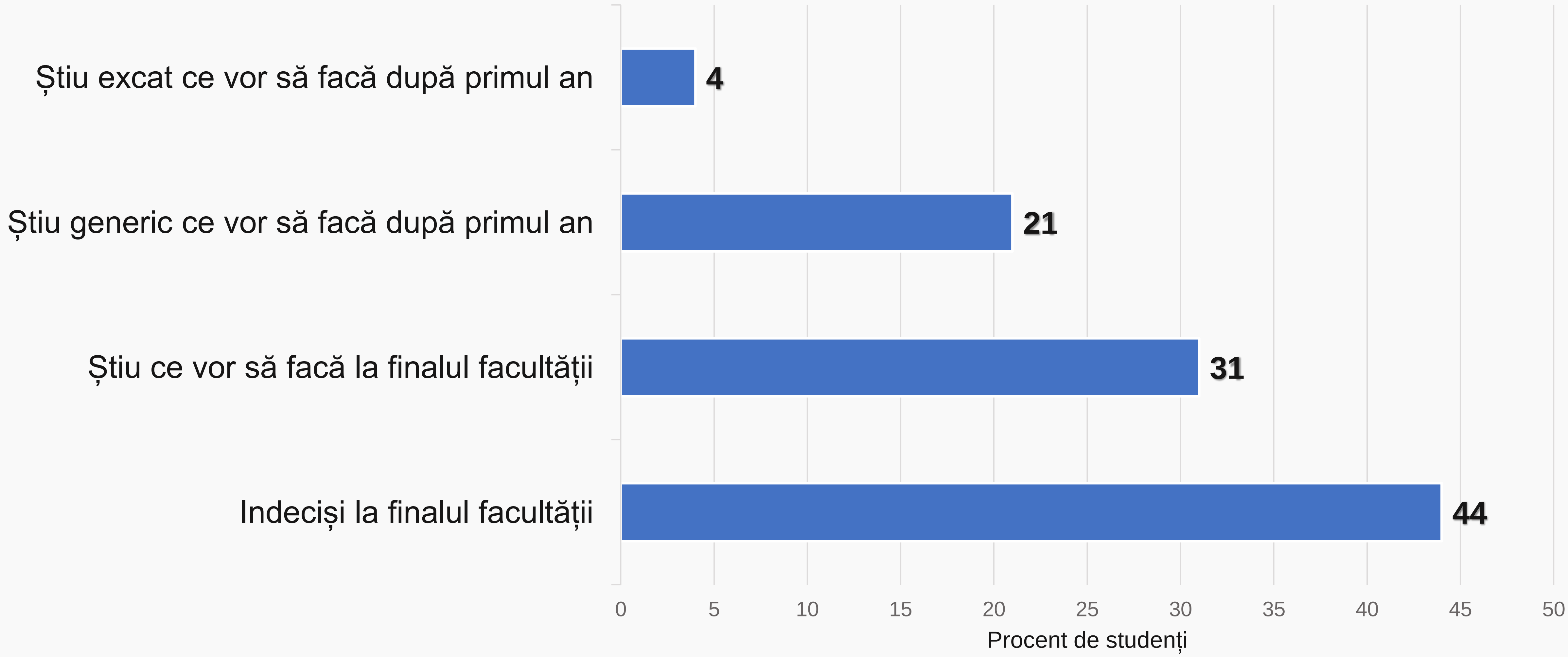
Structuri de date și algoritmi
Sortari

Dr. Ing. Alexandra Mocanu – alexa.mihaita@gmail.com

Recapitulare

- Importanță
  - De ce credeți ca este nevoie să studiați algortmi de sortare?
- Exemple practice
  - Unde sunt utili algoritmii de sortare?



