

Heap binar

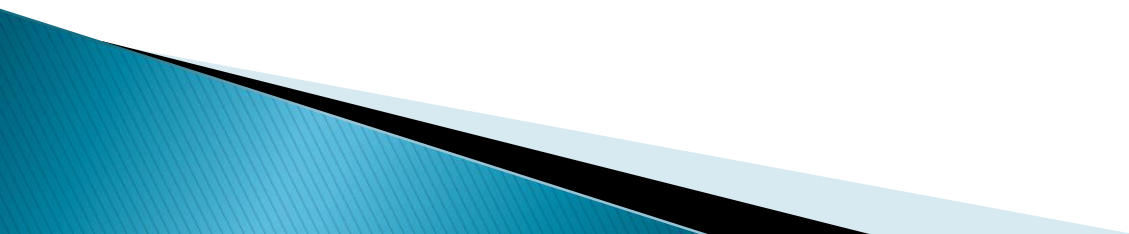
Heap

Un **heap binar** este un arbore binar complet in care fiecare nod respectă **proprietatea de heap**:

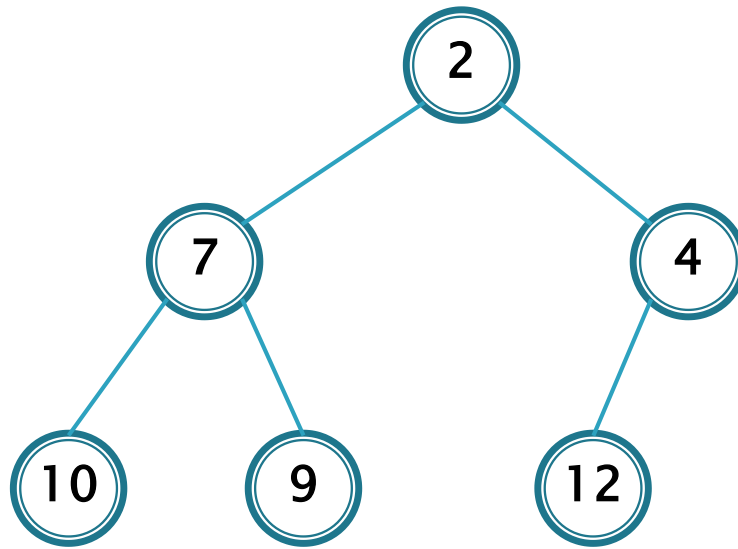
- ▶ **min-heap**: Valoarea din fiecare nod este mai mică sau egală decât valorile memorate în nodurile fii ai acestuia.
- ▶ **max-heap**: Valoarea din fiecare nod este mai mare sau egală decât valorile memorate in nodurile fii ai acestuia

Un heap binar se memorează ca **vector**.

Vom discuta despre min-heap (îl vom numi doar heap).



Heap



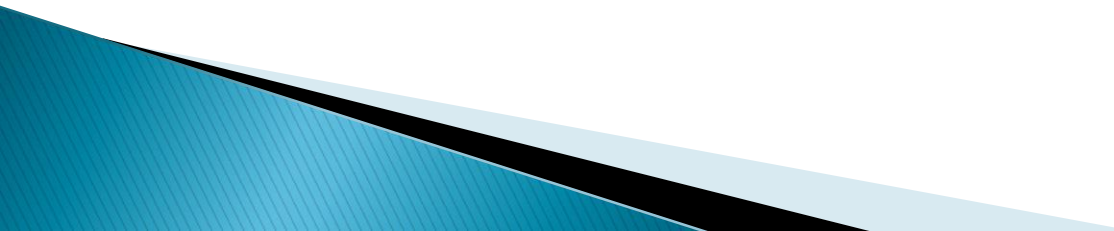
Înălțime $O(\log n)$

h:

2	7	4	10	9	12
---	---	---	----	---	----

Heap

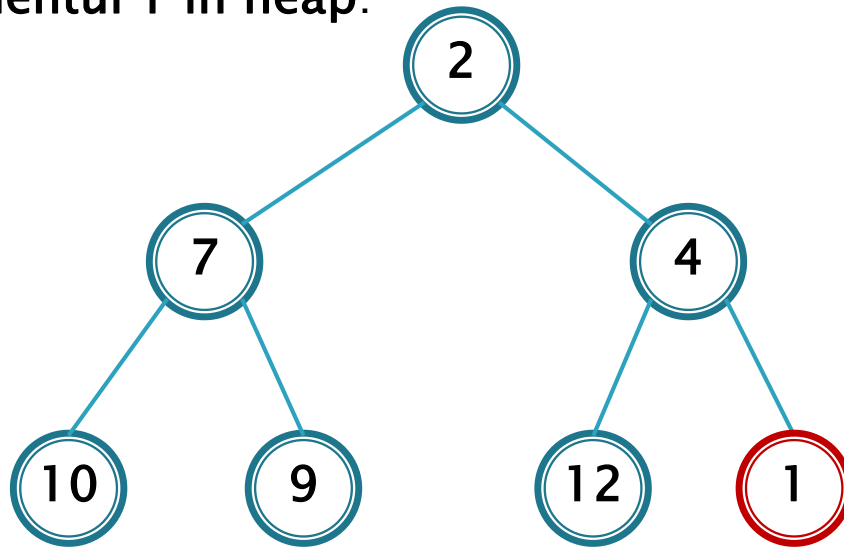
Operații:

- **Urcarea** unei element pentru repararea structurii după modificarea elementului – $O(\log n)$
 - **Coborârea** unei element pentru repararea structurii după modificarea elementului – $O(\log n)$
 - Determinarea minimului – $O(1)$
 - Extragere minimului – $O(\log n)$
 - Inserarea unui element – $O(\log n)$
 - Transformarea unui vector dat în heap – $O(n)$
 - Sortare HeapSort– $O(n \log n)$
- 

Inserare

Heap – Insert

Adăugăm elementul 1 în heap:



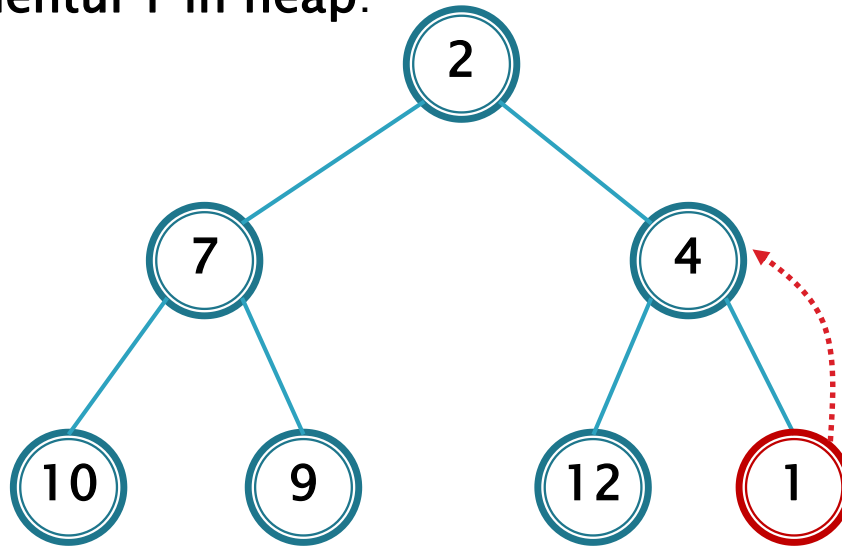
Pasul 1: îl adăugăm la final în h

h:

