

# Tema 4. Cues amb prioritats

Estructures de Dades i Algorismes

FIB

Transparències d' **Antoni Lozano**  
(amb edicions menors d'altres professors)

Q1 2021 – 22

## 1 Preliminars matemàtics

## 2 Cues amb prioritats

- Introducció
- Heaps
- Operacions bàsiques
- Implementació recursiva
- Implementació iterativa

## 3 Heapsort

- Algorisme bàsic
- Millores de l'algorisme bàsic

## 4 Altres aplicacions

- El problema de selecció

# Tema 4. Cues amb prioritats

## 1 Preliminars matemàtics

## 2 Cues amb prioritats

- Introducció
- Heaps
- Operacions bàsiques
- Implementació recursiva
- Implementació iterativa

## 3 Heapsort

- Algorisme bàsic
- Millores de l'algorisme bàsic

## 4 Altres aplicacions

- El problema de selecció

# Arbres binaris perfectes

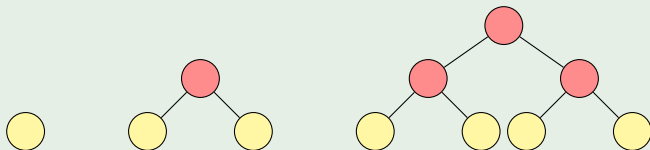
## Definició

El **nivell** d'un node en un arbre és la distància de l'arrel al node.

## Definició

Un **arbre binari** és **perfecte** si totes les fulles estan al mateix nivell.

## Exemples



## Definició

L' **alçada** d'un arbre és el nivell màxim dels nodes.

# Arbres binaris perfectes

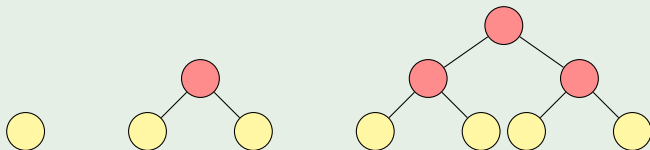
## Definició

El **nivell** d'un node en un arbre és la distància de l'arrel al node.

## Definició

Un **arbre binari** és **perfecte** si totes les fulles estan al mateix nivell.

## Exemples



## Definició

L' **alçada** d'un arbre és el nivell màxim dels nodes.

# Arbres binaris perfectes

## Proposició

Un arbre binari perfecte d'alçada  $h$  té  $2^{h+1} - 1$  nodes.

## Demostració

Farem inducció en l'alçada. Sigui  $T$  un arbre binari perfecte d'alçada  $h$ .

- **Base d'inducció:**  $h = 0$ .

L'arbre ha de tenir un sol node, però  $1 = 2^{0+1} - 1$ .

- **Pas d'inducció:**  $h > 0$ .

Els subarbres esquerre i dret tenen alçada  $h - 1$  i, per hipòtesi d'inducció, cadascun té  $2^h - 1$  nodes. El nombre de nodes de  $T$  és la suma d'aquests nodes més un (de l'arrel):

$$\text{nodes de } T = 2(2^h - 1) + 1 = 2^{h+1} - 2 + 1 = 2^{h+1} - 1.$$

# Arbres binaris perfectes

## Proposició

Un arbre binari perfecte d'alçada  $h$  té  $2^{h+1} - 1$  nodes.

## Demostració

Farem inducció en l'alçada. Sigui  $T$  un arbre binari perfecte d'alçada  $h$ .

- **Base d'inducció:**  $h = 0$ .

L'arbre ha de tenir un sol node, però  $1 = 2^{0+1} - 1$ .

- **Pas d'inducció:**  $h > 0$ .

Els subarbres esquerre i dret tenen alçada  $h - 1$  i, per hipòtesi d'inducció, cadascun té  $2^h - 1$  nodes. El nombre de nodes de  $T$  és la suma d'aquests nodes més un (de l'arrel):

$$\text{nodes de } T = 2(2^h - 1) + 1 = 2^{h+1} - 2 + 1 = 2^{h+1} - 1.$$

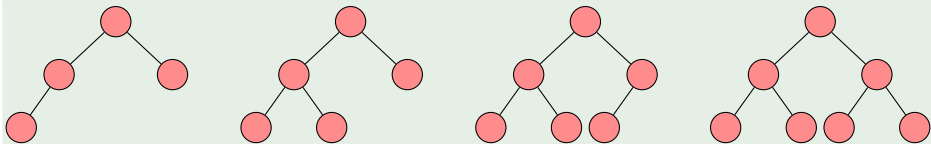
# Arbres binaris complets

## Definició

Un **arbre binari** d'alçada  $h$  és **complet** si

- 1 els  $h - 1$  primers nivells estan plens
- 2 el nivell  $h$  té les fulles el màxim a l'esquerra.