Știu excat ce vor să facă după primul an
Știu generic ce vor să facă după primul an
Știu ce vor să facă la finalul facultății
Indeciși la finalul facultății
4
21
31
44
0
5
10
15
20
25
30
35
40
45
50
Procent de studenți

Bubble sort

Complexitate: O(N^2)

for i=0, N-1
    for j=N, i+1
        if(not in order)
            then swap

5	2	6	1	9
2	5	6	1	9
2	5	6	1	9
2	5	6	1	9
2	5	1	6	9
2	5	1	6	9
2	5	1	6	9
2	1	5	6	9
2	1	5	6	9
2	1	5	6	9
1	2	5	6	9

Selection Sort

Complexitate: O(N^2)

```
for i=0, N-1
        for j=i+1, N
                find min
        swap (current, min)
```

5	2	6	1	9
2	5	6	1	9
1	5	6	2	9
1	2	6	5	9
1	2	5	6	9

Count Sort/ Rank Sort

Complexitate: O(N^2)

```
for i=0, N-1
        for j=i+1, N
                if(bigger)
                        increase index
```

5	2	6	1	9
2	1	3	0	4

0	1	2	3	4
1	2	5	6	9

MergeSort

Complexitate: O($N \log(N)$)

```
for i=0, N-1
        if(list1(j) < list2(k))
                rez(i) = list1(j++);
        else
                rez(i) = list2(k++);
```

2	5	1	6	9	Merge
2	5	1	6	9	1
2	5		6	9	1, 2
	5		6	9	1,2,5
			6	9	1,2,5,6, 9

QuickSort

Complexitate: $O(N \log(N))$

```
while(! modificat){
    pick pivot
    for i=0, N-1
        if(element > pivot){
                switch;
                modified = true;
        }
}
```

5	2	6	1	9
5	2	6	1	9
2	1	5	6	9
1	2	5	6	9

RadixSort

Complexitate: O(bN)

```
for(j=0, b)
        for(i=0,N-1)
                if(list(i).bit_j < list(i).bit_j)
                switch
```

5	2	6	1	9
1	0	0	1	1
0	1	1	0	0
1	0	1	0	0
0	0	0	0	1

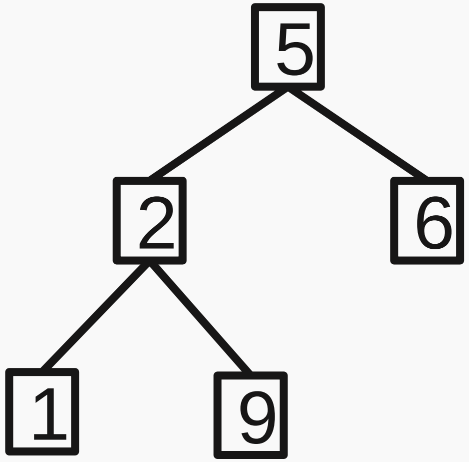
2	6	5	1	9
5	1	9	2	6
1	9	2	5	6
1	2	5	6	9

HeapSort

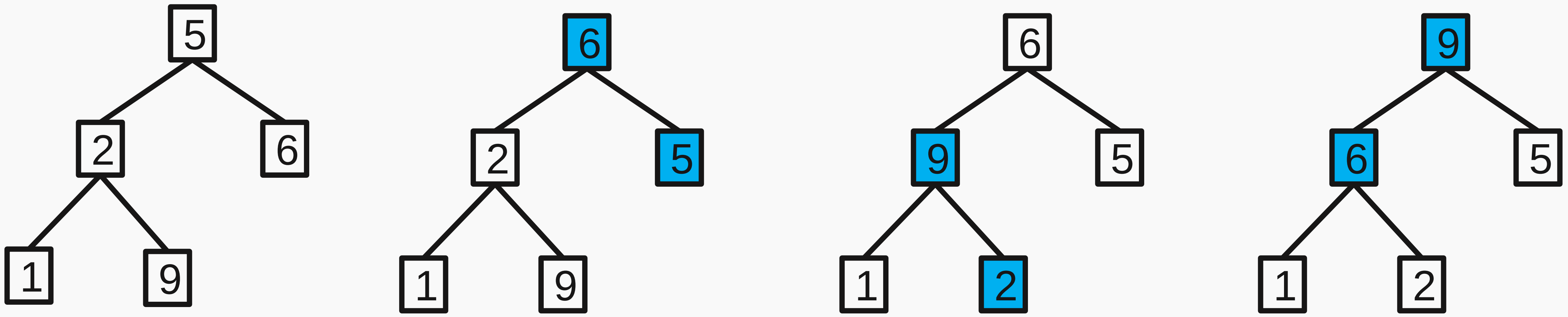
- Folosește o structură ajutătoare *Heap* de tip arbore binar in care fiecare nod al arborelui este un element din lista care urmează a fi sortată;
- Fazele algoritmului:
  - Popularea arborelui – stânga la dreapta;
  - Condiția Min/max - (nod părinte < ambii copii) | (nod părinte > ambii copii);
  - Popularea listei de elemente.

