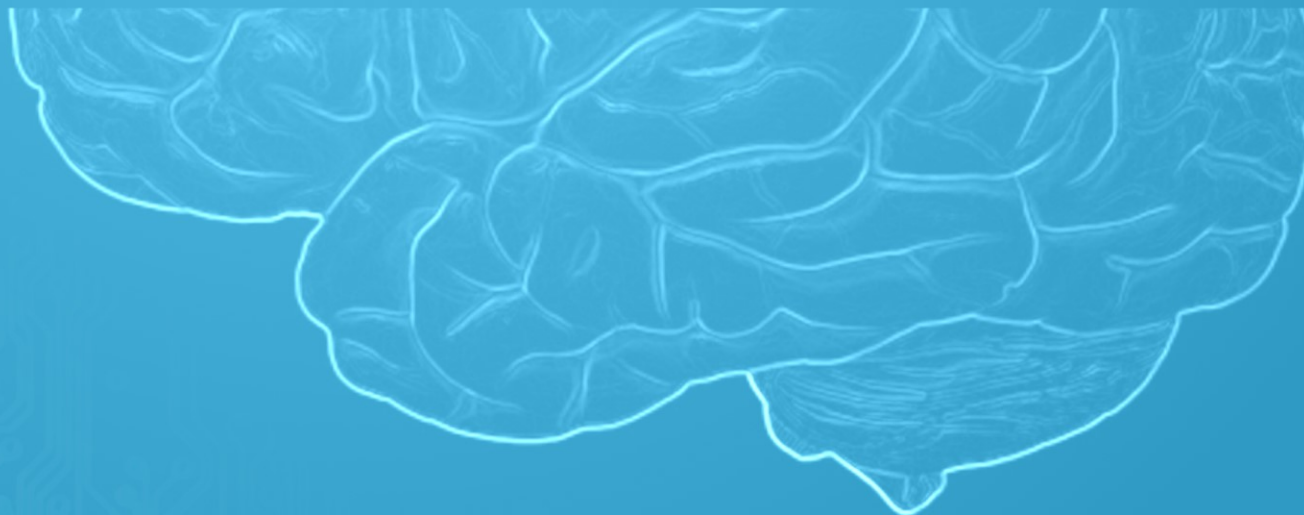




# Arhitecturi Paralele

## Abordarea algoritmilor în mod paralel

Lect. Dr. Ing. Cristian Chilipirea – [cristian.chilipirea@mta.ro](mailto:cristian.chilipirea@mta.ro)







# Rank Sort

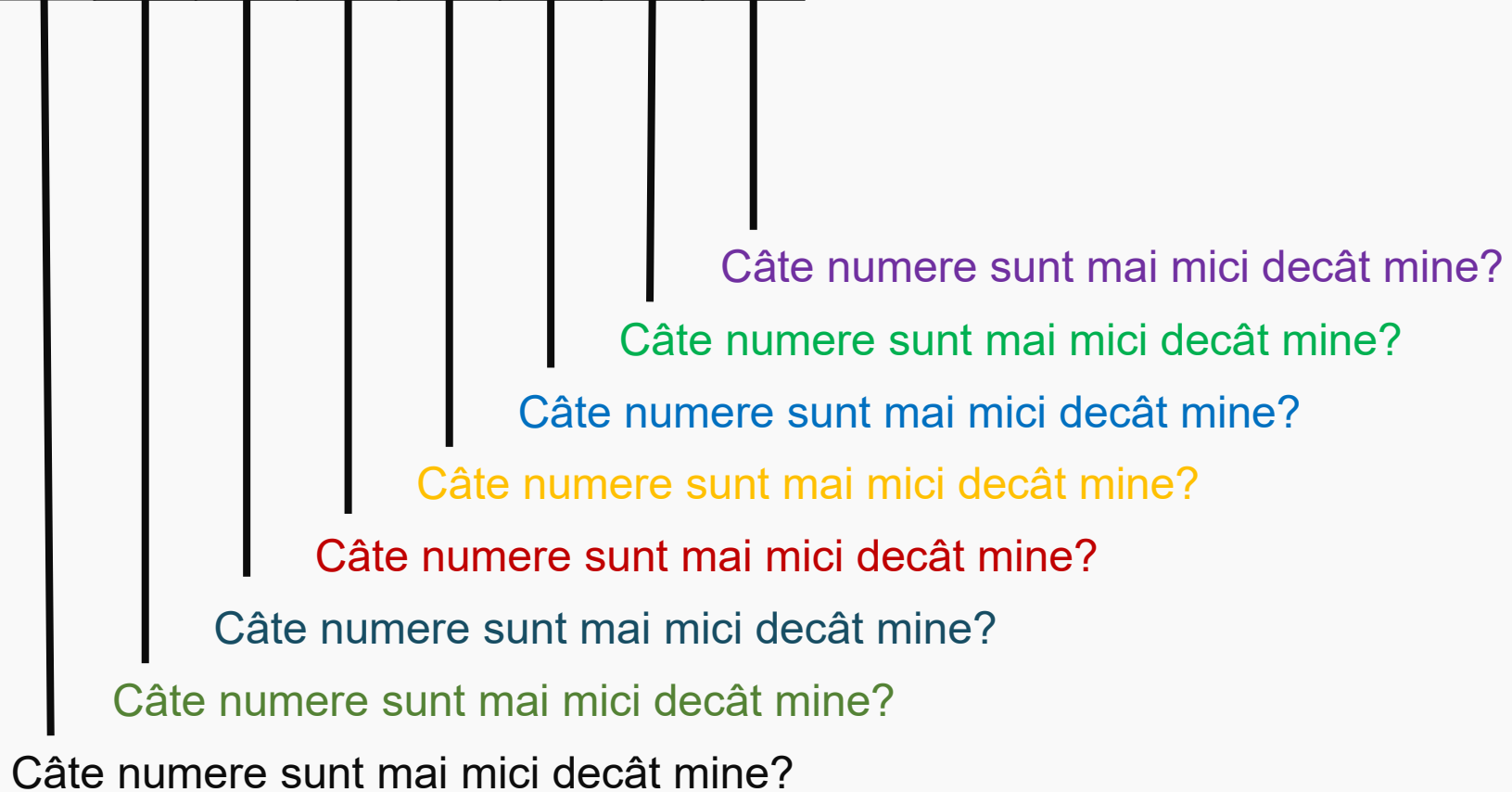
9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---

1	2	4	5	6	6	7	9
---	---	---	---	---	---	---	---



# Rank Sort

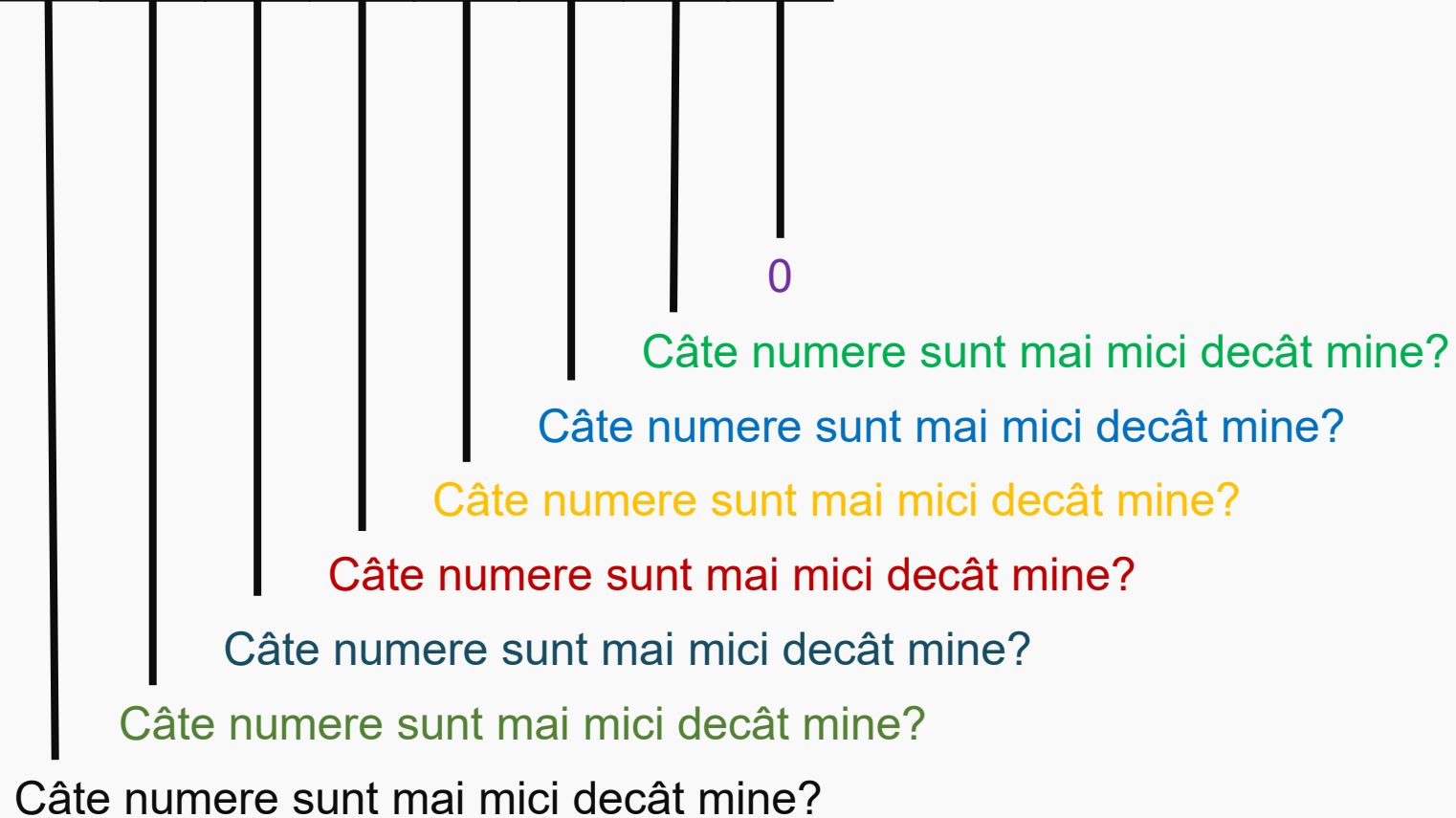
9	4	2	7	3	5	6	1
---	---	---	---	---	---	---	---





# Rank Sort

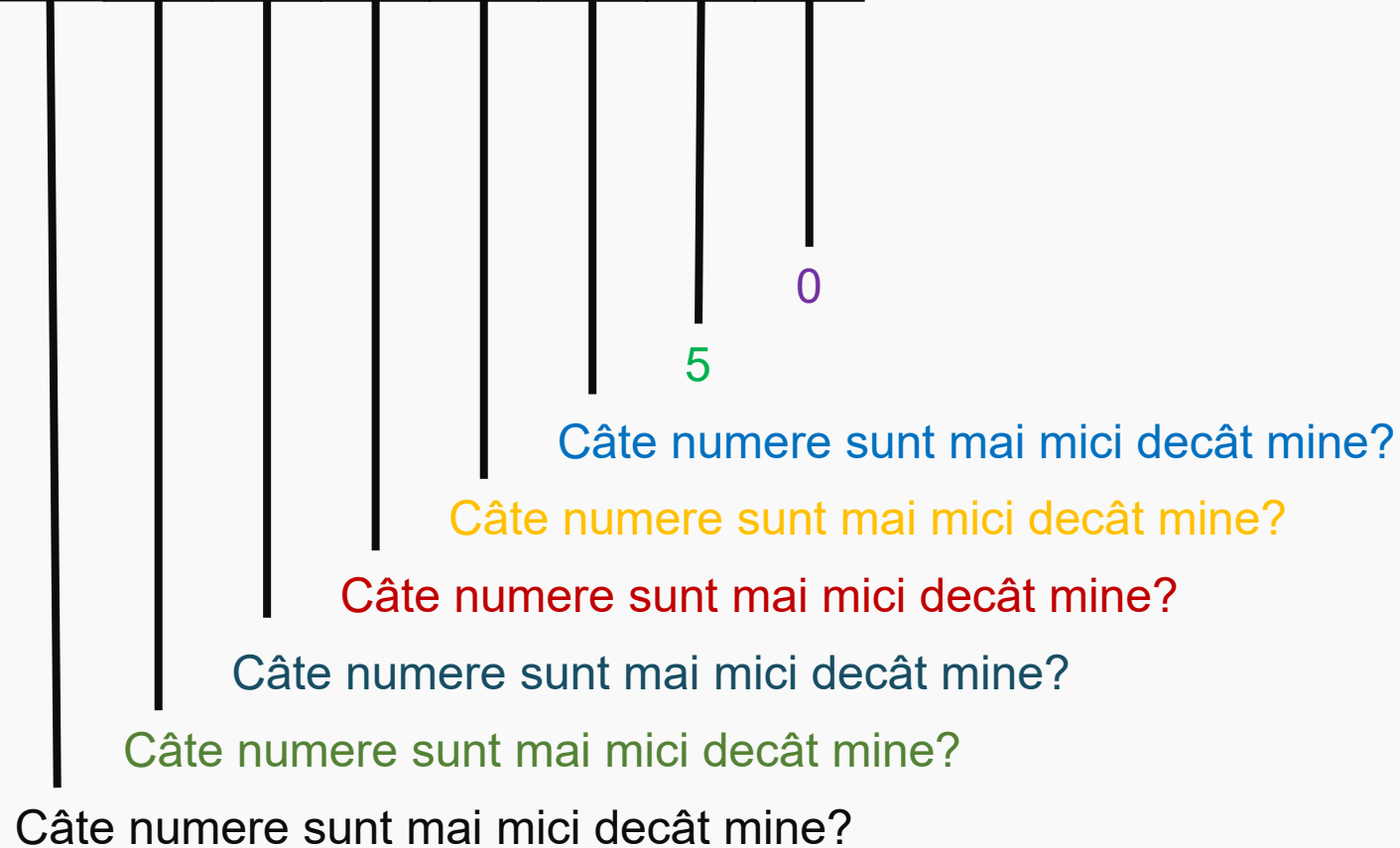
9	4	2	7	3	5	6	1
---	---	---	---	---	---	---	---





# Rank Sort

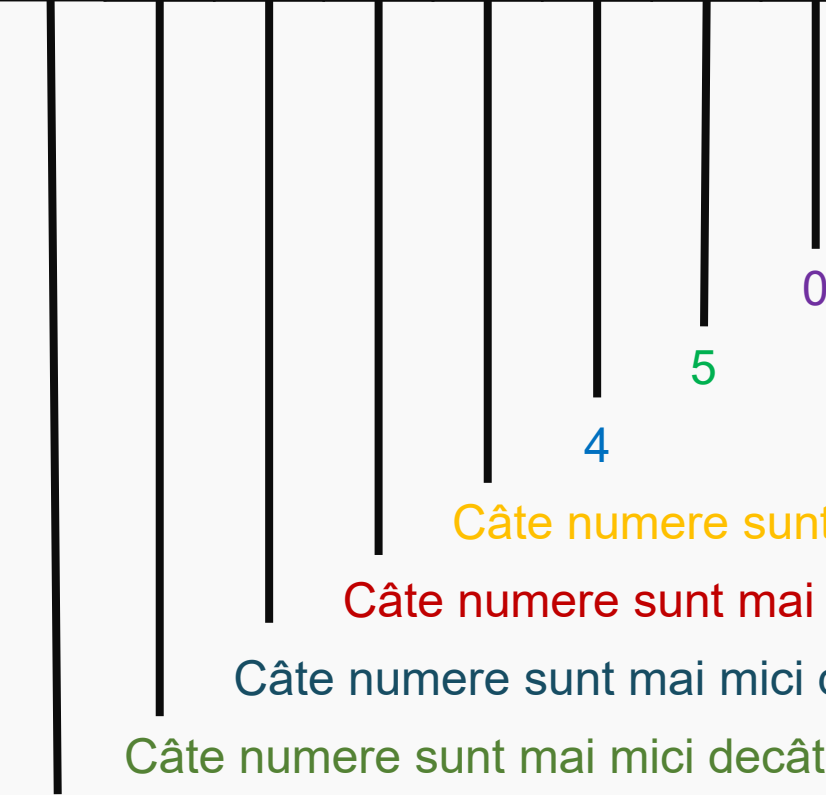
9	4	2	7	3	5	6	1
---	---	---	---	---	---	---	---





# Rank Sort

9	4	2	7	3	5	6	1
---	---	---	---	---	---	---	---



Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

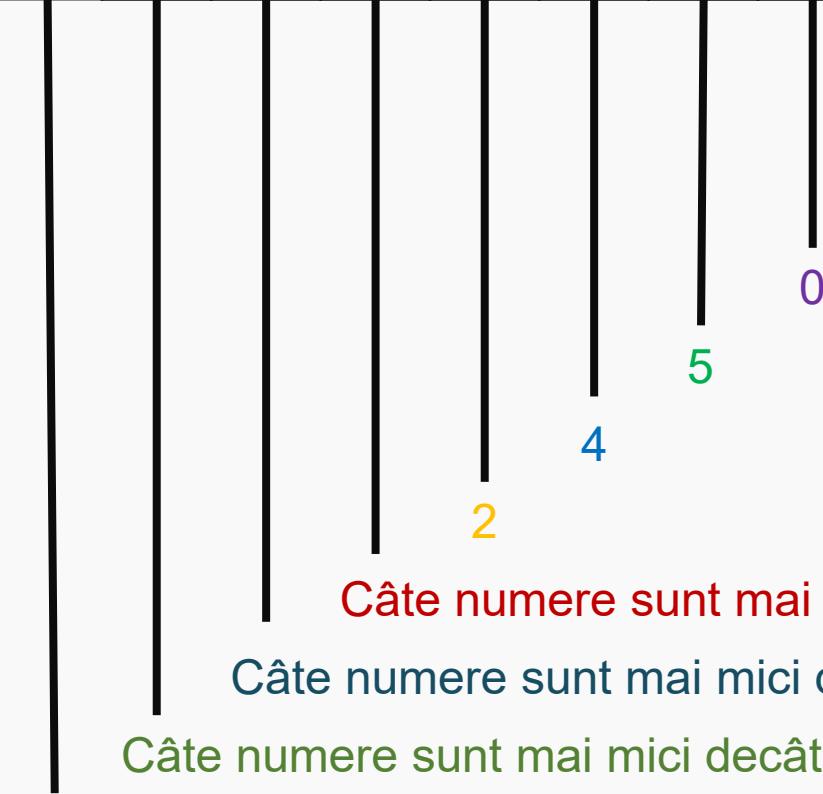
Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?