2	7	4	10	9	12	1
---	---	---	----	---	----	---

Heap – Insert

Adăugăm elementul 1 în heap:



Pasul 1: îl adăugăm la final în h

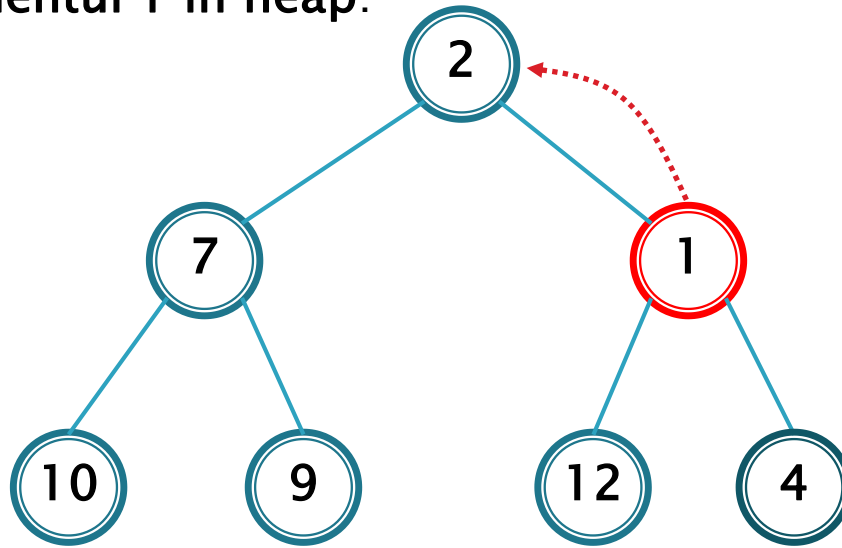
Pasul 2: îl interschimbăm cu tata
cât timp tata este mai mare (nu
se verifica proprietatea de heap)

h:

2	7	4	10	9	12	1
---	---	---	----	---	----	---

Heap – Insert

Adăugăm elementul 1 în heap:



Pasul 1: îl adăugăm la final în h

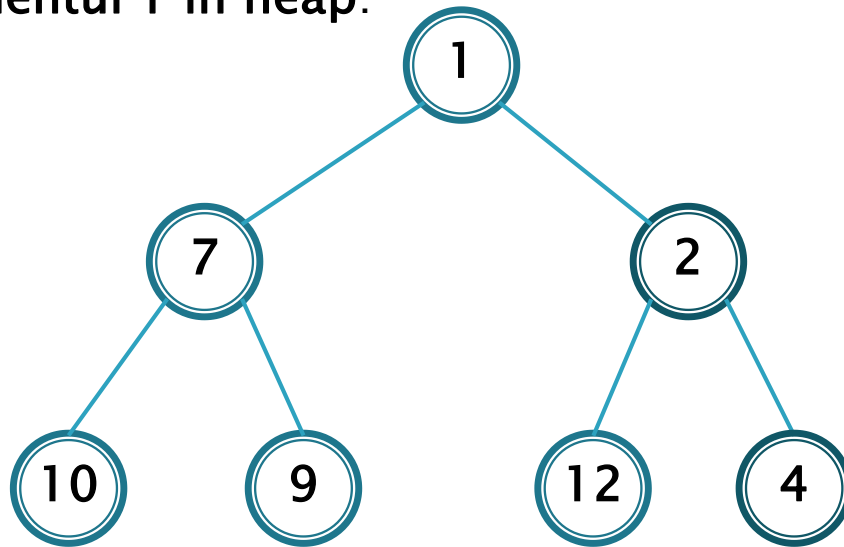
Pasul 2: îl interschimbăm cu tata
cât timp tata este mai mare (nu
se verifica proprietatea de heap)

h:

2	7	1	10	9	12	4
---	---	---	----	---	----	---

Heap – Insert

Adăugăm elementul 1 în heap:



$O(\log n)$

h:

1	7	2	10	9	12	4
---	---	---	----	---	----	---

Heap – Implementare Urcare și Inserare

- ▶ Pe ce poziție sunt fiii nodului de pe poziția i ?

Heap – Implementare Urcare și Inserare

- ▶ Pe ce poziție sunt fiii nodului de pe poziția i ?
 - **Depinde cum numerotăm pozițiile, de la 0 sau de la 1**
(=> diferențe între pseudocod, unde numerotarea este de la 1 și cod)

Heap – Implementare Urcare și Inserare

- ▶ Pe ce poziție sunt fiii nodului de pe poziția i ?

- **Depinde cum numerotăm pozițiile, de la 0 sau de la 1**

(=> diferențe între pseudocod, unde numerotarea este de la 1 și cod)

PARENT(i)

1 return $\lfloor i/2 \rfloor$

LEFT(i)

1 return $2i$

RIGHT(i)

1 return $2i + 1$

```
int parent(int i) {  
    return (i-1)/2; }
```

```
int left(int i) {  
    return (2*i + 1); }
```

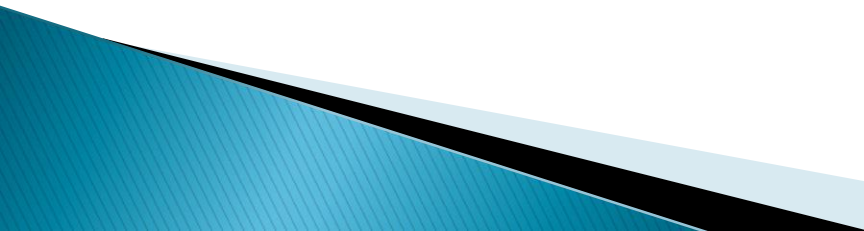
```
int right(int i) {  
    return (2*i + 2); }
```

Heap – Urcare cheie modificată

HEAP-UP (A, i)

```
1 while i > 1 and A[PARENT(i)] > A[i]
2   exchange A[i] with A[PARENT(i)]      O(log n)
3   i = PARENT(i)
```

```
void HeapUp(int h[MAXDIM], int dim_heap, int i ){
    while (i != 0 && h[parent(i)] > h[i])
    {
        swap( h[i], h[parent(i)] );
        i = parent(i);
    }
}
```



Heap – Inserare

MIN-HEAP-INSERT(A, key)

1 $A.heap-size = A.heap-size + 1$

2 $A[A.heap-size] = key$

3 HEAP-UP($A, A.heap-size$)

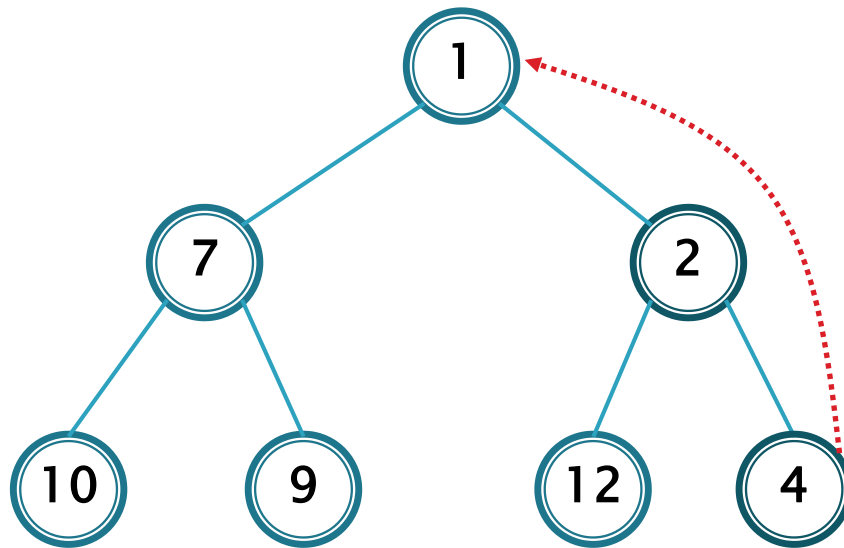
```
void MinHeapInsert(int h[MAXDIM], int &dim_heap, int x){
    //if (dim_heap >= MAXDIM) return;
    int i = dim_heap;
    h[i] = x;
    dim_heap++;
    HeapUp(h, dim_heap, i);
}
```