## Exemples





## Proposició

Un arbre binari complet d'alçada  $h$  té entre  $2^h$  i  $2^{h+1} - 1$  nodes.

## Demostració

Sigui  $T$  un arbre binari complet d'alçada  $h$ :

- El **mínim** nombre de nodes de  $T$  es produeix quan té un sol node a alçada  $h$ . Com que fins a alçada  $h - 1$ ,  $T$  té  $2^h - 1$  nodes, sumant l'únic node a alçada  $h$ , s'obtenen  $2^h$  nodes.
- El **màxim** nombre de nodes de  $T$  correspon a un arbre perfecte d'alçada  $h$ , que té  $2^{h+1} - 1$  nodes.

## Proposició

Un arbre binari complet d'alçada  $h$  té entre  $2^h$  i  $2^{h+1} - 1$  nodes.

## Demostració

Sigui  $T$  un arbre binari complet d'alçada  $h$ :

- El **mínim** nombre de nodes de  $T$  es produeix quan té un sol node a alçada  $h$ . Com que fins a alçada  $h - 1$ ,  $T$  té  $2^h - 1$  nodes, sumant l'únic node a alçada  $h$ , s'obtenen  $2^h$  nodes.
- El **màxim** nombre de nodes de  $T$  correspon a un arbre perfecte d'alçada  $h$ , que té  $2^{h+1} - 1$  nodes.

## Corol·lari

L'alçada d'un arbre binari complet de  $n$  nodes és  $\lfloor \log n \rfloor \in \Theta(\log n)$ .

## Demostració

Pel resultat anterior, un arbre binari complet d'alçada  $h$  i  $n$  nodes compleix:

$$2^h \leq n < 2^{h+1}.$$

Si prenem logaritmes en base 2, tenim

$$h \leq \log n < h + 1.$$

I prenent la part baixa del logaritme,

$$h = \lfloor \log n \rfloor.$$

Per tant,  $h \in \Theta(\log n)$ .

## Corol·lari

L'alçada d'un arbre binari complet de  $n$  nodes és  $\lfloor \log n \rfloor \in \Theta(\log n)$ .

## Demostració

Pel resultat anterior, un arbre binari complet d'alçada  $h$  i  $n$  nodes compleix:

$$2^h \leq n < 2^{h+1}.$$

Si prenem logaritmes en base 2, tenim

$$h \leq \log n < h + 1.$$

I prenent la part baixa del logaritme,

$$h = \lfloor \log n \rfloor.$$

Per tant,  $h \in \Theta(\log n)$ .

# Tema 4. Cues amb prioritats

## 1 Preliminars matemàtics

## 2 Cues amb prioritats

- Introducció
- Heaps
- Operacions bàsiques
- Implementació recursiva
- Implementació iterativa

## 3 Heapsort

- Algorisme bàsic
- Millores de l'algorisme bàsic

## 4 Altres aplicacions

- El problema de selecció

Moltes aplicacions requereixen processar les entrades seguint un ordre parcial determinat per **prioritats**.

- **Programació de tasques**: s'executen abans les més importants/curtes/...
- **Sistemes de simulació**: se simulen esdeveniments en ordre cronològic.
- **Algorismes voraços**: en cada moment es prova la millor opció disponible.

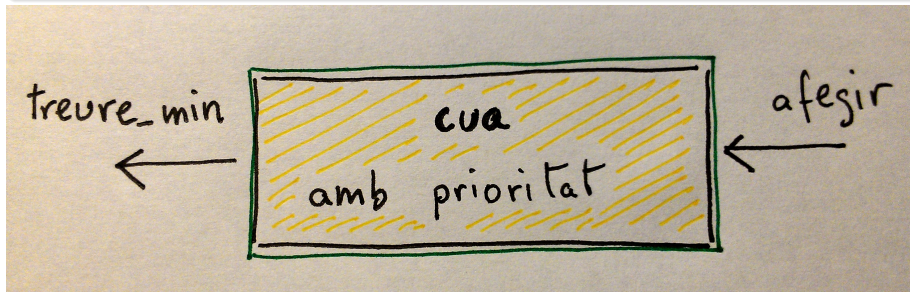
Les cues amb prioritat són una eina bàsica en el disseny d'algorismes.

