Metoda Divide et Impera

desparte şi stăpâneşte

FMI – TAP – Curs, verman@fmi.unibuc.ro

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.

Metoda Divide et Impera

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

  urmată de

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).

Schema generală

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

Schema generală

```
function DivImp(p,u)
    if u-p<ε
        r ← RezolvaDirect(p,u)
    else
        m ← Pozitie(p,u);
        r1 ← DivImp(p,m);
        r2 ← DivImp(m+1,u);
        r ← Combina(r1,r2)
    return r
end
```

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

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;
```

Căutarea binară

- **Se consideră vectorul $a=(a_1,\dots,a_n)$ ordonat crescător şi o valoare x. Se cere să se determine dacă x apare printre componentele vectorului.**

- Mai exact căutăm perechea (b,i) dată de:
  - `(true,i)` dacă $a_i = x$;
  - `(false,i)` dacă $\mathbf{a_{i-1} < x < a_i}$,

  unde, prin convenţie,

  $$a_0 = -\infty,\ a_{n+1} = +\infty.$$

Căutarea binară

```
void cautBin(int a[], int n, int x){
    int p = 1, u = n;
    while (p<=u){
        int i = (p+u)/2; //compar cu elem. din mijl. secv.
        if (a[i] > x )
            u = i-1; //caut in subsecv stanga
        else
            if (a[i] == x) {
                System.out.println("true "+i); return;
            }
            else
                p = i+1; //caut in subsecv dreapta
    }
    System.out.println("false "+p);
}
```

Căutarea binară

- **Corectitudine:**
  - La fiecare pas are loc relaţia (*invariant*)

$$a[p-1] < x < a[u+1]$$

Căutarea binară

- **Corectitudine:**
  - La fiecare pas are loc relaţia (*invariant*)

$$a[p-1] < x < a[u+1]$$

  - Ajungem la cazul p > u doar dacă la pasul anterior eram în una dintre situaţiile
    - **$u = p+1$ şi $x <= a[i]$ (unde $i = p$) $\Rightarrow$ $a[p-1] < x < a[p]$**
    - **$u = p$ (atunci $i = p = u$)**
      - **$x <= a[i]$ $\Rightarrow$ $a[p-1] < x < a[p]$**
      - **$x > a[i]$ $\Rightarrow$ $p' = p+1$ şi**

        $a[p'-1] = a[p] < x < a[u+1] = a[p']$

Căutarea binară

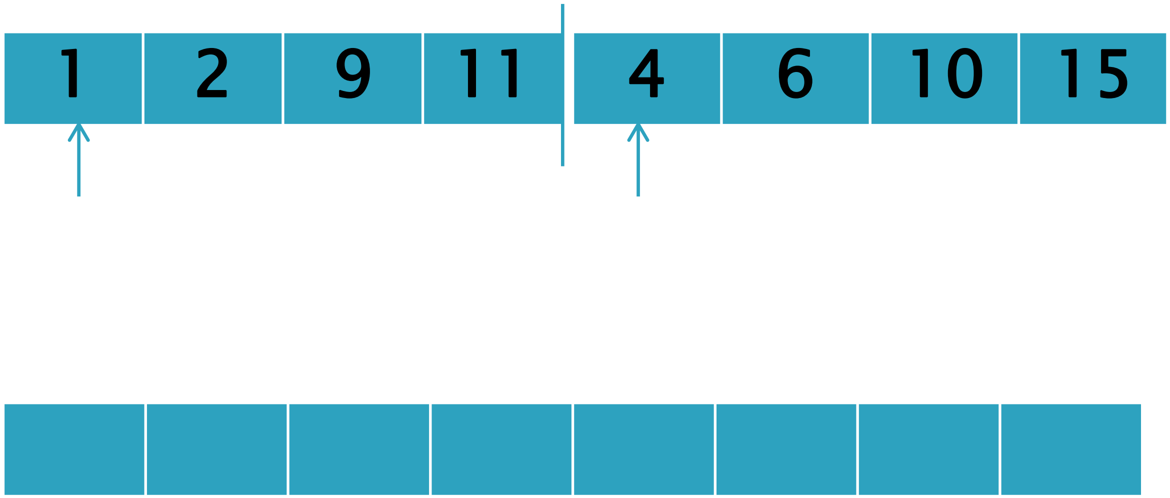
- **Complexitate:** O(log n)

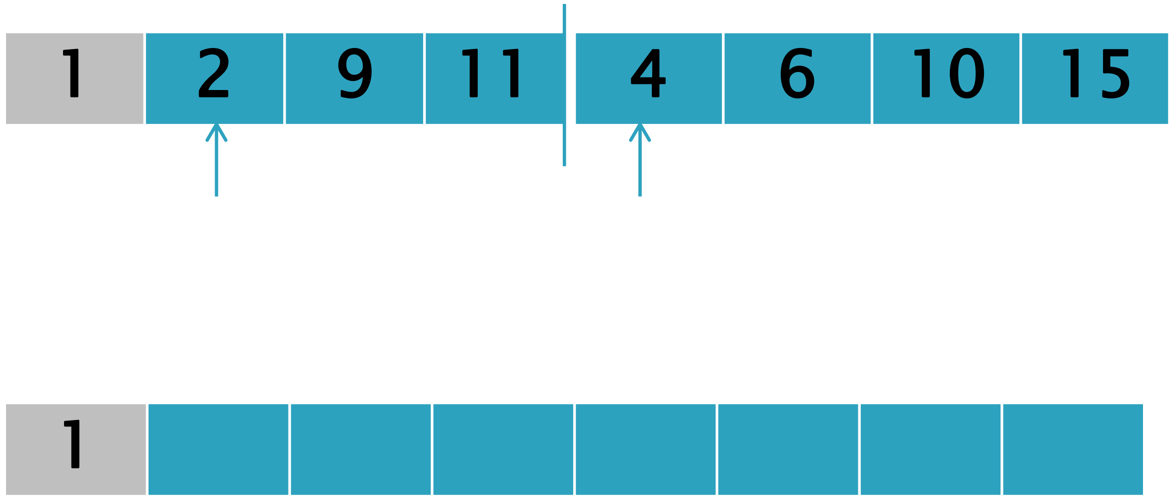
  - $T(n) = T(n/2)+c$
    $= \ldots = T(n/2^k)+kc$

  - $k = \log_2 n$

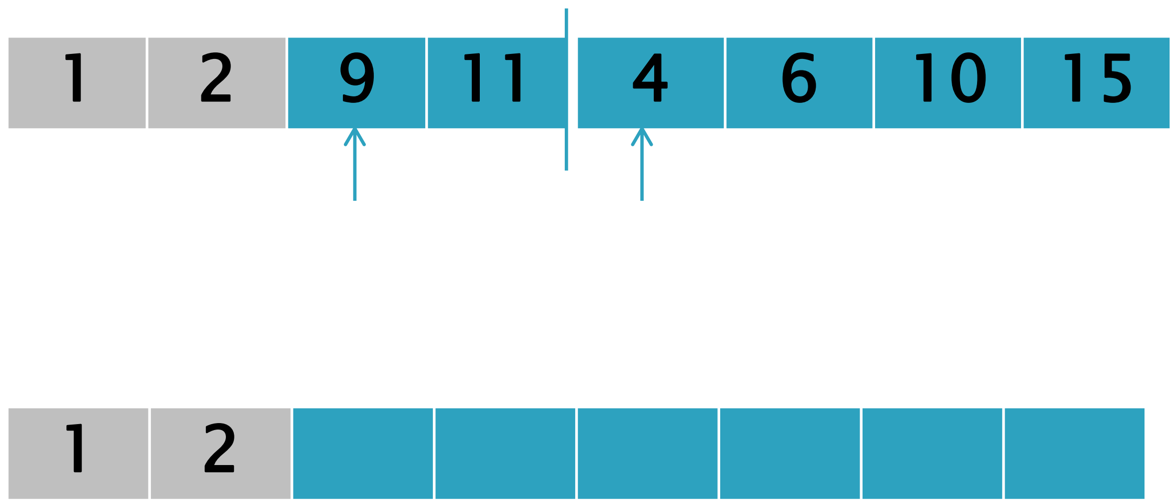
Sortare prin interclasare

- **Idee:**
  - împărţim vectorul în doi subvectori
  - ordonăm crescător fiecare subvector
  - asamblăm rezultatele prin ***interclasare***

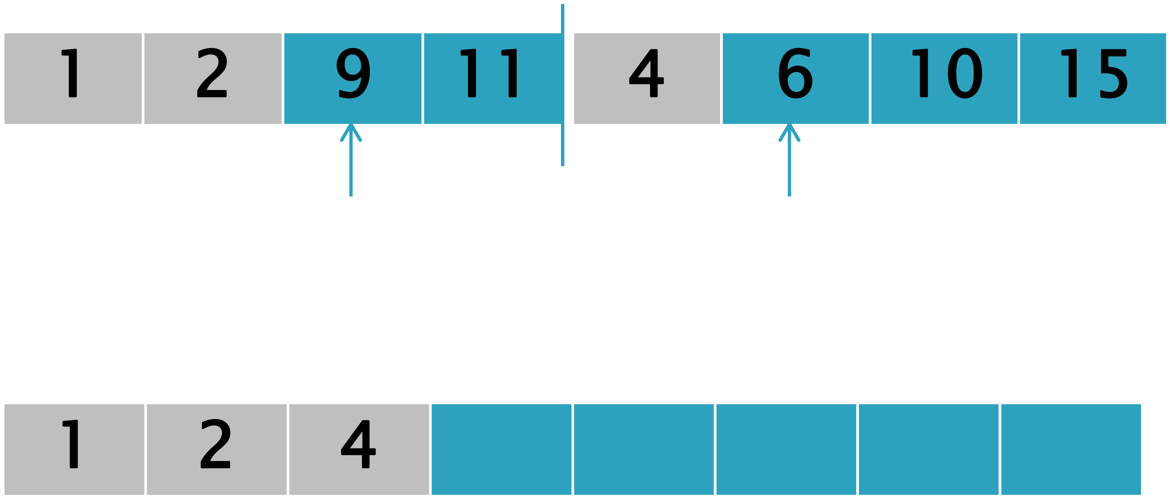

1
2
9
11
4
6
10
15

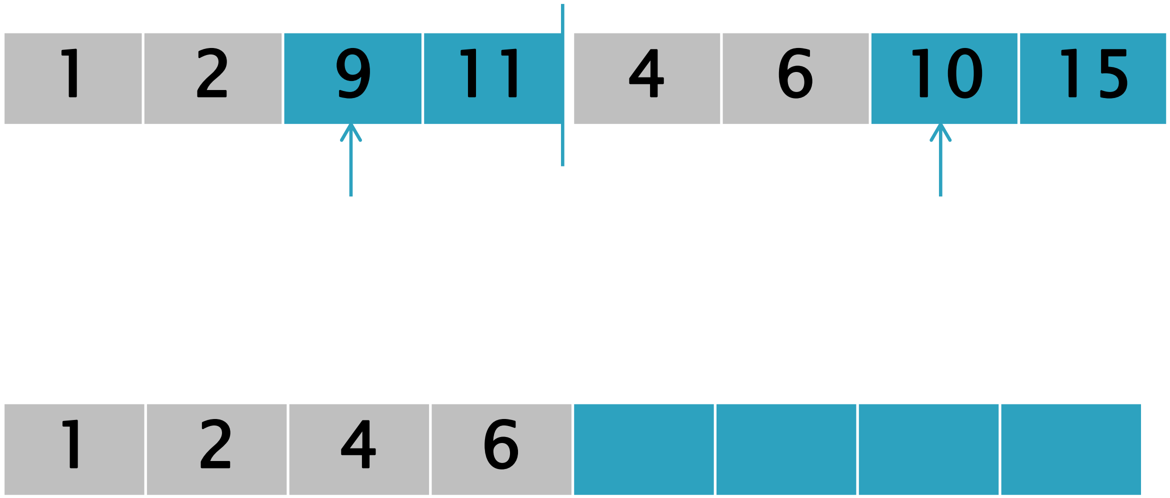


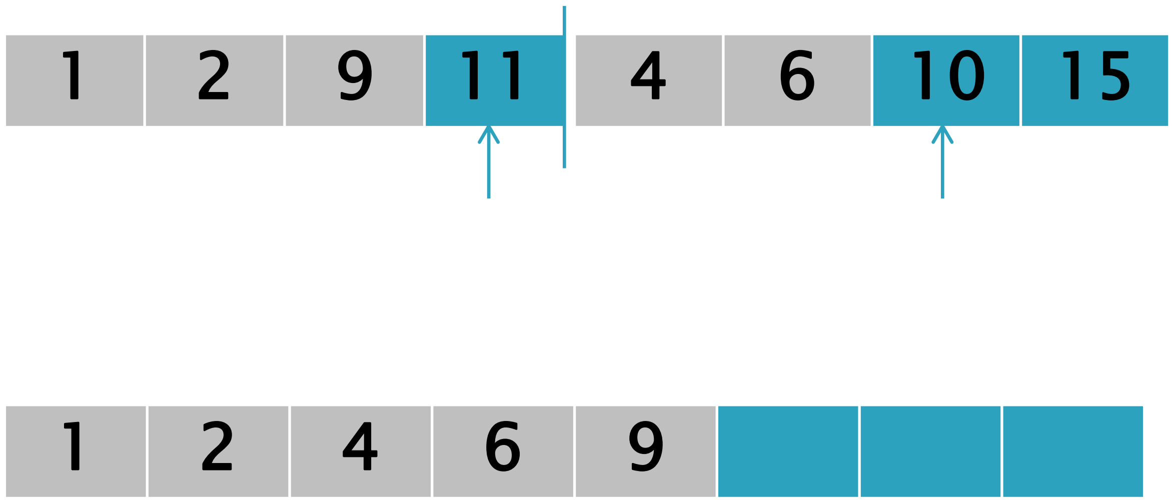
1 2 9 11 4 6 10 15
1

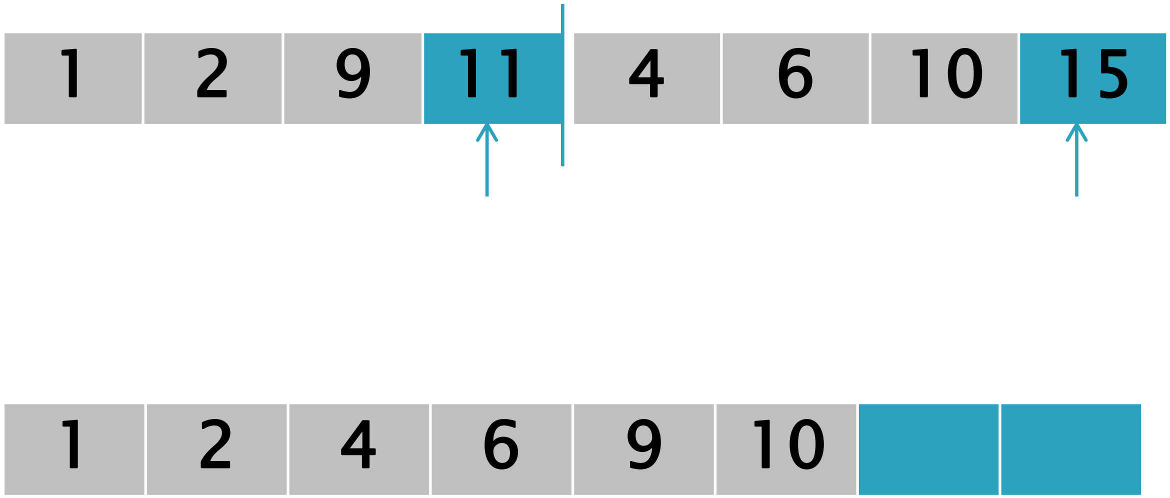


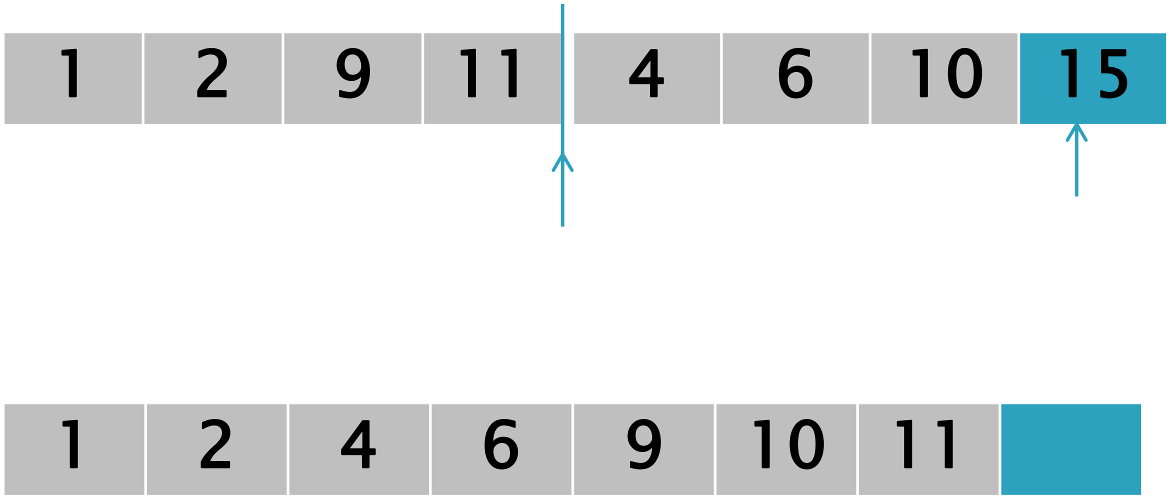
1 2 9 11 4 6 10 15
1 2


1
2
9
11
4
6
10
15
1
2
4


1
2
9
11
4
6
10
15
1
2
4
6


1
2
9
11
4
6
10
15
1
2
4
6
9


1 2 9 11 4 6 10 15
1 2 4 6 9 10


1 2 9 11 4 6 10 15
1 2 4 6 9 10 11

1	2	9	11	4	6	10	15

1	2	4	6	9	10	11	15

Sortare prin interclasare

