Structuri de date și algoritmi

Algoritmi de sortare

Lect. Dr. Ing. Cristian Chilipirea – cristian.chilipirea@mta.ro

De ce discutăm de sortări?

- În practică o să folosim în majoritatea cazurilor algoritmi de sortare gata implementați.
  - Aceștia se găsesc în majoritatea libajelor/librăriilor standard.
- Problema sortării și corectitudinea rezultatului sunt ușor de înțeles și verificat.
- Este o problemă foarte bine studiată și cunoscută. (Knuth – TAoCP vol:3 Searching and Sorting – 800 pages - 1973)
- Sunt multe metode/mulți algoritmi cunoscuți care rezolvă problema.
  - Ne ajută să gândim o problemă în multe feluri

Ce este o sortare?

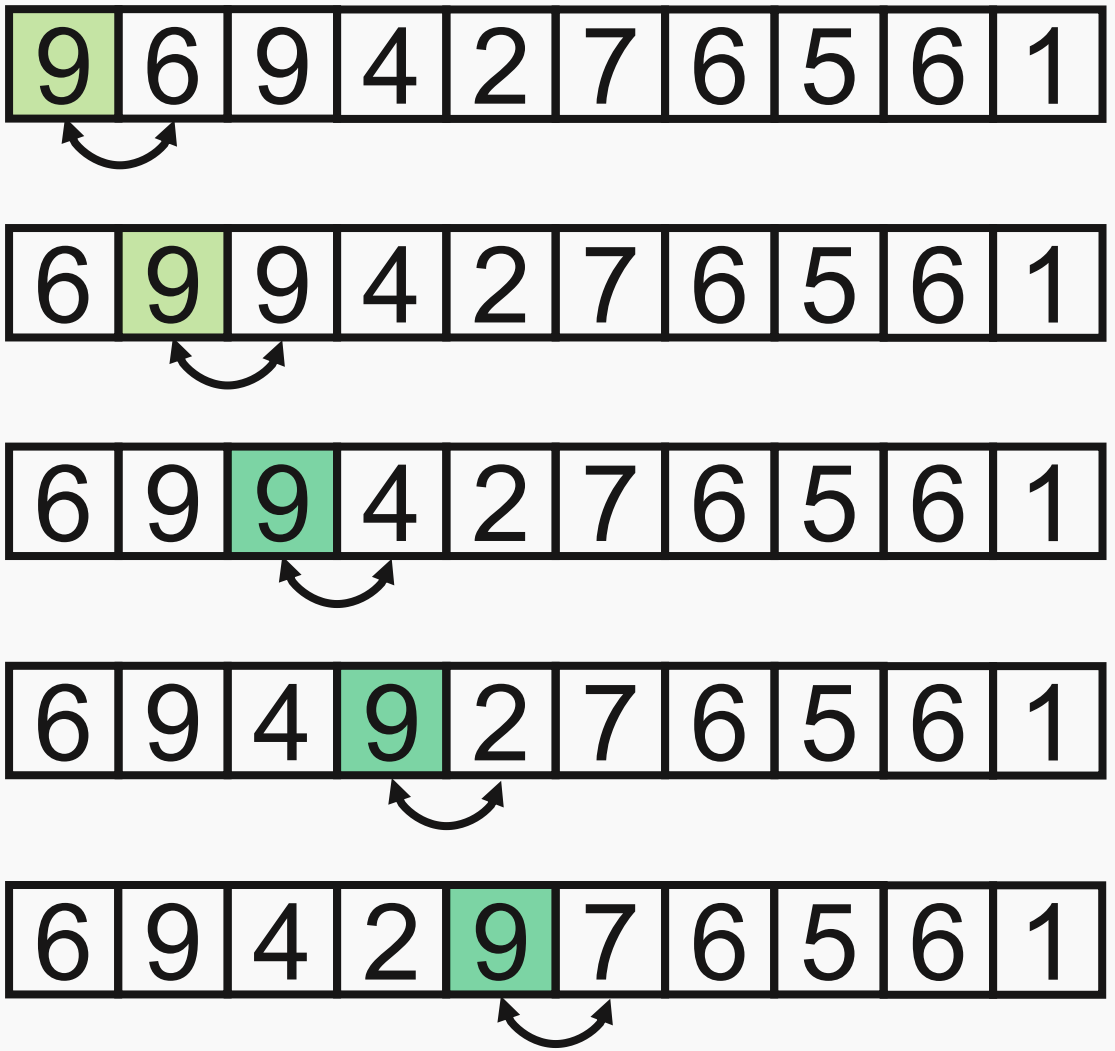
Fiind date **N** înregistrări $r_1, r_2, \ldots r_N$ având cheile $k_1, k_2, \ldots k_N$, trebuie găsita o permutare σ care să pună înregistrările în ordinea $r_{\sigma(1)}, r_{\sigma(2)}, \ldots r_{\sigma(N)}$ astfel încât:

$$k_{\sigma(1)} \leq k_{\sigma(2)} \leq \cdots \leq k_{\sigma(N)}$$

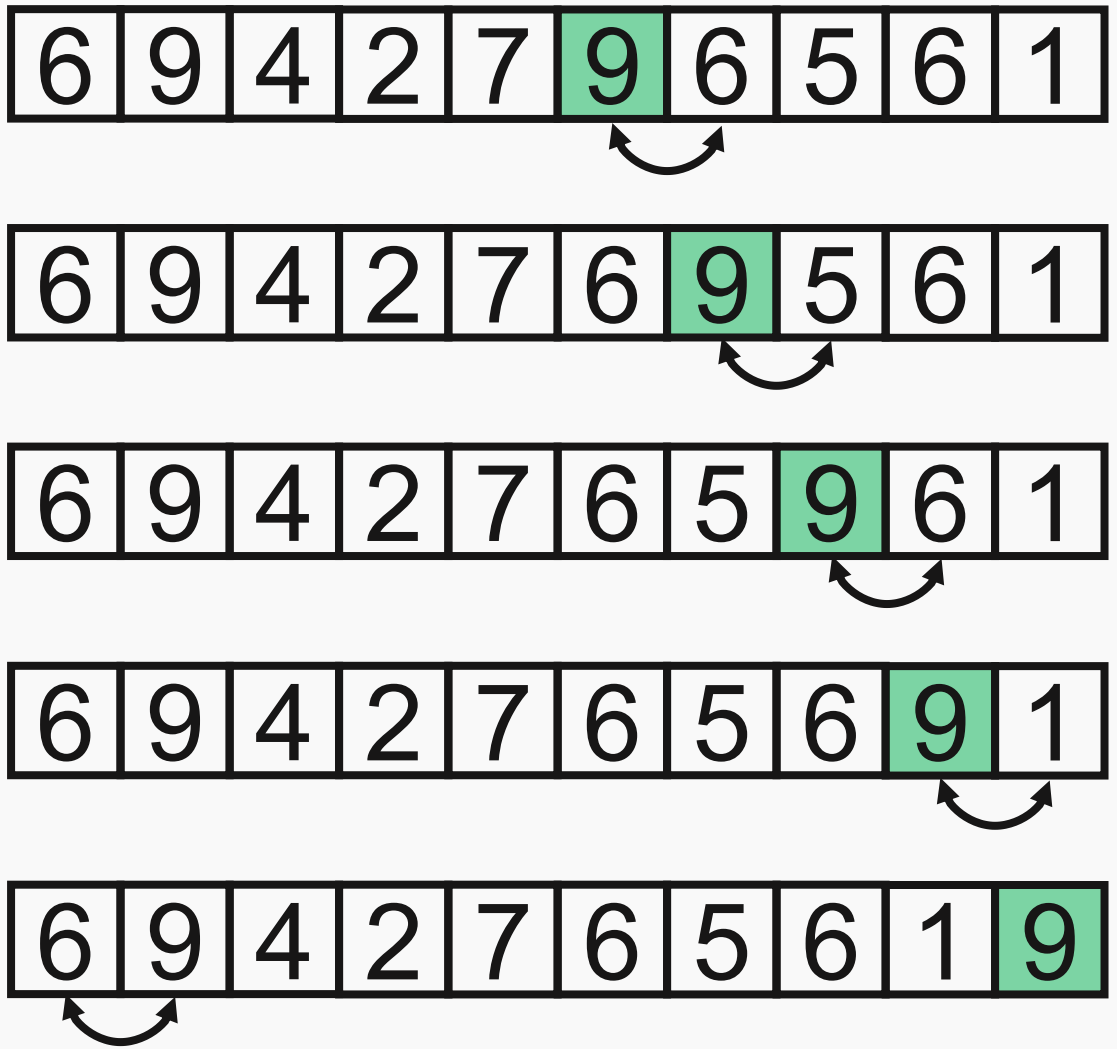
Bubble sort

- Dacă două elemente consecutive NU sunt în ordinea cerută se interschimbă.
- După aplicarea acestei reguli o dată pentru toate elementele (o parcurgere) cel mai mic/mare element ajunge pe ultima poziție.
- Se poate repeta până ce vectorul este sortat.
- Garantat în N pași este sortat.