HeapSort– Popularea arborelui binar

5	2	6	1	9

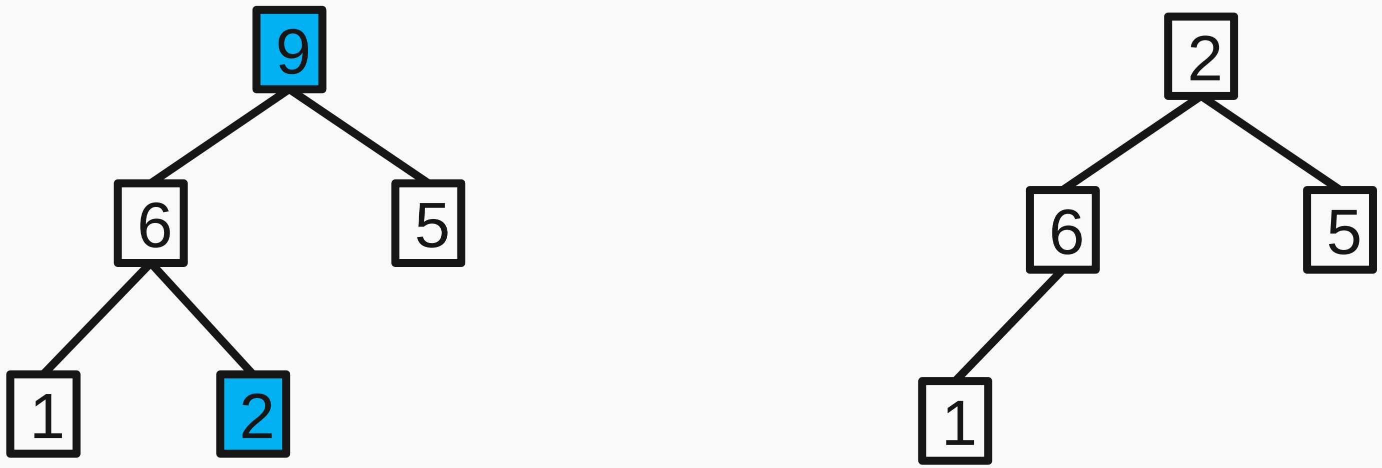


HeapSort– Condiția ~~Min~~/Max

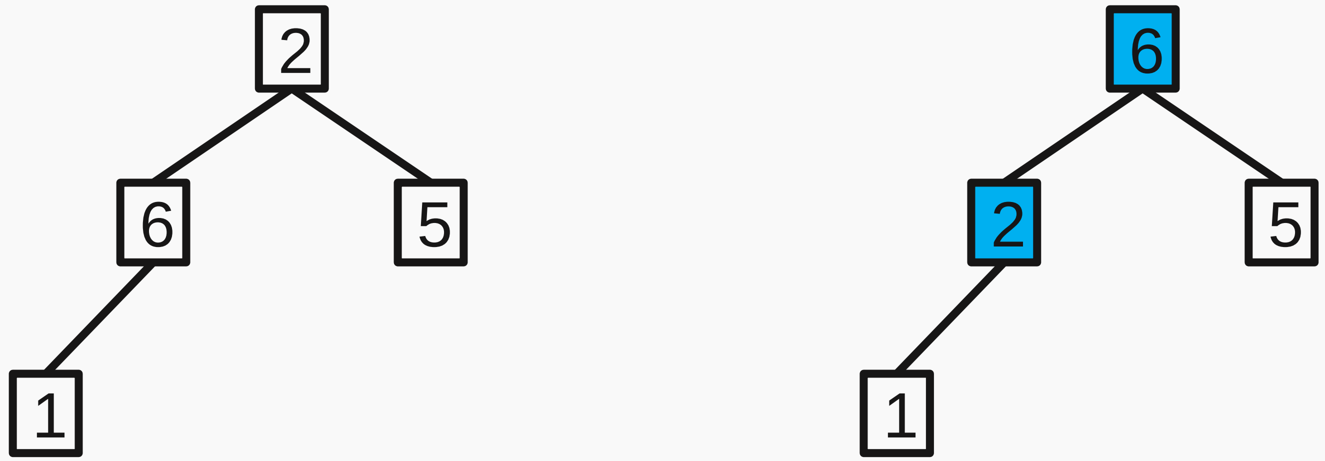


HeapSort – Populare lista elemente

5	2	6	1	9
				9

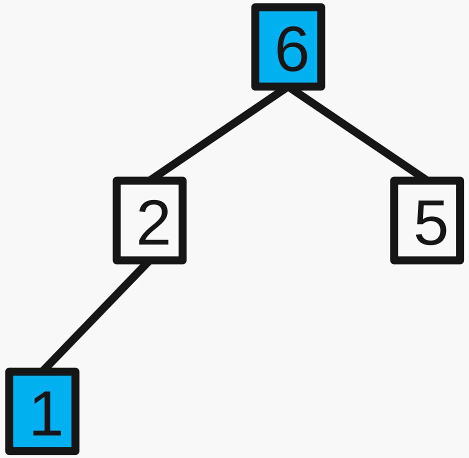


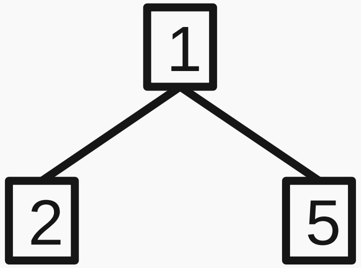
HeapSort – Condiția ~~Min~~/Max II



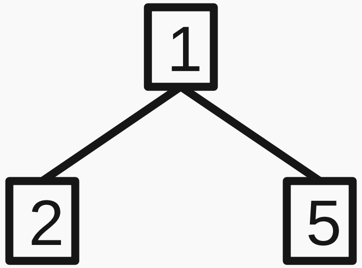
HeapSort – Populare lista elemente II

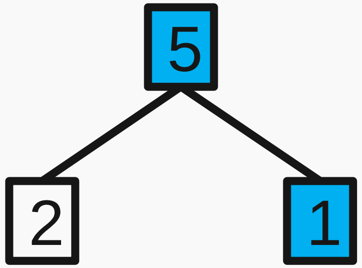
5	2	6	1	9
			6	9





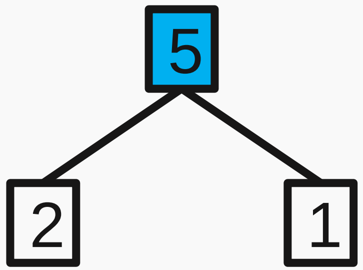
HeapSort – Condiția ~~Min~~/Max III

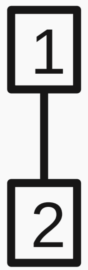




HeapSort – Populare lista elemente III

5	**2**	**6**	**1**	**9**
		5	6	9





HeapSort – Condiția ~~Min~~/Max IV + Populare listă elemente IV

5	2	6	1	9
	2	5	6	9

5	2	6	1	9
1	2	5	6	9

HeapSort

Complexitate?

