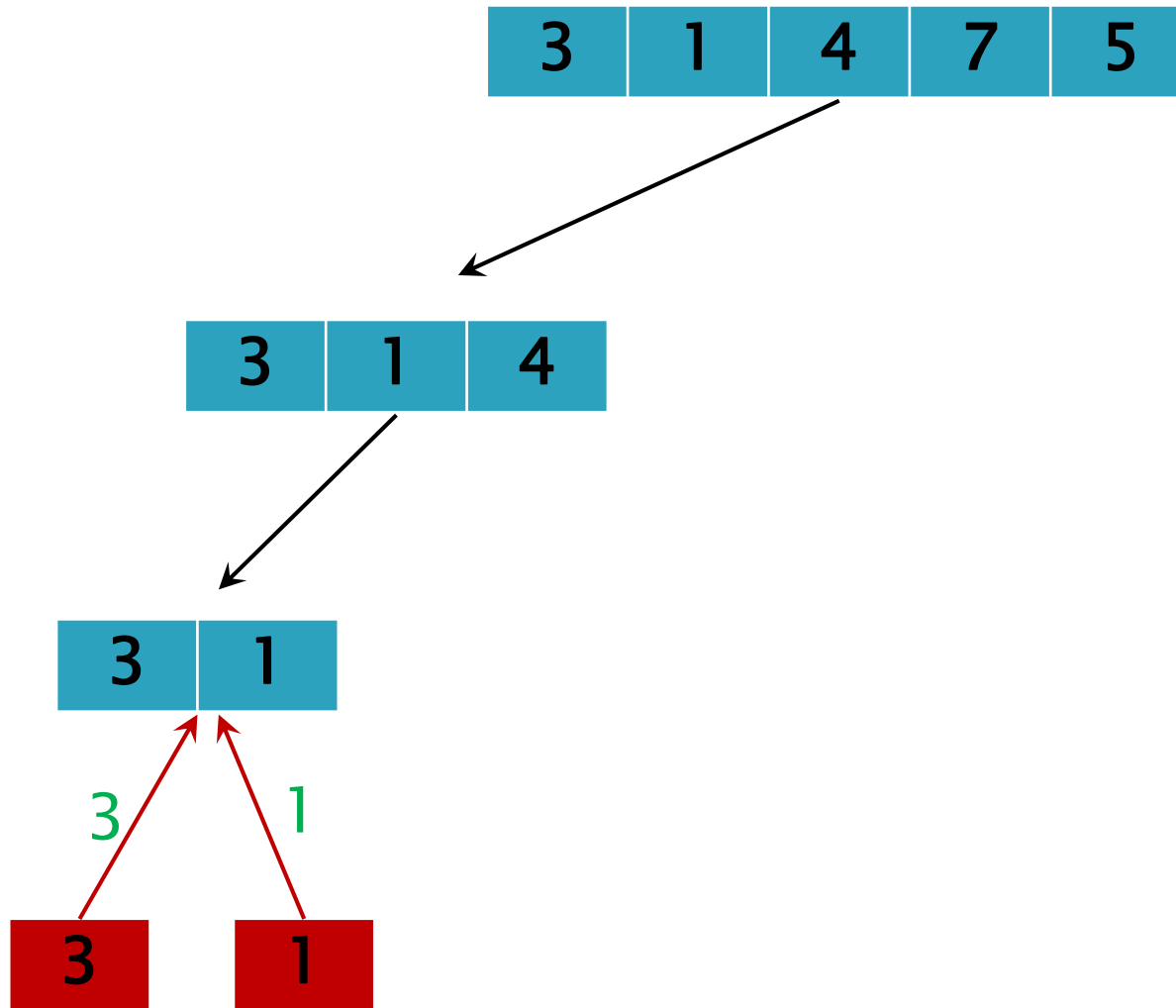
# Metoda Divide et Impera



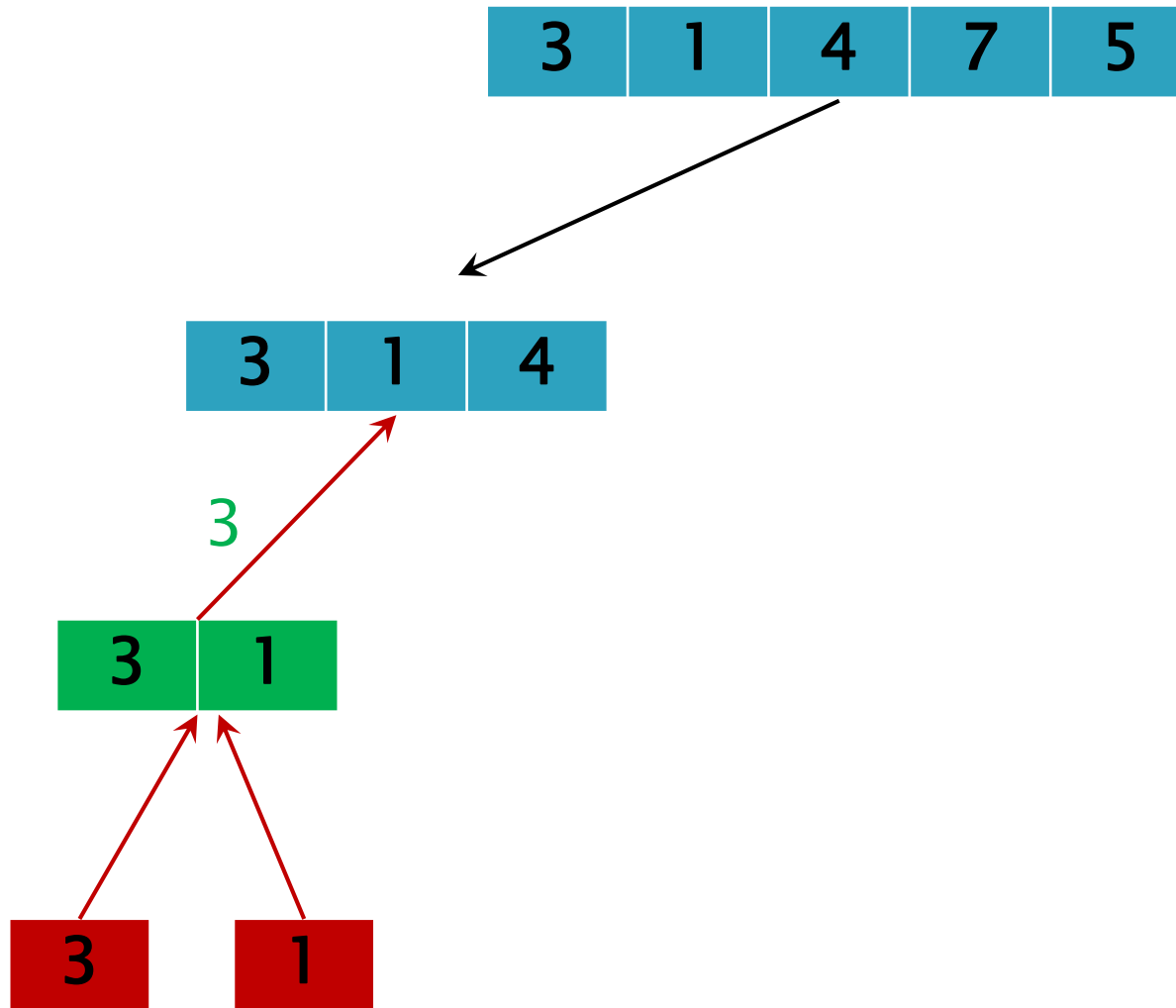
# Metoda Divide et Impera



# Metoda Divide et Impera

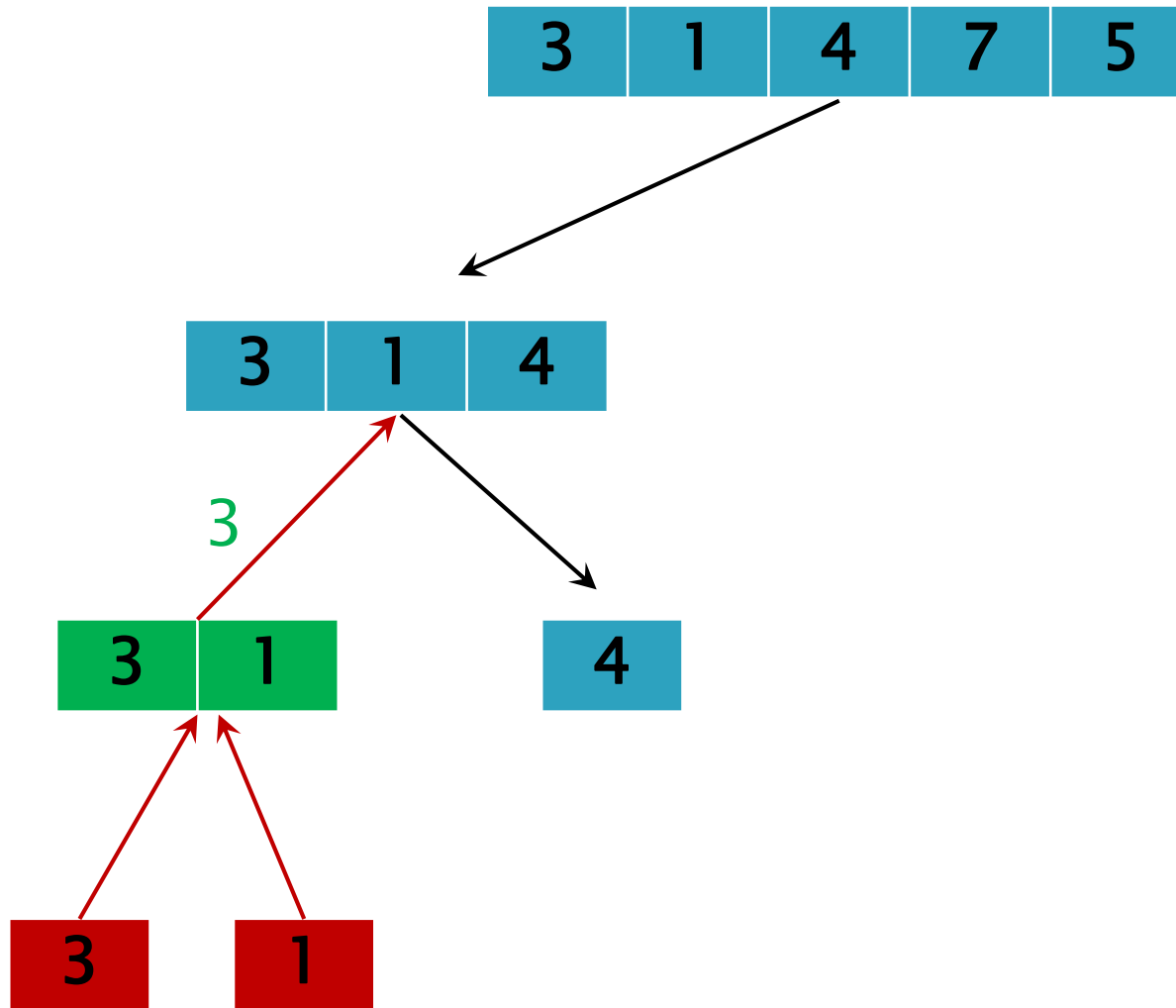


# Metoda Divide et Impera

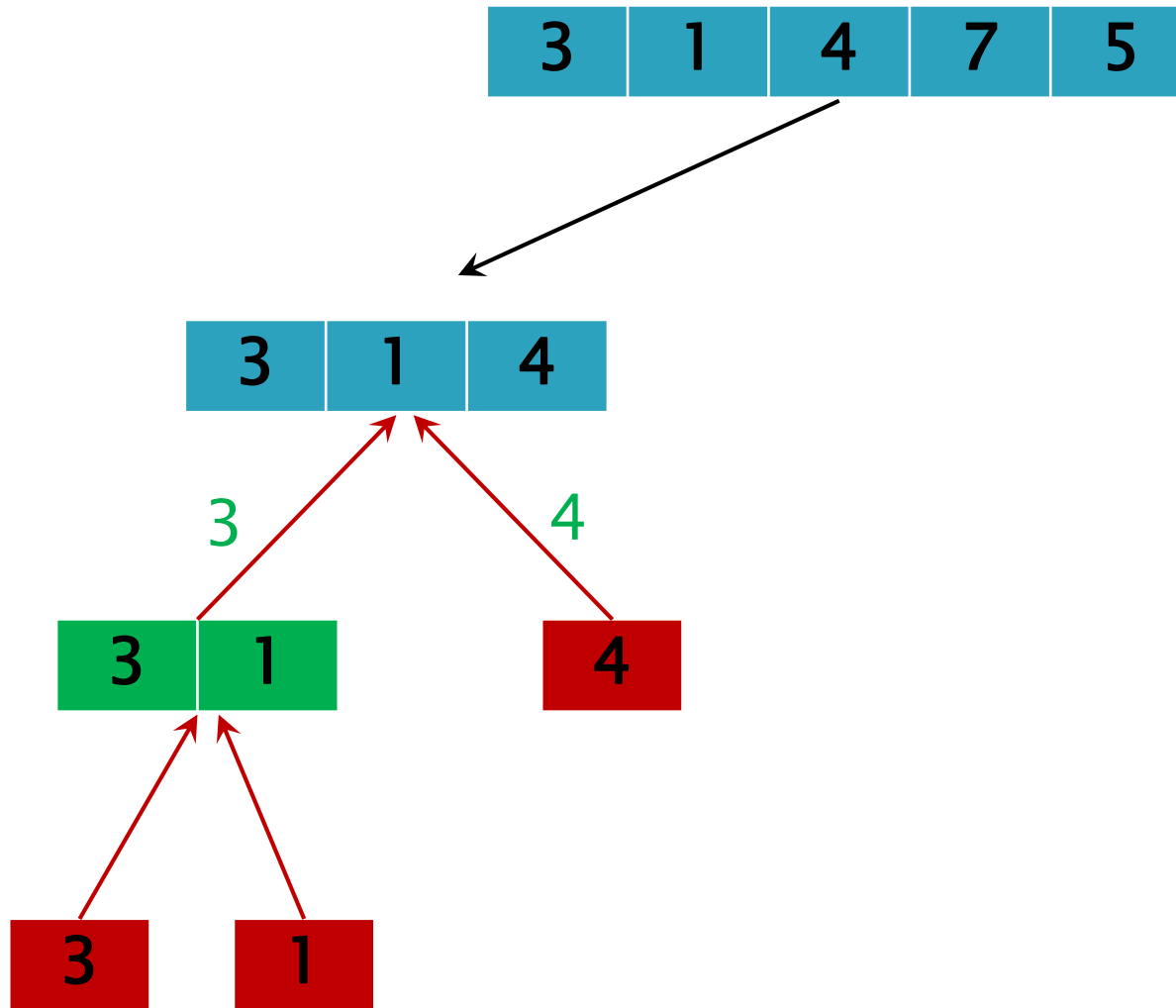




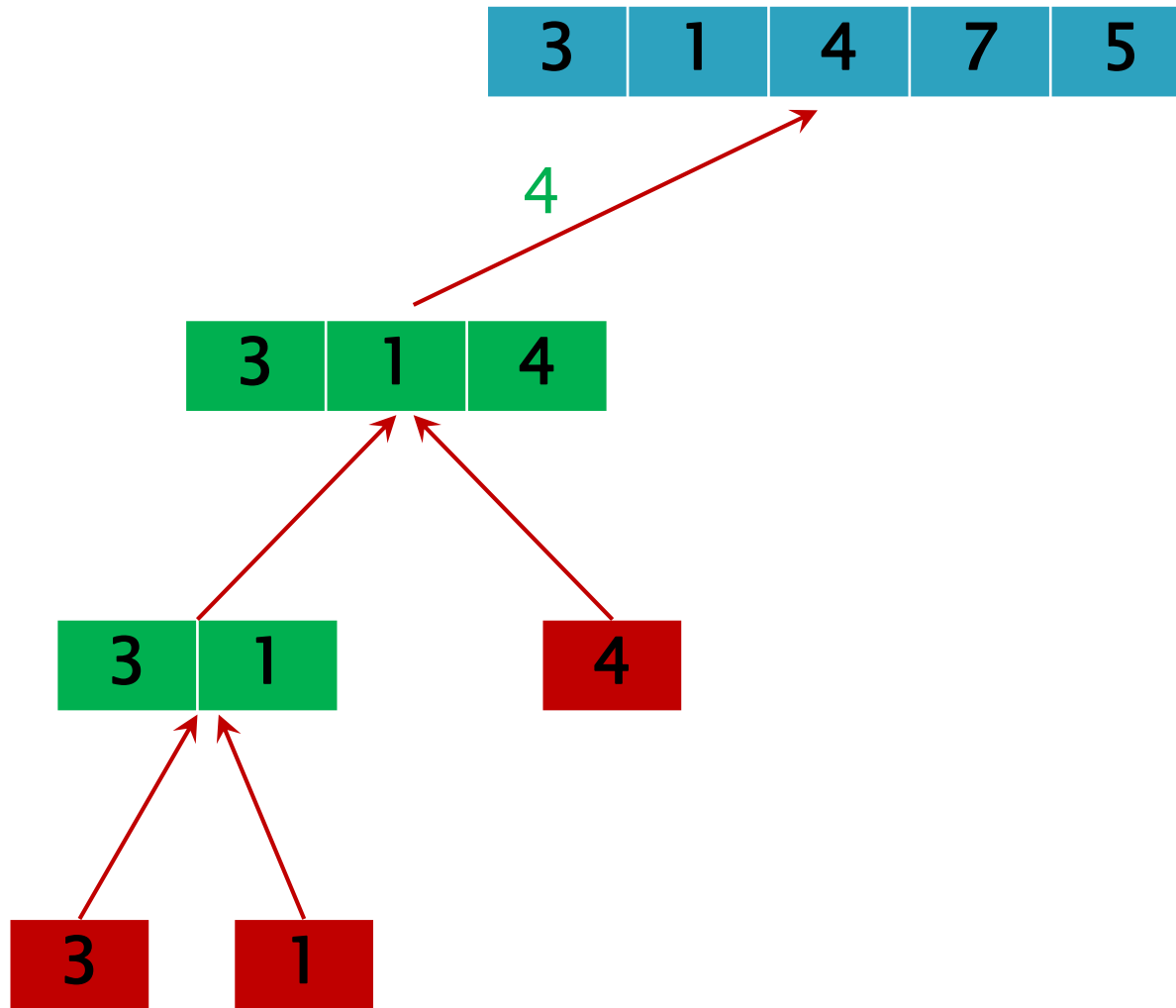
# Metoda Divide et Impera



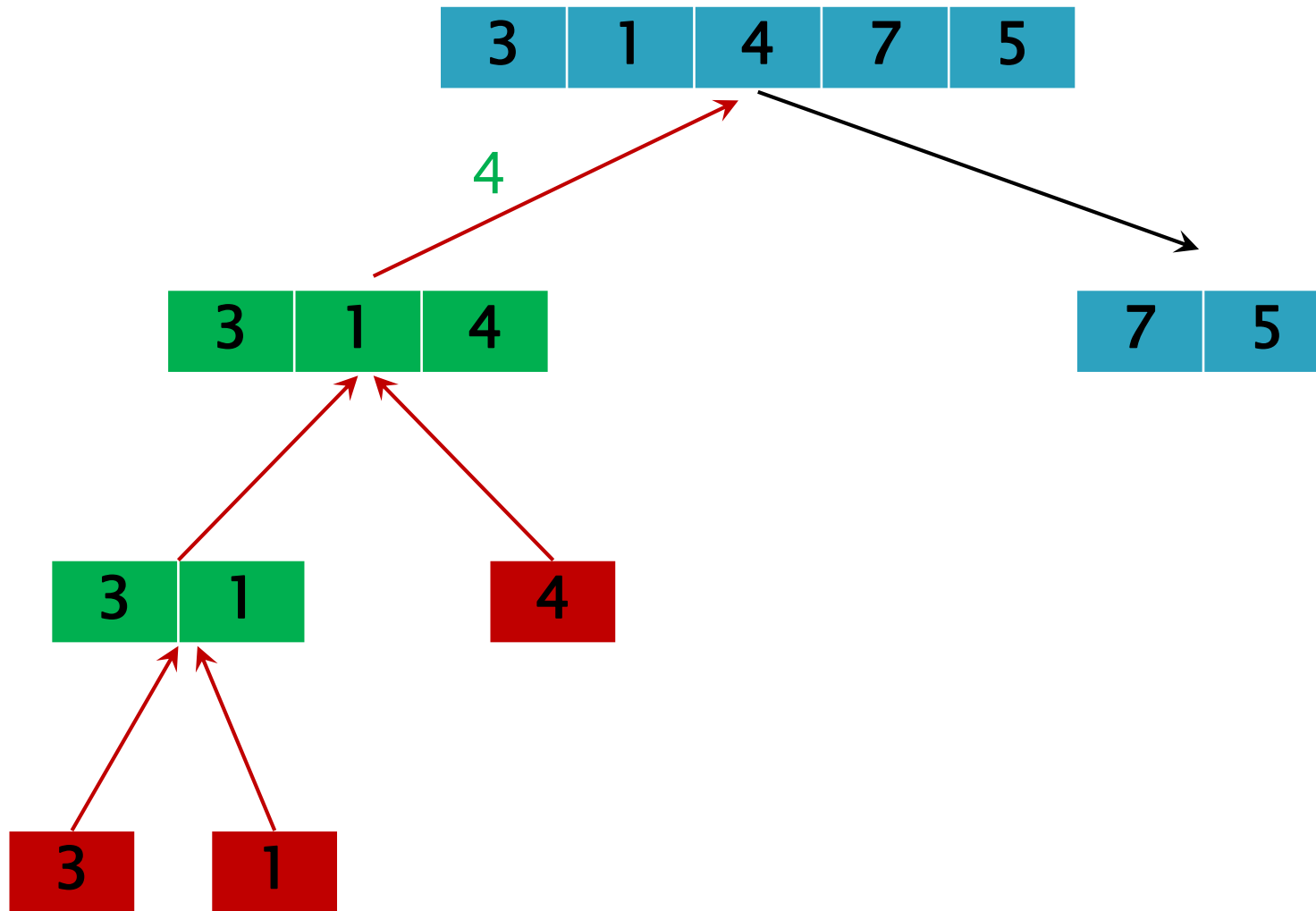
# Metoda Divide et Impera



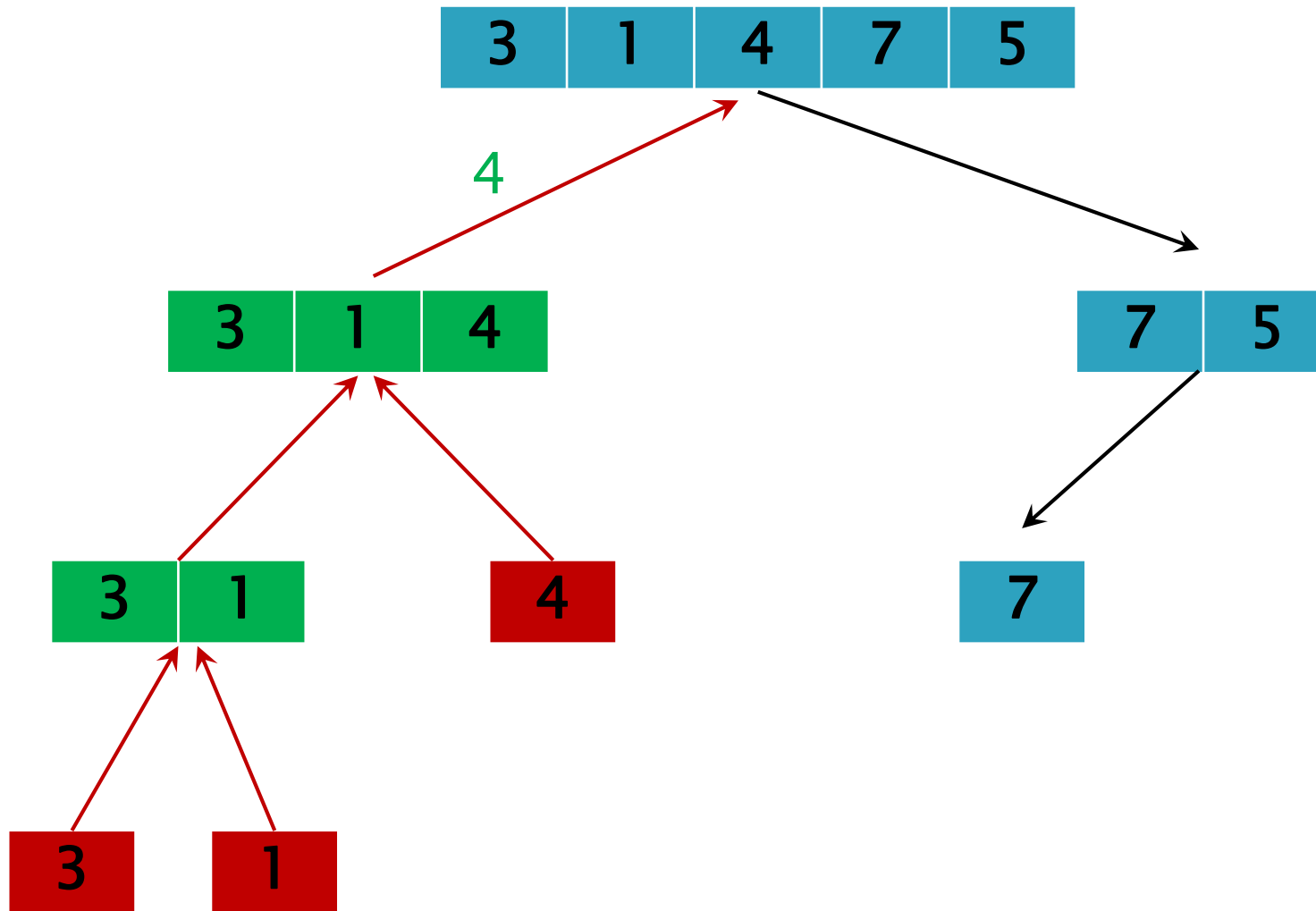
# Metoda Divide et Impera



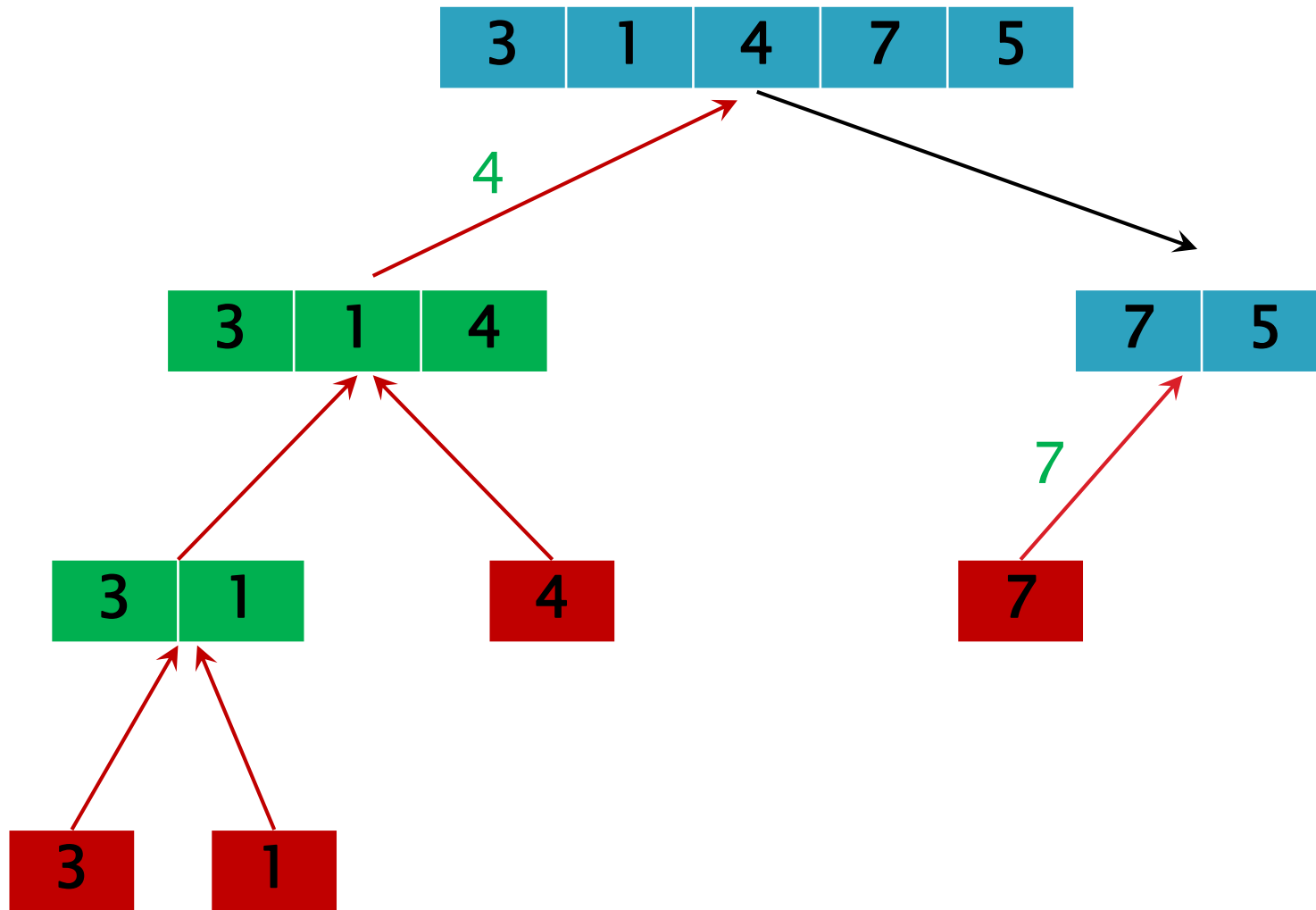
# Metoda Divide et Impera



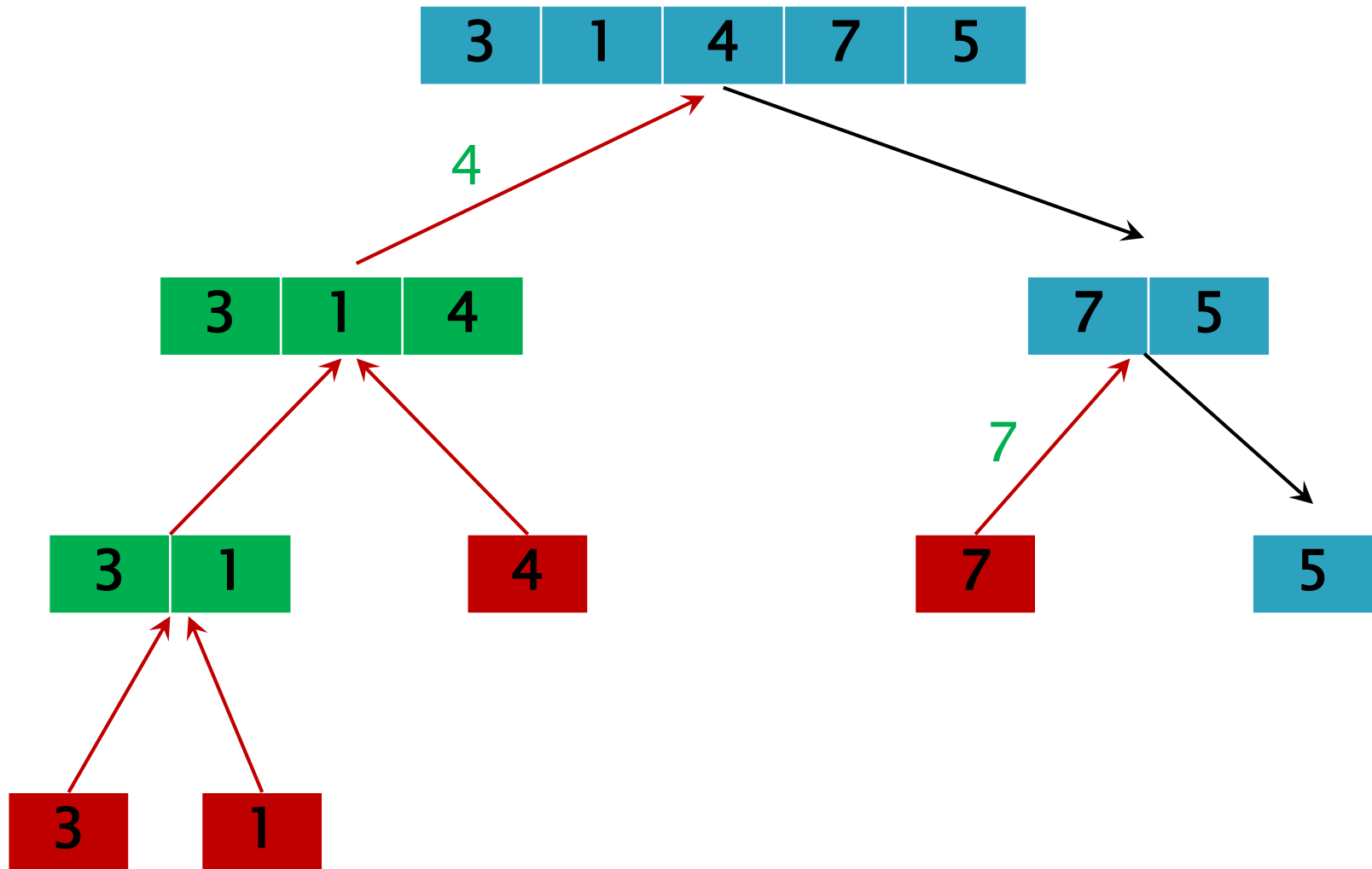
# Metoda Divide et Impera



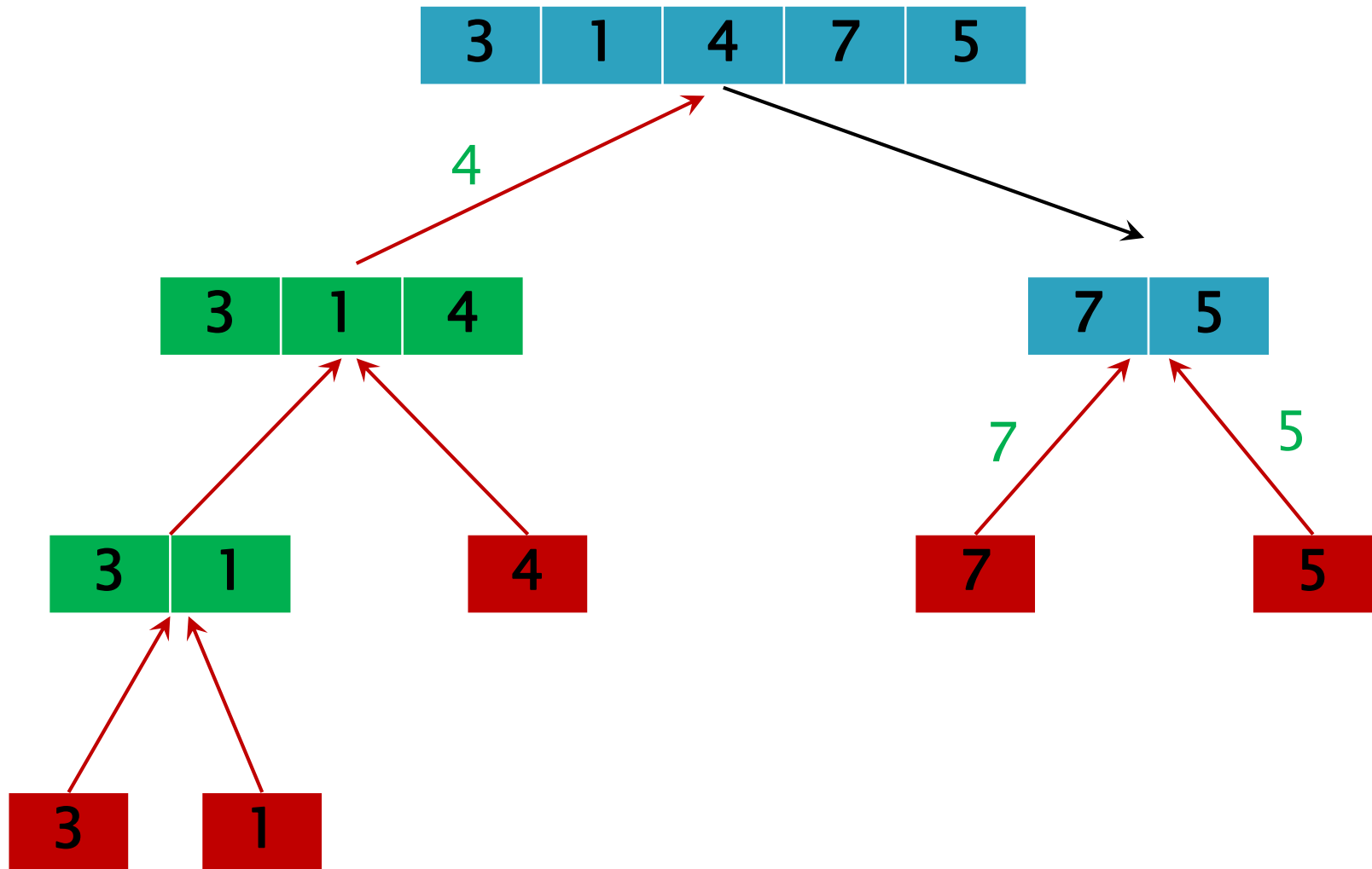
# Metoda Divide et Impera



# Metoda Divide et Impera

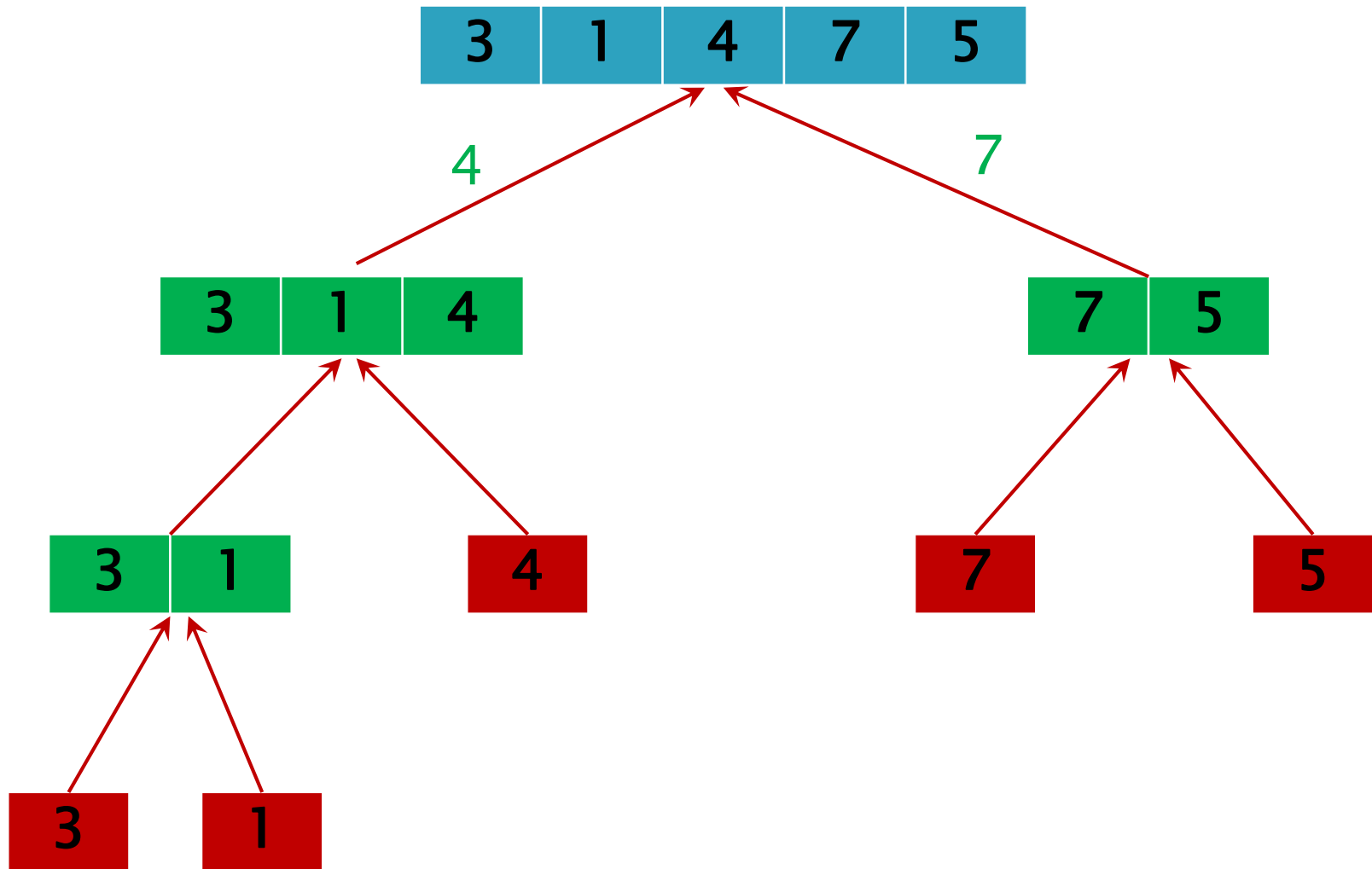


# Metoda Divide et Impera

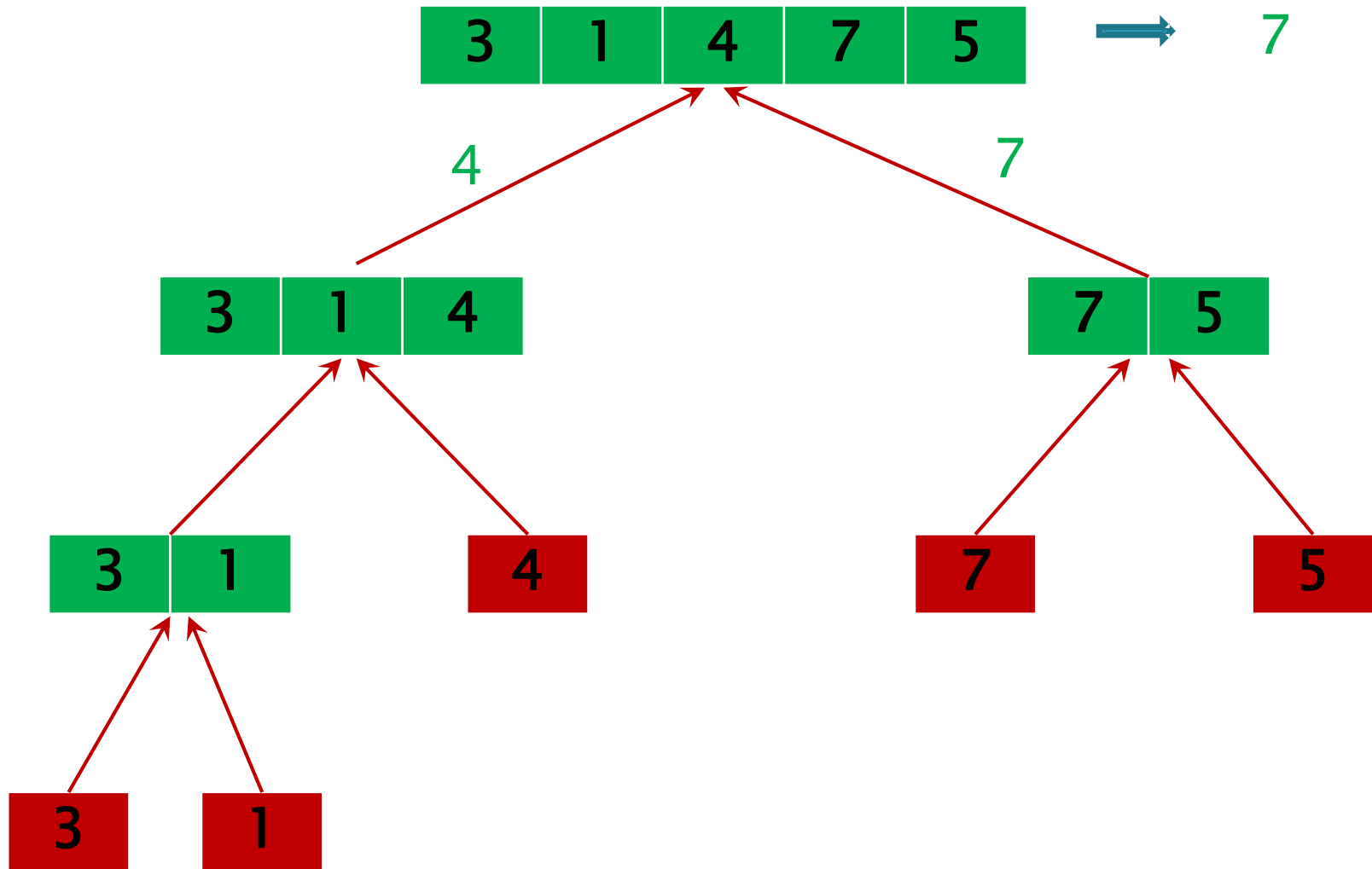




# Metoda Divide et Impera



# Metoda Divide et Impera



# Metoda Divide et Impera

- ▶ Complexitate – relație de recurență

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