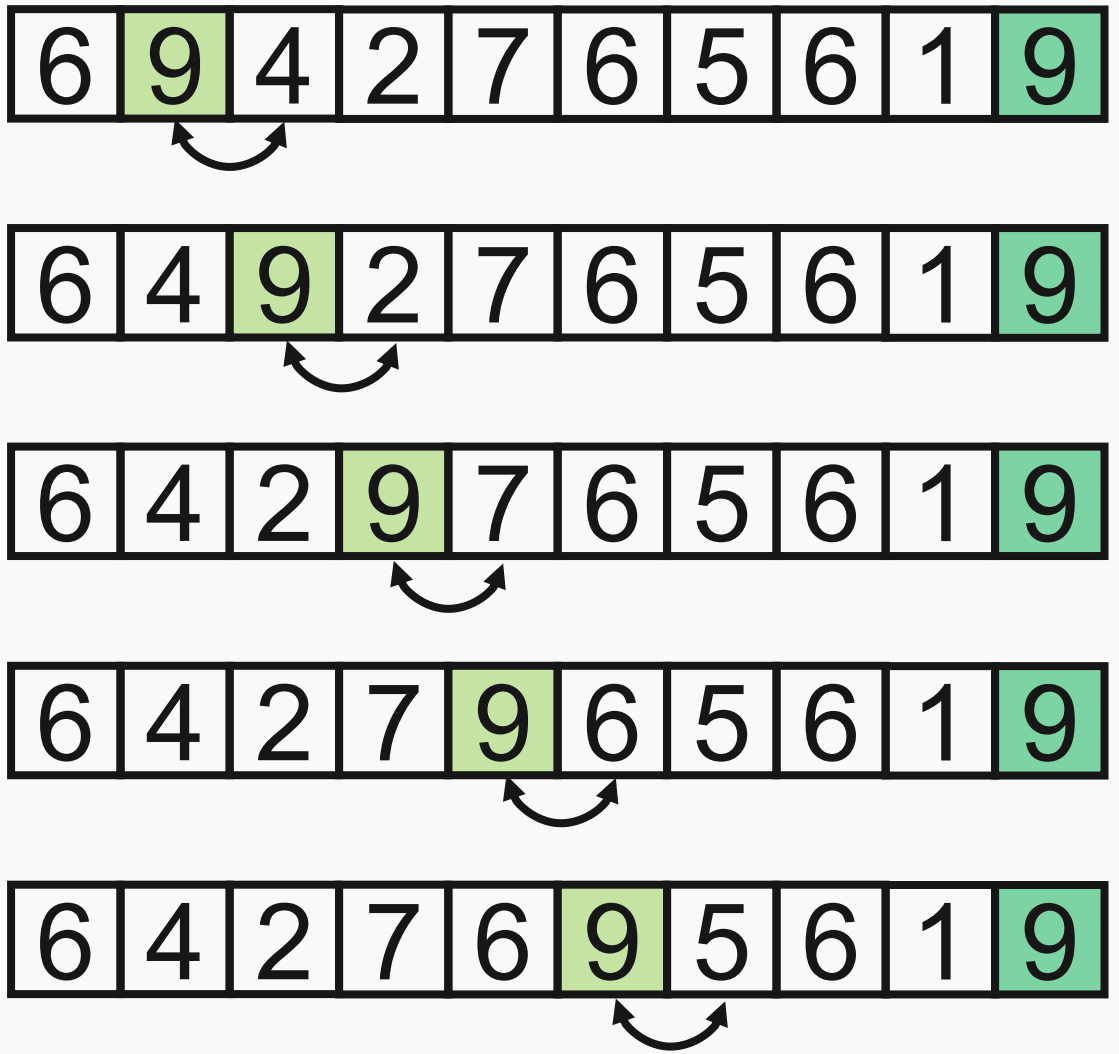
Bubble sort


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

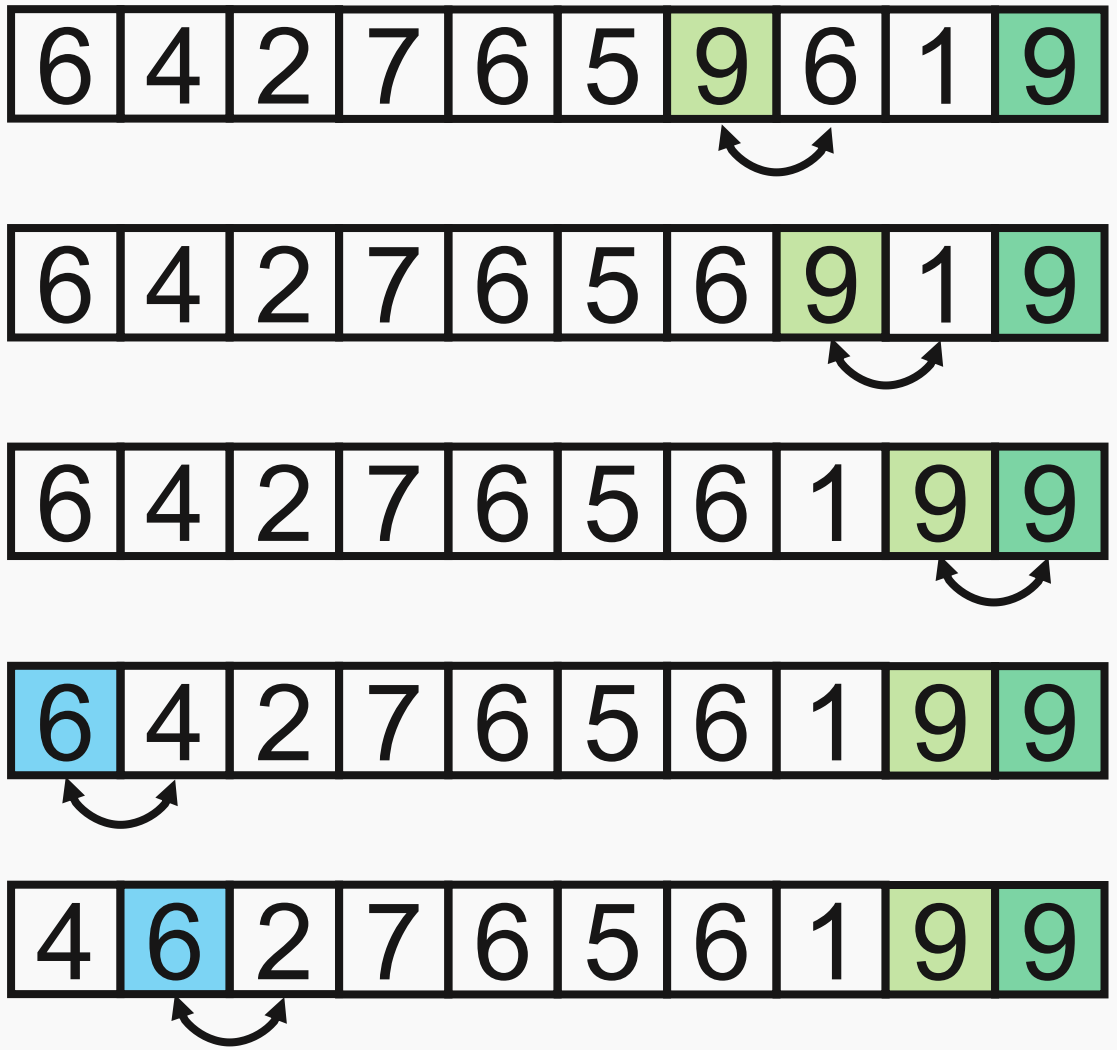
Bubble sort


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

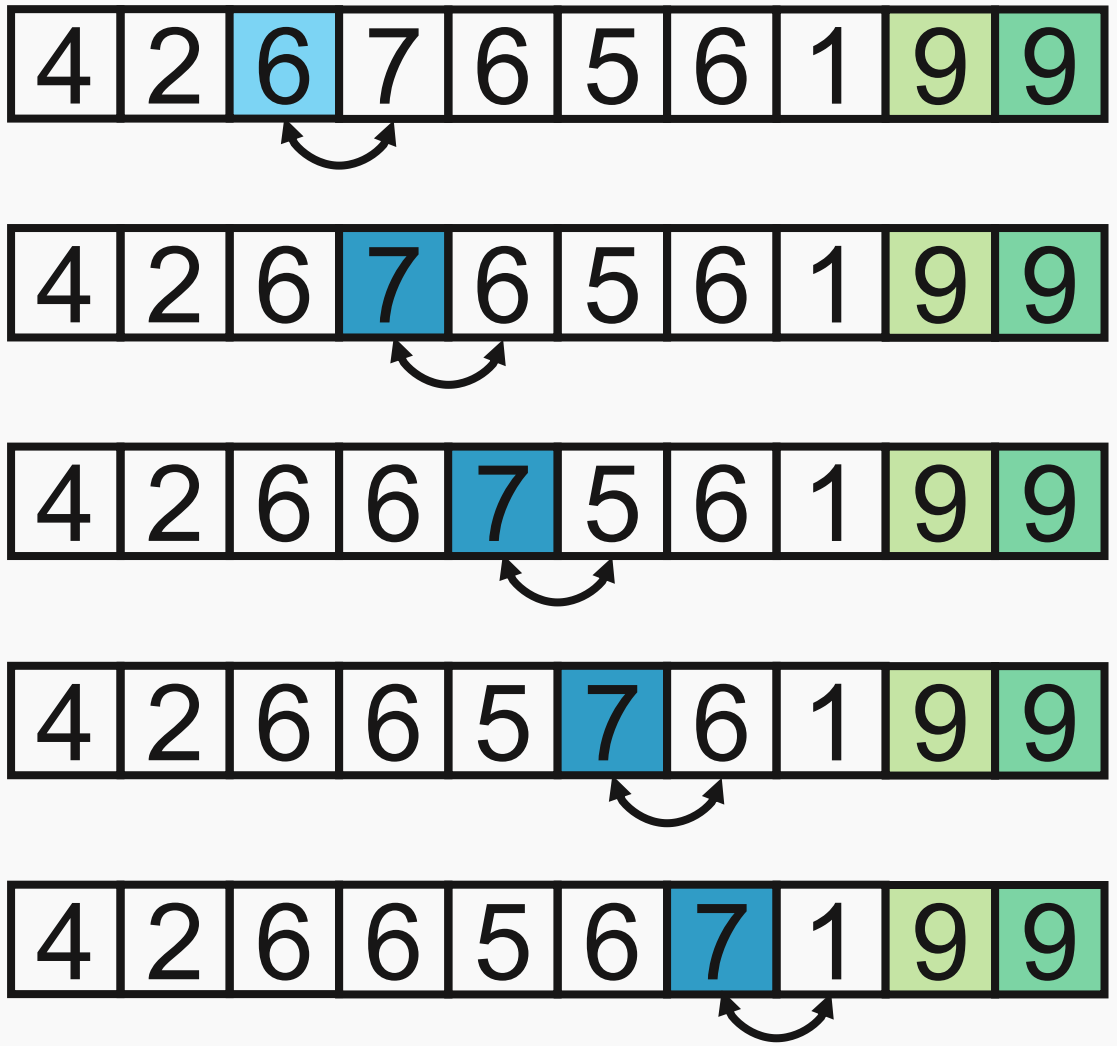
Bubble sort


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

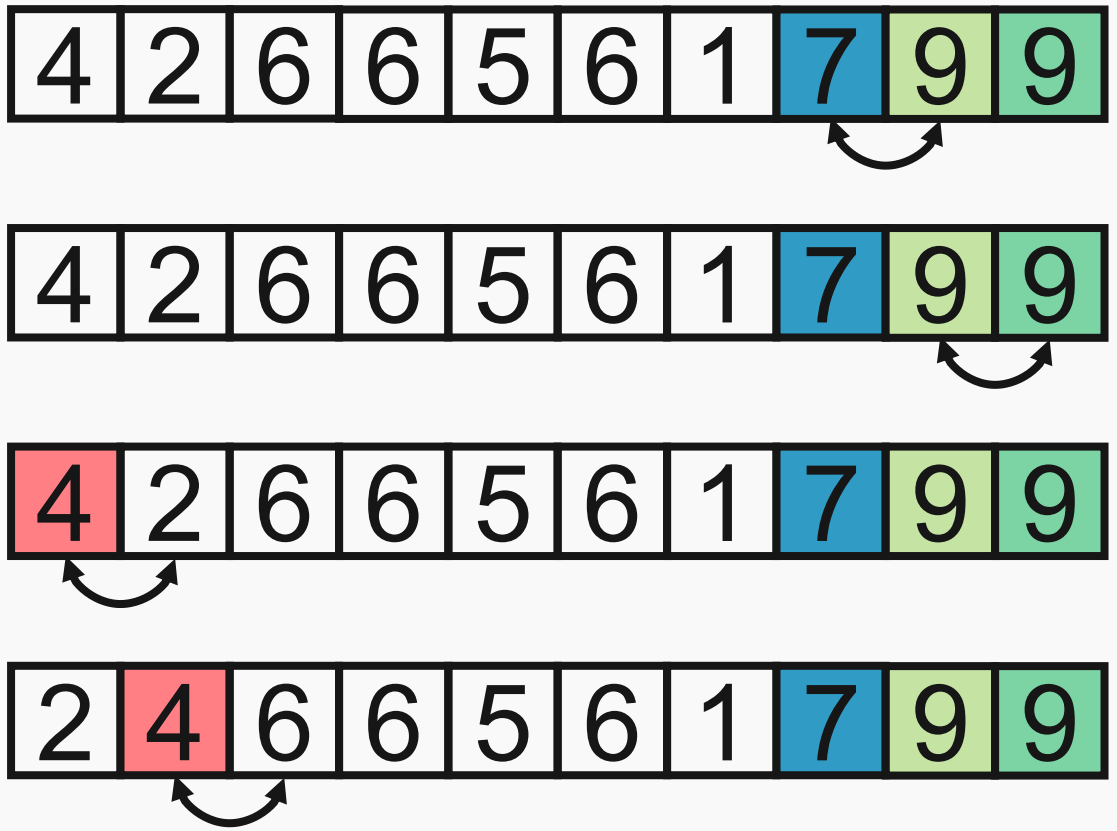
Bubble sort


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

Bubble sort


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

Bubble sort



.....

Complexitate?

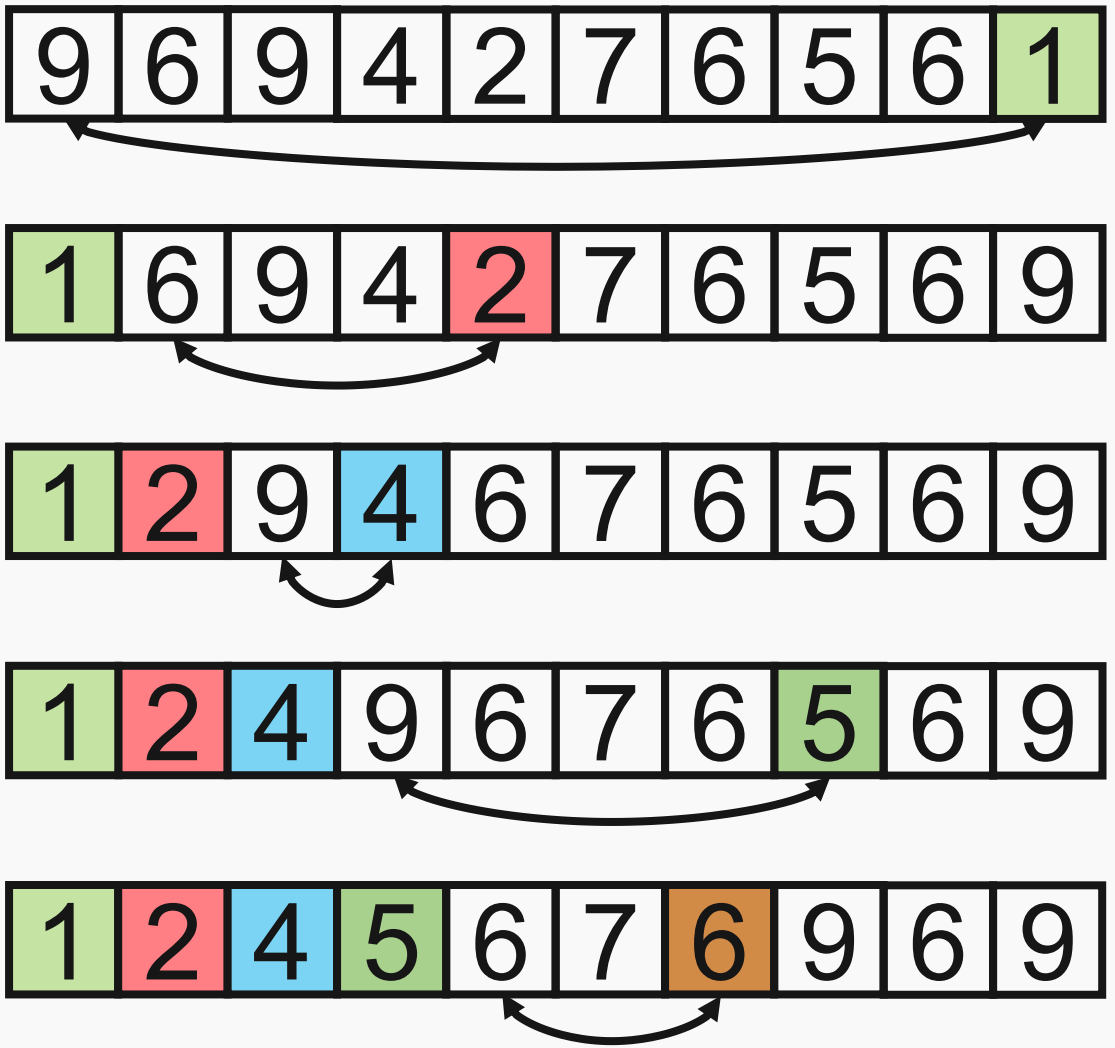
Complexitate?

$$O(N^2)$$

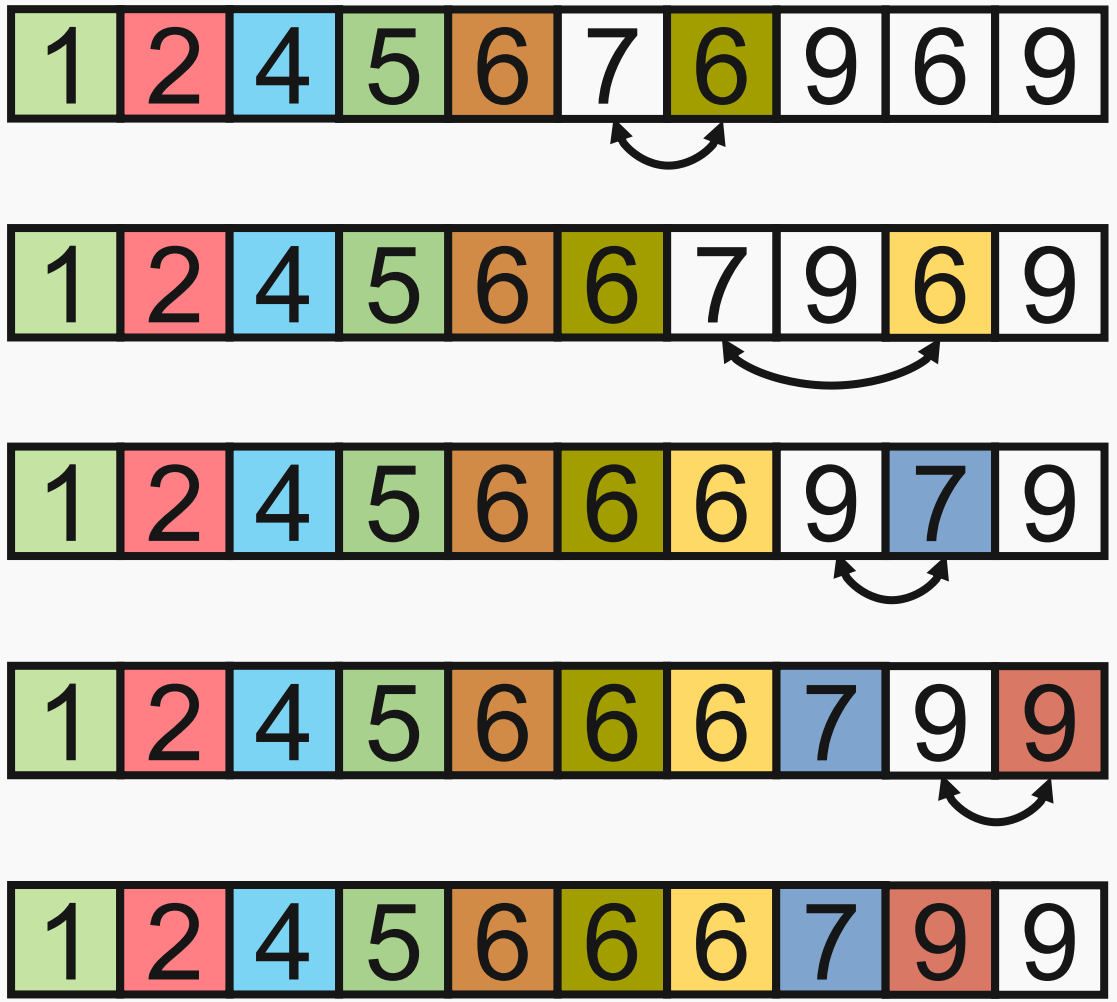
Selection Sort

- Se caută cel mai mic element.
- Se aduce pe prima poziție.
- Se caută următorul cel mai mic element.
- ...

Selection Sort



Selection Sort



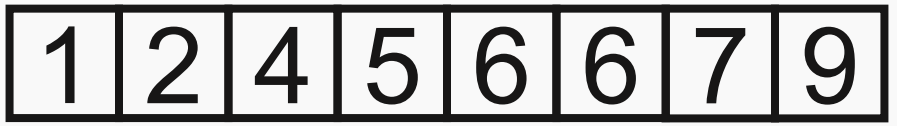
Complexitate?

$$O(N^2)$$

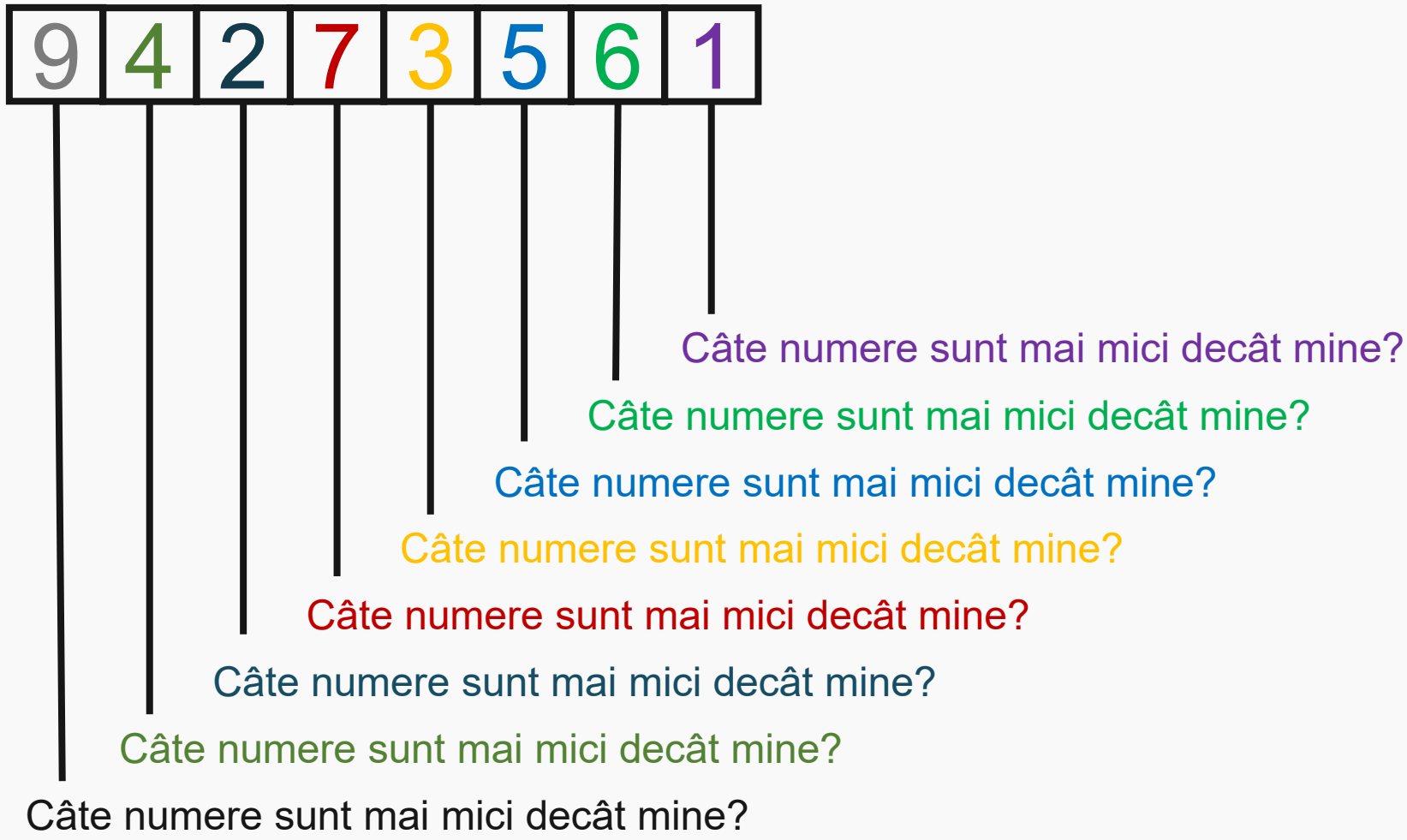
Count Sort / Rank Sort

- Câte elemente sunt mai mici decât un anumit element?
  - Aceea este poziția elementului în vectorul sortat.

