Metoda Divide et Impera

desparte şi stăpâneşte

Metoda Divide et Impera

- Constă în
  - **împărţirea repetată** a unei probleme de dimensiuni mari în mai multe subprobleme **de acelaşi tip**

Metoda Divide et Impera

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

Metoda Divide et Impera

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

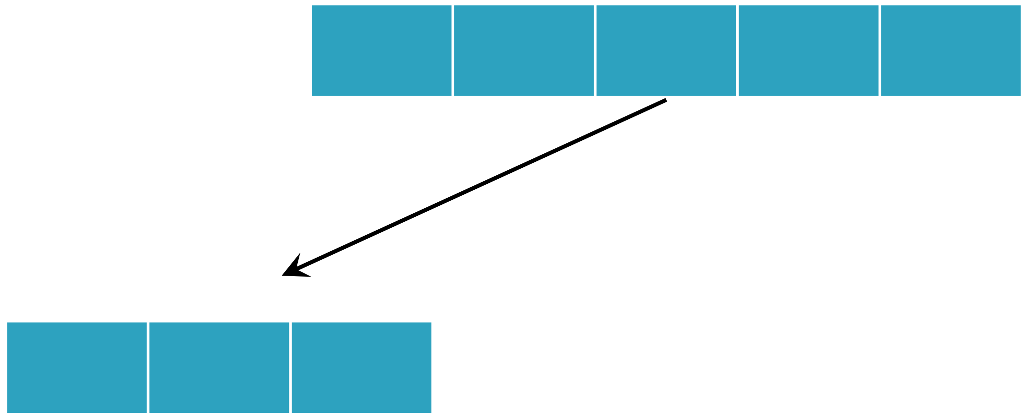
Schema generală

```
function DivImp(p,u)              – pentru a[p..u]
    if u-p<ε
        r ← RezolvaDirect(p,u)
    else
        m ← Pozitie(p,u); – de obicei mijlocul
        r1 ← DivImp(p,m);
        r2 ← DivImp(m+1,u);
        r ← Combina(r1,r2)
    return r
end
```