# Metoda Divide et Impera

## ► Complexitate – relație de recurență

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

- Suficient să presupunem că  $n = 2^k$   
(apoi înlocuim  $=$  cu  $\leq$  în relații)

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

# Metoda Divide et Impera

## ► Complexitate – relație de recurență

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

- Suficient să presupunem că  $n = 2^k$   
(apoi înlocuim  $=$  cu  $\leq$  în relații)
- Rezolvarea recurenței
  - metoda substituțiilor repetate
  - arbori de recurențe

=> demonstrație Teorema master

# Metoda Divide et Impera

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

# Metoda Divide et Impera

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

# Metoda Divide et Impera

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



# Metoda Divide et Impera

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

# Metoda Divide et Impera

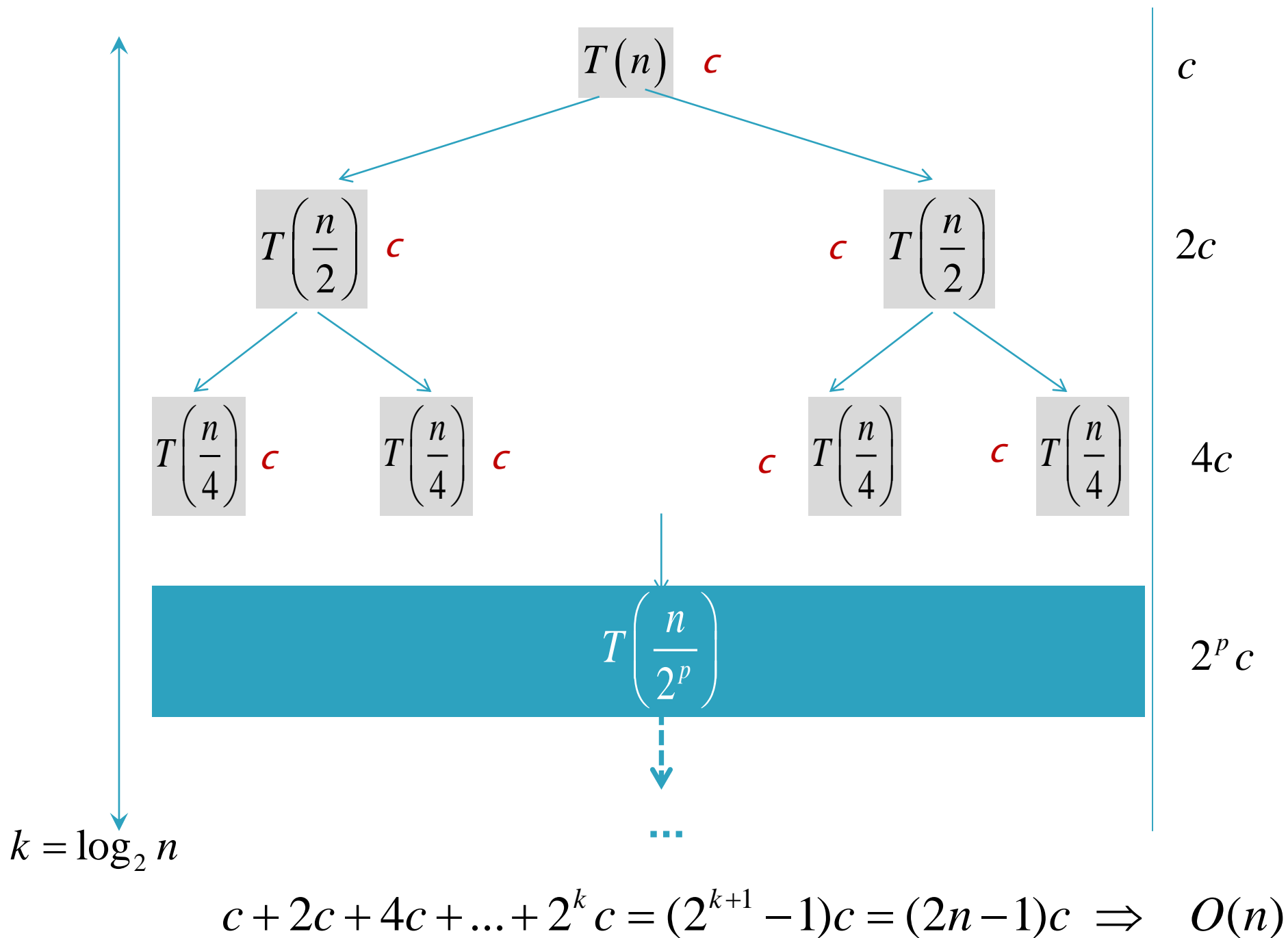
$$\begin{aligned}T(n) &= T(2^k) = 2 \cdot T(2^{k-1}) + c = \\&= 2 \cdot [2T(2^{k-2}) + c] + c = \\&= 2^2 \cdot T(2^{k-2}) + (1+2)c = \\&= 2^2 \cdot [2T(2^{k-3}) + c] + (1+2)c = \\&= 2^3 \cdot T(2^{k-3}) + (1+2+2^2)c = \dots \\&= \end{aligned}$$