Determinare și extragere minim

Determinare minim

- ▶ **Minimul este primul element**

Extragere minim

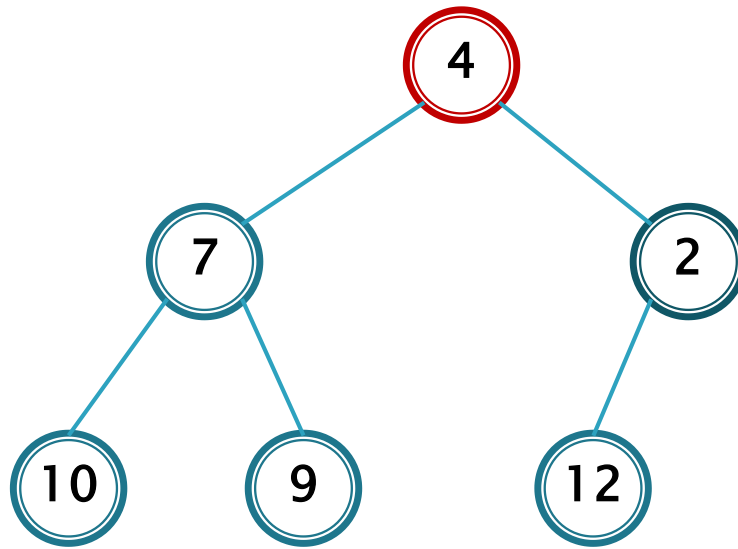


Pasul 1: mutăm ultimul element
în rădăcină (scade dimensiune h)

h:

1	7	2	10	9	12	4
---	---	---	----	---	----	---

Extragere minim

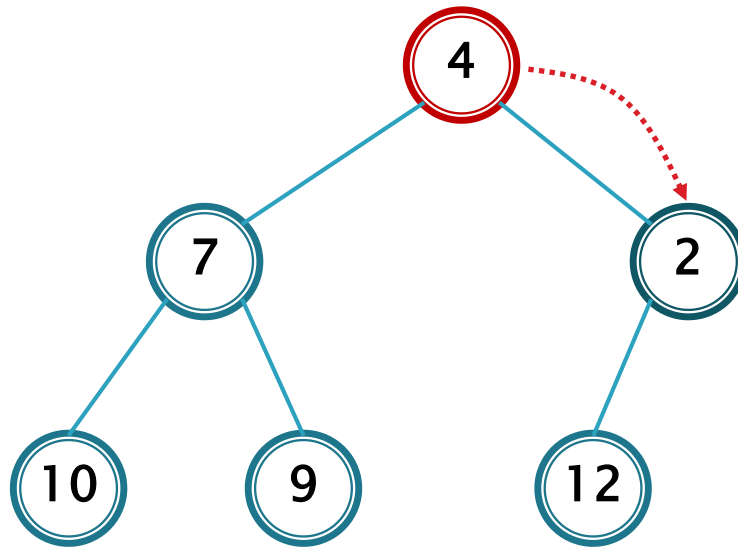


Pasul 1: mutăm ultimul element
în rădăcină (scade dimensiune h)

h:

4	7	2	10	9	12
---	---	---	----	---	----

Extragere minim



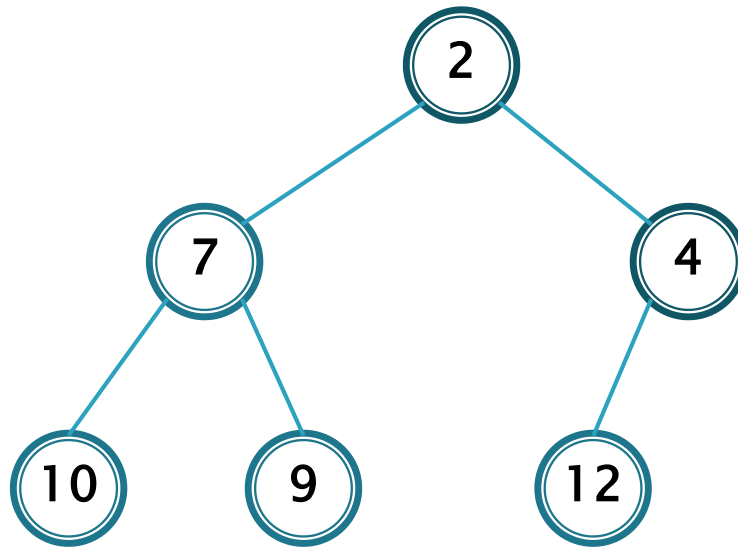
Pasul 1: mutăm ultimul element în rădăcină (scade dimensiune h)

Pasul 2: îl coborâm repetat în **cel mai mic dintre fii** cât timp are un fiu mai mic

h:

4	7	2	10	9	12
---	---	---	----	---	----

Extragere minim



Pasul 1: mutăm ultimul element în rădăcină (scade dimensiune h)

Pasul 2: îl coborâm repetat în **cel mai mic dintre fii** cât timp are un fiu mai mic

h:

2	7	4	10	9	12
---	---	---	----	---	----

Heap – Coborâre cheie modificată + Extragere minim

MIN-HEAPIFY(A, i)

```
1  $l = \text{LEFT}(i)$ 
2  $r = \text{RIGHT}(i)$ 
3 if  $l \leq A.\text{heap-size}$  and  $A[l] < A[i]$ 
4      $\text{smallest} = l$ 
5 else  $\text{smallest} = i$ 
6 if  $r \leq A.\text{heap-size}$  and  $A[r] < A[\text{smallest}]$ 
7      $\text{smallest} = r$ 
8 if  $\text{smallest} \neq i$ 
9     exchange  $A[i]$  with  $A[\text{smallest}]$ 
10  MIN-HEAPIFY( $A, \text{smallest}$ )
```

HEAP-EXTRACT-MIN(A)

```
1 if  $A.\text{heap-size} < 1$ 
2     error “heap underflow”
3  $\text{min} = A[1]$ 
4  $A[1] = A[A.\text{heap-size}]$ 
5  $A.\text{heap-size} = A.\text{heap-size} - 1$ 
6 MIN-HEAPIFY( $A, 1$ )
7 return  $\text{min}$ 
```

$O(\log n)$

Heap – Coborâre cheie modificată + Extragere minim

MIN-HEAPIFY(A, i)

```
1  $l = \text{LEFT}(i)$ 
2  $r = \text{RIGHT}(i)$ 
3 if  $l \leq A.\text{heap-size}$  and  $A[l] < A[i]$ 
4      $\text{smallest} = l$ 
5 else  $\text{smallest} = i$ 
6 if  $r \leq A.\text{heap-size}$  and  $A[r] < A[\text{smallest}]$ 
7      $\text{smallest} = r$ 
8 if  $\text{smallest} \neq i$ 
9     exchange  $A[i]$  with  $A[\text{smallest}]$ 
10  MIN-HEAPIFY( $A, \text{smallest}$ )
```