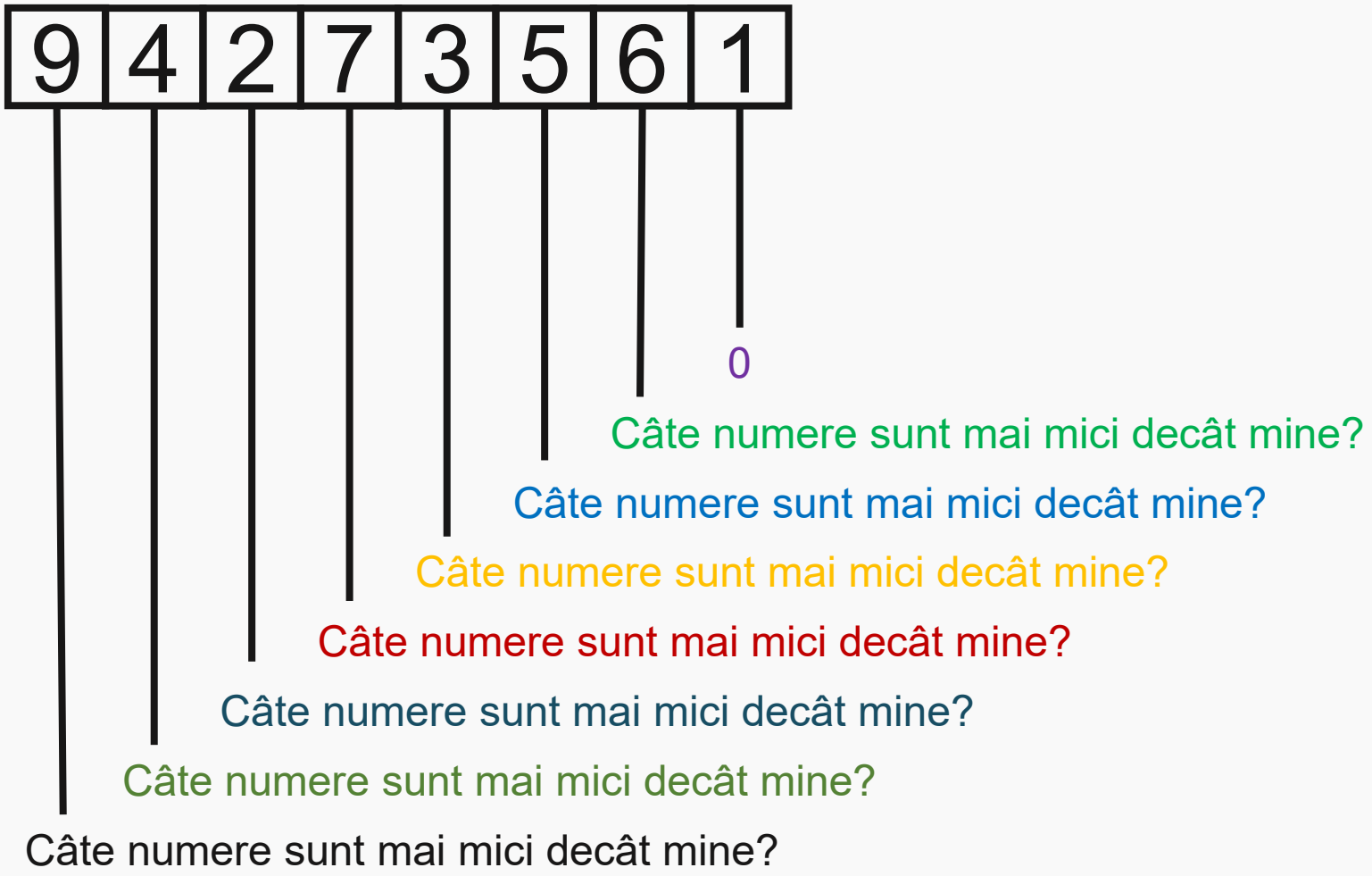
Count Sort / Rank Sort



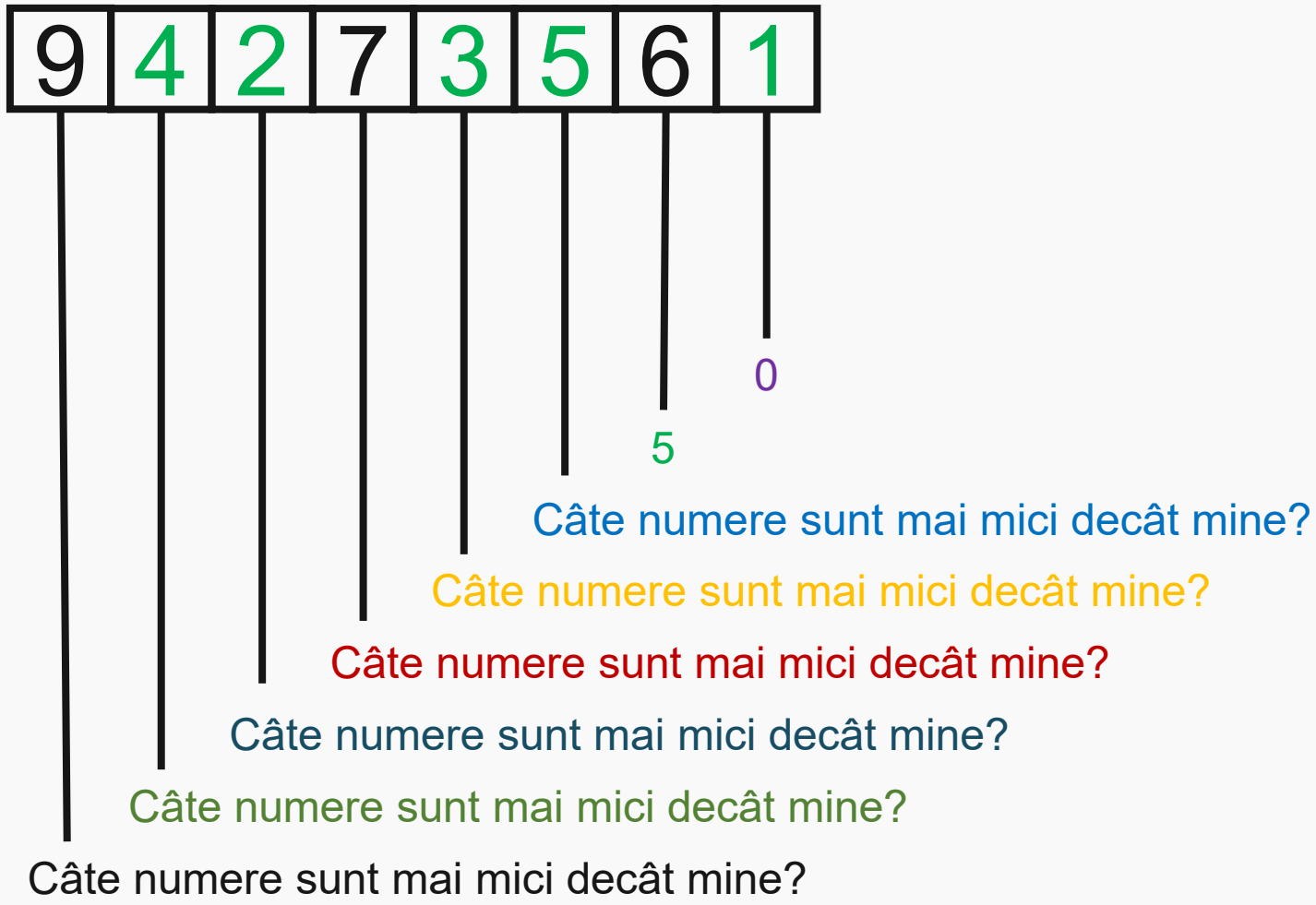
Count Sort / Rank Sort



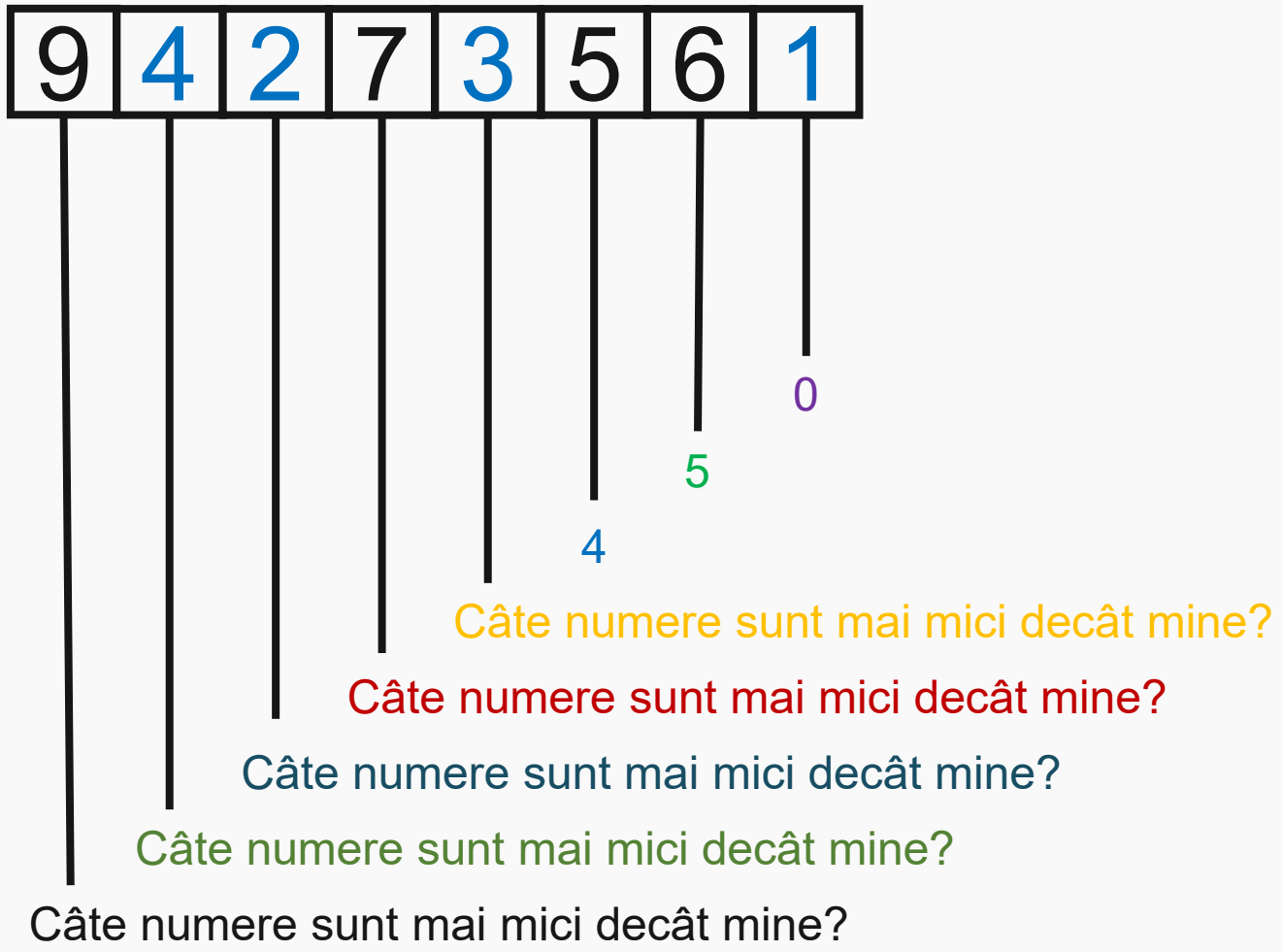
Count Sort / Rank Sort



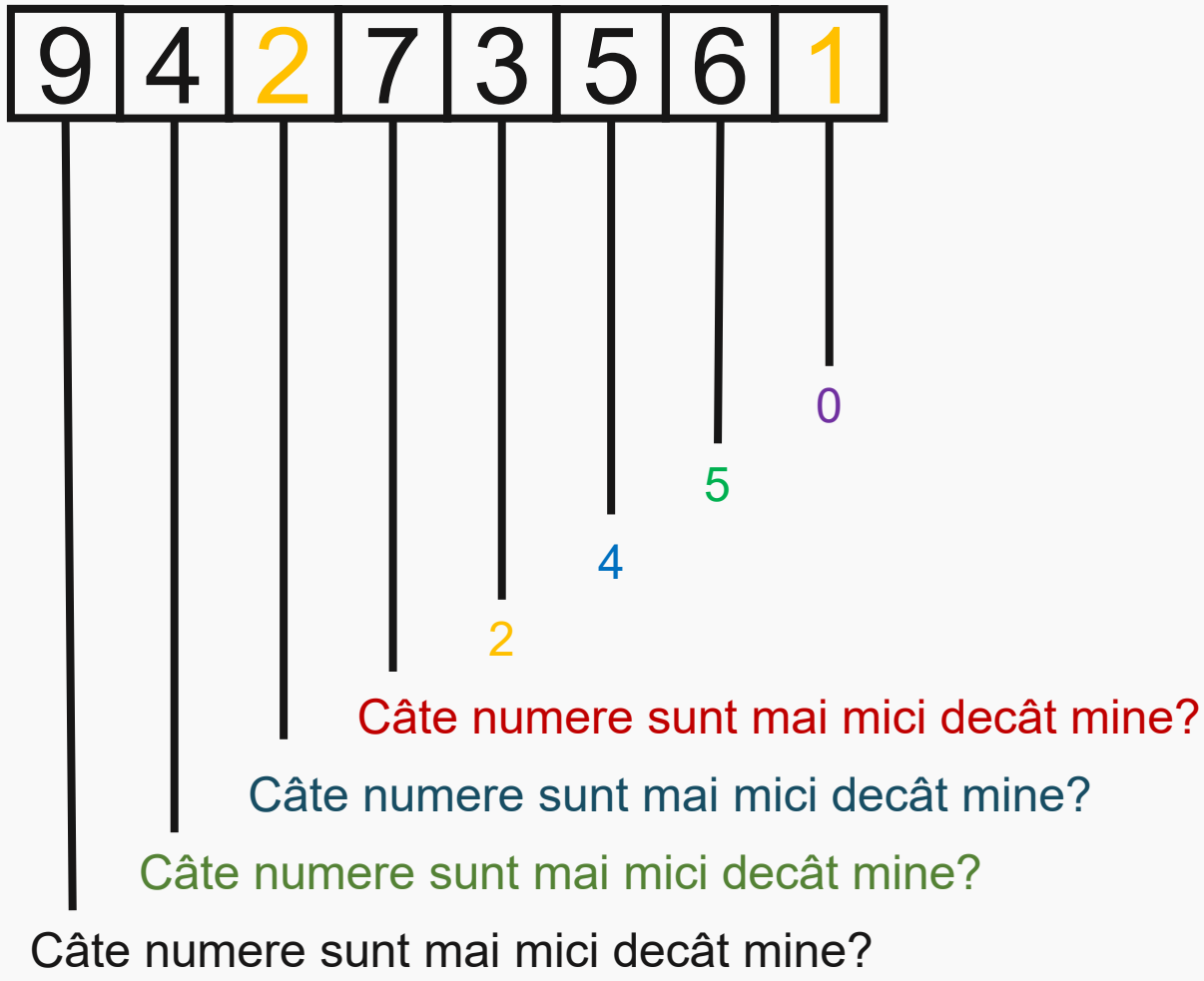
Count Sort / Rank Sort



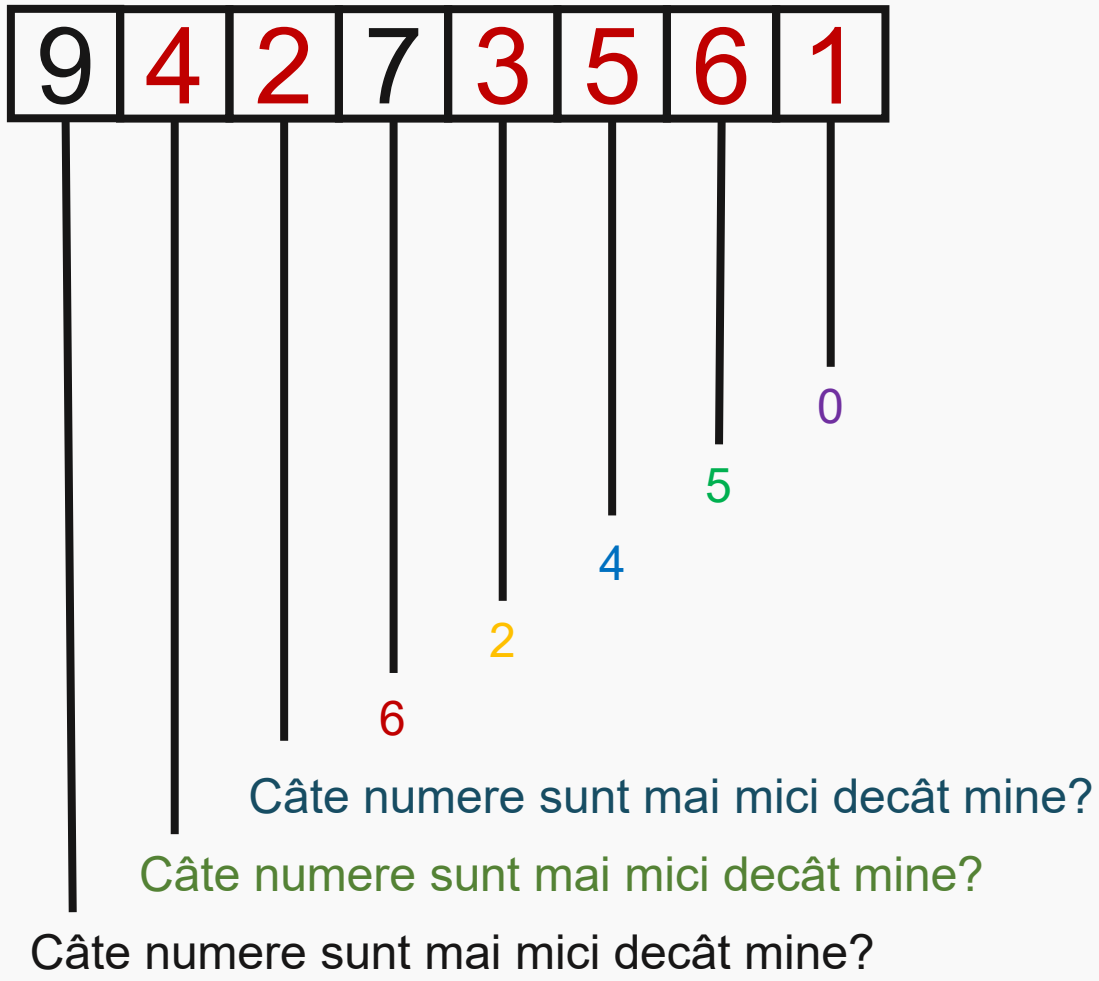