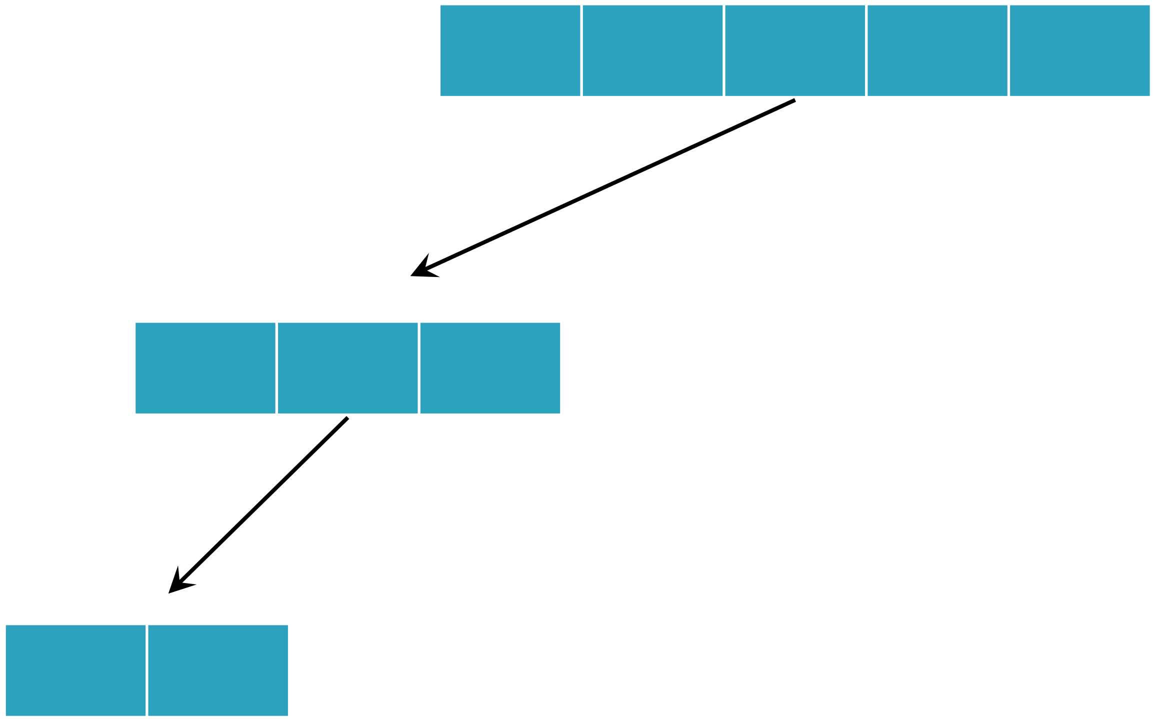
- **Apel:** `DivImp(1,n)`

Metoda Divide et Impera

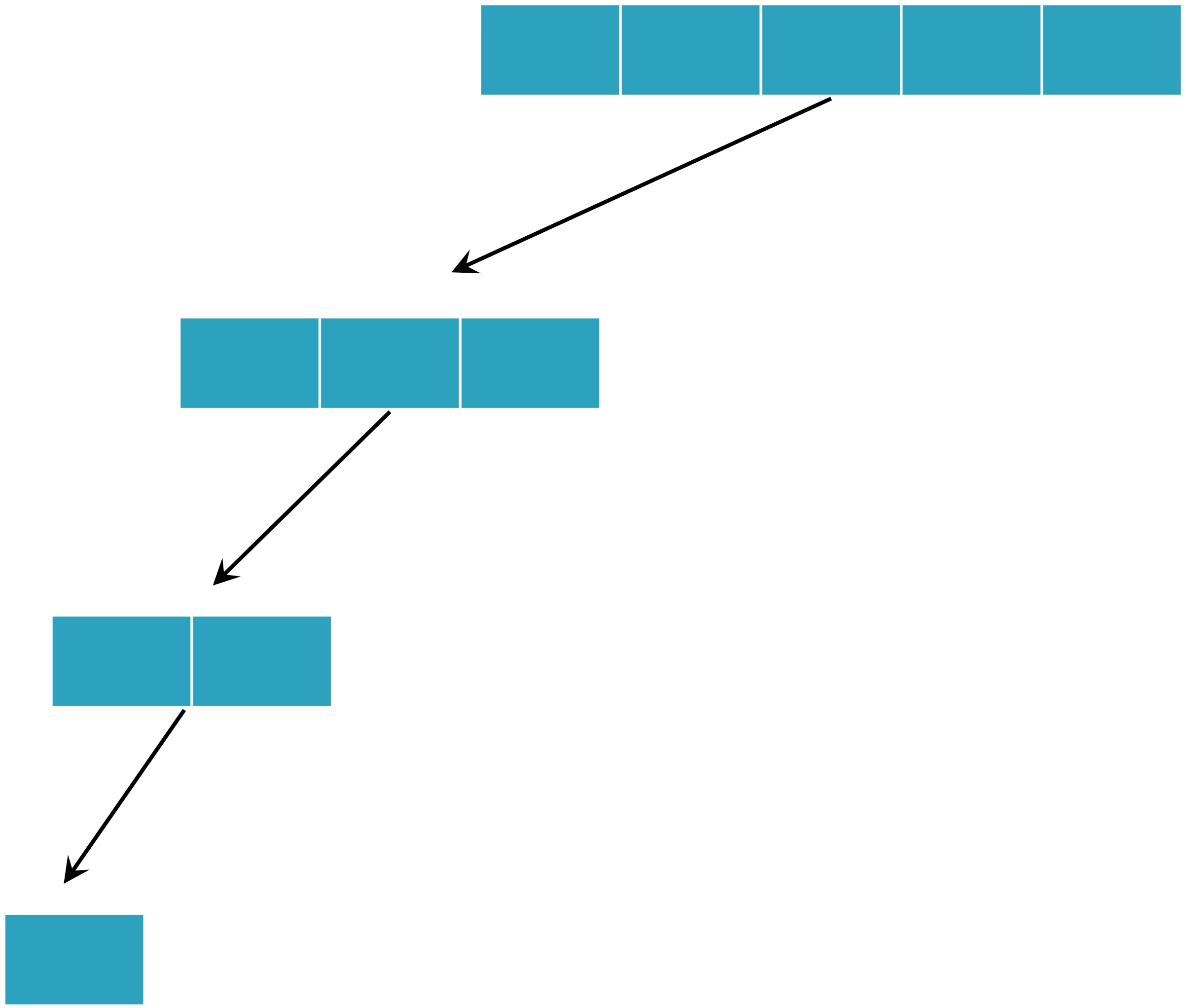
Metoda Divide et Impera



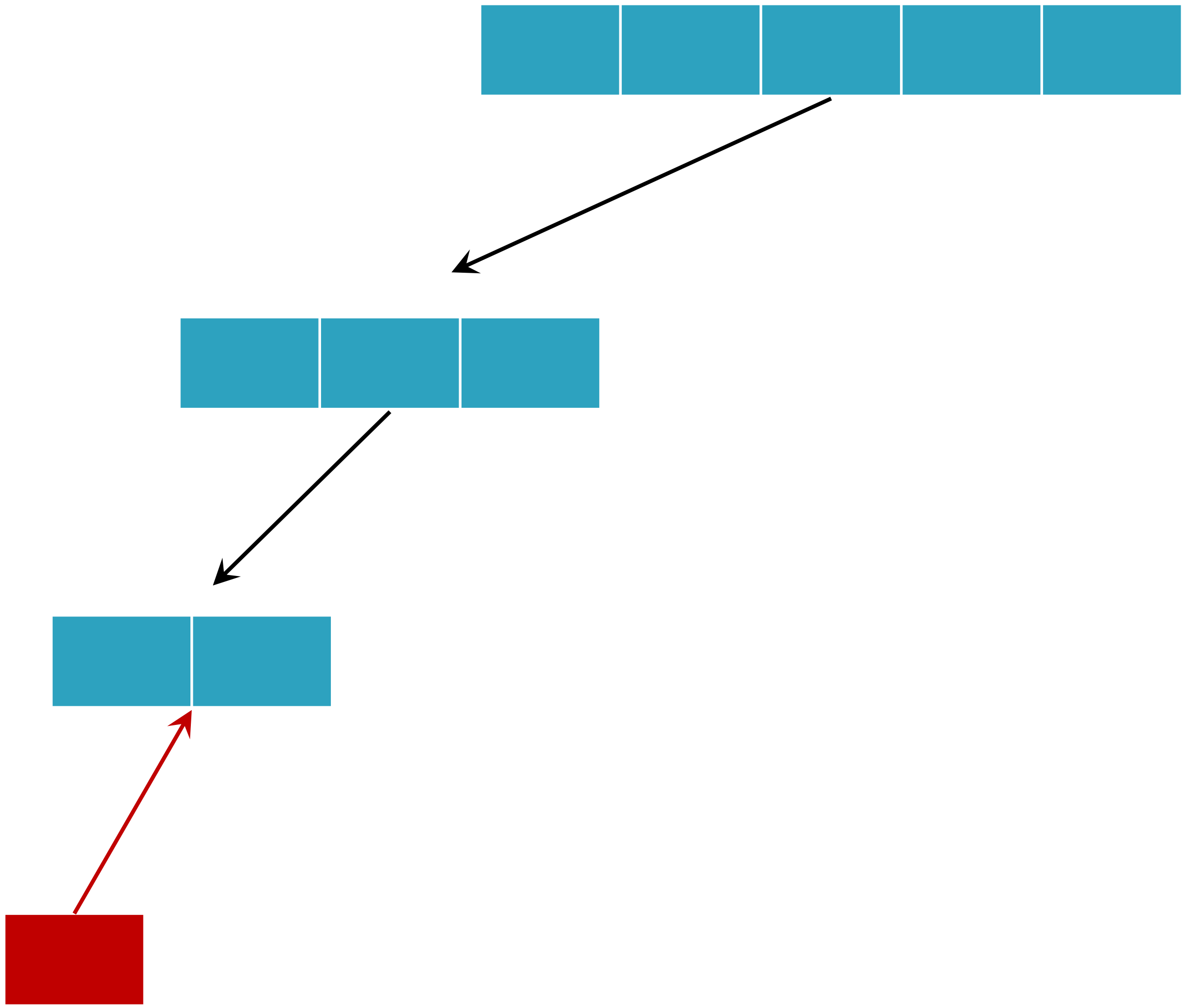
Metoda Divide et Impera



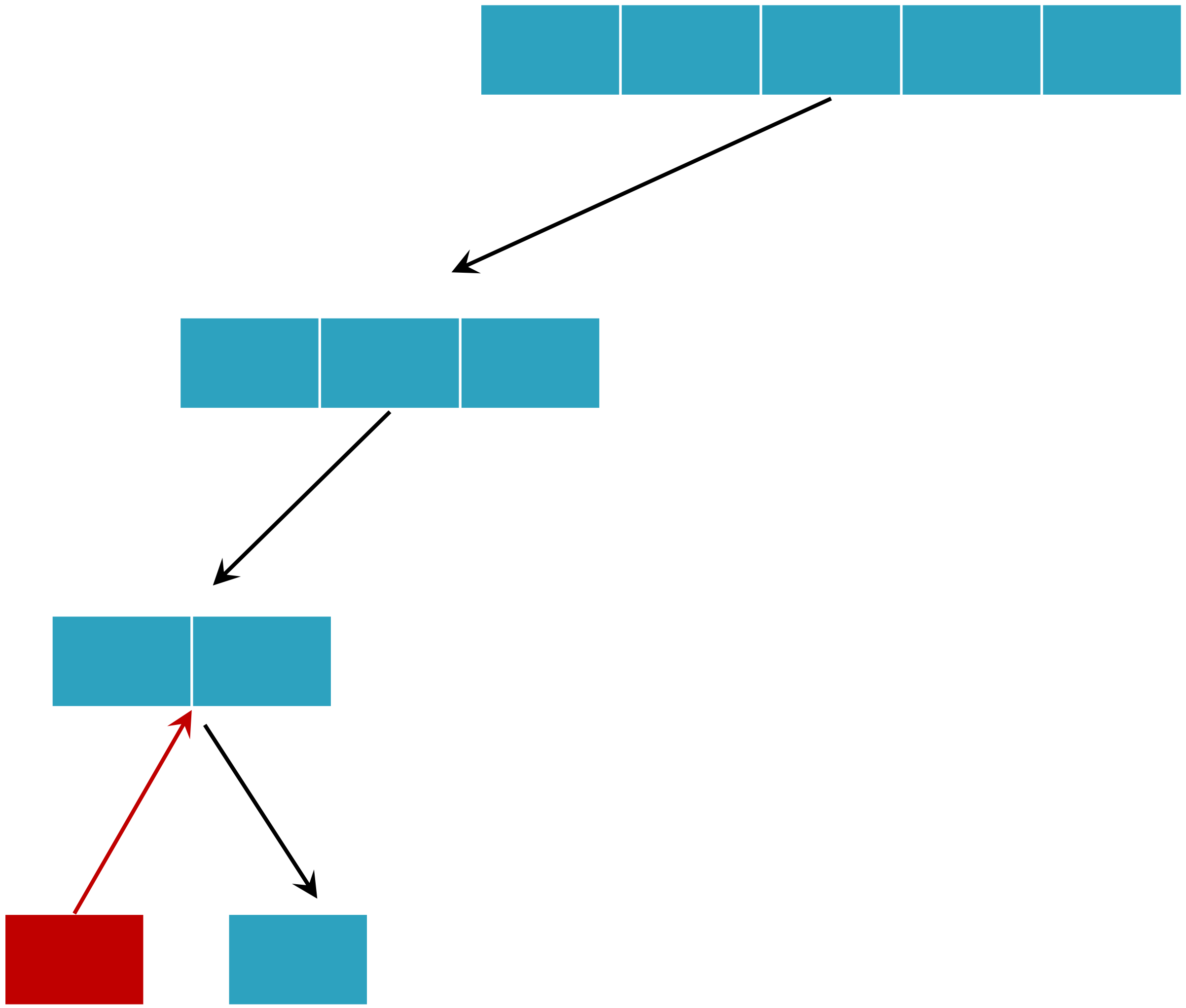
Metoda Divide et Impera



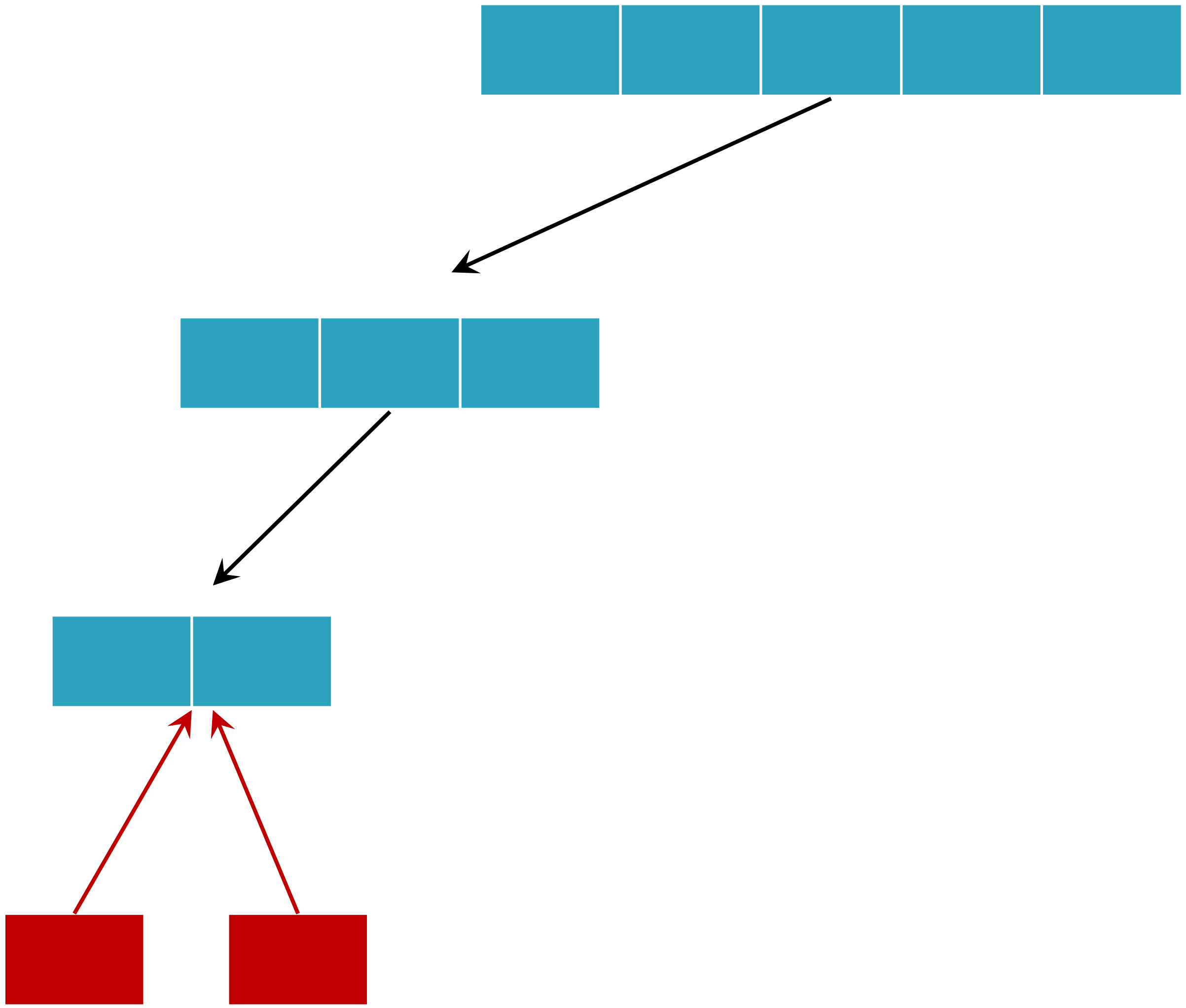
Metoda Divide et Impera



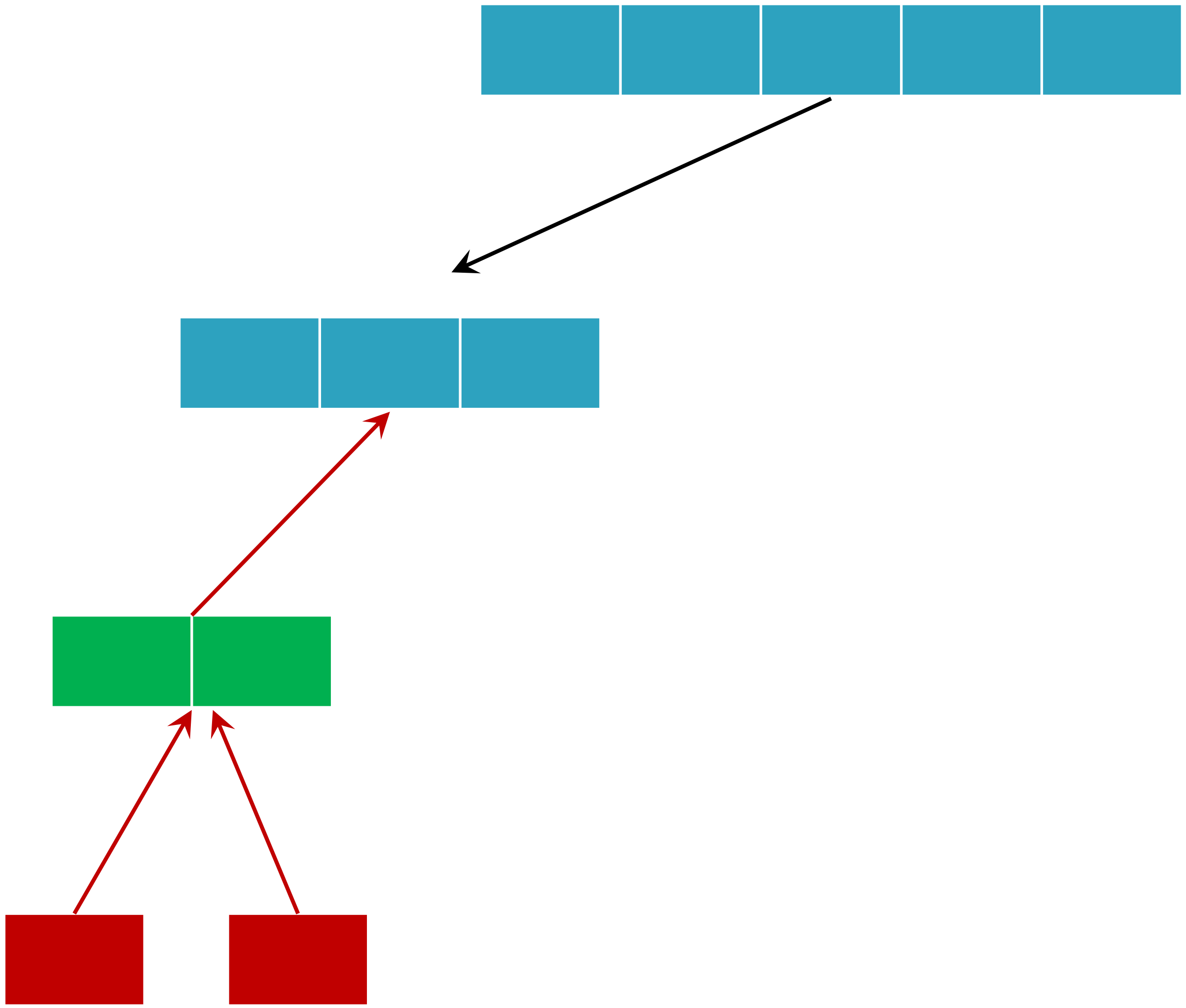
Metoda Divide et Impera



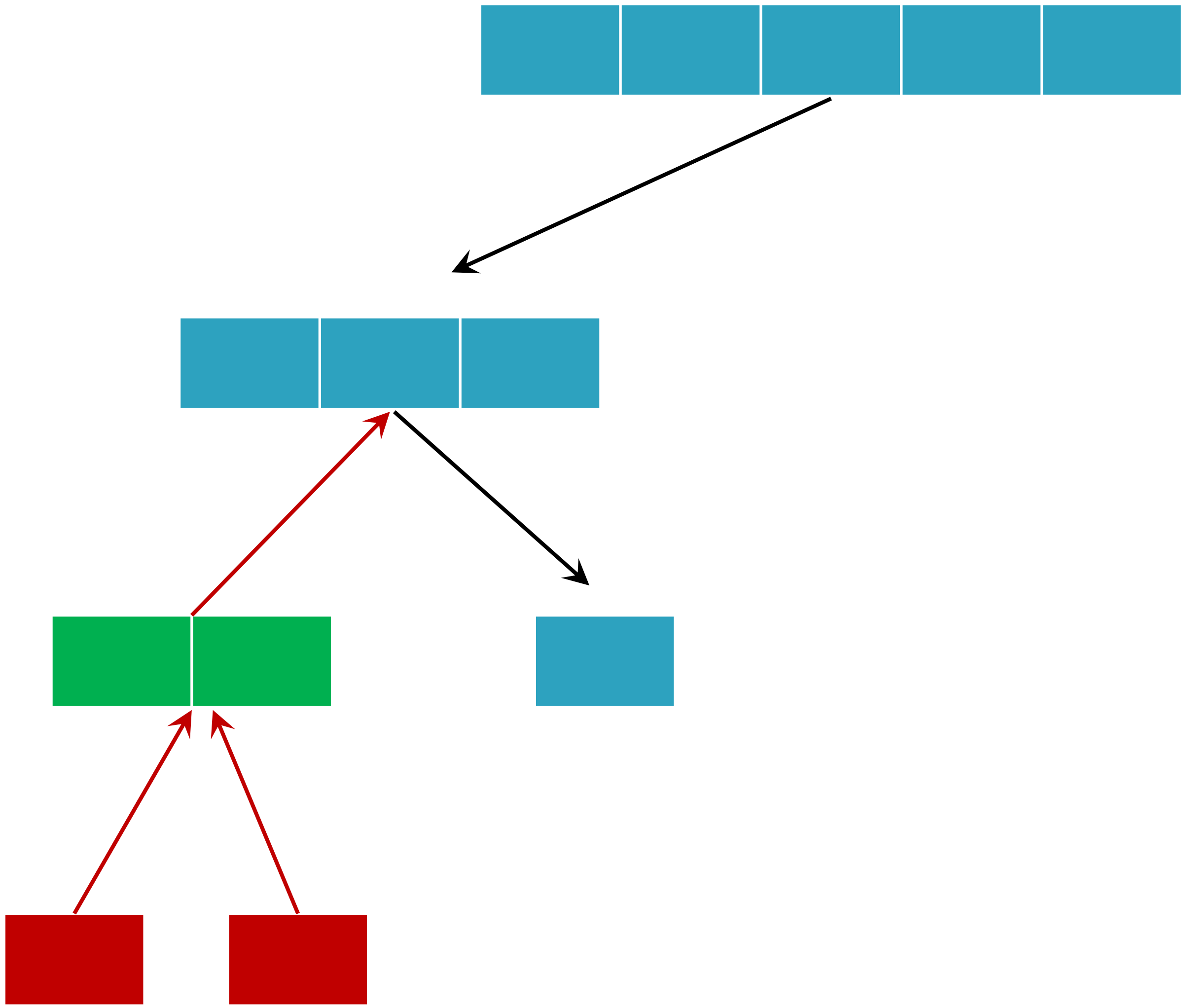
Metoda Divide et Impera



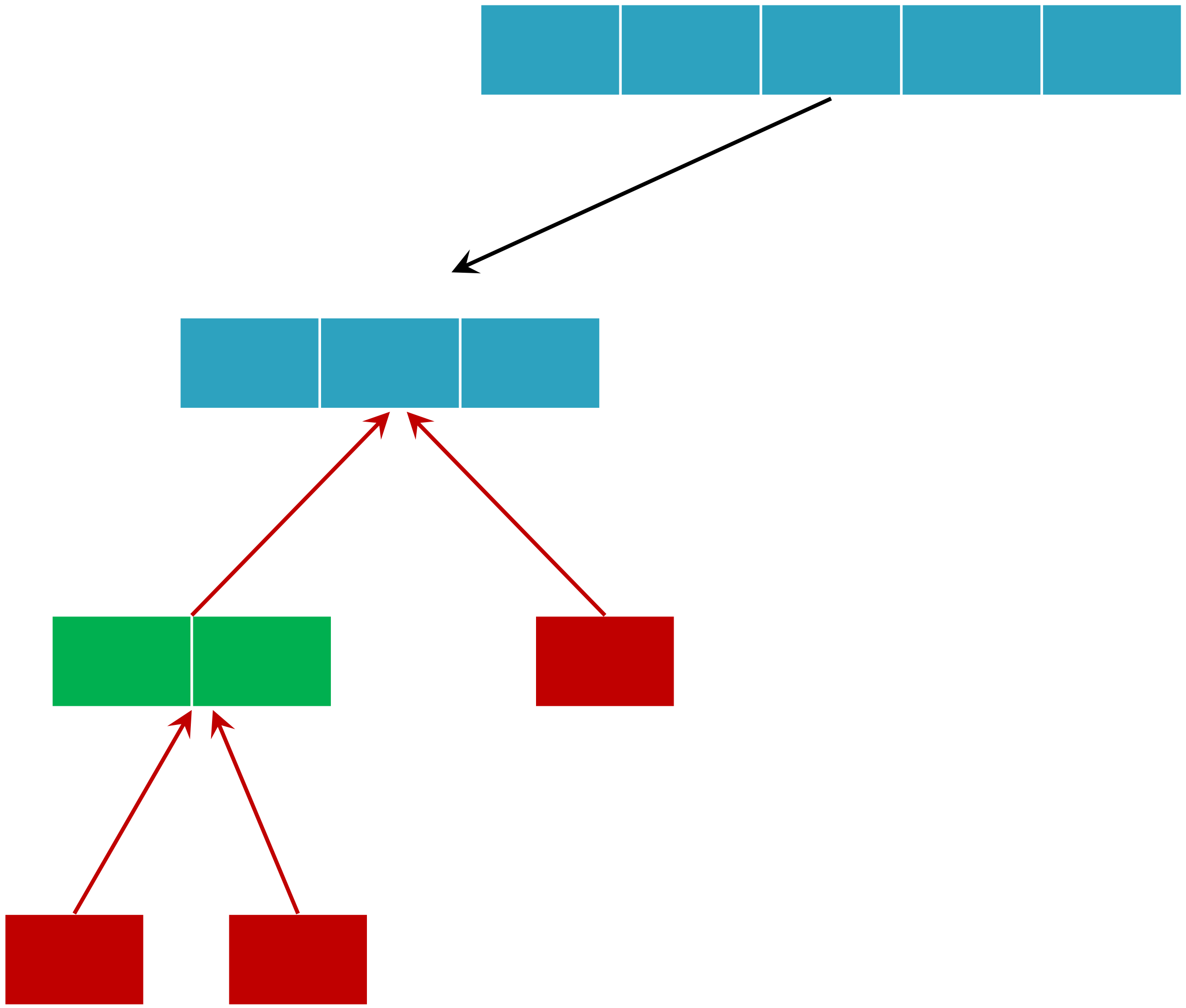
Metoda Divide et Impera



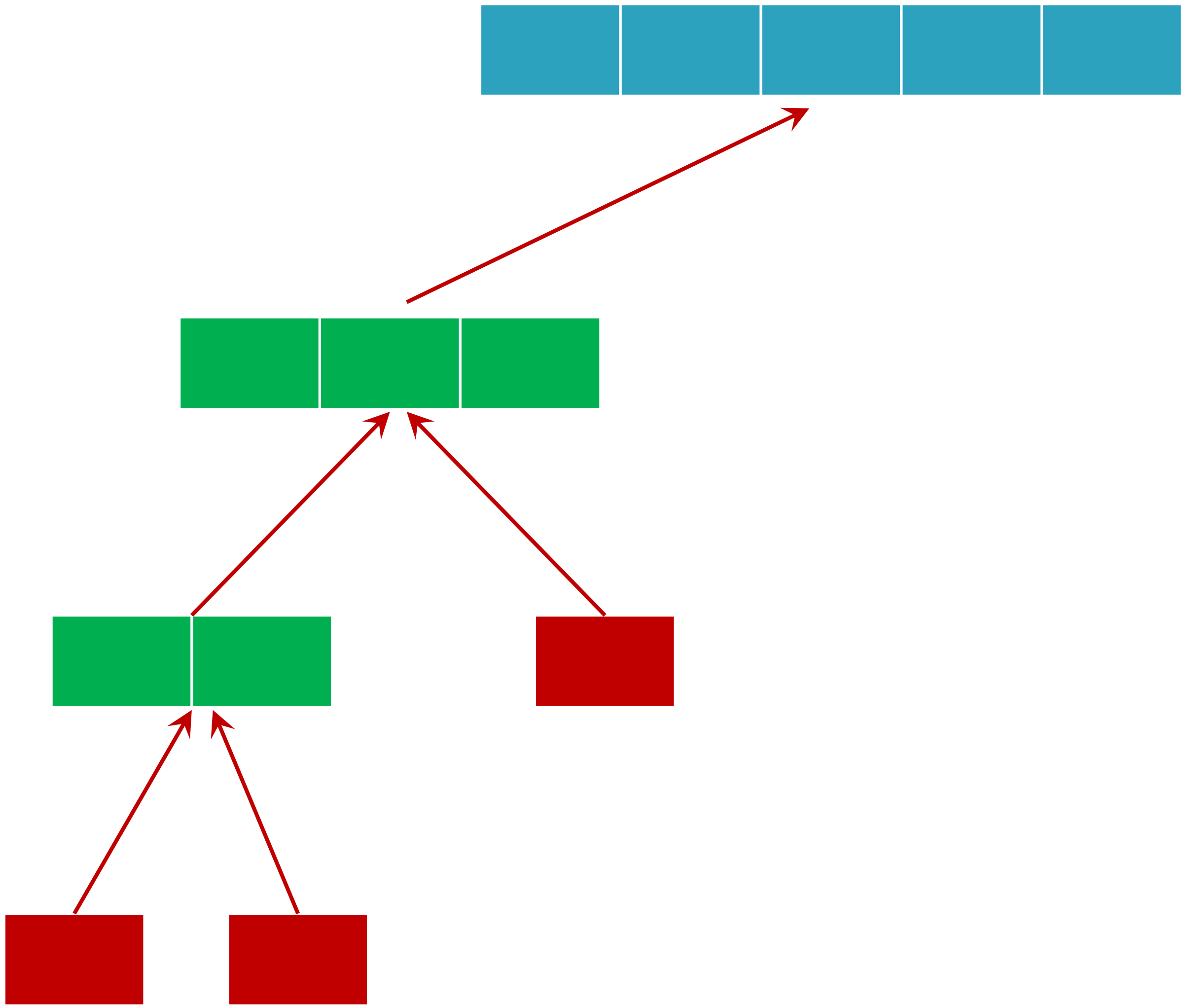
Metoda Divide et Impera