# Operacions

## Definició

Una **cua amb prioritat** és una estructura de dades que disposa de dues operacions bàsiques:

- **afegir**: afegir un element i
- **treure\_min**: treure i retornar l'element més petit.



# Implementacions senzilles

implementacions	afegir	treure_min
seqüencial desordenada	$\Theta(1)$	$\Theta(n)$
seqüencial ordenada (creixent)	$\Theta(n)$	$\Theta(n)$
seqüencial ordenada (decreixent)	$\Theta(n)$	$\Theta(1)$
heaps	$\Theta(\log n)$	$\Theta(\log n)$



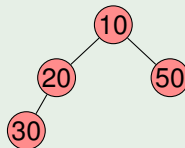
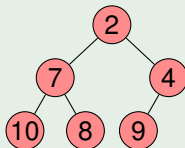
# Heaps

## Definició

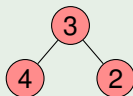
Un *min-heap* és un arbre binari complet on el valor d'un node és sempre més petit o igual que els valors dels nodes dels seus fills.

## Exemples

Són min-heaps:



No són min-heaps:



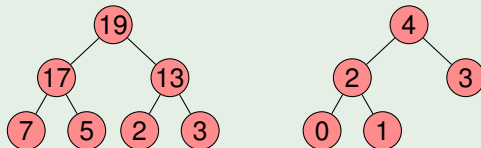
# Heaps

## Definició

Un *max-heap* és un arbre binari complet on el valor d'un node és sempre més gran o igual que els valors dels nodes dels seus fills.

## Exemples

Són max-heaps:



No són max-heaps:



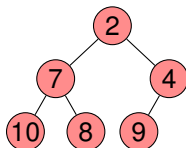
## Terminologia

- Quan parlem de **heaps** sense especificar res més, aquí ens referirem als **min-heaps**.
- En català, dels heaps en diem **munts** o **monticles**.

# Heaps

Els heaps es representen de manera compacta mitjançant vectors.

Per exemple, el heap



es representa amb el vector

	2	7	4	10	8	9		
0	1	2	3	4	5	6	7	8

No calen punters perquè:

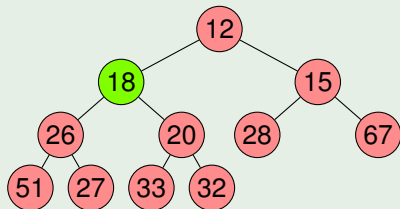
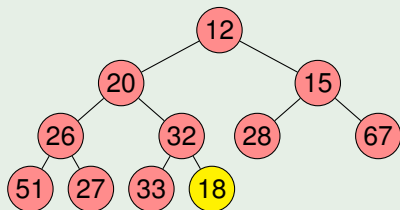
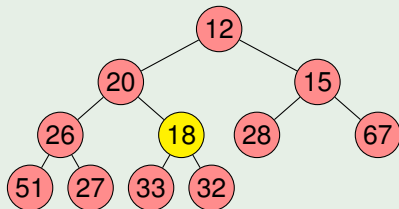
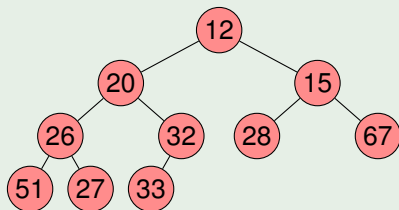
- el **pare** del node de la posició  $i$  és a la posició  $\lfloor i/2 \rfloor$
- el **fill esquerre** del node de la posició  $i$  és a la posició  $2i$
- el **fill dret** del node de la posició  $i$  és a la posició  $2i + 1$

# Operacions bàsiques

## Operació **afegir**

El més senzill és afegir l'element en la següent posició lliure del vector i fer-lo ascendir fins la posició en què es torna a complir la propietat del heap.