# Metoda Divide et Impera

$$\begin{aligned}T(n) &= T(2^k) = 2 \cdot T(2^{k-1}) + c = \\&= 2 \cdot [2T(2^{k-2}) + c] + c = \\&= 2^2 \cdot T(2^{k-2}) + (1+2)c = \\&= 2^2 \cdot [2T(2^{k-3}) + c] + (1+2)c = \\&= 2^3 \cdot T(2^{k-3}) + (1+2+2^2)c = \dots \\&= 2^k \cdot T(2^0) + (1+2+2^2+\dots+2^{k-1})c =\end{aligned}$$

# Metoda Divide et Impera

$$\begin{aligned}T(n) &= T(2^k) = 2 \cdot T(2^{k-1}) + c = \\&= 2 \cdot [2T(2^{k-2}) + c] + c = \\&= 2^2 \cdot T(2^{k-2}) + (1+2)c = \\&= 2^2 \cdot [2T(2^{k-3}) + c] + (1+2)c = \\&= 2^3 \cdot T(2^{k-3}) + (1+2+2^2)c = \dots \\&= 2^k \cdot T(2^0) + (1+2+2^2+\dots+2^{k-1})c = \\&= 2^k \cdot c + (2^k-1)c = cn + (n-1)c = (2n-1)c \Rightarrow \\&\Rightarrow T(n) = O(n)\end{aligned}$$



# Metoda Divide et Impera

- ▶ În termeni de arbori, DI constă în
  - **construirea dinamică a unui arbore** (prin împărțirea în subprobleme)urmată de
  - **parcurgerea în postordine a arborelui** (prin asamblarea rezultatelor parțiale).