# Rank Sort

9	4	2	7	3	5	6	1
---	---	---	---	---	---	---	---



Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

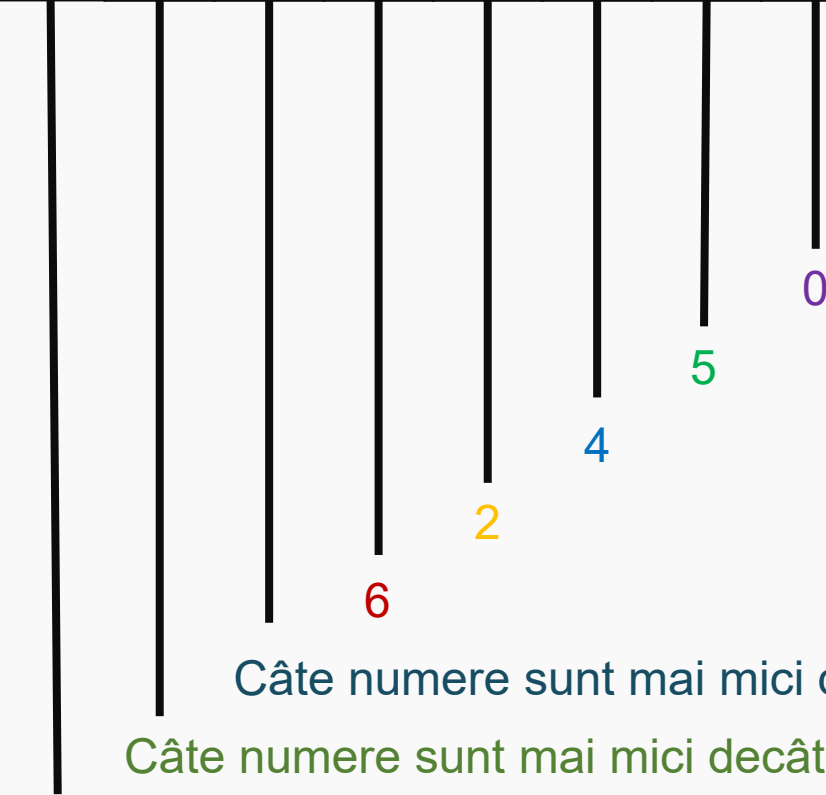
Câte numere sunt mai mici decât mine?





# Rank Sort

9	4	2	7	3	5	6	1
---	---	---	---	---	---	---	---



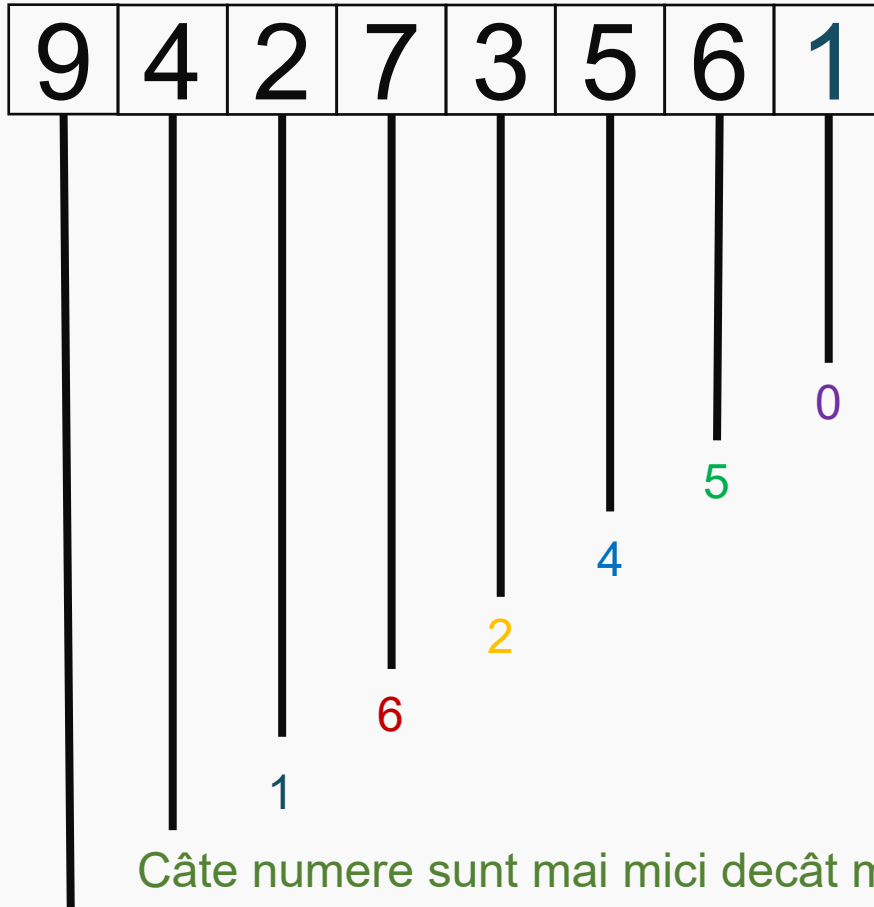
Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?



# Rank Sort

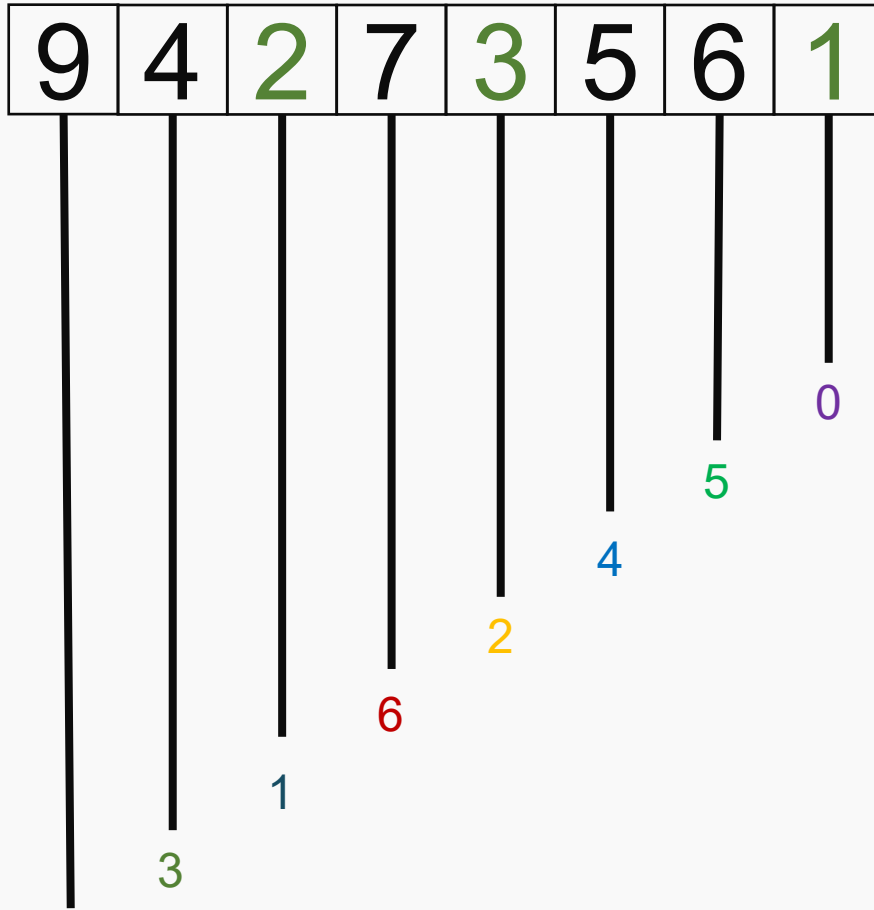


Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?



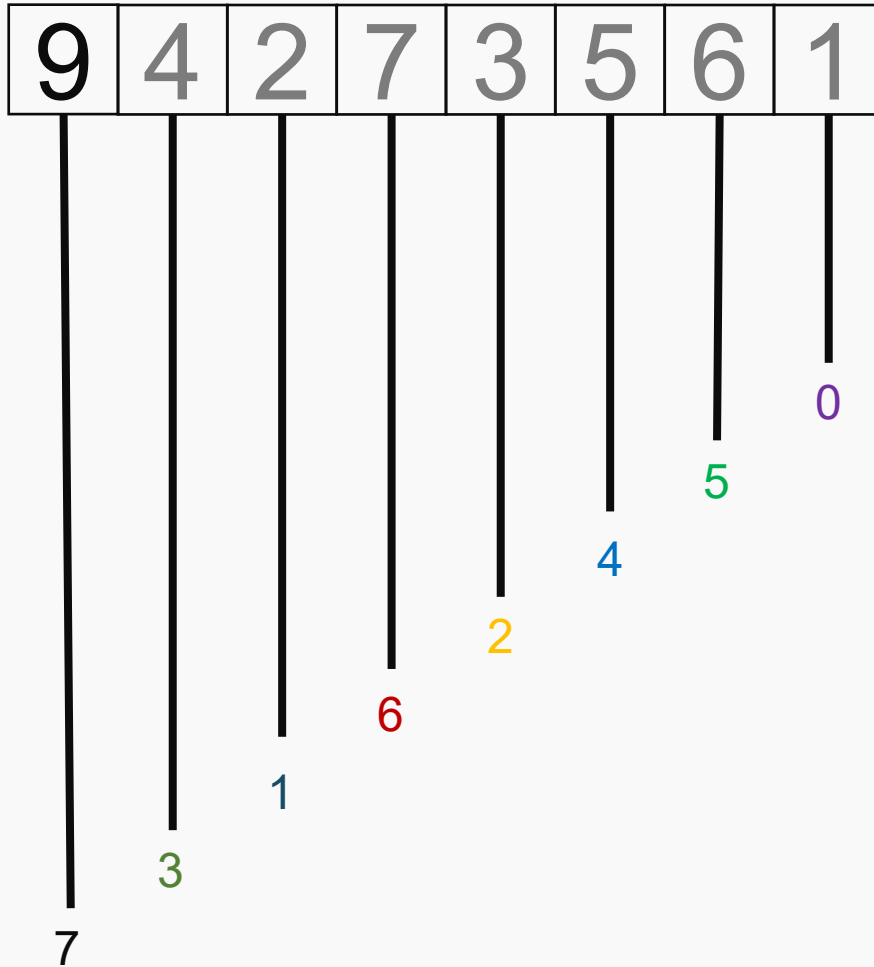
# Rank Sort



Câte numere sunt mai mici decât mine?