HeapSort

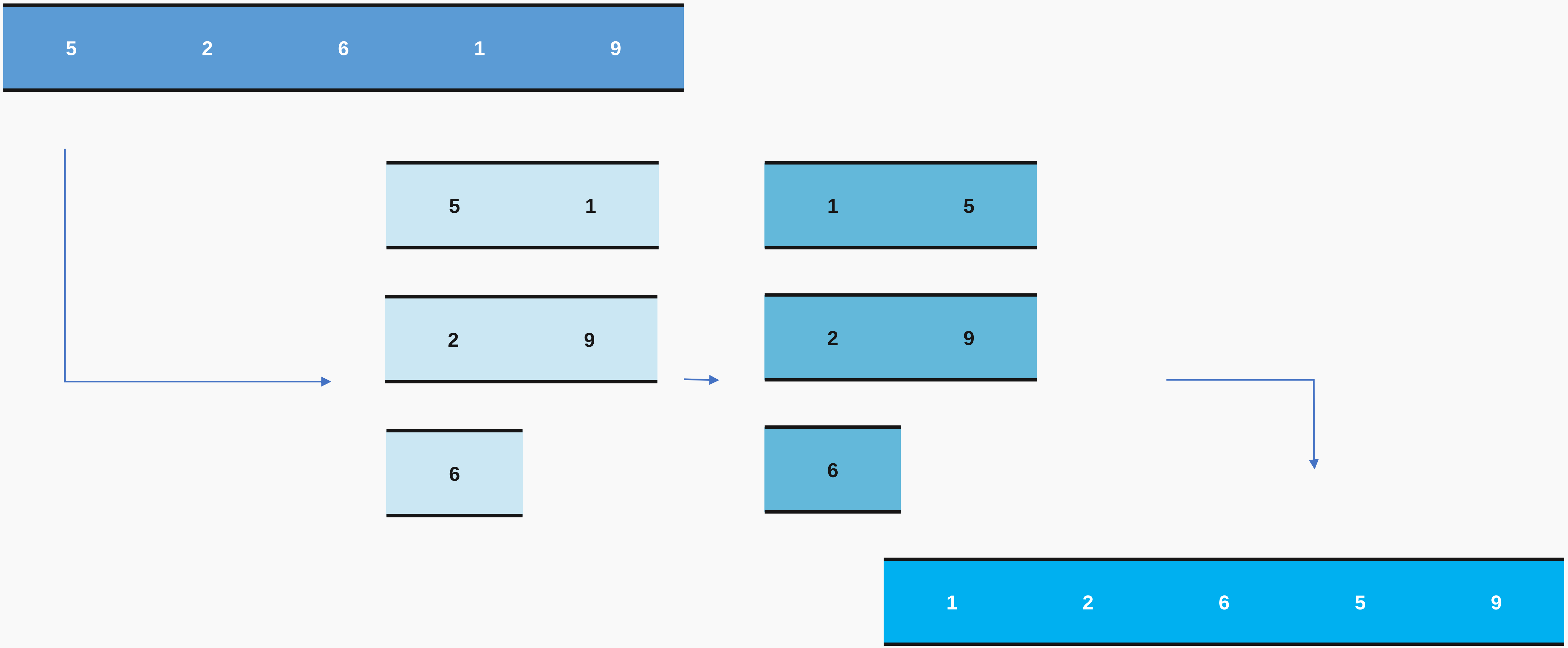
Complexitate: O(N logN)

ShellSort

- O variantă optimizată a lui InsertionSort în vederea mutării mai rapide a elementelor;
- Lista de elemente este văzută ca o mulțime care se sparge în submulțimi de K elemente(secvențe).
- Ex: Pentru o listă *L* de *N* elemente *L(1),L(2), ..L(N)* alegem *K*=4 *(K < N)* ceea ce rezultă în submulțimile:

  s (1) = *L(1), L(5), L(9), L(13), ...*
  s (2) = *L(2), L(6), L(10), L(14), ...*
  s (3) = *L(3), L(7), L(11), L(15), ...*
  s (4) = *L(4), L(8), L(12), L(16), ...*