Count Sort / Rank Sort



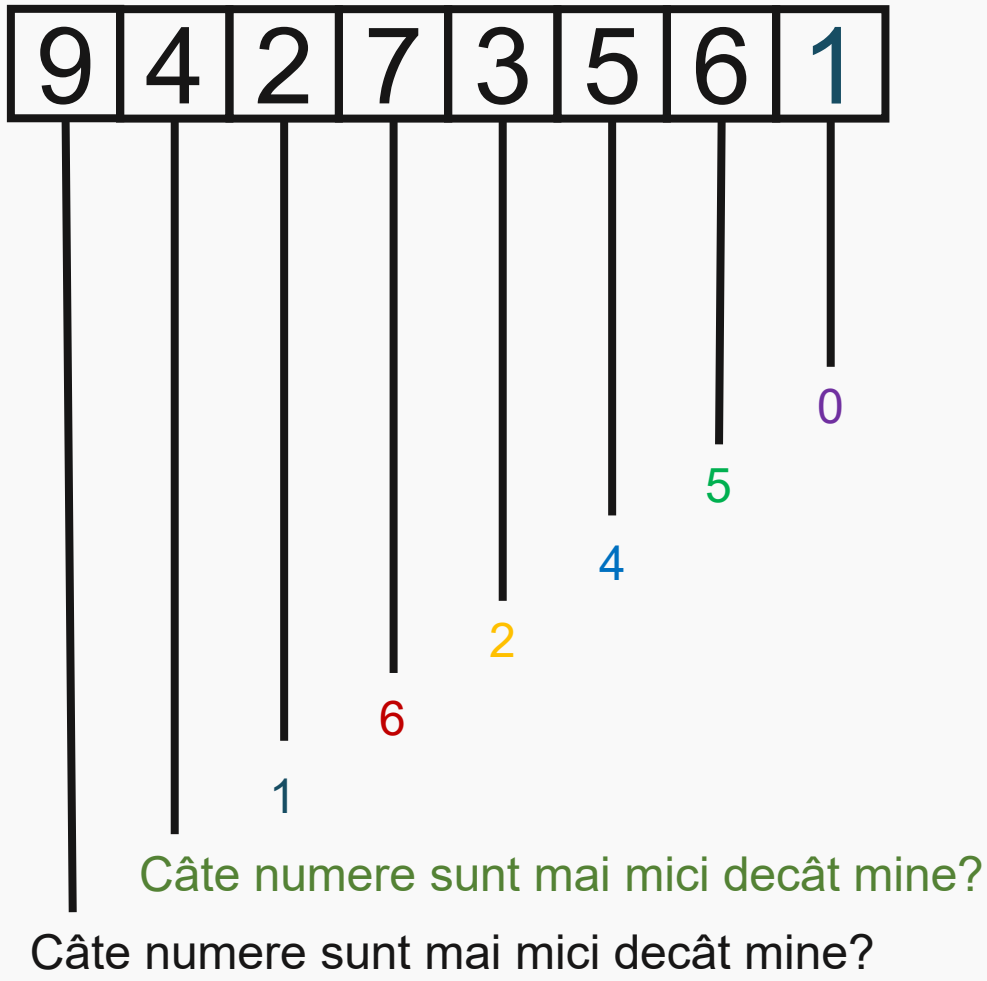
Count Sort / Rank Sort



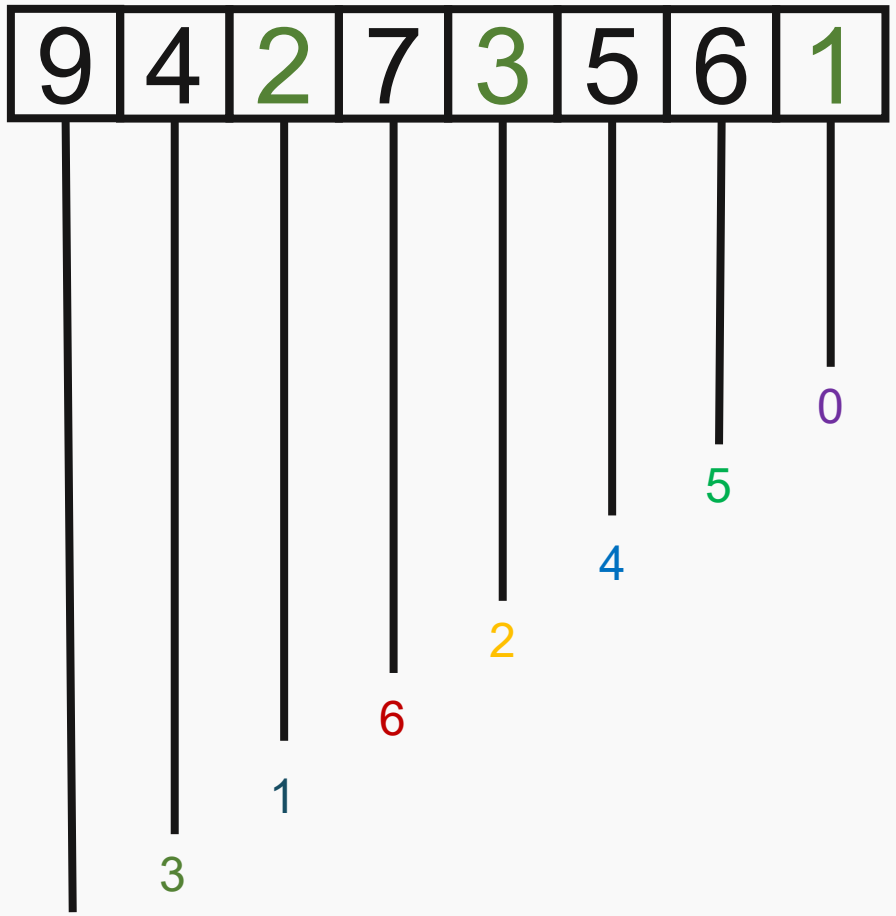
Count Sort / Rank Sort



Count Sort / Rank Sort

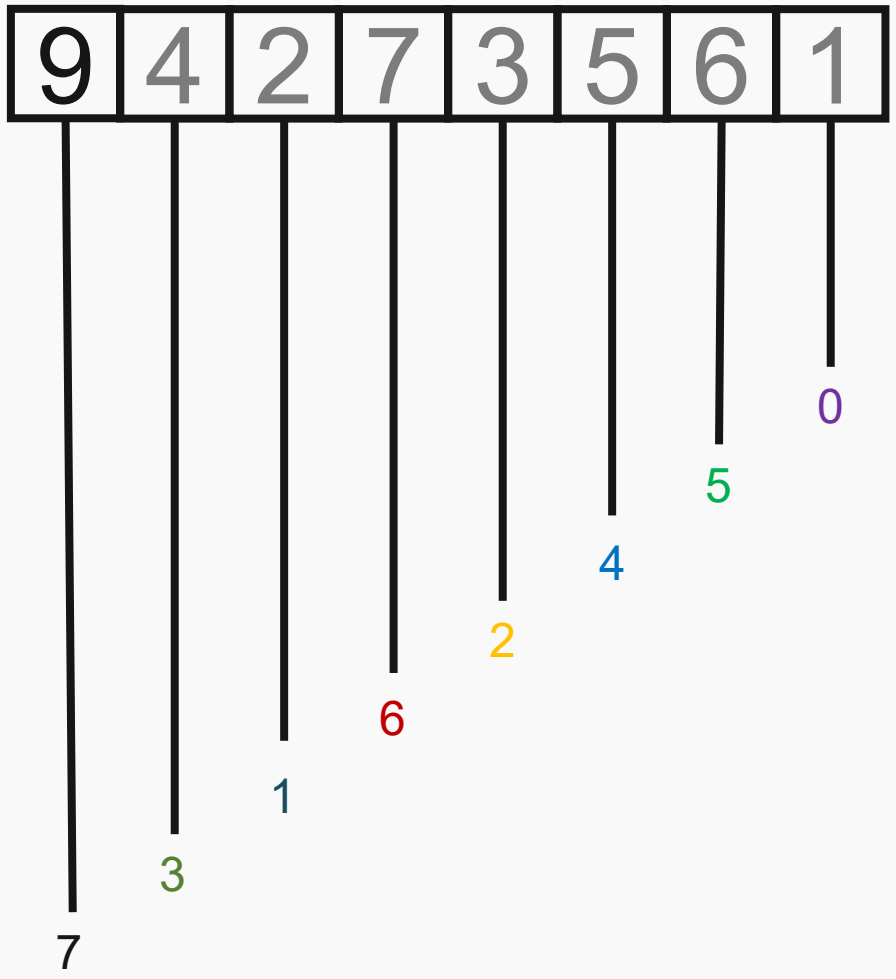


Count Sort / Rank Sort

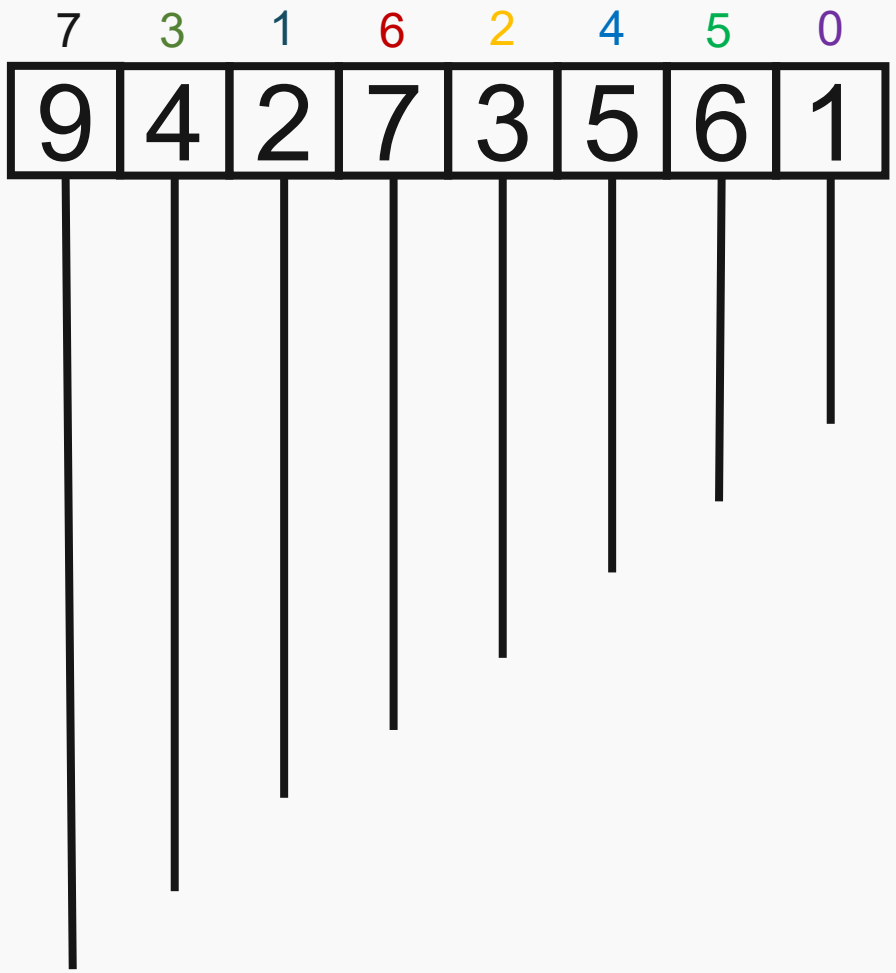


Câte numere sunt mai mici decât mine?