```
void sortInter(int p,int u)
    if (p==u) {}
    else {
        int m = (p+u)/2;
        sortInter(p,m);
        sortInter(m+1,u);
        interclaseaza(p,m,u);
    }
}
```

Sortare prin interclasare

- **Complexitate:** O(n log n)

Sortare prin interclasare

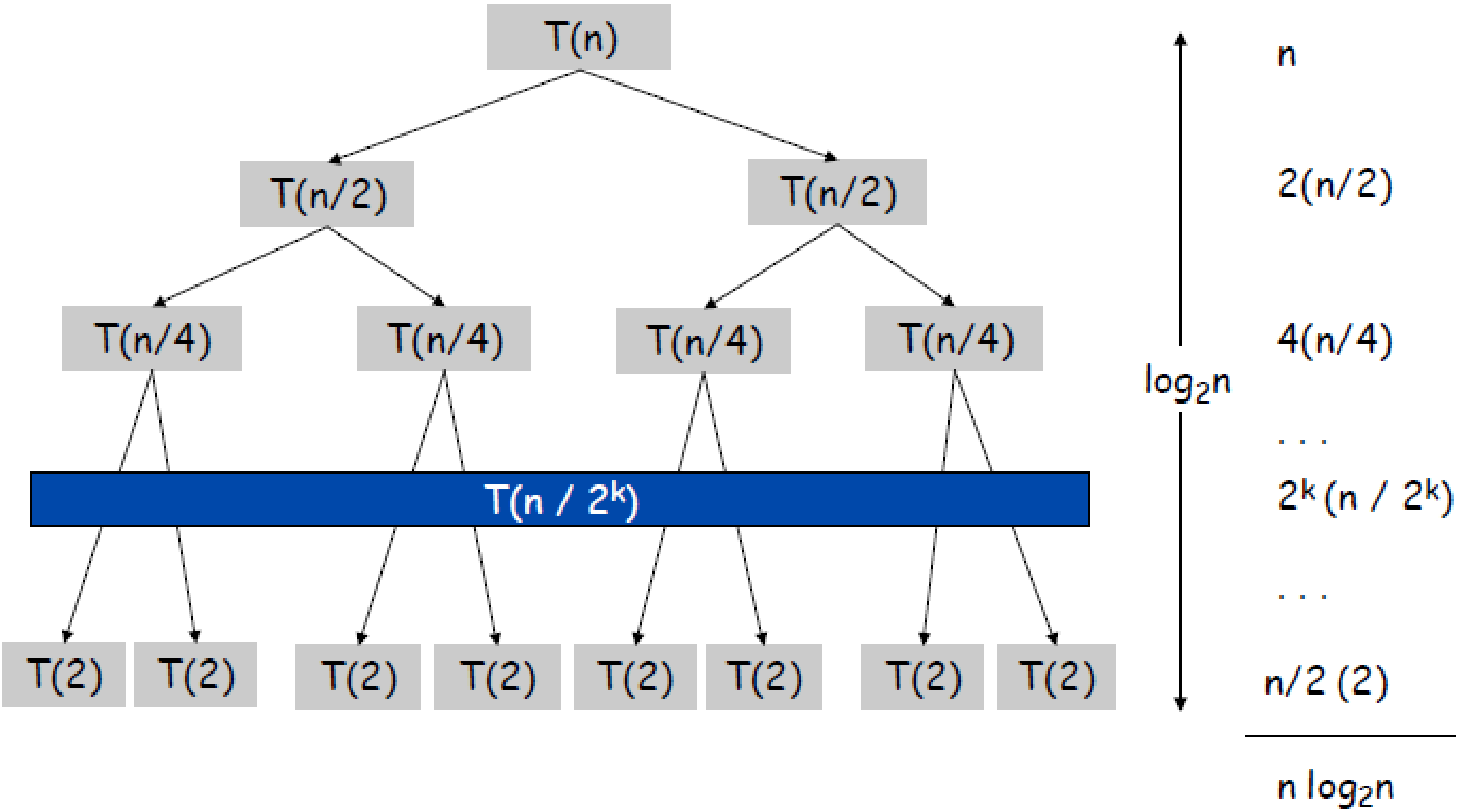
- **Complexitate:** $O(n \log n)$

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

- Pentru $n=2^k$

  $T(n) = 2T(n/2)+cn$, pentru $n>1$

Sortare prin interclasare



http://www.cs.princeton.edu/~wayne/kleinberg-tardos/

Sortare prin interclasare

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

$$=$$

Sortare prin interclasare

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

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

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

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

Sortare prin interclasare

Problemă

Se consideră un vector cu n elemente. Să se 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`

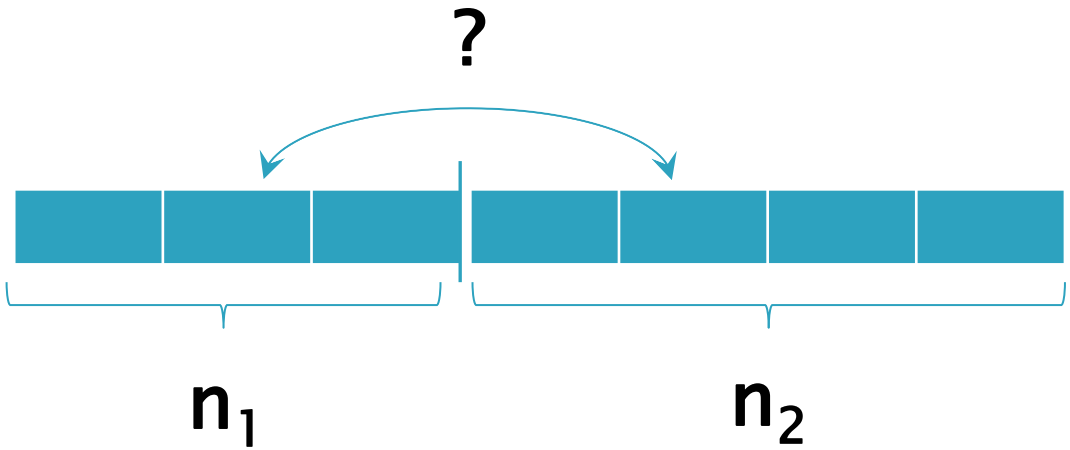
`(11,9), (11,4), (9,4), (11,6), (9,6)`

Numărare inversiuni

- Algoritm $O(n^2)$ – evident

Numărare inversiuni

- Algoritm Divide et Impera



$n_1+n_2+?$

- **Are loc relaţia**

  numărul de inversiuni din vector $=$

  nr. de inversiuni din subvectorul stâng

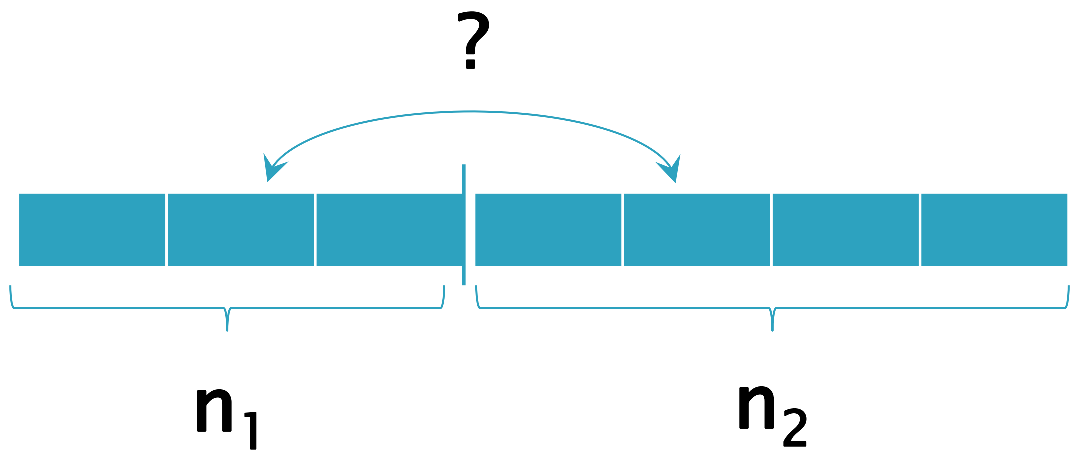
  $+$

  nr. de inversiuni din subvectorul drept

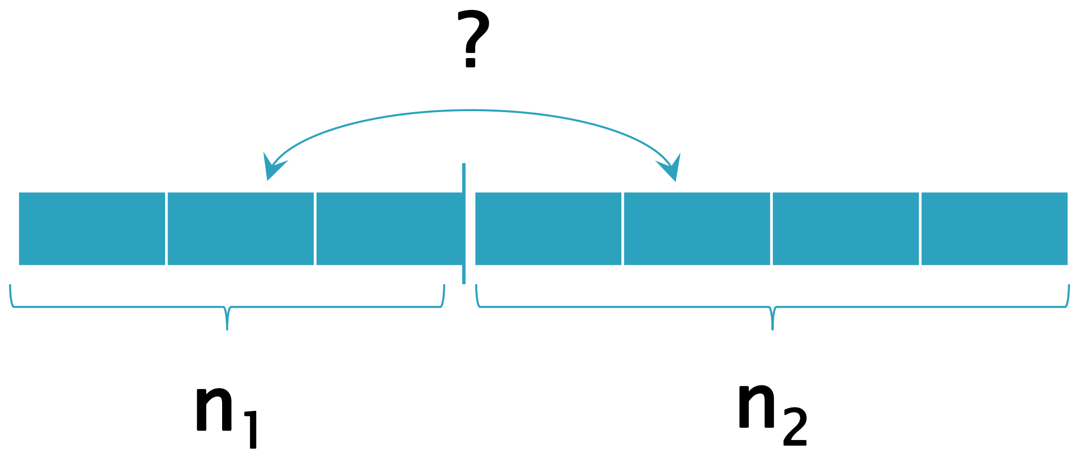
  $+$

  nr de inversiuni (i, j) cu

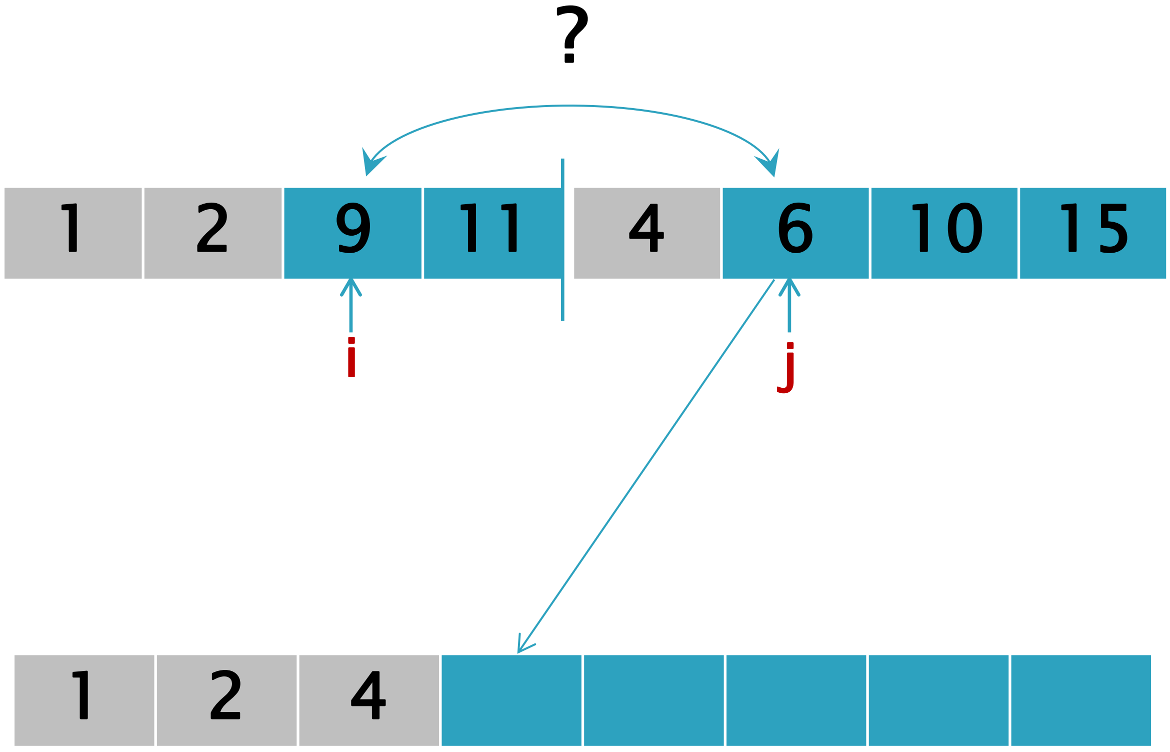
  - $i \leq m$ indice în subvectorul stâng și
  - $j > m$ indice în subvectorul drept

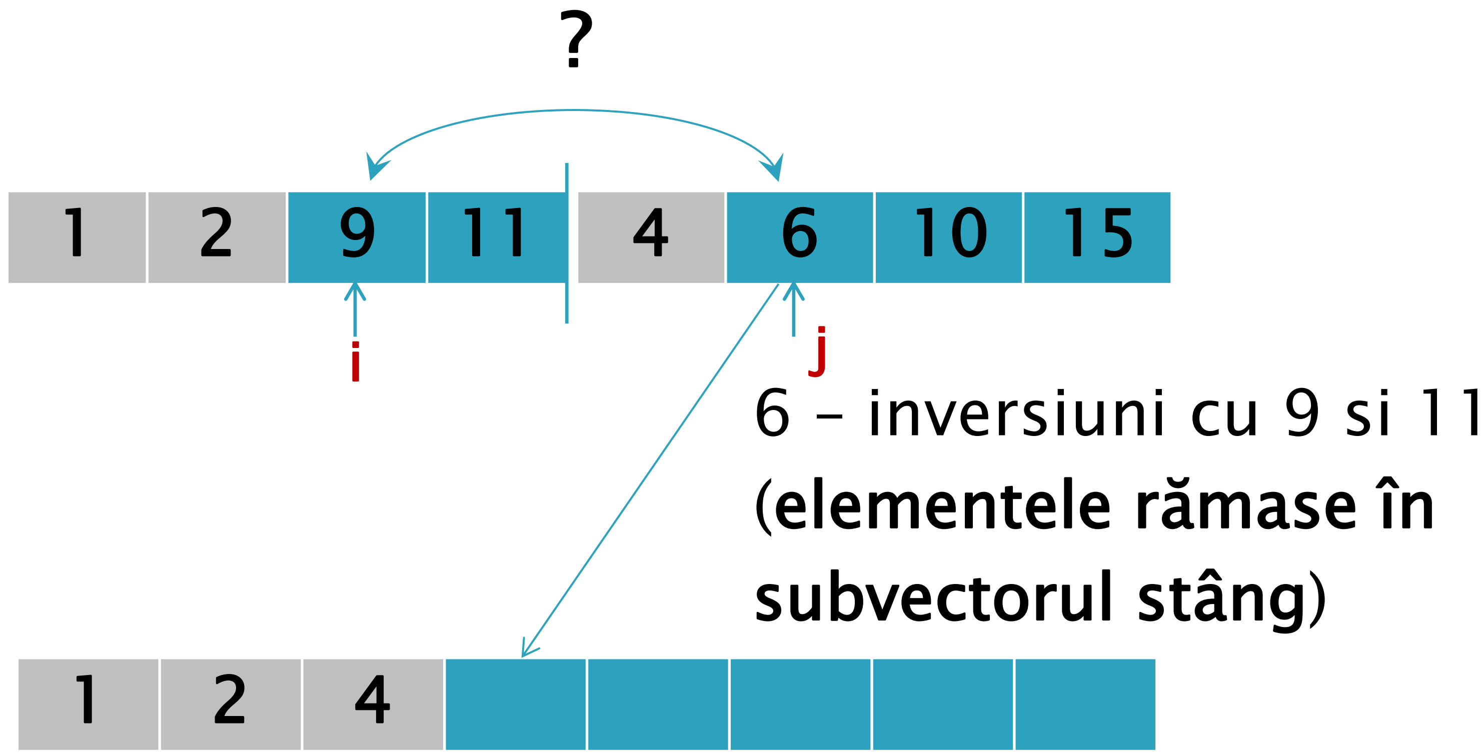


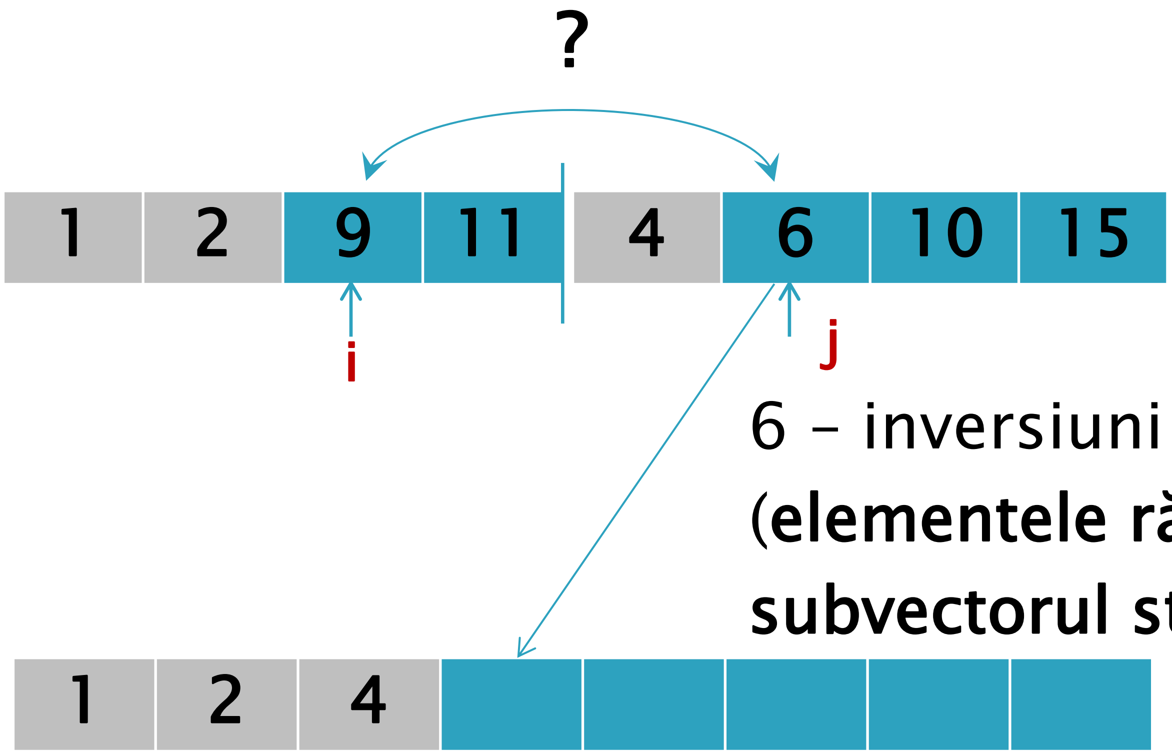
$$n_1 + n_2 + ?$$



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


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

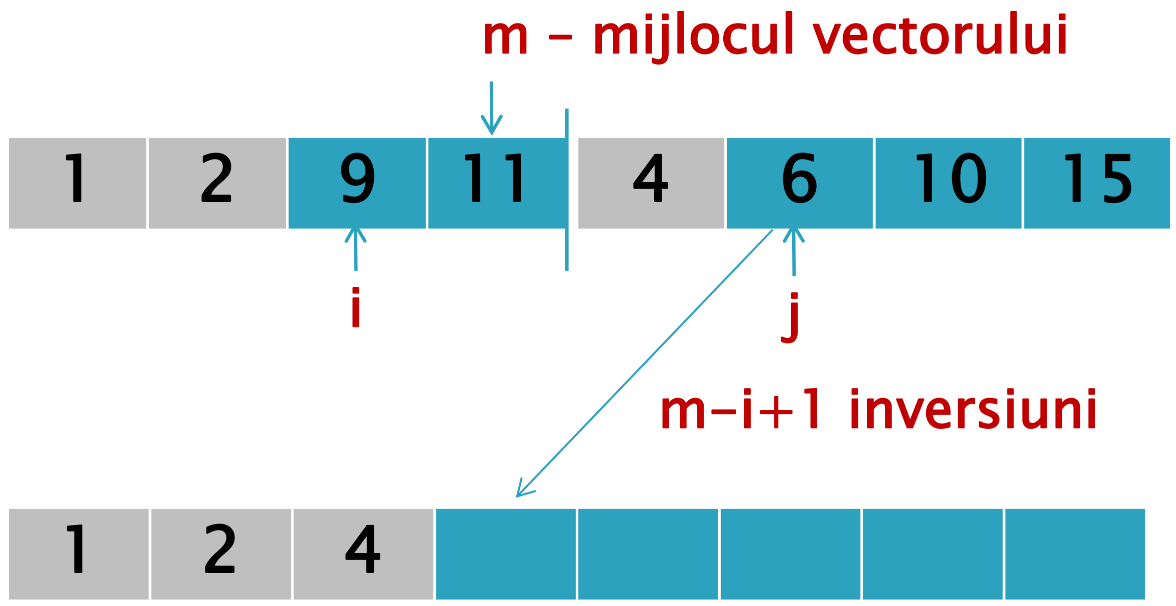

?
1
2
9
11
4
6
10
15
i
j
6 – inversiuni cu 9 si 11
(**elementele rămase în subvectorul stâng**)
1
2
4



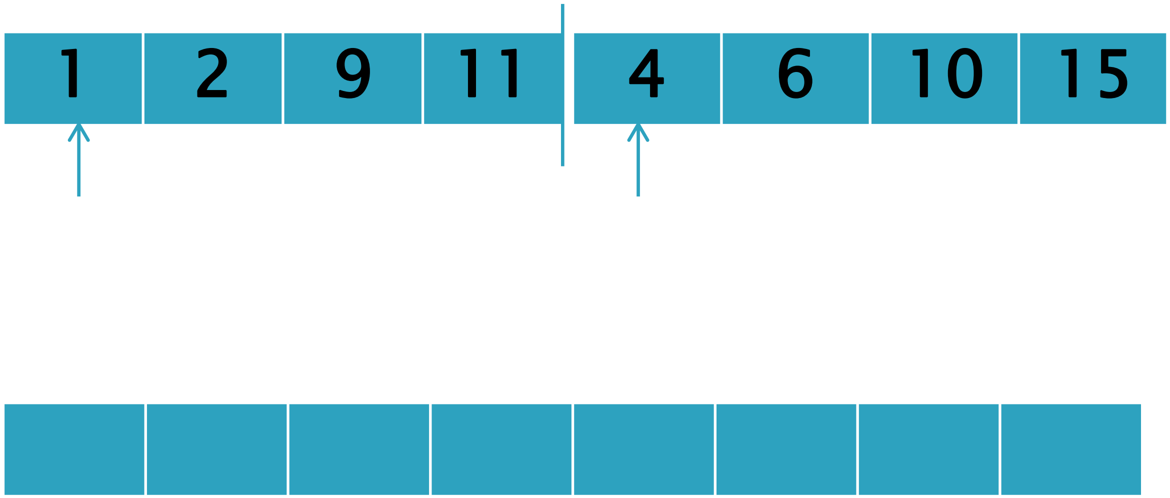
6 – inversiuni cu 9 si 11 (**elementele rămase în subvectorul stâng**)

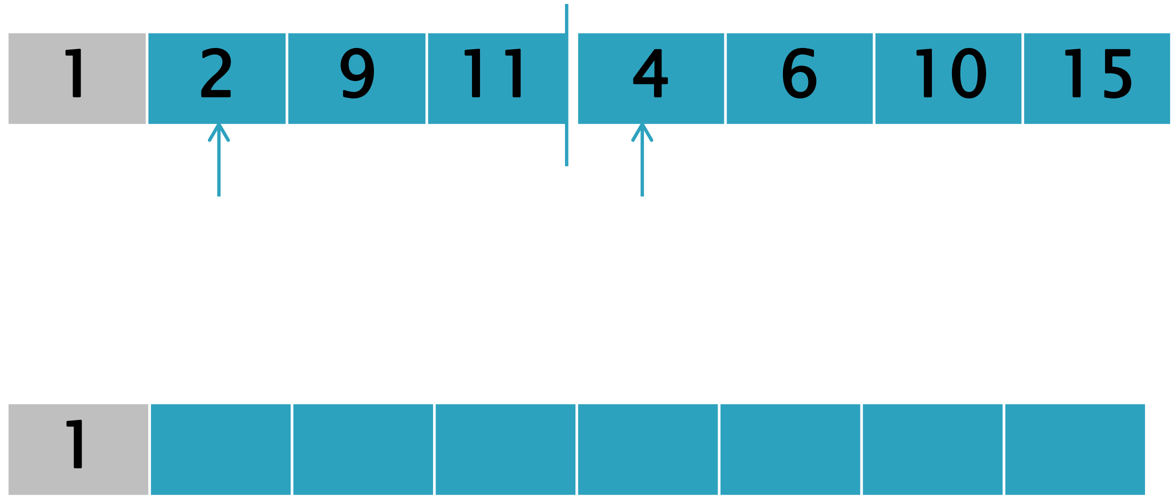
Când a[j] cu j > m este adăugat în vectorul rezultat, el este **mai mic (doar)** decât toate elementele din subvectorul stâng **neadăugate** încă în vectorul rezultat

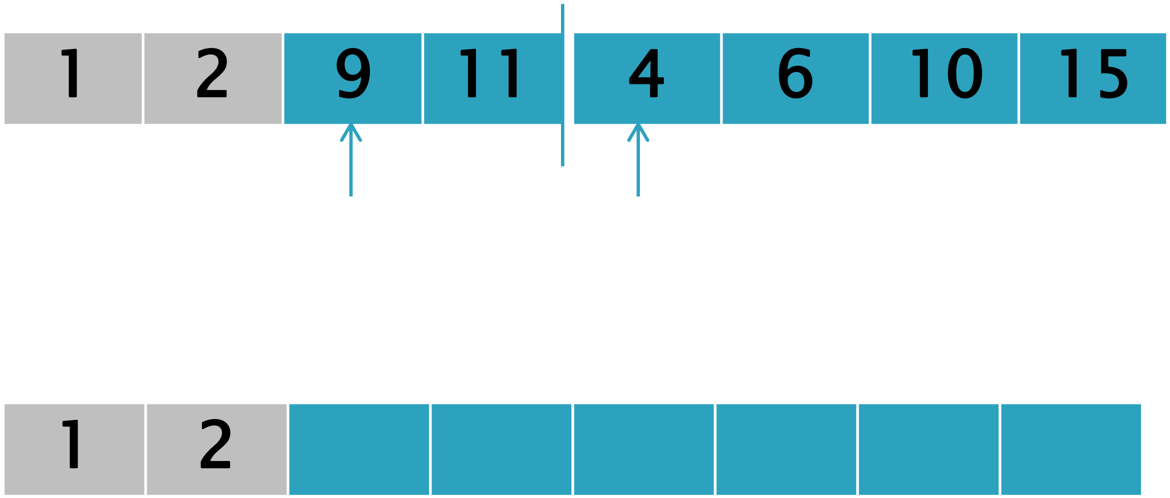
- a[j] determină m – i + 1 inversiuni

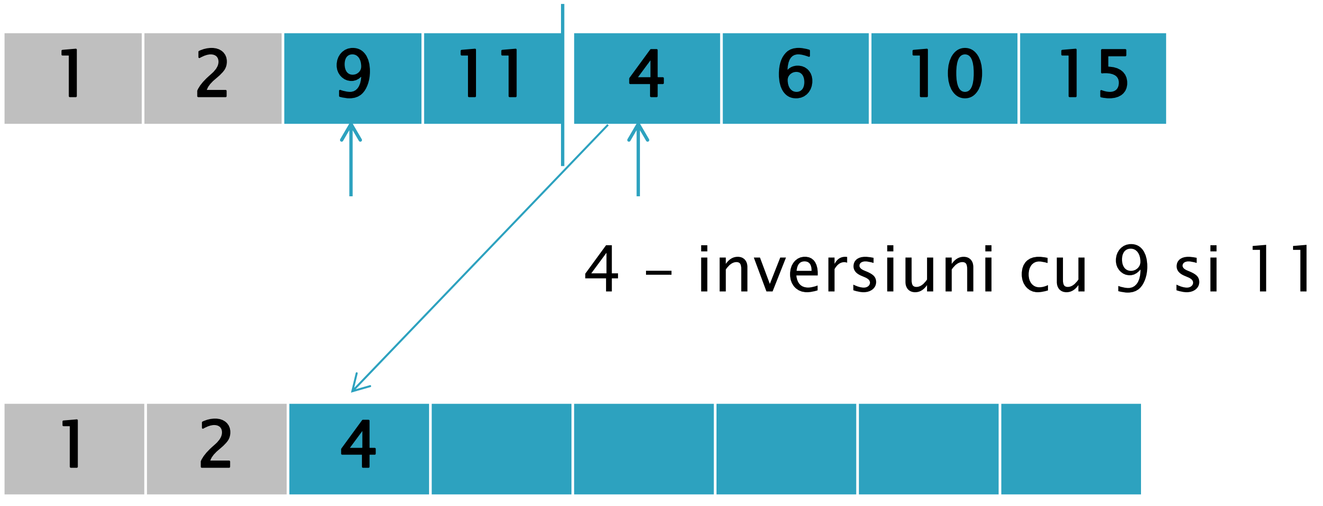


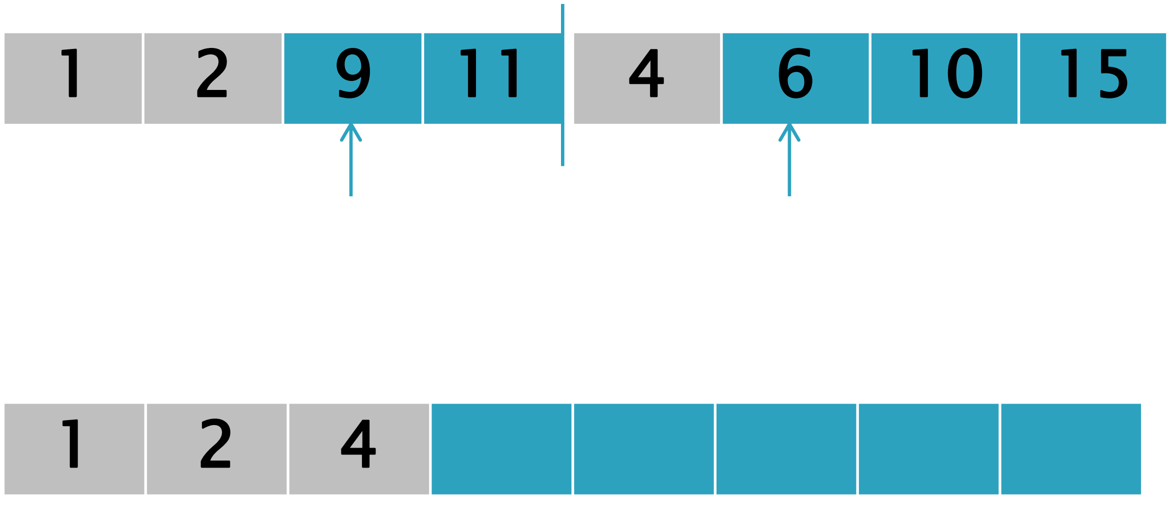
Exemplu – numărarea inversiunilor la interclasare

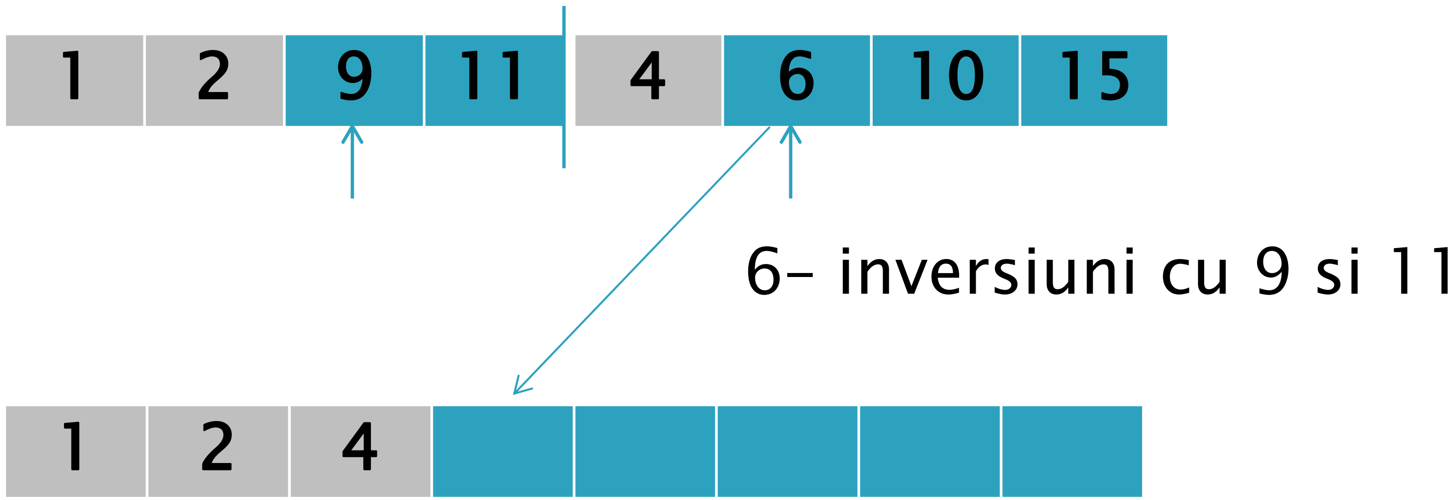

1
2
9
11
4
6
10
15

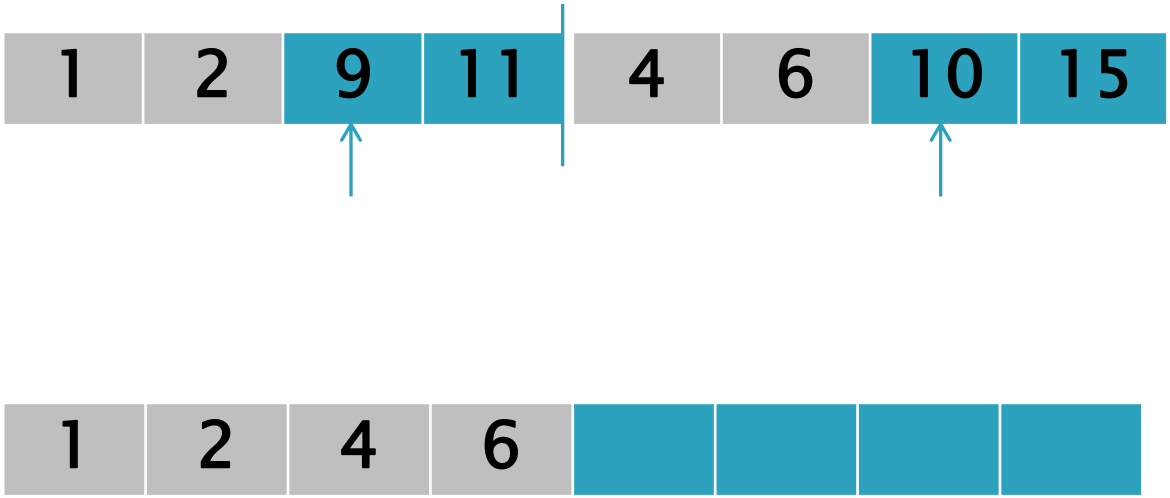

1
2
9
11
4
6
10
15
1

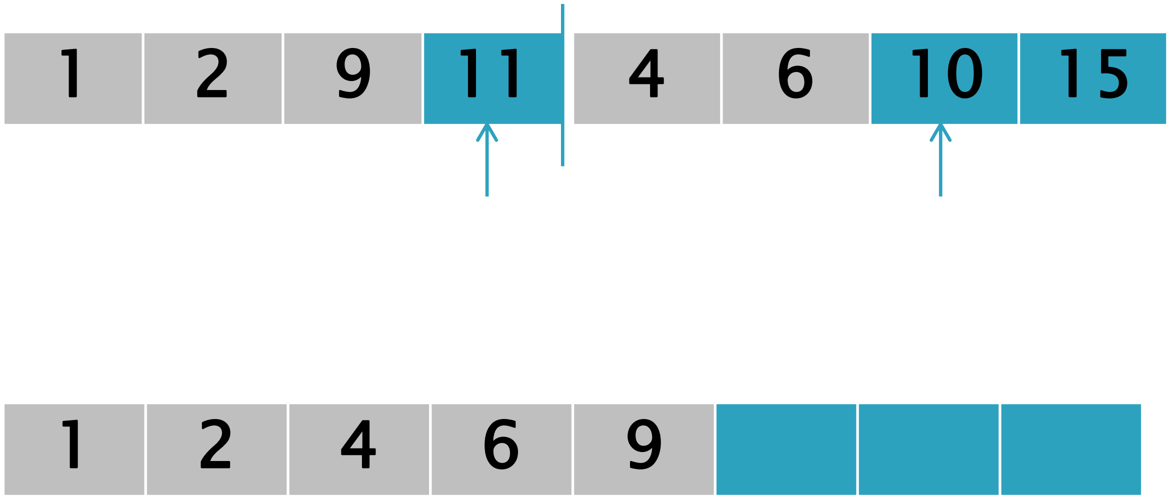

1 2 9 11 4 6 10 15
1 2


1
2
9
11
4
6
10
15
4 – inversiuni cu 9 si 11
1
2
4

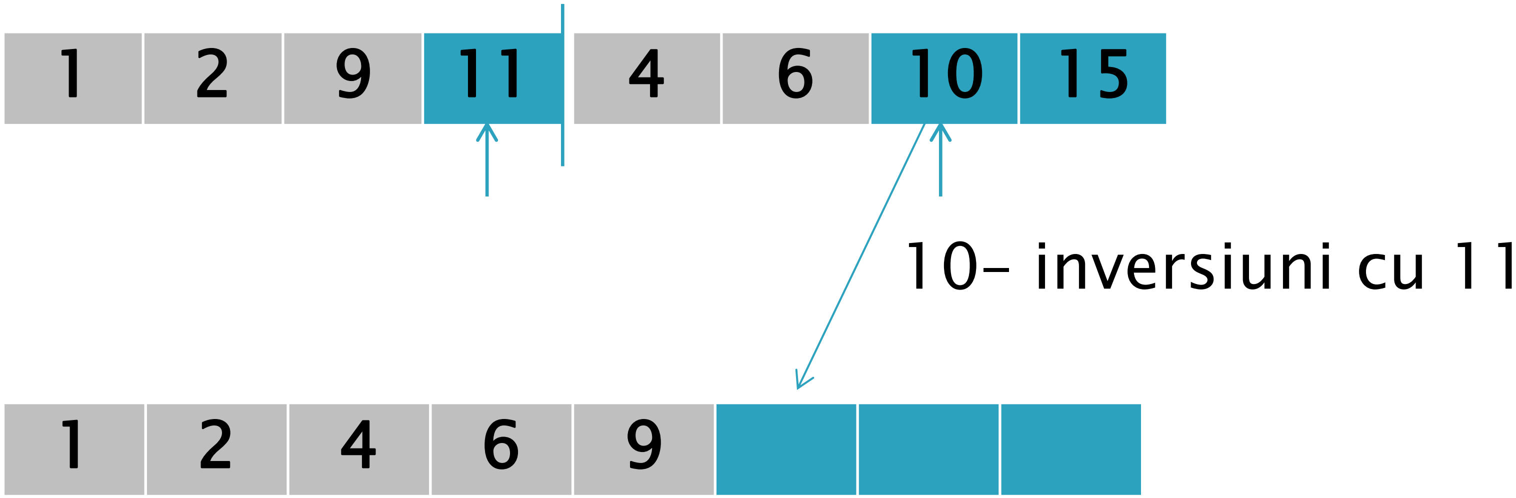

1
2
9
11
4
6
10
15
1
2
4

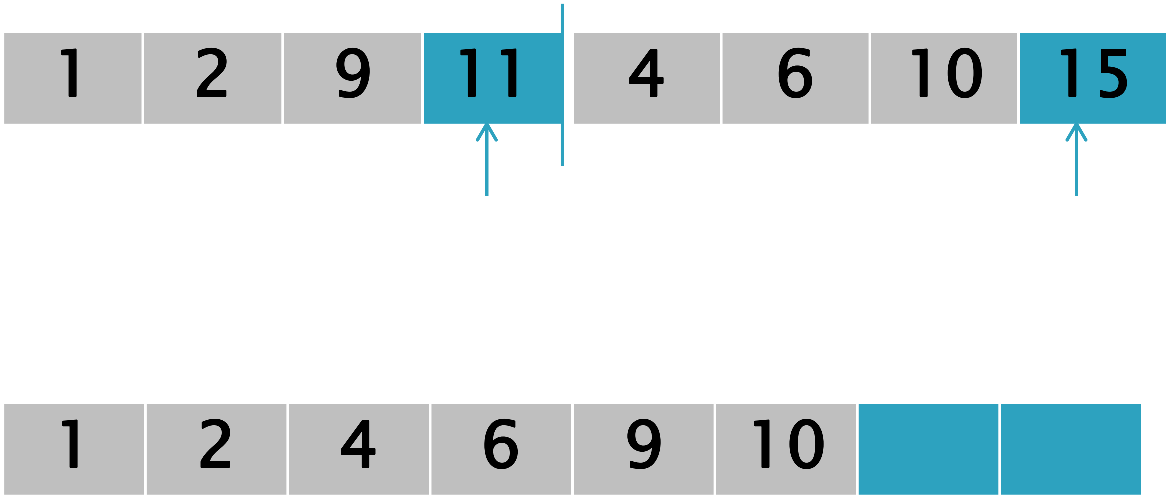

1 2 9 11 4 6 10 15
6- inversiuni cu 9 si 11
1 2 4


1 2 9 11 4 6 10 15
1 2 4 6



1 2 9 11 4 6 10 15
1 2 4 6 9


1 2 9 11 4 6 10 15
10- inversiuni cu 11
1 2 4 6 9


1 2 9 11 4 6 10 15
1 2 4 6 9 10

1	2	9	11	4	6	10	15

1	2	4	6	9	10	11	

1	2	9	11	4	6	10	15

15- nicio inversiune

1	2	4	6	9	10	11	

1	2	9	11	4	6	10	15

1	2	4	6	9	10	11	15

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;













  }
```

```
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++;}









  }
```

```
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);
        }















}
```