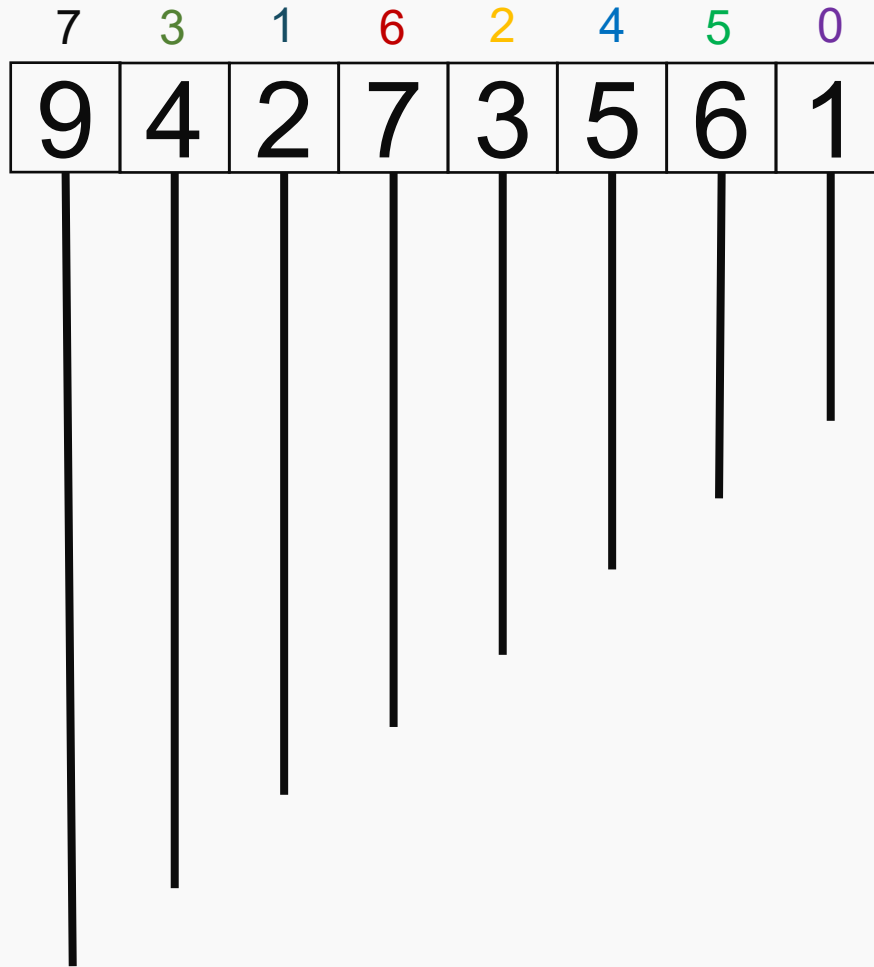


# Rank Sort





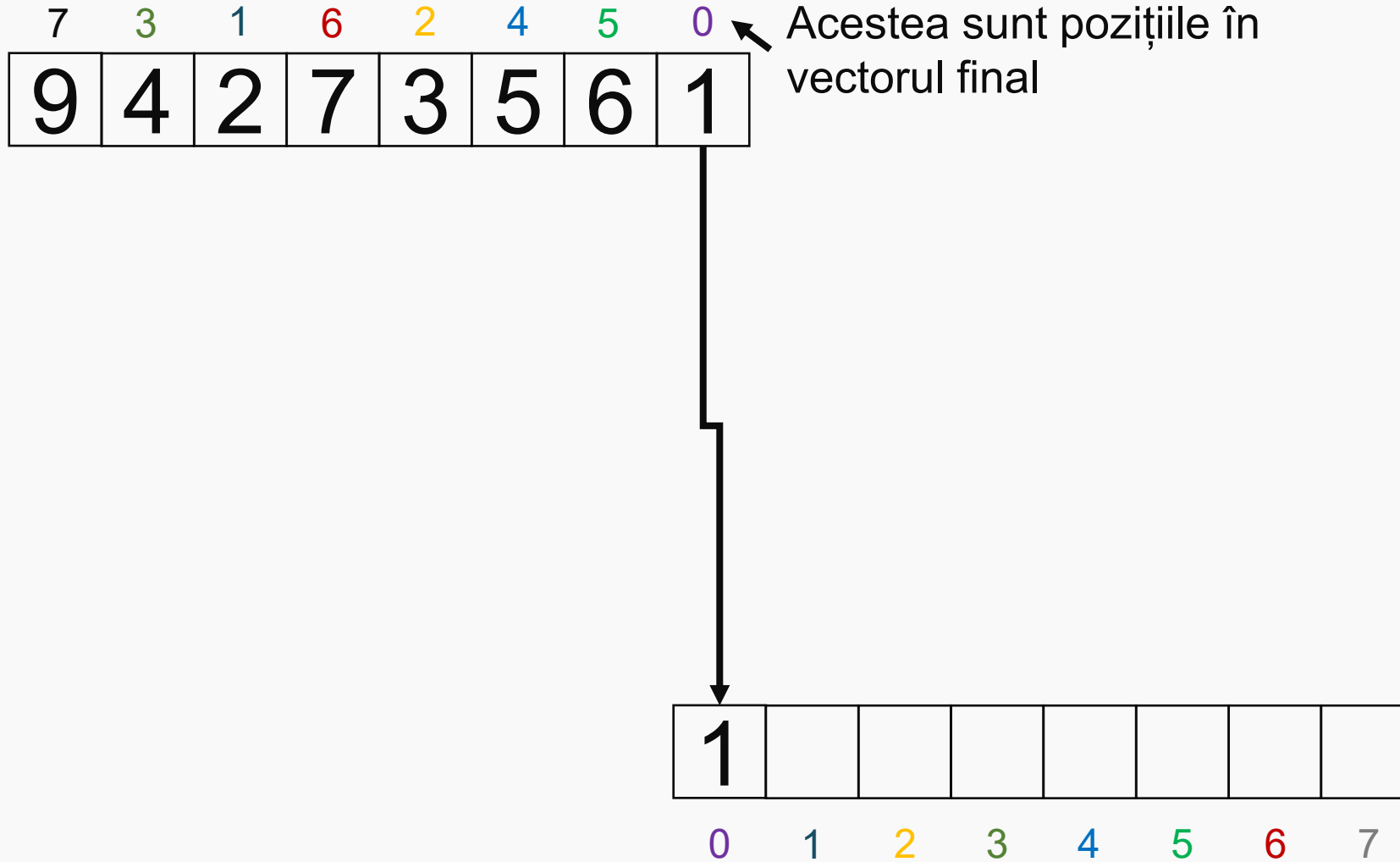
# Rank Sort



Acestea sunt pozițiile în vectorul final

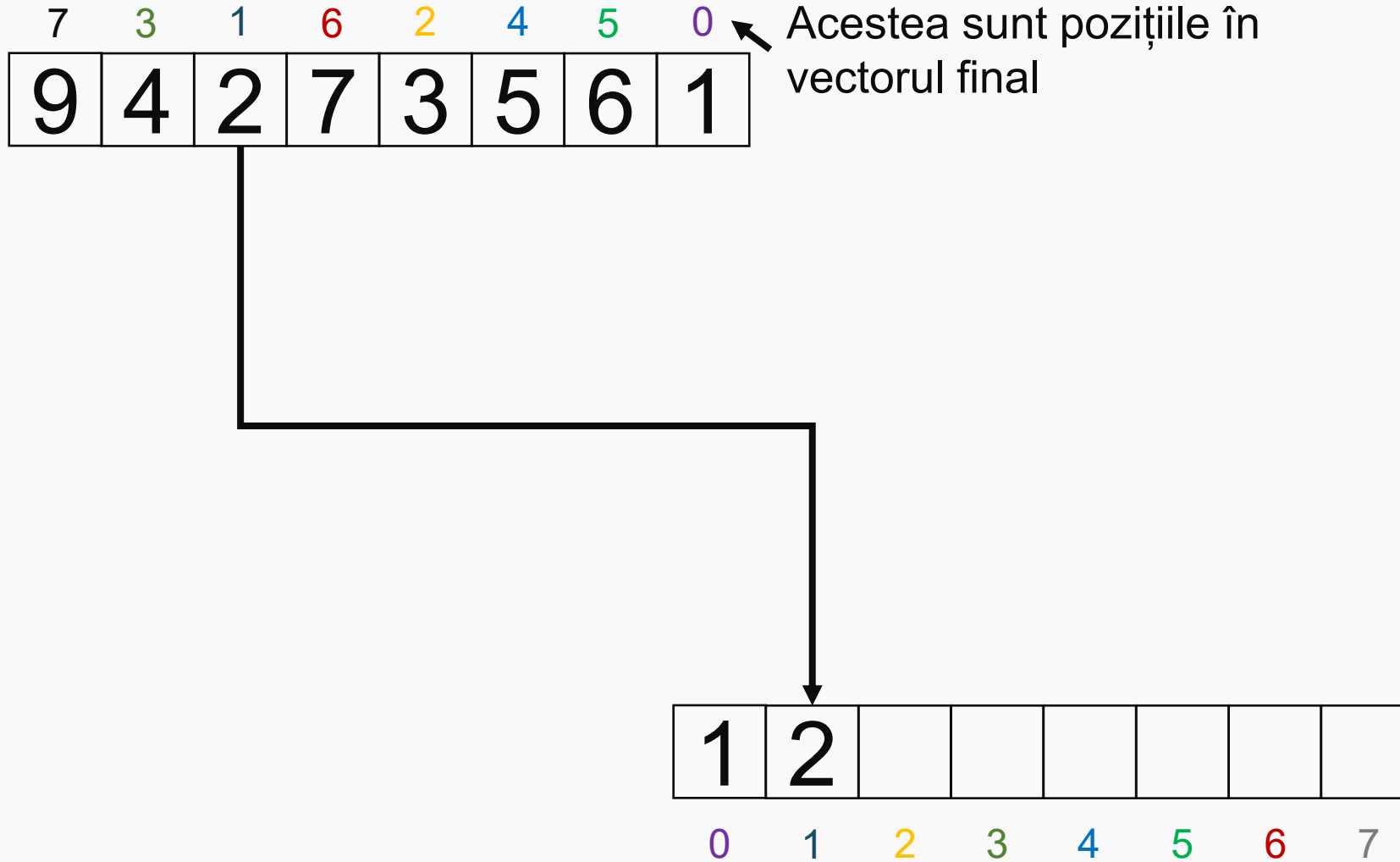


# Rank Sort





# Rank Sort





# Rank Sort

7	3	1	6	2	4	5	0
9	4	2	7	3	5	6	1

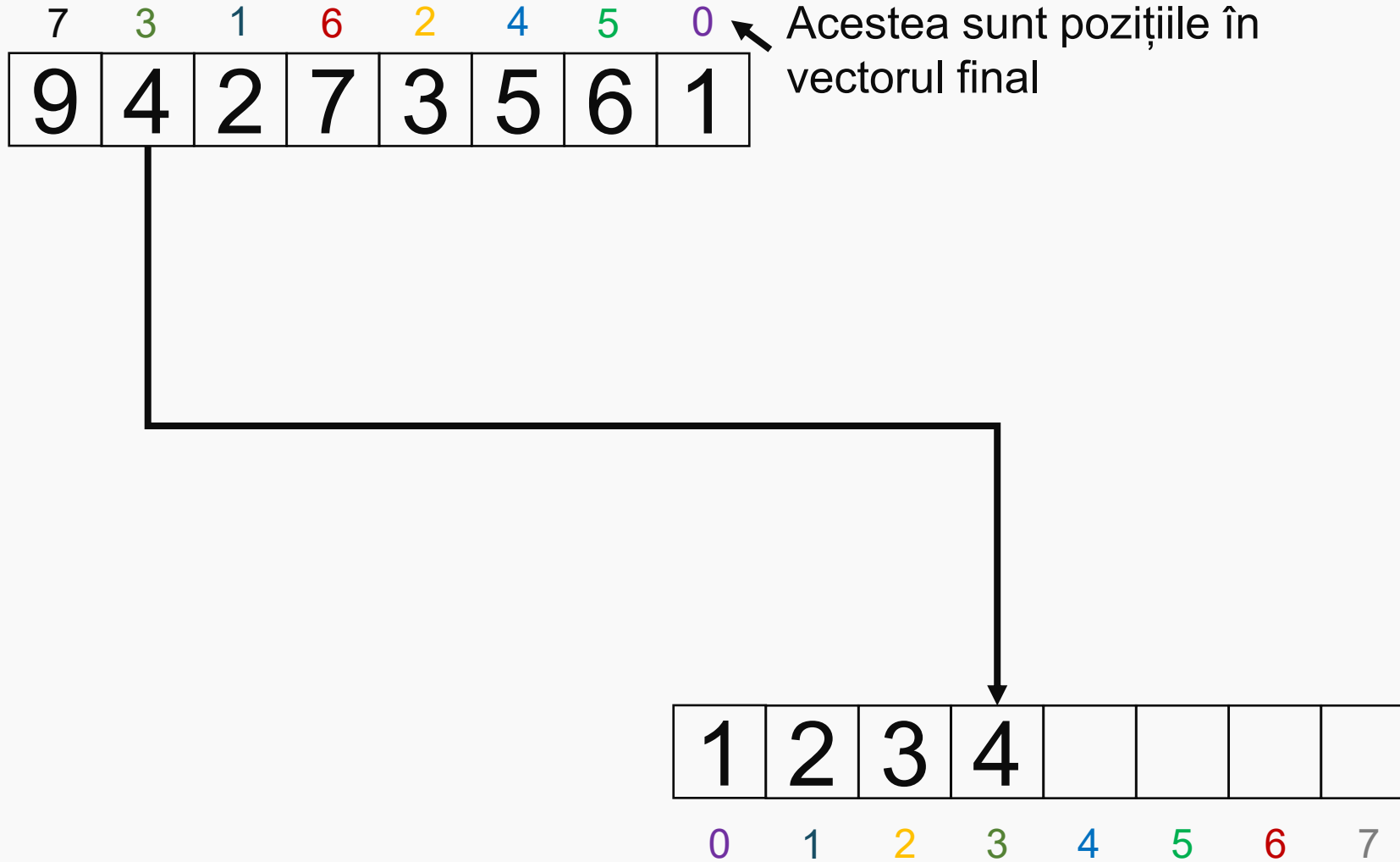
Acestea sunt pozițiile în  
vectorul final

1	2	3					
0	1	2	3	4	5	6	7





# Rank Sort





# Rank Sort

7	3	1	6	2	4	5	0
9	4	2	7	3	5	6	1

Acestea sunt pozițiile în  
vectorul final

1	2	3	4	5			
0	1	2	3	4	5	6	7



# Rank Sort

7	3	1	6	2	4	5	0
9	4	2	7	3	5	6	1

Acestea sunt pozițiile în  
vectorul final

1	2	3	4	5	6		
0	1	2	3	4	5	6	7



# Rank Sort

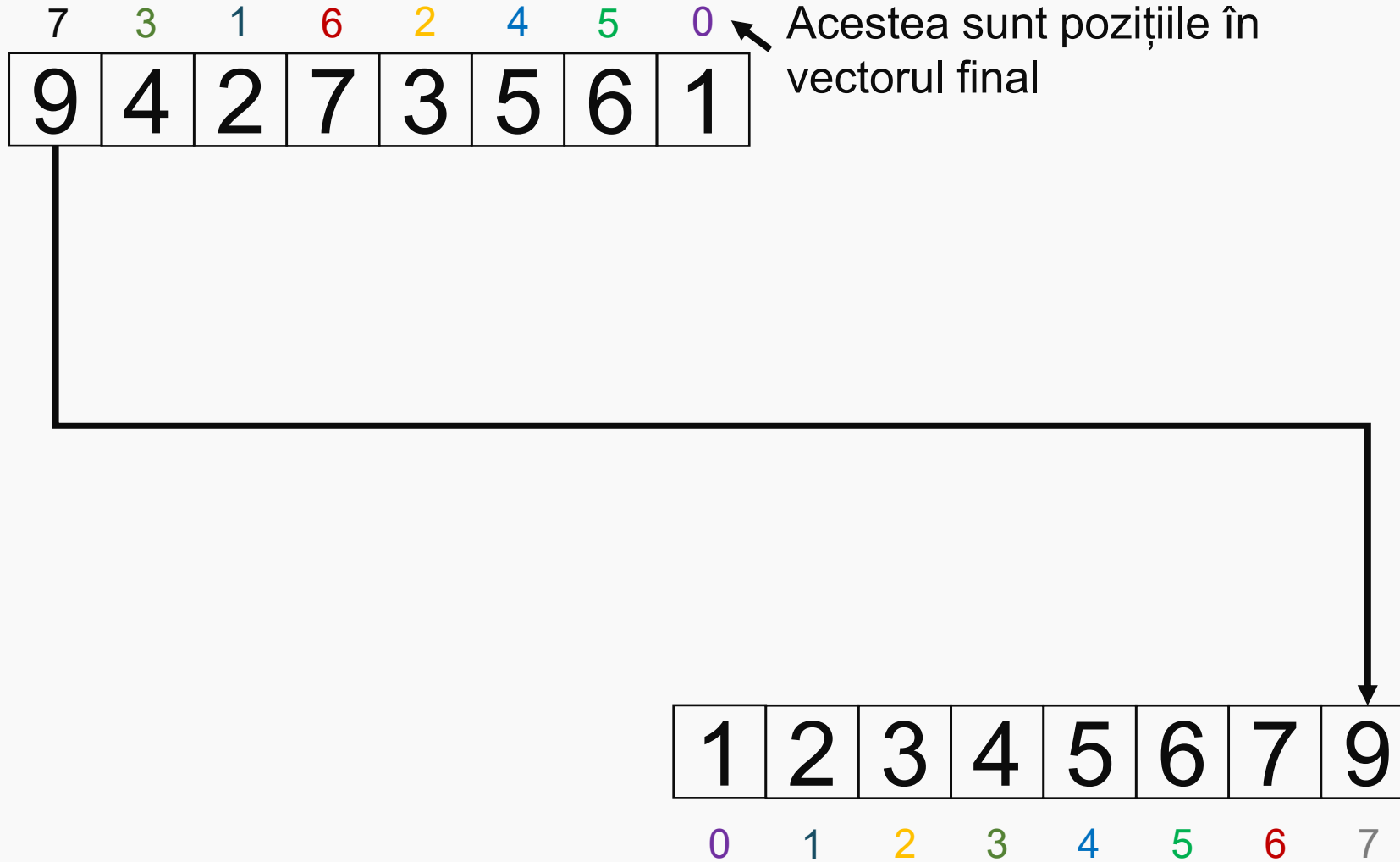
7	3	1	6	2	4	5	0
9	4	2	7	3	5	6	1

Acestea sunt pozițiile în  
vectorul final

1	2	3	4	5	6	7	
0	1	2	3	4	5	6	7



# Rank Sort





# Rank Sort Paralel

9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---

1	2	4	5	6	6	7	9
---	---	---	---	---	---	---	---



# Rank Sort Paralel

9	4	2	7	3	5	6	1
---	---	---	---	---	---	---	---

**Răspunsul la toate  
întrebările poate fi  
determinat în paralel**

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

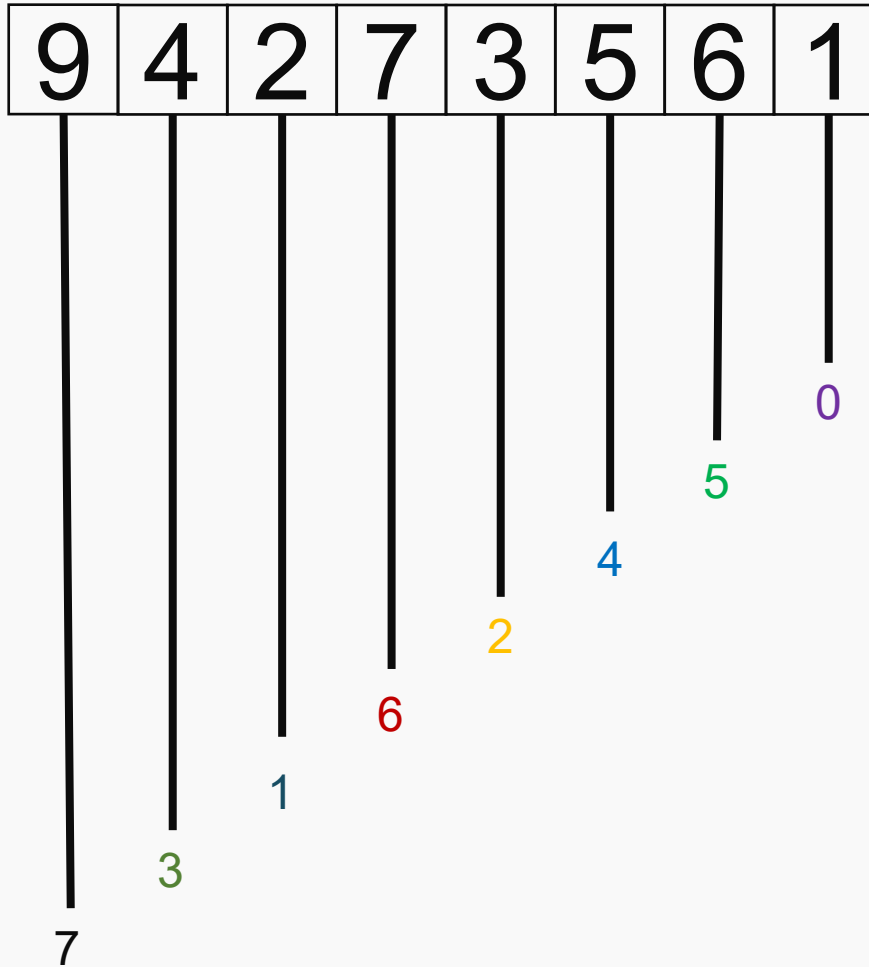
Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?

Câte numere sunt mai mici decât mine?



# Rank Sort Paralel



**Răspunsul la toate  
întrebările poate fi  
determinat în paralel**



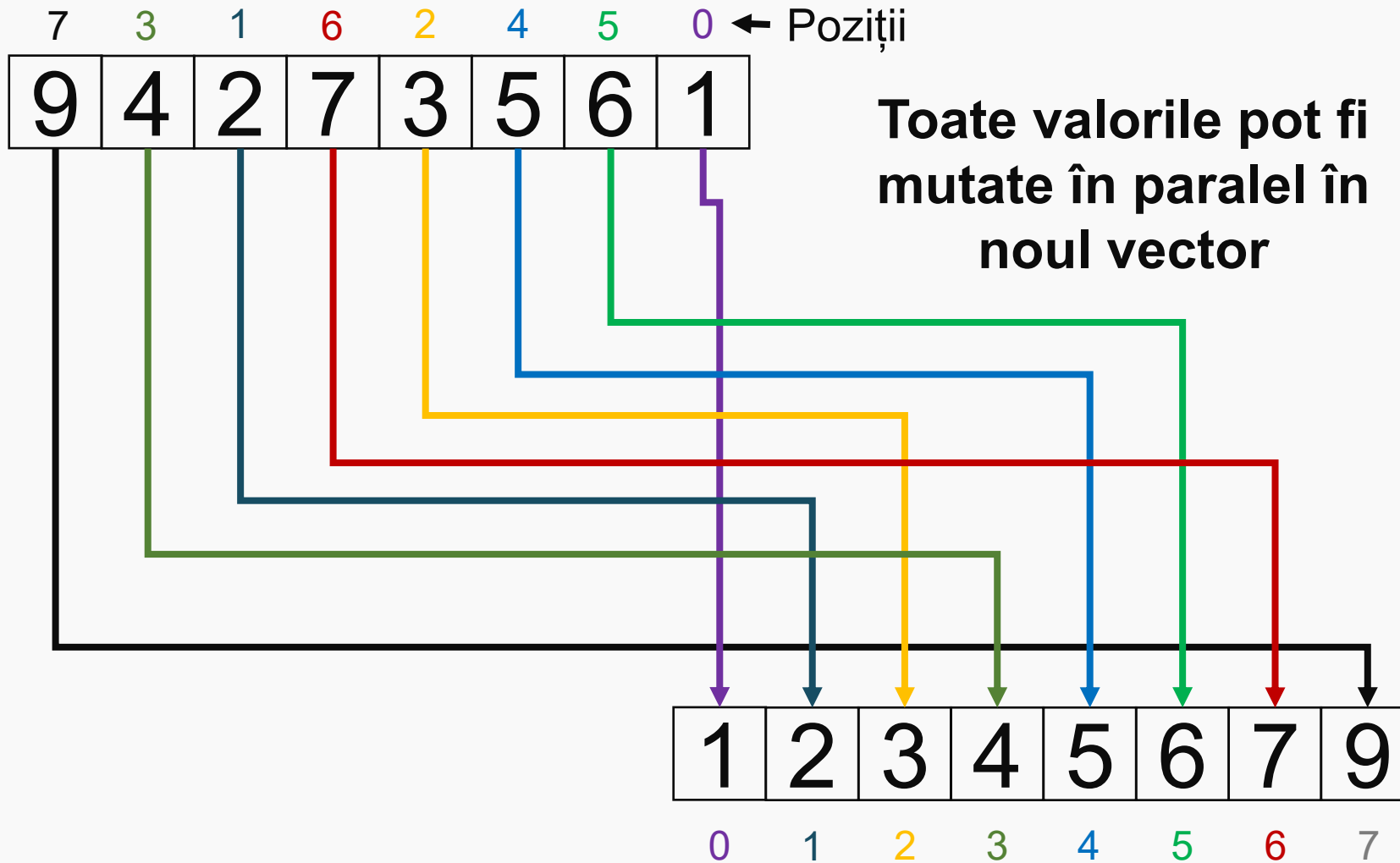


# Rank Sort Paralel





# Rank Sort Paralel







# Algoritm merge

2	3	4	5	7
---	---	---	---	---

1	2	4	4	6
---	---	---	---	---

Avem ca intrare două  
liste **sortate** dorim să  
le unim într-o listă  
**sortată**

1	2	2	3	4	4	4	5	6	7
---	---	---	---	---	---	---	---	---	---



# Algoritm merge

2	3	4	5	7
---	---	---	---	---

1	2	4	4	6
---	---	---	---	---

Soluție:

Se extrage mereu cel mai mic element  
(Garantat să fie pe prima poziție în  
una din cele două liste)