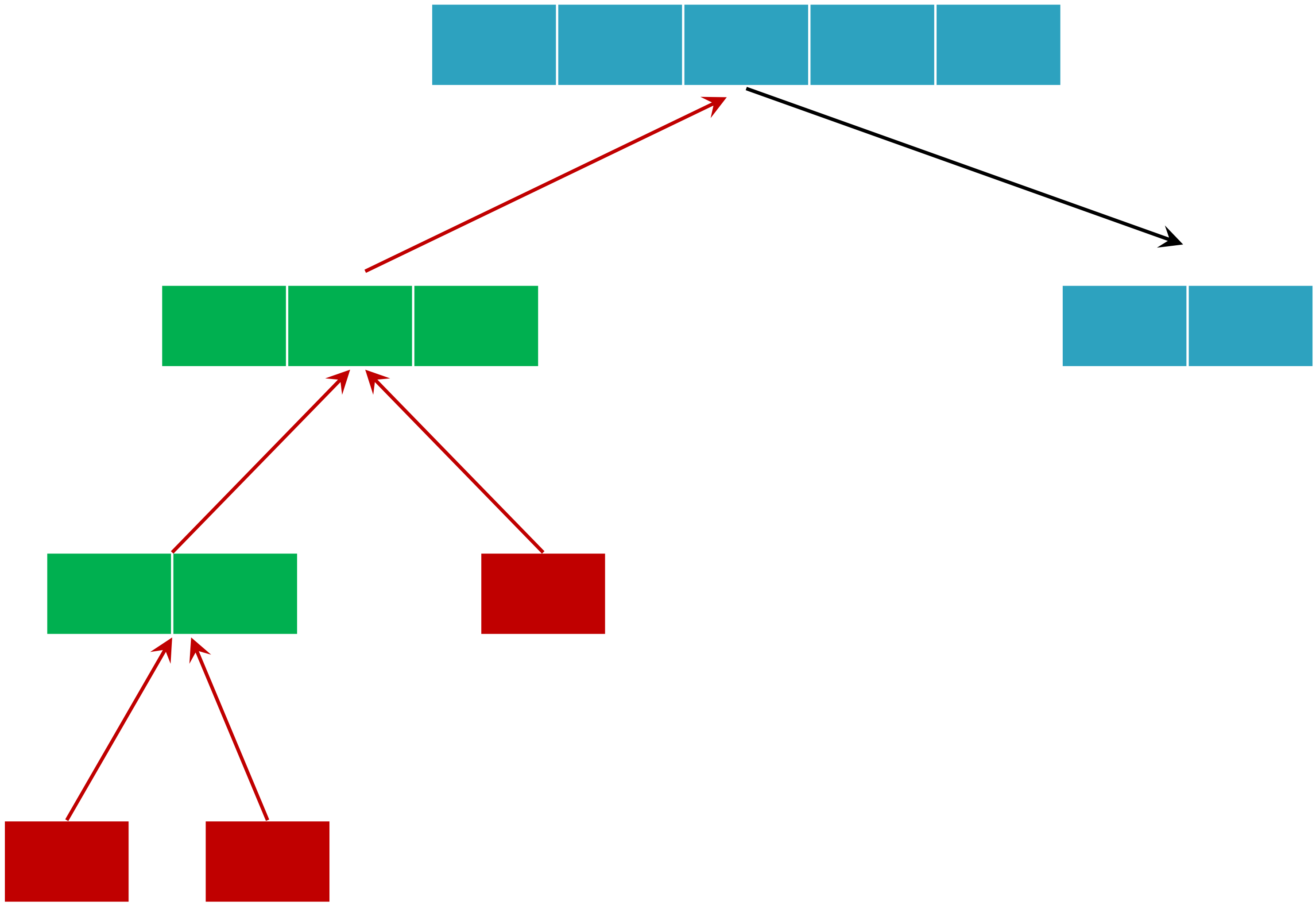
Metoda Divide et Impera



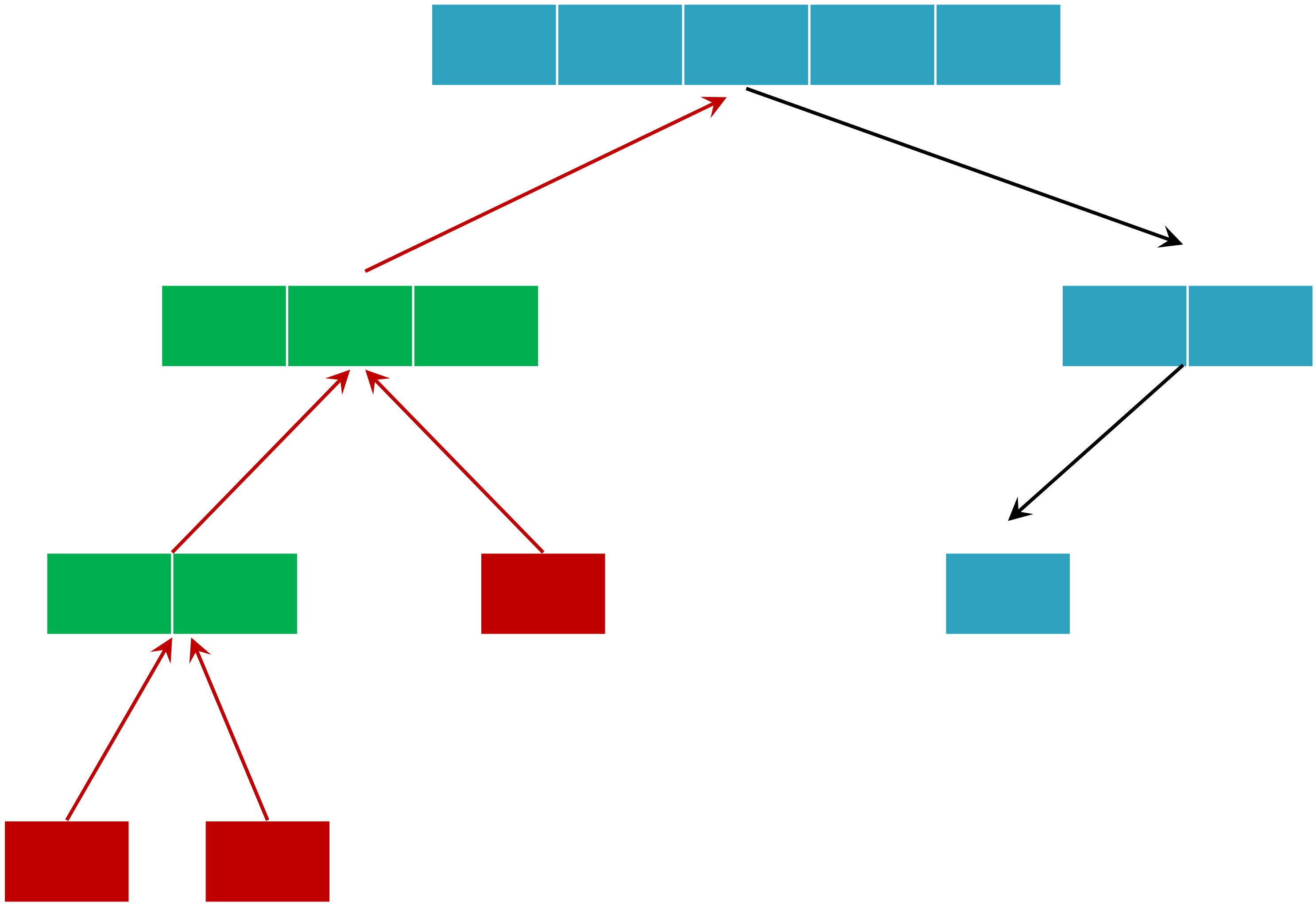
Metoda Divide et Impera



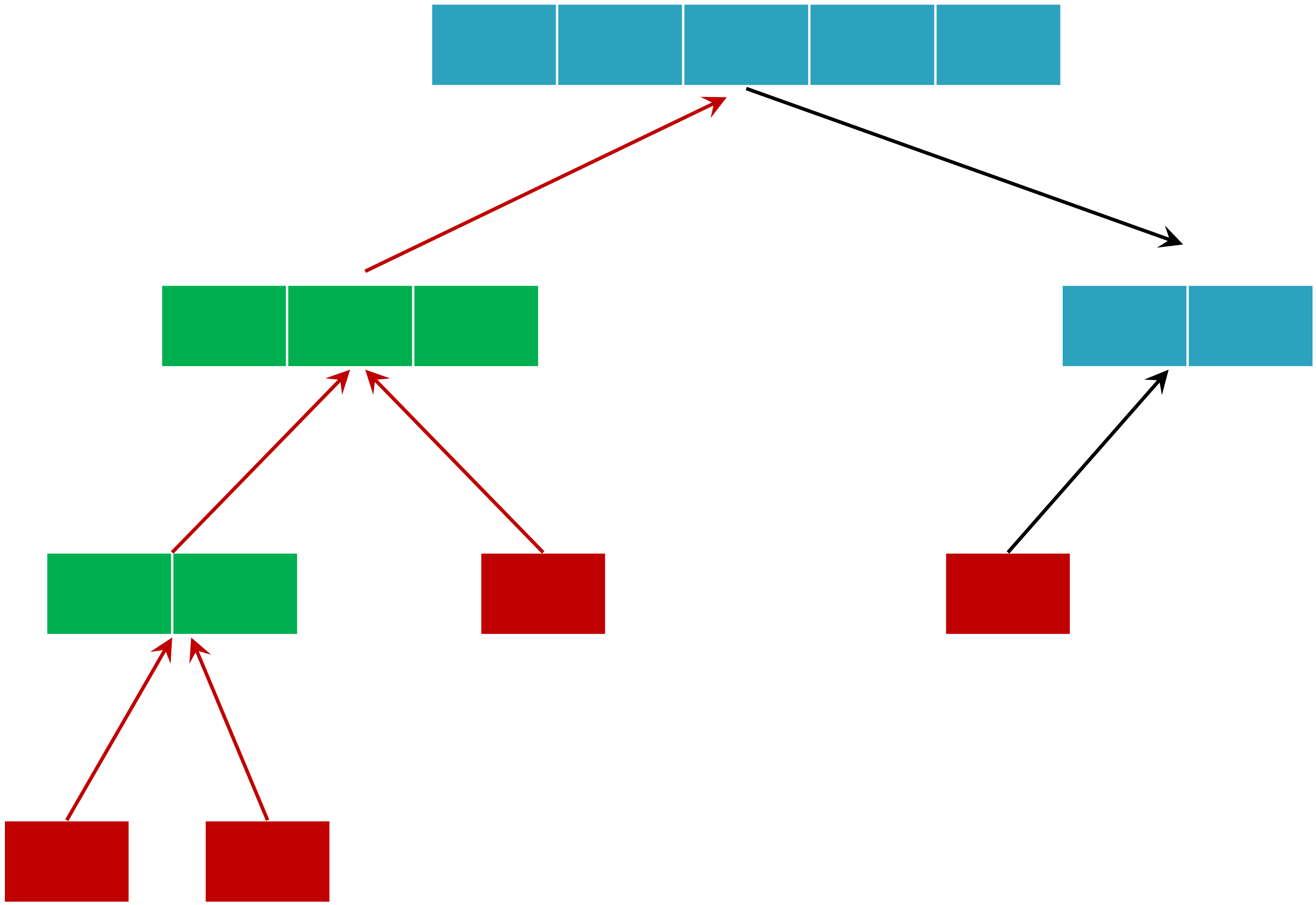
Metoda Divide et Impera



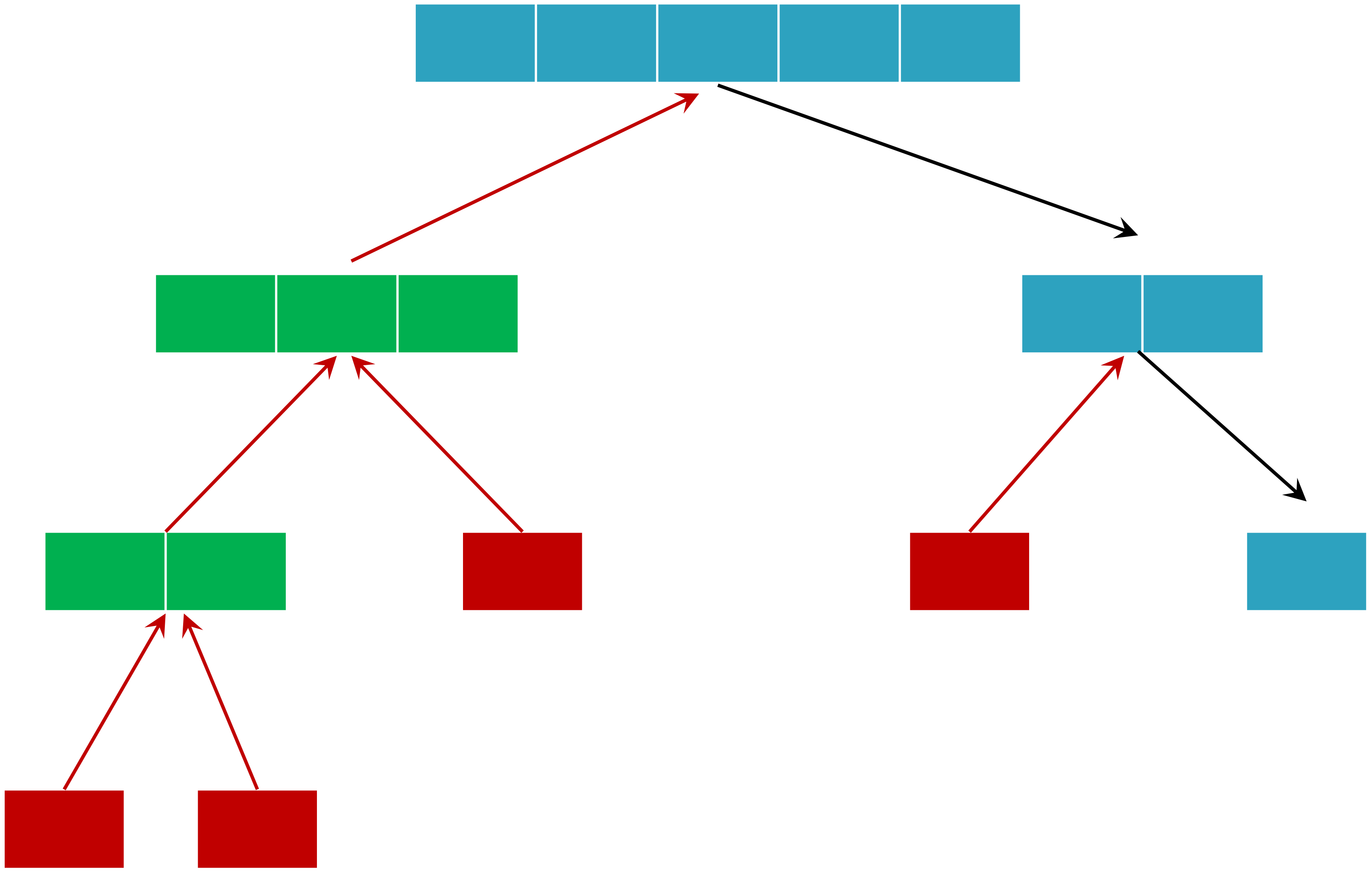
Metoda Divide et Impera



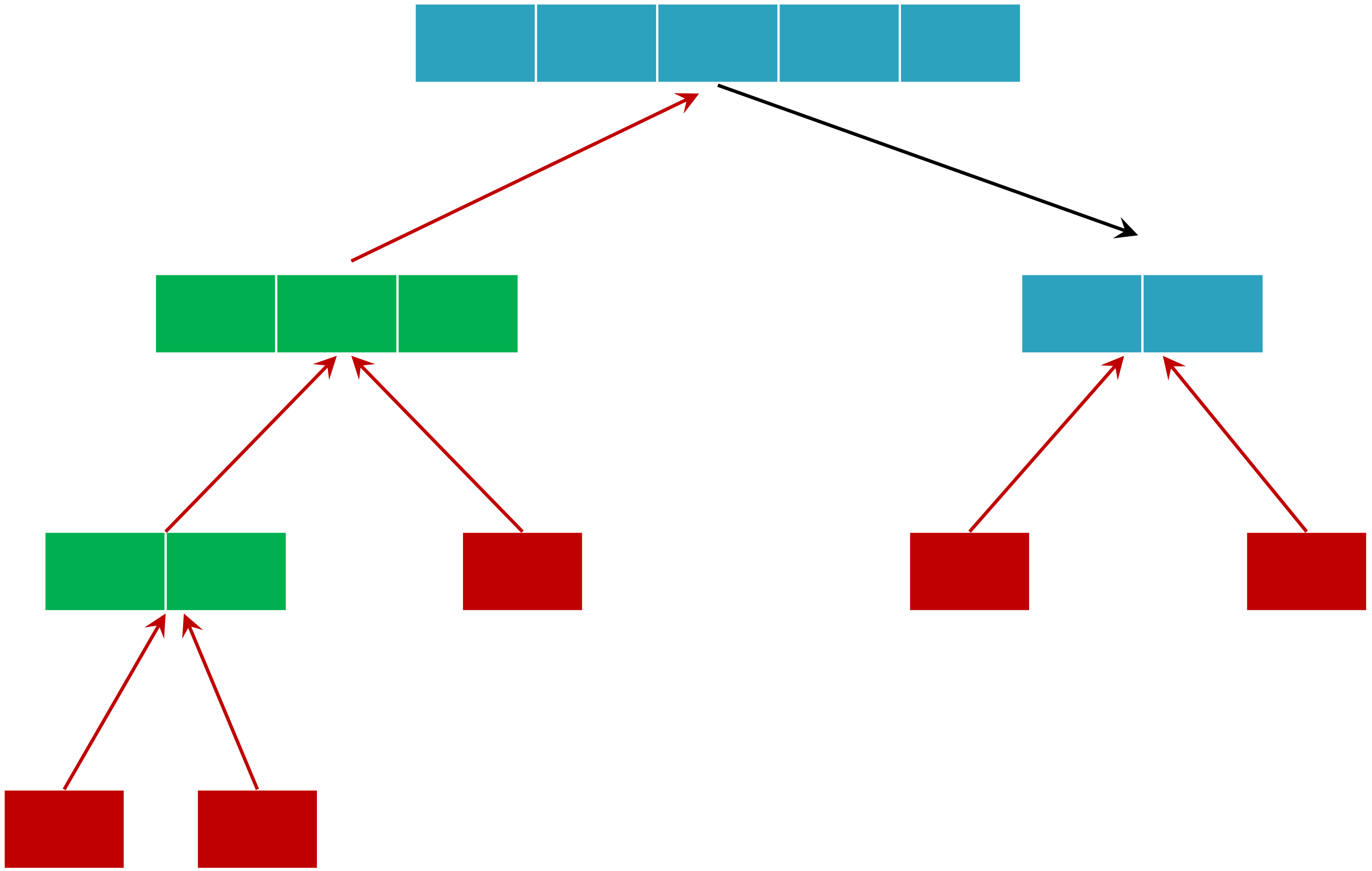
Metoda Divide et Impera



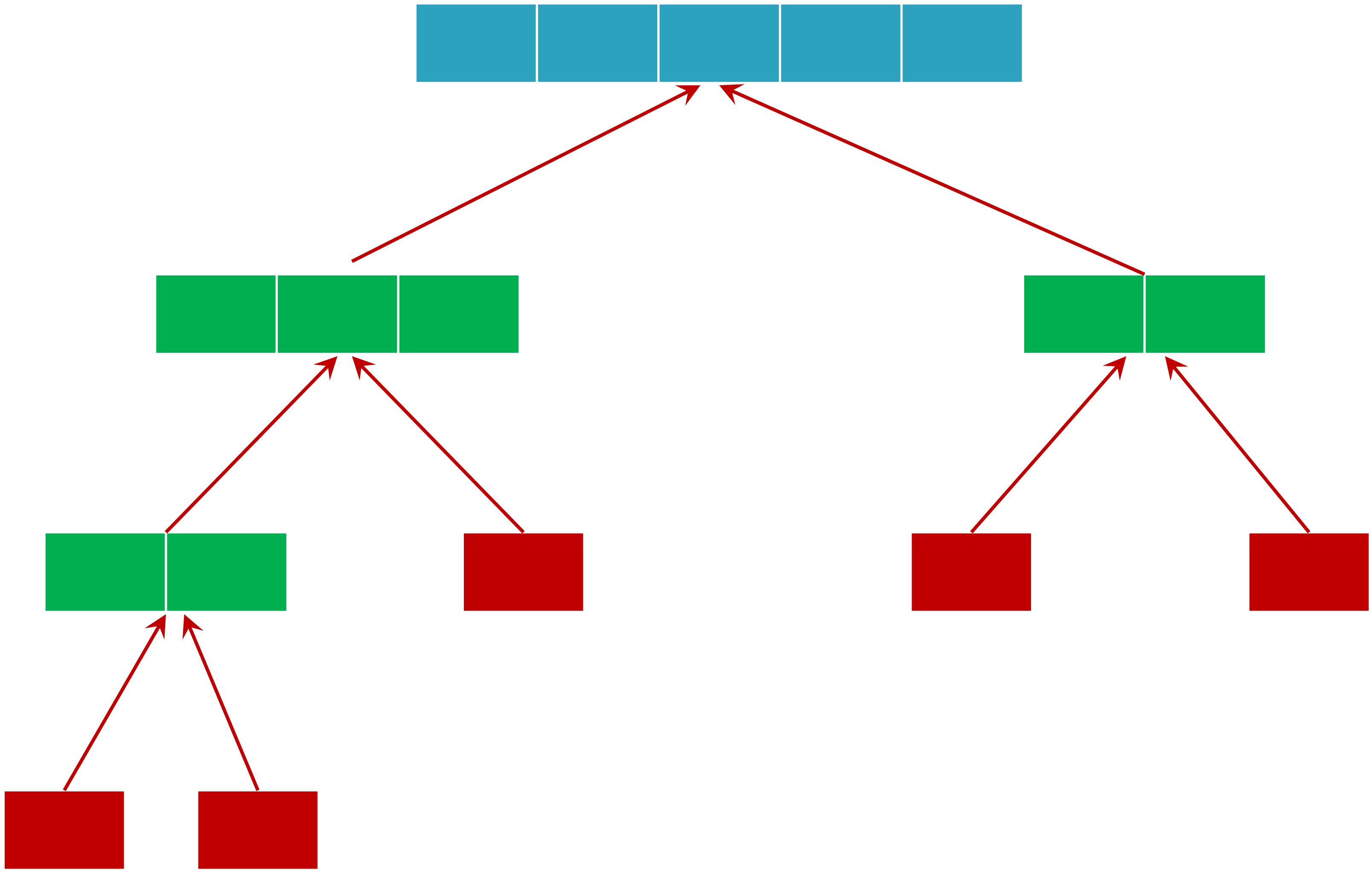
Metoda Divide et Impera



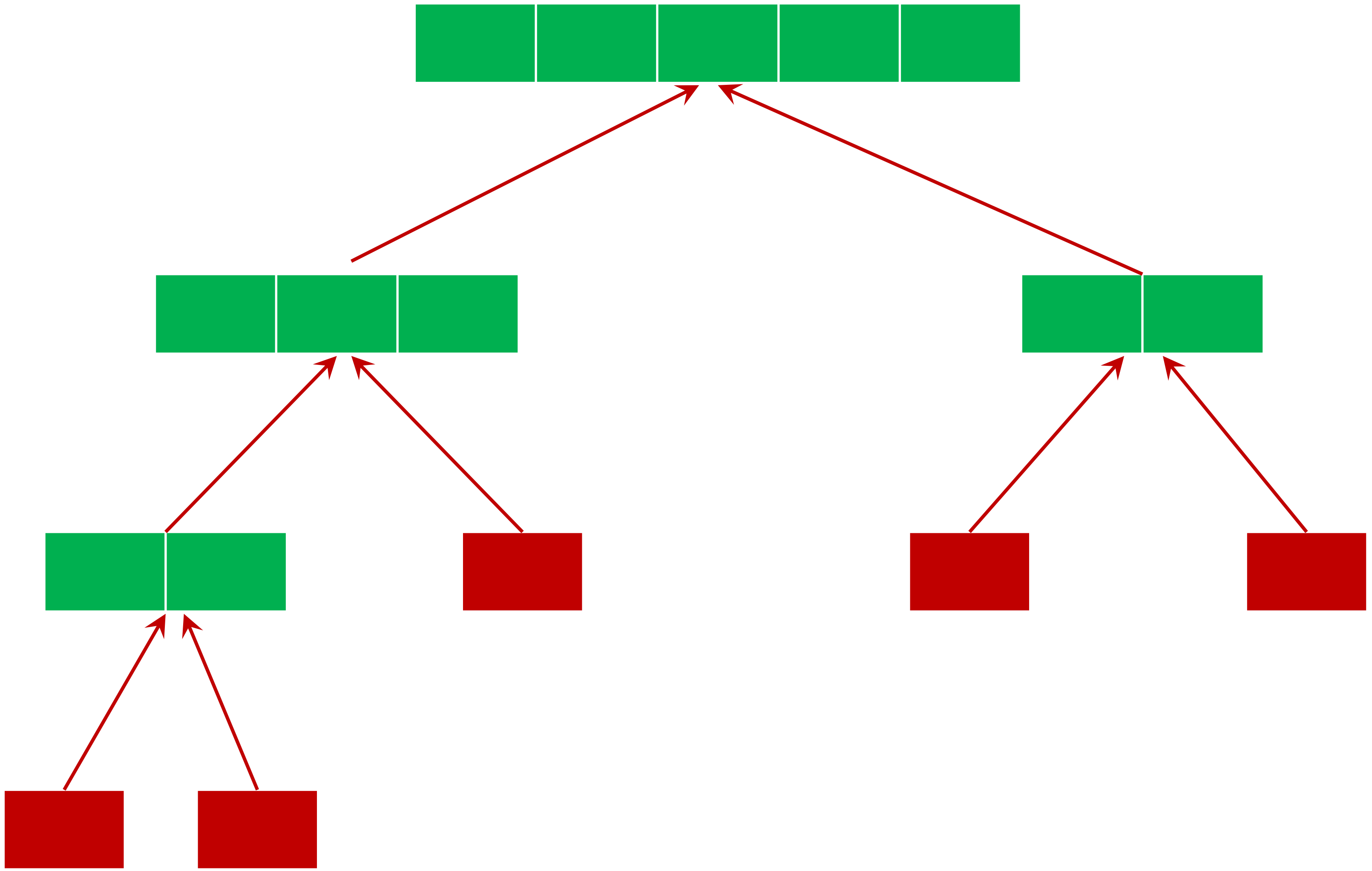
Metoda Divide et Impera



Metoda Divide et Impera



Metoda Divide et Impera



Metoda Divide et Impera

- **În termeni de arbori**, DI constă în