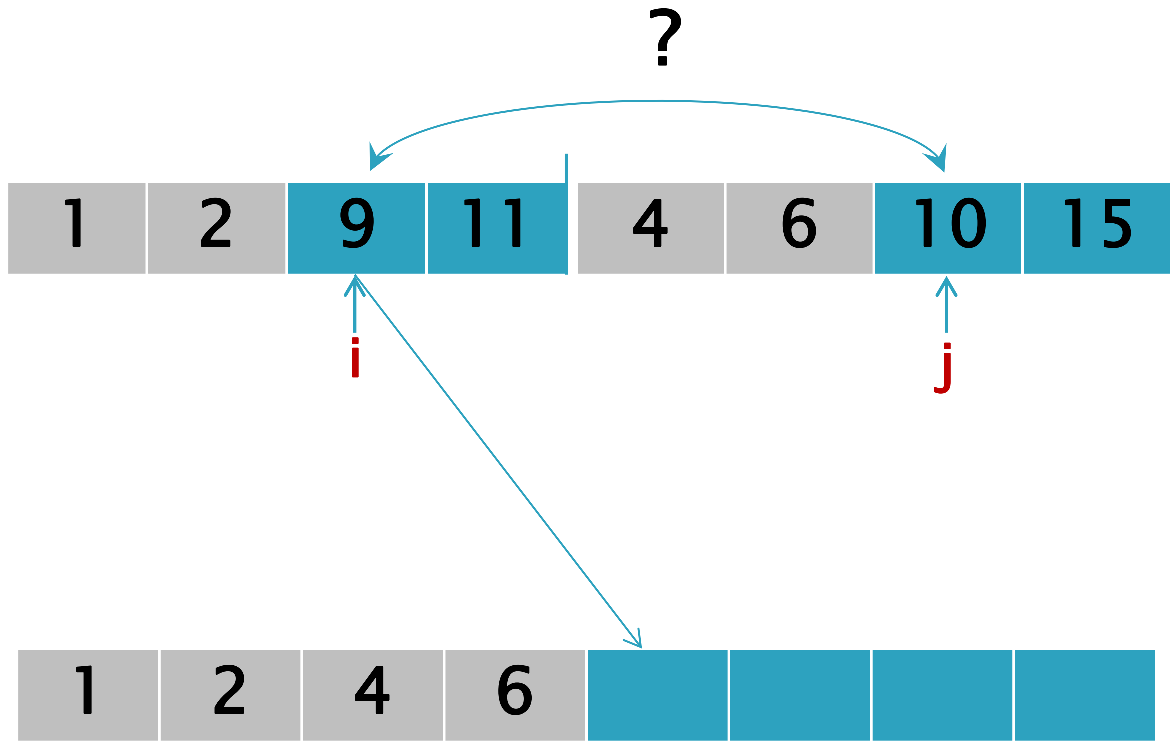
```
int interclasare(int p,int m,int u){
      int b[]=new int[u-p+1];
      int i=p,j=m+1, k=0,nr=0;
      while ((i<=m)&&(j<=u)){
             if (a[i]<=a[j]){ b[k]=a[i]; i++;}
             else{ b[k]=a[j]; j++;
                    nr = nr + (m-i+1);
             }
             k++;
      }



  }
```

```
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++; }


  }
```

```
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];

}
```

```
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;
  }
```

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


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

?