Complexitate:  $O(N)$

1	2	2	3	4	4	4	5	6	7
---	---	---	---	---	---	---	---	---	---



# Algorithm merge

2	3	4	5	7
1	2	4	4	6

---



# Algorithm merge

2	3	4	5	7
	2	4	4	6

---

1
---



# Algorithm merge

3	4	5	7
2	4	4	6

---

1	2
---	---





# Algoritm merge

3	4	5	7
4	4	6	

---

1	2	2
---	---	---



# Algoritm merge

4	5	7
4	4	6

---

1	2	2	3
---	---	---	---



# Algoritm merge

5	7	
4	4	6

1	2	2	3	4
---	---	---	---	---



# Algoritm merge

5	7
---	---

4	6
---	---



1	2	2	3	4	4
---	---	---	---	---	---



# Algoritm merge

5	7
6	

---

1	2	2	3	4	4	4
---	---	---	---	---	---	---



# Algorithm merge

7

6



1	2	2	3	4	4	4	5
---	---	---	---	---	---	---	---



# Algorithm merge

7

---

1	2	2	3	4	4	4	5	6
---	---	---	---	---	---	---	---	---



# Algorithm merge

---

1	2	2	3	4	4	4	5	6	7
---	---	---	---	---	---	---	---	---	---







# Algoritm merge

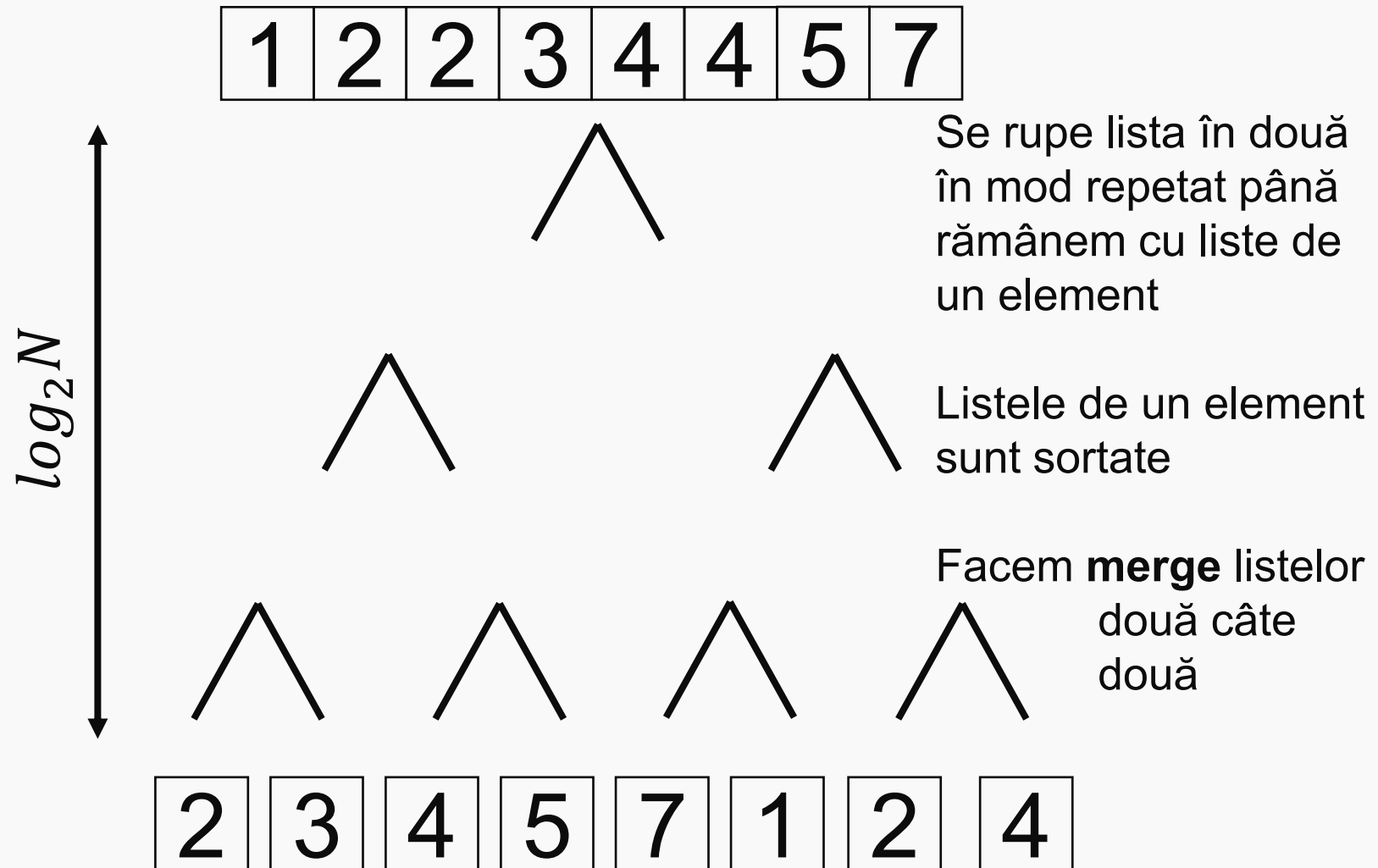


Folosim acest semn pentru a  
reprezenta operația **MERGE**





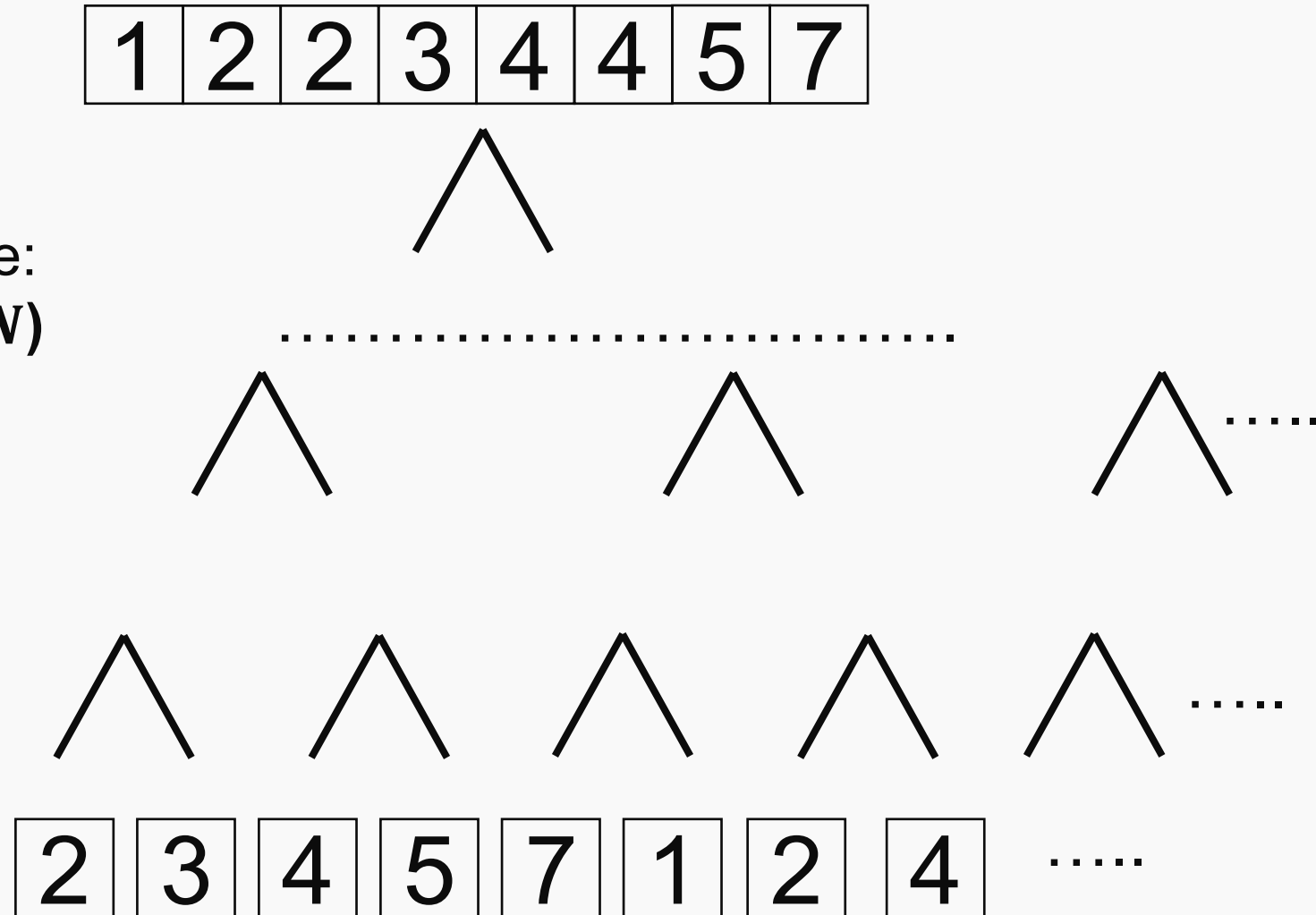
# Merge sort





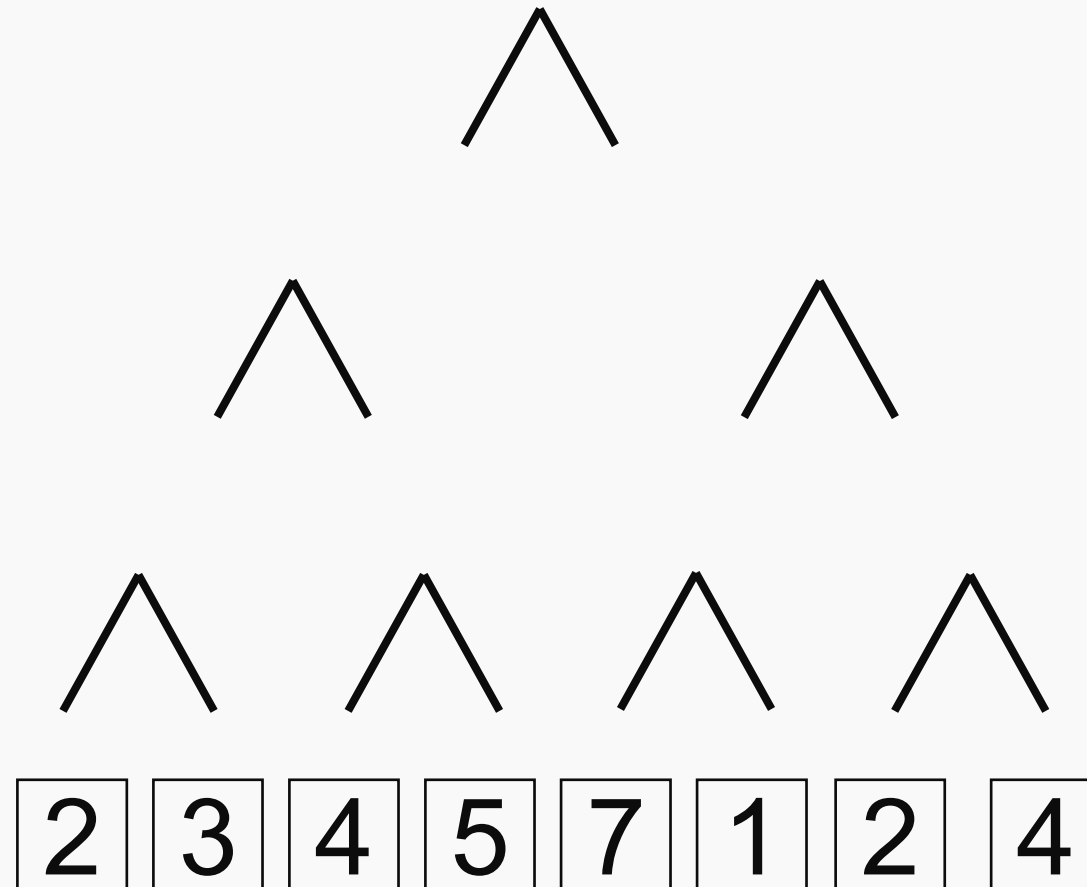
# Merge sort

Complexitate:  
 $O(N * \log_2 N)$