# Exemple clasice

- ▶ Căutare binară
- ▶ Sortarea prin interclasare (Merge Sort)
- ▶ Sortarea rapidă (Quick Sort)
- ▶ Problema turnurilor din Hanoi

# Căutarea binară





# Căutarea binară

- ▶ Se consideră vectorul  $a=(a_1,\dots,a_n)$  **ordonat crescător** și o valoare  $x$ . Se cere să se determine dacă  $x$  apare printre componentele vectorului.
- ▶ Mai exact căutăm perechea  $(b,i)$  dată de:
  - $(\text{True}, i)$     dacă  $a_i = x$ ;
  - $(\text{False}, i)$     dacă  **$a_{i-1} < x < a_i$** ,unde, prin convenție,  
$$a_0 = -\infty, a_{n+1} = +\infty.$$

# 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)
```

# Căutarea binară

## Implementare nerecursivă

```
def cautare_binara_nerecursiv(x,ls):  
    p = 0  
    u = len(ls) - 1  
    while p<=u:  
        mij = (p + u) // 2  
        if x == ls[mij]:  
            return (True, mij)  
        elif x < ls[mij]:  
            u = mij - 1  
        else:  
            p = mij + 1  
    return (False,u)
```

# Căutarea binară

► Complexitate:  $O(\log n)$

- $n=2^k$

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

# Căutarea binară

► Complexitate:  $O(\log n)$

- $n=2^k$

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

# Căutarea binară

► Complexitate:  $O(\log n)$

- $n=2^k$

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

# Căutarea binară

► Complexitate:  $O(\log n)$

- $n=2^k$

- $$\begin{aligned} T(n) &= T(n/2) + c = [T(n/2^2) + c] + c = T(n/2^2) + 2c = \\ &= \dots = T(n/2^k) + kc = T(1) + (\log_2 n)c \end{aligned}$$

$k = \log_2 n$

$$\Rightarrow O(k) = O(\log_2 n)$$

# Sortarea prin interclasare

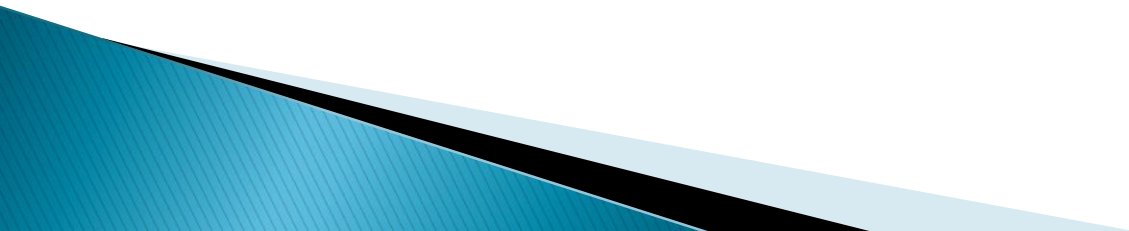


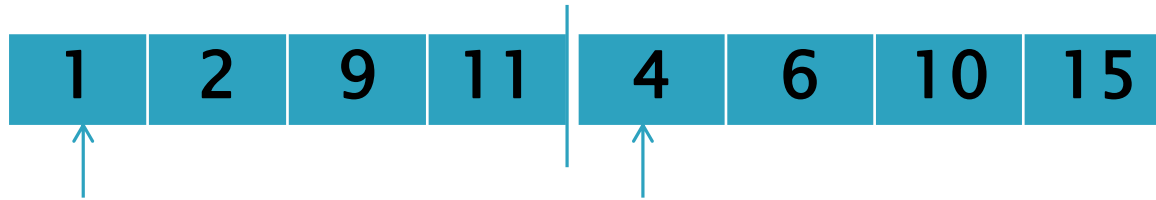


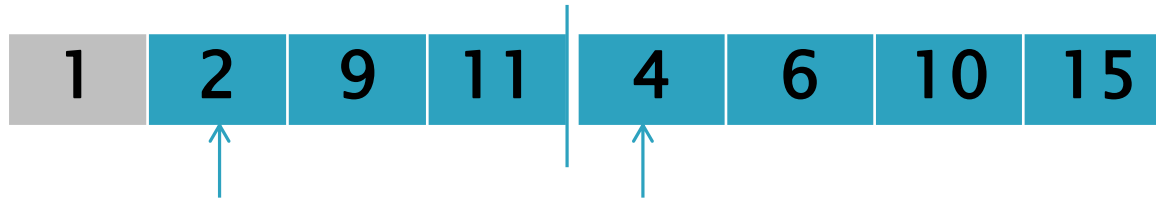
# Sortare prin interclasare

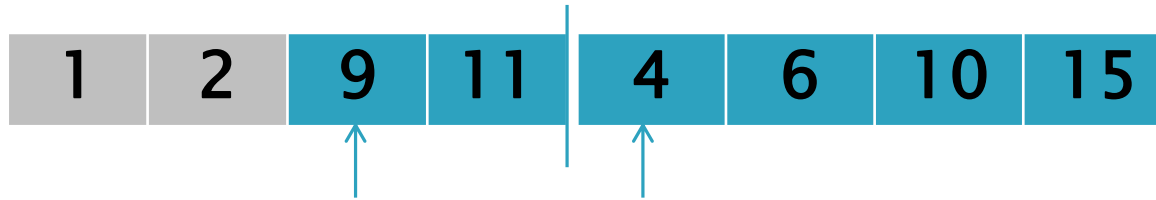
## ▶ Idee:

- împărțim vectorul în doi subvectori
- ordonăm crescător fiecare subvector
- asamblăm rezultatele prin *interclasare*



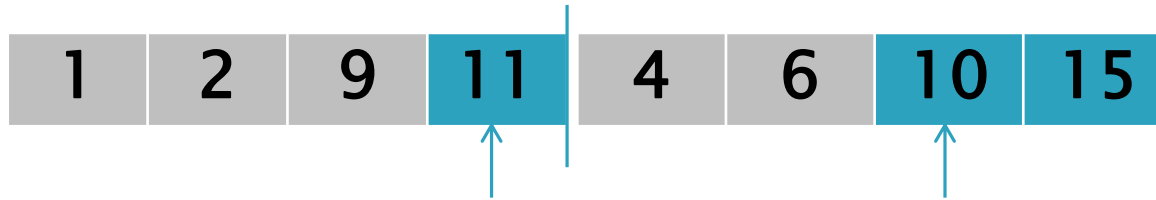


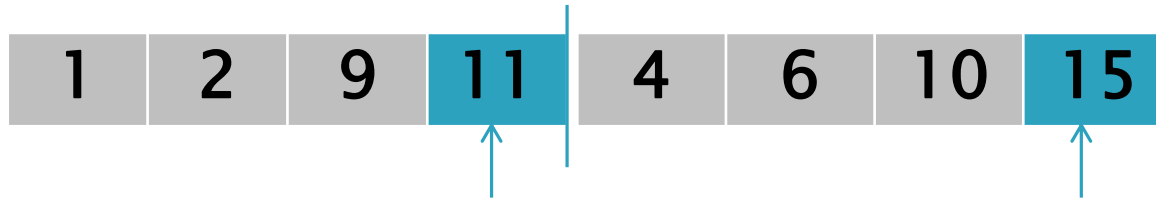




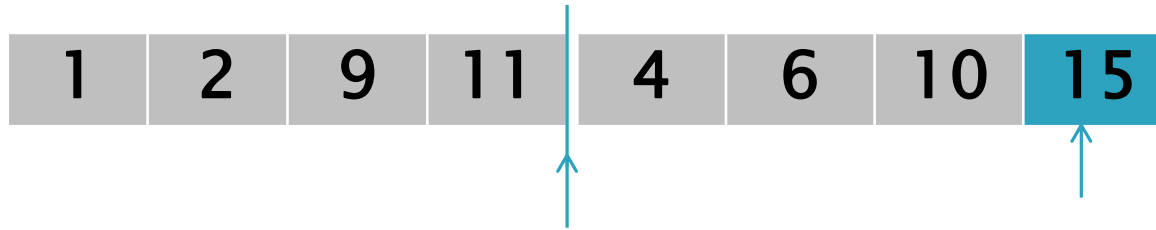


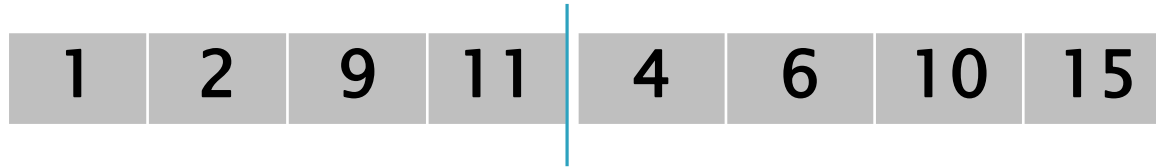






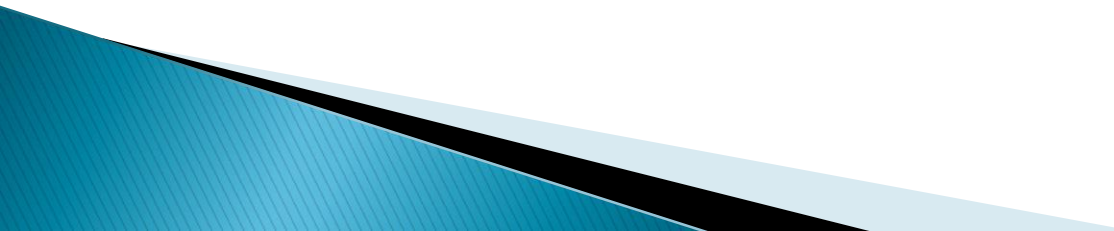






# Sortare prin interclasare

```
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)
```



```
def interclaseaza(a, p, m, u):  
    b = [None]*(u-p+1)  
    k1 = p;    k2 = m + 1; kb = 0  
    while (k1<=m) and (k2 <= u):  
        if a[k1] <= a[k2]:  
            b[kb] = a[k1]; k1 += 1  
        else:  
            b[kb] = a[k2]; k2+= 1  
        kb += 1
```

```

def interclaseaza(a, p, m, u):
    b = [None]*(u-p+1)
    k1 = p;   k2 = m + 1; kb = 0
    while (k1<=m) and (k2 <= u):
        if a[k1] <= a[k2]:
            b[kb] = a[k1]; k1 += 1
        else:
            b[kb] = a[k2]; k2+= 1
        kb += 1