### Exemple (afegir la clau 18)

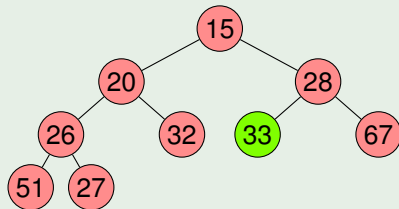
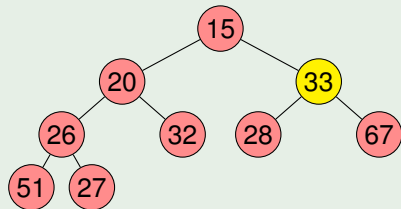
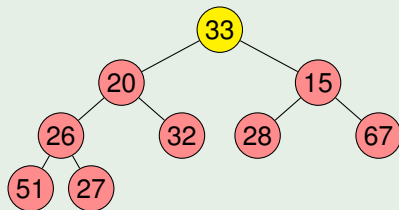
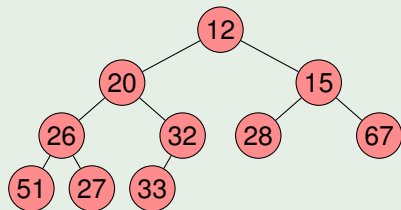


# Operacions bàsiques

## Operació **treure-min**

L'element en l'última posició del vector es trasllada a la primera i es fa descendir fins que troba la seva posició. Es retorna l'antiga arrel.

### Exemple



## Definició de la classe CuaPrio

```
template <typename Elem>
class CuaPrio {

private:
    vector<Elem> t; // Taula on es forma el heap
                  // (la posició 0 no s'utilitza)
```

# Implementació recursiva: funcions públiques

## Constructora

Crea una cua amb prioritat buida. Cost:  $\Theta(1)$ .

```
CuaPrio () {  
    t.push_back(Elem());  
}
```

## Consultar la talla

Retorna la talla de la cua amb prioritat. Cost:  $\Theta(1)$ .

```
int talla () {  
    return t.size()-1;  
}
```



# Implementació recursiva: funcions públiques

## Consultar si és buida

Indica si la cua amb prioritats és buida. Cost:  $\Theta(1)$ .

```
bool buida () {  
    return t.talla()==0;  
}
```

## Retornar element mínim

Retorna un element amb prioritats mínima. Cost:  $\Theta(1)$ .

```
Elem minim () {  
    if (buida()) throw ErrorPrec("CuaPrio buida");  
    return t[1];  
}
```

## afegir

Afegeix un nou element. Cost:  $\Theta(\log n)$ .

```
void afegir (Elem& x) {  
    t.push_back(x);  
    surar(talla());  
}
```

## treure\_min

Treu i retorna l'element mínim. Cost:  $\Theta(\log n)$ .

```
Elem treure_min () {  
    if (buida()) throw ErrorPrec("CuaPrio buida");  
    Elem x = t[1];  
    t[1] = t.back();  
    t.pop_back();  
    enfonsar(1);  
    return x;  
}
```

## surar

Fer ascendir un element fins que ocupi una posició compatible amb la condició d'ordenació del heap. Cost:  $\Theta(\log n)$ .

```
void surar (int i) {  
    if (i!=1 and t[i/2]>t[i]) {  
        swap(t[i],t[i/2]);  
        surar(i/2);  
    }  
}
```

## enfonsar

Fer descendir un element fins que ocupi una posició compatible amb la condició d'ordenació del heap. Cost:  $\Theta(\log n)$ .

```
void enfonsar (int i) {  
    int n = talla();  
    int c = 2*i;  
    if (c<=n) {  
        if (c+1<=n and t[c+1]<t[c]) c++;  
        if (t[i]>t[c]) {  
            swap(t[i],t[c]);  
            enfonsar(c);  
        }  
    }  
}
```

# Implementació iterativa

Les operacions que canvien són **afegir** i **treure\_min**, on les antigues **surar** i **enfonsar** estan optimitzades. Els costos asimptòtics són els mateixos que en el cas recursiu:  $\Theta(\log n)$ .