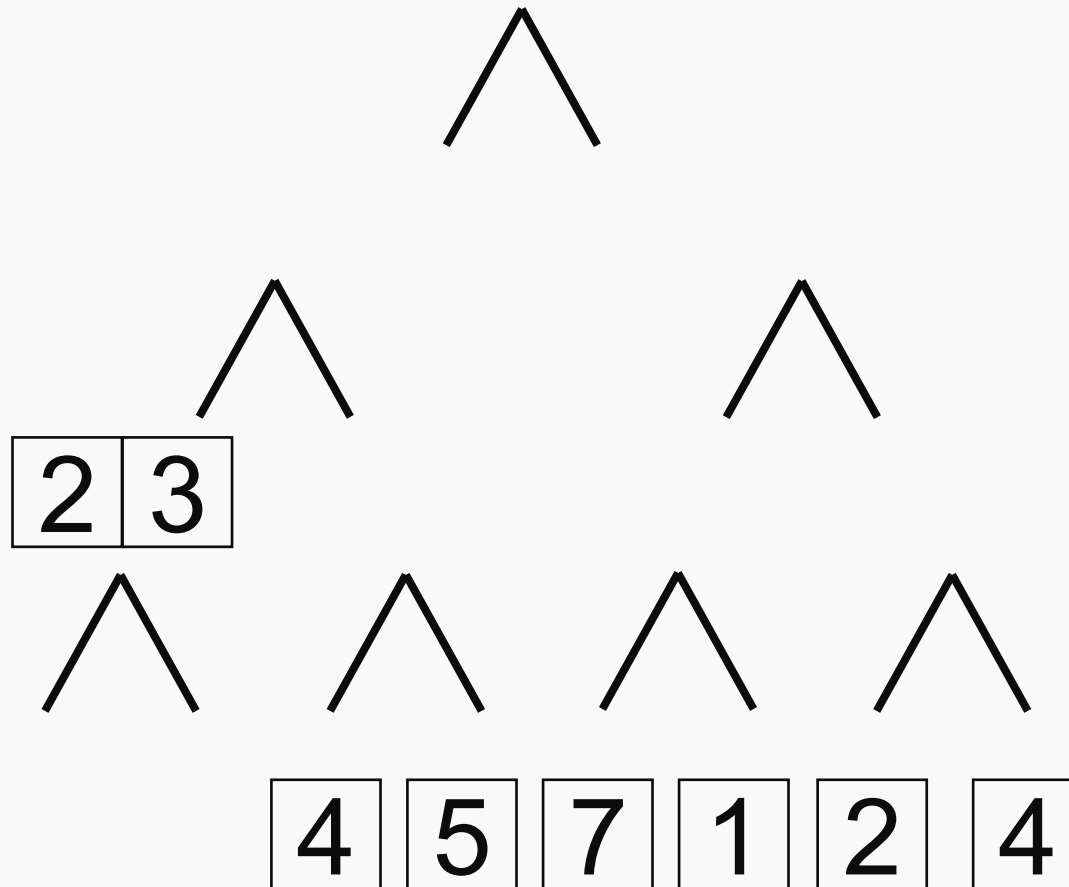


# Merge sort



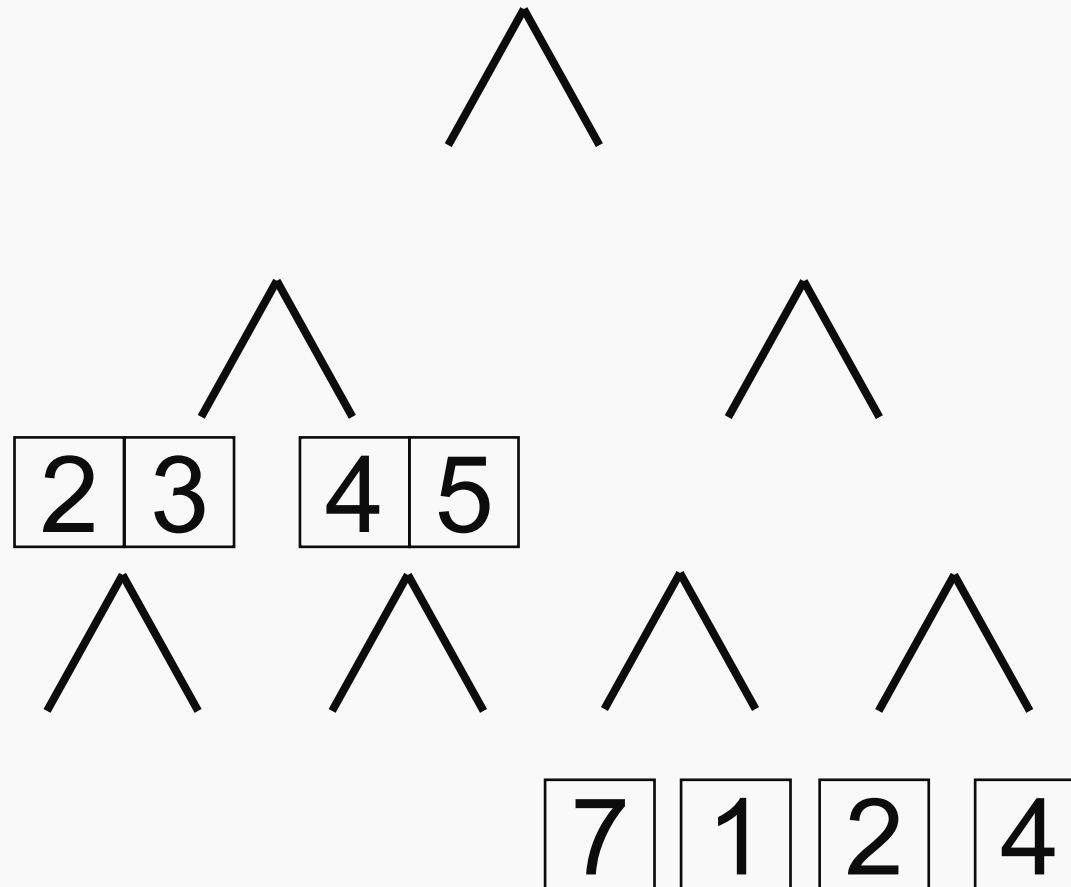


# Merge sort



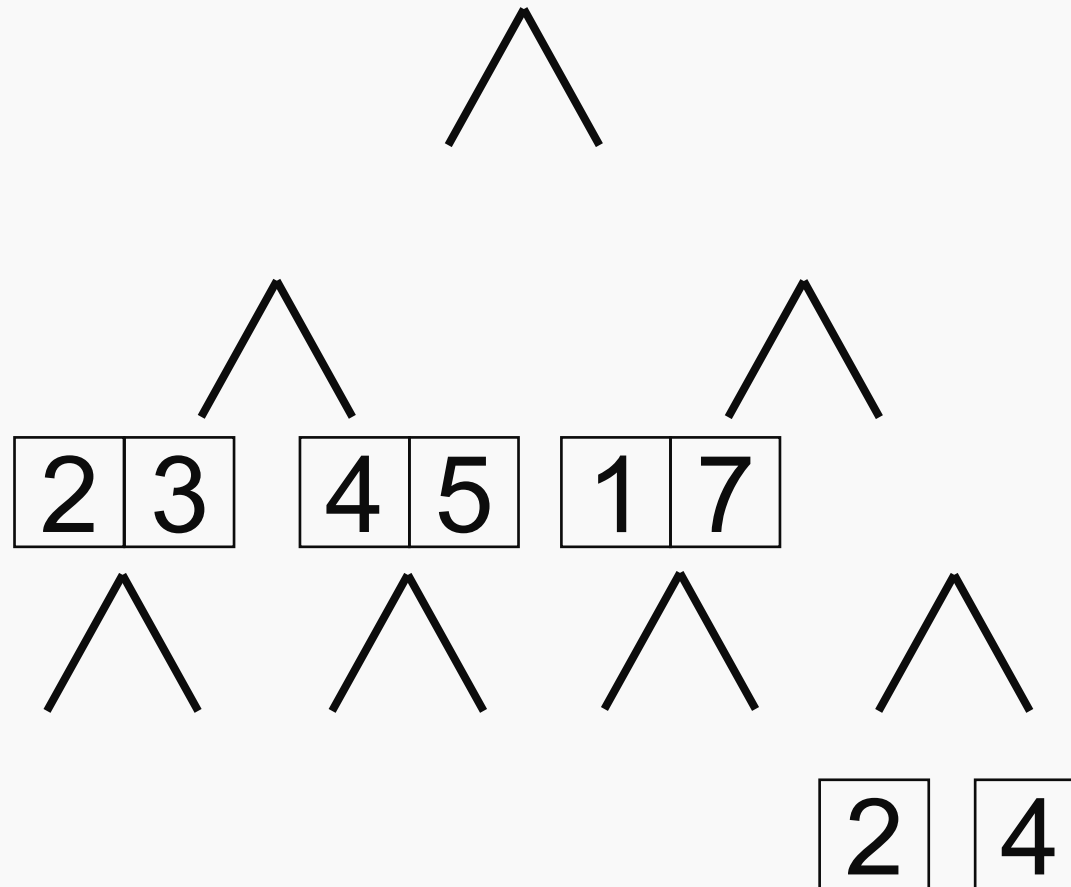


# Merge sort





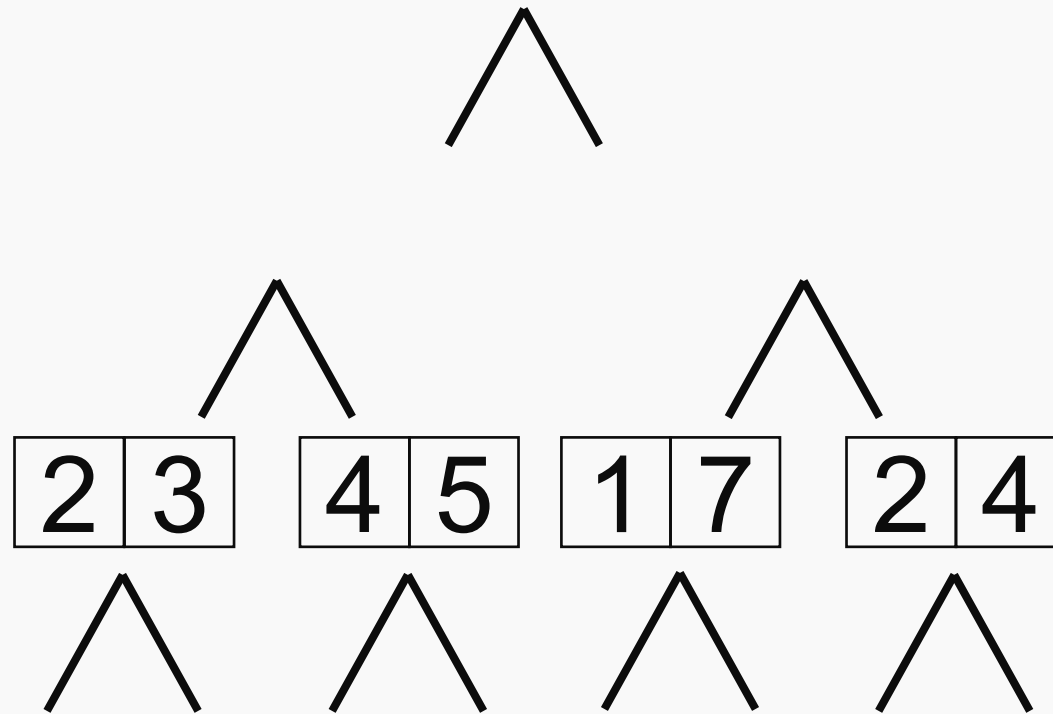
# Merge sort





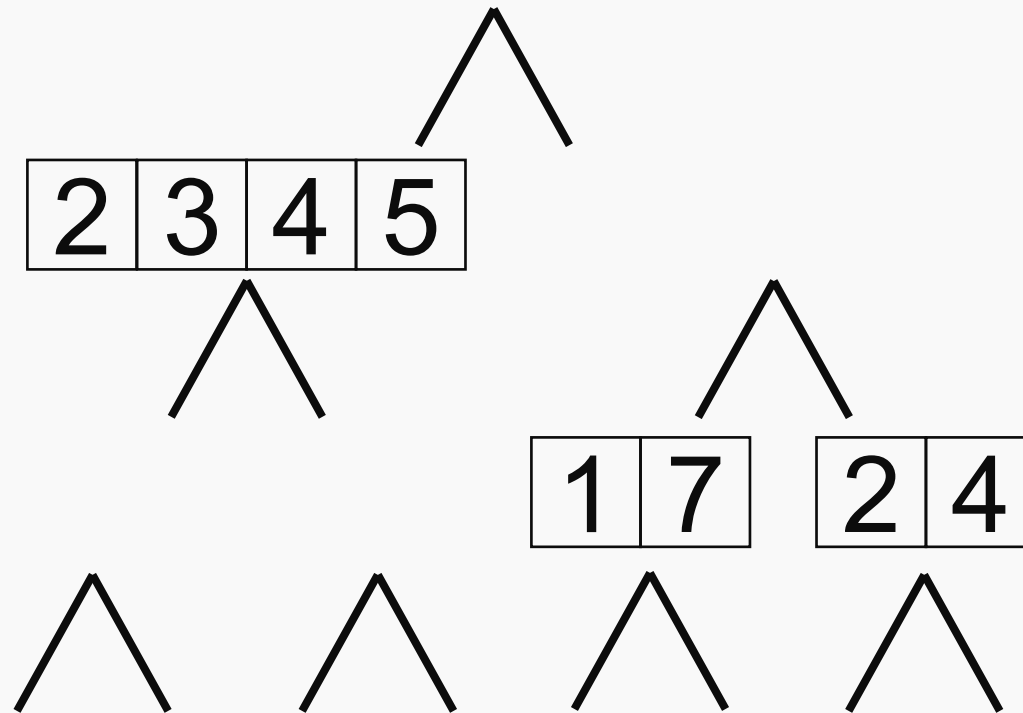


# Merge sort





# Merge sort







# Merge sort

1	2	2	3	4	4	5	7
---	---	---	---	---	---	---	---

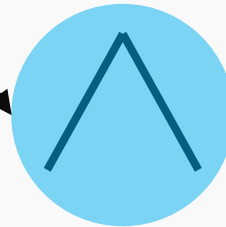




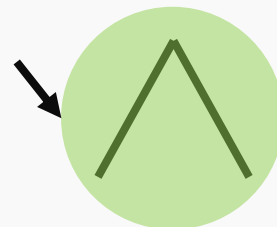


# Merge sort paralel

Operațiile  
**albastre**  
pot fi  
executate în  
paralel





Operațiile  
**verzi** pot fi  
executate în  
paralel

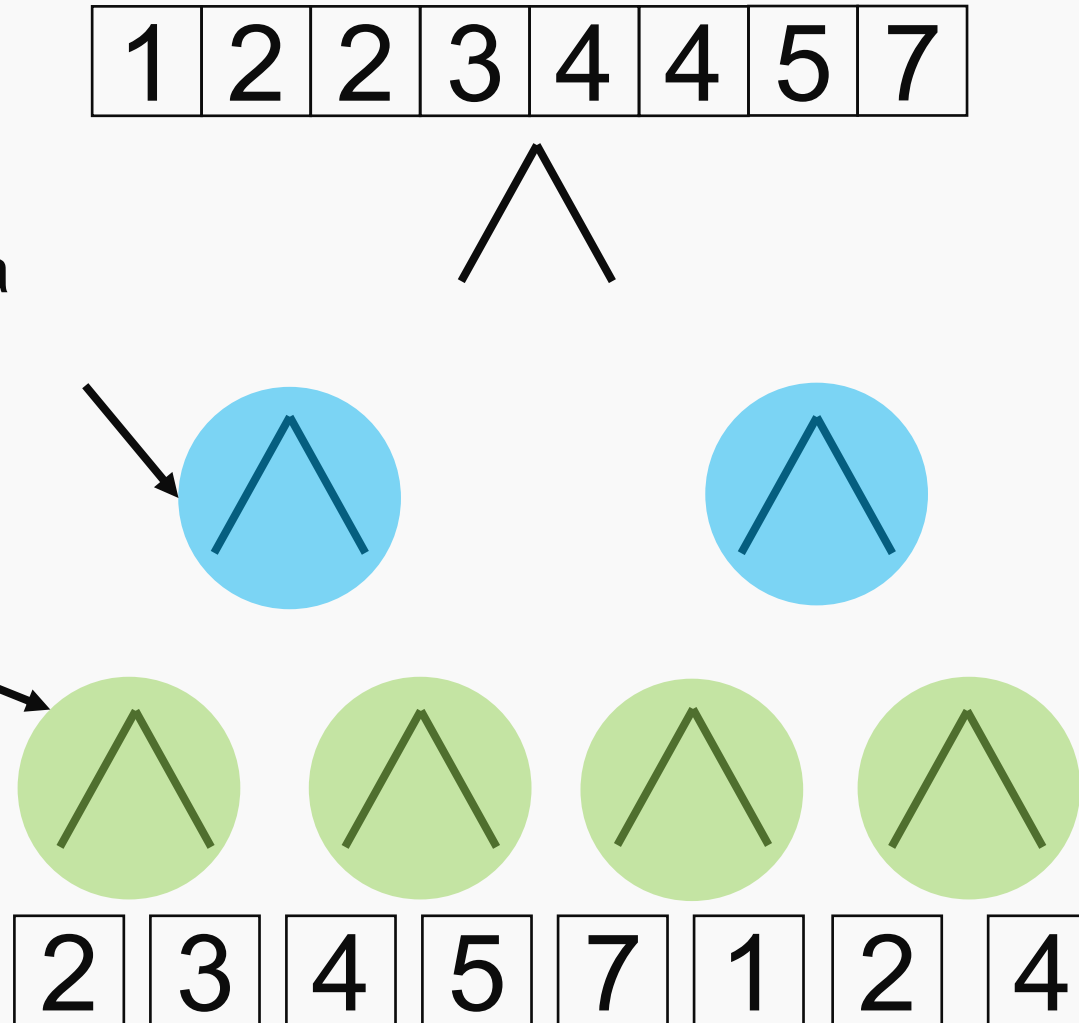




# Merge sort paralel

O operație **verde** cu una **albastră** nu poate fi executată în paralel

Operația   
depinde de  
rezultatul  
operațiilor 



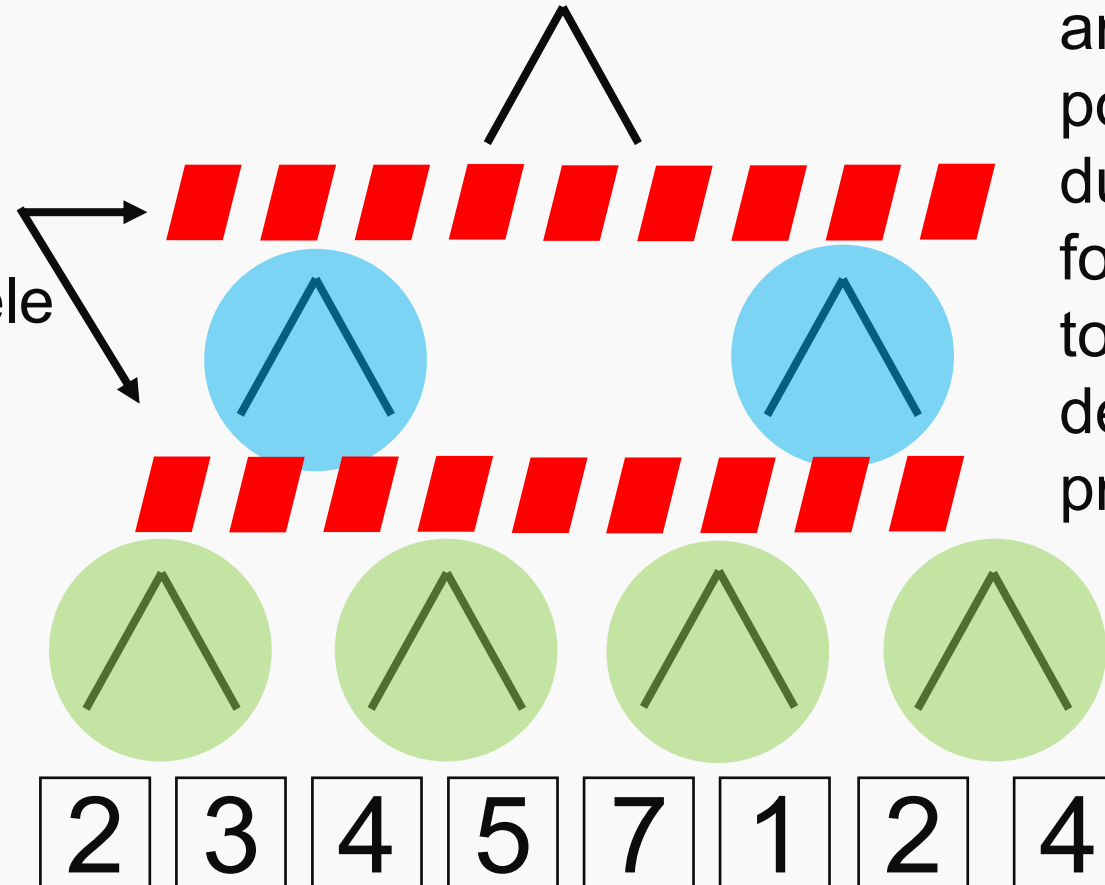


# Merge sort paralel

Soluție:

**Barrier**

Între nivelele  
arborelui

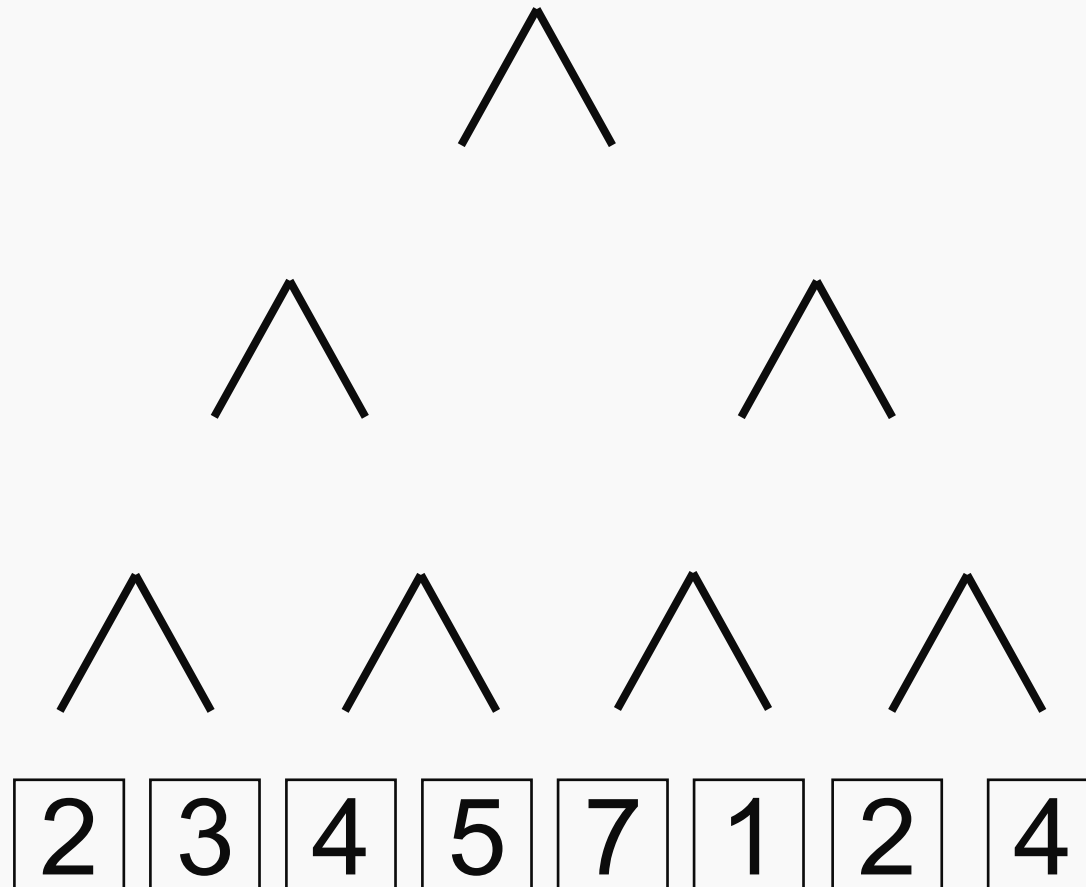


Operațiile după  
un nivel al  
arborelui pot  
porni doar  
după ce au  
fost terminate  
toate operațiile  
de pe nivelul  
precedent.