    while k1<=m:
        b[kb] = a[k1]; kb += 1; k1 += 1

    while k2<=u:
        b[kb] = a[k2]; kb += 1; k2 += 1

    for i in range(p,u+1):
        a[i] = b[i-p]

```

# Sortare prin interclasare

- **Complexitate:**
- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn$ , pentru  $n > 1$

# Sortare prin interclasare

- **Complexitate:**
- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn$ , pentru  $n > 1$
- Pentru  $n = 2^k$

$$T(n) = 2T(n/2) + cn, \text{ pentru } n > 1$$

# Complexitate

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



# Complexitate

$$\begin{aligned} T(n) = T(2^k) &= 2 \textcolor{red}{T}(2^{k-1}) + c \cdot 2^k = \\ &= 2 [2\textcolor{red}{T}(2^{k-2}) + \textcolor{red}{c} \cdot 2^{k-1}] + c \cdot 2^k = \end{aligned}$$

# Complexitate

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

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

$$= \dots$$


# Complexitate

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

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

$$= \dots = 2^i T(2^{k-i}) + i \cdot c \cdot 2^k =$$

$$= 2^k T(1) + k \cdot c \cdot 2^k = nc + c \cdot n \cdot \log_2 n$$

# Complexitate

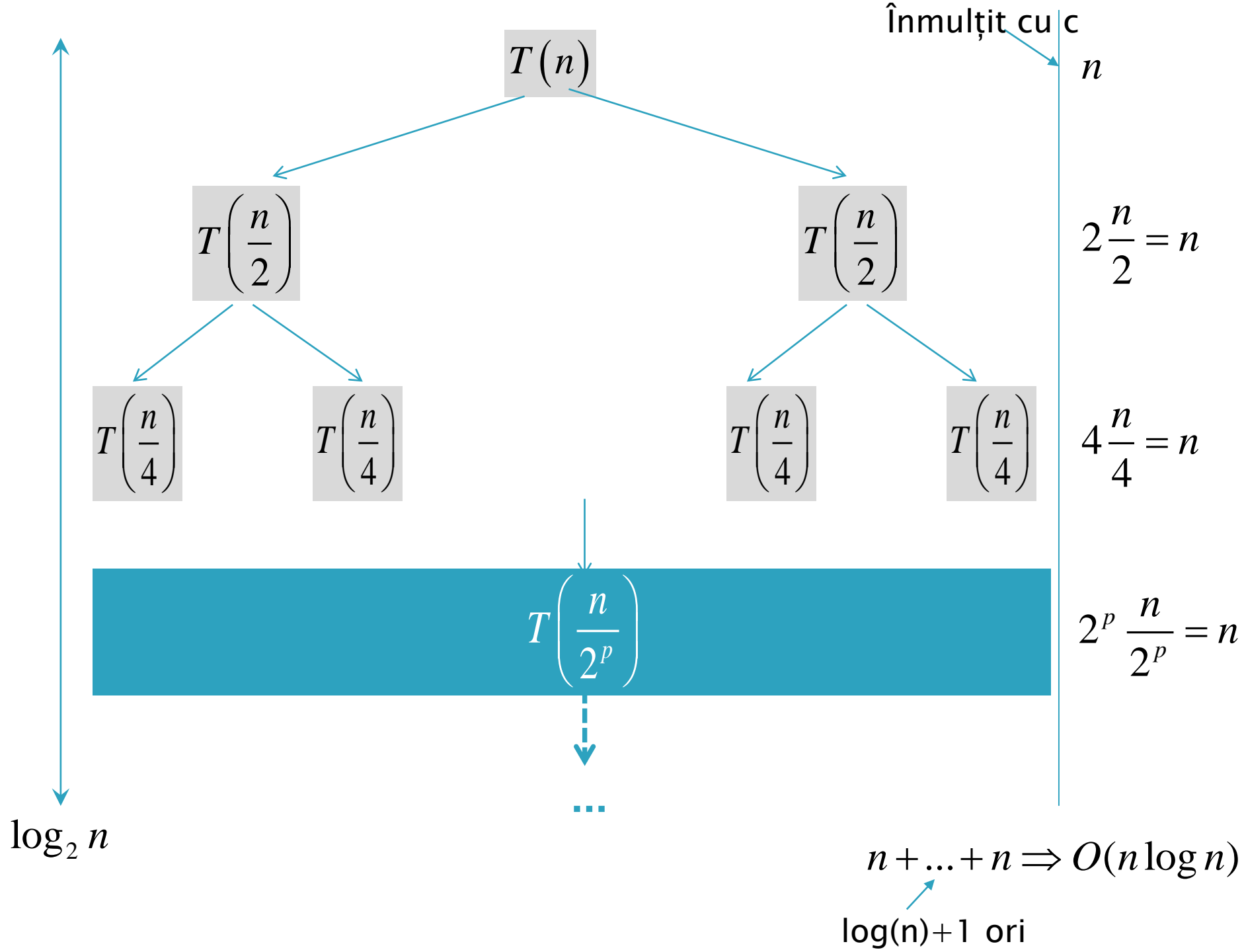
$$T(n) = T(2^k) = 2 \textcolor{red}{T(2^{k-1})} + c \cdot 2^k =$$

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

$$= \dots = 2^i T(2^{k-i}) + i \cdot c \cdot 2^k =$$

$$= 2^k T(1) + k \cdot c \cdot 2^k = nc + c \cdot n \cdot \log_2 n$$

$$\Rightarrow \mathbf{O(n \log(n))}$$



# Aplicație

# Numărare inversiuni

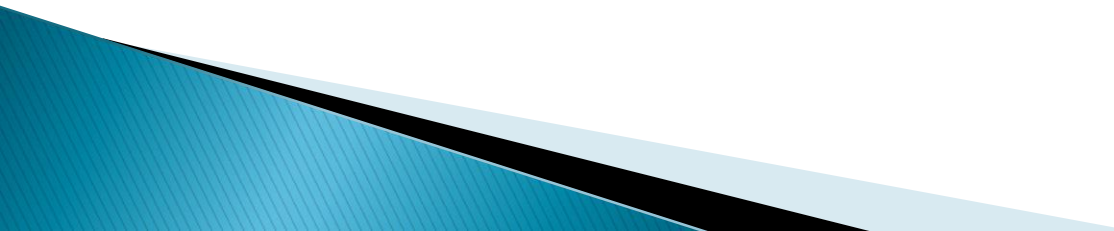


## Problemă

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

- Inversiune = pereche  $(i, j)$  cu proprietatea că  $i < j$  și  $a_i > a_j$
- Exemplu  $1, 2, 11, 9, 4, 6 \Rightarrow 5$  inversiuni  
 $((11, 9), (11, 4), (9, 4), (11, 6), (9, 6))$

# Aplicații

- ▶ **Măsură a diferenței între două liste ordonate**
  - ▶ **“Gradul de ordonare” al unui vector**
  - ▶ **Probleme de analiză a clasificărilor (ranking)**
    - Asemănarea între preferințele a doi utilizatori – sugestii de utilizatori cu preferințe similare
    - Asemănări dintre rezultatele întoarse de motoare diferite de căutare pentru aceeași cerere
    - collaborative filtering
- 



# Aplicații

- ▶ Suficient să presupunem că prima clasificare este

$1, 2, 3, \dots, n$

- ▶ Gradul de asemănare dintre clasificări = numărul de inversiuni din a doua clasificare

# Aplicații

Preferințe  
utilizator 1

Arghezi



Bacovia



Blaga



Barbu



Preferințe  
utilizator 2



Blaga



Arghezi



Barbu



Bacovia

# Aplicații

Preferințe  
utilizator 1

Arghezi

Bacovia

Blaga

Barbu



Blaga

Arghezi

Barbu

Bacovia

Preferințe  
utilizator 2

3 inversiuni

# Numărare inversiuni



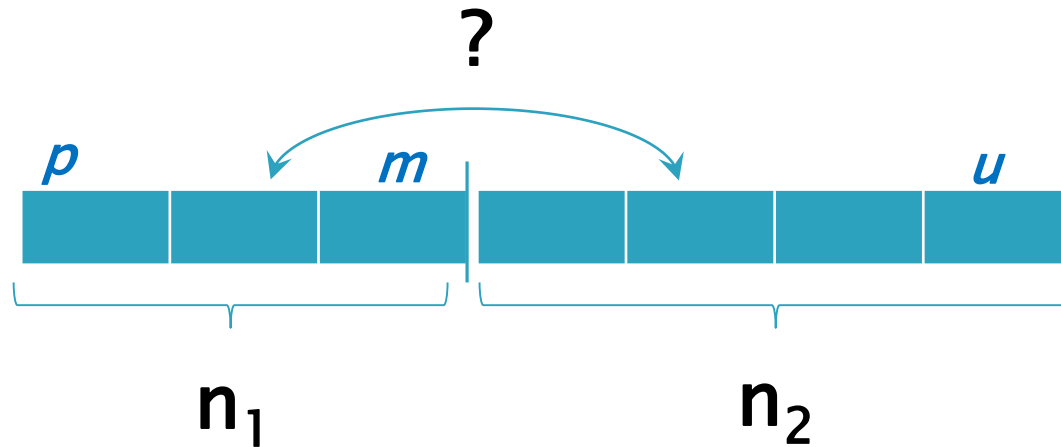
Numărul maxim de inversiuni pentru un vector cu  $n$  elemente?

# Numărare inversiuni

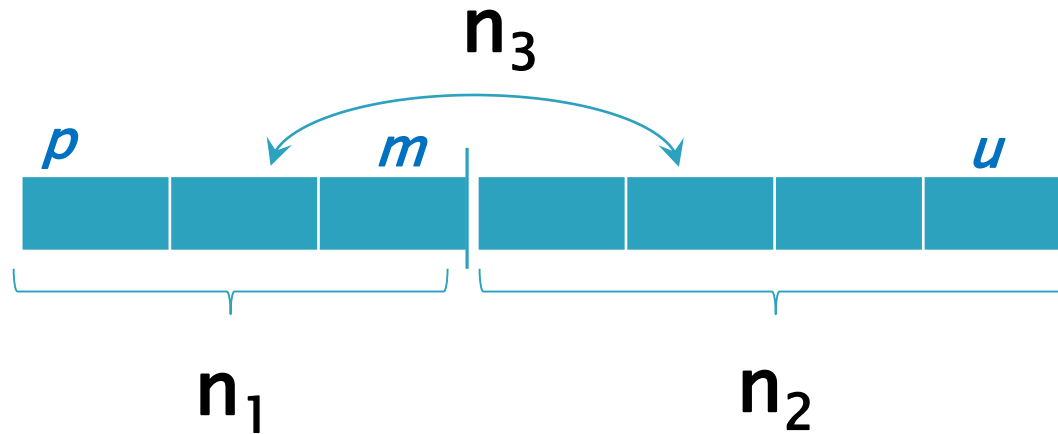
- ▶ Algoritm  $\Theta(n^2)$  – evident

# Numărare inversiuni

## ► Algoritm Divide et Impera



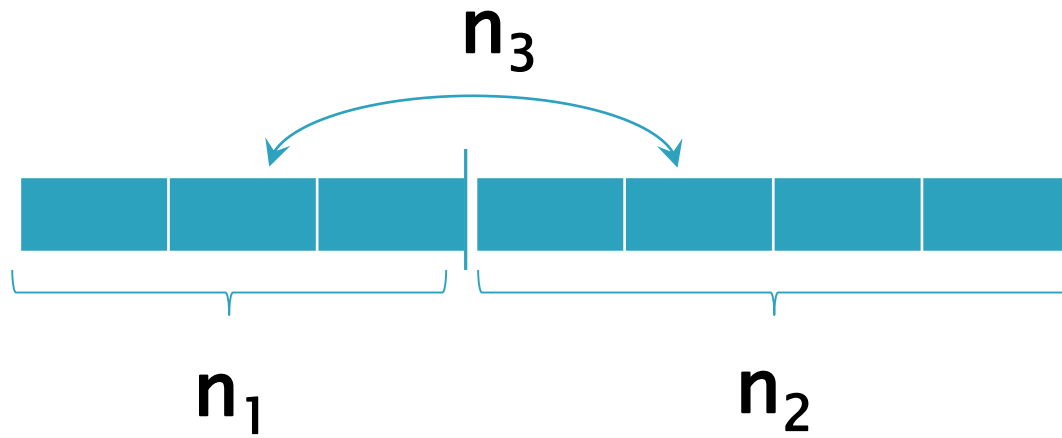
$$n_1 + n_2 + ?$$



$$n_1 + n_2 + n_3$$

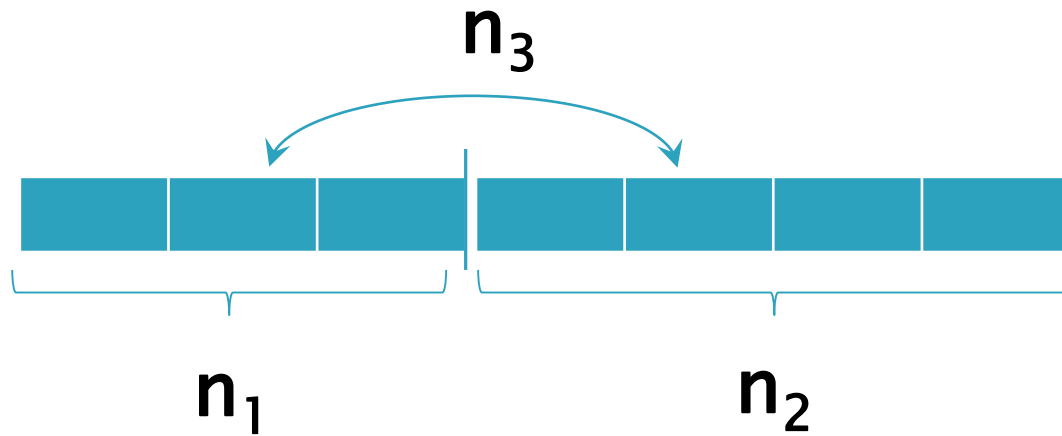


Cum calculăm eficient  $n_3$ ?



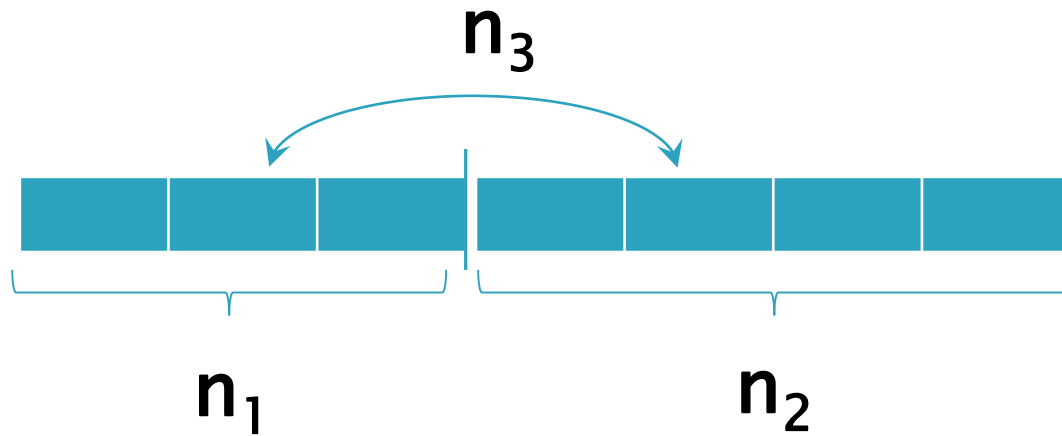
Cum calculăm eficient  $n_3$ ?