$O(\log n)$

Se poate și nerecursiv

```
void MinHeapify (int h[MAXDIM], int dim_h, int i) {
    //fiul minim
    while (left(i) < dim_h) {
        int smallest = left(i);
        int r = right(i);

        if (r < dim_h && h[r] < h[smallest])
            smallest = r;
        if (h[i] > h[smallest])
            swap(h[i], h[smallest]);
        else
            break;
        i = smallest;
    }
}
```

Transformare vector în heap

Construire heap

BUILD-MIN-HEAP(A)

1 $A.heap\text{-}size = A.length$

2 **for** $i = \lfloor A.length / 2 \rfloor$ **downto** 1

3 MIN-HEAPIFY(A, i)

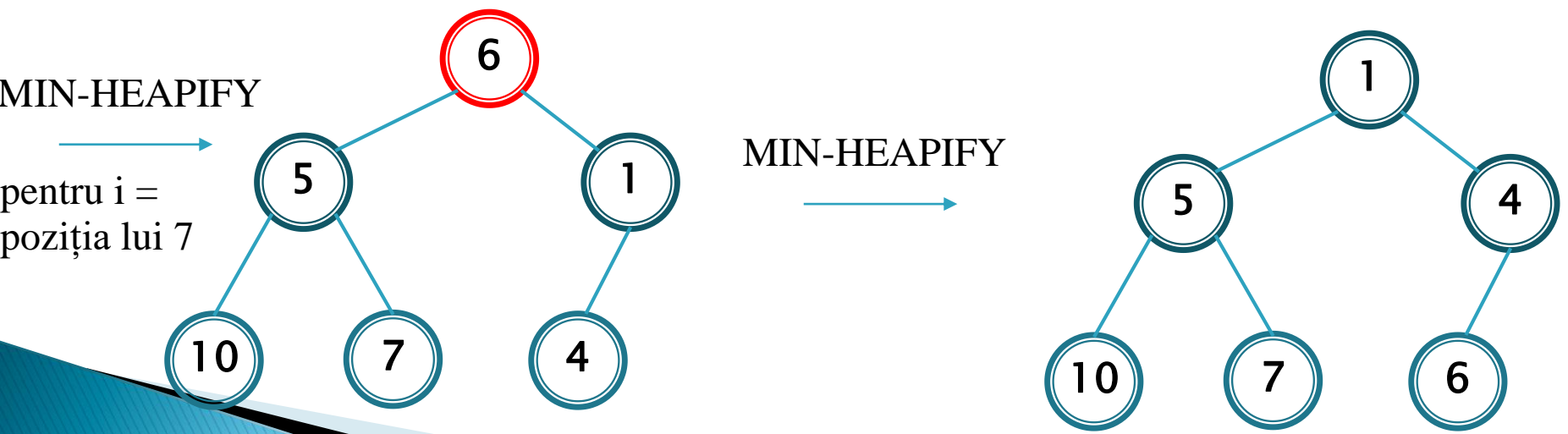
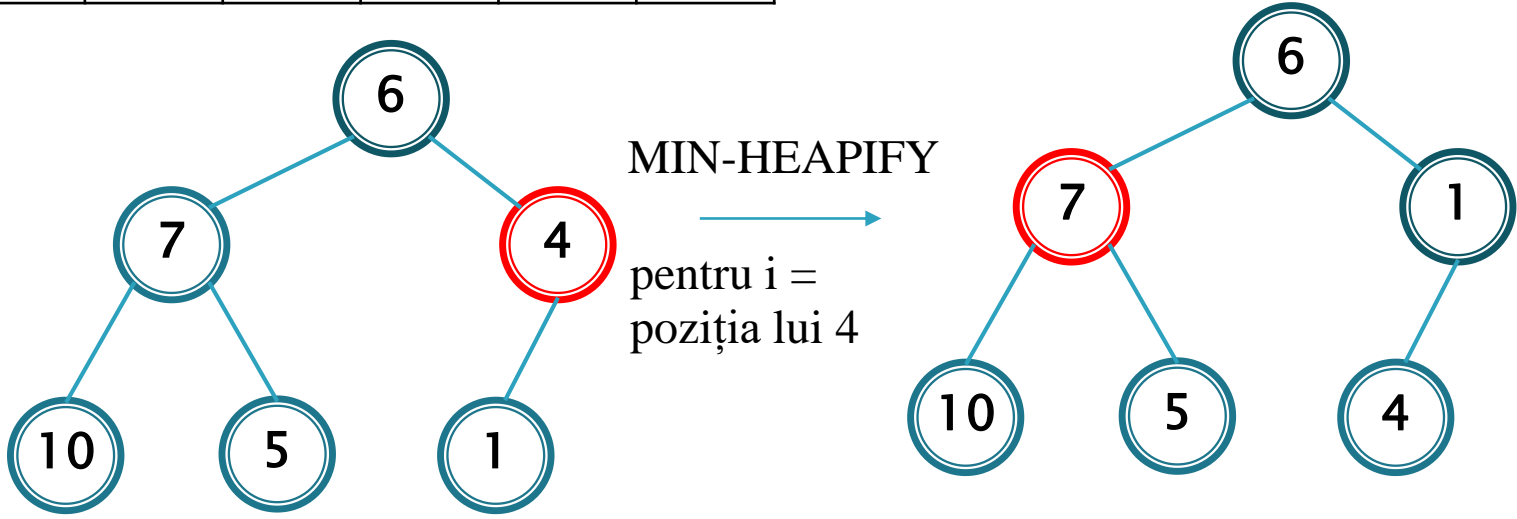
```
for (int i=dim_heap/2-1; i>=0; i--) {  
    MinHeapify(h, dim_heap, i);  
}
```

$O(n)$!!!!

Construire heap

h:

6	7	4	10	5	1
---	---	---	----	---	---



Sortare HeapSort

Sortare HeapSort

HEAPSORT(A)