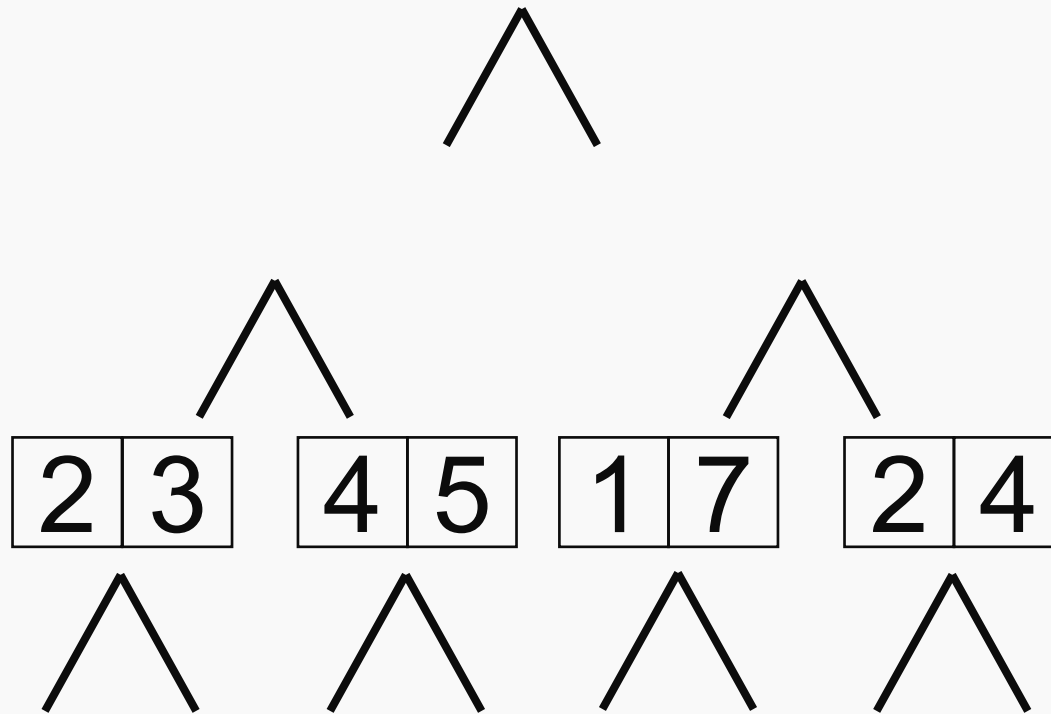


# Merge sort paralel



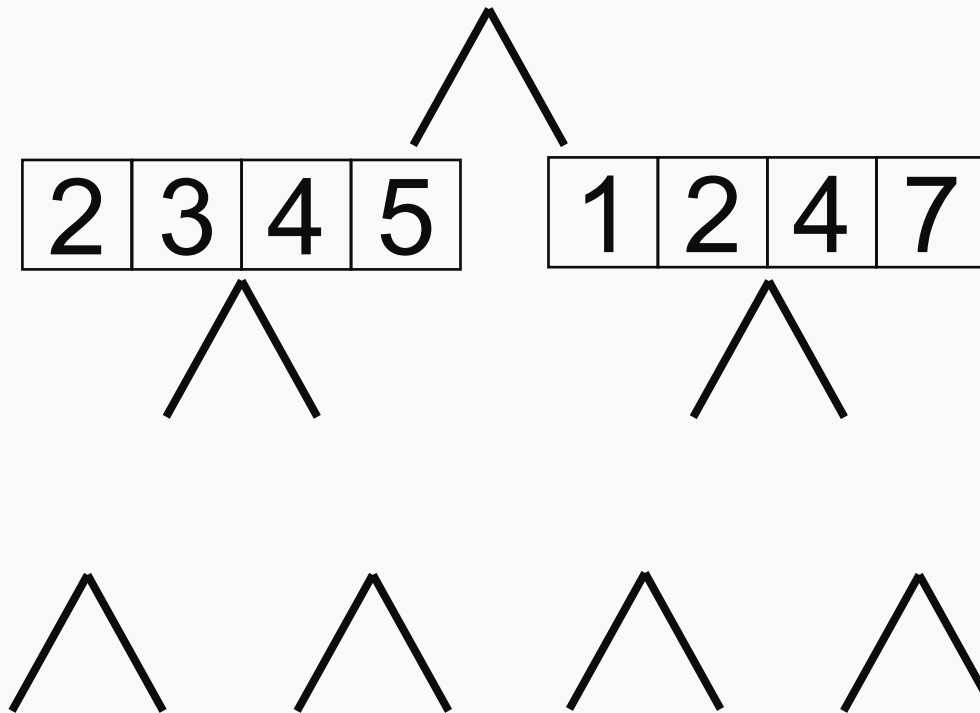


# Merge sort paralel





# Merge sort paralel





# Merge sort paralel

1	2	2	3	4	4	5	7
---	---	---	---	---	---	---	---



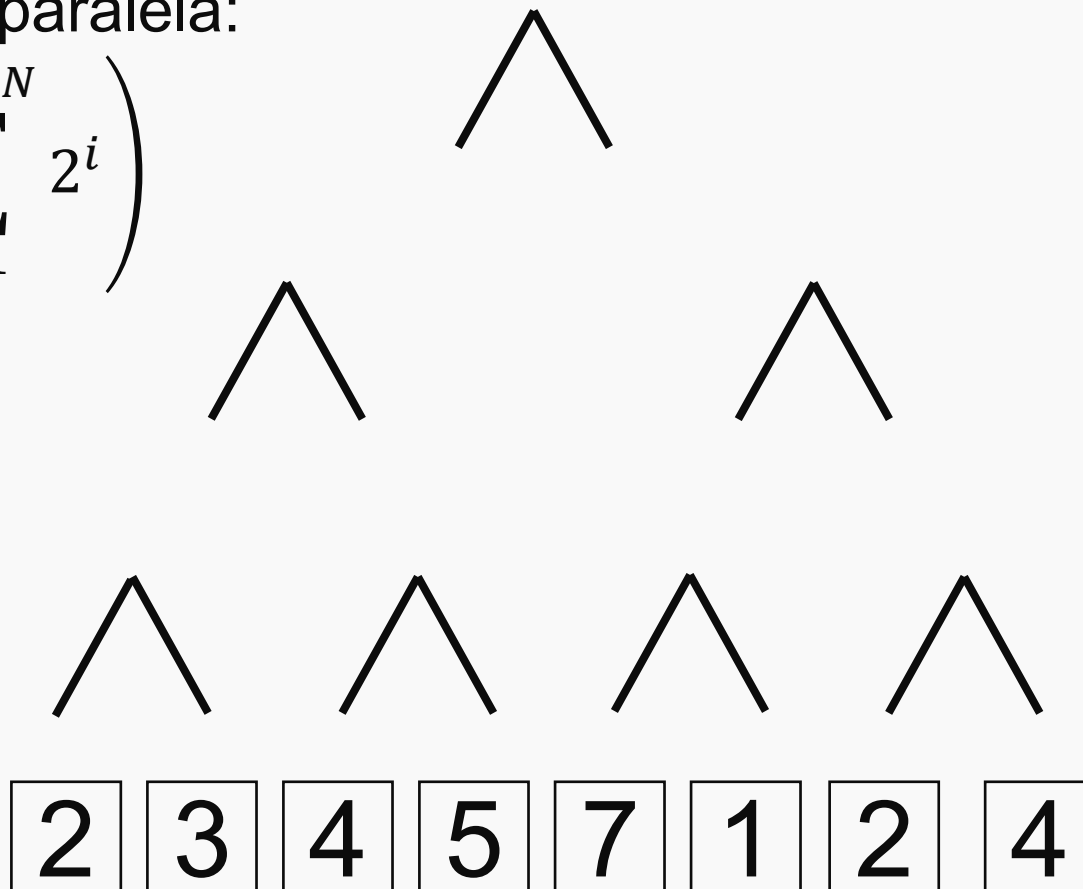


# Merge sort paralel - complexitate

Complexitate paralelă:

$$O\left(\sum_{i=1}^{\log_2 N} 2^i\right)$$

Dacă  $P = N$





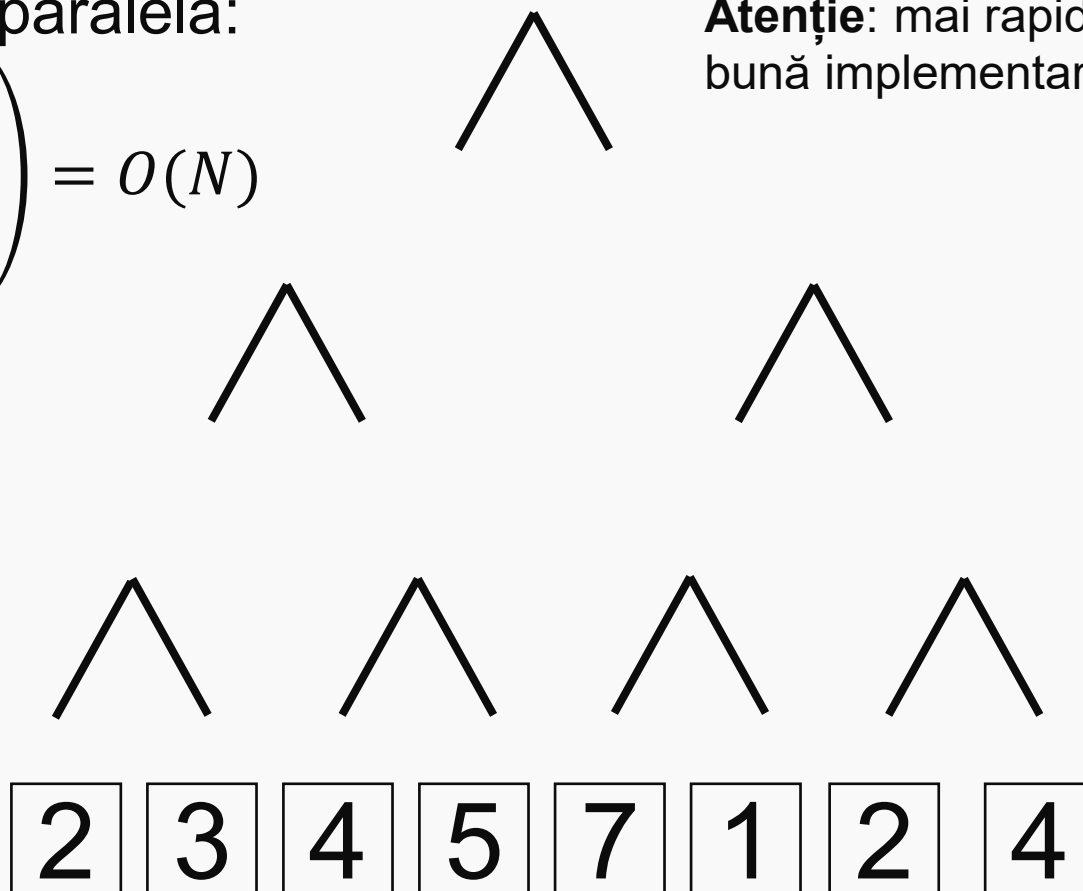
# Merge sort paralel - complexitate

Complexitate paralelă:

$$O\left(\sum_{i=1}^{\log_2 N} 2^i\right) = O(N)$$

Dacă  $P = N$

**Atenție:** mai rapid decât cea mai bună implementare secvențială





# Merge sort paralel - complexitate

Cea mai bună soluție paralelă: paralelizează și operația merge

Articol

[Parallel Merge Sort – Richard Cole](#)



## Parallel Merge Sort

*Richard Cole*

New York University



**Abstract.** We give a parallel implementation of merge sort on a CREW PRAM that uses  $n$  processors and  $O(\log n)$  time; the constant in the running time is small. We also give a more complex version of the algorithm for the EREW PRAM; it also uses  $n$  processors and  $O(\log n)$  time. The constant in the running time is still moderate, though not as small.

### 1. Introduction

1975]; this procedure merges two sorted arrays, each of length at most  $n$ , in time  $O(\log \log n)$  using a linear number of processors. When used in the obvious way, Valiant's procedure leads to an implementation of merge sort on  $n$  processors using  $O(\log n \log \log n)$  time. More recently, Kruskal [K, 1983] improved this sorting algorithm to obtain a sorting algorithm that runs in time  $O(\log n \log \log n / \log \log \log n)$  on  $n$  processors. (The