Count Sort / Rank Sort

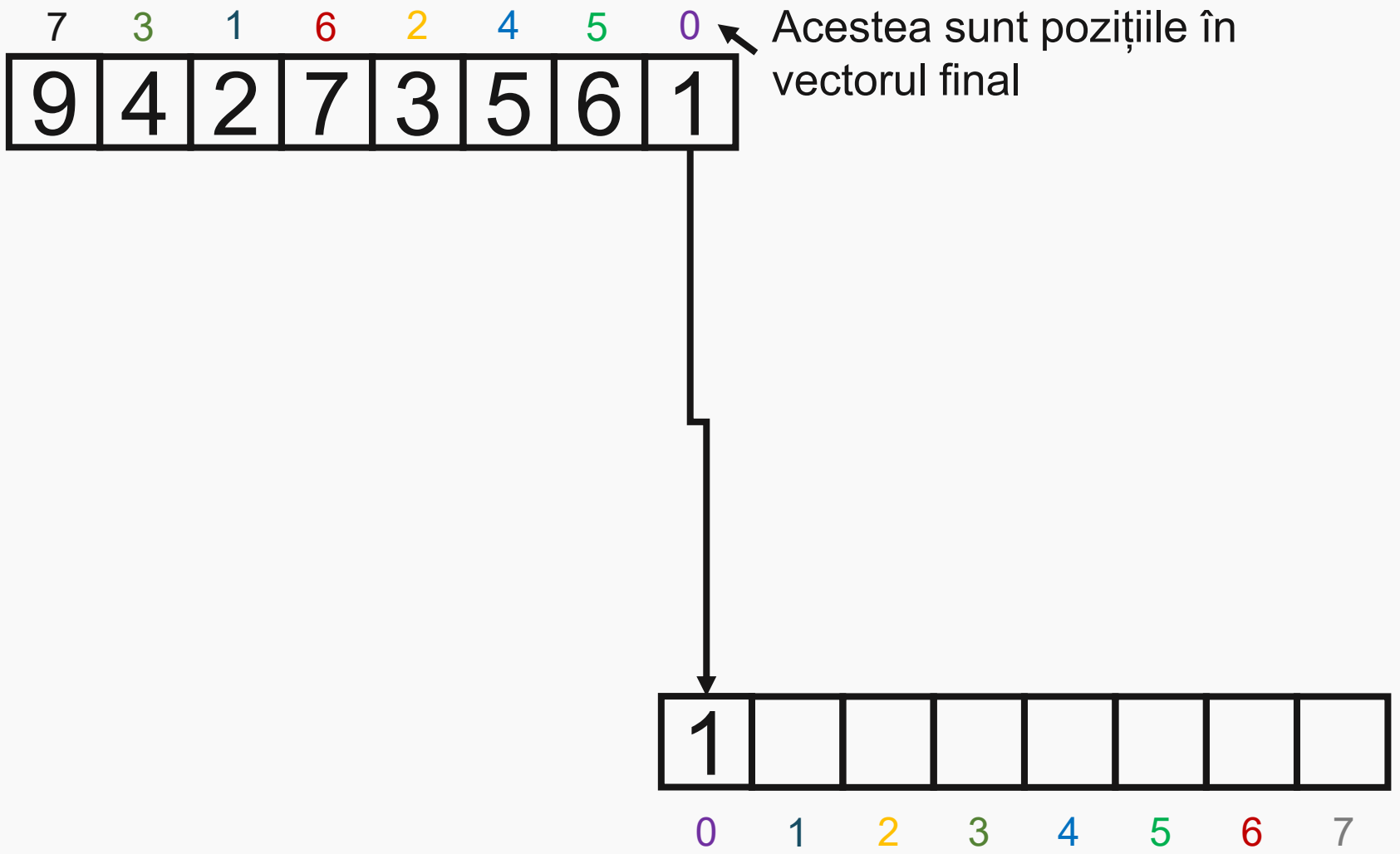


Count Sort / Rank Sort

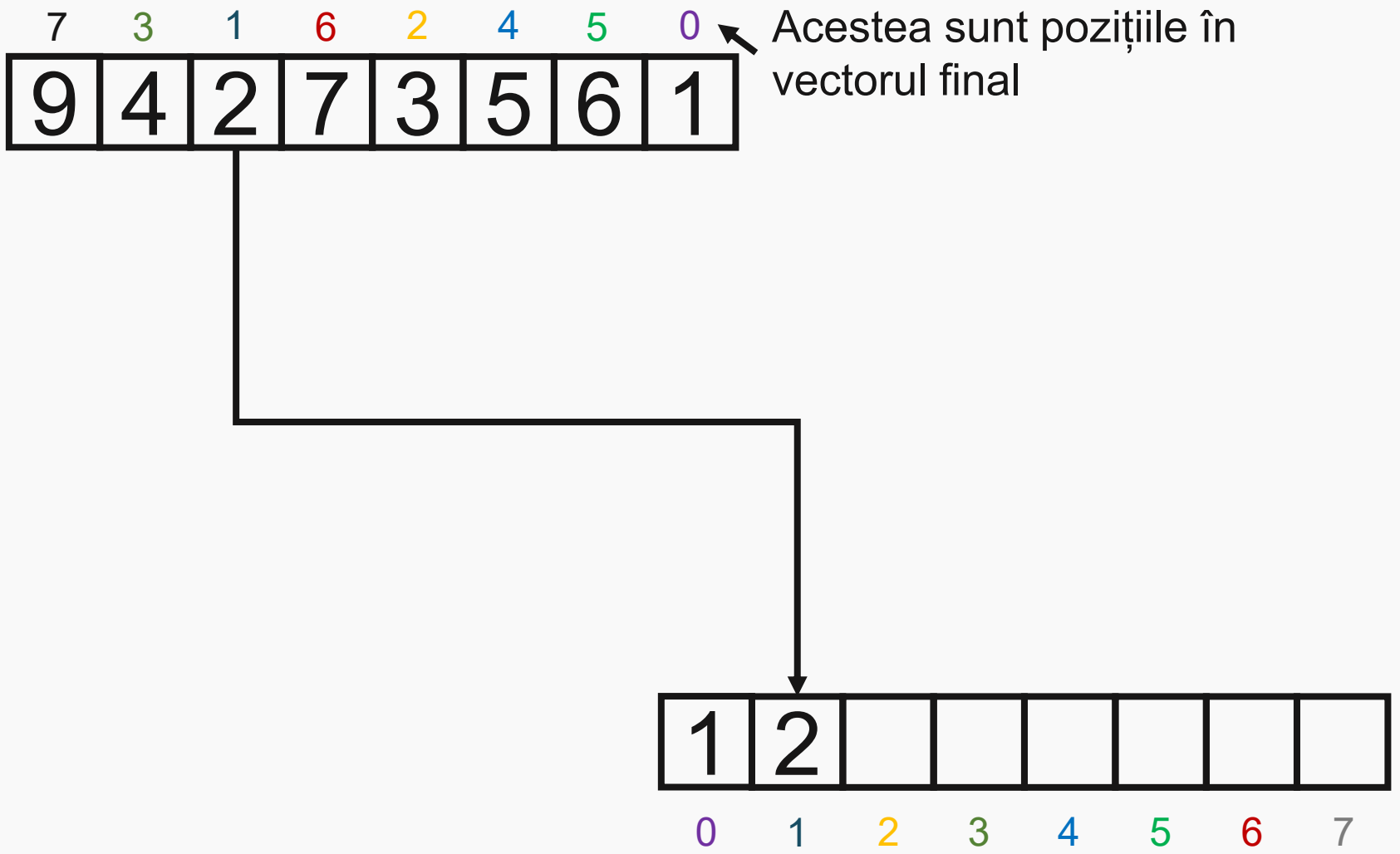


Acestea sunt pozițiile în vectorul final

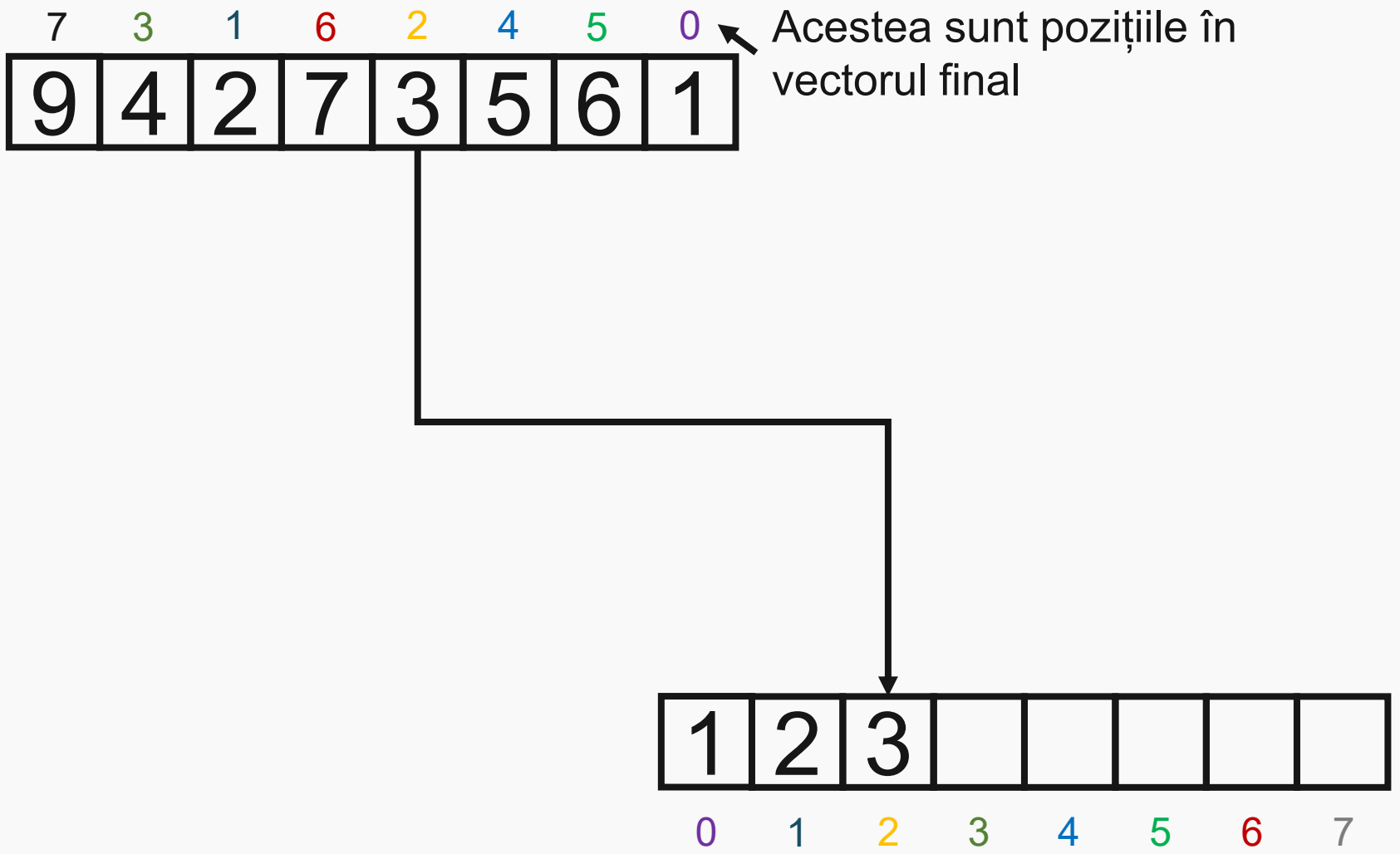
Count Sort / Rank Sort



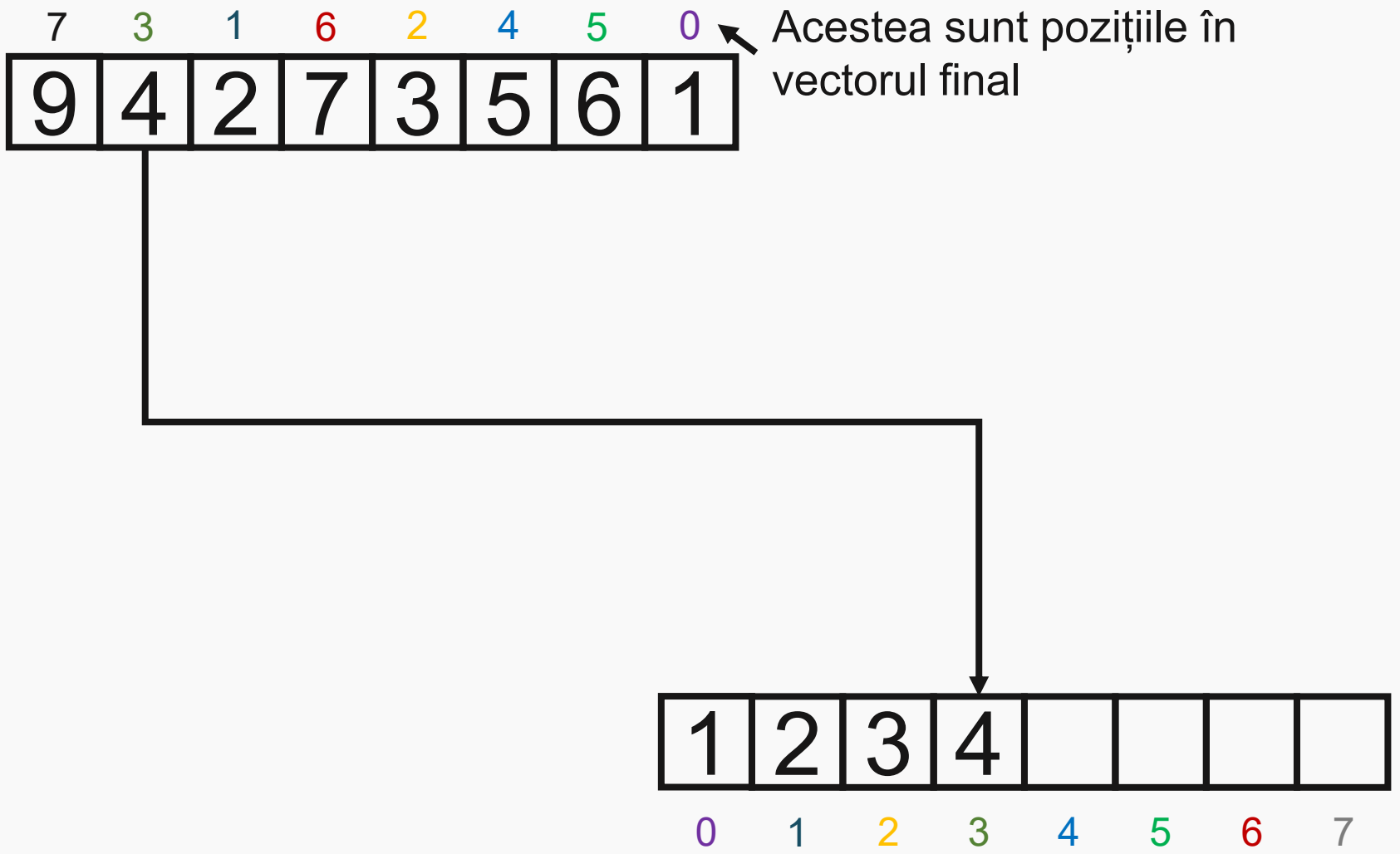
Count Sort / Rank Sort



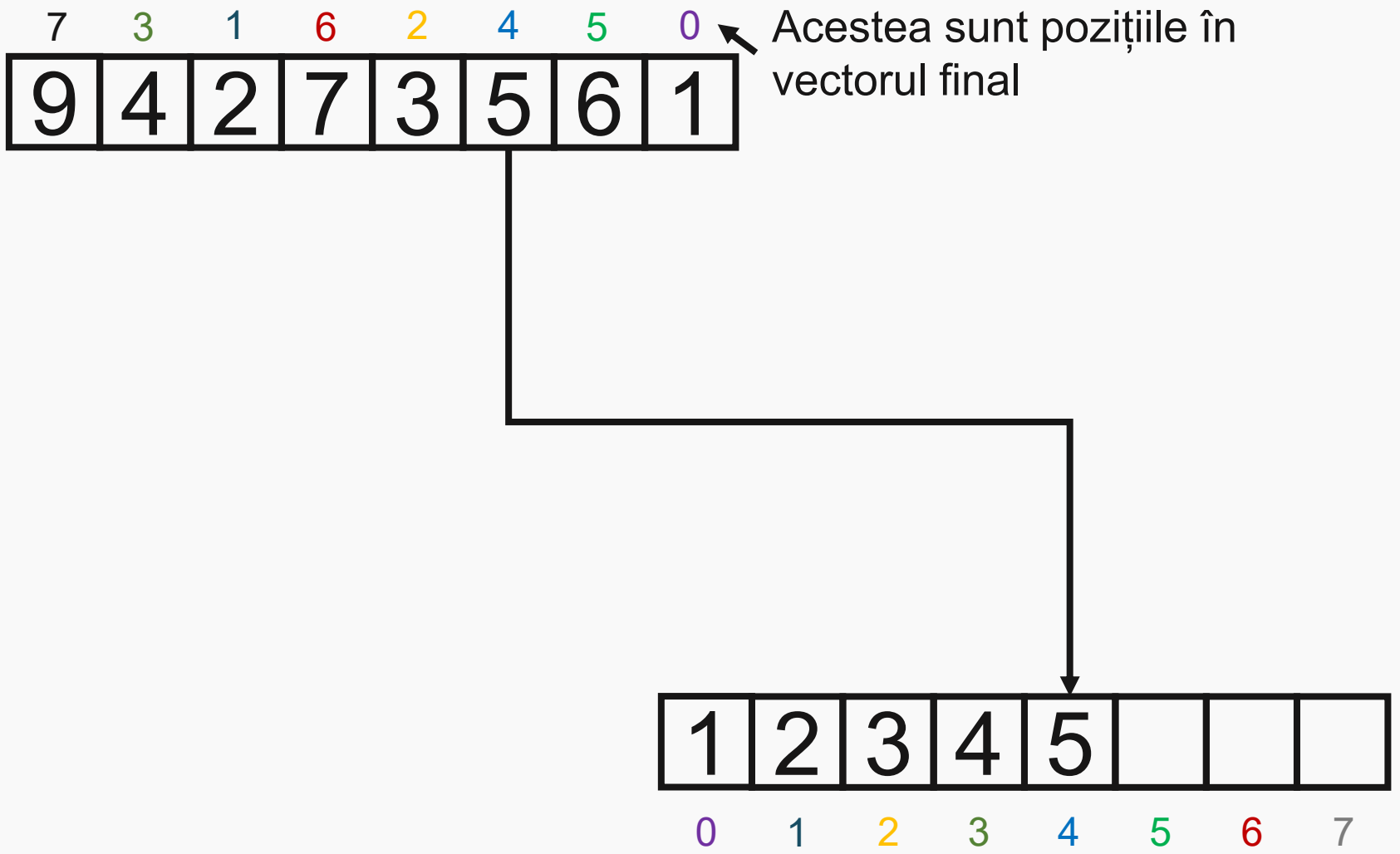
Count Sort / Rank Sort



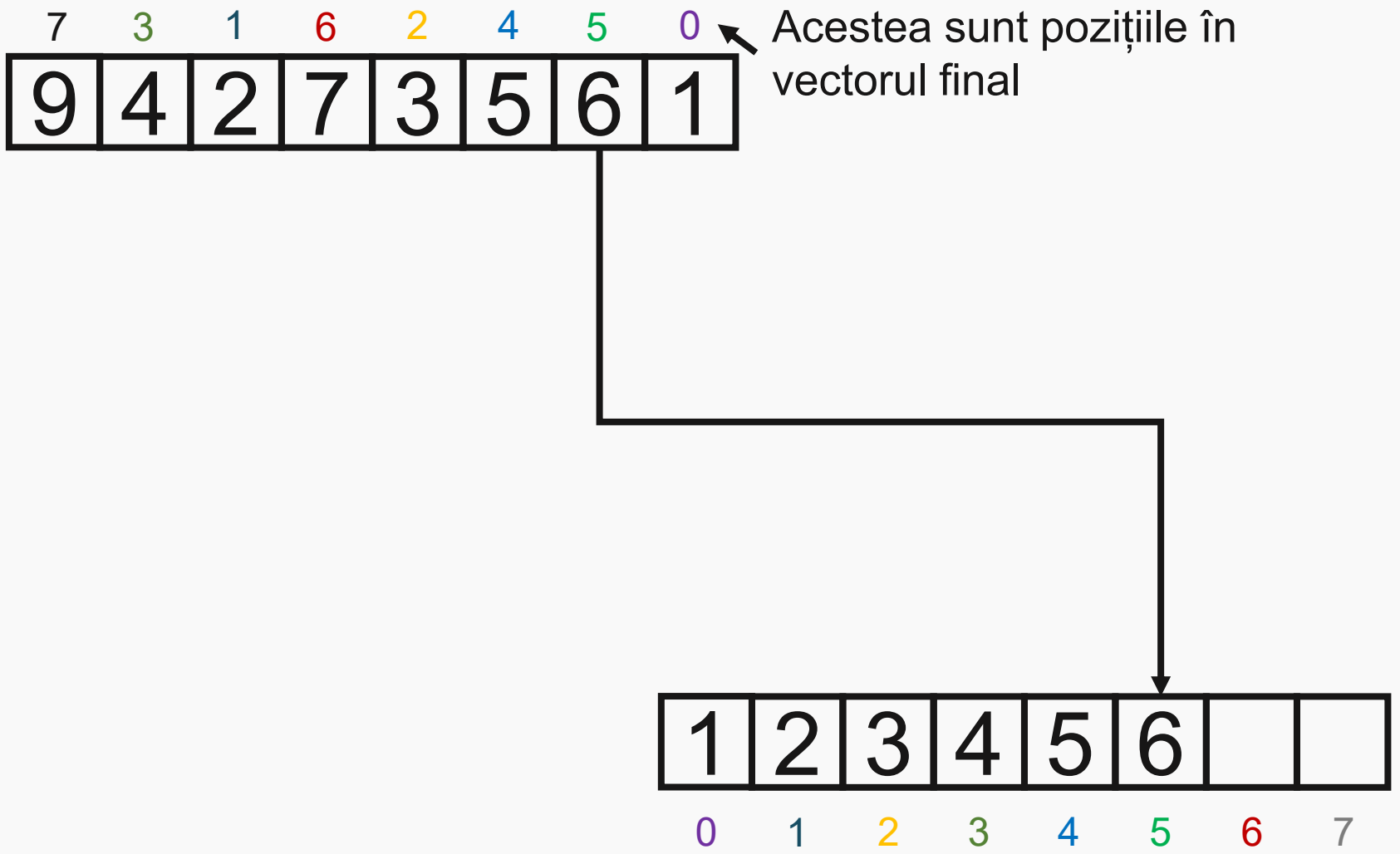
Count Sort / Rank Sort



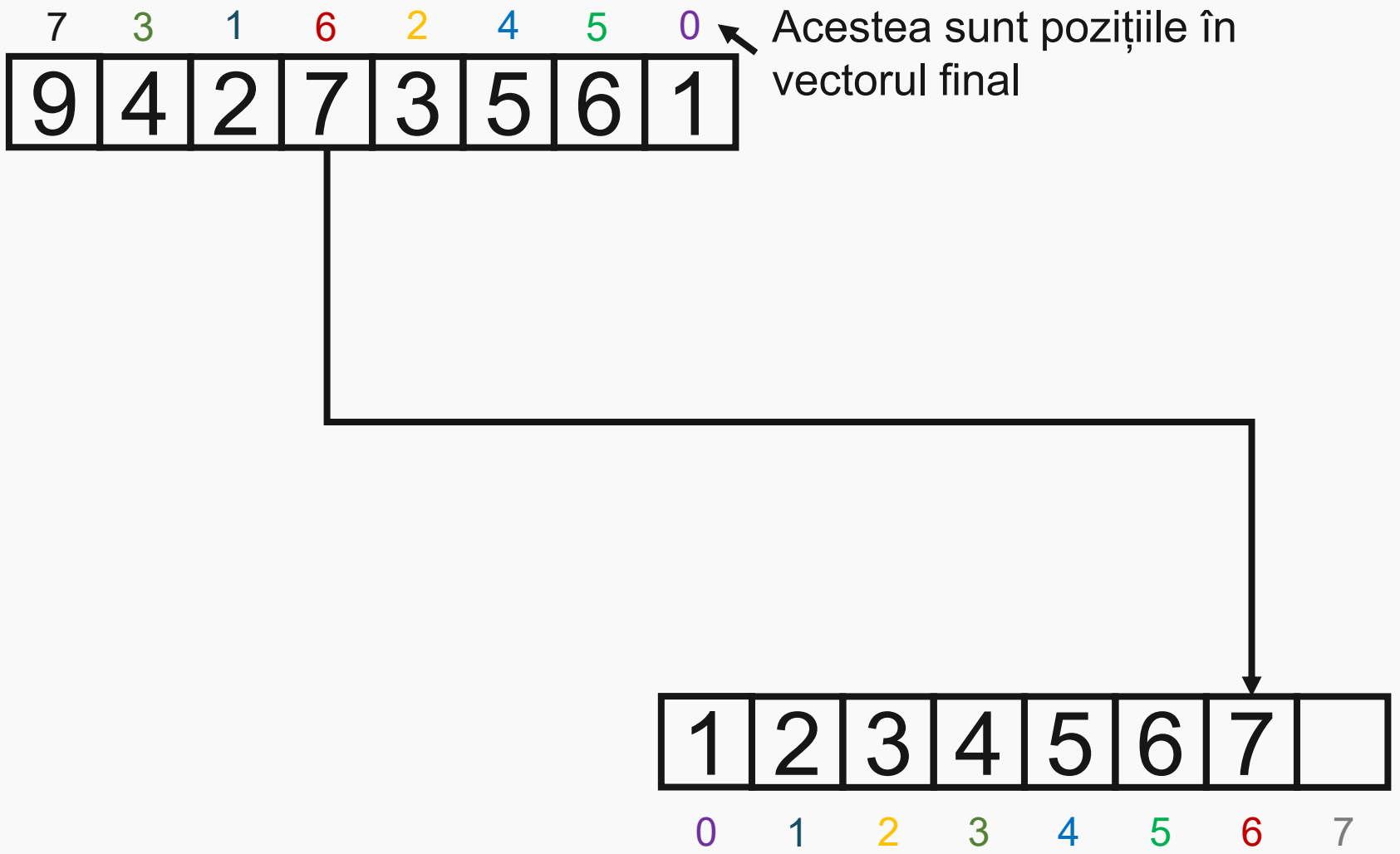
Count Sort / Rank Sort



Count Sort / Rank Sort



Count Sort / Rank Sort



Count Sort / Rank Sort

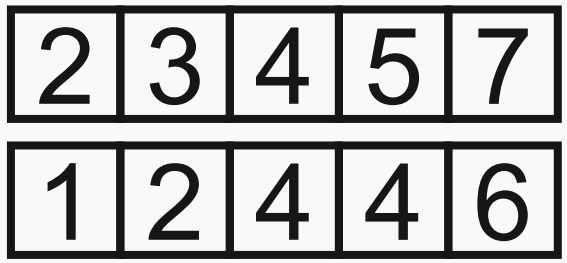
Acestea sunt pozițiile în vectorul final

Complexitate?