1 BUILD-MAX-HEAP(A)

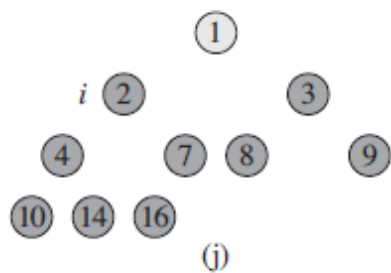
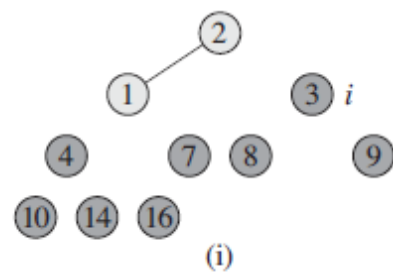
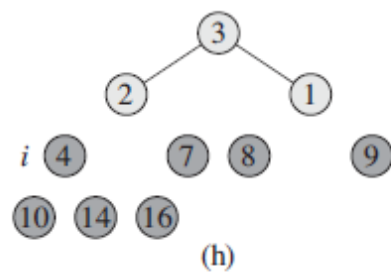
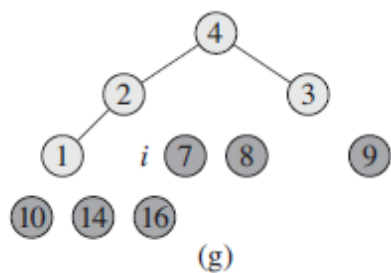
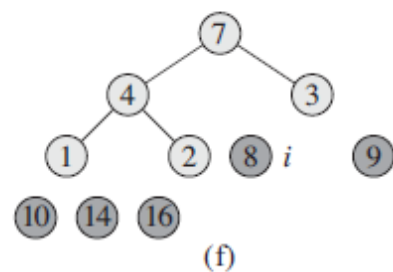
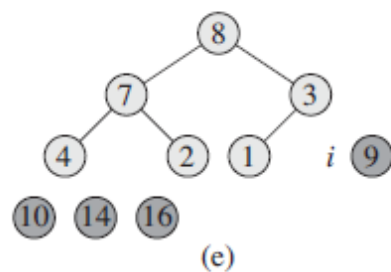
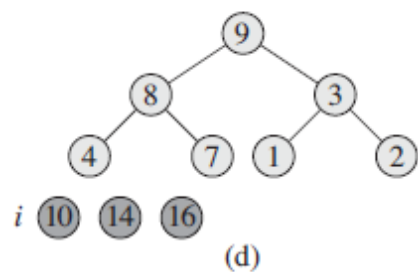
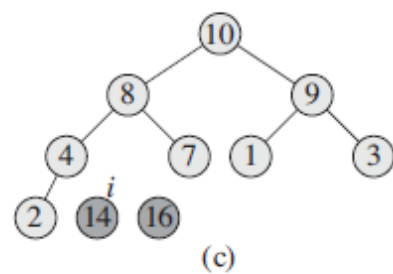
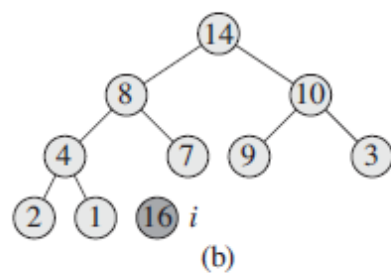
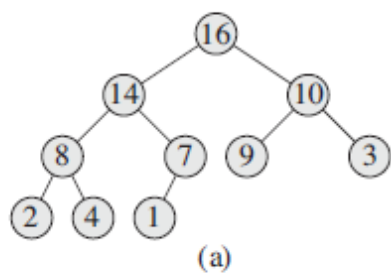
2 **for** $i = A.length$ **downto** 2

3 exchange $A[1]$ with $A[i]$ → Maximul ajunge pe ultima poziție

4 $A.heap-size = A.heap-size - 1$ → Scade dimensiunea heap
5 MAX-HEAPIFY($A, 1$) (ce este corect poziționat
se ignoră din heap)

Se coboară $A[1]$ pentru reparare heap

$O(n \log n)$



A

1	2	3	4	7	8	9	10	14	16
---	---	---	---	---	---	---	----	----	----

(k)

Cele mai mari k elemente

Cele mai mari k elemente dintr-un șir

- ▶ <https://www.pbinfo.ro/probleme/3011/lastk>
- ▶ Dacă putem memora toate elementele – Divide et Impera (al p-lea minim – cu ideea de pivotare de la Quicksort)
- ▶ Dacă nu putem memora elementele (sunt multe)?

Cele mai mari k elemente dintr-un șir

- ▶ <https://www.pbinfo.ro/probleme/3011/lastk>
- ▶ Dacă putem memora toate elementele – Divide et Impera (al p-lea minim – cu ideea de pivotare de la Quicksort)
- ▶ Dacă nu putem memora elementele (sunt multe)?
 - Parcurgem element cu element
 - Memorăm doar cele mai mari k elemente până la pasul curent

Cele mai mari k elemente dintr-un șir

- ▶ <https://www.pbinfo.ro/probleme/3011/lastk>
- ▶ Dacă putem memora toate elementele – Divide et Impera (al p-lea minim – cu ideea de pivotare de la Quicksort)
- ▶ Dacă nu putem memora elementele (sunt multe)?
 - Parcurgem element cu element
 - Memorăm doar cele mai mari k elemente până la pasul curent
- ▶ Cum actualizăm cele mai mari k elemente când parcurgem un element nou?

Cele mai mari k elemente dintr-un șir

- ▶ <https://www.pbinfo.ro/probleme/3011/lastk>
- ▶ Dacă putem memora toate elementele – Divide et Impera (al p-lea minim – cu ideea de pivotare de la Quicksort)
- ▶ Dacă nu putem memora elementele (sunt multe)?
 - Parcurgem element cu element
 - Memorăm doar cele mai mari k elemente până la pasul curent
- ▶ Cum actualizăm cele mai mari k elemente când parcurgem un element nou?
 - Înlocuim cel mai mic dintre cele k elemente cu elementul curent, dacă elementul curent este mai mare decât el

Cele mai mari k elemente dintr-un șir

- ▶ Idee algoritm:

A = primele k elemente din șir

Pentru fiecare x dintre elementele ramase:

$y = \min(A)$

dacă $x > y$ atunci:

elimina_min(A)

adauga(A, x)

- ▶ Cum memorăm A?

Cele mai mari k elemente dintr-un șir

- ▶ Idee algoritm:

A = primele k elemente din șir

Pentru fiecare x dintre elementele ramase:

$y = \min(A)$

dacă $x > y$ atunci:

elimina_min(A)

adauga(A, x)

- ▶ Cum memorăm A?

- Min Heap \rightarrow complexitate $O(n \log k)$

Interclasarea a k şiruri ordonate

Interclasarea a k șiruri ordonate

- ▶ Două câte două – ordine de interclasare Greedy
- ▶ Mai eficient?

Interclasarea a k șiruri ordonate

- ▶ Două câte două – ordine de interclasare Greedy
- ▶ Mai eficient?
 - Interclasăm “simultan”
 - Păstrăm un min-heap cu elementele curente din fiecare vector
 - Minimul din heap se adaugă în rezultat, iar în heap se adaugă următorul element din vectorul căruia aparținea minimul

Interclasarea a k șiruri ordonate

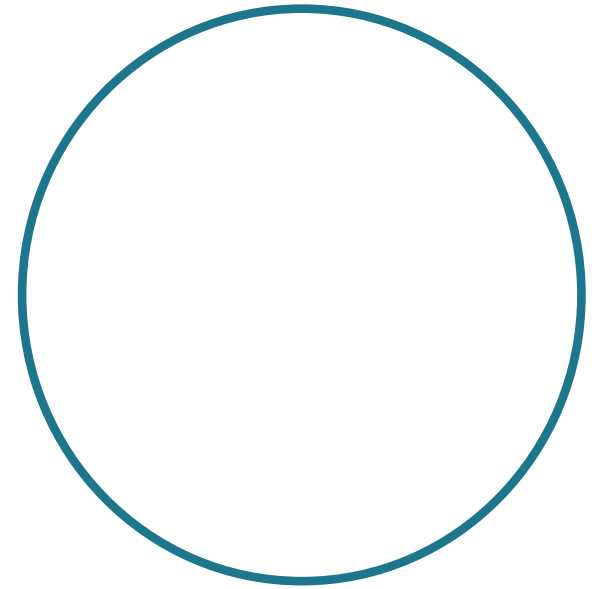
- ▶ Două câte două – ordine de interclasare Greedy
- ▶ Mai eficient?
 - Interclasăm “simultan”
 - Păstrăm un min-heap cu elementele curente din fiecare vector
 - Minimul din heap se adaugă în rezultat, iar în heap se adaugă următorul element din vectorul căruia aparținea minimul