  - **construirea dinamică a unui arbore** (prin împărţirea în subprobleme)

  urmată de
  - **parcurgerea în postordine a arborelui** (prin asamblarea rezultatelor parţiale).

Exemplu – cmmdc al elementelor unui vector

```
function DivImp(p,u)
    if u-p<2
        r ← Cmmdc2(a[p],a[u])
    else
        m ← ⌊(p+u)/2⌋;
        r1 ← DivImp(p,m);
        r2 ← DivImp(m+1,u);
        r ← Cmmdc2 (r1,r2)
    return r
end;
```

Metoda Divide et Impera

- Ştim algoritmi polinomiali – complexitate

Exemple clasice (de recapitulat)

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

Sortarea prin interclasare – Aplicaţie

Numărare inversiuni

Problemă

Se consideră un vector cu n elemente distincte.
Să de determine <u>numărul</u> 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`

`(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, …, n`

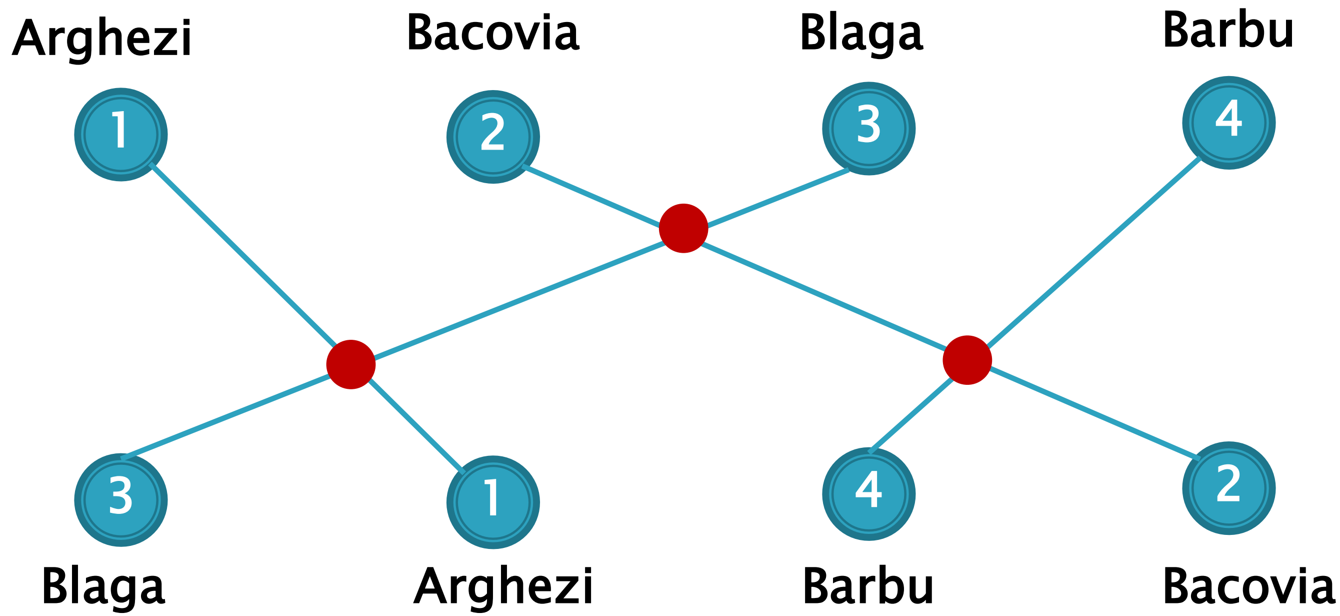
- Gradul de asemănare dintre clasificări = numărul de inversiuni din a doua clasificare

Aplicații

Preferințe utilizator 1	Arghezi 1	Bacovia 2	Blaga 3	Barbu 4
Preferințe utilizator 2	3 Blaga	1 Arghezi	4 Barbu	2 Bacovia

Aplicații

Preferințe utilizator 1



Preferințe utilizator 2

3 inversiuni

Numărare inversiuni

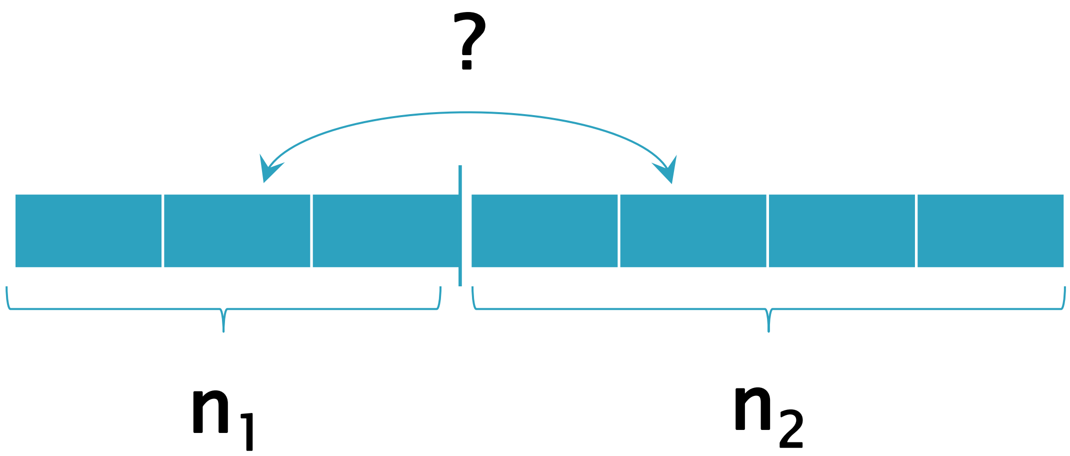
Numărul maxim de inversiuni ?