ShellSort, k=3



ShellSort, k=1

1	2	6	5	9
1	2	6	5	9
1	2	5	6	9

ShellSort – Secvențe de pași

- S. lui Shell de tipul $\left[\frac{N}{2^k}\right]$: $\left[\frac{N}{2}\right], \left[\frac{N}{4}\right], \left[\frac{N}{8}\right], \dots 1$
- S. lui Shell de tipul $2^k\!: 2^n, 2^{n-1}, \dots 2, 1$
- S. lui Hibbard de tipul $2^k - 1$: 1, 3, 7, 15, ….
- S. lui Knuth de tipul $\frac{3^k-1}{2}$: 1, 4, 13, 40, ….
- S. lui Pratt de tipul $2^p 3^q$: $1, 2, 3, 4, 6, 8, 9, 12, \dots$

Algorithmul ShellSort

Complexitate: depinde de secvența aleasă

Secvența	**Complexitate**
Shell	$\boldsymbol{O(N^2)}$
Hibbard	$\boldsymbol{O(N\sqrt{N})}$
Knuth	$\boldsymbol{O(N\sqrt{N})}$
Pratt	$\boldsymbol{O(N\ln(N)^2)}$

Shear sort
(Row-column sort)
(Snake sort)

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

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<

Shear sort

4	6	9	9	>
7	6	5	2	<
2	3	6	9	>
5	5	4	1	<

Sortarea liniilor pare în ordine crescătoare
Sortarea liniilor impare în ordine descrescătoare

Shear sort

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

\> \> \> \>

Sortarea coloane în ordine crescătoare

Shear sort

1	2	3	4	>
5	5	4	2	<
5	6	6	9	>
9	9	7	6	<

Se repetă de $log_2 n$ ori

Shear sort

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

> > > >

Shear sort

1	2	2	3	>
5	5	4	4	<
5	6	6	6	>
9	9	9	7	<

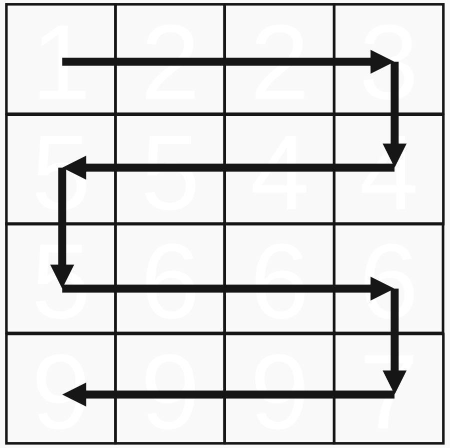
Metoda par/impar asigura compararea elementului maxim de pe linia i cu cel minim de pe linia i+1.

Shear sort

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

> > > >

Shear sort



Lista finală se citește în mod “șerpuit”.

Shear sort

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

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

Shear Sort

Complexitate?

Shear Sort

Complexitate: $O(N \log N)$

Comparație

Algoritm	Cazul cel mai Favorabil	Cazul cel mai Defavorabil	Cazul Mediu	Cost Memorie
Insertion Sort	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$	$\theta(1)$
ShellSort	$\boldsymbol{\theta(nlog(n))}$ (Knuth)	$\boldsymbol{\theta(n\sqrt{n})}$ (Knuth)	$\boldsymbol{\theta(n\sqrt[4]{n})}$ (Knuth)	$\theta(1)$
Merge Sort	$\theta(nln(n))$	$\theta(nln(n))$	$\theta(nln(n))$	$\theta(n)$
Quick Sort	$\boldsymbol{\theta(nln(n))}$	$\boldsymbol{\theta(n^2)}$	$\boldsymbol{\theta(nln(n))}$	$\boldsymbol{\theta(1)}$
Radix Sort	$\boldsymbol{\theta(Kn)}$	$\boldsymbol{\theta(Kn)}$	$\boldsymbol{\theta(Kn)}$	$\theta(n)$

https://www.toptal.com/developers/sorting-algorithms

Demonstratie corectitudine BubbleSort

- *Ipoteza inducție matematică*: algoritmul BubbleSort poate cu succes să sorteze liste de dimensiune *k.*
- *Scop inducție matematică*: verificarea că algoritmul BubbleSort funcționează și pentru liste de dimensiune *(k+1).*
- *Invarianta:*
    - pentru buclă for exterior (linia 1)
    - pentru buclă for interior (linia 2)