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Min-heap

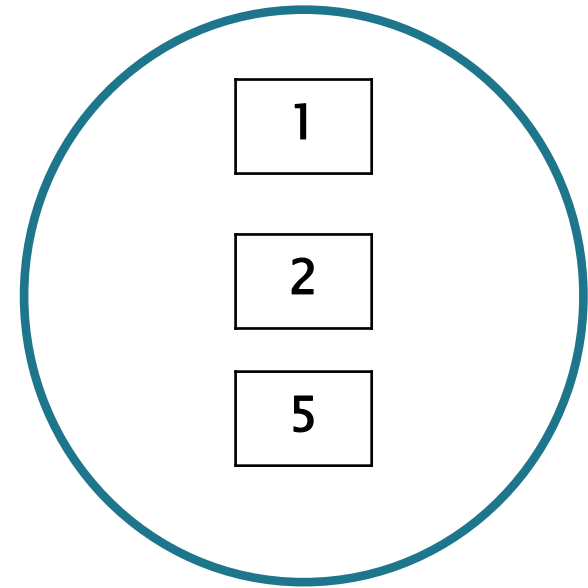
--	--	--	--	--	--	--	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Min-heap

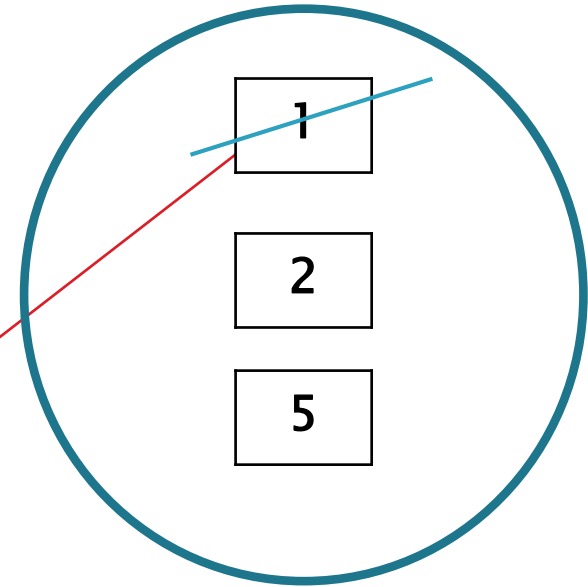
--	--	--	--	--	--	--	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----

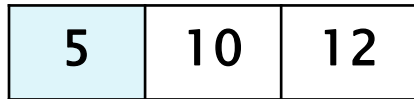
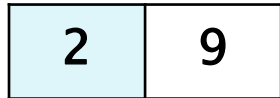


Min-heap

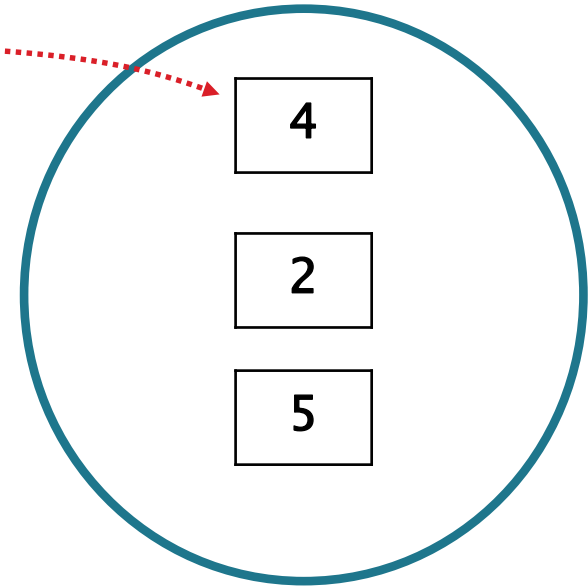
Extragem minimul
şi îl adăugăm în rezultat

1							
---	--	--	--	--	--	--	--

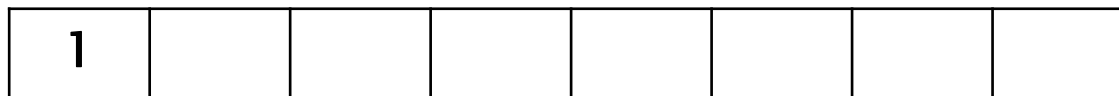
Interclasarea a k şiruri ordonate



Avansam în vectorul
de unde am extras
elementul



Min-heap

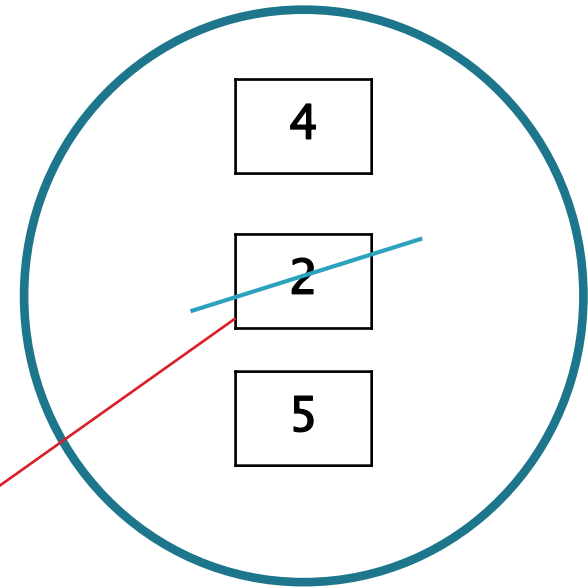


Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

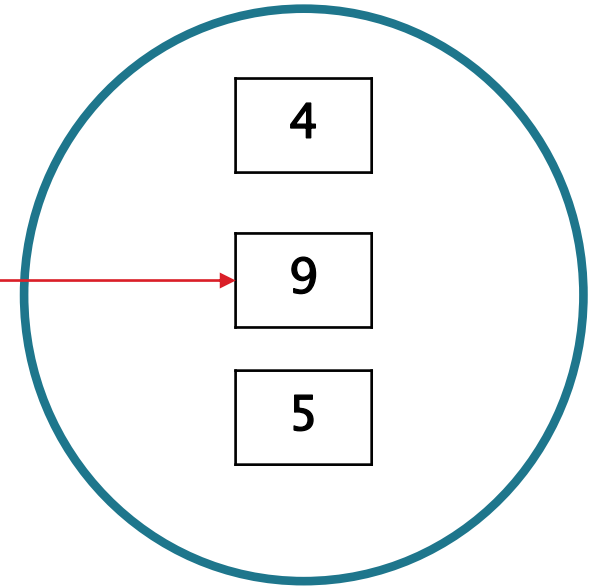
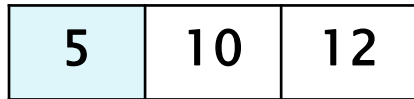
5	10	12
---	----	----