Cum calculăm eficient  $n_3$ ?

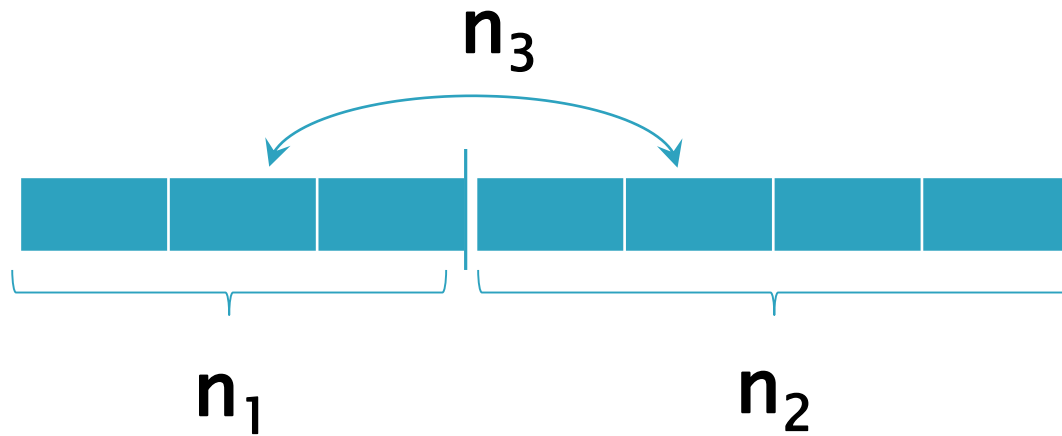
- **Încercare:** Considerăm fiecare pereche  $(i,j)$  cu  $i$  în subvectorul stâng și  $j$  în cel drept



Cum calculăm eficient  $n_3$ ?

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

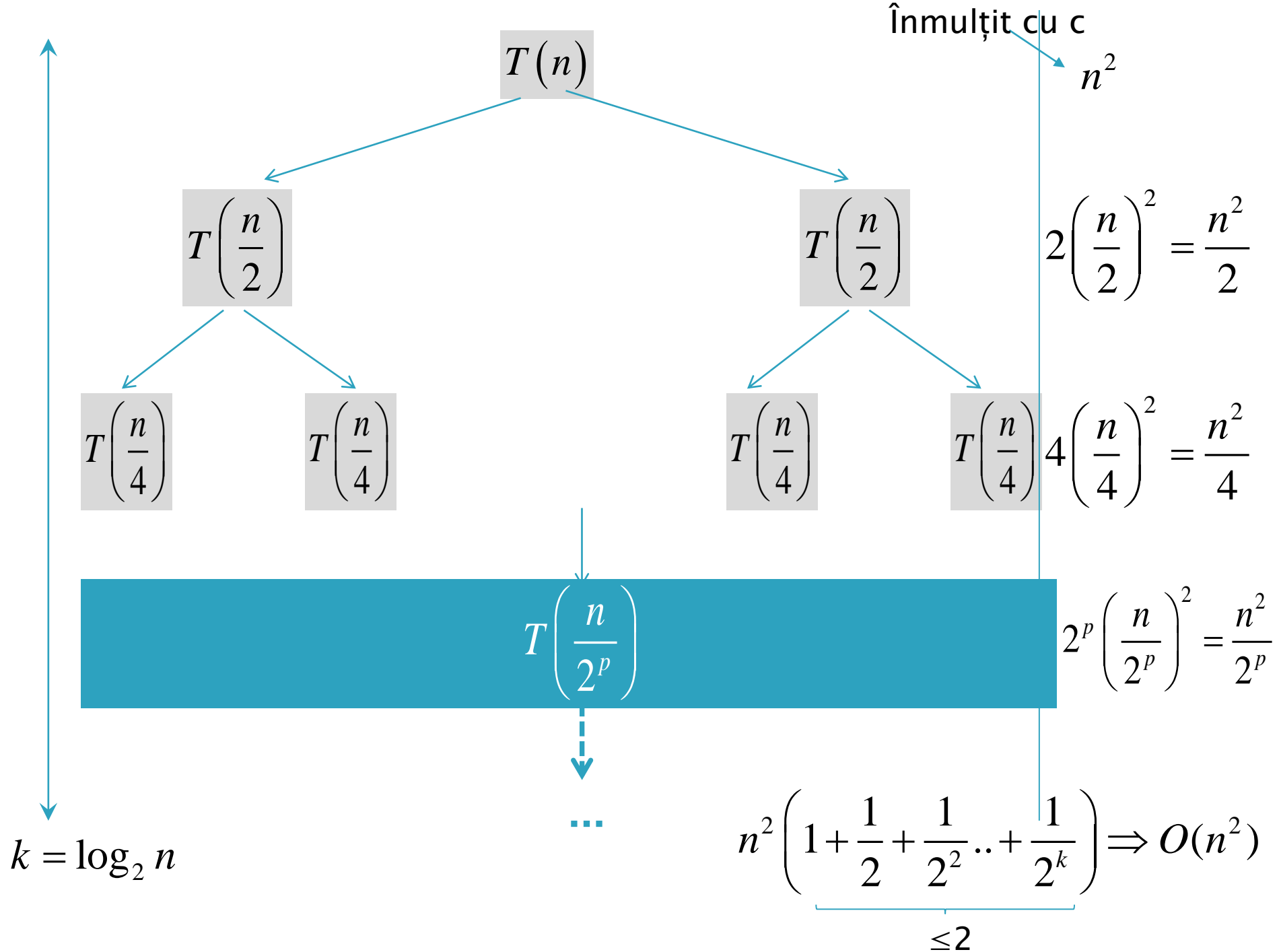
$$T(n) = 2T(n/2) + cn^2$$



Cum calculăm eficient  $n_3$ ?

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

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



# Complexitate

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

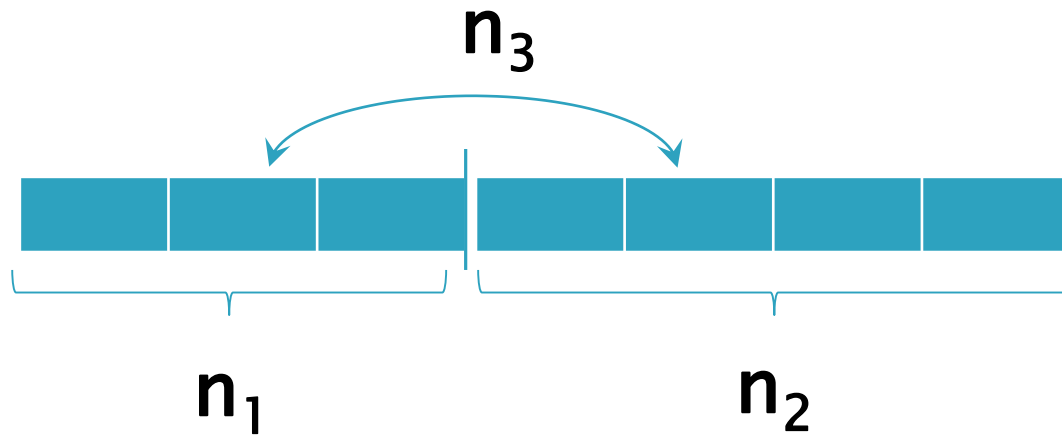
$$= 2[2T(n/2^2) + c \cdot (n/2)^2] + cn^2 =$$

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

$$= \dots =$$

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

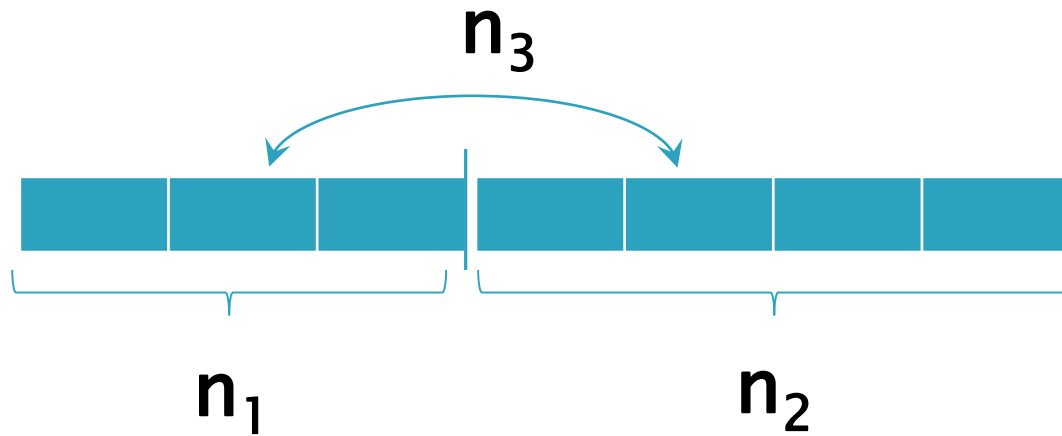
$n = 2^k \Rightarrow n^2 / 2^k \Rightarrow$  tot suma anterioară



Cum calculăm eficient  $n_3$ ?

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

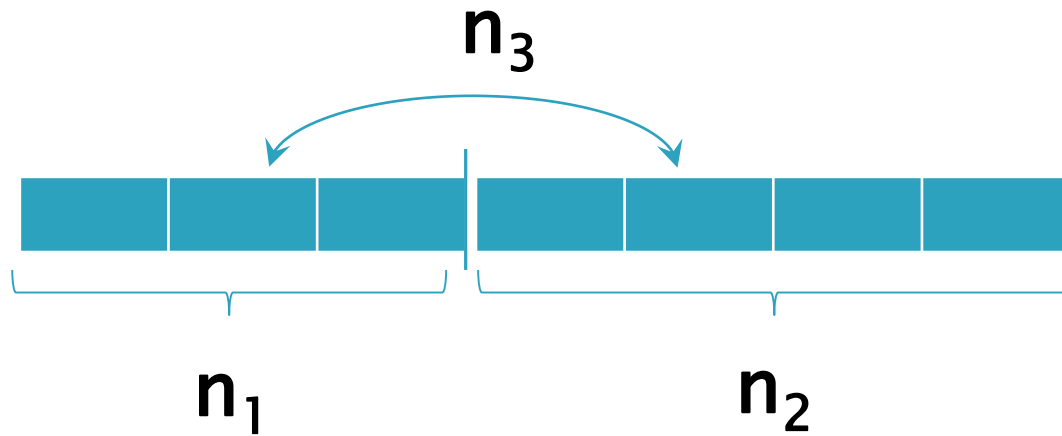
$$T(n) = 2T(n/2) + cn^2 \quad - \text{tot } O(n^2)$$



Cum calculăm eficient  $n_3$ ?



Dacă subvectorii stâng și drept sunt **sortați crescător**, numărarea inversiunilor  $(i,j)$  date de elemente din subvectori diferiți **se poate face la interclasare**



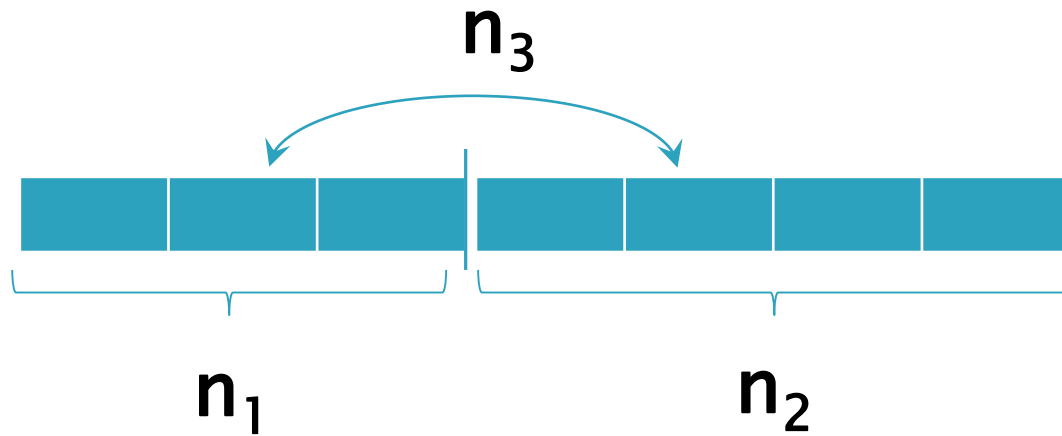
Cum calculăm eficient  $n_3$ ?



Dacă subvectorii stâng și drept sunt **sortați crescător**, numărarea inversiunilor  $(i,j)$  date de elemente din subvectori diferiți **se poate face la interclasare**

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



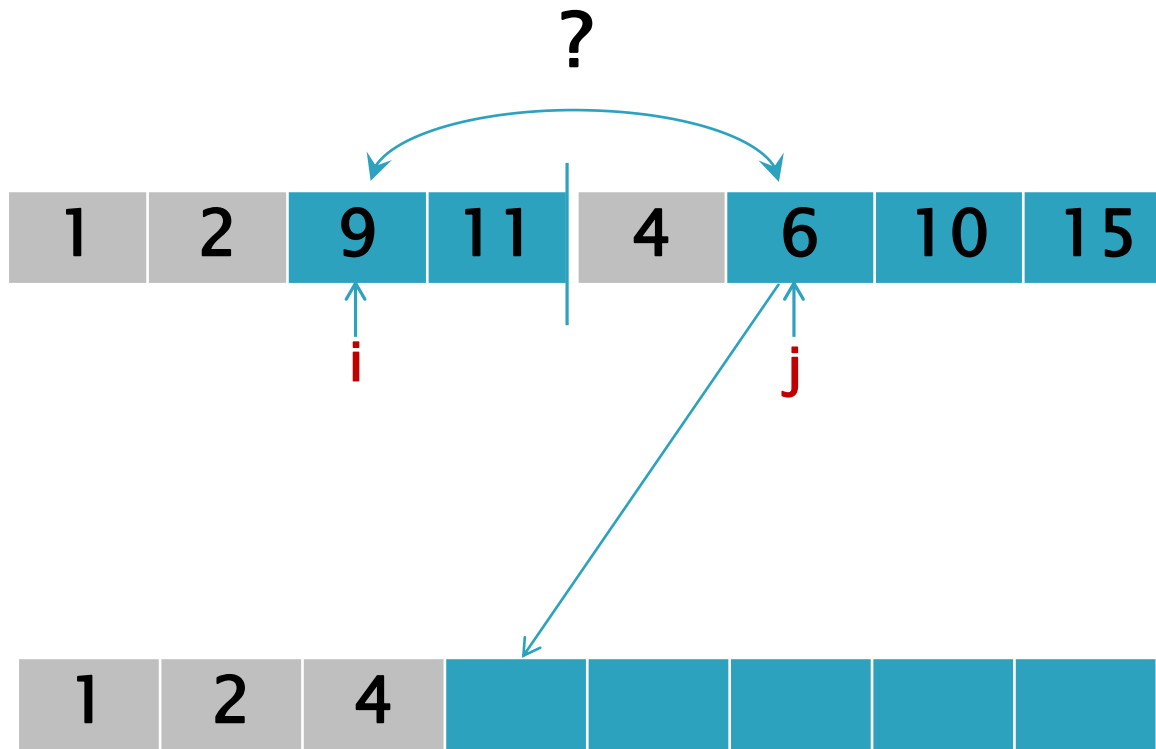


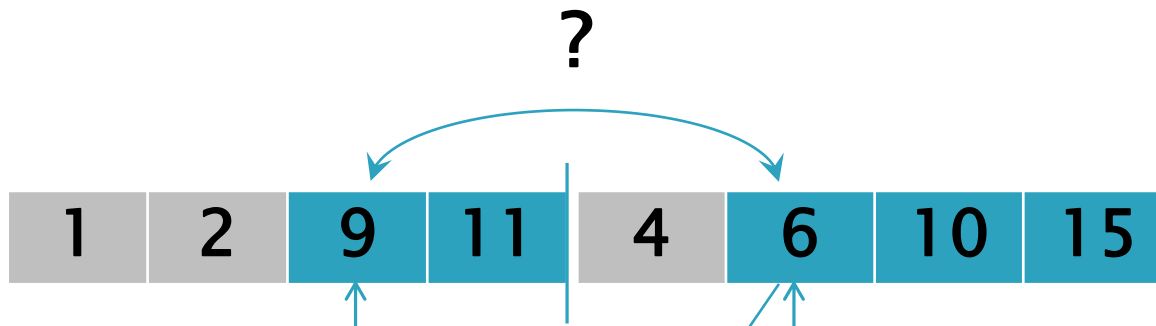
Cum calculăm eficient  $n_3$ ?



Dacă subvectorii stâng și drept sunt **sortați crescător**, numărarea inversiunilor  $(i,j)$  date de elemente din subvectori diferiți **se poate face la interclasare**

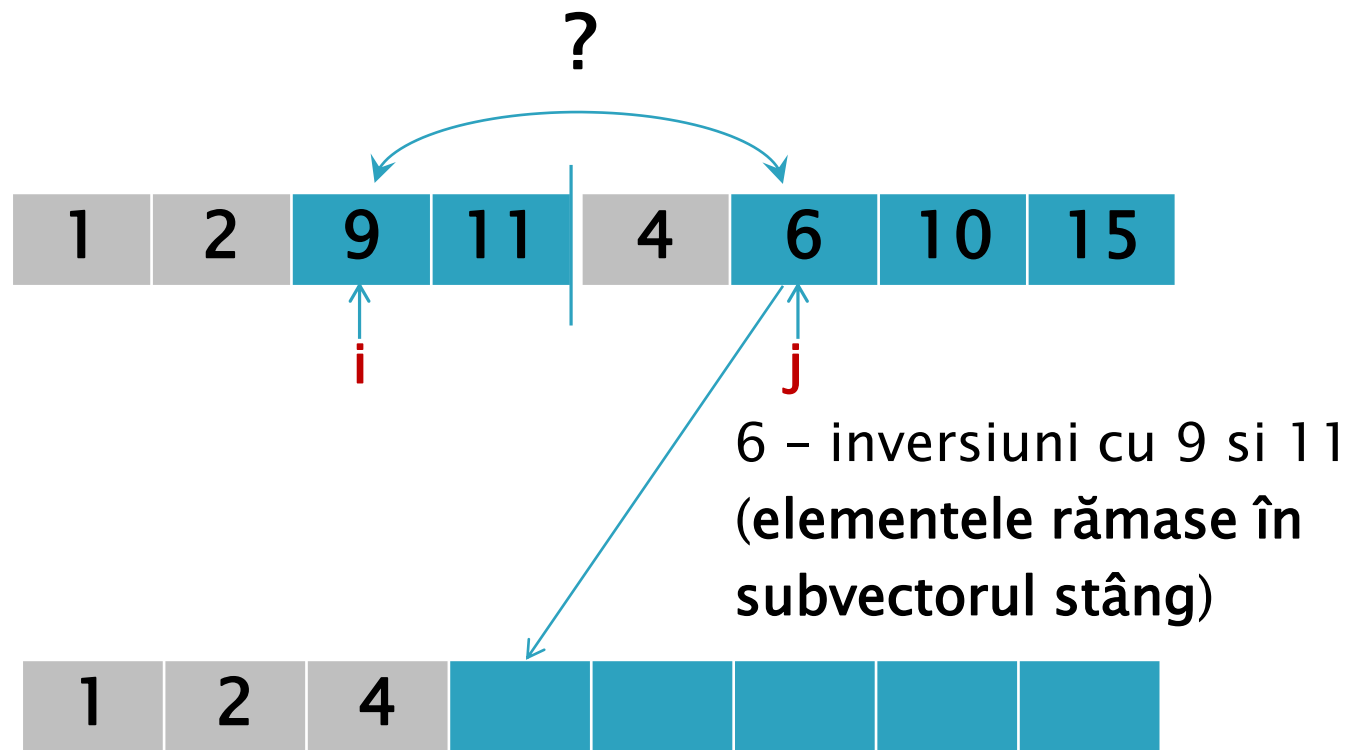
$$T(n) = 2T(n/2) + cn \Rightarrow T(n) = O(n \log n)$$





6 – inversiuni cu 9 si 11  
(**elementele rămase în  
subvectorul stâng**)

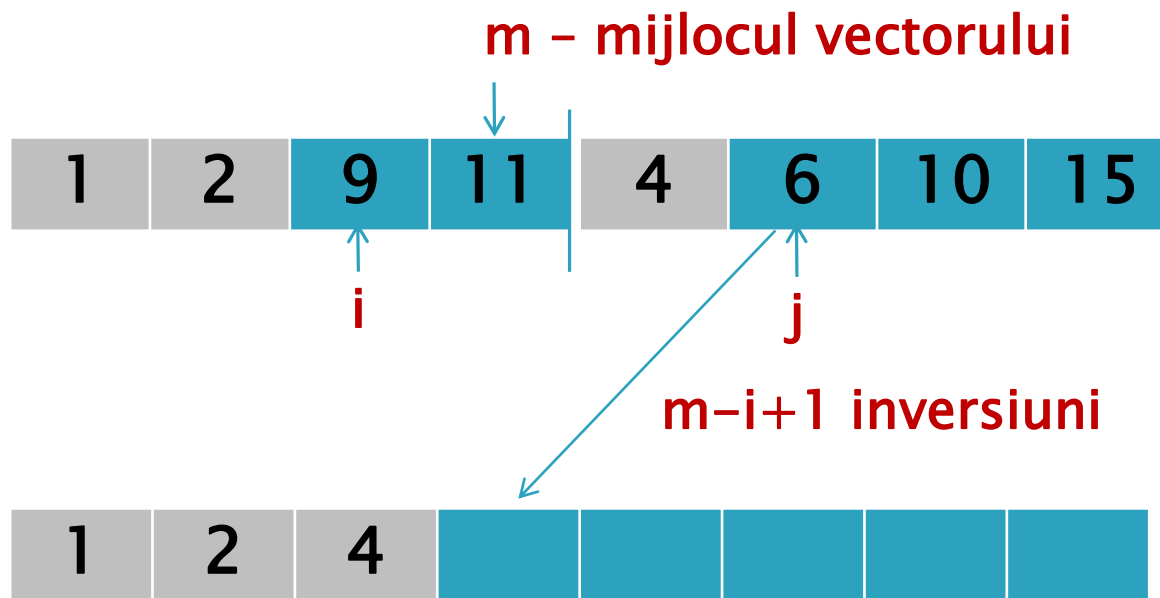




Când  $a[j]$  cu  $j > m$  este adăugat în vectorul rezultat, el este **mai mic (doar)** decât toate elementele din subvectorul stâng **neadăugate** încă în vectorul rezultat

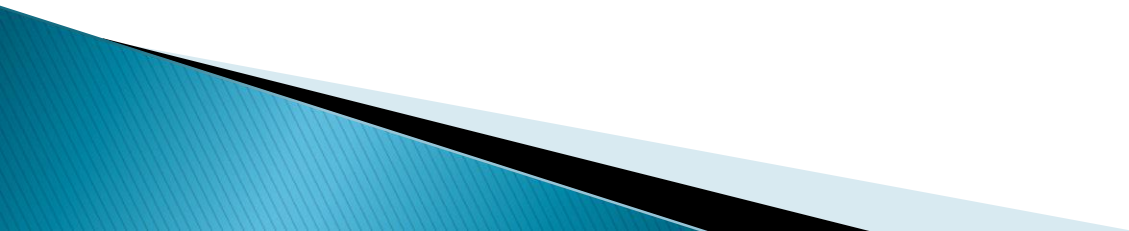


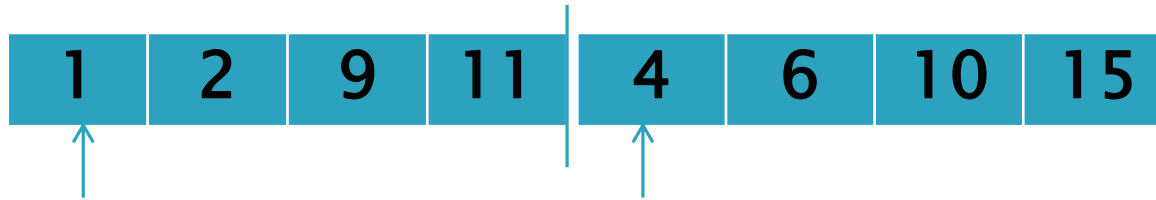
Câte inversiuni determină deci  $a[j]$  cu elementele din subvectorul stâng?

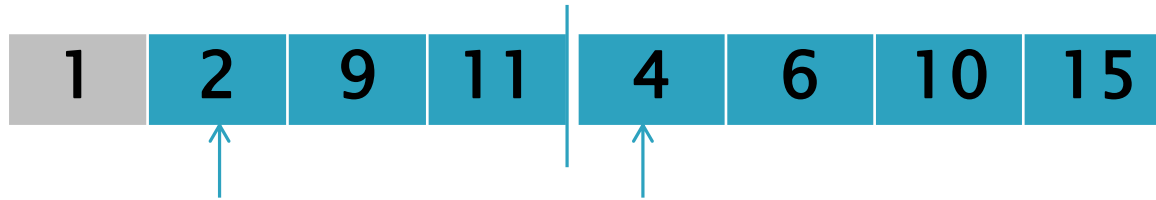


- $a[j]$  determină  $m - i + 1$  inversiuni

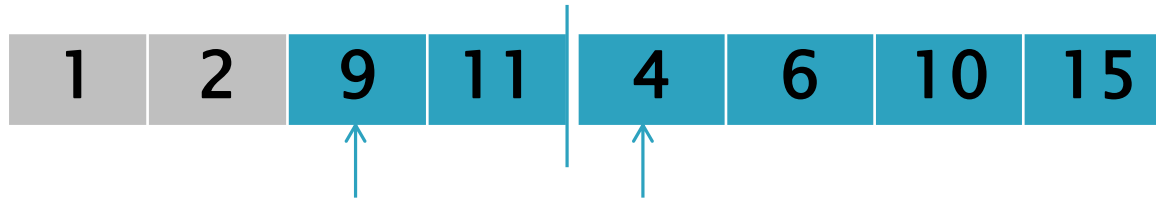
# **Exemplu – numărarea inversiunilor la interclasare**

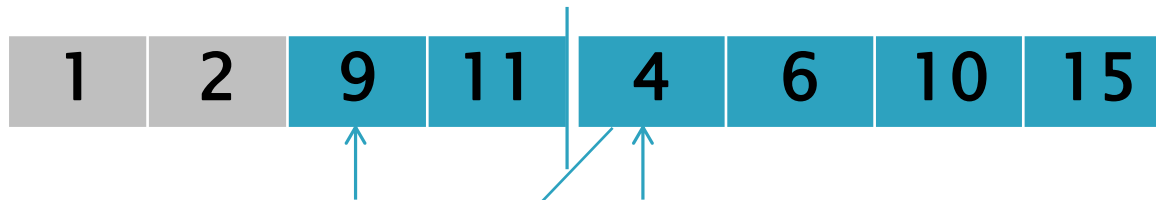












4 – inversiuni cu 9 si 11



Inversiuni = 2



Inversiuni = 2



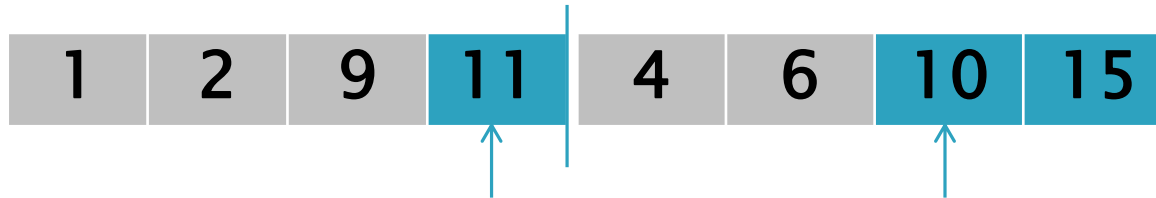
6- inversiuni cu 9 si 11



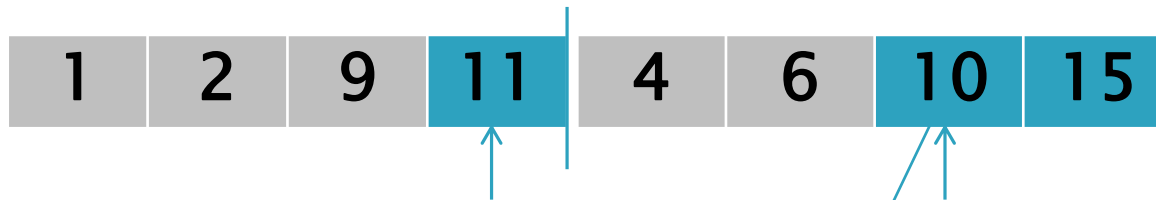
Inversiuni = 2 + 2



Inversiuni = 2 + 2



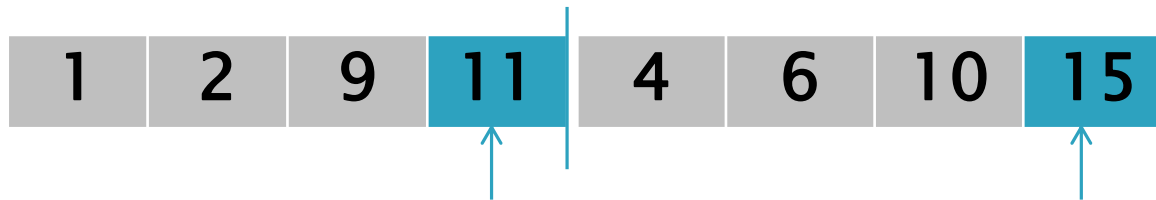
Inversiuni = 2 + 2



10- inversiuni cu 11

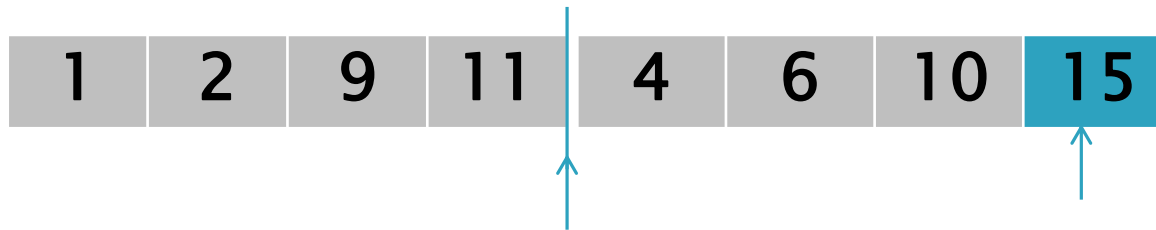


$$\text{Inversiuni} = 2 + 2 + 1$$



$$\text{Inversiuni} = 2 + 2 + 1$$





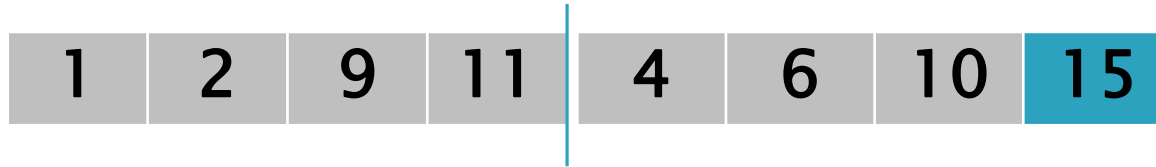
$$\text{Inversiuni} = 2 + 2 + 1$$



15- nicio inversiune



$$\text{Inversiuni} = 2 + 2 + 1 + 0$$



$$\text{Inversiuni} = 2 + 2 + 1 + 0 = 5$$