Complexitate?

$$O(N^2)$$

Algoritm merge



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

Algoritm merge

2	3	4	5	7
1	2	4	4	6

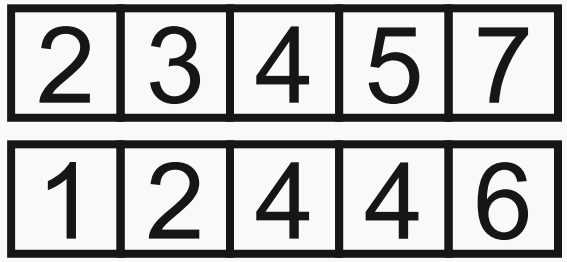
Soluție:

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

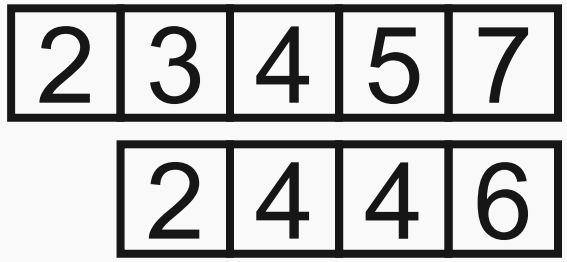
Complexitate: $\boldsymbol{O(N)}$

1	2	2	3	4	4	4	5	6	7

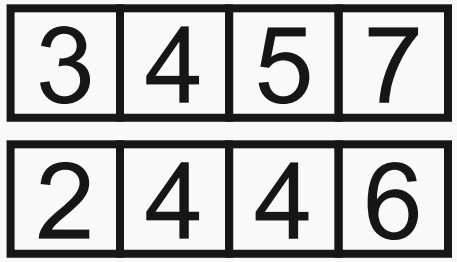
Algoritm merge



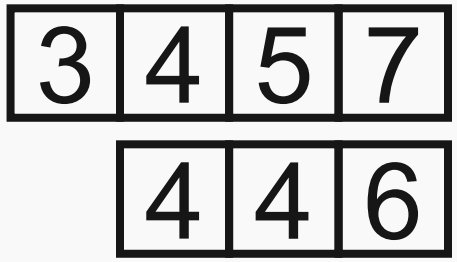
Algoritm merge



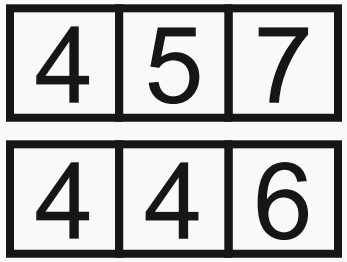
Algoritm merge

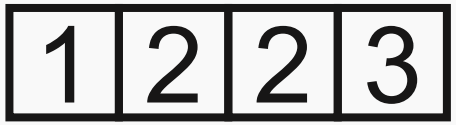


Algoritm merge

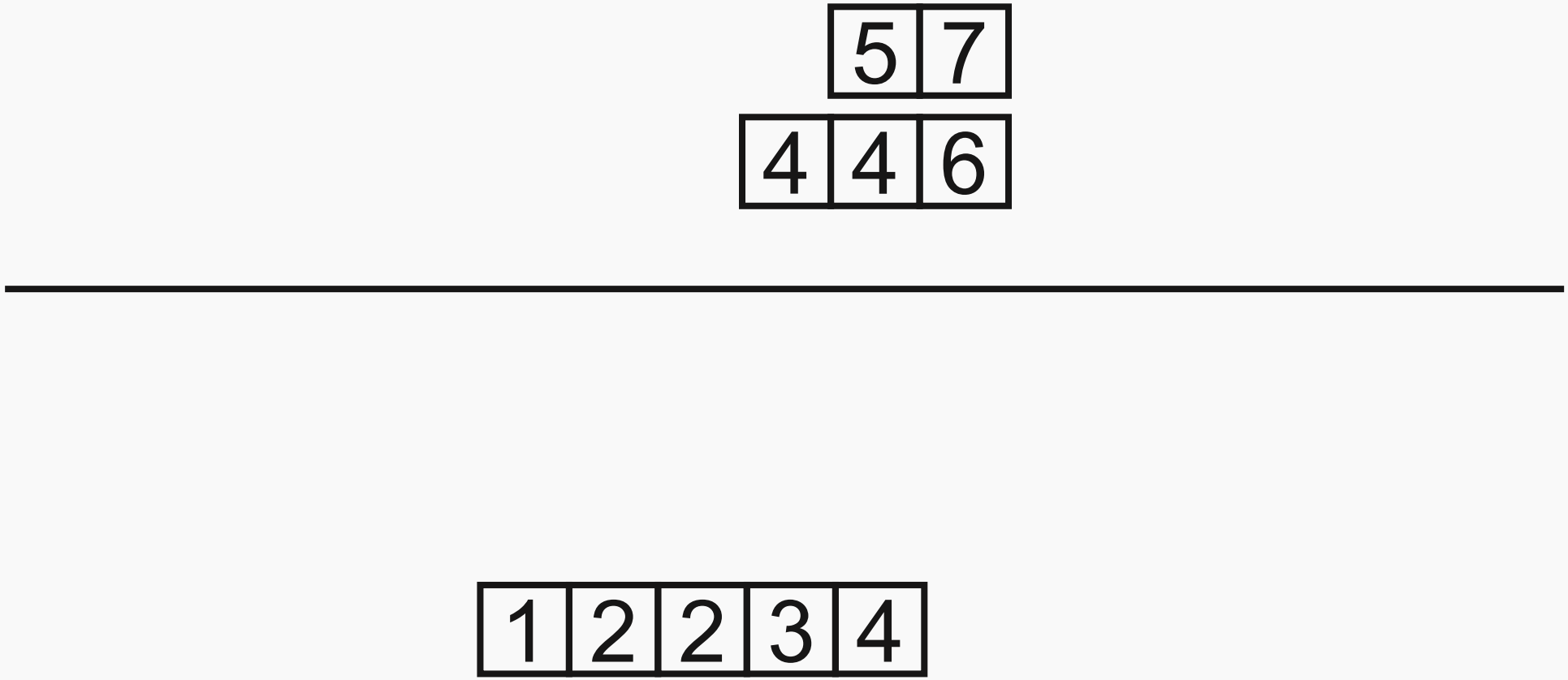


Algoritm merge

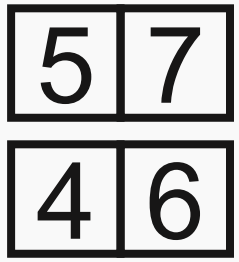