Min-heap

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

Interclasarea a k şiruri ordonate



Min-heap

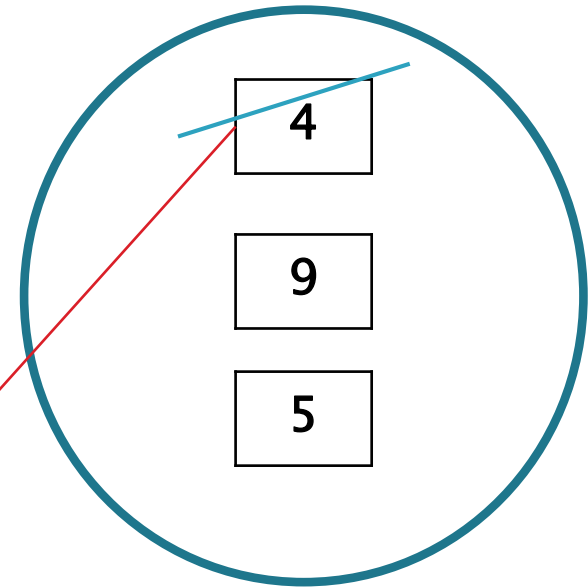


Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

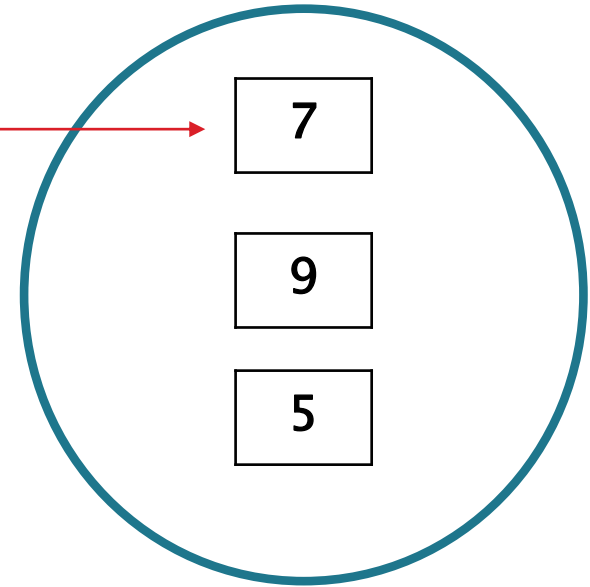
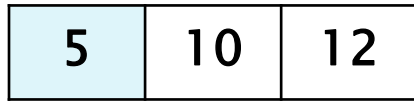
5	10	12
---	----	----



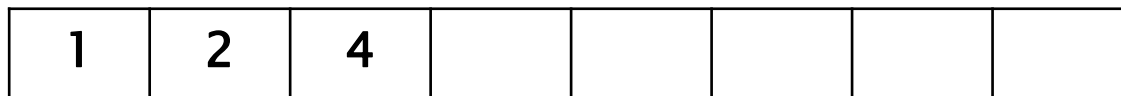
Min-heap

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

Interclasarea a k şiruri ordonate



Min-heap

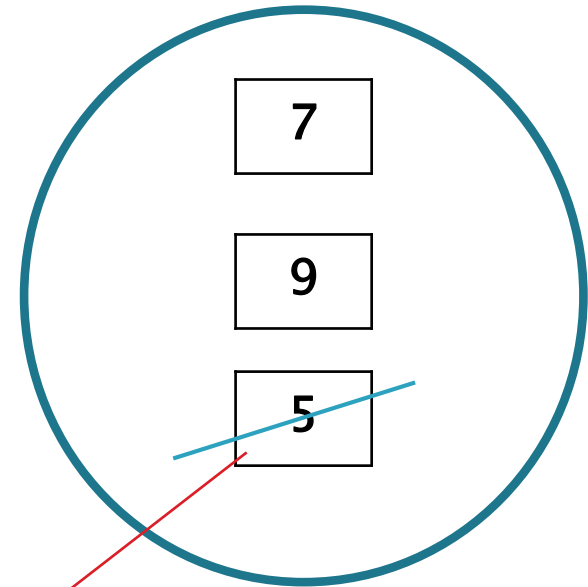


Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

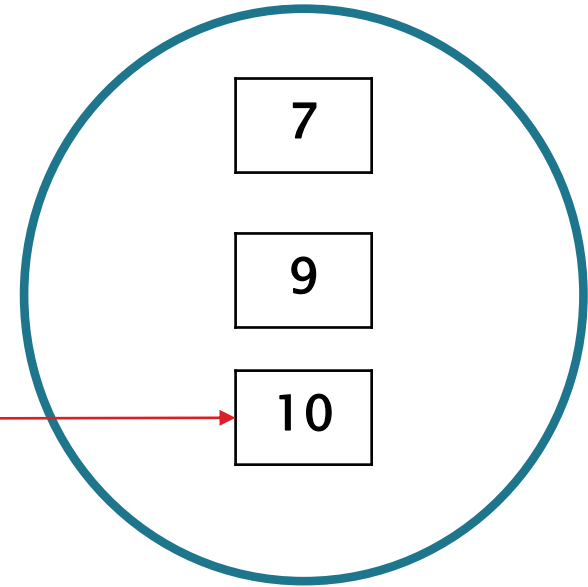
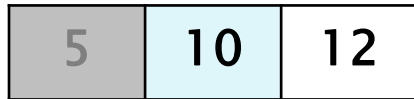
5	10	12
---	----	----



Min-heap

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

Interclasarea a k şiruri ordonate



Min-heap

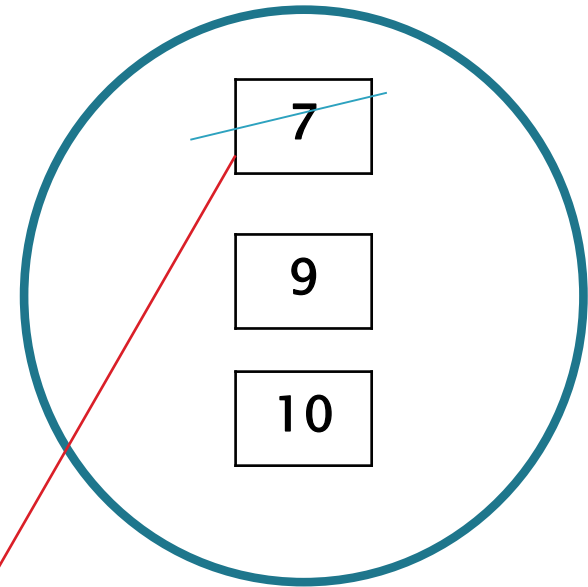


Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Min-heap

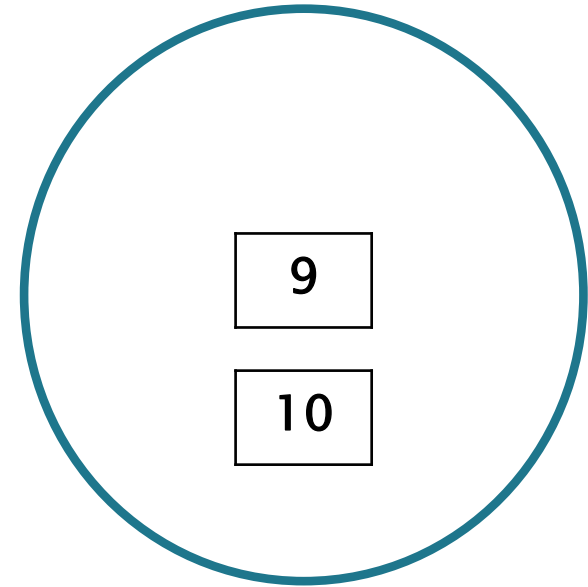
1	2	4	5	7			
---	---	---	---	---	--	--	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