# Căutare binară

Căutăm 3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9



# Căutare binară

Căutăm 3

Între pozițiile 0 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9



**$3 < 7$**




# Căutare binară

Căutăm 3

Între pozițiile 0 7

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

  
**3 < 5**




# Căutare binară

Căutăm 3

Între pozițiile 0 3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

  
**3 > 2**




# Căutare binară

Căutăm 3

Între pozițiile 3 3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

  
**3 = 3**  
**end**

**Complexitate  $O(\log_2(N))$**



# Căutare binară

Căutăm 3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

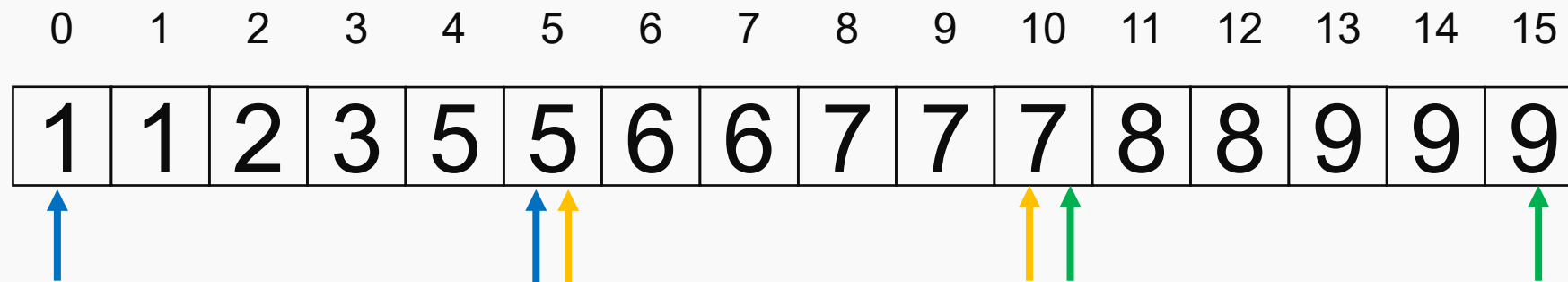




# Căutare paralelă – implementare naivă

Căutăm 3

Între pozițiile 0 15



**Fiecare thread este responsabil de o zonă**





# Căutare paralelă – implementare naivă

Căutăm 3

Între pozițiile 0 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9



Elementul căutat este în bucata mea





# Căutare paralelă – implementare naivă

Căutăm 3

Între pozițiile 0 15



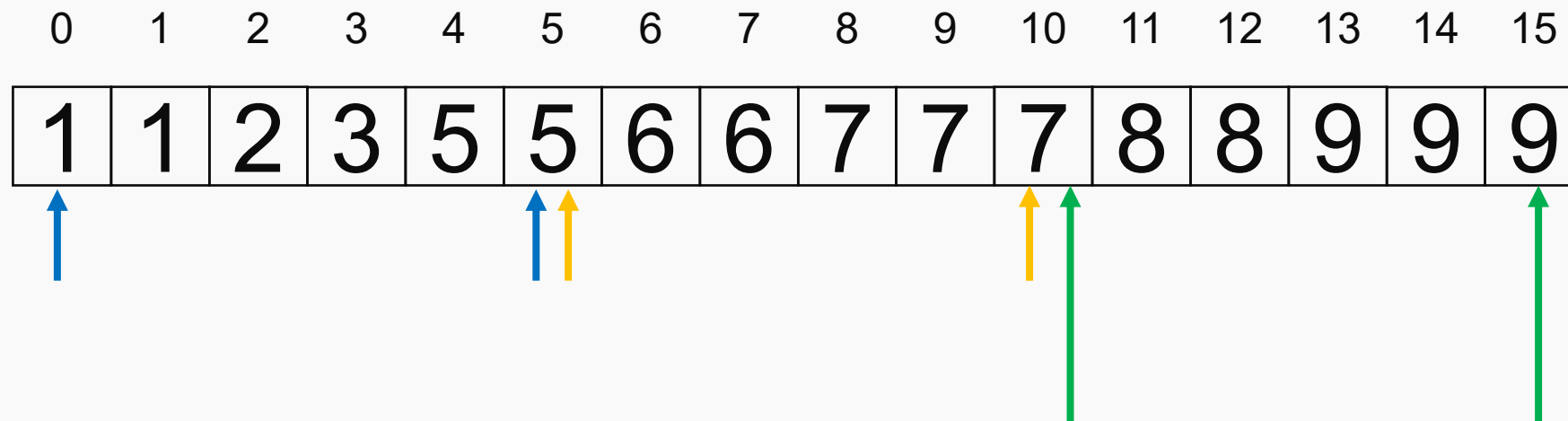
Elementul nu este la mine, mă opresc



# Căutare paralelă – implementare naivă

Căutăm 3

Între pozițiile 0 15



Elementul nu este la mine, mă opresc




# Căutare paralelă – implementare naivă

Căutăm 3

Între pozițiile 1 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

  
**Caut binar**




# Căutare paralelă – implementare naivă

Căutăm 3

Între pozițiile 1 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

  
Caut binar

**Doar un singur pas s-a executat în paralel.  
Am pornit thread-uri degeaba.**

**Complexitate:  $O(\log_2(N))$  la fel ca secvențial**

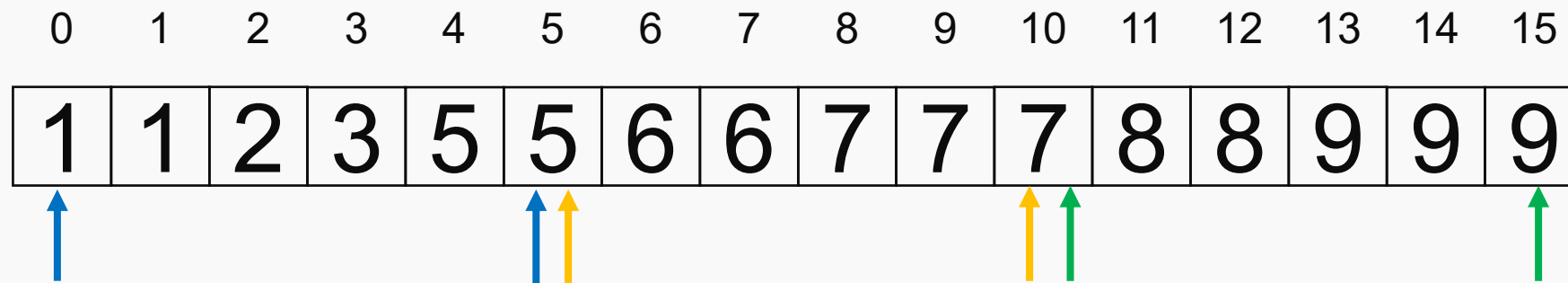




# Căutare paralelă

Căutăm 3

Între pozițiile 0 15



**Fiecare thread este responsabil de o zonă.**

**Când trecem la pasul următor toate thread-urile se mută în noua zonă**



# Căutare paralelă

Căutăm 3

Între pozițiile 0 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9



Elementul căutat este în bucata mea





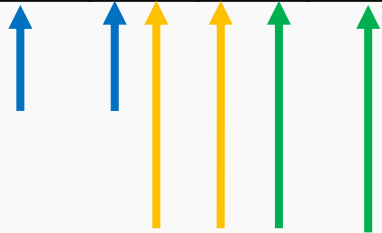


# Căutare paralelă

Căutăm 3

Între pozițiile 1 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9



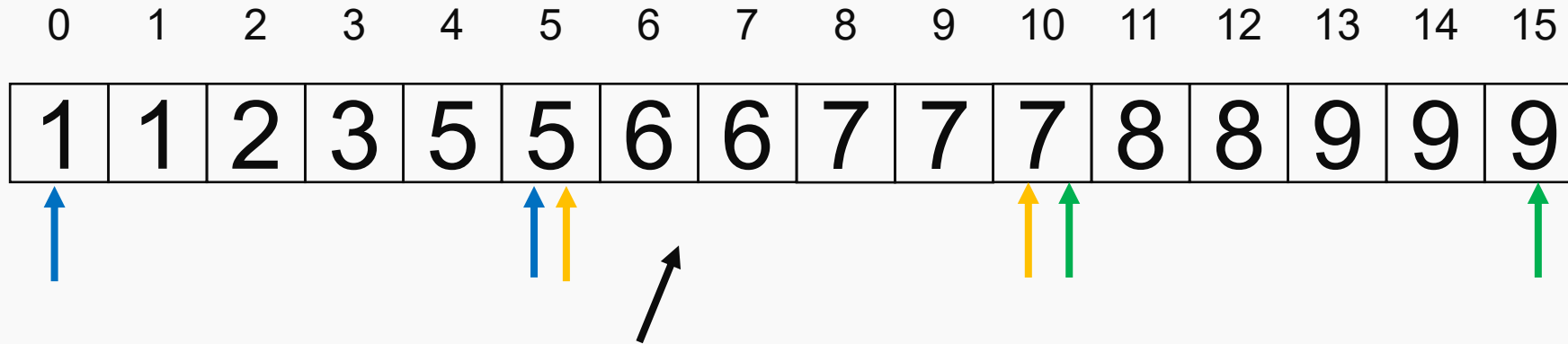
Am găsit elementul

Am găsit elementul

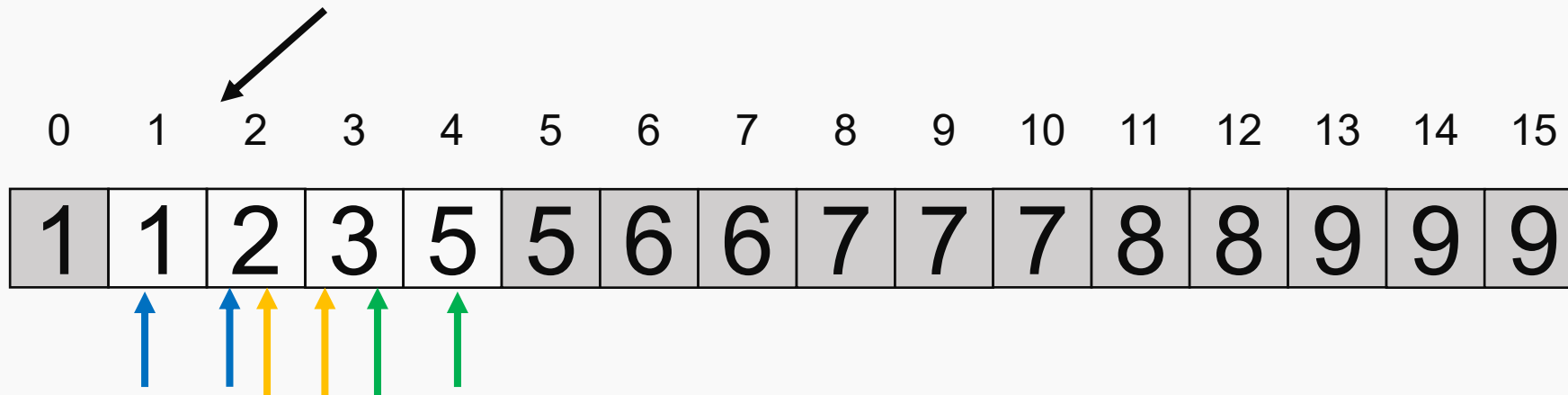
**Toate** thread-urile caută în noua zonă



# Căutare paralelă

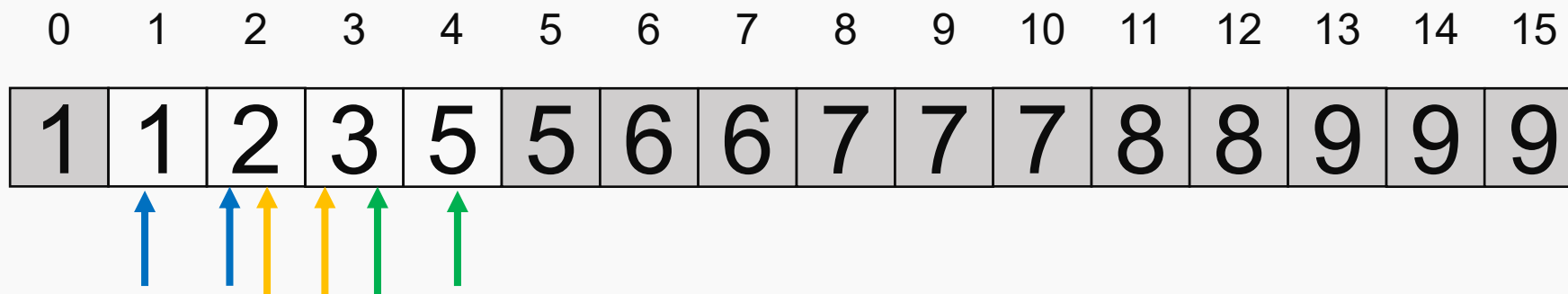
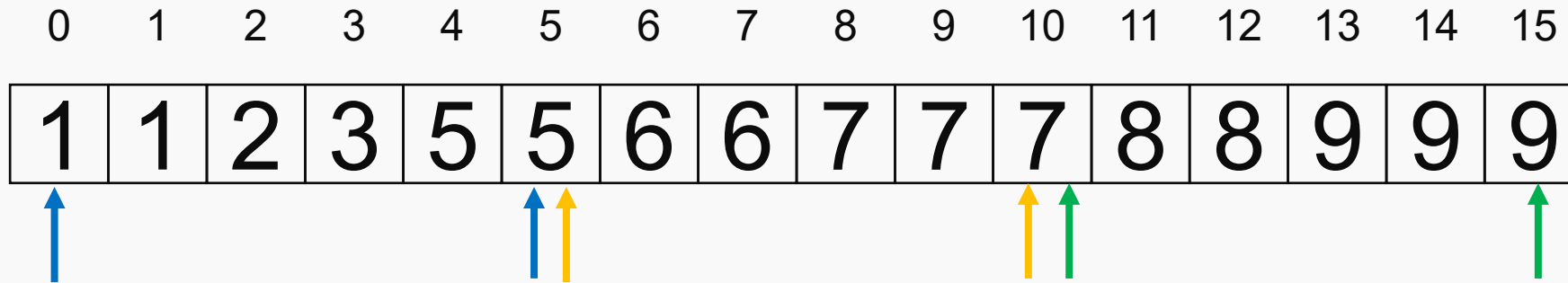


Operațiile aceste **NU** pot executa paralel





# Căutare paralelă





## Căutare paralelă – soluția 2

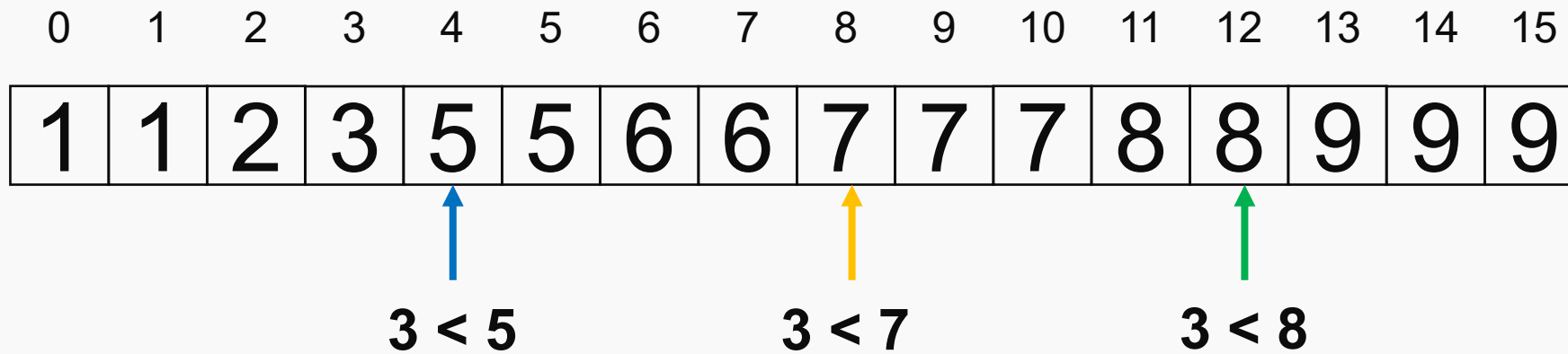
Mai greu de implementat  
Mai puține thread-uri



## Căutare paralelă – soluția 2

Căutăm 3

Între pozițiile 0 15





## Căutare paralelă – soluția 2

Căutăm 3

Între pozițiile 0 3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

$3 > 1$   
 $3 > 2$   
 $3 = 3$  end

Complexitate:  $O(\log_p(n))$



## Căutare paralelă – soluția 2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

$3 < 5$

$3 < 7$

$3 < 8$

Operațiile aceste **NU** pot executa paralel

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9

$3 > 1$

$3 = 3$  end

$3 > 2$



## Căutare paralelă – soluția 2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9


$$3 < 5$$



$$3 < 7$$


$$3 < 8$$



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	5	5	6	6	7	7	7	8	8	9	9	9


$$3 > 1$$


$$3 = 3$$


$$3 > 2$$





# Căutare paralelă - Complexitate

$$O(\log_p(N))$$

Speedup?



# Căutare paralelă - Complexitate

$$O(\log_p(N))$$

Speedup?

$$S = \frac{\log_2(N)}{\log_p(N)}$$



# Căutare paralelă - Complexitate

$$O(\log_p(N))$$

Speedup?

$$S = \frac{\log_2(N)}{\log_p(N)} = \frac{\log(P)}{\log(2)} = \log_2(P)$$





# Merge sort paralel - idee

Operația de merge poate și ea fi paralelizată.  
Pentru a o paraleliza ne bazăm pe căutare binară  
(sau chiar paralelă) și pe rank sort.





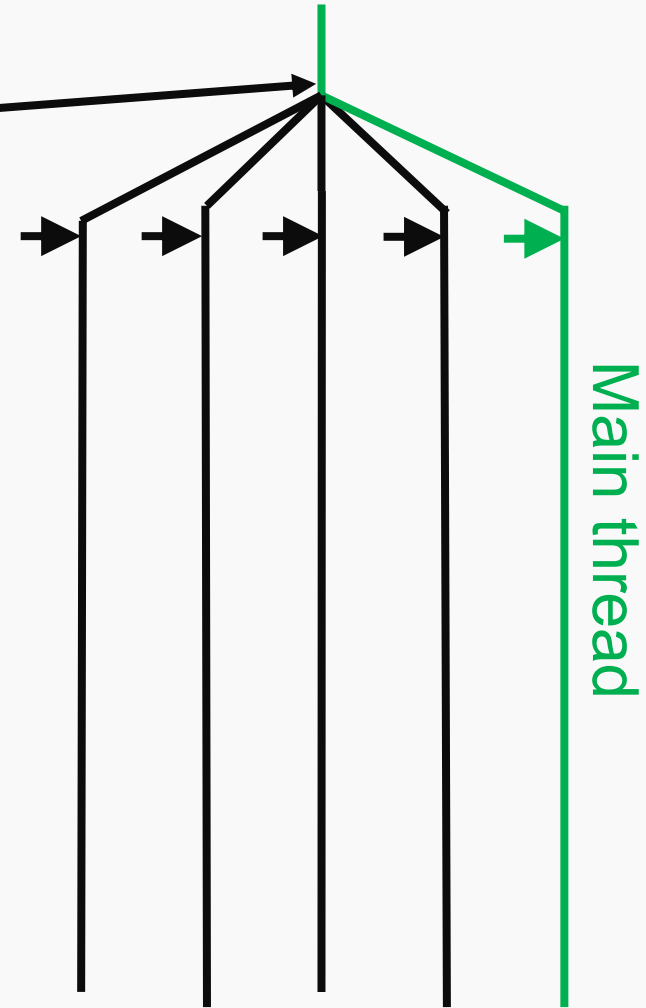
# **Executor Service sau Replicated Workers sau Thread Pool**

**Abordare de probleme recursive în paralel**



# Replicated Workers

`startWorkers();`



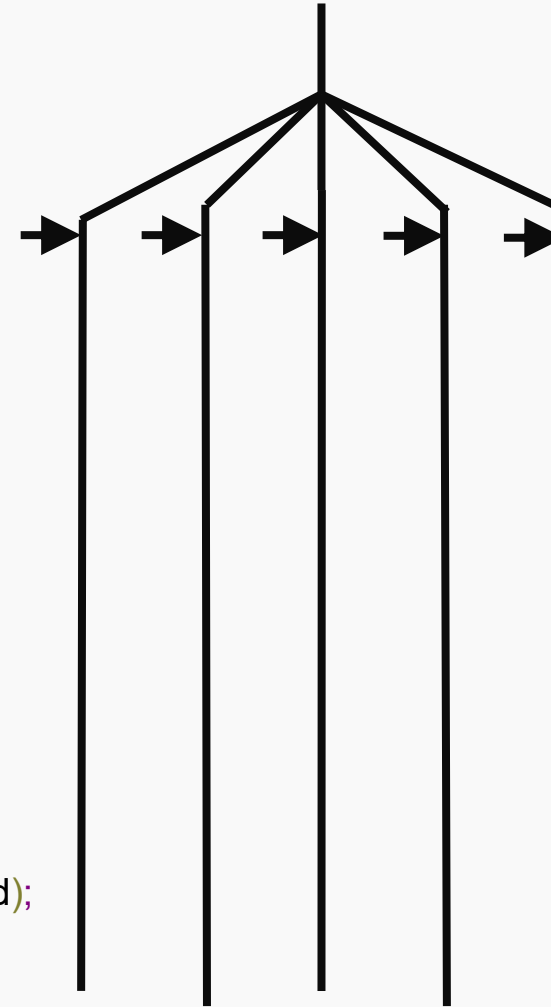




# Replicated Workers

```
Task makeTask(int i)
{
    Task task;
    int * newData = (int*)malloc(sizeof(int));
    newData[0]=i;
    task.data=newData;
    task.runTask = printSomething;
    return task;
}

void printSomething (void * data, int thread_id)
{
    int task_id = *(int*)data;
    if(task_id>N) {
        forceShutDownWorkers();
        return;
    }
    printf("Something %i from thread %i\n", task_id, thread_id);
    putTask(makeTask(task_id+1));
}
```



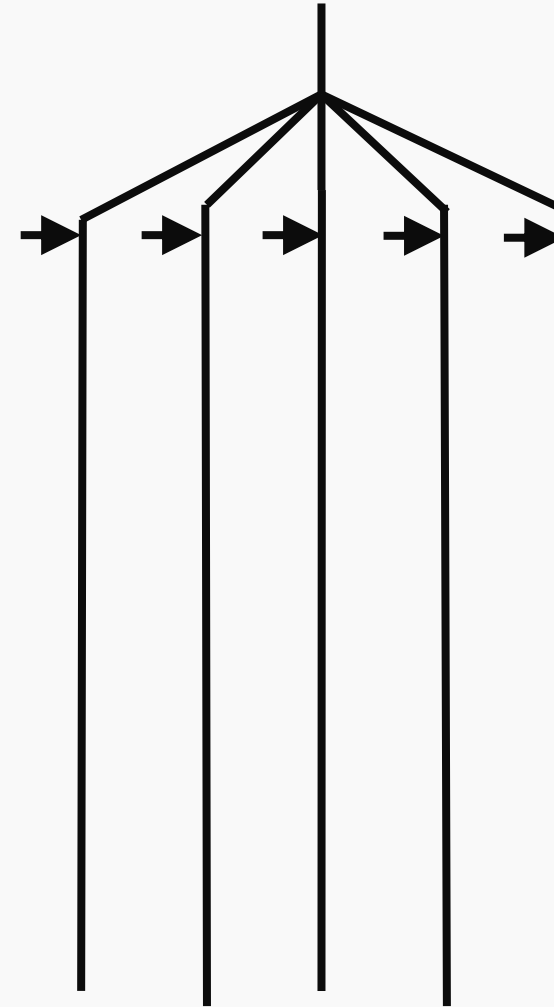


# Replicated Workers

```
putTask(Task1);  
putTask(Task2);  
putTask(Task3);  
.....
```

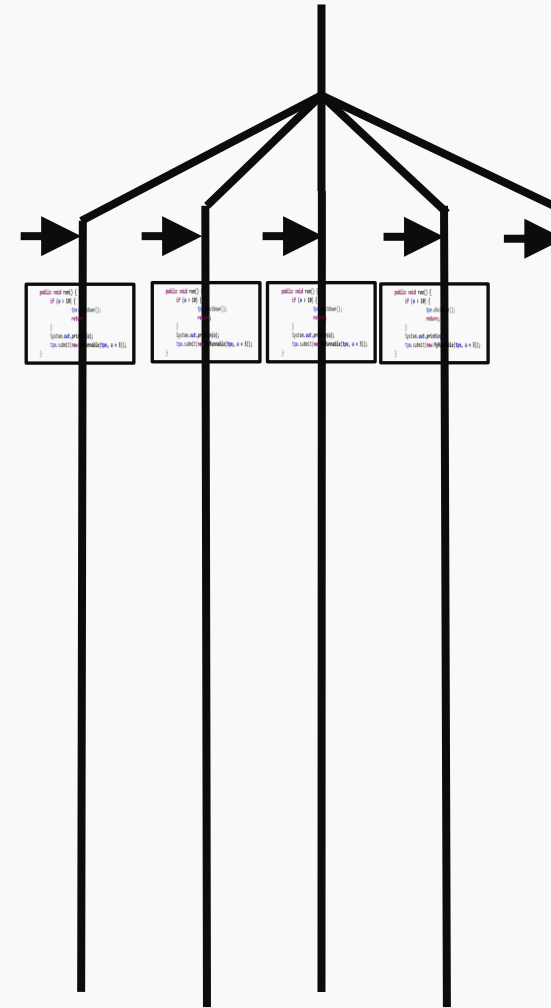


```
public void run() {  
    if (isDone()) {  
        return;  
    }  
    try {  
        putTask(Task1);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
  
public void run() {  
    if (isDone()) {  
        return;  
    }  
    try {  
        putTask(Task2);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
  
public void run() {  
    if (isDone()) {  
        return;  
    }  
    try {  
        putTask(Task3);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
  
public void run() {  
    if (isDone()) {  
        return;  
    }  
    try {  
        putTask(Task4);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
  
public void run() {  
    if (isDone()) {  
        return;  
    }  
    try {  
        putTask(Task5);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}
```