Numărare inversiuni

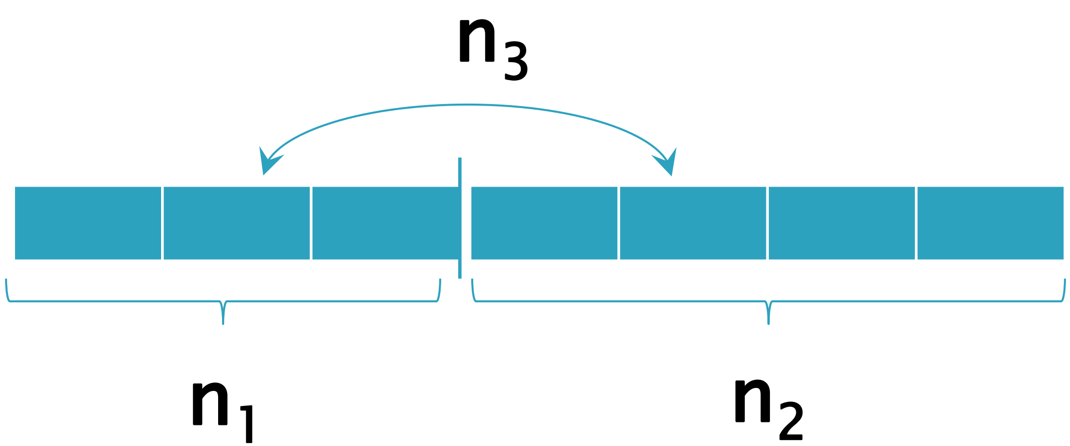
- **Algoritm $\Theta(n^2)$** – evident

Numărare inversiuni

- Algoritm Divide et Impera

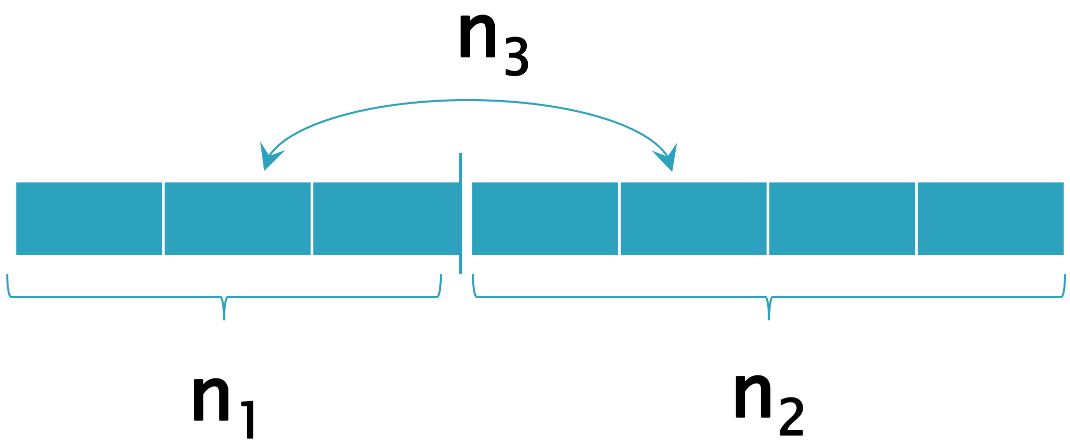


$n_1 + n_2 + ?$



$$n_1+n_2+n_3$$

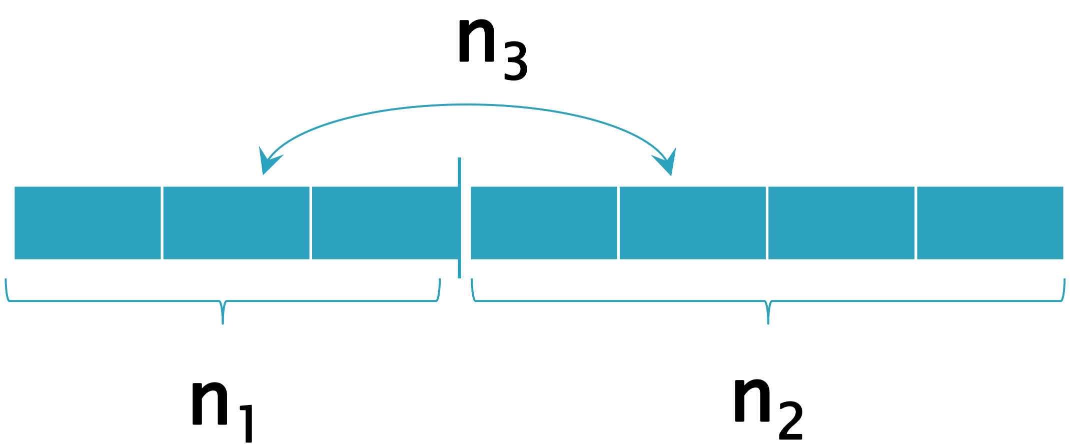
Cum calculăm eficient n_3?



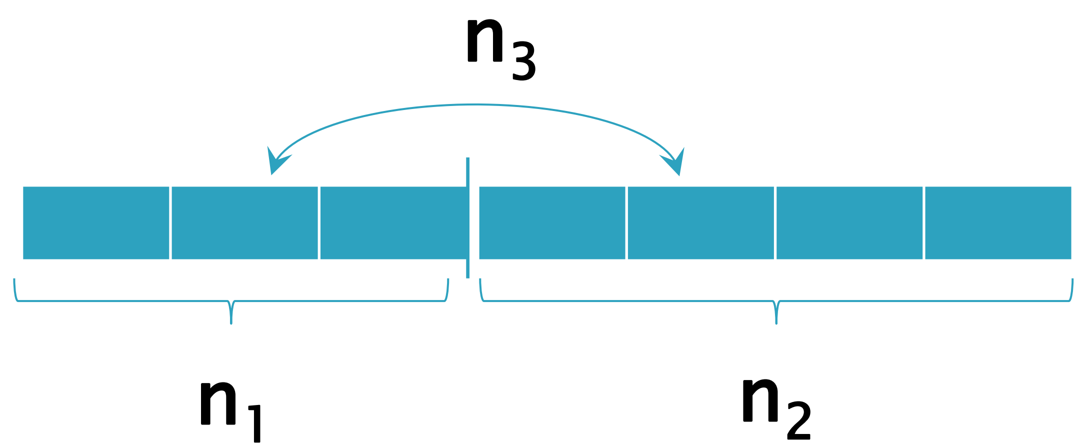
Cum calculăm eficient n_3?

- Considerând fiecare pereche (i,j) cu i în subvectorul stâng și j în cel drept – tot $O(n^2)$

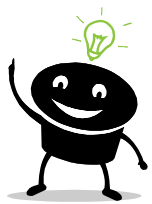
$$T(n) = 2T(n/2)+cn^2$$



Cum calculăm eficient n_3?

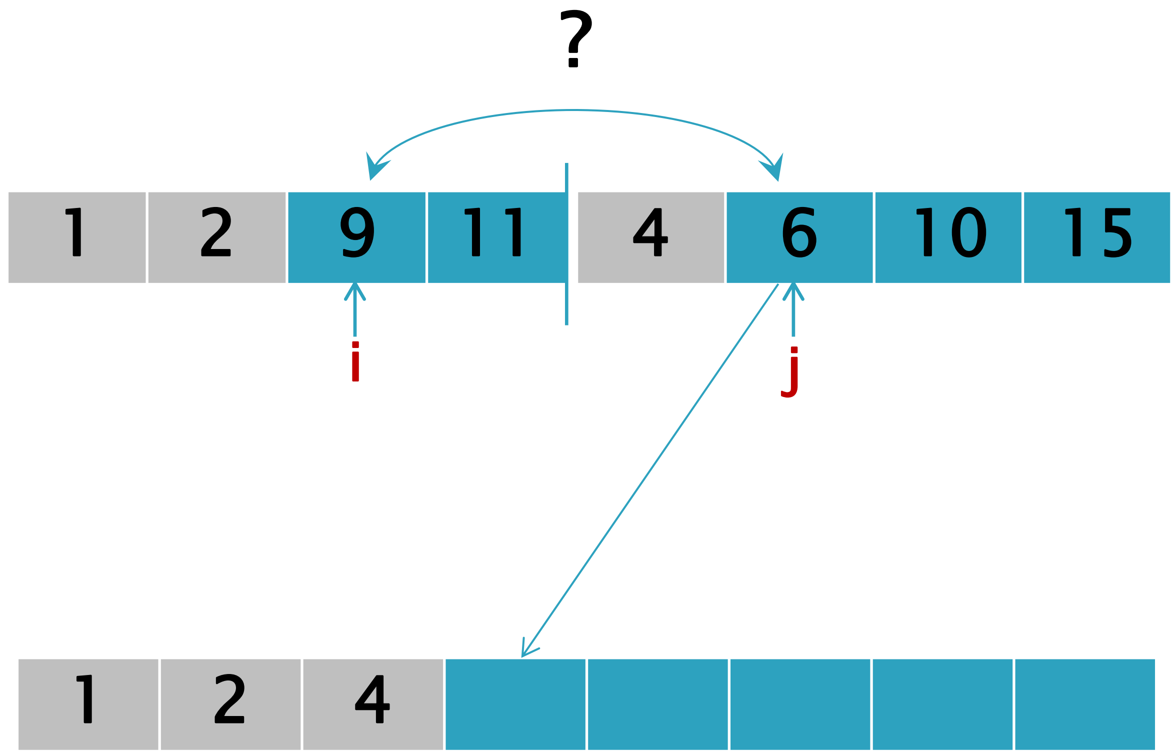


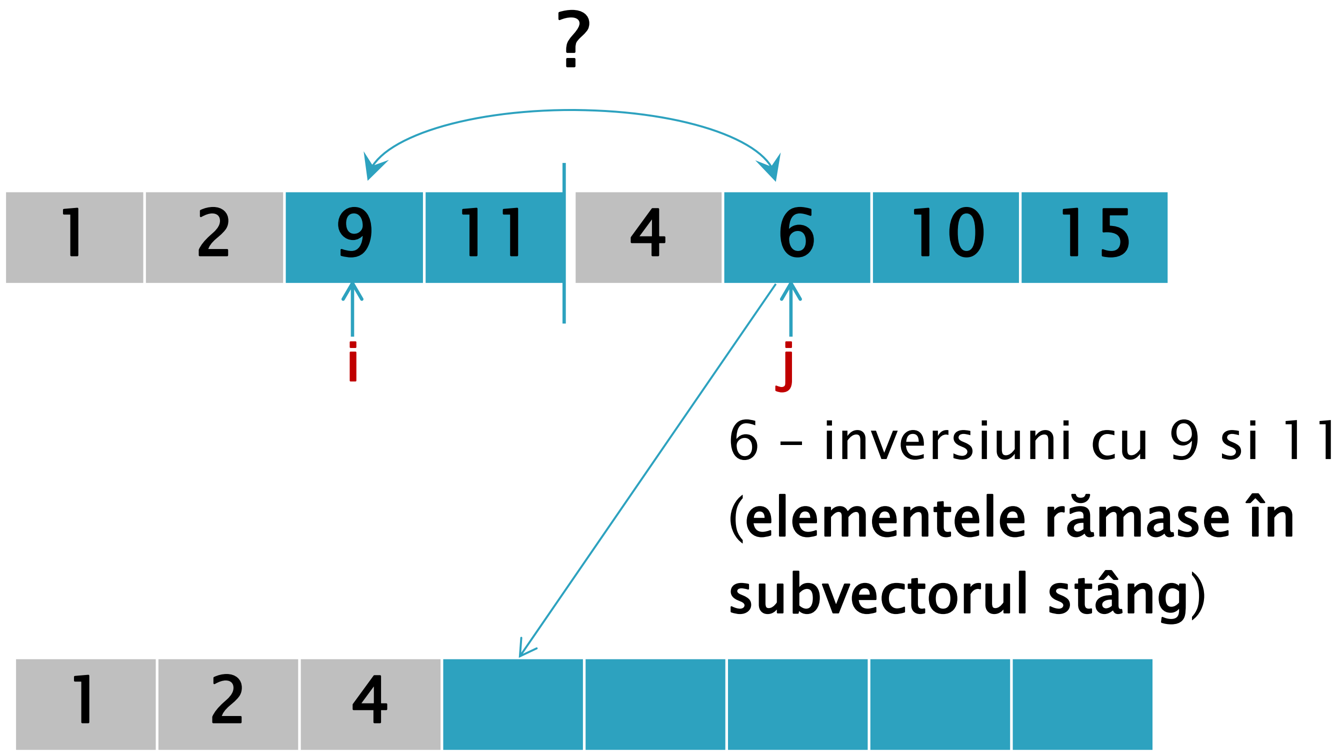
Cum calculăm eficient n_3?

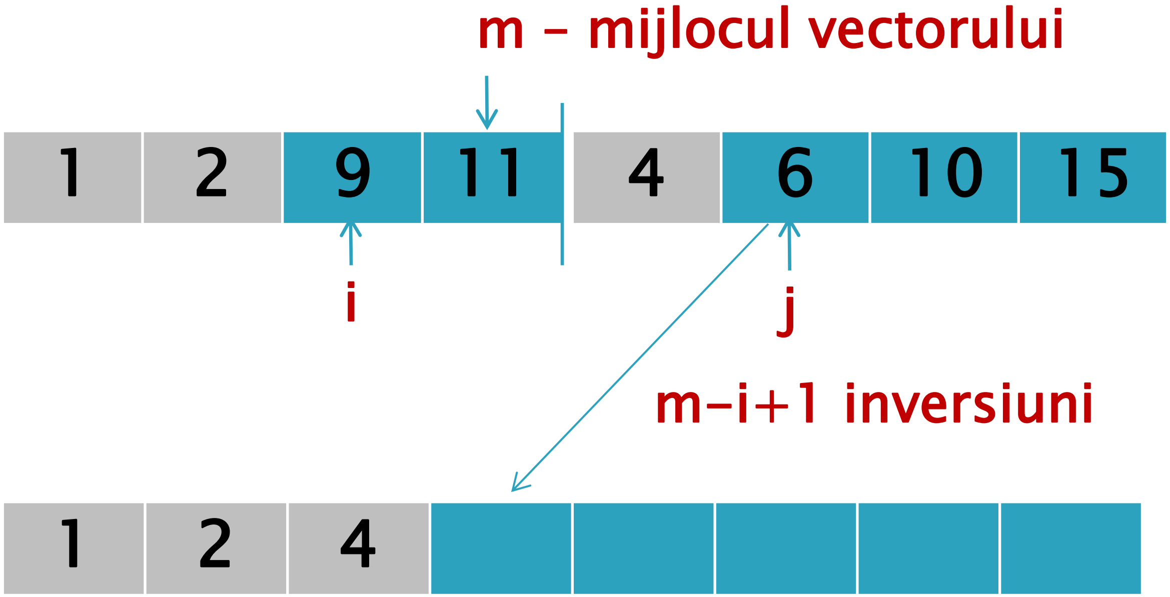


Dacă subvectorii stâng şi drept sunt **sortaţi crescător**, numărarea inversiunilor (i,j) date de elemente din subvectori diferiţi se poate face la interclasare