Pseudocod BubbleSort

```
1: for i=0, N-1
2:        for j=N, i+1
3:                if(not in order)
4:                        then swap
```

Invariant pentru buclă for exterior (linia 1)

- La pasul i = *k*, elementele listei 0,1,...,k-1 sunt sortate;
- Pentru restul elementelor *k+1,..., N* se execută cel de-al doilea *for(liniile 2-4)*, ceea ce garantează că elementul de la pozitia *k* are valoarea minimă dintre elementele *k+1,..., N*

Pseudocod BubbleSort

```
1: for i=0, N-1
2:         for j=N, i+1
3:                 if(not in order)
4:                         then swap
```

Invariant pentru buclă for interioară (linia 2)

- La pasul i = *k*, elementele listei *k+1,...,* *N* sunt sortate;
- Pentru restul elementelor 0,1,...,k se execută primul *for(liniile 1-4).* Prin verificări succesive, se stabilește noua poziție a elementului de la poziția *k* astfel încât elementele *k*, *k+1,...,N* să fie sortate.

Pseudocod BubbleSort

```
1: for i=0, N-1
2:      for j=N, i+1
3:           if(not in order)
4:                then swap
```

Analiza complexităţii QuickSort

- În cele două cazuri (pivot minim sau maxim), avem:

$$T(N) = T_{partitie} + T(N-1) = O(N) + T(N-1)$$

Prin rezolvarea formulei de recurenţă : $\boldsymbol{T(N) = O(N^2)}$

- Cazul cel mai favorabil este atunci când pivotul este ales în aşa fel încât să împartă mulţimea iniţială în două submulţimi de dimensiuni egale (val. mediană):

$$T(N) = T_{partitie} + 2T\left(\frac{N}{2}\right) = O(N) + 2T\left(\frac{N}{2}\right)$$

Prin rezolvarea formulei de recurenţă **:** $\boldsymbol{T(N) = O(N ln(N))}$

Analiza complexităţii Quick Sort

- În cazul generic al alegerii pivotului la poziția *q* avem:

$$T(N) = T_{partitie} + T(q) + T(N-q) = O(N) + T(q) + T(N-q)$$

- Pivotul poate fi cu aceeaşi probabilitate (1/N) oricare dintre elementele vectorului (presupunem distincte):

$$T(N) = \frac{1}{N}\left[\sum_{q=1,N} T(q) + T(N-q)\right] + O(N)$$

Analiza complexităţii Quick Sort

$$T(N) = \frac{1}{N}\left[\sum_{q=1,N} T(q) + T(N-q)\right] + O(N)$$

$$T(N) = \frac{1}{N}\left[\sum_{q=1,N-1} \{T(q) + T(N-q)\} + \boldsymbol{T(N-1)+T(1)}\right] + \boldsymbol{O(N)}$$

$$\frac{\boldsymbol{T(N-1)+T(1)}}{\boldsymbol{N}} \leq \frac{O(N^2)}{N} = \boldsymbol{O(N)};$$

$$\boldsymbol{O(N)} + \boldsymbol{O(N)} = O(N)$$

Analiza complexităţii Quick Sort

$$T(N) = \frac{2}{N}\sum_{q=1,N-1} T(q) + \boldsymbol{O(N)} = \frac{2}{N}\sum_{q=1,N-1} T(q) + \boldsymbol{aN + b}$$

$$\boldsymbol{NT(N) = 2 \sum_{q=1,N-1} T(q) + aN^2 + bN}$$

$$(N+1)T(N+1) = 2 \sum_{q=1,N} T(q) + a(N+1)^2 + b(N+1)$$

-

$$(N+1)T(N+1) - NT(N) = 2\sum_{q=1,N} T(q) + a(N+1)^2 + b(N+1) - (2\sum_{q=1,N-1} \mathrm{T(q)} + \mathrm{aN}^2 + bN)$$