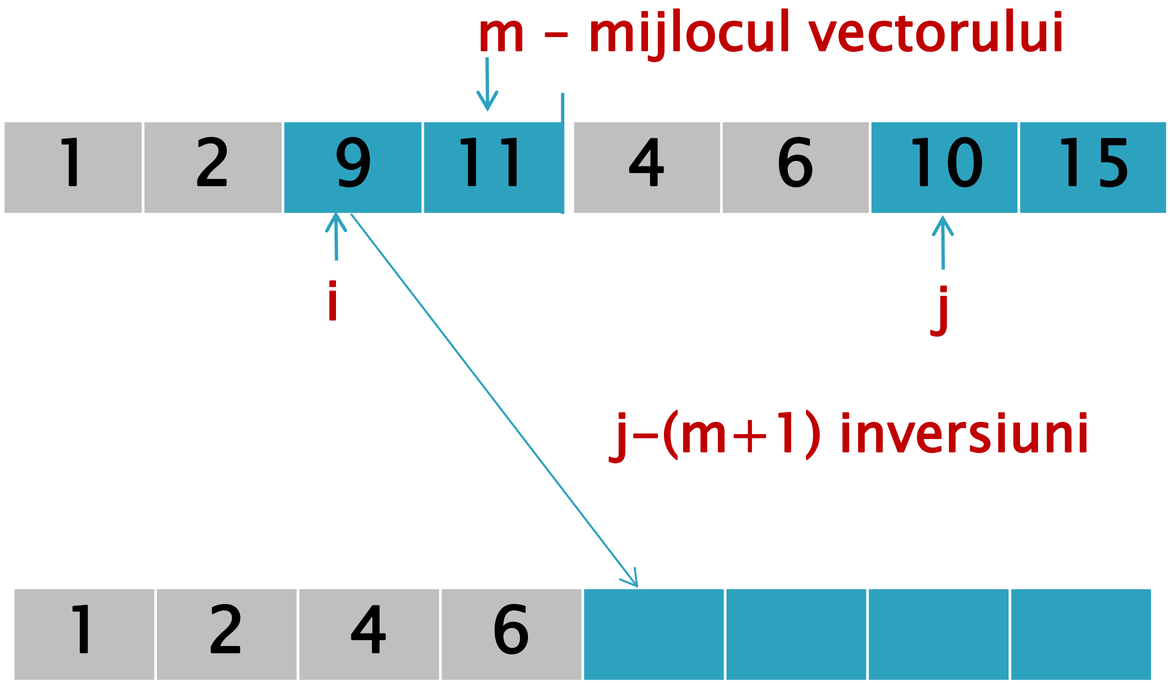
1	2	9	11	4	6	10	15

i

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

1	2	4	6				

În acest caz a[i] este mai mare (doar) decât toate elementele din subvectorul drept adăugate deja la vectorul rezultat.


m – mijlocul vectorului
1
2
9
11
4
6
10
15
i
j
j–(m+1) inversiuni
1
2
4
6

```
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;
}
```

Quicksort

- **Idee:**
  - poziționăm primul element al secvenței 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

Quicksort

```
void quicksort(int p,int u)
    if (p==u) {}
    else {
        int m = poz(p,u);
        quicksort(p,m-1);
        quicksort(m+1,u);
    }
}
```

```
int poz(int p,int u){
    int i=p,j=u, depli=0,deplj=-1;
```

```
int poz(int p,int u){
    int i=p,j=u, depli=0,deplj=-1;
    while(i<j){
       if(a[i]>a[j]){
          int aux=a[i];a[i]=a[j];a[j]=aux;



       }

    }

 }
```

```
int poz(int p,int u){
   int i=p,j=u, depli=0,deplj=-1;
   while(i<j){
      if(a[i]>a[j]){
         int aux=a[i];a[i]=a[j];a[j]=aux;
         aux=depli;
         depli=-deplj;
         deplj=-aux;
      }

   }

}
```

```
int poz(int p,int u){
   int i=p,j=u, depli=0,deplj=-1;
   while(i<j){
     if(a[i]>a[j]){
        int aux=a[i];a[i]=a[j];a[j]=aux;
        aux=depli;
        depli=-deplj;
        deplj=-aux;
     }
     i+=depli;j+=deplj;
   }
   return i;
 }
```

Exemplu – poziţionare pivot

10	2	6	12	4	11

10	2	6	12	4	11

10	2	6	12	4	11

10	2	6	12	4	11

10	2	6	12	4	11

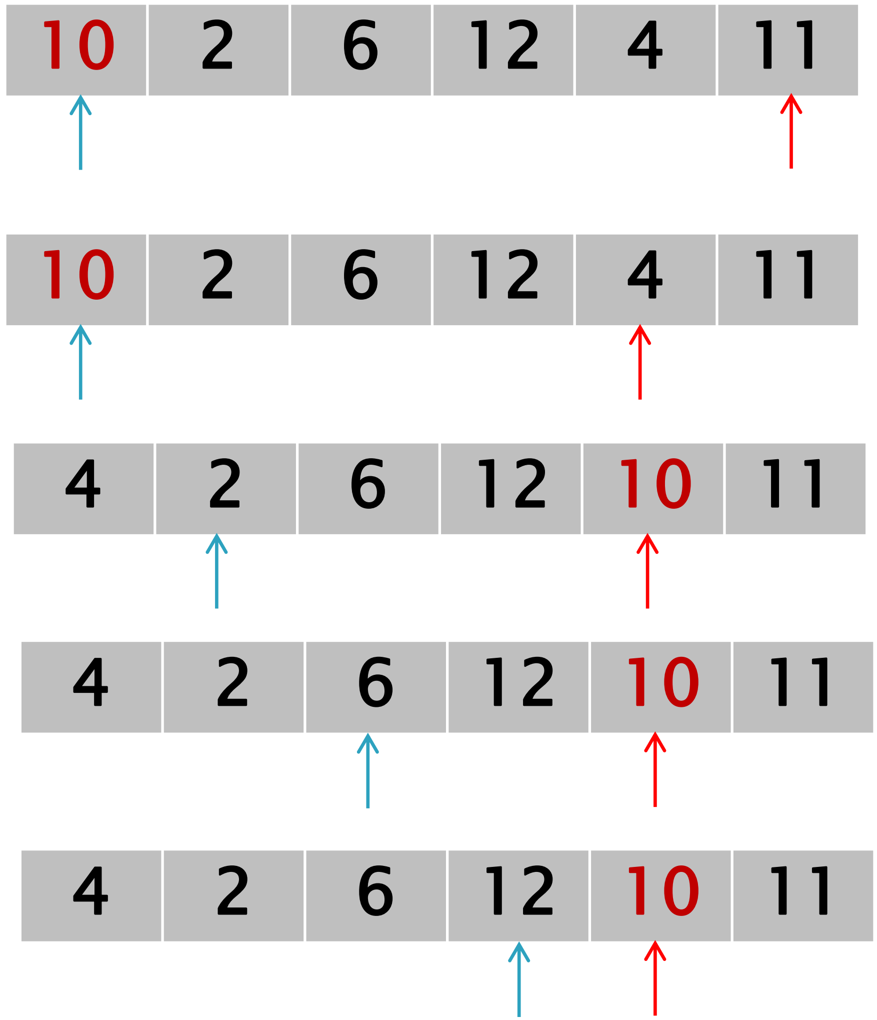
4	2	6	12	10	11

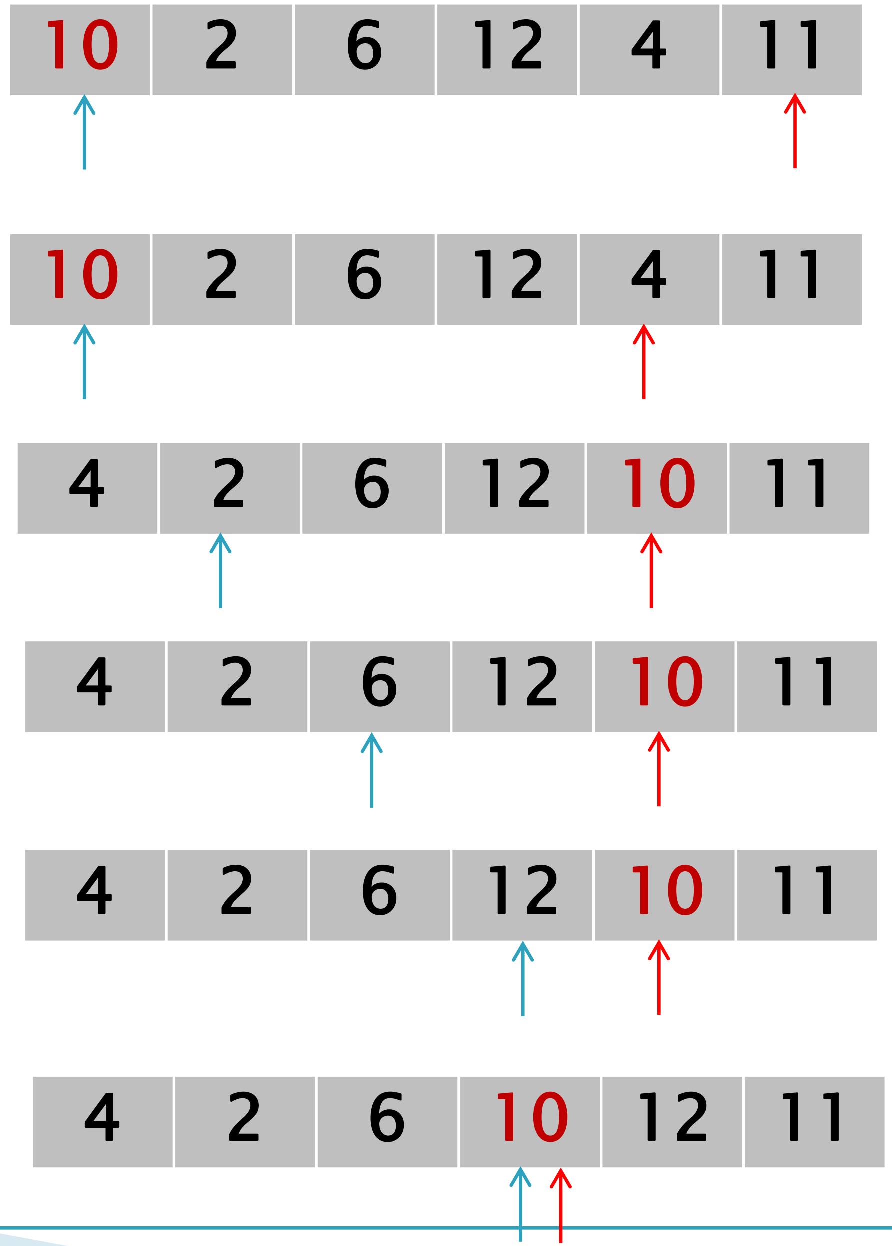
10	2	6	12	4	11

10	2	6	12	4	11

4	2	6	12	10	11

4	2	6	12	10	11


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


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

Quicksort

- **Complexitate:**
  - **Defavorabil**: $O(n^2)$

Quicksort

- **Complexitate:**
  - **Defavorabil**: $O(n^2)$
  - **Mediu**: $O(n \log n)$

Quicksort

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

Quicksort

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

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

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

Quicksort

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

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

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

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

Quicksort

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

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

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

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

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

Quicksort

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

Quicksort

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

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

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

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

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

Quicksort

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

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

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

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

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

$$+$$

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

Quicksort

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

$$\leq 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots\ldots + \frac{1}{3}\right)$$

sume Riemann inferioare pentru funcţia f(x)=1 / x

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

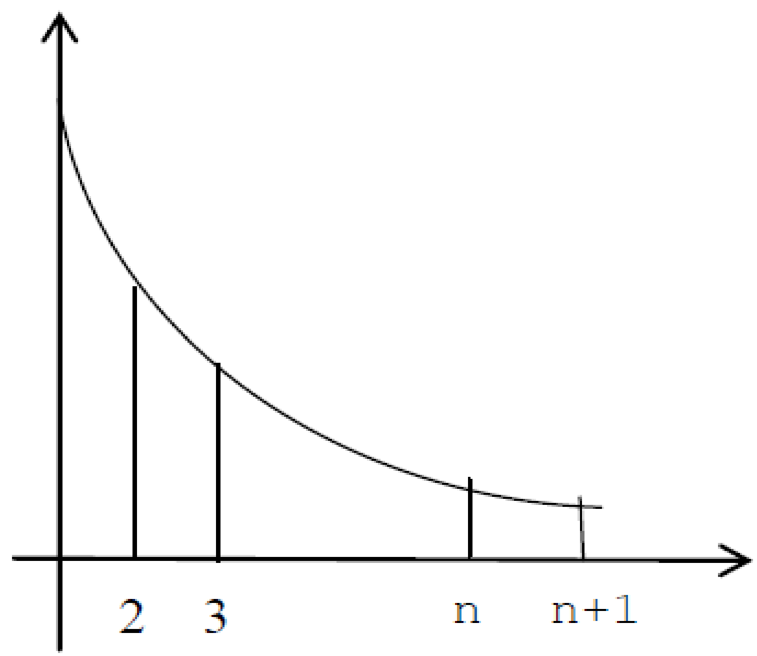
Quicksort

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

$$\leq 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots\ldots + \frac{1}{3}\right)$$

sume Riemann inferioare pentru funcţia f(x)=1/ x

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



Quicksort

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

$$\leq 2\left(\frac{1}{n+1}+\frac{1}{n}+\ldots\ldots+\frac{1}{3}\right)$$

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

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

$$\frac{T(n)}{n+1} \leq 2\int_{2}^{n+1}\frac{1}{x}dx = 2\ln x\,\Big|_{2}^{n+1} \leq 2\cdot\ln(n+1)$$

Quicksort

- **Variantă** – pivot aleator

```
int pozRand(int p,int u){
    r ← random(p,u)
    a[r] ↔ a[p]
    return poz(p,u)
}
```

Statistici de ordine

Problemă

Dat un vector a de n numere şi un indice k, $1 \le k \le n$, 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

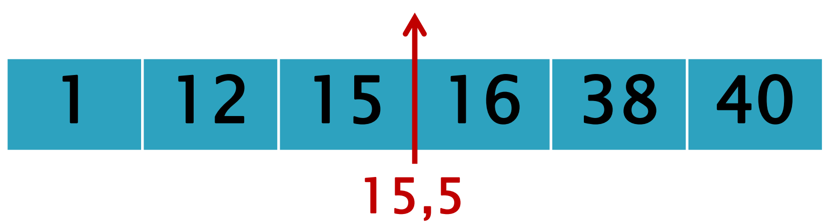
Statistici de ordine

- **Mediana** = punctul de la jumătatea drumului unei mulțimi
  = 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.

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



Statistici de ordine

Idee

Al k-lea minim – folosim poziţionarea de la quicksort.

Statistici de ordine

Fie m poziţia pivotului

-
-
-

Statistici de ordine

Fie m poziţia pivotului

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

Statistici de ordine

Fie m poziţia pivotului

- Dacă m = k, pivotul este al k-lea minim
- Dacă m > k, al k-lea minim este în stânga pivotului
-

Statistici de ordine

Fie m poziţia pivotului

- Dacă $m = k$, pivotul este al k-lea minim
- Dacă $m > k$, al k-lea minim este în stânga pivotului
- Dacă $m < k$, al k-lea minim este în dreapta pivotului

k = 2

10	2	6	12	4	11

poziţionare pivot

4	2	6	10	12	11

m = 4 > k $\Rightarrow$ stânga

k = 2

10	2	6	12	4	11

poziţionare pivot

4	2	6	10	12	11

$m = 4 > k \longrightarrow$ stânga

4	2	6

poziţionare pivot

2	4	6

$m = 2 = k \longrightarrow$ stop;

pivotul este al k-lea minim

```java
//pentru numerotare de la 0
int selKMin(int p, int u){
    int m = pozRand(p,u);
    if(m == k-1) return a[m];
    if(m < k-1) return selKMin(m+1,u);
    return selKMin(p,m);
}

int selKMin(){
    return selKMin(0,n-1);
}
```

- AlKMinim.java

Complexitate

- **Timpul mediu**

$$\begin{cases} T(n) \le n-1+\dfrac{1}{n}\displaystyle\sum_{k=1}^{n} T(\max\{k-1,n-k\}) \\ \qquad \le n-1+\dfrac{\mathbf{2}}{n}\displaystyle\sum_{k=\left\lceil \frac{\boldsymbol{n}}{\mathbf{2}} \right\rceil}^{n} T(k) \end{cases}$$

Complexitate

- **Timpul mediu**

$$\begin{cases} T(n) \leq n-1+\dfrac{1}{n}\displaystyle\sum_{k=1}^{n} T(\max\{k-1, n-k\}) \\ \qquad\quad \leq n-1+\dfrac{\mathbf{2}}{n}\displaystyle\sum_{k=\left\lceil \frac{\boldsymbol{n}}{\mathbf{2}} \right\rceil}^{n} T(k) \end{cases}$$

$$T(n) \leq cn$$

Statistici de ordine

Problema elementului majoritar

Se dă un şir de *n* numere naturale. Se cere determinarea unui element care apare de cel puţin *[n/2]+1* ori în şir, dacă există.

Acest element se numeşte element majoritar

http://infoarena.ro/problema-majoritatii-votului

Mediana a doi vectori sortați

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.

Mediana a doi vectori sortați

Exemplu: $n = 5$

1 12 15 16 38

2 13 17 30 45

Mediana a doi vectori sortați

Exemplu: n = 5

1 12 15 16 38

2 13 17 30 45

⇓

1 2 12 13 15 16 17 30 38 45

Mediana (15+16)/2 = 15,5

Mediana a doi vectori sortați

- **Algoritm O(n)** – interclasăm vectorii şi apoi aflăm mediana în timp constant (din elementele de la mijlocul vectorului, conform definiţiei)

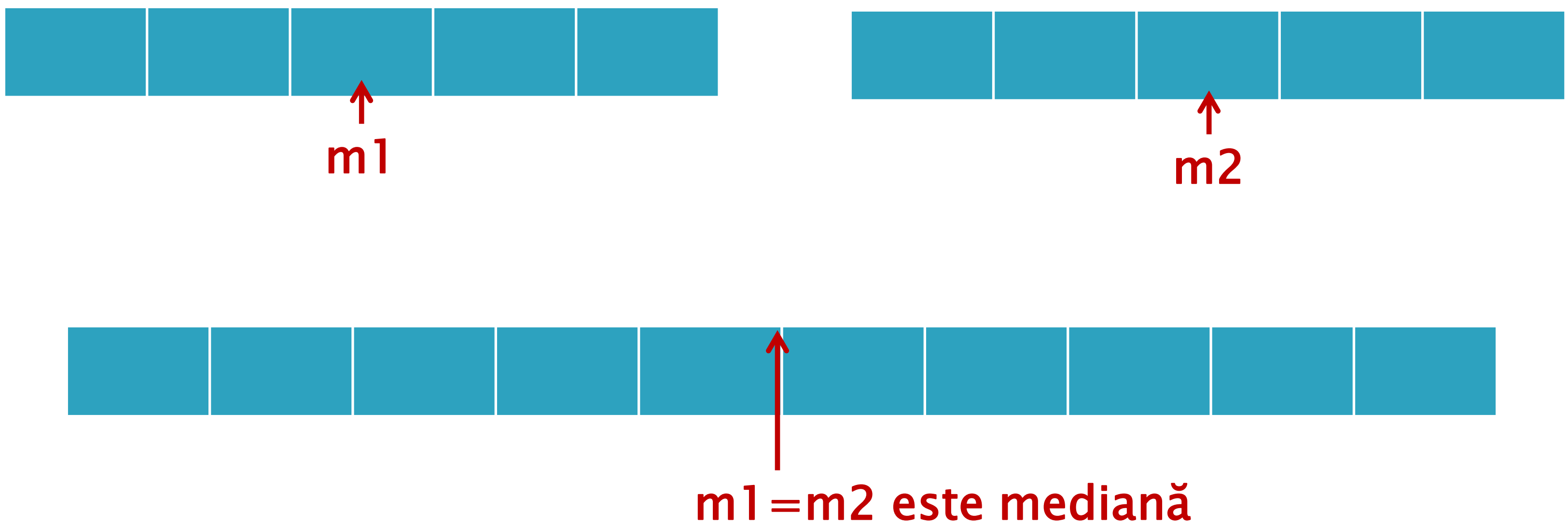
Mediana a doi vectori sortați

- **Algoritm O(log n)**

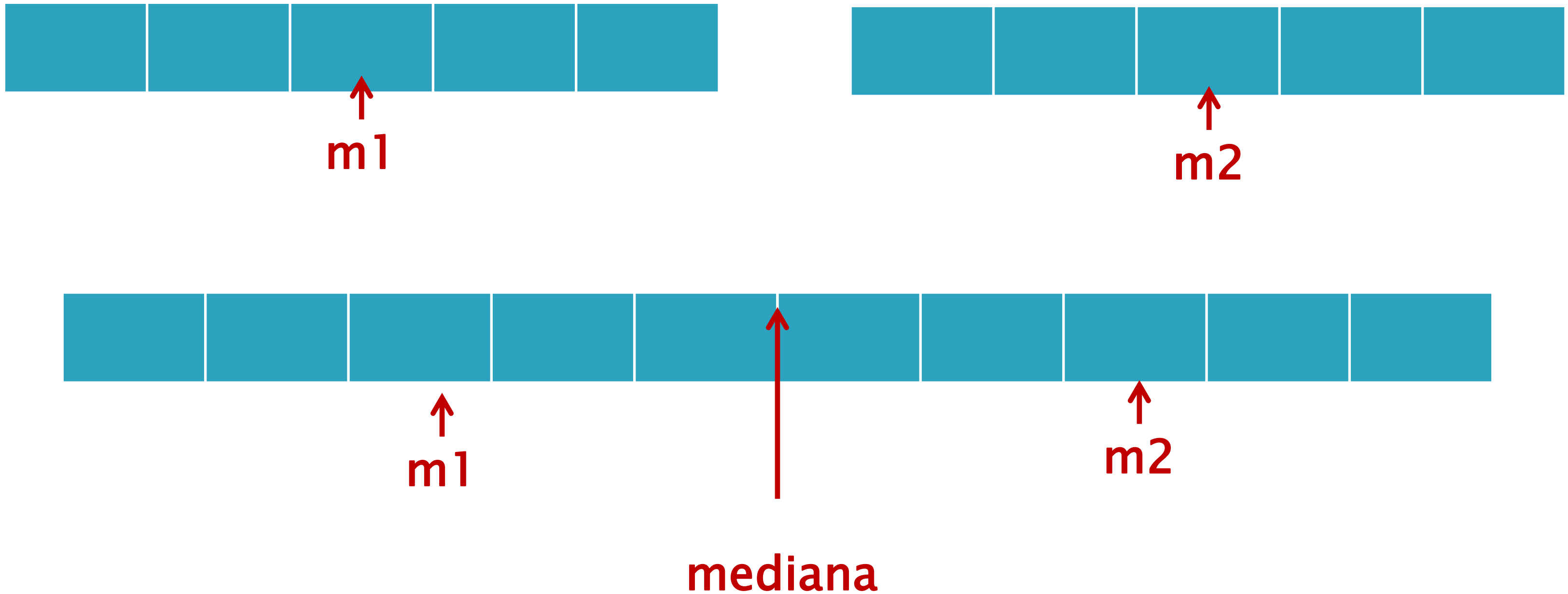
- Fie `m1` mediana vectorului `a` și `m2` mediana vectorului `b`