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

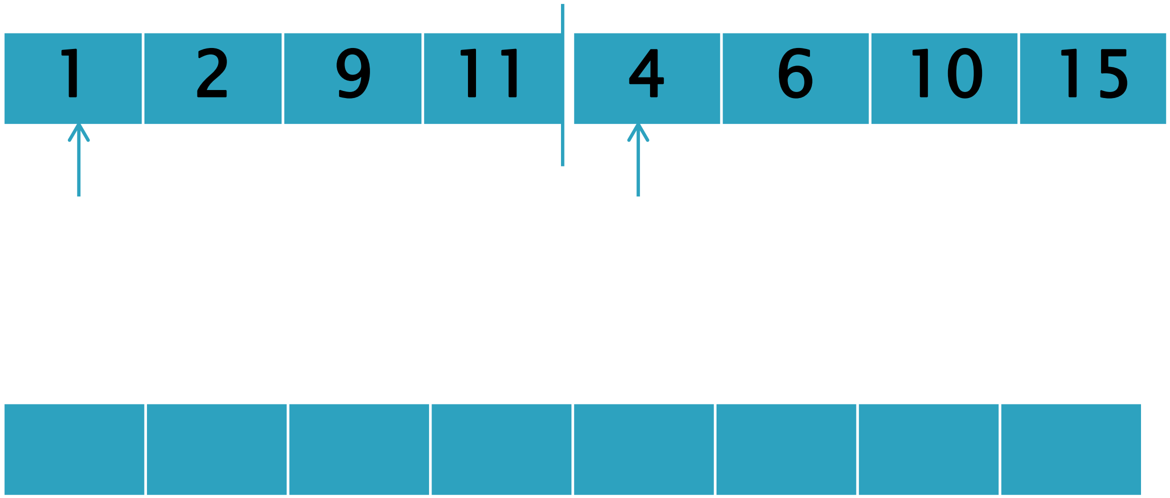

?
1 2 9 11 4 6 10 15
i
j
1 2 4

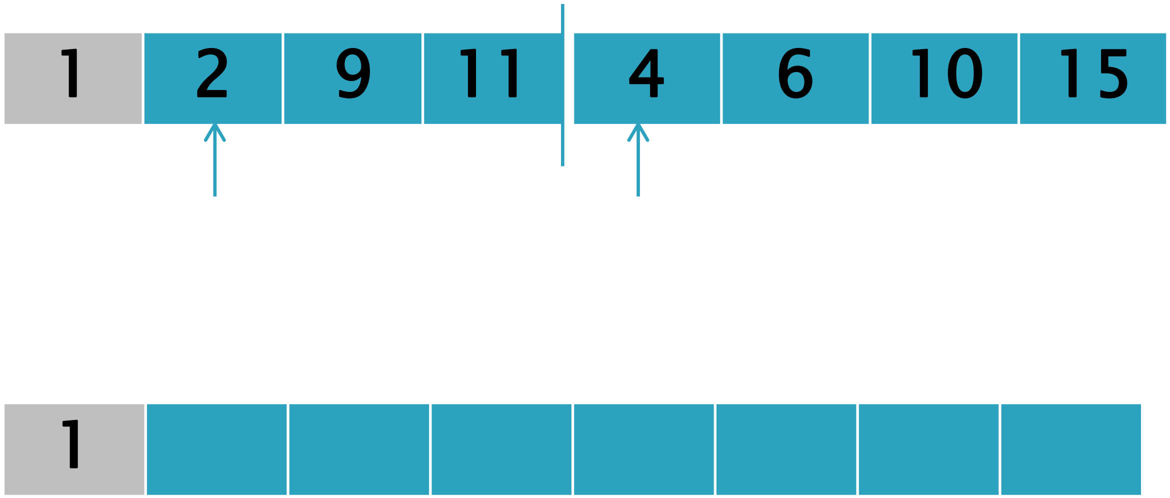

?
1 2 9 11 4 6 10 15
i
j
6 – inversiuni cu 9 si 11
(**elementele rămase în subvectorul stâng**)
1 2 4