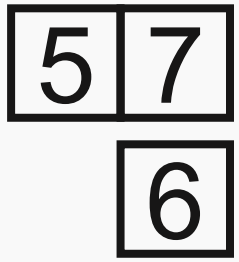
Algoritm merge



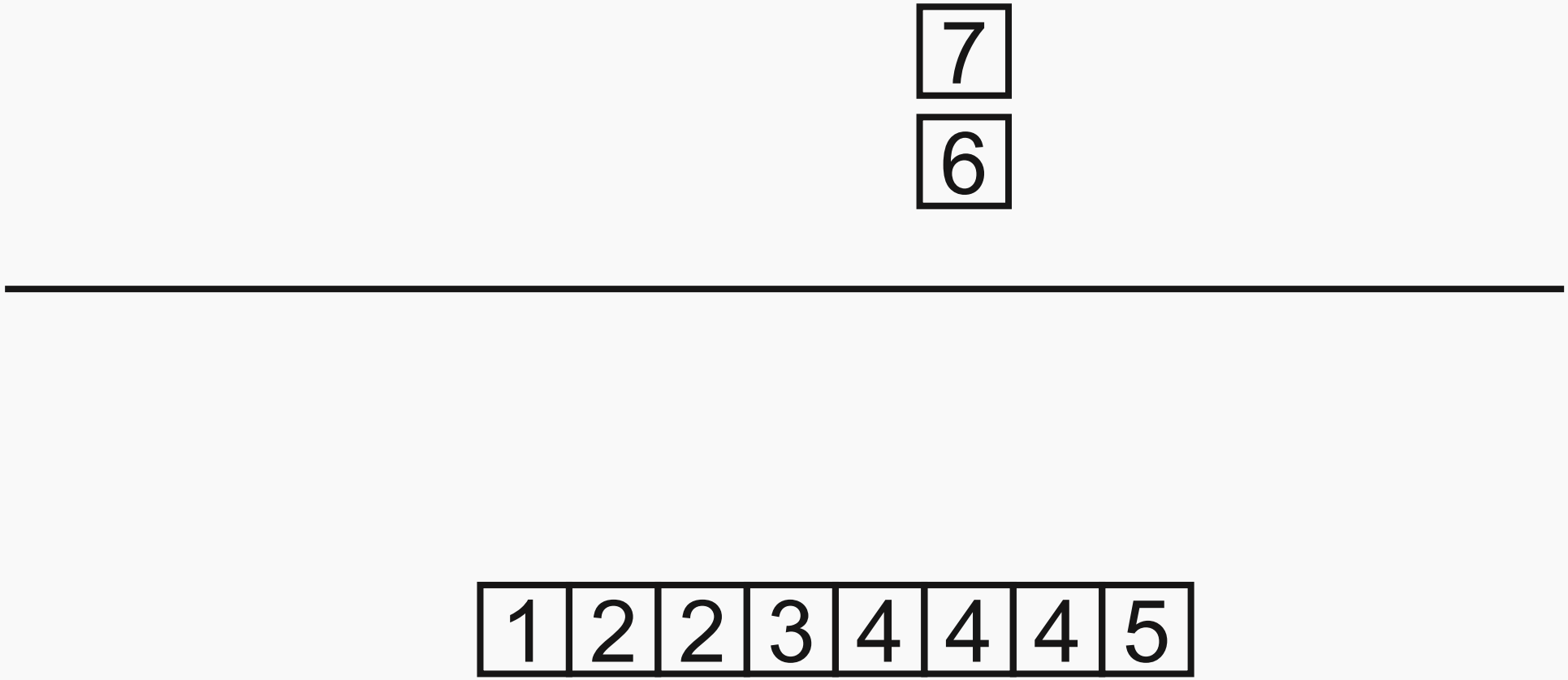
Algoritm merge



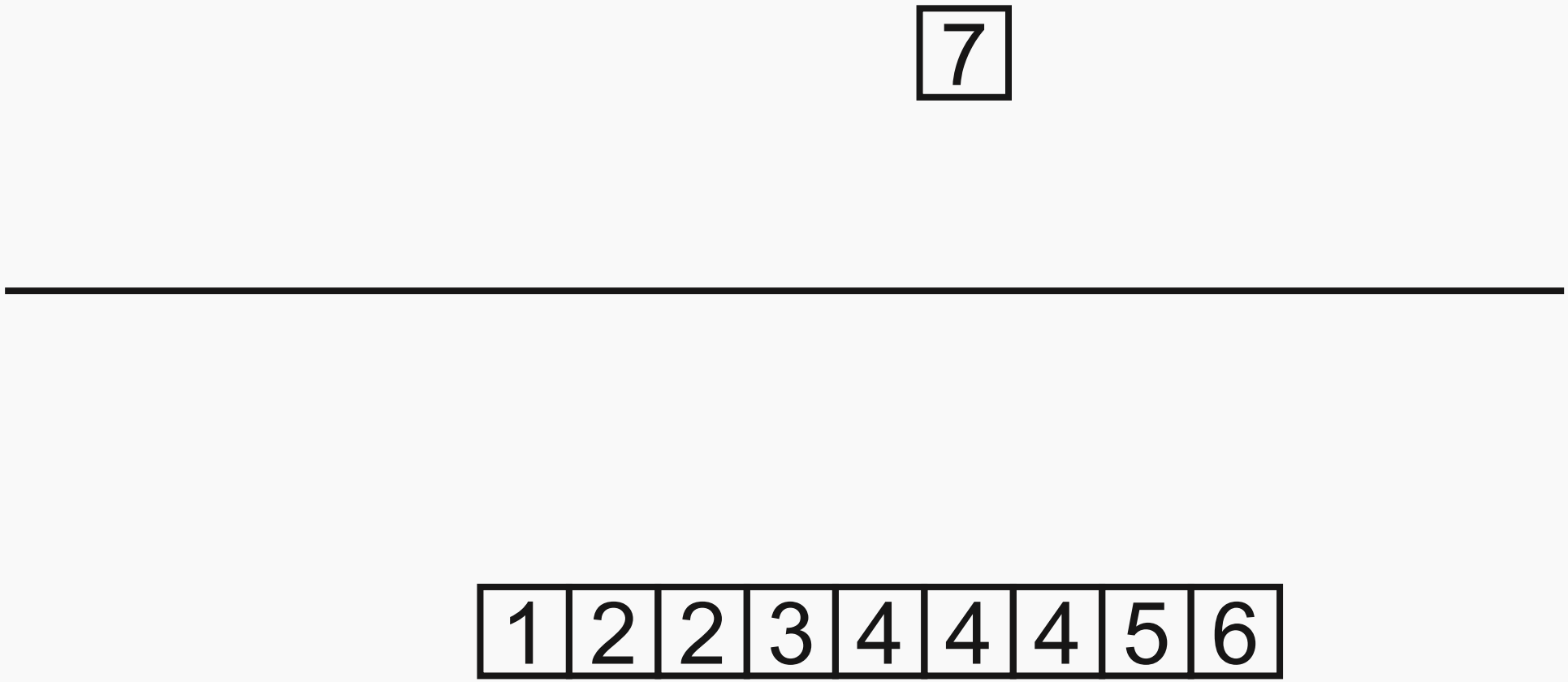
Algoritm merge



Algoritm merge

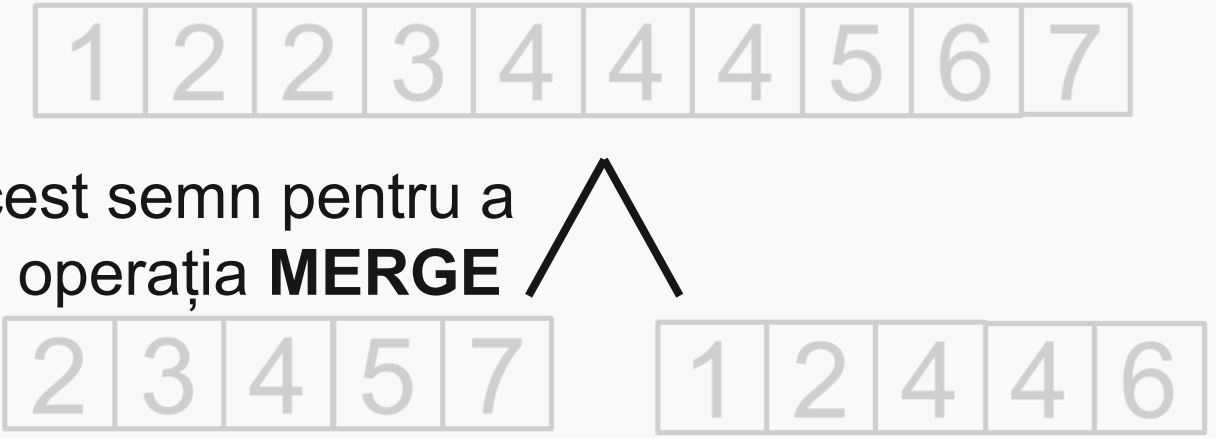


Algoritm merge

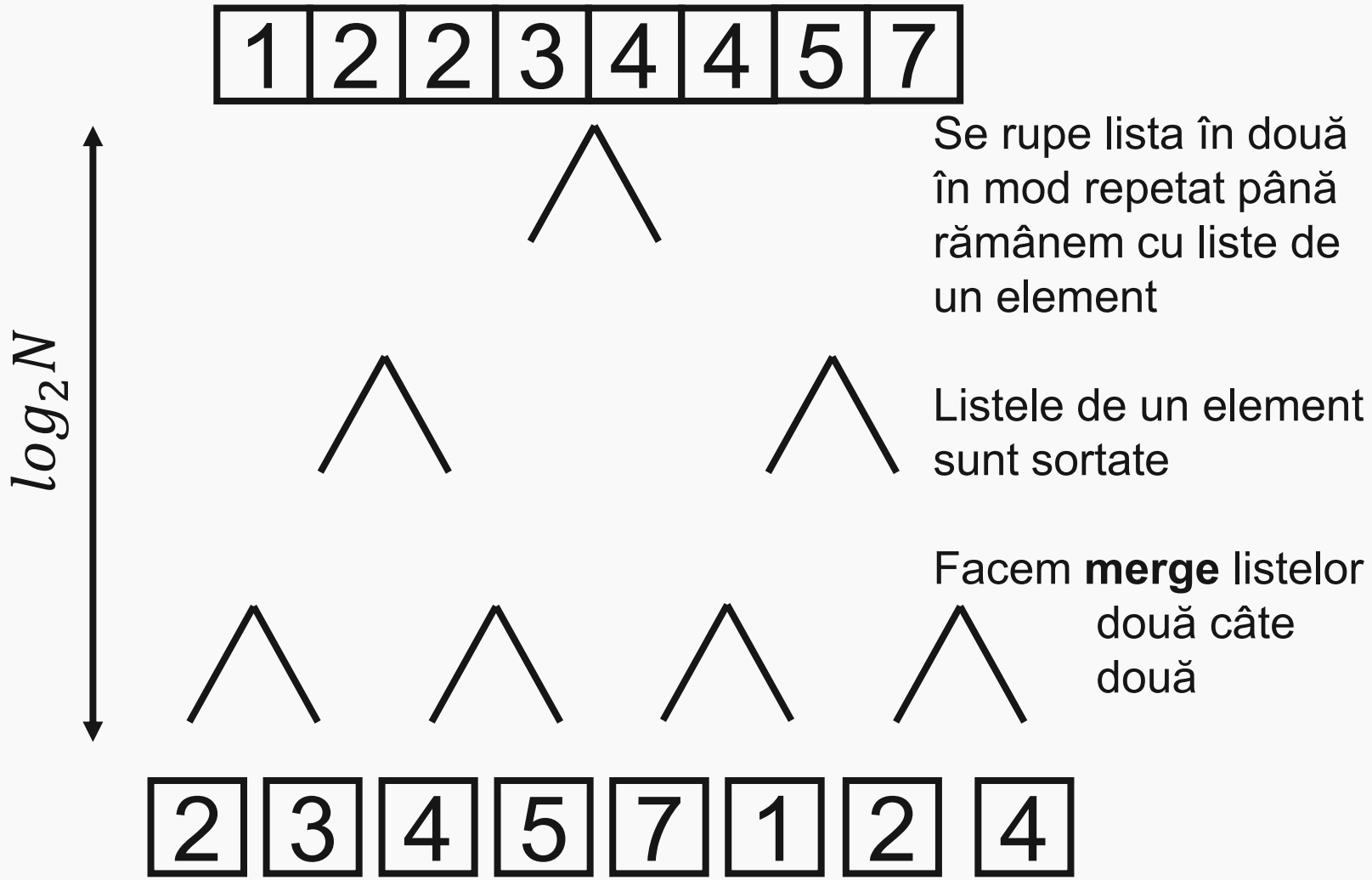


Algoritm merge

Algoritm merge

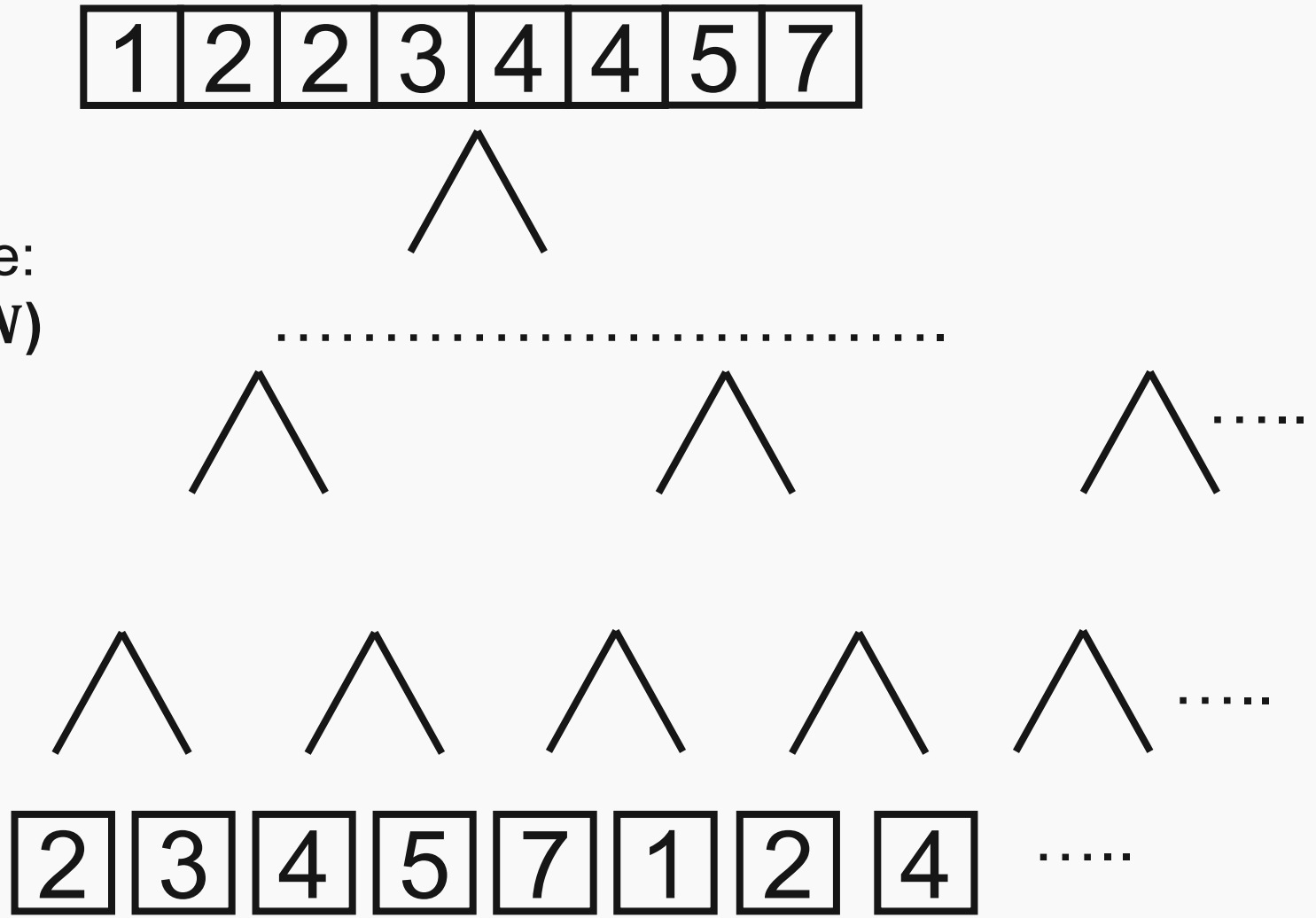


Merge sort

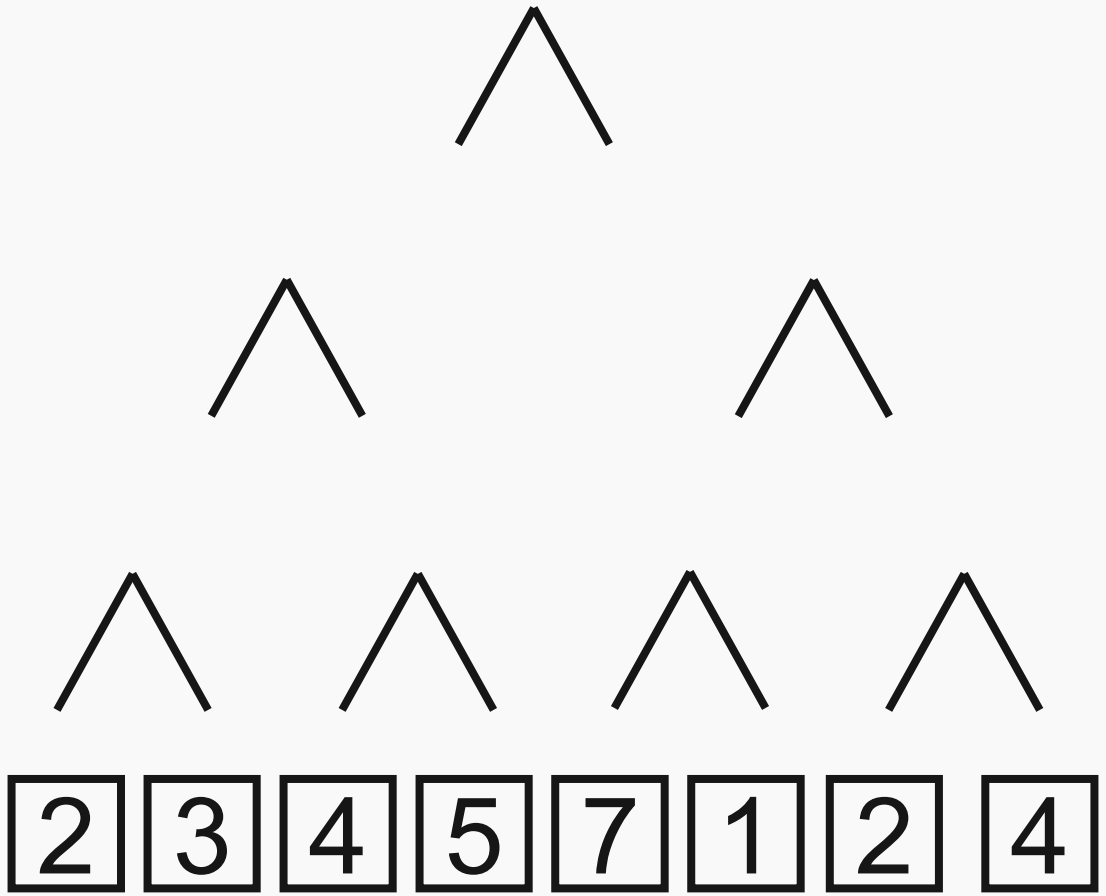

1 2 2 3 4 4 5 7
$log_2 N$
Se rupe lista în două în mod repetat până rămânem cu liste de un element
Listele de un element sunt sortate
Facem **merge** listelor două câte două
2 3 4 5 7 1 2 4

Merge sort

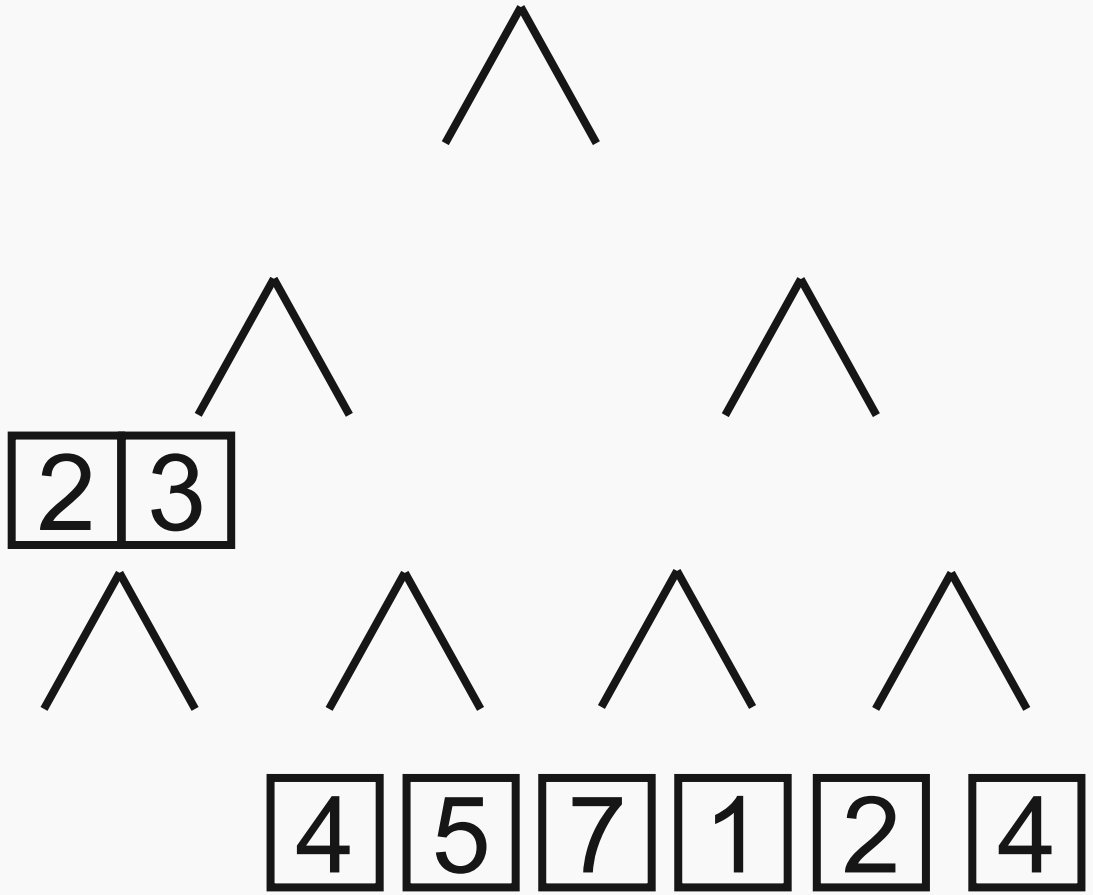
Complexitate:
O($N * log_2 N$)