## afegir

```
void afegir (Elem& x) {
    t.push_back(x);
    int i = talla();
    while (i!=1 and t[i/2]>x) {
        t[i] = t[i/2];
        i = i/2;
    }
    t[i] = x;
}
```

## treure\_min

```
Elem treure_min () {  
    if (buida()) throw ErrorPrec("CuaDePrio buida");  
    int n = talla();  
    Elem e = t[1], x = t[n];  
    t.pop_back(); --n;  
    int i = 1; c = 2*i;  
    while (c<=n) {  
        if (c+1<=n and t[c+1]<t[c]) ++c;  
        if (x<=t[c]) break;  
        t[i] = t[c];  
        i = c;  
        c = 2*i;  
    }  
    t[i] = x;  
    return e;  
}
```

# Tema 4. Cues amb prioritats

## 1 Preliminars matemàtics

## 2 Cues amb prioritats

- Introducció
- Heaps
- Operacions bàsiques
- Implementació recursiva
- Implementació iterativa

## 3 Heapsort

- Algorisme bàsic
- Millores de l'algorisme bàsic

## 4 Altres aplicacions

- El problema de selecció



Les cues amb prioritats es poden fer servir per ordenar en temps  $\Theta(n \log n)$ .

L'algorisme es diu **heapsort** i va ser presentat el 1964 per J.W.J. Williams.

Donat un vector de  $n$  elements,

- 1 s'afegeixen els  $n$  elements a un *heap*:  $\Theta(n \log n)$
- 2 es fan  $n$  operacions **treure\_min** per construir un vector ordenat:  
 $\Theta(n \log n)$

El temps total és  $\Theta(n \log n)$ , que és òptim per a un algorisme d'ordenació.

## Heapsort

Amb vectors separats per al *heap* i l'entrada/sortida.

Temps:  $\Theta(n \log n)$ .

Espai auxiliar:  $n$ .

```
template <typename elem>
void heapsort (vector<elem>& T) {
    CuaPrio<elem> h;
    for (int i=0; i<n; ++i)
        h.afegir(T[i]);
    for (int i=0; i<n; ++i)
        T[i] = h.treure_min();
}
```

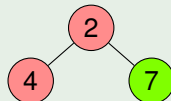
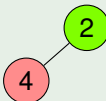
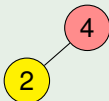
## Exemple

Suposem que partim del vector:

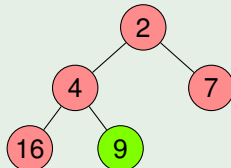
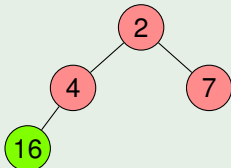
4	2	7	16	9	3	1	5
---	---	---	----	---	---	---	---

i afegim els elements a un heap, un per un.

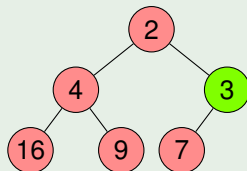
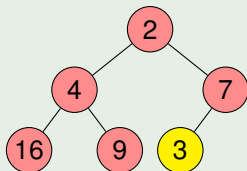
+4, +2, +7:



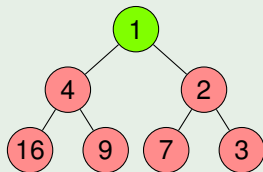
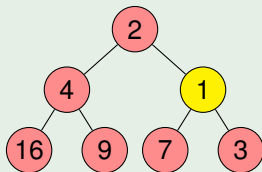
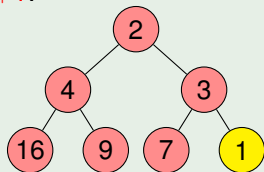
+16, +9:



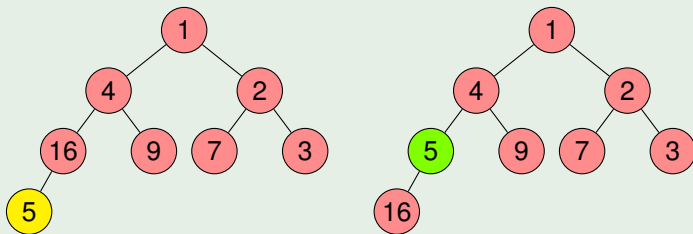
+3:



+1:



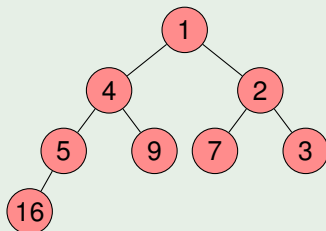
+5:



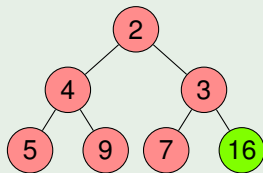
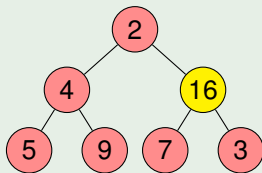
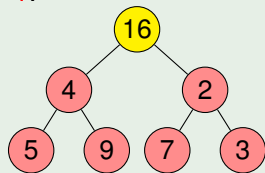
El *heap* resultant s'emmagatzema en el vector:

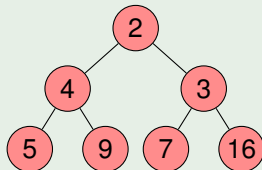
1	4	2	5	9	7	3	16
1	2	3	4	5	6	7	8

Ara traspassem els elements en ordre al vector original.