```
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)
```

► **Apel:**    **x** = nr\_inversiuni(v,0, len(v)-1)

```

def interclaseaza(a, p, m, u):
    b = [None]*(u-p+1)
    nr = 0
    i = p; j = m + 1; k = 0
    while (i<=m) and (j <= u):
        if a[i] <= a[j]:
            b[k] = a[i]; i += 1
        else:
            b[k] = a[j]; j += 1; nr += (m-i+1)
        k+=1

    while i<=m:
        b[k] = a[i]; k += 1; i += 1

    while j<=u:
        b[k] = a[j]; k += 1; j += 1

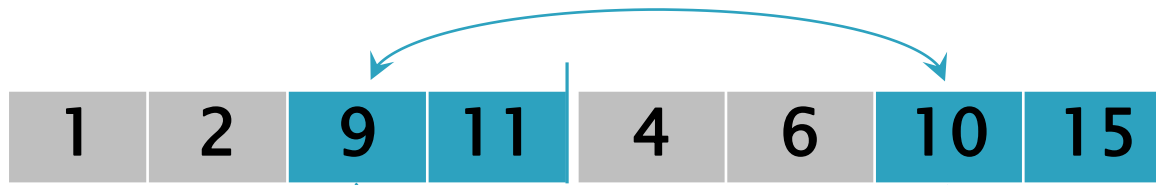
    for i in range(p,u+1):
        a[i] = b[i-p]

    return nr

```

- ▶ **Varianta 2** – puteam număra inversiunile dintre subvectori și atunci când un element  $a[i]$  cu  $i \leq m$  (din subvectorul stâng) este adăugat în vectorul rezultat.

?

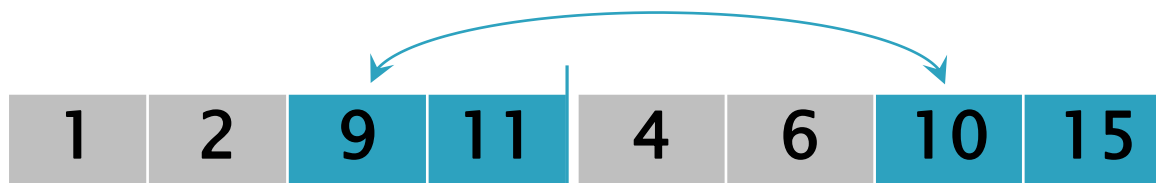


i

j



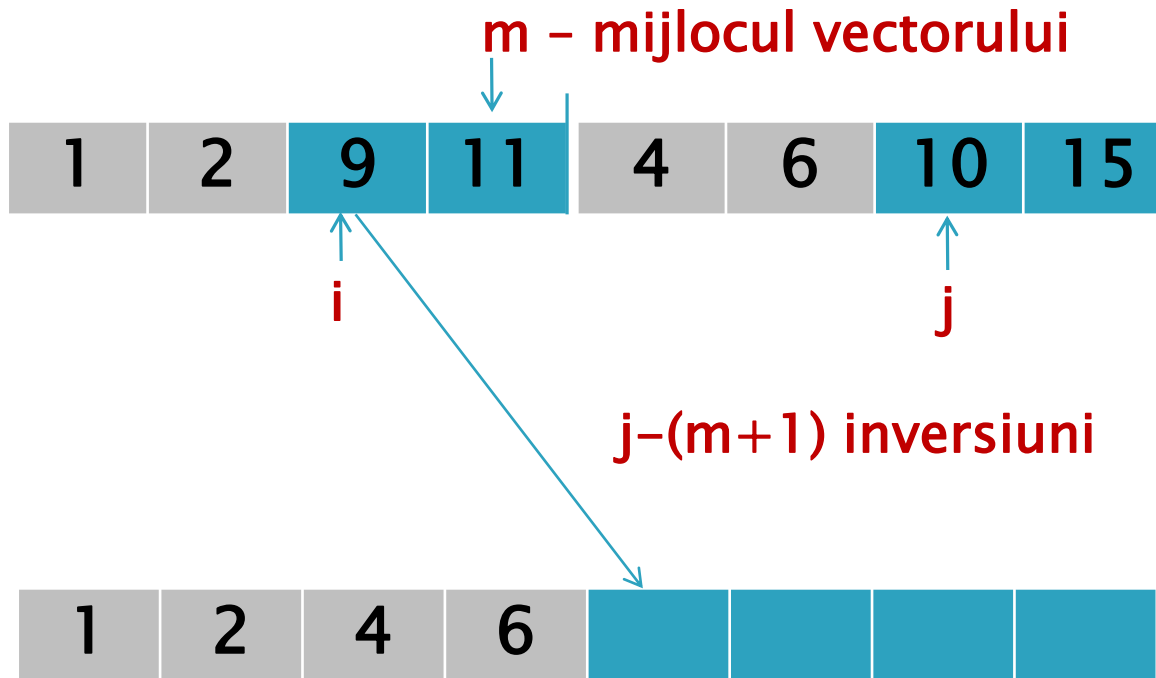
?



9 – inversiuni cu 4 si 6  
(elementele deja adăugate  
din subvectorul drept)







- $a[i]$  determină  $j - m - 1$  inversiuni

- ▶ **Temă** – Propuneți un algoritm similar pentru determinarea numărului de inversiuni ale unui vector oarecare (ale cărui elemente **nu sunt neapărat distincte**)

# Quicksort

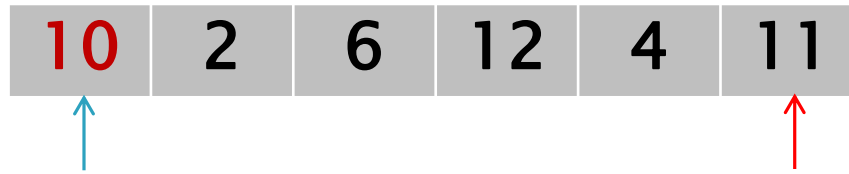
## Sortarea rapidă

# Quicksort

## ► Idee:

- poziționăm primul element al secvenței (**pivotul**) pe poziția sa finală = astfel încât elementele din stânga sa sunt mai mici, iar cele din dreapta mai mari
- ordonăm crescător elementele din stânga
- ordonăm crescător elementele din dreapta

## **Exemplu – poziționare pivot**



10	2	6	12	4	11
----	---	---	----	---	----



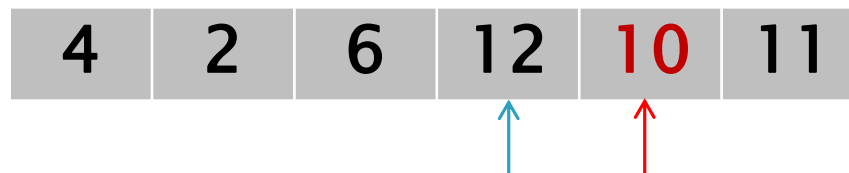
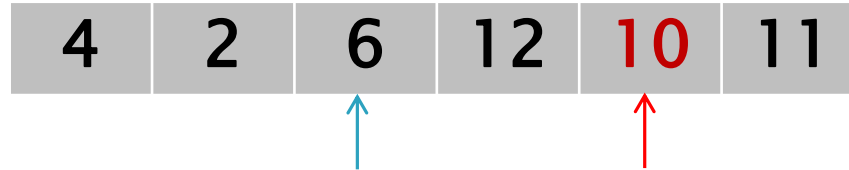
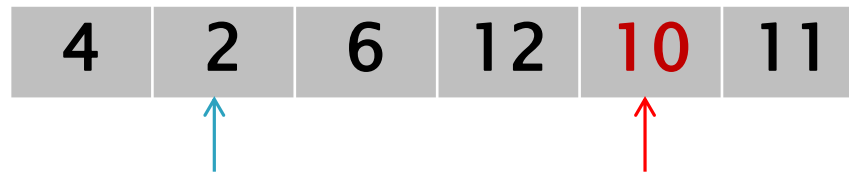
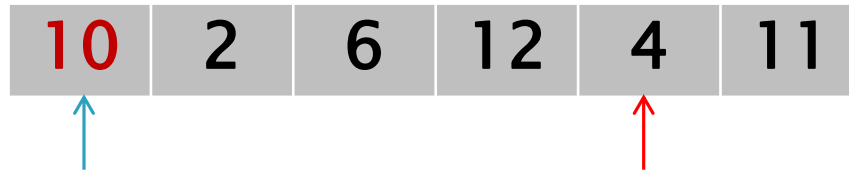
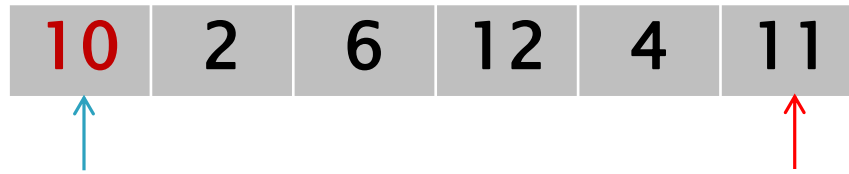
10	2	6	12	4	11
----	---	---	----	---	----

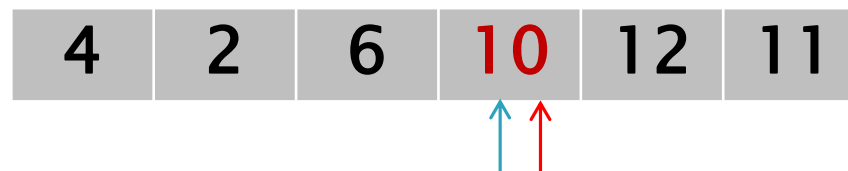
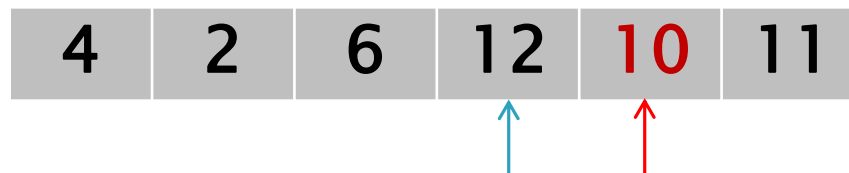
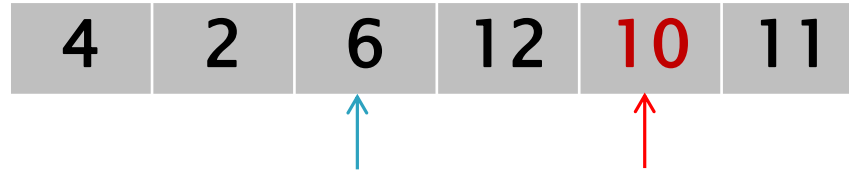
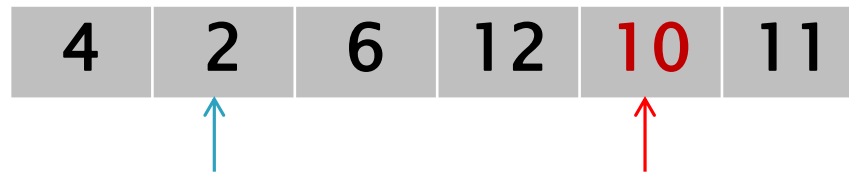
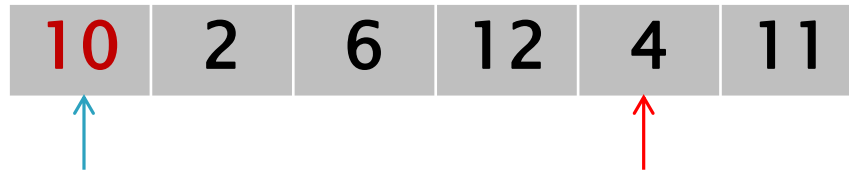
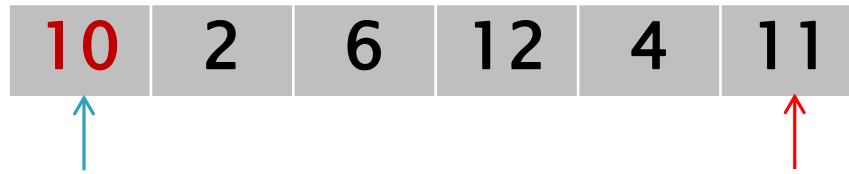












# Quicksort

```
def quick_sort_di(v, p, u):  
    if p >= u:  
        return  
    m = poz(v, p, u)  
    quick_sort_di(v, p, m - 1)  
    quick_sort_di(v, m + 1, u)
```

```
def poz(v, p, u):  
    i = p  
    j = u  
    depli = 0  
    deplj = -1  
    while i < j:  
        if v[i] > v[j]:  
            v[i], v[j] = v[j], v[i]  
            depli, deplj = -deplj, -depli  
            #aux= depli; depli= -deplj; deplj= -aux;  
        i += depli  
        j += deplj  
    return i
```

# Quicksort

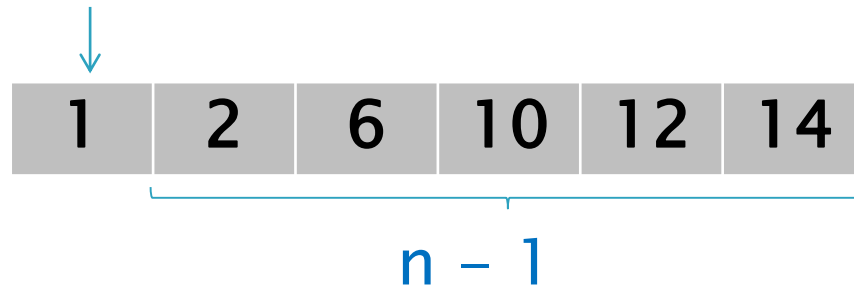
## ► Complexitate:

- Defavorabil:  $O(n^2)$

- pentru vector deja sortat  $\Rightarrow$
- una dintre subprobleme are dimensiune  $n-1$
- pivotarea  $n-1$

$$T(n) = T(n-1) + n-1$$

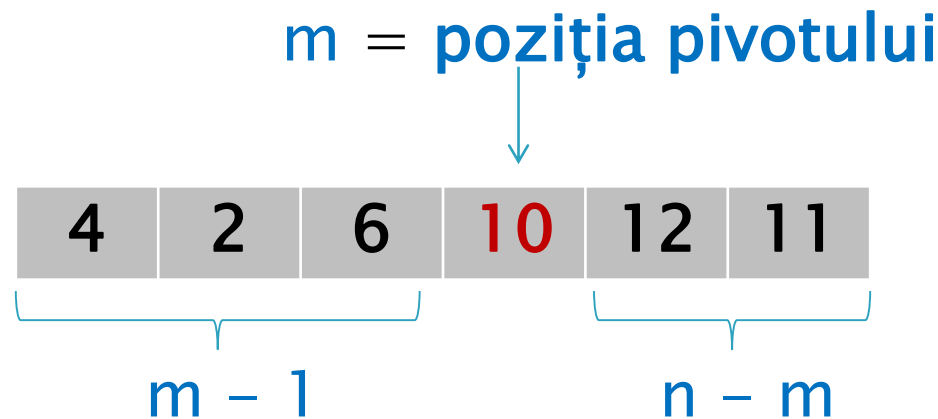
$m = 1 =$  poziția pivotului



# Quicksort

## ► Complexitate:

- **Mediu:**  $O(n \log n)$  – cu **pivot ales aleator**
- Alegem aleator un element ca pivot, îl interschimbăm cu primul element și folosim procedura de pivotare anterioară



# Quicksort – pivot aleator

```
def quick_sort_di(v, p, u):  
    if p >= u:  
        return  
  
    m = poz_rand(v, p, u)  
    quick_sort_di(v, p, m - 1)  
    quick_sort_di(v, m + 1, u)
```



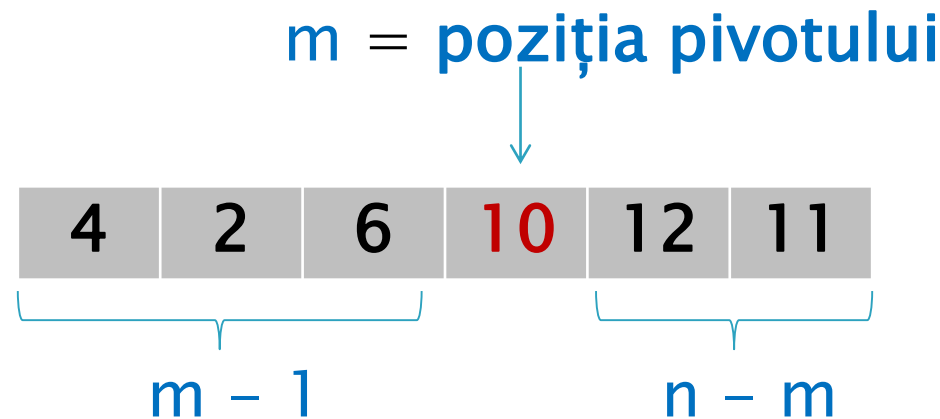
# Quicksort – pivot aleator