Analiza complexităţii Quick Sort

$$(N+1)T(N+1) - NT(N) = 2\sum_{q=1,N-1} T(q) + 2T(N) + aN^2 + 2aN + a + bN + b - 2\sum_{q=1,N-1} \mathrm{T(q)} - \mathrm{aN}^2 - bN$$

$$(N+1)T(N+1) - NT(N) = 2T(N) + a(2N+1) + b$$

$$(N+1)T(N+1) = (N+2)T(N) + 2a(N+1) + b - a$$

$$\frac{\boldsymbol{T(N+1)}}{\boldsymbol{N+2}} = \frac{\boldsymbol{T(N)}}{\boldsymbol{N+1}} + \frac{\boldsymbol{2a}}{\boldsymbol{(N+2)}} + \frac{\boldsymbol{B}}{\boldsymbol{(N+1)(N+2)}}$$

Analiza complexitatii Quick Sort

$$\frac{T(N+1)}{N+2} = \frac{T(N)}{N+1} + \frac{2a}{N+2} + \frac{B}{(N+1)(N+2)}$$

$$\boldsymbol{\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2a}{N+1} + \frac{B}{N(N+1)}}$$

$$\frac{T(N-1)}{N} = \frac{T(N-2)}{N-1} + \frac{2a}{N} + \frac{B}{(N-1)N}$$

………………………………………………

$$\frac{T(1)}{2} = \frac{T(0)}{1} + \frac{2a}{2} + \frac{B}{1*2}$$

Analiza complexitatii Quick Sort

$$\frac{T(N+1)}{N+2} = \cancel{\frac{T(N)}{N+1}} + \frac{2a}{N+2} + \frac{B}{(N+1)(N+2)}$$

$$\boldsymbol{\cancel{\frac{T(N)}{N+1}} = \cancel{\frac{T(N-1)}{N}} + \frac{2a}{N+1} + \frac{B}{N(N+1)}}$$

$$\cancel{\frac{T(N-1)}{N}} = \cancel{\frac{T(N-2)}{N-1}} + \frac{2a}{N} + \frac{B}{(N-1)N}$$

....

$$\cancel{\frac{T(1)}{2}} = \frac{T(0)}{1} + \frac{2a}{2} + \frac{B}{1*2}$$

Analiza complexitatii Quick Sort

$$\frac{T(N)}{N+1} = \frac{T(0)}{1} + B\sum_{1\le k\le N}\frac{1}{k(k+1)} + 2a\sum_{1\le k\le N}\frac{1}{k+1}$$

$$\frac{T(N)}{N+1} = \frac{\color{red}{T(0)}}{1} + B\sum_{1\le k\le N}\left[\frac{1}{k} - \frac{1}{k+1}\right] + 2a\sum_{1\le k\le N}\frac{1}{k+1}$$

$$\frac{T(N)}{N+1} = B\left[1 - \frac{1}{N+1}\right] + 2a\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N+1}\right)$$

$$\frac{T(N)}{N+1} = B\frac{N}{N+1} + 2a\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\mathrm{N}+1}\right)$$

Analiza complexitatii Quick Sort

$$\frac{T(N)}{N+1} = B\frac{N}{N+1} + 2a\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N+1}\right)$$

$$\mathrm{T(N)} = \mathrm{BN{+}2a(N{+}1)}\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\mathrm{N+1}} - 1\right)$$

$$\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\mathrm{N+1}}\right) > \int_{1}^{N+1}\frac{1}{x}dx = ln(N+1)$$

$$T(N) \approx BN + 2a(N+1)(ln(n+1) - 1), \textit{ unde a, b - constante}$$

$$\boldsymbol{T(N) = O[N ln(N)]}$$