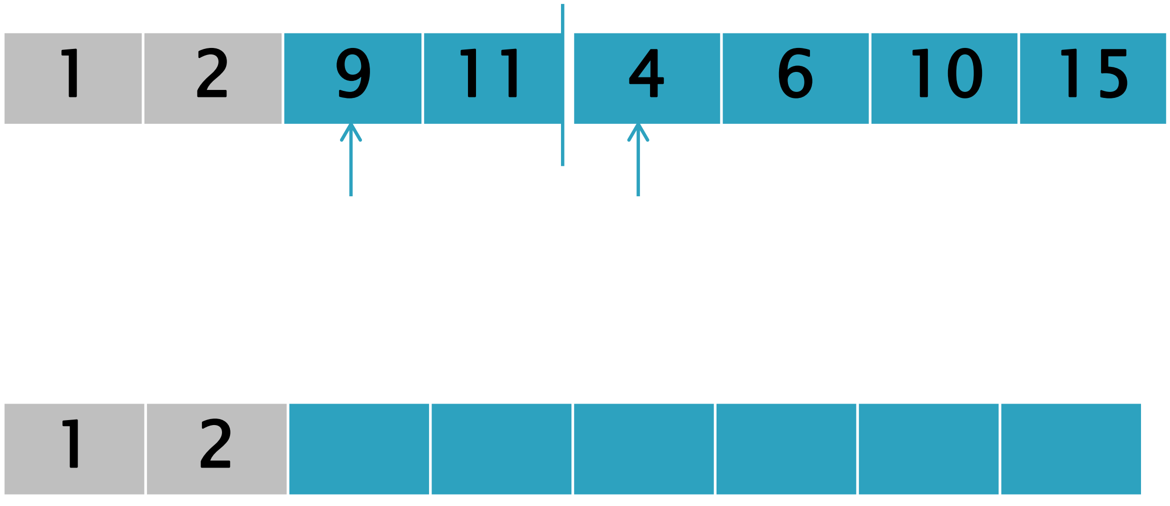


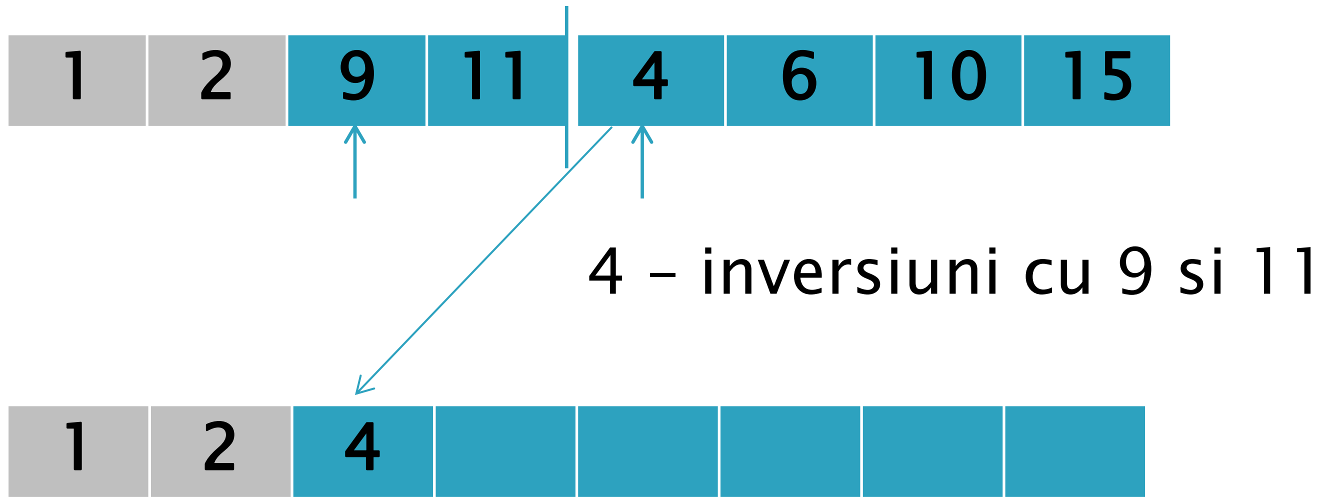
- a[j] determină m – i + 1 inversiuni

Exemplu – numărarea inversiunilor la interclasare

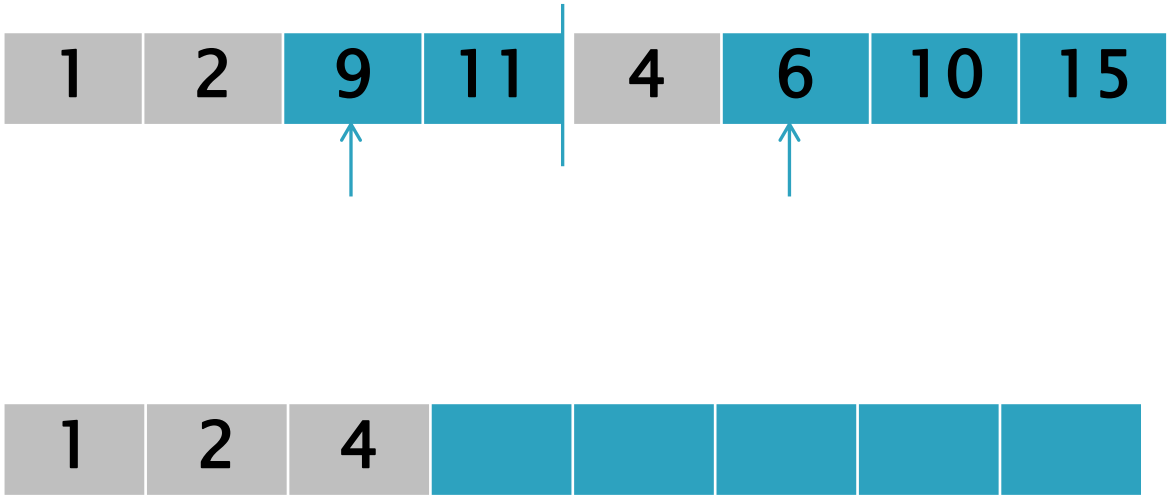

1 2 9 11
4 6 10 15


1
2
9
11
4
6
10
15
1

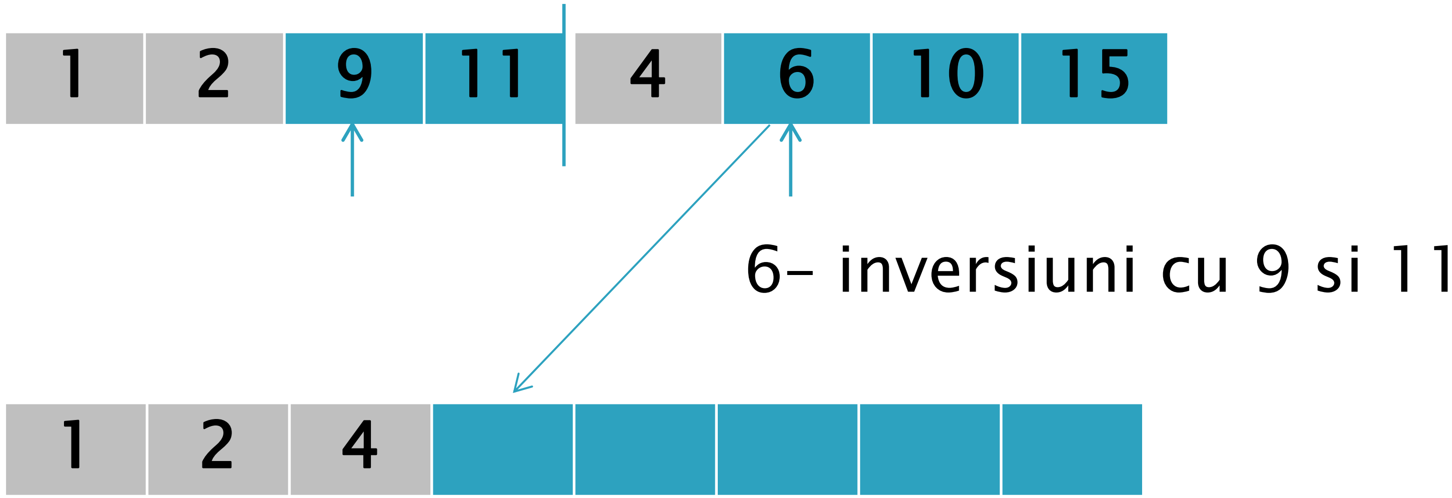

1
2
9
11
4
6
10
15
1
2



Inversiuni = 2



Inversiuni = 2



Inversiuni = 2 + 2

1	2	9	11	4	6	10	15

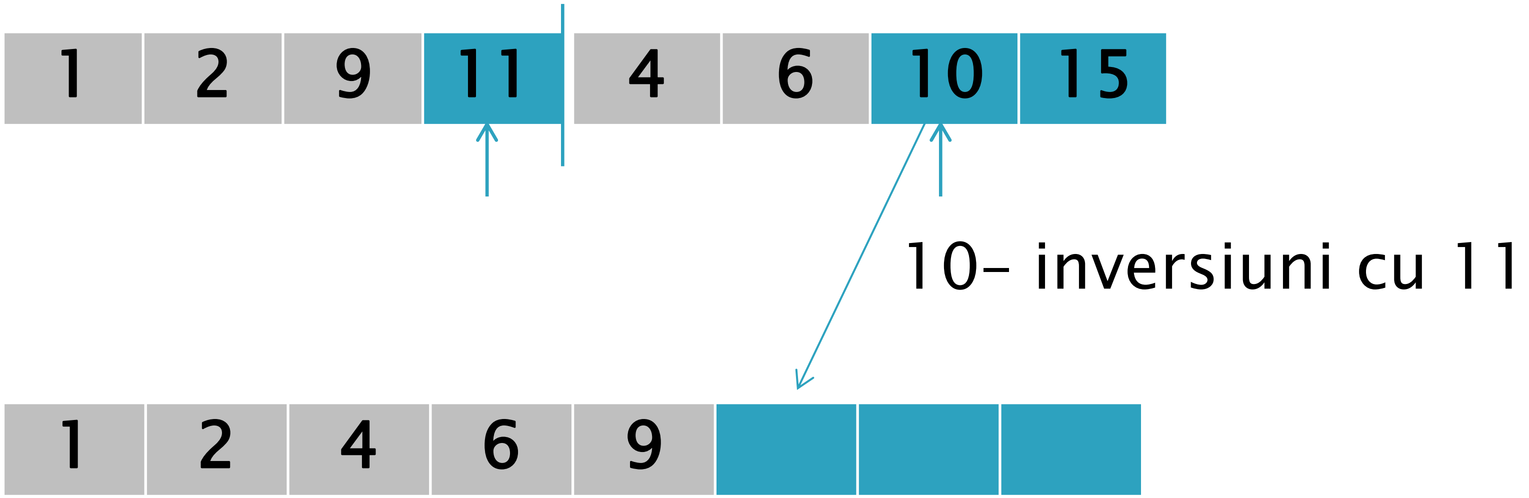
1	2	4	6				

Inversiuni = 2 + 2

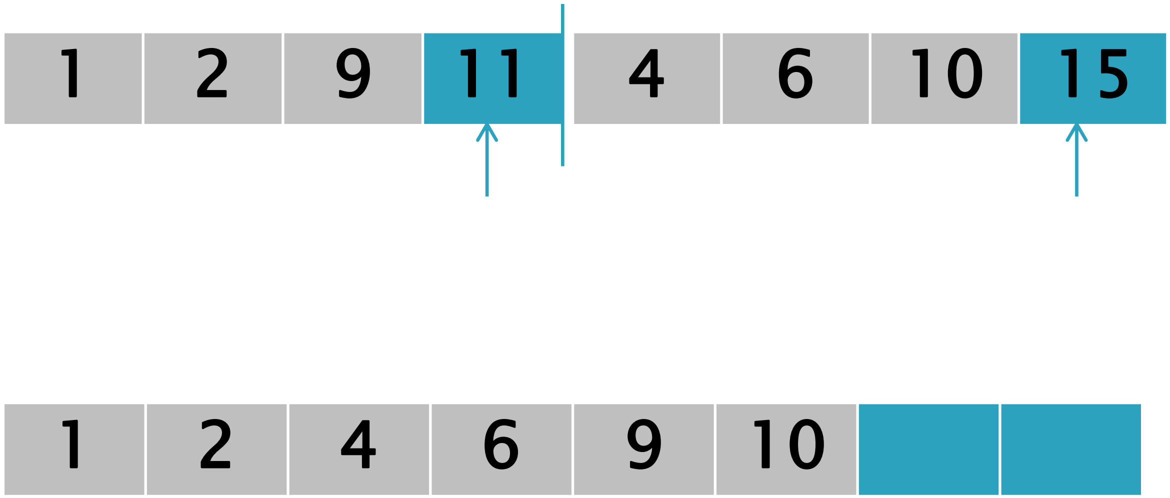
1	2	9	11	4	6	10	15

1	2	4	6	9			

Inversiuni = 2 + 2



Inversiuni = 2 + 2 + 1

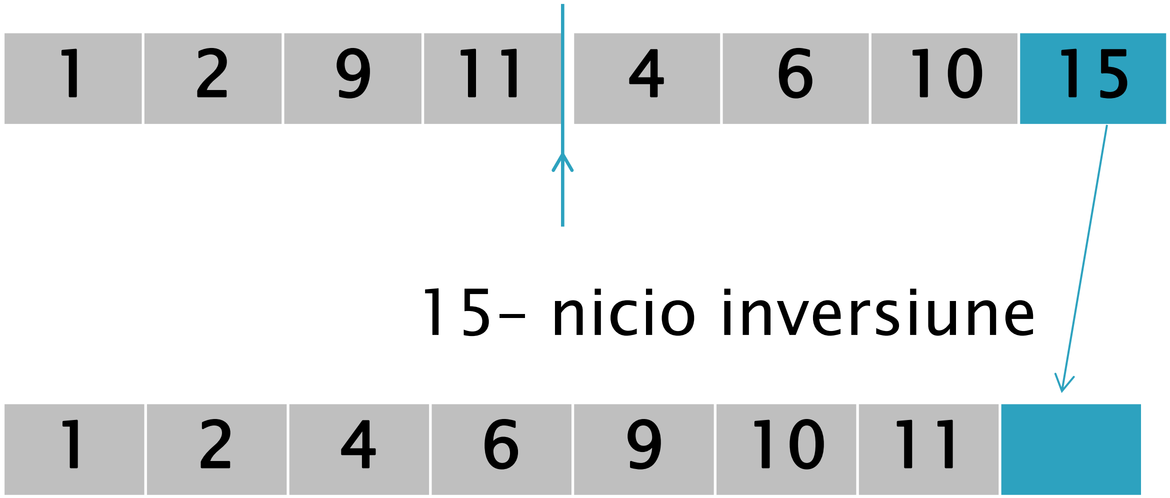


Inversiuni = 2 + 2 + 1

1	2	9	11	4	6	10	15

1	2	4	6	9	10	11	

Inversiuni = 2 + 2 + 1



Inversiuni = 2 + 2 + 1 + 0

1	2	9	11	4	6	10	15

1	2	4	6	9	10	11	15

Inversiuni = 2 + 2 + 1 + 0 = 5

Algoritm

```
int nrInv(int p,int u){
    if (p == u){
        return 0;
    }
    else {
        int m = (p+u)/2;
        int n1 = nrInv(p,m);
        int n2 = nrInv(m+1,u);
        return interclasare(p,m,u)+n1+n2;
    }
}
```

```
int interclasare(int p,int m,int u){
        int b[]=new int[u-p+1];
        int i=p,j=m+1, k=0,nr=0;
        while ((i<=m)&&(j<=u)){
                if (a[i]<=a[j]){ b[k]=a[i]; i++;}
                else{ b[k]=a[j]; j++;
                        nr = nr + (m-i+1);
                }
                k++;
        }
        while(i<=m){ b[k]=a[i]; k++; i++; }
        while(j<=u){ b[k]=a[j]; k++; j++; }
        for (i=p;i<=u;i++)
                a[i]=b[i-p];
        return nr;
}
```

Complexitate

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

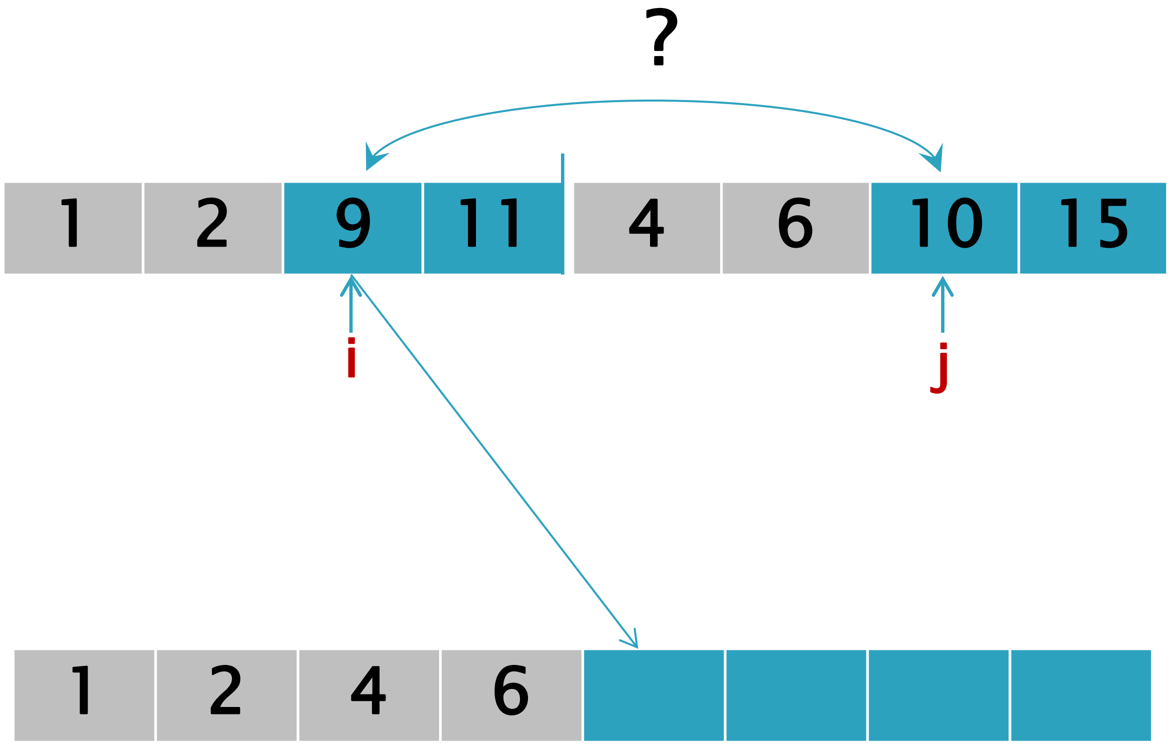
$$= 2\ [2T(2^{k-2}) + c \cdot 2^{k-1}] + c \cdot 2^k = 2^2 T(2^{k-2}) + 2 \cdot c \cdot 2^k$$

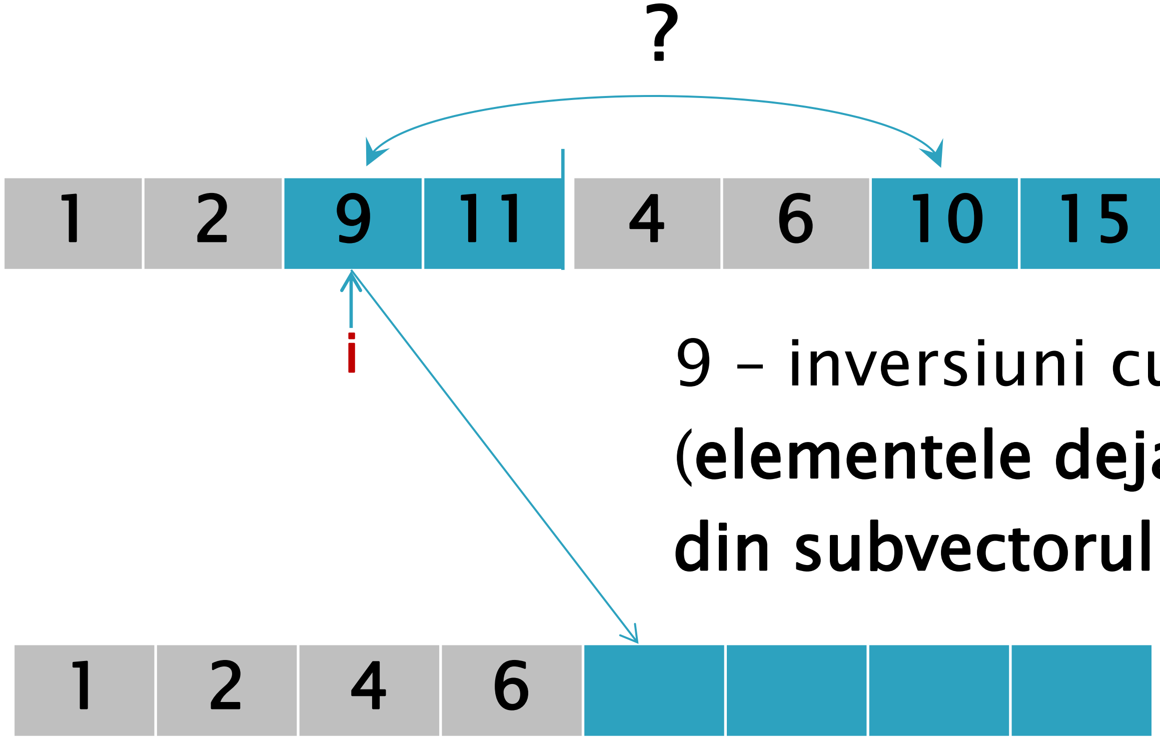
$$= \ldots = 2^i T(2^{k-i}) + i \cdot c \cdot 2^k =$$

$$= 2^k T(1) + k \cdot c \cdot 2^k = nt_0 + c \cdot\ n\ \cdot \log_2 n$$

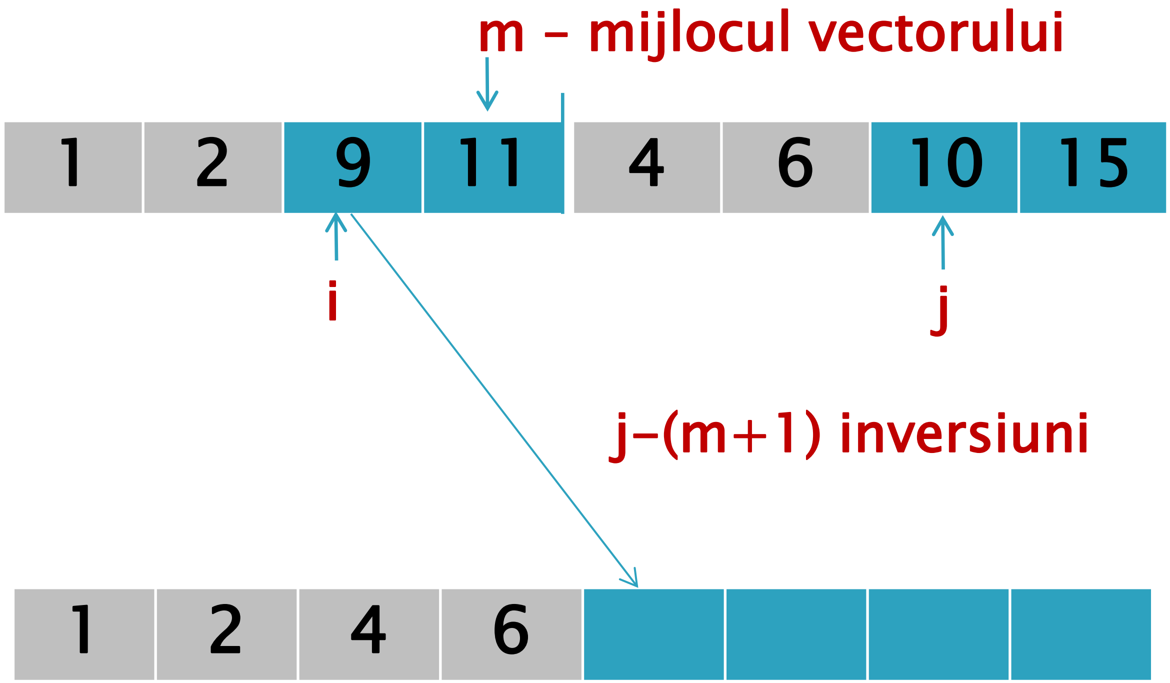
- **Arbore subprobleme**

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


?
1
2
9
11
4
6
10
15
i
j
1
2
4
6



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



- a[i] determină j – m – 1 inversiuni

```
int interclasare(int p,int m,int u){
      int b[]=new int[u-p+1];
      int i=p,j=m+1, k=0,nr=0;
      while ((i<=m)&&(j<=u)){
            if (a[i]<=a[j]){ b[k]=a[i]; i++;
                  nr = nr + (j-m-1);
            }
            else{ b[k]=a[j];j++;    }
            k++;
      }
      while(i<=m){ b[k]=a[i]; k++; i++;
            nr = nr + (j-m-1);
      }
      while(j<=u){ b[k]=a[j]; k++; j++; }
      for (i=p;i<=u;i++)
            a[i]=b[i-p];
      return nr;
}
```