```
def poz_rand(v, p, u):  
    r = random.randint(p, u)  
    v[r], v[p] = v[p], v[r]  
    return poz(v, p, u)
```

# Quicksort

## ► Complexitate:

- Mediu:  $O(n \log n)$  – cu **pivot ales aleator**
- Justificarea complexității caz mediu –  
SUMPLIMENTAR



# Quicksort

Timp mediu



$$\begin{cases} T(n) = n - 1 + \frac{1}{n} \sum_{k=1}^n [T(k-1) + T(n-k)] \\ T(1) = T(0) = 0 \end{cases}$$

# Quicksort

$$\begin{cases} T(n) = n - 1 + \frac{1}{n} \sum_{k=1}^n [T(k-1) + T(n-k)] \\ T(1) = T(0) = 0 \end{cases}$$

$$nT(n) = n(n-1) + 2[T(0) + T(1) + \dots + T(n-1)]$$

# Quicksort

$$\begin{cases} T(n) = n - 1 + \frac{1}{n} \sum_{k=1}^n [T(k-1) + T(n-k)] \\ T(1) = T(0) = 0 \end{cases}$$

$$nT(n) = n(n-1) + 2[T(0) + T(1) + \dots + T(n-1)]$$

$$(n-1)T(n-1) = (n-1)(n-2) + 2[T(0) + \dots + T(n-2)]$$

# Quicksort

$$\begin{cases} T(n) = n - 1 + \frac{1}{n} \sum_{k=1}^n [T(k-1) + T(n-k)] \\ T(1) = T(0) = 0 \end{cases}$$

$$nT(n) = n(n-1) + 2[T(0) + T(1) + \dots + T(n-1)]$$

$$(n-1)T(n-1) = (n-1)(n-2) + 2[T(0) + \dots + T(n-2)]$$

**Scădem cele două relații:**

$$nT(n) - (n-1)T(n-1) = 2(n-1) + 2T(n-1)$$

# Quicksort

$$\begin{cases} T(n) = n - 1 + \frac{1}{n} \sum_{k=1}^n [T(k-1) + T(n-k)] \\ T(1) = T(0) = 0 \end{cases}$$

$$nT(n) = n(n-1) + 2[T(0) + T(1) + \dots + T(n-1)]$$

$$(n-1)T(n-1) = (n-1)(n-2) + 2[T(0) + \dots + T(n-2)]$$

$$nT(n) - (n-1)T(n-1) = 2(n-1) + 2T(n-1)$$

$$nT(n) = (n+1)T(n-1) + 2(n-1)$$

# Quicksort

$$\begin{cases} T(n) = n - 1 + \frac{1}{n} \sum_{k=1}^n [T(k-1) + T(n-k)] \\ T(1) = T(0) = 0 \end{cases}$$

$$nT(n) = n(n-1) + 2[T(0) + T(1) + \dots + T(n-1)]$$

$$(n-1)T(n-1) = (n-1)(n-2) + 2[T(0) + \dots + T(n-2)]$$

$$nT(n) - (n-1)T(n-1) = 2(n-1) + 2T(n-1)$$

$$nT(n) = (n+1)T(n-1) + 2(n-1) \quad | \quad :n(\mathbf{n+1})$$



# Quicksort

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

# Quicksort

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

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + 2\left(\frac{2}{n+1} - \frac{1}{n}\right)$$

# Quicksort

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

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + 2\left(\frac{2}{n+1} - \frac{1}{n}\right)$$

$$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + 2\left(\frac{2}{n} - \frac{1}{n-1}\right)$$

.....

$$\frac{T(2)}{3} = \frac{T(1)}{2} + 2\left(\frac{2}{3} - \frac{1}{2}\right)$$

# Quicksort

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

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + 2\left(\frac{2}{n+1} - \frac{1}{n}\right)$$

$$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + 2\left(\frac{2}{n} - \frac{1}{n-1}\right)$$

.....

$$\frac{T(2)}{3} = \frac{T(1)}{2} + 2\left(\frac{2}{3} - \frac{1}{2}\right)$$

+

---


$$\frac{T(n)}{n+1} = 2\left(\frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3}\right) + \frac{2}{n+1} - 1$$

# Quicksort

$$\begin{aligned}\frac{T(n)}{n+1} &= 2 \left( \frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right) + \frac{2}{n+1} - 1 \\ &\leq 2 \left( \frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right)\end{aligned}$$

sume Riemann inferioare pentru funcția  $f(x)=1/x$

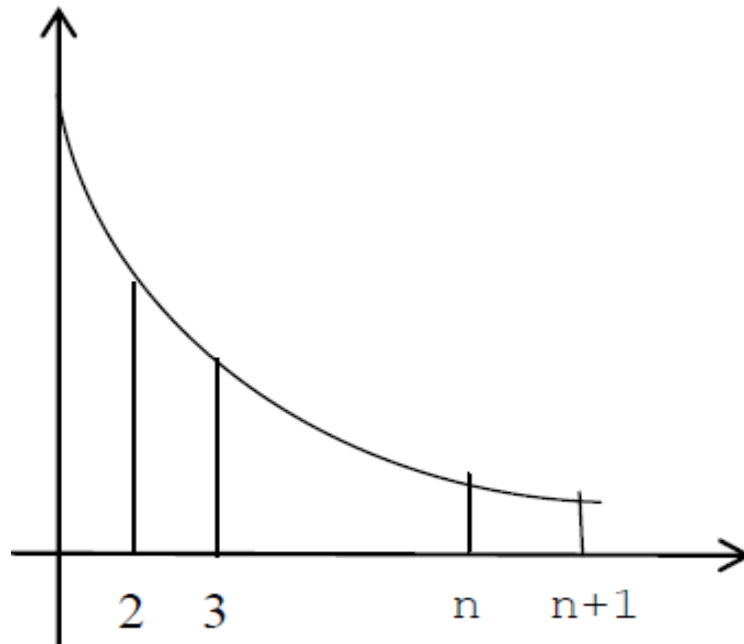
$$\Delta = \{2, 3, \dots, n+1\}$$

# Quicksort

$$\begin{aligned}\frac{T(n)}{n+1} &= 2 \left( \frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right) + \frac{2}{n+1} - 1 \\ &\leq 2 \left( \frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right)\end{aligned}$$

sume Riemann inferioare pentru funcția  $f(x) = 1/x$

$$\Delta = \{2, 3, \dots, n+1\}$$



# Quicksort

$$\begin{aligned}\frac{T(n)}{n+1} &= 2 \left( \frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right) + \frac{2}{n+1} - 1 \\ &\leq 2 \left( \frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right)\end{aligned}$$

sume Riemann inferioare pentru funcția  $f(x)=1/x$

$$\Delta = \{2, 3, \dots, n+1\}$$

$$\frac{T(n)}{n+1} \leq 2 \int_2^{n+1} \frac{1}{x} dx = 2 \ln x \Big|_2^{n+1} \leq 2 \ln(n+1)$$

# Statistici de ordine



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



# Statistici de ordine

A  $i$ -a statistică de ordine a unei mulțimi = al  $i$ -lea cel mai mic element.

- ▶ **Minimul** = prima statistică de ordine
- ▶ **Maximul** = a  $n$ -a statistică de ordine

# Statistici de ordine

- ▶ **Mediana** = punctul de la jumătatea unei mulțimi  
= o valoare  $v$  a.î. numărul de elemente din mulțime mai mici decât  $v$  este egal cu numărul de elemente din mulțime mai mari decât  $v$ .

# Statistici de ordine

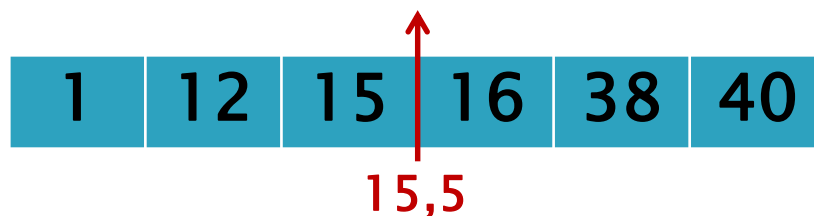
## ► Mediana

Dacă  $n$  este impar, atunci mediana este

a  $\lceil n/2 \rceil$ -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



## ► Mediană inferioară / superioară

# Statistici de ordine – utilitate

- ▶ **Statistică**
- ▶ **Mediana** – pentru o mulțime  $A=\{a_1, \dots, a_n\}$  valoarea  $\mu$  care minimizează expresia

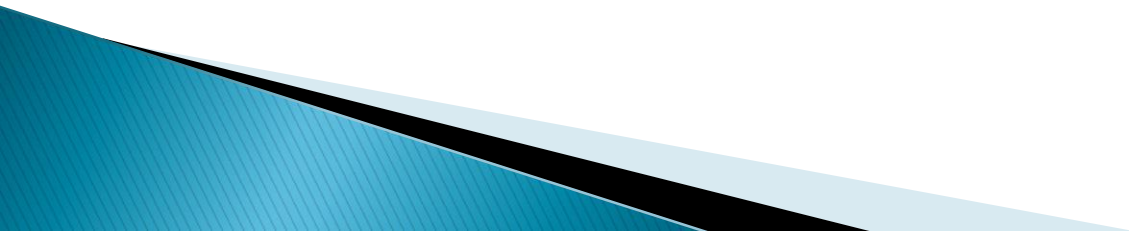
$$\sum_{i=1}^n |\mu - a_i|$$

este mediana (inf/sup)

# Statisticile de ordine

## Idee

Al  $k$ -lea minim



# Statistici de ordine

## Idee

Al k-lea minim – folosim poziționarea de la quicksort (**pivot aleator**)

# Statistici de ordine

Fie  $m$  poziția pivotului

- Dacă  $m = k$ , pivotul este al  $k$ -lea minim
- 
-

# Statistici de ordine

Fie  $m$  poziția pivotului

- Dacă  $m = k$ , pivotul este al  $k$ -lea minim
- Dacă  $m > k$ , al  $k$ -lea minim este în **stânga** pivotului (al  $k$ -lea minim din stanga)
-



# Statistici de ordine

Fie  $m$  poziția pivotului

- Dacă  $m = k$ , pivotul este al  $k$ -lea minim
- Dacă  $m > k$ , al  $k$ -lea minim este în **stânga** pivotului (al  $k$ -lea minim din stanga)
- Dacă  $m < k$ , al  $k$ -lea minim este în **dreapta** pivotului (al  $(k-m)$ -lea minim din dreapta)

$k = 2$

10	2	6	12	4	11
----	---	---	----	---	----

poziționare pivot

4	2	6	10	12	11
---	---	---	----	----	----

$m = 4 > k \Rightarrow$  stânga

$k = 2$

10	2	6	12	4	11
----	---	---	----	---	----

poziționare pivot

4	2	6	10	12	11
---	---	---	----	----	----

$m = 4 > k \longrightarrow$  stânga

4	2	6
---	---	---

poziționare pivot

2	4	6
---	---	---

$m = 2 = k \longrightarrow$  stop;  
pivotul este al  $k$ -lea minim

**#pentru numerotare de la 0**


```
def sel_k_min(v, k, p, u):  
    m = poz_rand(v, p, u)  
    if m == k - 1:  
        return v[m]  
    if m < k - 1:  
        return sel_k_min(v, k, m + 1, u)  
    return sel_k_min(v, k, p, m - 1)
```

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

# Complexitate

- Timpul mediu

$m = \text{poziția pivotului}$


$$\left\{ \begin{array}{l} T(n) \leq n - 1 + \frac{1}{n} \sum_{m=1}^n T(\max\{m-1, n-m\}) \\ \leq n - 1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} T(m) \end{array} \right.$$

$$T(n) \leq cn, c \geq 4$$

– se demonstrează prin inducție

# Complexitate

- Detalii pas inducție

$$T(n) \leq n - 1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} T(m) \leq n - 1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} (cm)$$

# Complexitate

- Detalii pas inducție

$$\begin{aligned} T(n) &\leq n-1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} T(m) \leq n-1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} (cm) \\ &\leq n-1 + \frac{2}{n} c \left[ \frac{1}{2} n(n-1) - \frac{1}{2} \frac{n}{2} \left( \frac{n}{2} - 1 \right) \right] = \\ &= \end{aligned}$$

# Complexitate

- Detalii pas inducție

$$\begin{aligned}T(n) &\leq n-1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} T(m) \leq n-1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} (cm) \\&\leq n-1 + \frac{2}{n} c \left[ \frac{1}{2} n(n-1) - \frac{1}{2} \frac{n}{2} \left( \frac{n}{2} - 1 \right) \right] = \\&= n-1 + \frac{2}{n} c \frac{n}{2} \left( n-1 - \frac{n}{4} + \frac{1}{2} \right) = n-1 + c \left( \frac{3}{4} n - \frac{1}{2} \right)\end{aligned}$$

Dar  $c \geq 4 \Rightarrow$



# Complexitate

- Detalii pas inducție

$$\begin{aligned}T(n) &\leq n-1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} T(m) \leq n-1 + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} (cm) \\&\leq n-1 + \frac{2}{n} c \left[ \frac{1}{2} n(n-1) - \frac{1}{2} \frac{n}{2} \left( \frac{n}{2} - 1 \right) \right] = \\&= n-1 + \frac{2}{n} c \frac{n}{2} \left( n-1 - \frac{n}{4} + \frac{1}{2} \right) = n-1 + c \left( \frac{3}{4} n - \frac{1}{2} \right)\end{aligned}$$

Dar  $c \geq 4 \Rightarrow$

$$T(n) \leq c \left( \frac{n-1}{4} + \frac{3}{4} n - \frac{1}{2} \right) = c \left( n - \frac{1}{4} - \frac{1}{2} \right) \leq cn$$

# Statistici de ordine

Algoritm  $O(n)$  caz defavorabil (suplimentar)



# Statistici de ordine

Algoritm  $O(n)$  caz defavorabil

## Idee

În algoritmul anterior `sel_k_min` se folosește în loc de pivot aleatoriu un pivot calculat astfel:

# Statistici de ordine

## Algoritm $O(n)$ caz defavorabil

### Idee

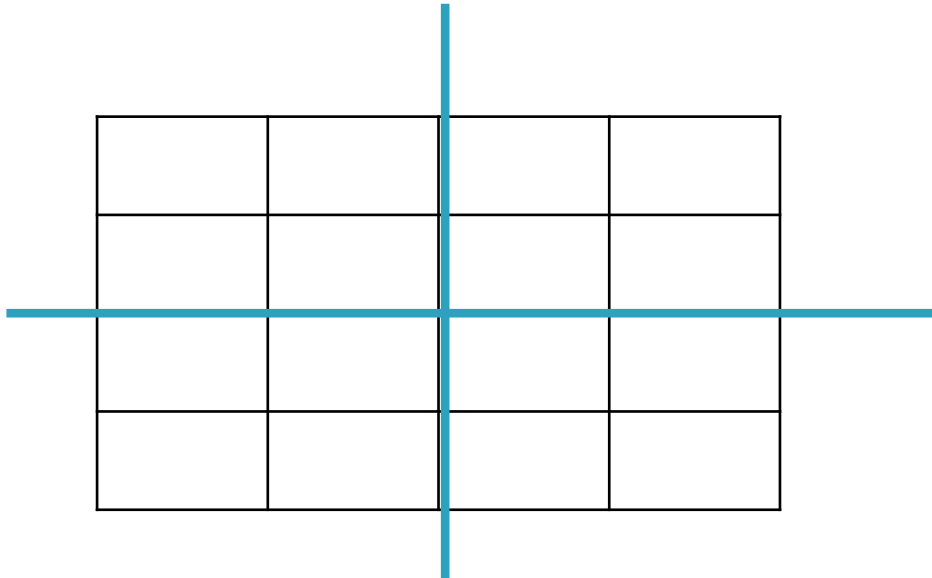
În algoritmul anterior `sel_k_min` se folosește în loc de pivot aleatoriu un pivot calculat astfel:

- \_ se împarte vectorul în grupe de 5 (cu cel mult o excepție – ultimul grup)
- se formează un vector `mediane[]` având ca elemente mediana fiecărui grup
- se calculează mediana acestui vector folosind aceeași funcție `sel_k_min mediane,  $\left\lceil \frac{n}{5} \right\rceil, \left\lceil \frac{n}{10} \right\rceil$`   $\Rightarrow$   
 $\Rightarrow$  **aceasta este pivotul**

# Divide et impera – Matrice



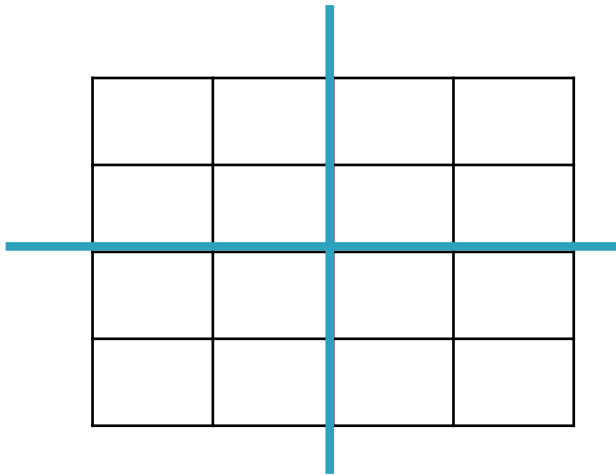
Să se calculeze suma elementelor unei matrice pătratice de dimensiune  $n$  unde  $n$  este putere a lui 2 folosind Divide et Impera (împărțind matricea succesiv în 4 subcadrame)



# 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 subcadrame)



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

```
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))
```