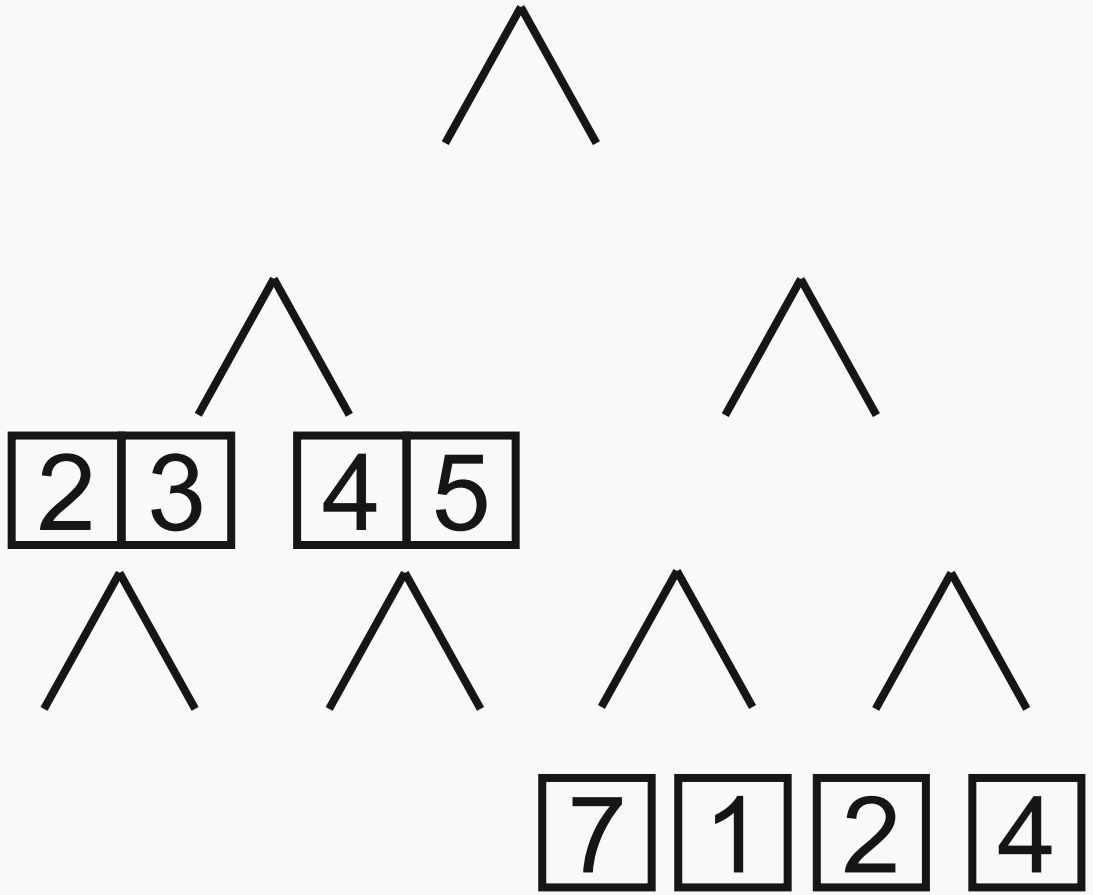
Merge sort



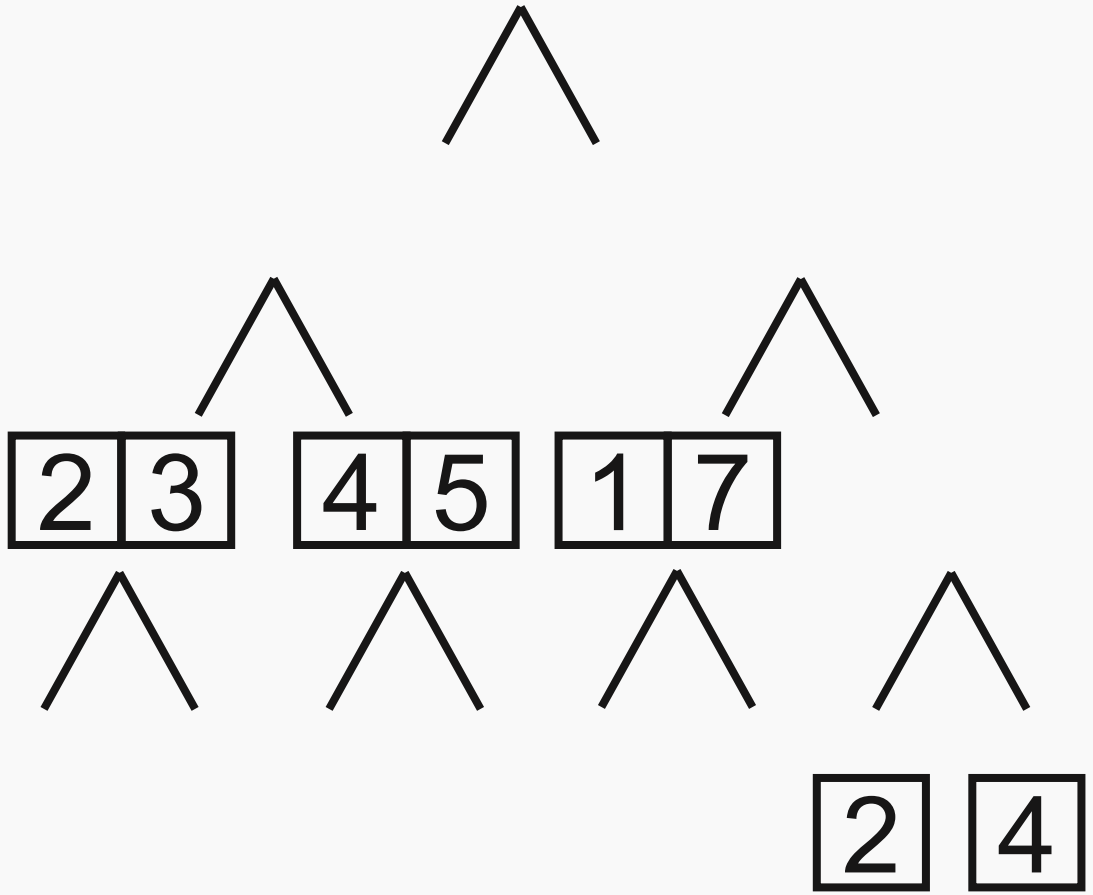
Merge sort



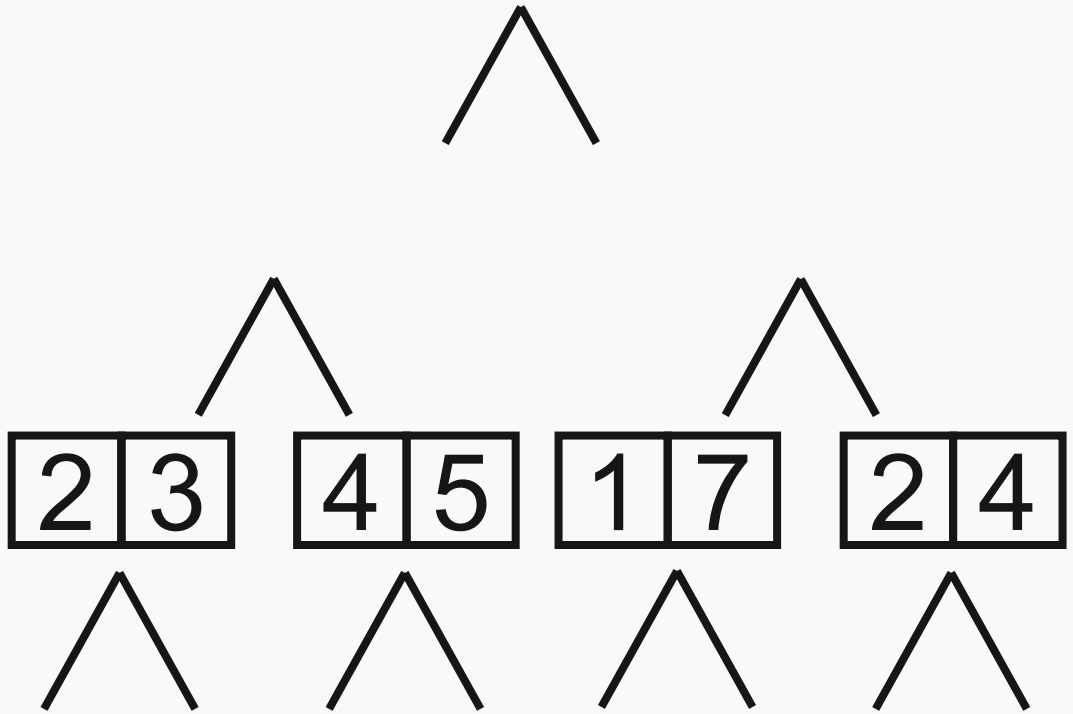
Merge sort



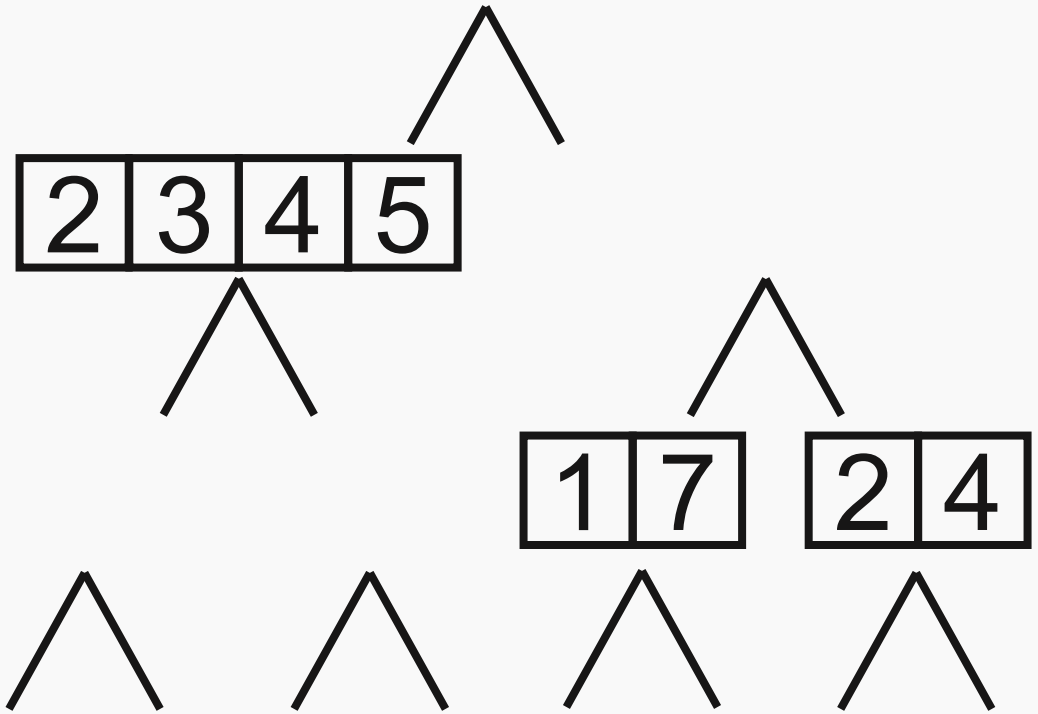
Merge sort



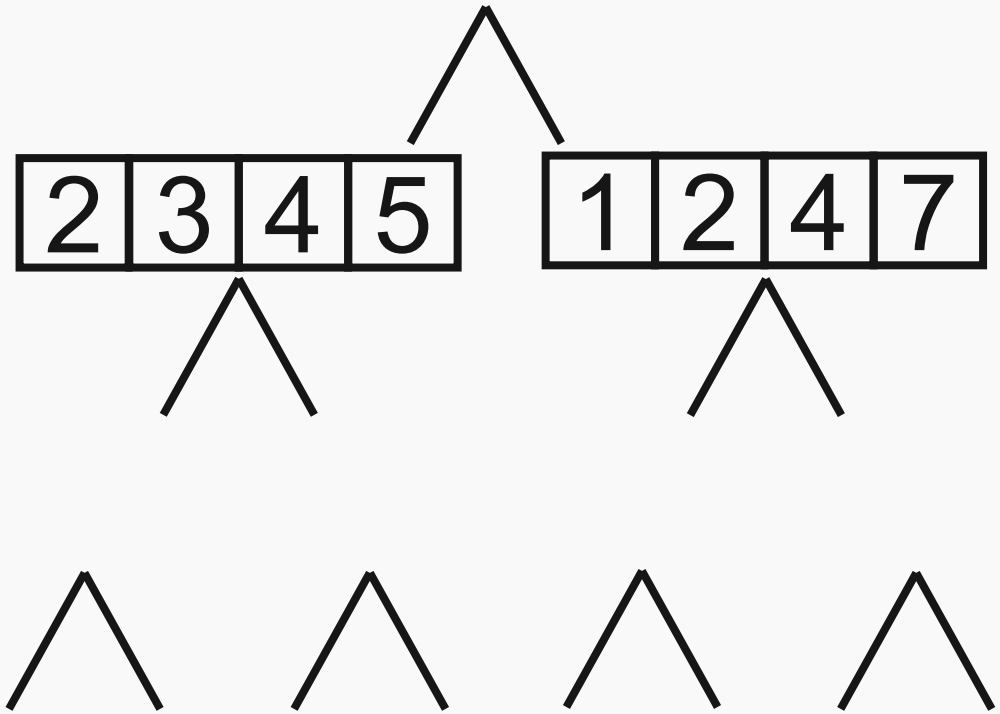
Merge sort



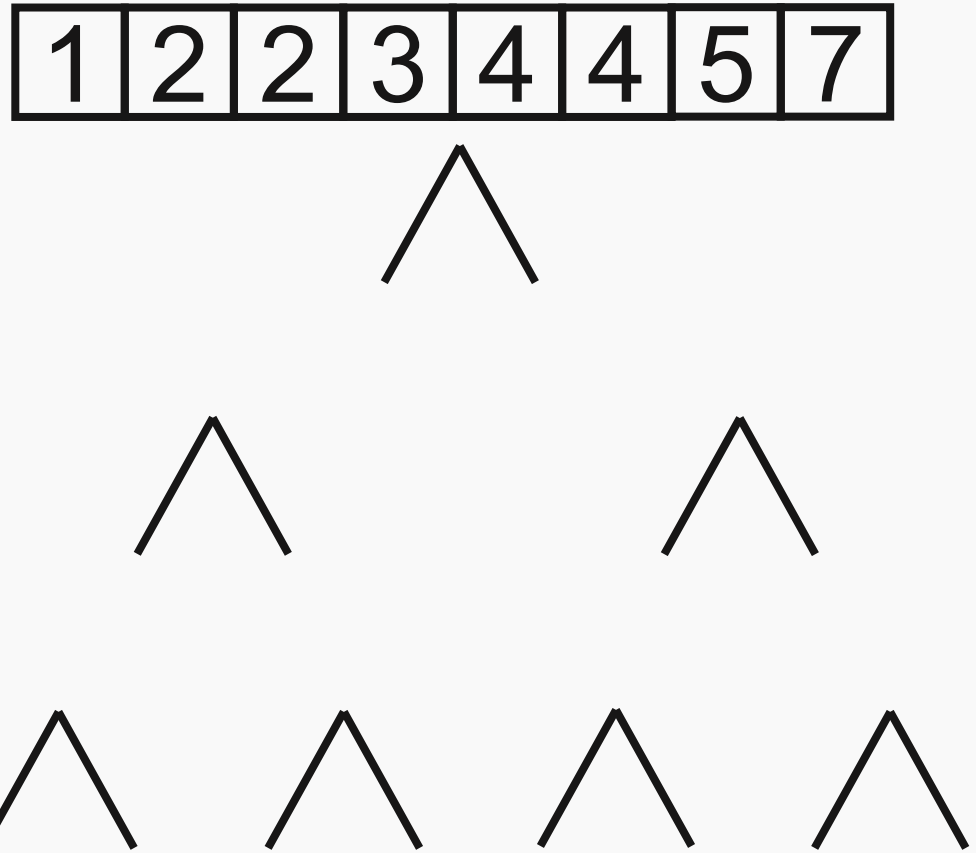
Merge sort



Merge sort



Merge sort



Complexitate?

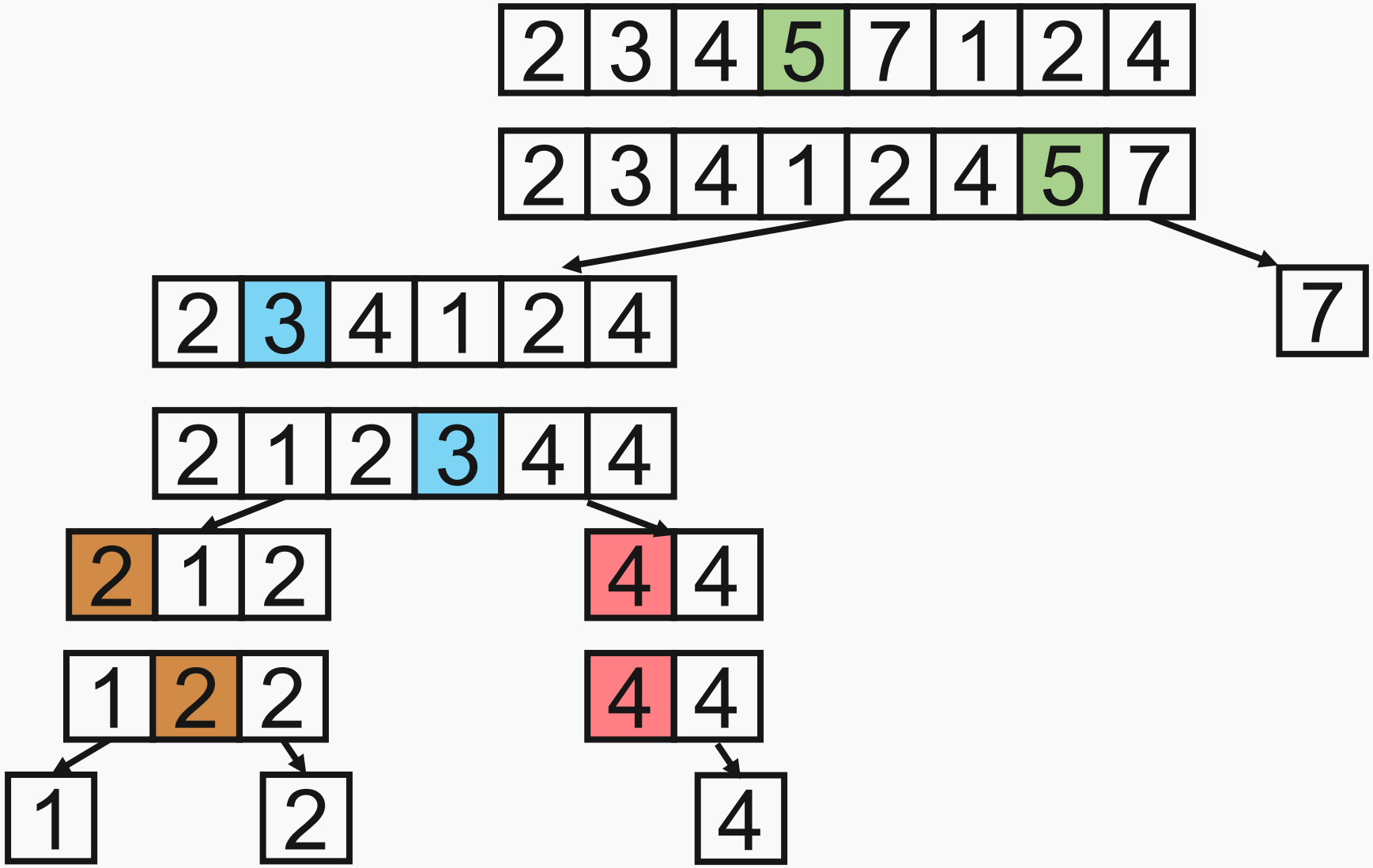
Complexitate?

$$O(N\ log(N))$$

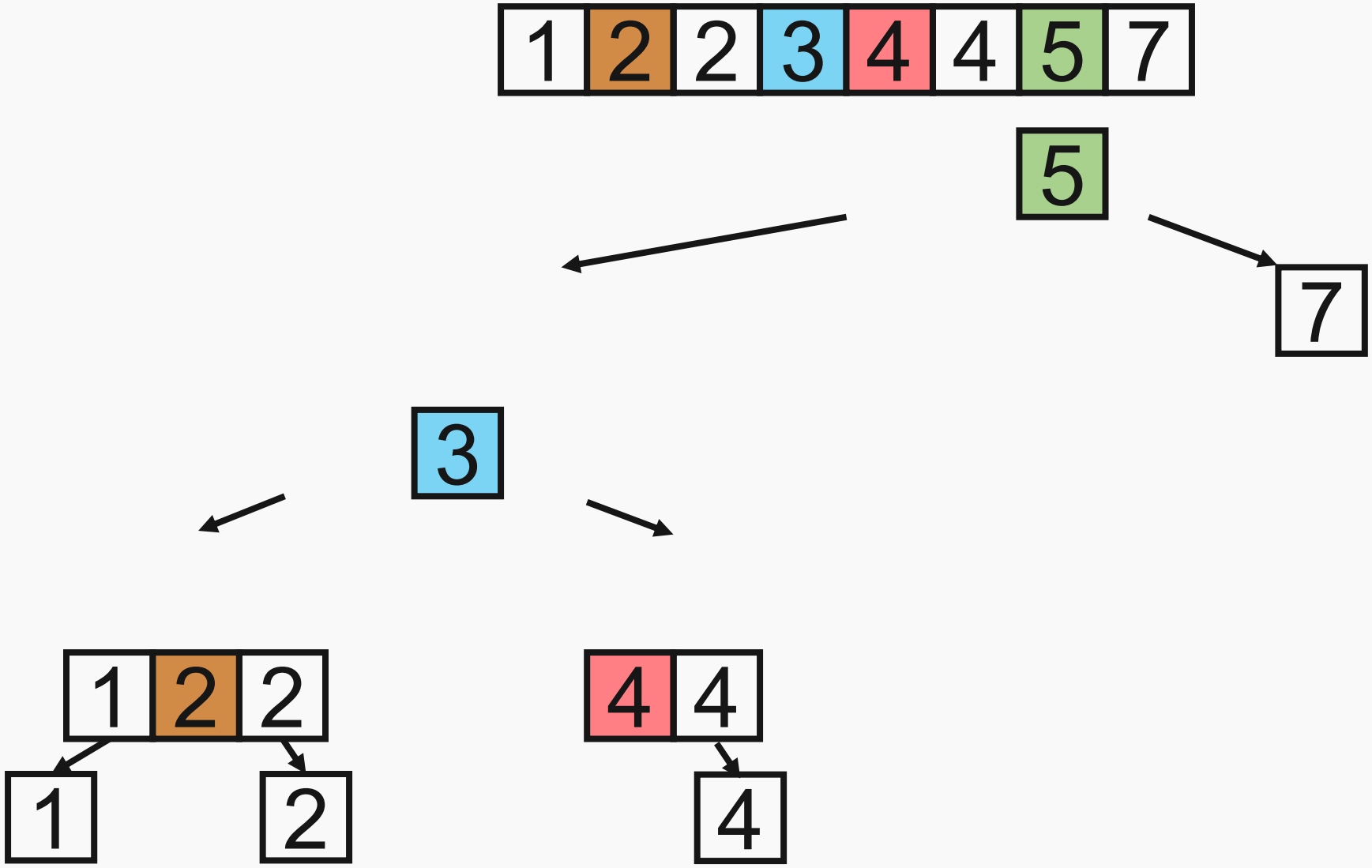
QuickSort

- Se alege un pivot (chiar și random).
- Se pun toate elementele mai mici decât pivotul în stânga sa.
- Se pun toate elementele mai mari decât pivotul în dreapta sa.
  - În această operație pivotul poate fi și el mutat pentru a face loc.
- Se apelează recursiv QuickSort pentru elementele din stânga respectiv dreapta pivotului.

QuickSort



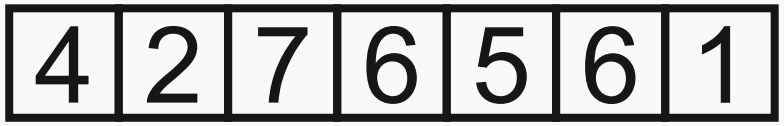
QuickSort

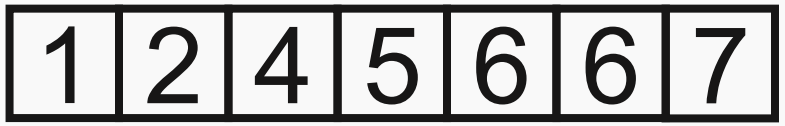


Complexitate?

$$O(N\ log(N))$$

Radix Sort





Radix Sort

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

Radix Sort

4	1	0	***0***
2	0	1	***0***
7	1	1	***1***
6	1	1	***0***
5	1	0	***1***
6	1	1	***0***
1	0	0	***1***

Sortează după cel mai nesemnificativ bit

Radix Sort

4	1	0	***0***
2	0	1	***0***
6	1	1	***0***
6	1	1	***0***
7	1	1	***1***
5	1	0	***1***
1	0	0	***1***

Sortează după cel mai nesemnificativ bit

Radix Sort

4	1	***0***	0
2	0	***1***	0
6	1	***1***	0
6	1	***1***	0
7	1	***1***	1
5	1	***0***	1
1	0	***0***	1

Sortează după al doilea cel mai nesemnificativ bit

Radix Sort

4	1	***0***	0
5	1	***0***	1
1	0	***0***	1
2	0	***1***	0
6	1	***1***	0
6	1	***1***	0
7	1	***1***	1

Sortează după al doilea cel mai nesemnificativ bit

Radix Sort

4	***1***	0	0
5	***1***	0	1
1	***0***	0	1
2	***0***	1	0
6	***1***	1	0
6	***1***	1	0
7	***1***	1	1

Sortează după cel mai semnificativ bit

Radix Sort

1	***0***	0	1
2	***0***	1	0
4	***1***	0	0
5	***1***	0	1
6	***1***	1	0
6	***1***	1	0
7	***1***	1	1

Sortează după cel mai semnificativ bit

Complexitate?

Complexitate?

$$O(bN)$$