- Comparăm m1 şi m2

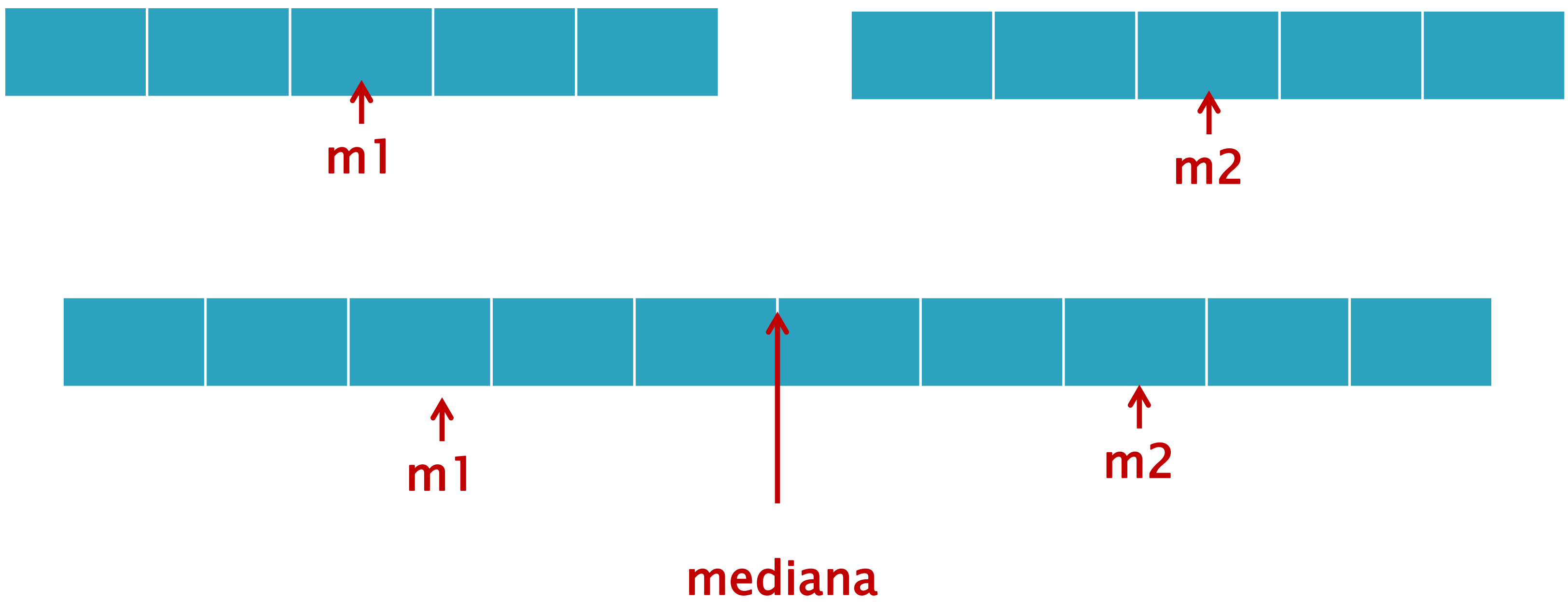
- Fie `m1` mediana vectorului `a` și `m2` mediana vectorului `b`



- Fie `m1` mediana vectorului `a` și `m2` mediana vectorului `b`

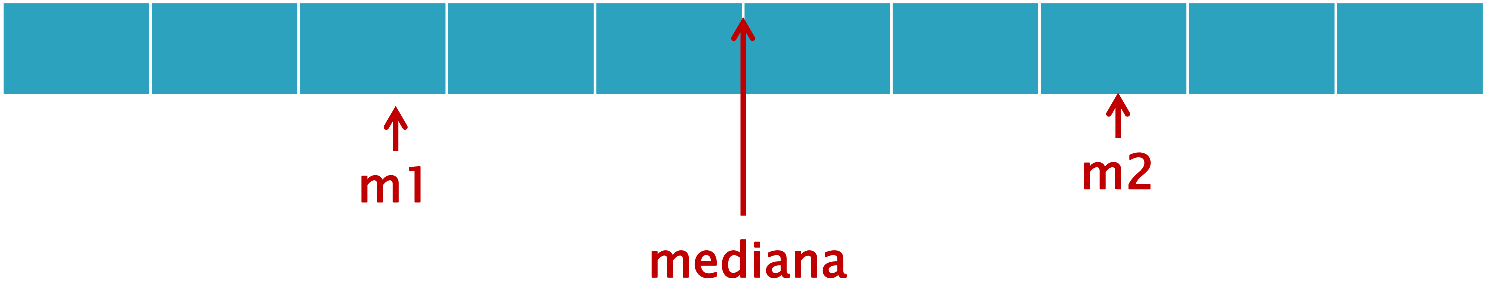


- Fie `m1` mediana vectorului `a` și `m2` mediana vectorului `b`



Pentru a determina mediana este suficient să considerăm:

- Subvectorul drept din primul vector (≥ m1)
- Subvectorul stâng din al doilea vector (≤ m2)

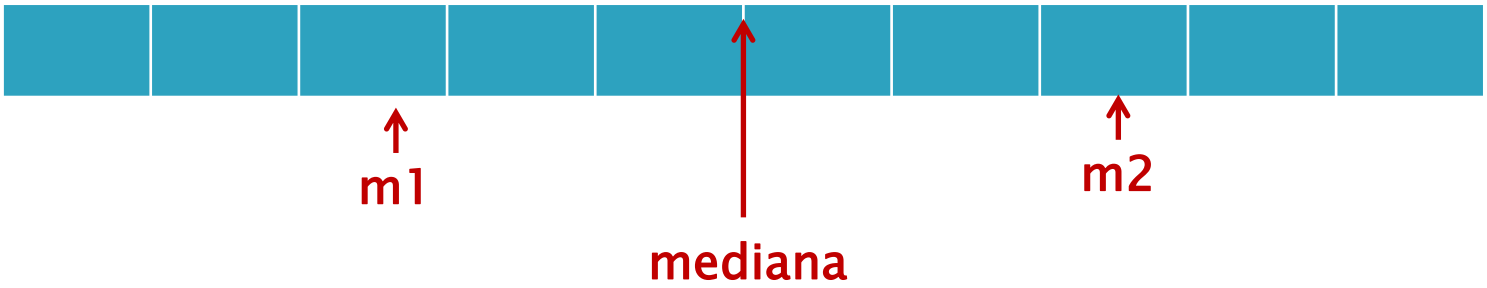


Pentru a determina mediana este suficient să considerăm:

- Subvectorul drept din primul vector (≥ m1)
- Subvectorul stâng din al doilea vector (≤ m2)

Corectitudine:

- Mediana se află în intervalul [m1, m2]

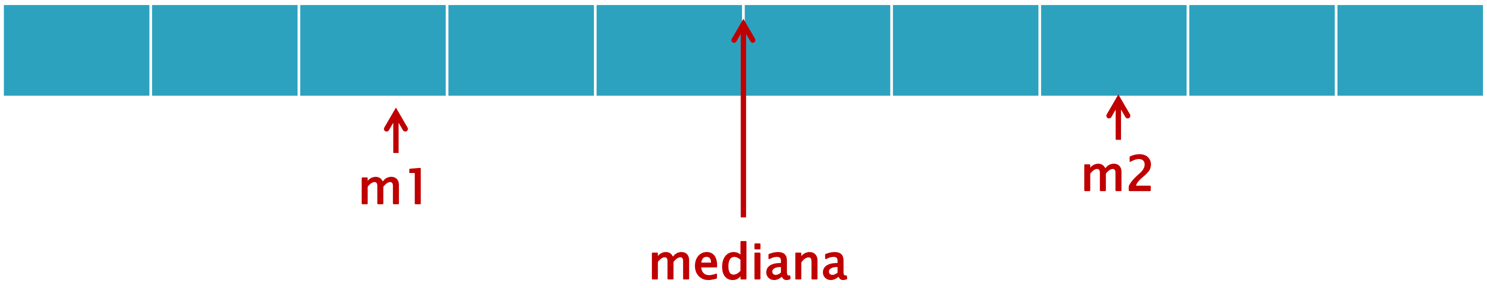


Pentru a determina mediana este suficient să considerăm:

- Subvectorul drept din primul vector (≥ m1)
- Subvectorul stâng din al doilea vector (≤ m2)

<u>Corectitudine</u>:

- Mediana se află în intervalul [m1, m2]
- Prin acest proces se elimina
  - (n-1)/2 elemente mai mici decât mediana
  - (n-1)/2 elemente mai mari decât mediana



Pentru a determina mediana este suficient să considerăm:

- Subvectorul drept din primul vector ($\geq$ m1)
- Subvectorul stâng din al doilea vector ($\leq$ m2)

Corectitudine:

- Mediana se află în intervalul [m1, m2]
- Prin acest proces se elimina
  - (n-1)/2 elemente mai mici decât mediana
  - (n-1)/2 elemente mai mari decât mediana

  Astfel:

  mediana noii probleme = mediana problemei iniţiale

Pseudocod

- Fie `m1` mediana vectorului `a` și `m2` mediana vectorului `b`
  - Dacă `m1 = m2` atunci **această valoare este mediana**

- Fie `m1` mediana vectorului `a` și `m2` mediana vectorului `b`
  - Dacă `m1 = m2` atunci **această valoare este mediana**
  - Dacă `m1 > m2` atunci mediana se află în unul din **subvectorii**

    $a[\ \mathbf{0..\lfloor n/2 \rfloor}\ ],\quad b[\ \mathbf{\lfloor (n-1)/2 \rfloor..n-1}\ ]$

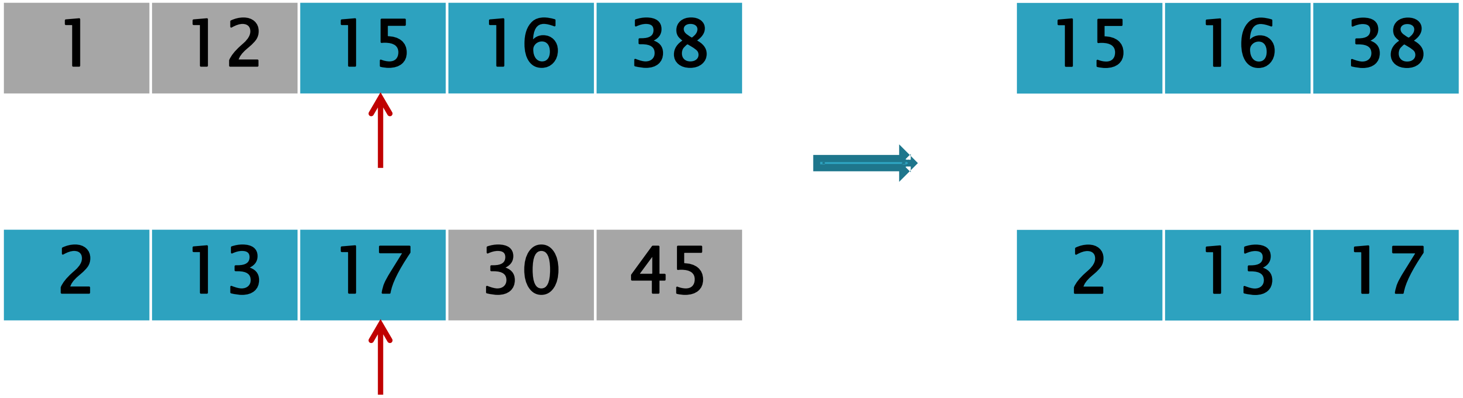
- Fie `m1` mediana vectorului `a` și `m2` mediana vectorului `b`
  - Dacă `m1 = m2` atunci **această valoare este mediana**
  - Dacă `m1 > m2` atunci mediana se află în unul din **subvectorii**

    $a[\,\mathbf{0..\lfloor n/2 \rfloor}\,],\quad b[\,\mathbf{\lfloor (n-1)/2 \rfloor..n-1}\,]$
  - Dacă `m1 < m2` atunci mediana se află în unul din **subvectorii**

    $a[\,\mathbf{\lfloor (n-1)/2 \rfloor..n-1}\,],\quad b[\,\mathbf{0..\lfloor n/2 \rfloor}\,]$

1	12	15	16	38

2	13	17	30	45


1 12 15 16 38
2 13 17 30 45
15 16 38
2 13 17

- **Ştim să rezolvăm direct**:
  - n = 1: (a[1]+b[1])/2
  - n = 2: (max{a[1],b[1]}+min{a[2],b[2]})/2