# Replicated Workers



```
public void run() {
    if (isRunning()) {
        return;
    }
    try {
        // ...
    } catch (Exception e) {
        // ...
    }
}

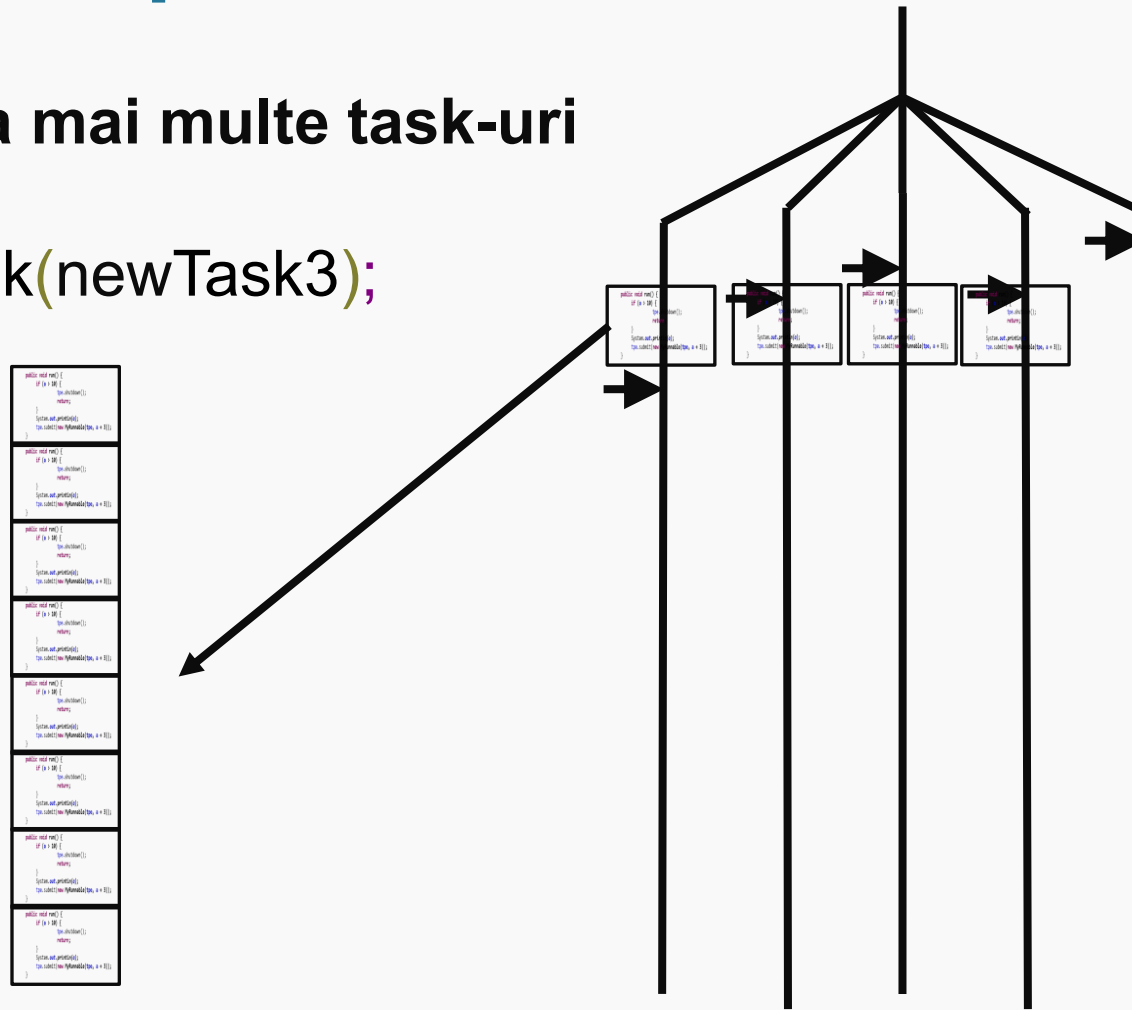
public void run() {
    if (isRunning()) {
        return;
    }
    try {
        // ...
    } catch (Exception e) {
        // ...
    }
}
```



# Replicated Workers

Task-urile pot crea mai multe task-uri

```
putTask(newTask3);
```





# Replicated Workers

```
public void run() {
    if (isAlive()) {
        doWork();
    }
    System.out.println(
        "doWork() new Worker() time, s = " + E);
}

public void run() {
    if (isAlive()) {
        doWork();
    }
    System.out.println(
        "doWork() new Worker() time, s = " + E);
}

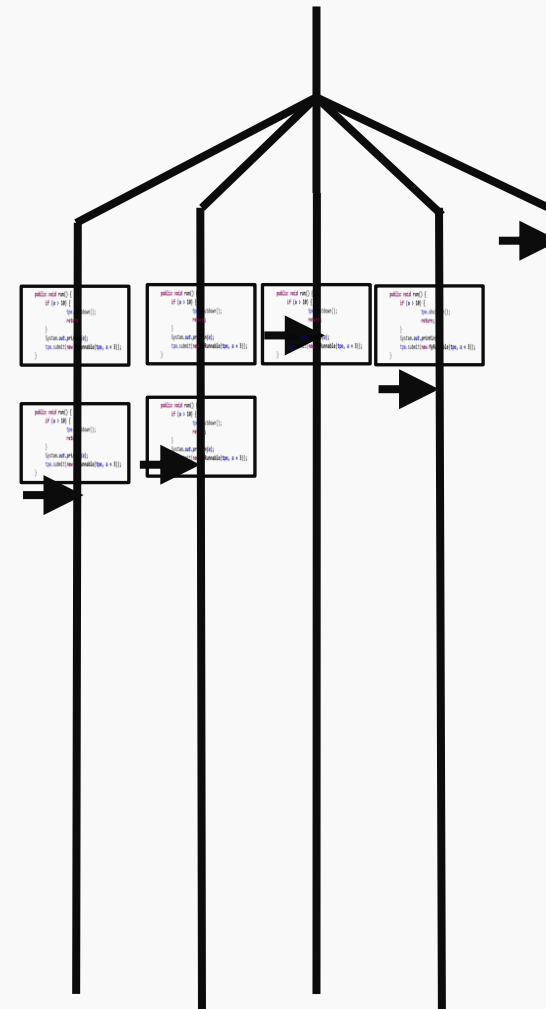
public void run() {
    if (isAlive()) {
        doWork();
    }
    System.out.println(
        "doWork() new Worker() time, s = " + E);
}

public void run() {
    if (isAlive()) {
        doWork();
    }
    System.out.println(
        "doWork() new Worker() time, s = " + E);
}

public void run() {
    if (isAlive()) {
        doWork();
    }
    System.out.println(
        "doWork() new Worker() time, s = " + E);
}

public void run() {
    if (isAlive()) {
        doWork();
    }
    System.out.println(
        "doWork() new Worker() time, s = " + E);
}

public void run() {
    if (isAlive()) {
        doWork();
    }
    System.out.println(
        "doWork() new Worker() time, s = " + E);
}
```





# Replicated Workers

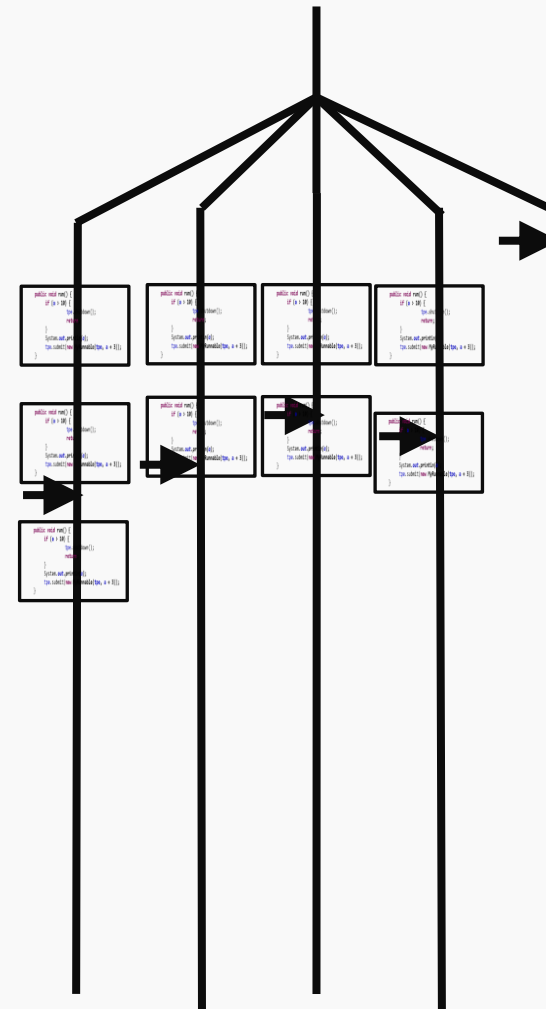
```
public void run() {  
    if (isAlive()) {  
        doAction();  
    }  
    System.out.println();  
    try { Thread.sleep(1000); } catch (InterruptedException e) {}  
}
```

```
public void run() {  
    if (isAlive()) {  
        doAction();  
    }  
    System.out.println();  
    try { Thread.sleep(1000); } catch (InterruptedException e) {}  
}
```

```
public void run() {  
    if (isAlive()) {  
        doAction();  
    }  
    System.out.println();  
    try { Thread.sleep(1000); } catch (InterruptedException e) {}  
}
```

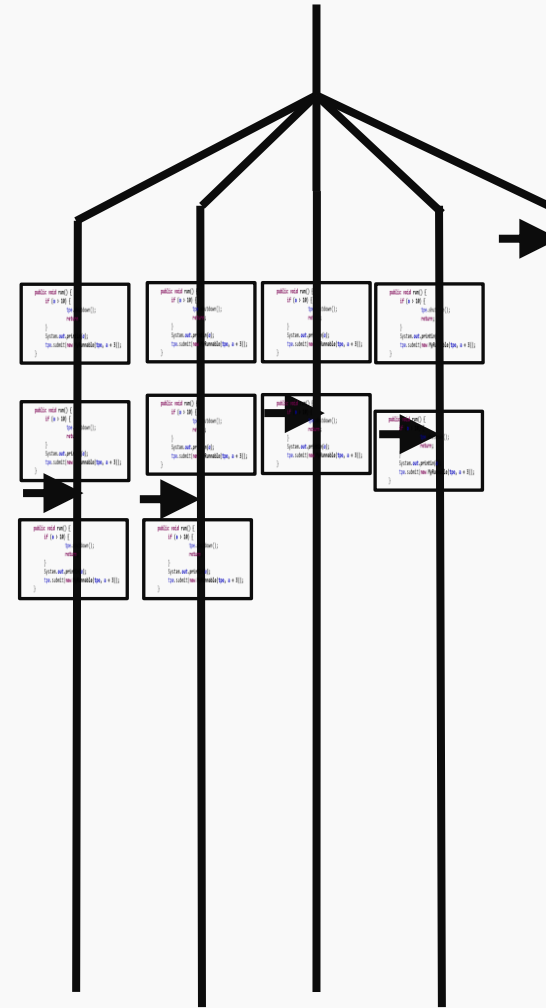
```
public void run() {  
    if (isAlive()) {  
        doAction();  
    }  
    System.out.println();  
    try { Thread.sleep(1000); } catch (InterruptedException e) {}  
}
```

```
public void run() {  
    if (isAlive()) {  
        doAction();  
    }  
    System.out.println();  
    try { Thread.sleep(1000); } catch (InterruptedException e) {}  
}
```





# Replicated Workers



```
public void run() {
    if (isPrime()) {
        return;
    }
    System.out.println("Not a prime number");
}

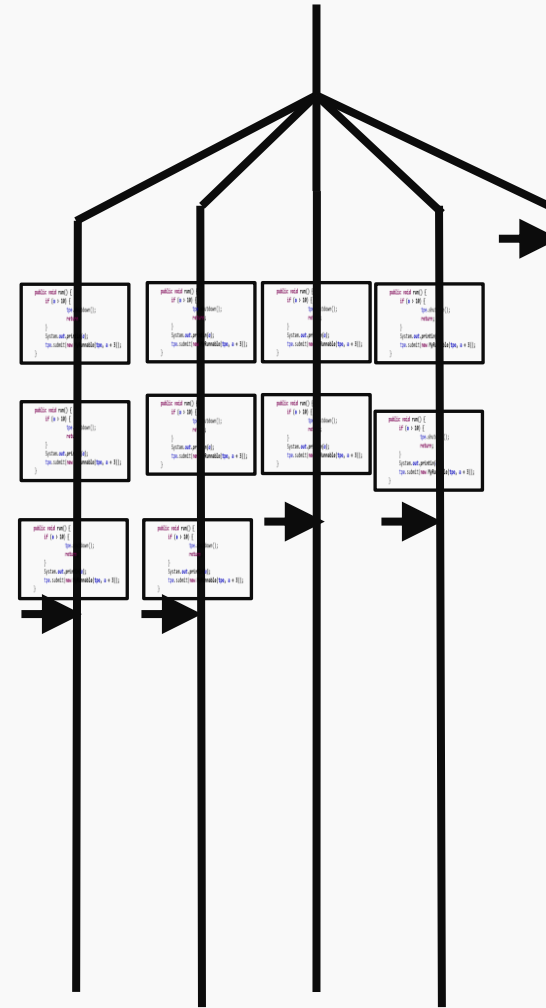
public void run() {
    if (isPrime()) {
        return;
    }
    System.out.println("Not a prime number");
}

public void run() {
    if (isPrime()) {
        return;
    }
    System.out.println("Not a prime number");
}

public void run() {
    if (isPrime()) {
        return;
    }
    System.out.println("Not a prime number");
}
```



# Replicated Workers



```
public void run() {
    if (is + BE) {
        // ...
    }
    // ...
    // ...
    // ...
}
```

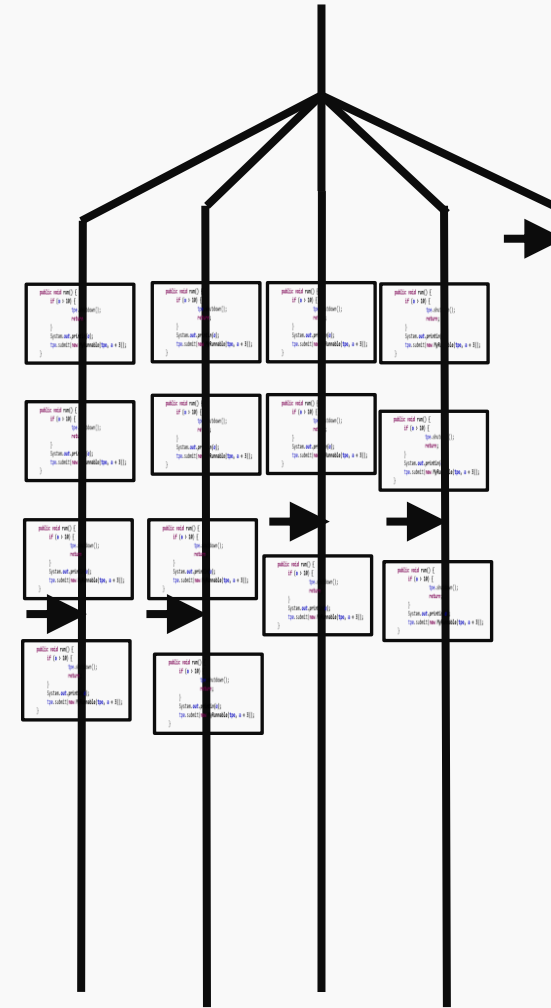




# Replicated Workers

Când oprim thread-urile?  
Depinde de problemă.  
Dacă e suficient putem să oprim  
imediat după găsirea unei soluții.  
Trebuie ținut cont că unele probleme  
nu au soluție.

`joinWorkerThreads();`







# N Queens Problem

		Q	
Q			
			Q
	Q		

**Se cere aranjare a N regine pe o tablă NxN în  
așa fel încât să nu se atace**



# N Queens Problem

**Nu avem voie mai mult de o regină pe linie**

		Q	
Q			
			Q
	Q		



# N Queens Problem

**Nu avem voie mai mult de o regină pe coloană**

		Q	
Q			
			Q
	Q		



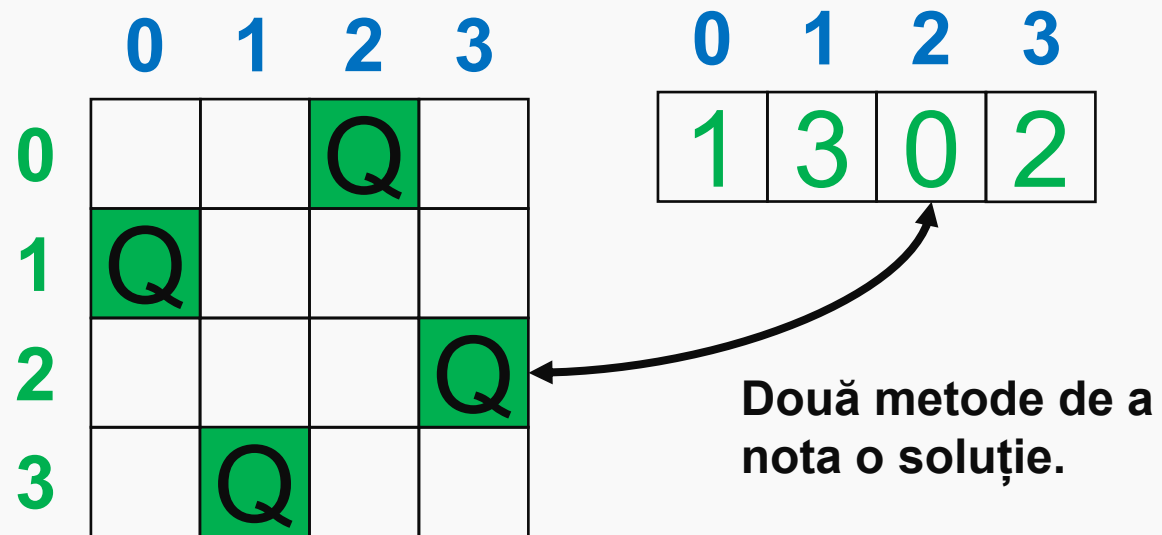
# N Queens Problem

**Nu avem voie mai mult de o regină pe diagonală**

		Q	
Q			
			Q
	Q		



# N Queens Problem



**Poziția din vector reprezintă coloana pe care este așezată regina.**

**Valoarea din vector reprezintă linia pe care este așezată regina.**



# N Queens Problem – Soluție

	0	1	2	3
0	Q			
1				
2				
3				

0	1	2	3
0			





# N Queens Problem – Soluție

	0	1	2	3
0	Q	Q		
1				
2				
3				

0	1	2	3
0	0		

**Conflict linie**



# N Queens Problem – Soluție

	0	1	2	3
0	Q			
1		Q		
2				
3				

0	1	2	3
0	1		

**Conflict diagonală**



# N Queens Problem – Soluție

	0	1	2	3
0	Q			
1				
2		Q		
3				

0	1	2	3
0	2		

**OK.**

**Și tot așa până  
punem toate  
reginele**



# N Queens Problem – Soluție paralelă

0 1 2 3

0			
---	--	--	--

1			
---	--	--	--

2			
---	--	--	--

3			
---	--	--	--



# N Queens Problem – Soluție paralelă

0	1	2	3	
0	0			x
0	1			x
0	2			
0	3			

0	1	2	3	
2	0			
2	1			x
2	2			x
2	3			x

Și tot așa...

0	1	2	3	
1	0			x
1	1			x
1	2			x
1	3			

0	1	2	3	
3	0			
3	1			
3	2			x
3	3			x