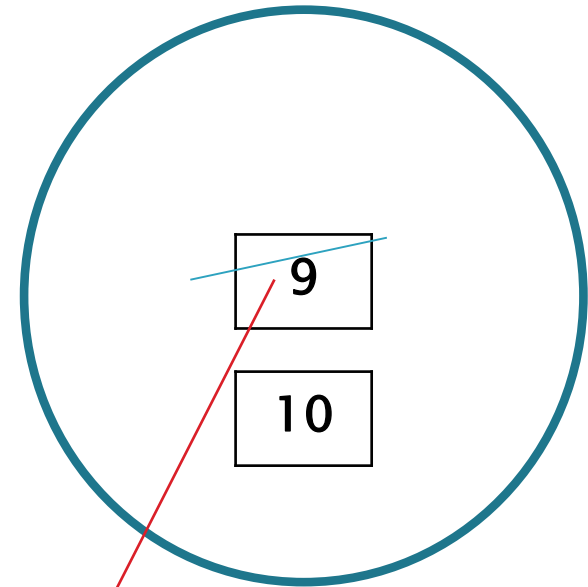
1	2	4	5	7			
---	---	---	---	---	--	--	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Min-heap

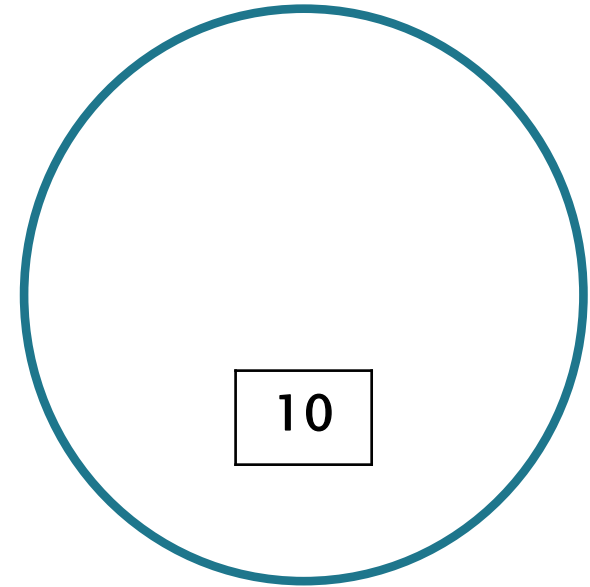
1	2	4	5	7	9		
---	---	---	---	---	---	--	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Min-heap

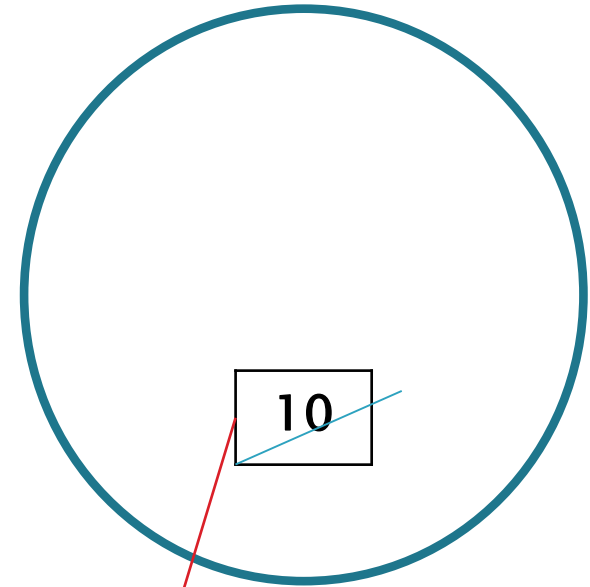
1	2	4	5	7	9		
---	---	---	---	---	---	--	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Min-heap

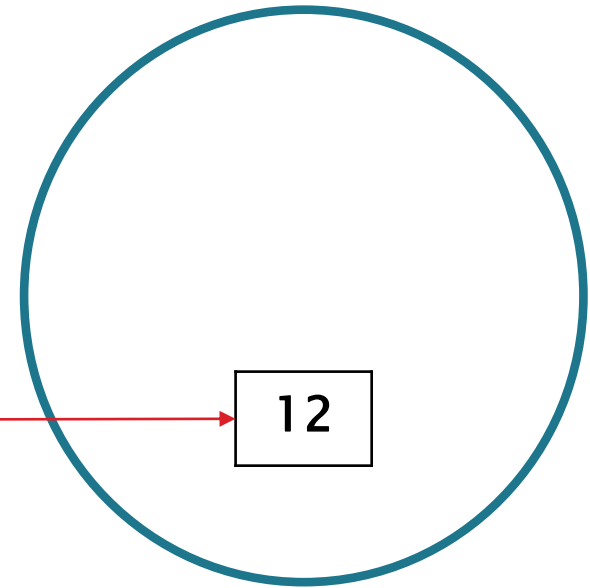
1	2	4	5	7	9	10	
---	---	---	---	---	---	----	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Min-heap

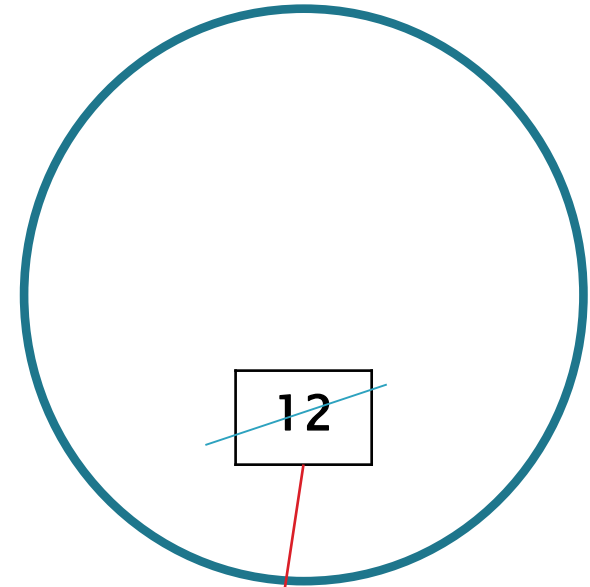
1	2	4	5	7	9	10	
---	---	---	---	---	---	----	--

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



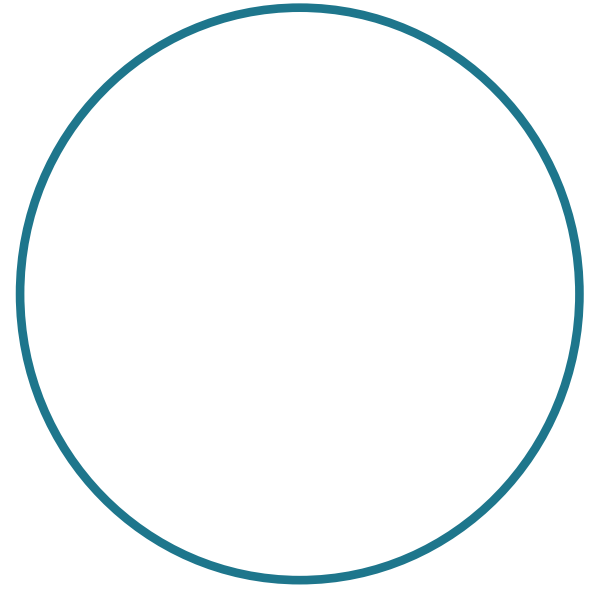
1	2	4	5	7	9	10	12
---	---	---	---	---	---	----	----

Interclasarea a k şiruri ordonate

1	4	7
---	---	---

2	9
---	---

5	10	12
---	----	----



Complexitate?

1	2	4	5	7	9	10	12
---	---	---	---	---	---	----	----