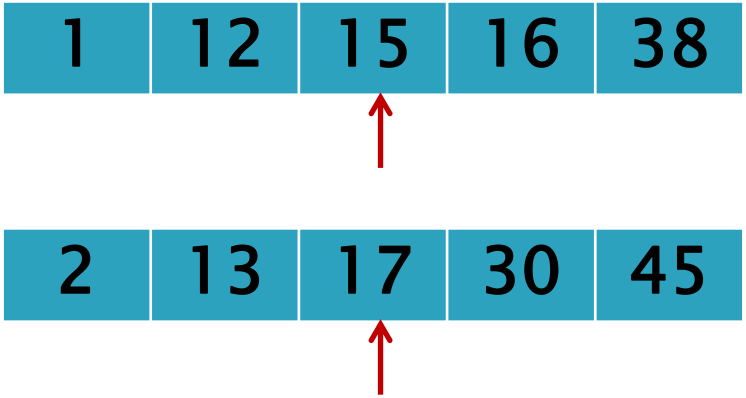
Mediana a doi vectori sortați

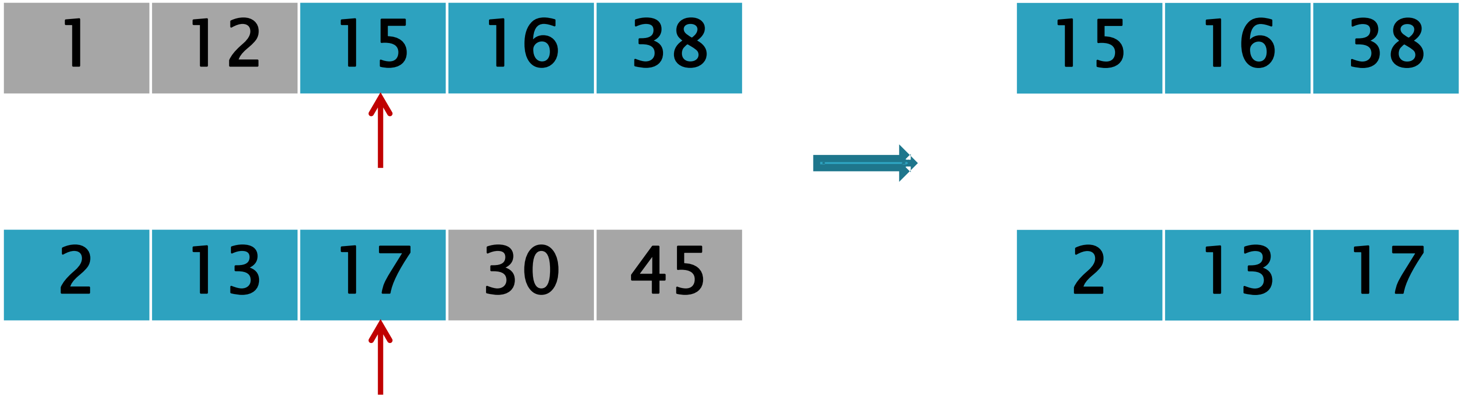
- **Complexitate:** $O(\log n)$

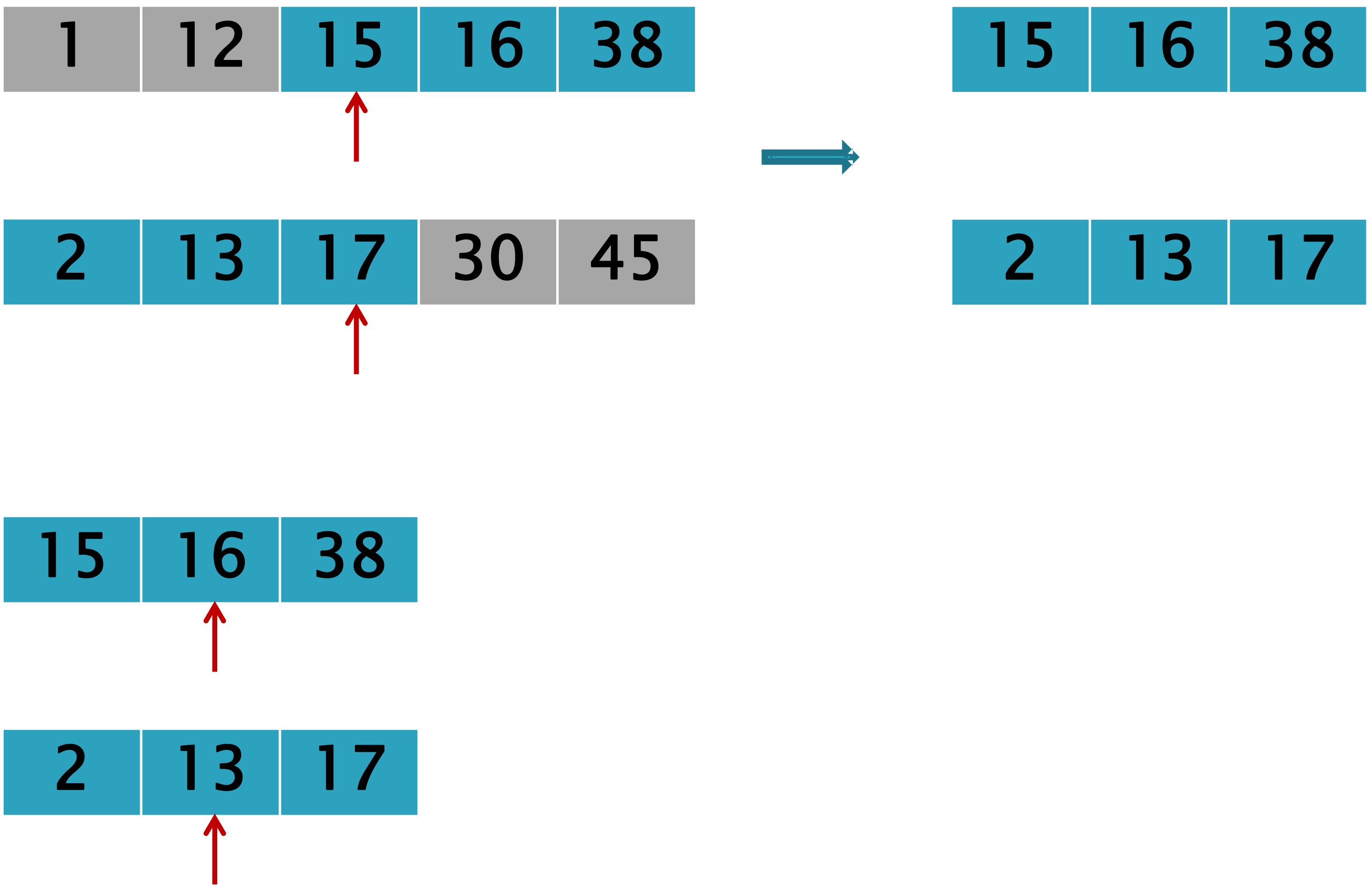
Exemplu

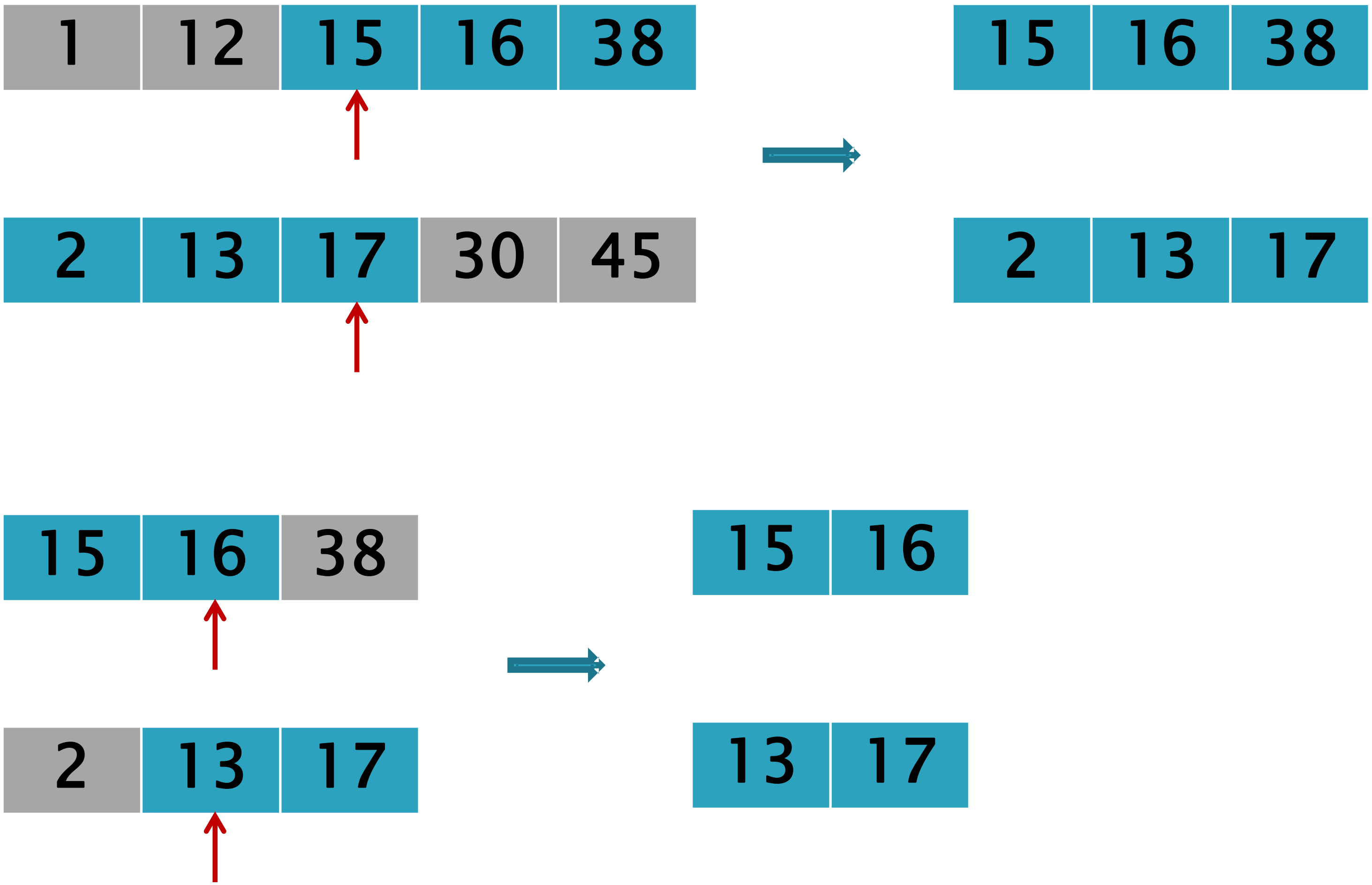
1	12	15	16	38

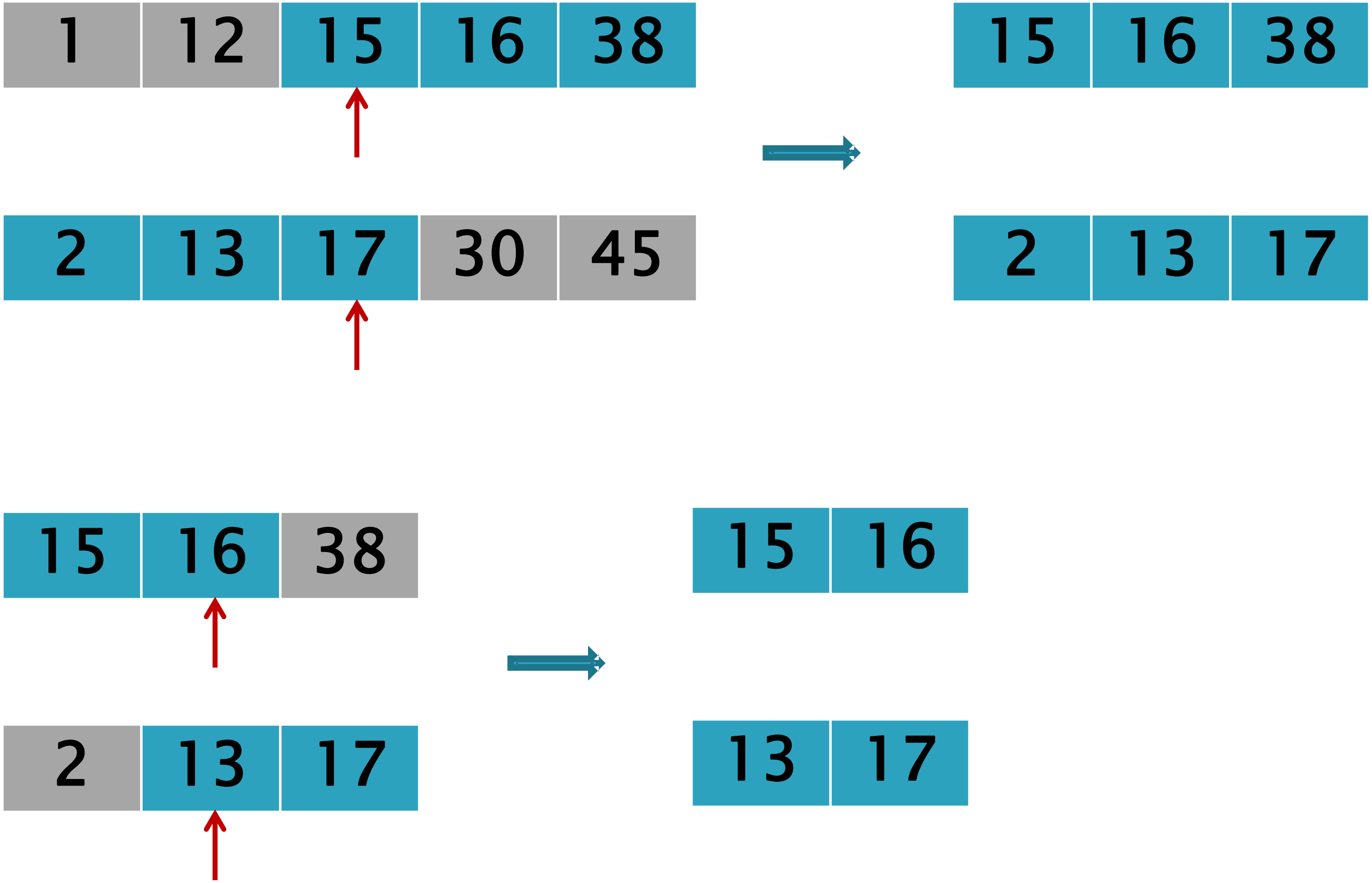
2	13	17	30	45


1 12 15 16 38
2 13 17 30 45


1 12 15 16 38
2 13 17 30 45
15 16 38
2 13 17


1 12 15 16 38
2 13 17 30 45
15 16 38
2 13 17
15 16 38
2 13 17


1 12 15 16 38
2 13 17 30 45
15 16 38
2 13 17
15 16 38
2 13 17
15 16
13 17



$$\textbf{Mediana} = \frac{\max\{13,15\}+\min\{16,17\}}{2} = \frac{15+16}{2}$$

$$= 15{,}5$$

Exemplul 2

1	12	15	16	38	40

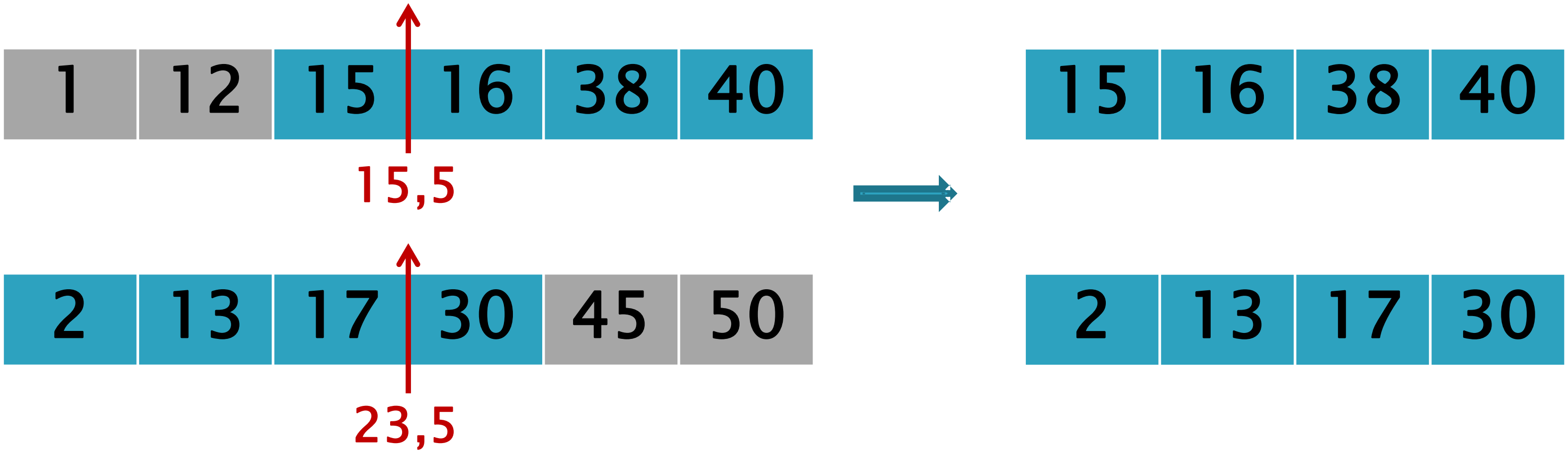
2	13	17	30	45	50

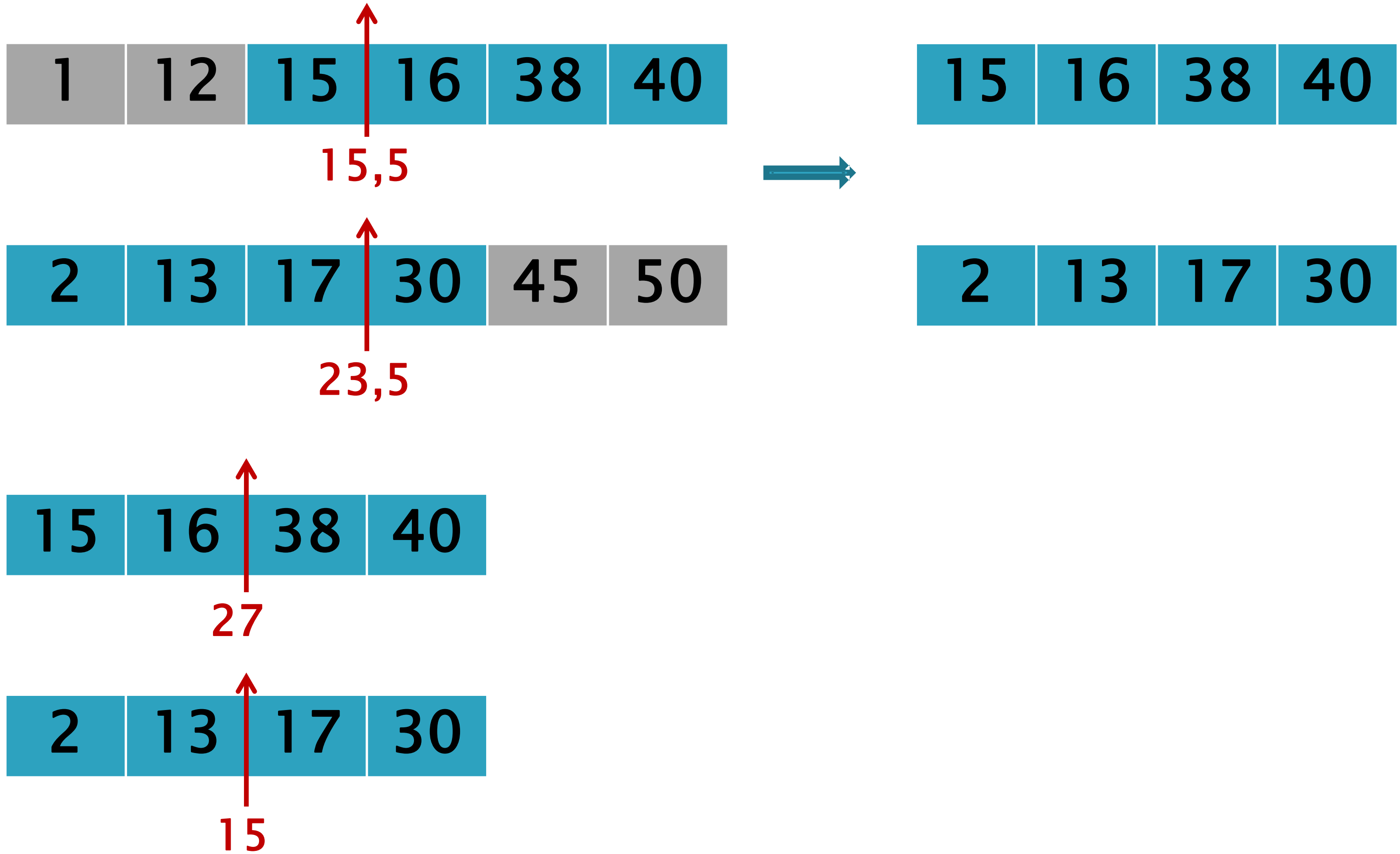
1	12	15	16	38	40

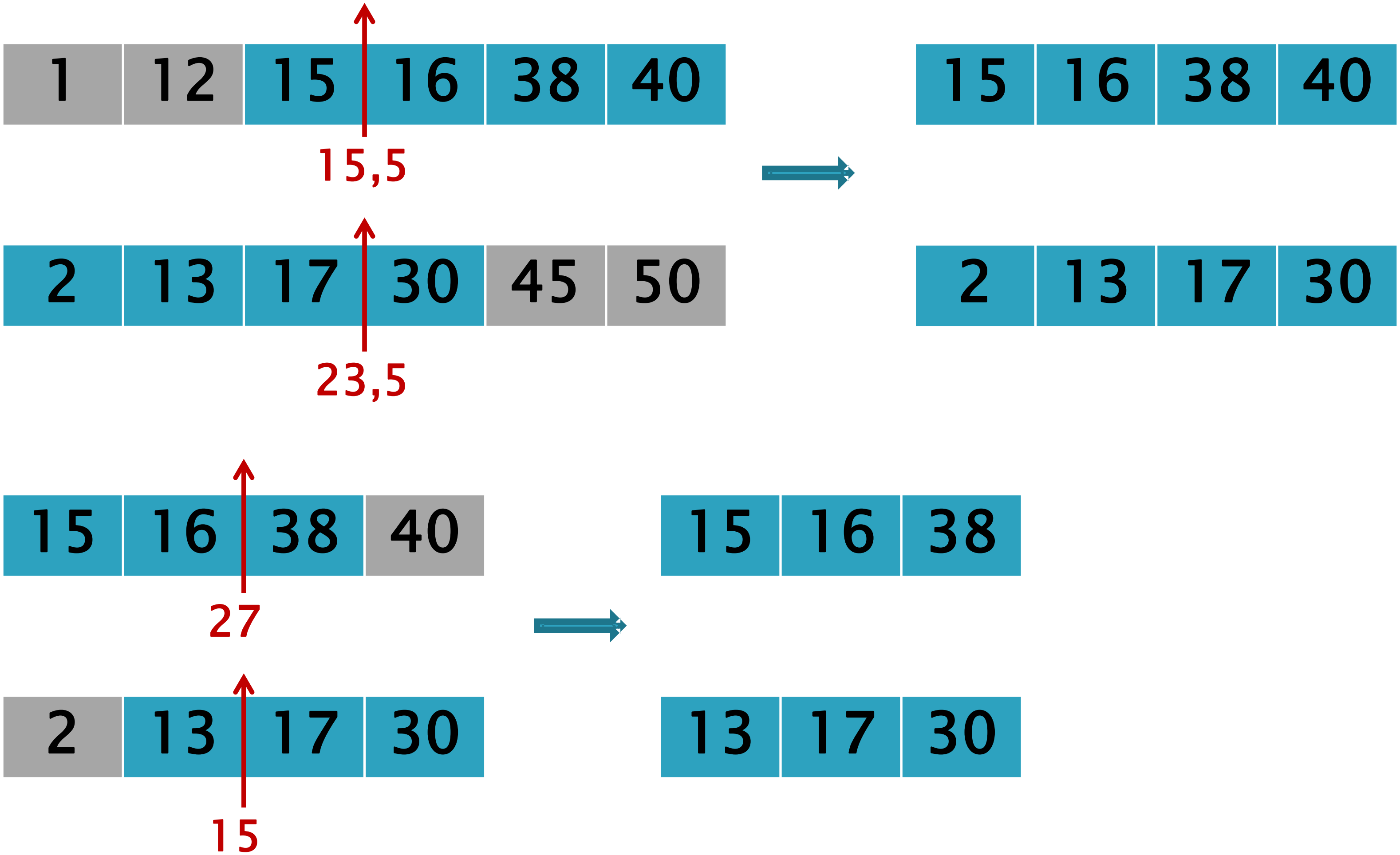
15,5

2	13	17	30	45	50

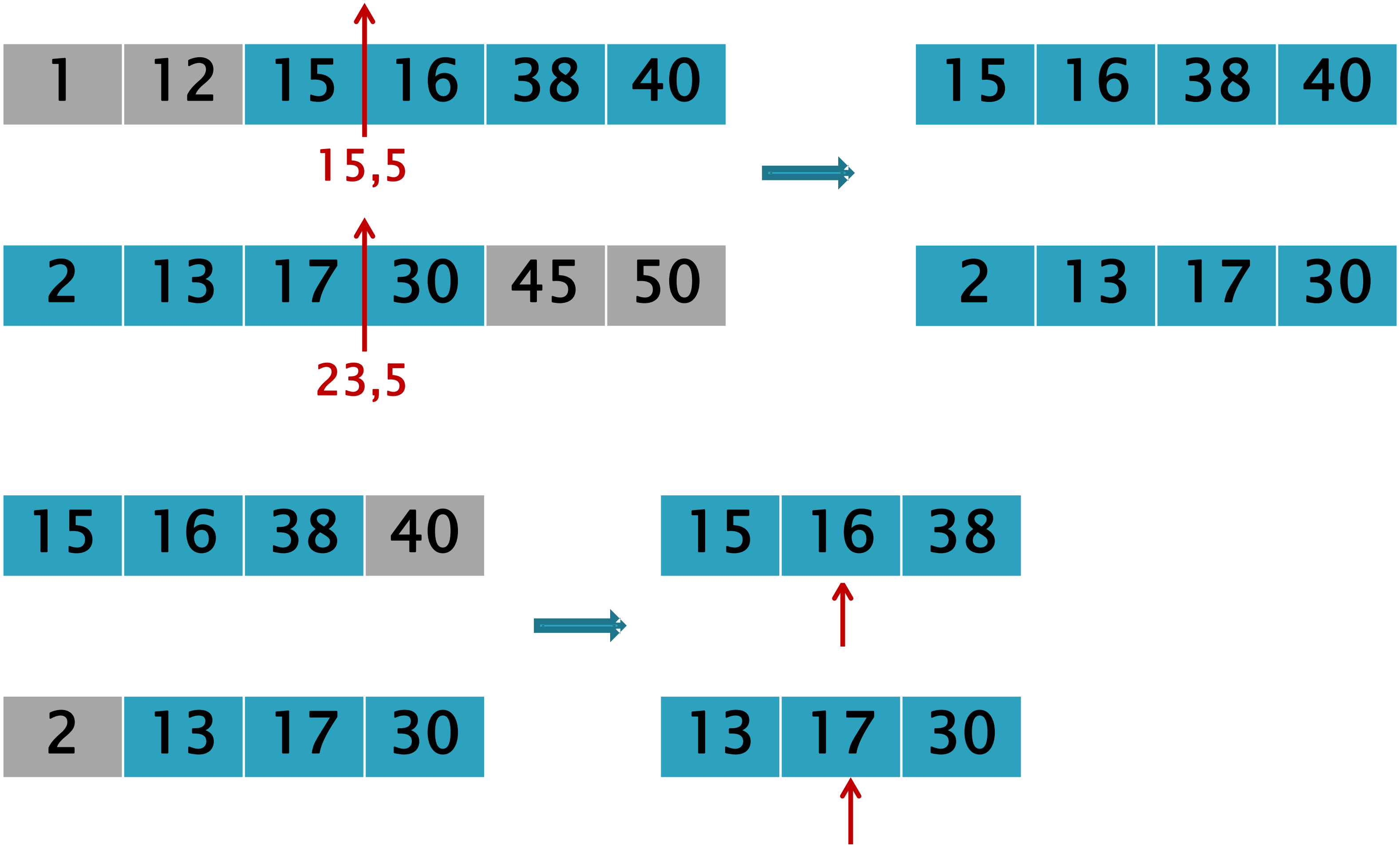
23,5

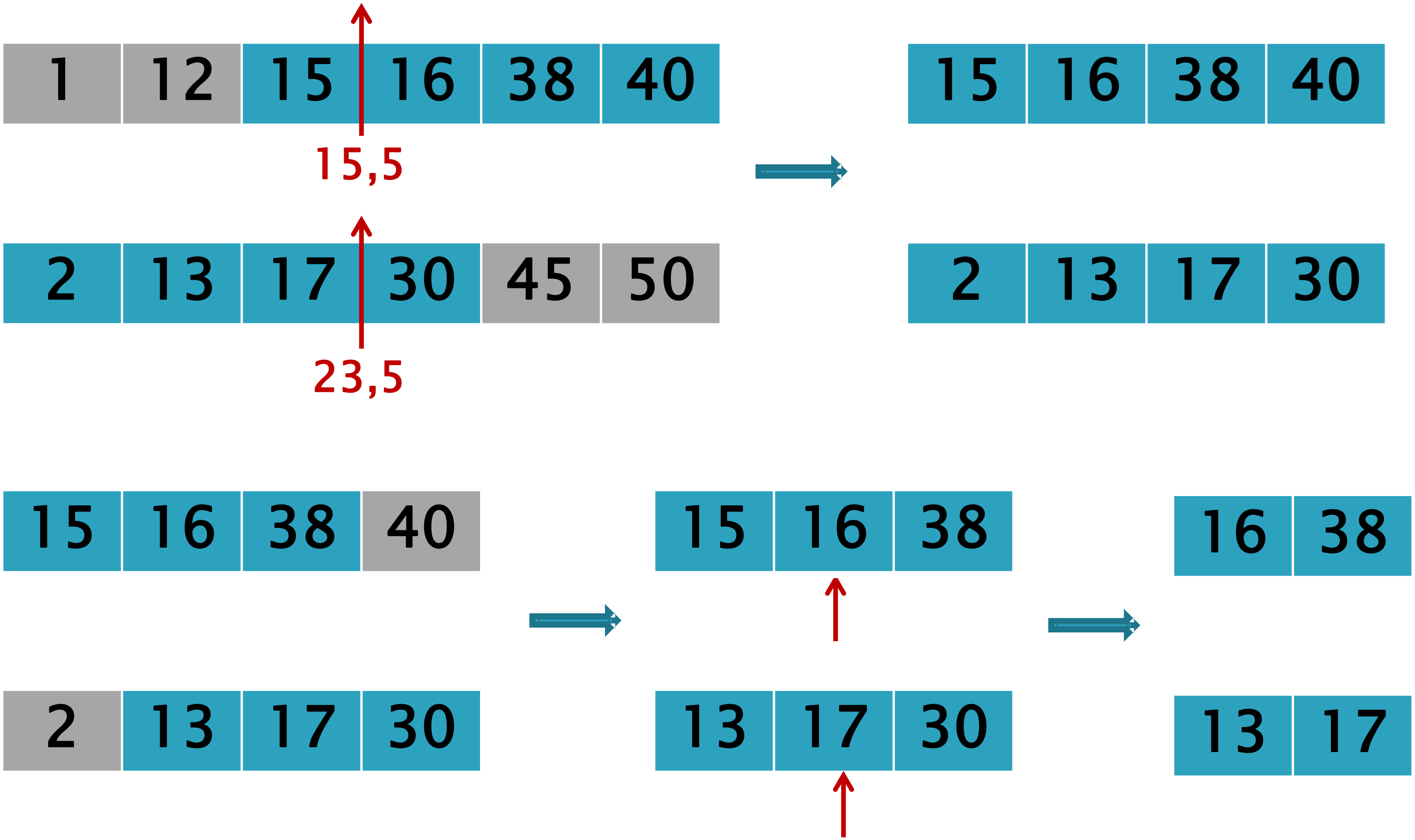

1 12 15 16 38 40
15,5
2 13 17 30 45 50
23,5
15 16 38 40
2 13 17 30

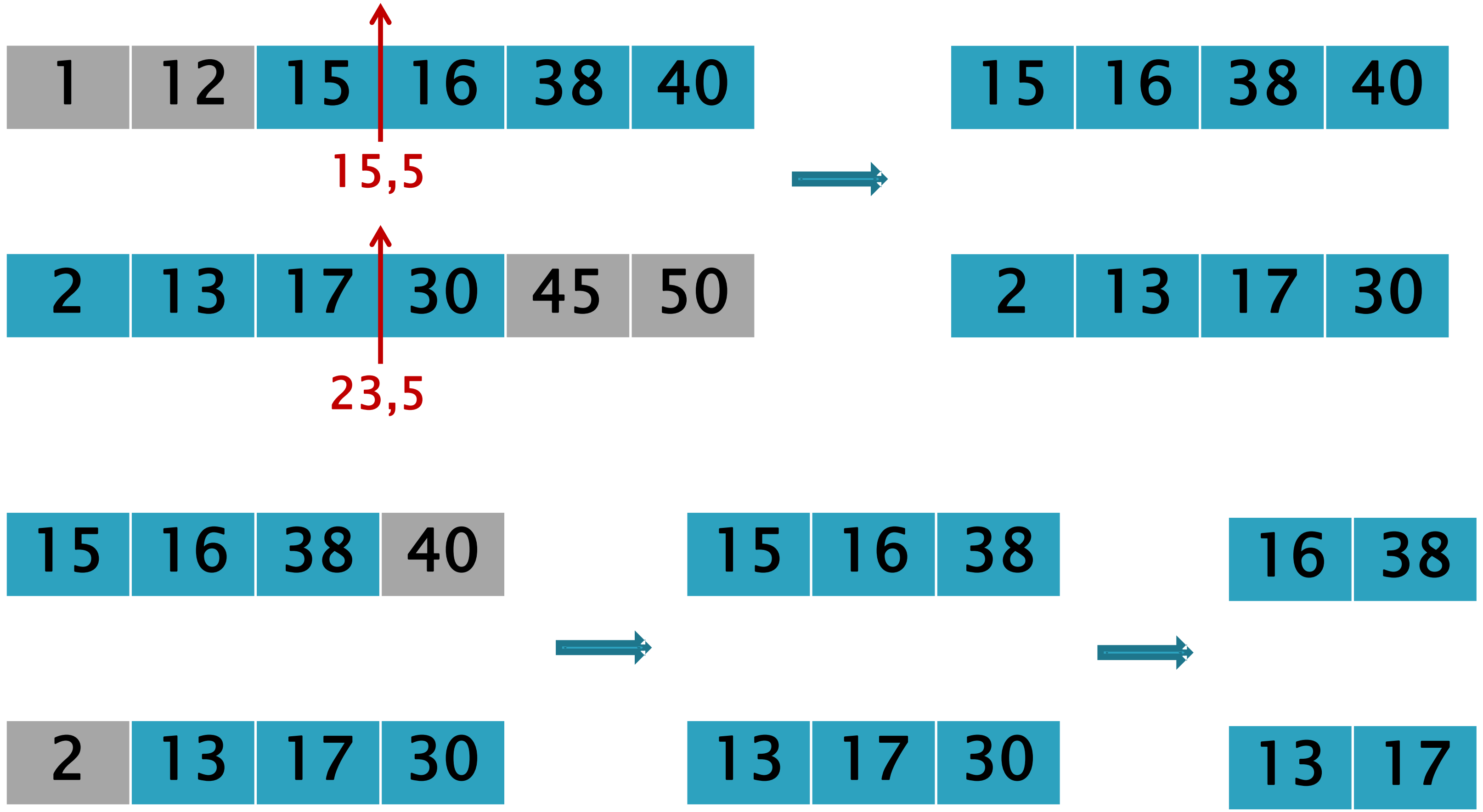

1 12 15 16 38 40
15,5
2 13 17 30 45 50
23,5
15 16 38 40
2 13 17 30
15 16 38 40
27
2 13 17 30
15



1 12 15 16 38 40
15,5
15 16 38 40
2 13 17 30 45 50
23,5
2 13 17 30
15 16 38 40
27
15 16 38
2 13 17 30
15
13 17 30


1 12 15 16 38 40
15,5
2 13 17 30 45 50
23,5
15 16 38 40
2 13 17 30
15 16 38 40
2 13 17 30
15 16 38
13 17 30


1 12 15 16 38 40
15,5
2 13 17 30 45 50
23,5
15 16 38 40
2 13 17 30
15 16 38 40
2 13 17 30
15 16 38
13 17 30
16 38
13 17



$$\textbf{Mediana} = \frac{\max\{13,16\}+\min\{17,38\}}{2} = \frac{16+17}{2}$$

$$= 16{,}5$$

Algoritm

```
double calculMediana(int pa, int ua,int pb, int ub){
```

```
double calculMediana(int pa, int ua,int pb, int ub){
    int n = ua-pa+1;
    if (n<=2) //rezolv direct
      return (max(a[pa],b[pb])+min(a[ua],b[ub]))/2.0;
```

```
double calculMediana(int pa, int ua,int pb, int ub){
    int n = ua-pa+1;
    if (n<=2) //rezolv direct
      return (max(a[pa],b[pb])+min(a[ua],b[ub]))/2.0;
    double m1=mediana(a,pa,ua);//mediana lui a[pa..ua]
    double m2=mediana(b,pb,ub);//mediana lui b[pb..ub]
```

```
double calculMediana(int pa, int ua,int pb, int ub){
    int n = ua-pa+1;
    if (n<=2) //rezolv direct
      return (max(a[pa],b[pb])+min(a[ua],b[ub]))/2.0;
    double m1=mediana(a,pa,ua);//mediana lui a[pa..ua]
    double m2=mediana(b,pb,ub);//mediana lui b[pb..ub]
    if(m1==m2) return m1;




}
```

```
double calculMediana(int pa, int ua,int pb, int ub){
    int n = ua-pa+1;
    if (n<=2) //rezolv direct
      return (max(a[pa],b[pb])+min(a[ua],b[ub]))/2.0;

    double m1=mediana(a,pa,ua);//mediana lui a[pa..ua]
    double m2=mediana(b,pb,ub);//mediana lui b[pb..ub]

    if(m1==m2) return m1;
    if (m1>m2)
      return calculMediana(pa, pa+n/2, pb+(n-1)/2,ub);



}
```

```
double calculMediana(int pa, int ua,int pb, int ub){
    int n = ua-pa+1;
    if (n<=2) //rezolv direct
      return (max(a[pa],b[pb])+min(a[ua],b[ub]))/2.0;

    double m1=mediana(a,pa,ua);//mediana lui a[pa..ua]
    double m2=mediana(b,pb,ub);//mediana lui b[pb..ub]

    if(m1==m2) return m1;
    if (m1>m2)
      return calculMediana(pa, pa+n/2, pb+(n-1)/2,ub);