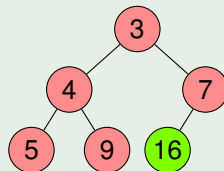
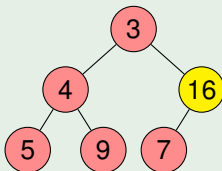
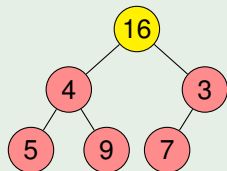


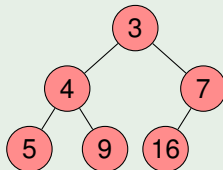
-1:



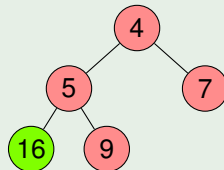
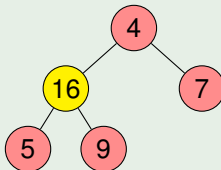
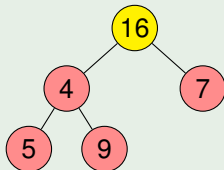


-2:

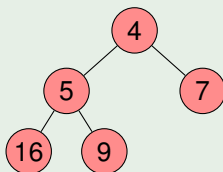




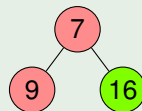
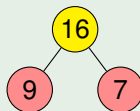
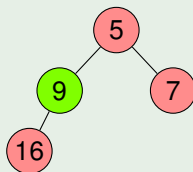
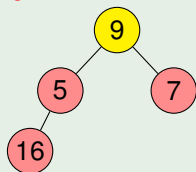
−3:



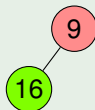
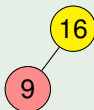




-4, -5:



-7, -9, -16:



## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

4	2	7	16	9	3	1	5
---	---	---	----	---	---	---	---

1    2    3    4    5    6    7    8

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

heap

## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

	2	7	16	9	3	1	5
--	---	---	----	---	---	---	---

1    2    3    4    5    6    7    8

4							
---	--	--	--	--	--	--	--

heap

## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

		7	16	9	3	1	5
--	--	---	----	---	---	---	---

1    2    3    4    5    6    7    8

2	4						
---	---	--	--	--	--	--	--

heap

## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

			16	9	3	1	5
--	--	--	----	---	---	---	---

1    2    3    4    5    6    7    8

2	4	7					
---	---	---	--	--	--	--	--

heap

## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

				9	3	1	5
--	--	--	--	---	---	---	---

1    2    3    4    5    6    7    8

2	4	7	16				
---	---	---	----	--	--	--	--

heap

## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

					3	1	5
--	--	--	--	--	---	---	---

1    2    3    4    5    6    7    8

2	4	7	16	9			
---	---	---	----	---	--	--	--

heap

## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

						1	5
--	--	--	--	--	--	---	---

1    2    3    4    5    6    7    8

2	4	3	16	9	7		
---	---	---	----	---	---	--	--

heap



## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

							5
--	--	--	--	--	--	--	---

1    2    3    4    5    6    7    8

1	4	2	16	9	7	3	
---	---	---	----	---	---	---	--

heap

## Exemple: evolució dels vectors (operació **afegir**)

entrada/sortida

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

1   2   3   4   5   6   7   8

1	4	2	5	9	7	3	16
---	---	---	---	---	---	---	----

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1   2   3   4   5   6   7   8

1	4	2	5	9	7	3	16
---	---	---	---	---	---	---	----

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1    2    3    4    5    6    7    8

2	4	3	5	9	7	16	
---	---	---	---	---	---	----	--

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1    2    3    4    5    6    7    8

3	4	7	5	9	16		
---	---	---	---	---	----	--	--

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1    2    3    4    5    6    7    8