    else
      return calculMediana(pa+(n-1)/2, ua, pb,pb+n/2);
}
```

```
double calculMediana(int pa, int ua,int pb, int ub){
    int n = ua-pa+1;
    if (n<=2) //rezolv direct
      return (max(a[pa],b[pb])+min(a[ua],b[ub]))/2.0;
    double m1=mediana(a,pa,ua);//mediana lui a[pa..ua]
    double m2=mediana(b,pb,ub);//mediana lui b[pb..ub]
    if(m1==m2) return m1;
    if (m1>m2)
      return calculMediana(pa, pa+n/2, pb+(n-1)/2,ub);
    else
      return calculMediana(pa+(n-1)/2, ua, pb,pb+n/2);
}

double calculMediana(){
     return calculMediana(0,n-1,0,n-1);
}
```

```java
 double mediana(double[] v, int pv, int uv){
     int n=uv-pv+1;
     int m=(uv+pv)/2;
     if (n%2==0)
           return(v[m]+v[m+1])/2.0;
     else
           return v[m];
}
```

[Mediana.java]()

Metoda 2

- **Idee:**
  - putem testa în timp constant dacă **un element fixat a[i] este mediana dorită:**

    `b[j] ≤ a[i] ≤ b[j+1]`

    pentru `j = n – i – 1`

Metoda 2

- **Idee:**
  - putem testa în timp constant dacă **un element fixat a[i] este mediana dorită:**

    $$b[j] \leq a[i] \leq b[j+1]$$

    pentru $j = n - i - 1$

  - căutăm binar mediana în a
  - dacă nu o găsim căutăm binar mediana în b

Cele mai apropiate puncte

Se dau n puncte în plan prin coordonatele lor.

Să se determine distanţa dintre cele mai apropiate două puncte.

Cele mai apropiate puncte

- **Varianta 1**: Considerăm toate perechile de puncte

  $O(n^2)$

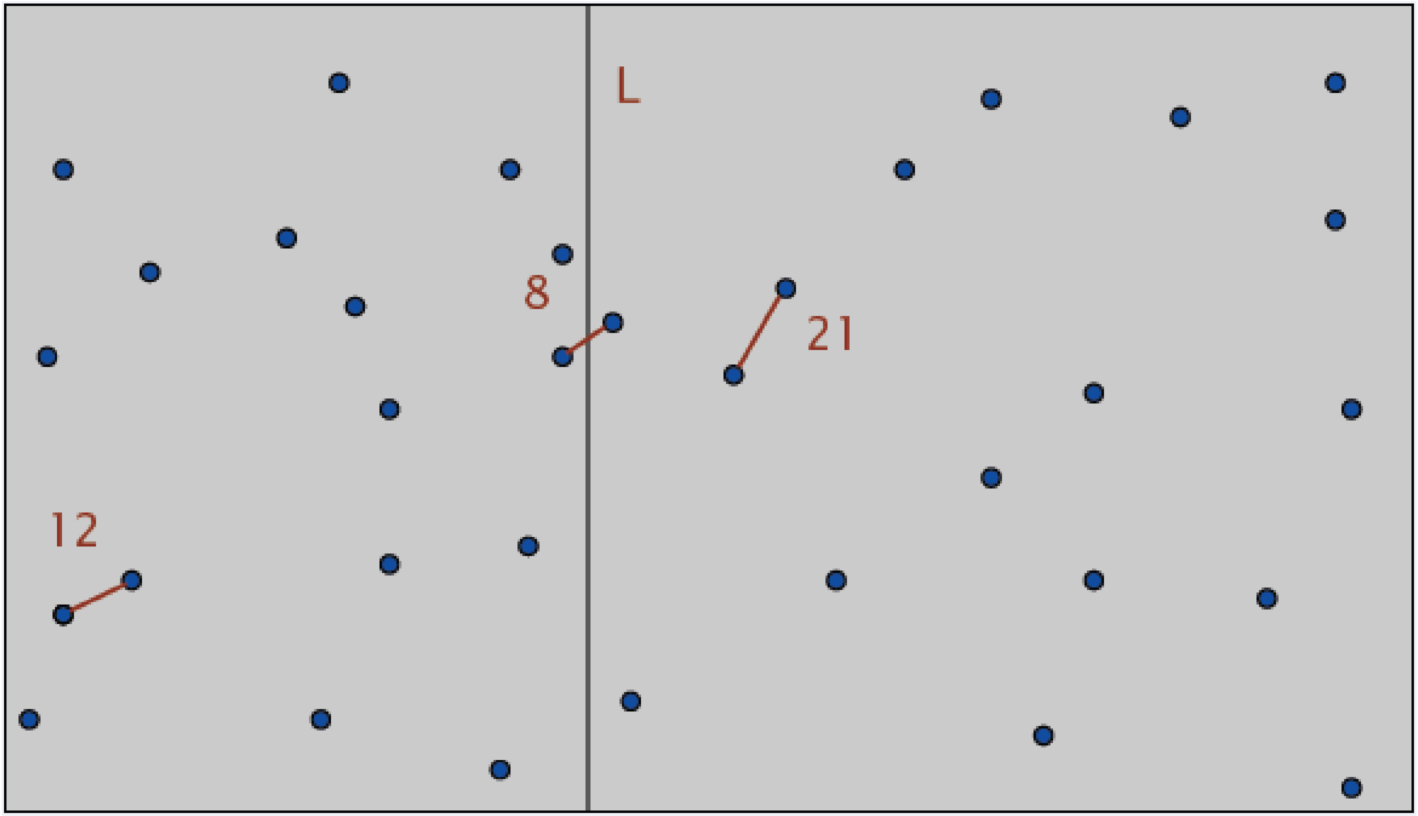
Cele mai apropiate puncte

- **Varianta 2:** Divide et Impera **O(n log n)**

Cele mai apropiate puncte

- **Varianta 2:** Divide et Impera **O(n log n)**
- **Ipoteză**: Punctele au abscise distincte

Cele mai apropiate puncte



http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/

Cele mai apropiate puncte

- Împărţim mulţimea de puncte în două submulţimi cu n/2 puncte (printr-o dreaptă verticală L)
-
-
-

Cele mai apropiate puncte

- Împărţim mulţimea de puncte în două submulţimi cu n/2 puncte (printr-o dreaptă verticală L)
- Rezolvăm problema pentru cele două submulţimi şi obţinem distanţele minime $\mathbf{d_1}$, respectiv $\mathbf{d_2}$
-
-

Cele mai apropiate puncte

- Împărţim mulţimea de puncte în două submulţimi cu n/2 puncte (printr-o dreaptă verticală L)
- Rezolvăm problema pentru cele două submulţimi şi obţinem distanţele minime $\mathbf{d_1}$, respectiv $\mathbf{d_2}$
- Determinăm distanţa minimă $\mathbf{d_3}$ între două puncte din submulţimi diferite
-

Cele mai apropiate puncte

- Împărţim mulţimea de puncte în două submulţimi cu n/2 puncte (printr-o dreaptă verticală L)
- Rezolvăm problema pentru cele două submulţimi şi obţinem distanţele minime $\mathbf{d_1}$, respectiv $\mathbf{d_2}$
- Determinăm distanţa minimă $\mathbf{d_3}$ între două puncte din submulţimi diferite
- Returnăm minimul dintre distanţele $\mathbf{d_1}$, $\mathbf{d_2}$ şi $\mathbf{d_3}$

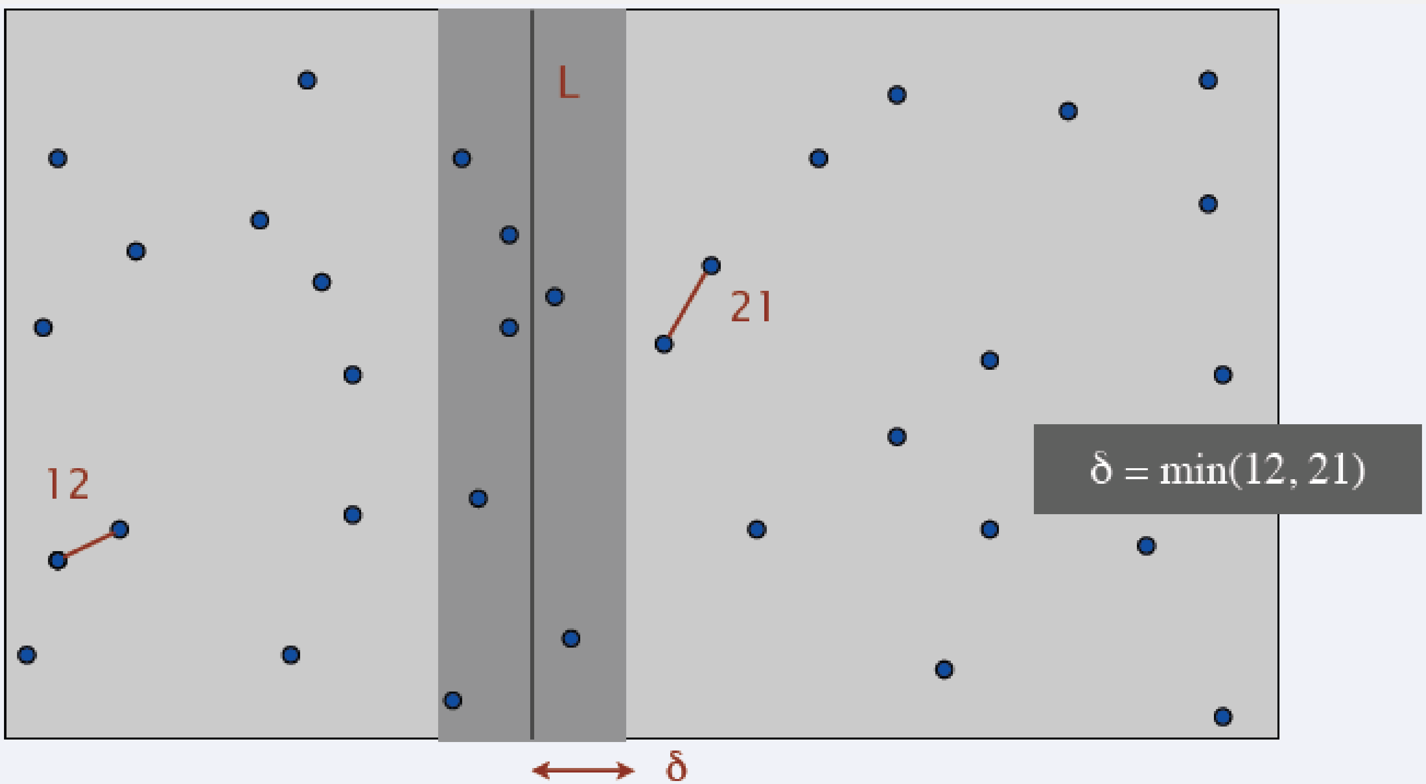
Cele mai apropiate puncte

Cum determinăm eficient distanţa minimă d_3 între două puncte din submulţimi diferite?

Cele mai apropiate puncte

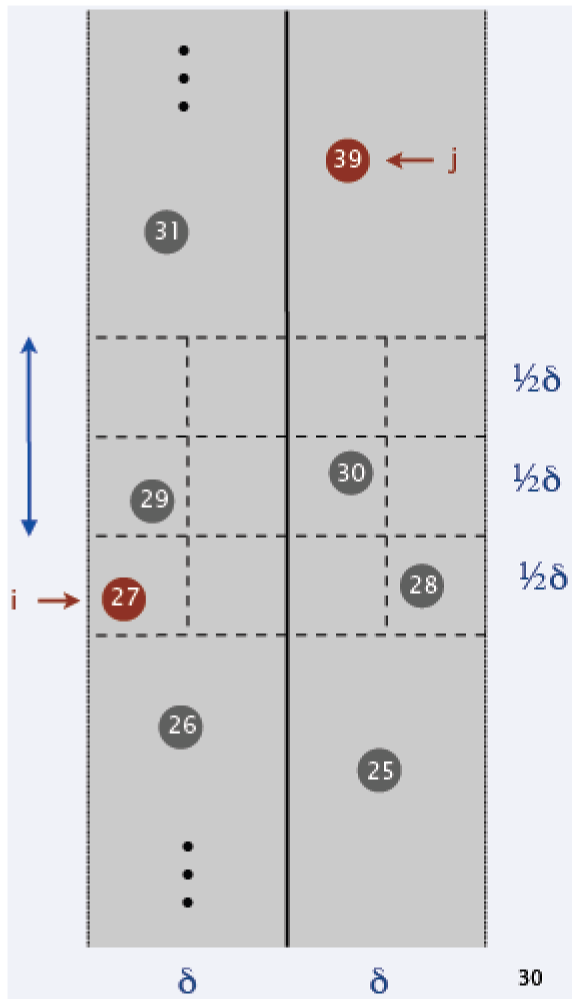
Fie $\delta = \min\{d_1, d_2\}$.

1. Este suficient să considerăm puncte la distanţă cel mult δ de dreapta L



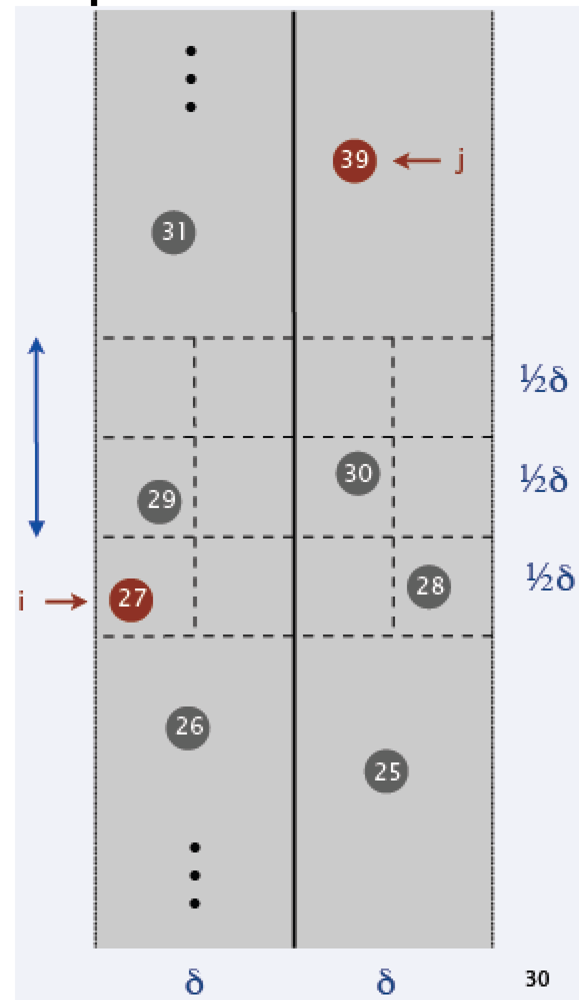
Cele mai apropiate puncte

2. Două puncte din submulţimi diferite aflate la o distanţă mai mică decât δ se situează într-un dreptunghi de dimensiuni δ × 2δ, centrat pe dreapta L



Cele mai apropiate puncte

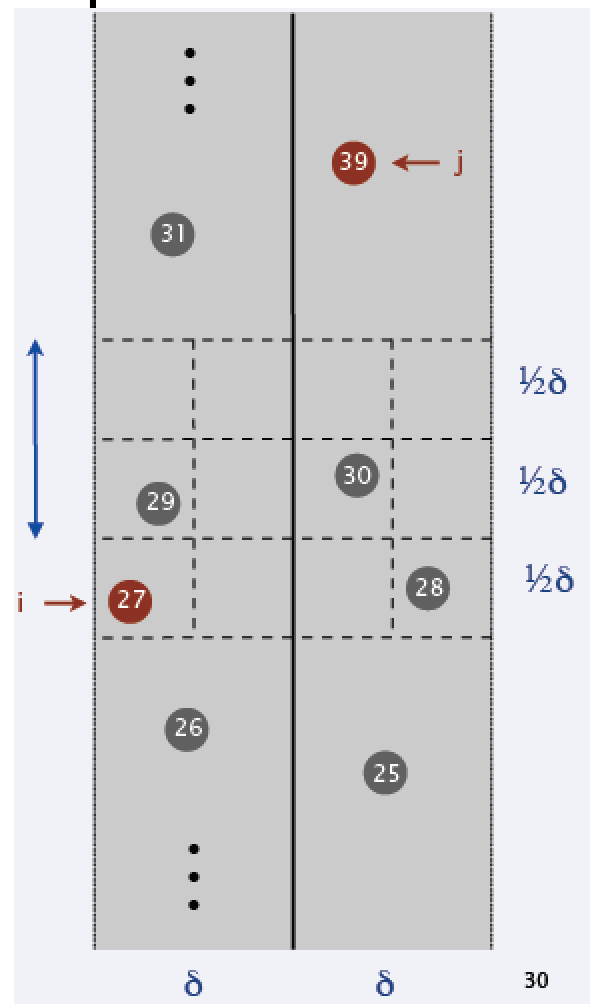
3. Deoarece d_1, $d_2 \geq \delta$, într-un dreptunghi de dimensiuni $\delta \times 2\delta$, centrat pe dreapta L se pot afla maxim 8 puncte



Cele mai apropiate puncte

3. Deoarece d_1, $d_2 \geq \delta$, într-un dreptunghi de dimensiuni $\delta \times 2\delta$, centrat pe dreapta L se pot afla maxim 8 puncte

Consecinţă: Pentru a calcula d_3 avem nevoie doar de 7 puncte care urmează după fiecare punct p din bandă, în şirul Y al punctelor sortate crescător după ordonată



Cele mai apropiate puncte

4. Pentru a nu sorta de fiecare dată punctele din bandă crescător după ordonată se pot interclasa şirurile deja sortate (! în etapa de divide) după ordonată ale punctelor din cele două submulţimi

Algoritm

- X - mulţimea punctelor sortate după abscisă
- Y=X
- **DivImp**(&X, &Y, st, dr) **//Y -sortate după ordonată**

```
dacă |X|<4 atunci
   sorteaza dupa ordonata (Y ,st, dr)
   d = min(perechi de elemente din X[st..dr])
```

- X - mulţimea punctelor sortate după abscisă
- Y=X
- **DivImp**(&X, &Y, st, dr) **//Y -sortate după ordonată**

```
dacă |X|<4 atunci
   sorteaza dupa ordonata (Y ,st, dr)
   d = min(perechi de elemente din X[st..dr])
altfel
   mid=(st+dr)/2
   d1=divimp(X[st..mij], Y, st, mij)
   d2=divimp(X[mij+1..dr], Y, mij+1,dr)
   d=min{d1, d2}
```

- X – mulţimea punctelor sortate după abscisă
- Y=X
- **DivImp**(&X, &Y, st, dr) **//Y -sortate după ordonată**

```
  dacă |X|<4 atunci
    sorteaza dupa ordonata (Y ,st, dr)
    d = min(perechi de elemente din X[st..dr])
  altfel
    mid=(st+dr)/2
    d1=divimp(X[st..mij], Y, st, mij)
    d2=divimp(X[mij+1..dr], Y, mij+1,dr)
    d=min{d1, d2}
    interclaseaza(Y, st, mij, dr)
```

- X – mulţimea punctelor sortate după abscisă
- Y=X
- **DivImp**(&X, &Y, st, dr) **//Y -sortate după ordonată**

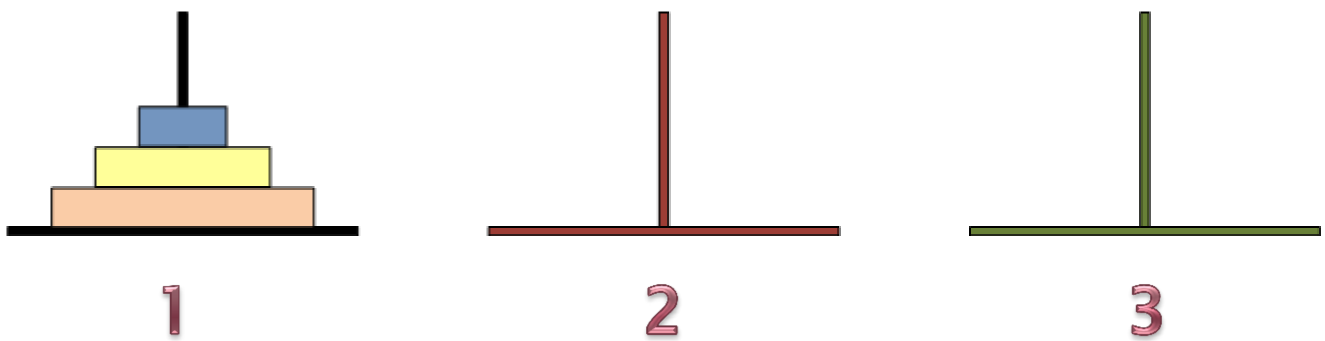
```
dacă |X|<4 atunci
   sorteaza dupa ordonata (Y ,st, dr)
   d = min(perechi de elemente din X[st..dr])
altfel
   mid=(st+dr)/2
   d1=divimp(X[st..mij], Y, st, mij)
   d2=divimp(X[mij+1..dr], Y, mij+1,dr)
   d=min{d1, d2}
   interclaseaza(Y, st, mij, dr)
   calculează d3 considerând punctele p din Y
   aflate la distanta ≤ d de abscisa punctului
   X[mid] si perechile formate de p cu fiecare din
   cele 7 puncte care îi urmează în Y
```

- X – mulţimea punctelor sortate după abscisă
- Y=X
- **DivImp**(&X, &Y, st, dr) **//Y -sortate după ordonată**

```
dacă |X|<4 atunci
   sorteaza dupa ordonata (Y ,st, dr)
   d = min(perechi de elemente din X[st..dr])
altfel
   mid=(st+dr)/2
   d1=divimp(X[st..mij], Y, st, mij)
   d2=divimp(X[mij+1..dr], Y, mij+1,dr)
   d=min{d1, d2}
   interclaseaza(Y, st, mij, dr)
   calculează d3 considerând punctele p din Y
   aflate la distanta ≤ d de abscisa punctului
   X[mid] si perechile formate de p cu fiecare din
   cele 7 puncte care îi urmează în Y
   d=min{d, d3}
   return d
```

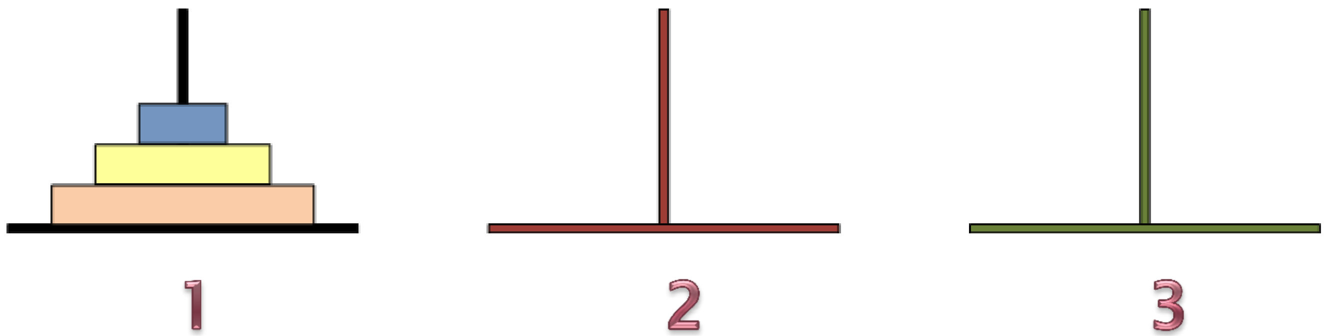
Problema turnurilor din Hanoi

- Se consideră 3 tije. Iniţial pe tija 1 se află n discuri cu diametrele descrescătoare privind de la bază către vârf, iar pe tijele 2 şi 3 nu se află nici un disc.



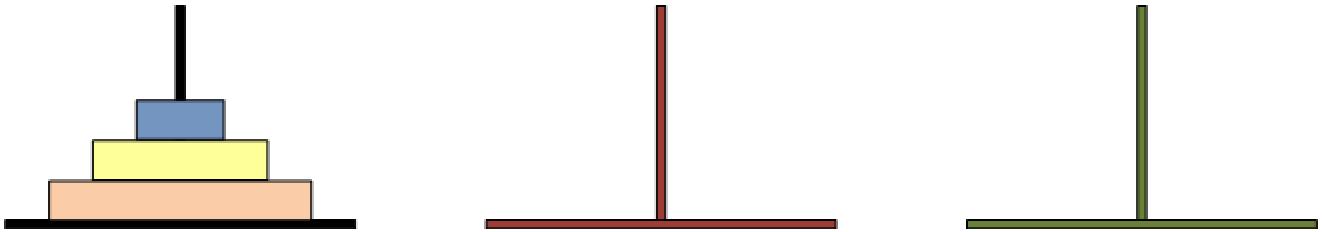
Problema turnurilor din Hanoi

- Se cere să se mute aceste discuri pe tija 2, ajutându-ne şi de tija 3.
- O **mutare** (i,j) constă în deplasarea discului din vârful tijei i în vârful tijei j. Mutarea este **corectă** dacă stiva j este goală sau are în vârf un disc de diametru mai mare decât discul deplasat



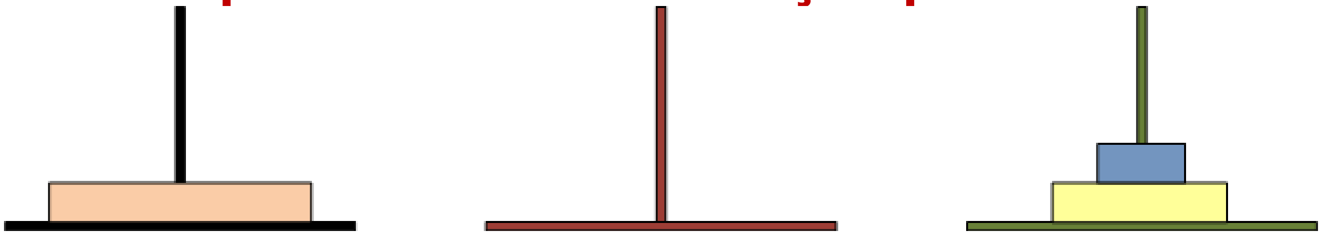
Problema turnurilor din Hanoi

- Pentru n = 3



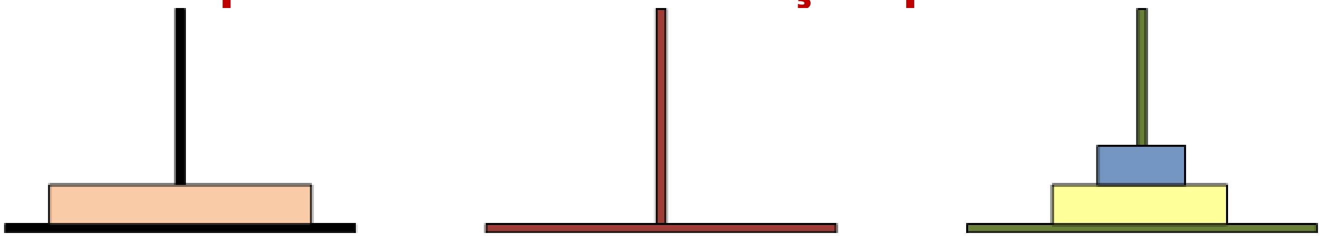
Problema turnurilor din Hanoi

- Mutăm două discuri de pe stiva 1 pe stiva 3, folosind stiva 2 – **subproblemă de acelaşi tip**

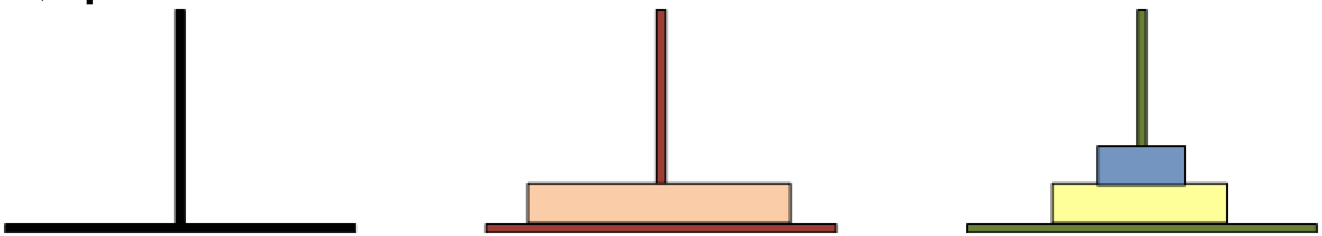


Problema turnurilor din Hanoi

- Mutăm două discuri de pe stiva 1 pe stiva 3, folosind stiva 2 – **subproblemă de acelaşi tip**

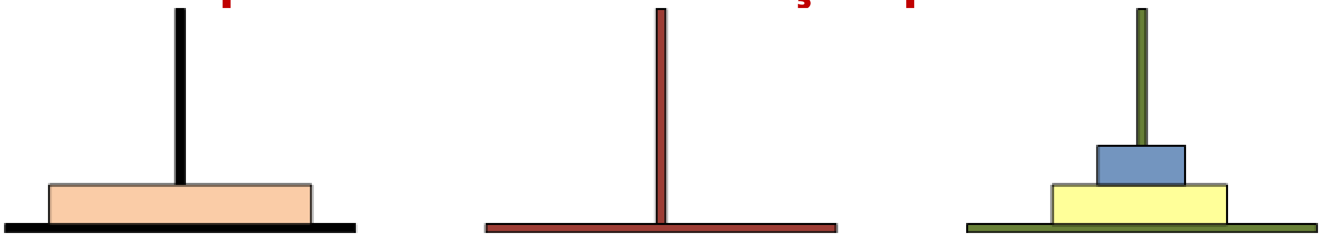


- Mutăm unicul disc rămas pe stiva 1 (cu diametrul maxim) pe stiva 2

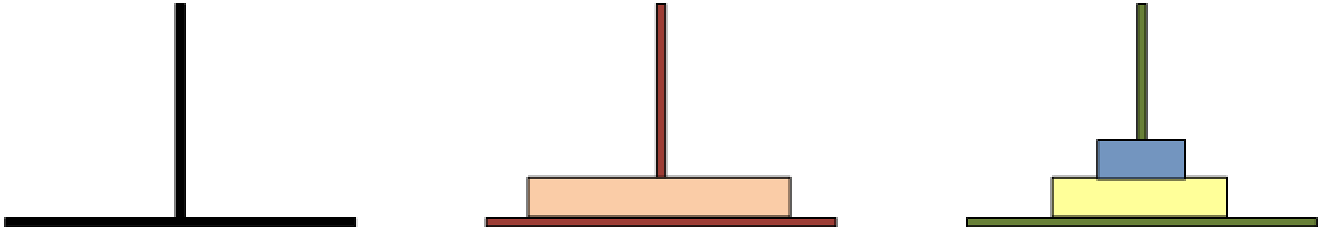


Problema turnurilor din Hanoi

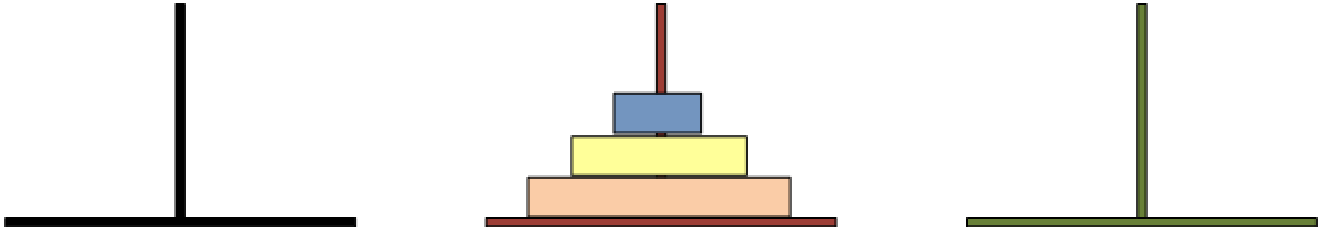
- Mutăm două discuri de pe stiva 1 pe stiva 3, folosind stiva 2 – **subproblemă de acelaşi tip**



- Mutăm unicul disc rămas pe stiva 1 (cu diametrul maxim) pe stiva 2



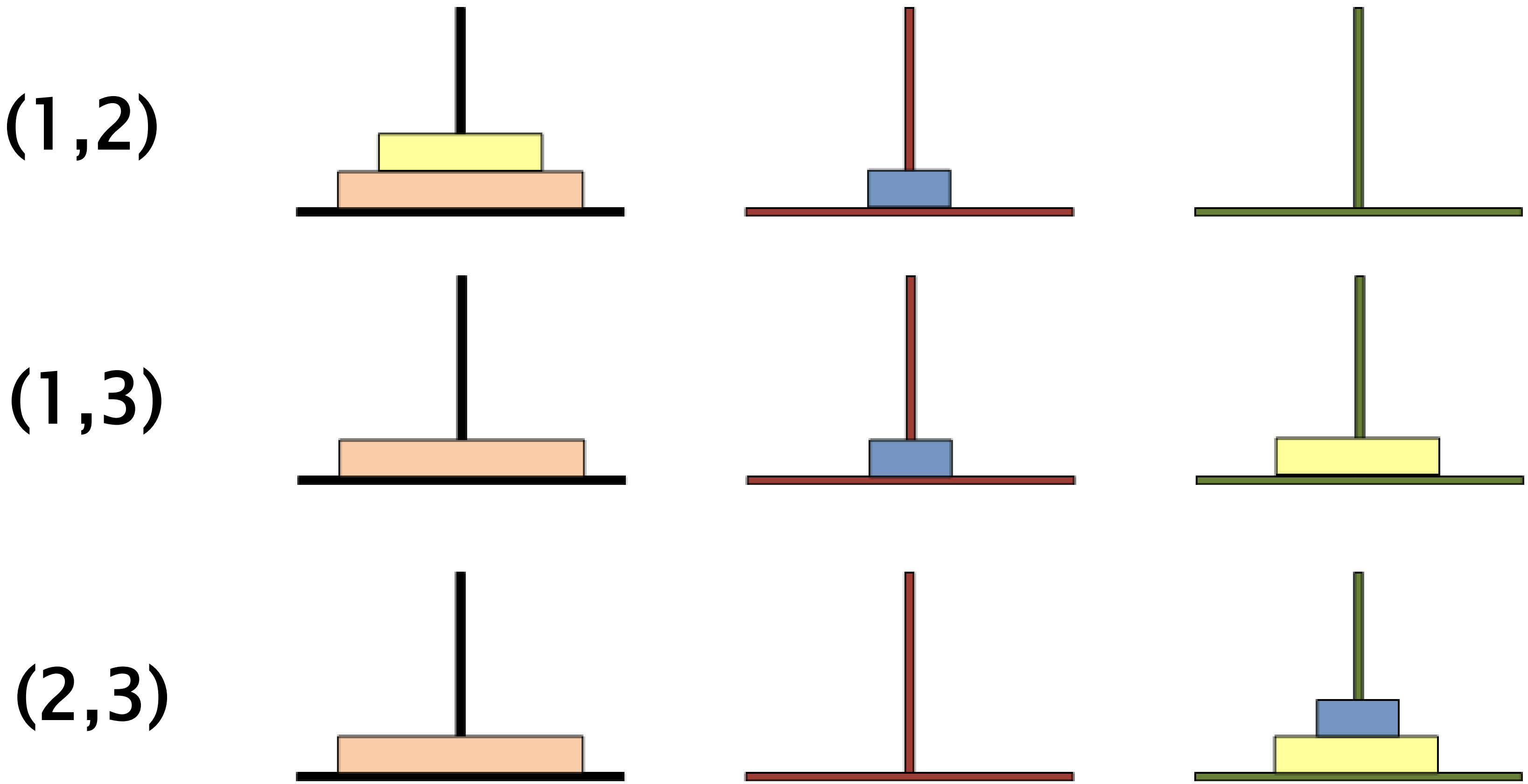
- Mutăm cele două discuri de pe stiva auxiliară 3 pe stiva 2 folosind stiva 1 – **subproblemă de acelaşi tip**



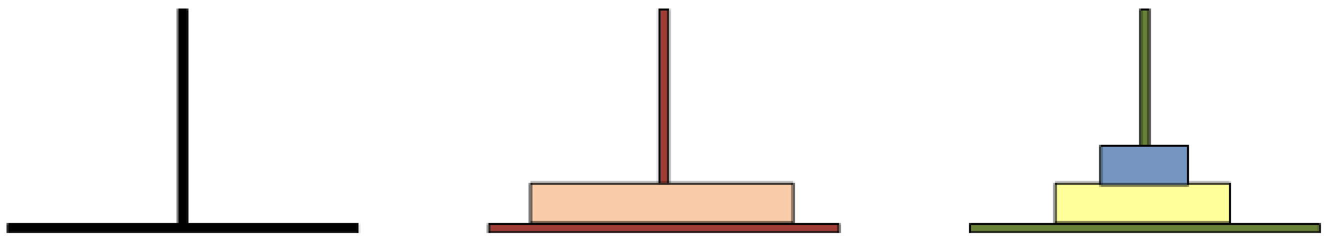
Problema turnurilor din Hanoi

- Rezolvăm, recursiv, **subproblemele de acelaşi tip**

Mutăm două discuri de pe stiva 1 pe stiva 3, folosind stiva 2



Mutăm unicul disc rămas pe stiva 1 (cu diametrul maxim) pe stiva 2



Mutăm două discuri de pe stiva 3 pe stiva 2, folosind stiva 1

(3,1)

(3,2)

(1,2)