4	5	7	16	9			
---	---	---	----	---	--	--	--

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1    2    3    4    5    6    7    8

5	9	7	16				
---	---	---	----	--	--	--	--

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1    2    3    4    5    6    7    8

7	9	16					
---	---	----	--	--	--	--	--

heap



## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1    2    3    4    5    6    7    8

9	16						
---	----	--	--	--	--	--	--

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

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

1    2    3    4    5    6    7    8

16							
----	--	--	--	--	--	--	--

heap

## Exemple: evolució dels vectors (operació **treure\_min**)

entrada/sortida

1	2	3	4	5	7	9	16
---	---	---	---	---	---	---	----

1    2    3    4    5    6    7    8

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

heap

## Primera millora

Implementar l'algorisme sobre un únic vector fent una divisió en:

- una part esquerra per mantenir el *heap*
- una part dreta per a l'entrada/sortida

Cada cop que es fa una operació de **treure\_min**, s'escriu el mínim com a primer element de la part dreta. Els elements queden ordenats de manera **descendent**.

Si es volen en ordre ascendent, es pot fer servir un **max-heap**.

Exemple: evolució dels vectors (operació **afegir**)



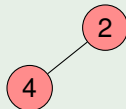
4	2	7	16	9	3	1	5
---	---	---	----	---	---	---	---

Exemple: evolució dels vectors (operació **afegir**)

4

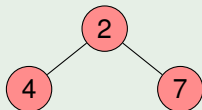
4	2	7	16	9	3	1	5
---	---	---	----	---	---	---	---

Exemple: evolució dels vectors (operació **afegir**)



2	4	7	16	9	3	1	5
---	---	---	----	---	---	---	---

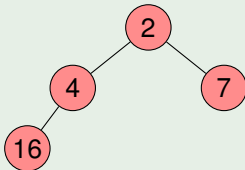
Exemple: evolució dels vectors (operació **afegir**)



2	4	7	16	9	3	1	5
---	---	---	----	---	---	---	---

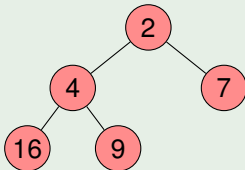


Exemple: evolució dels vectors (operació **afegir**)



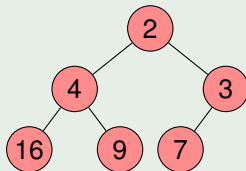
2	4	7	16	9	3	1	5
---	---	---	----	---	---	---	---

Exemple: evolució dels vectors (operació **afegir**)



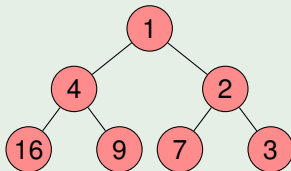
2	4	7	16	9	3	1	5
---	---	---	----	---	---	---	---

Exemple: evolució dels vectors (operació **afegir**)



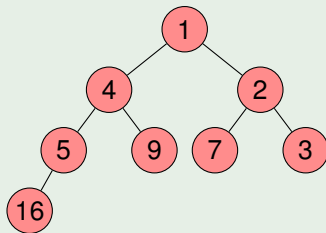
2	4	3	16	9	7	1	5
---	---	---	----	---	---	---	---

## Exemple: evolució dels vectors (operació **afegir**)



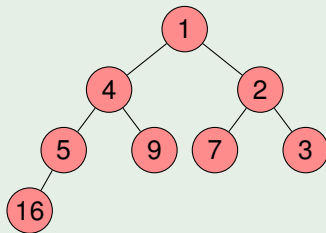
1	4	2	16	9	7	3	5
---	---	---	----	---	---	---	---

## Exemple: evolució dels vectors (operació **afegir**)



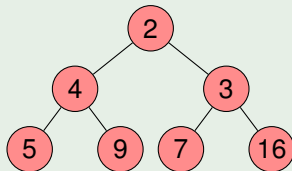
1	4	2	5	9	7	3	16
---	---	---	---	---	---	---	----

## Exemple: evolució dels vectors (operació **treure\_min**)



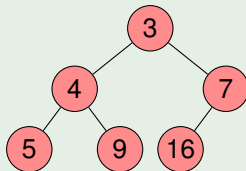
1	4	2	5	9	7	3	16
---	---	---	---	---	---	---	----

Exemple: evolució dels vectors (operació **treure\_min**)



2	4	3	5	9	7	16	1
---	---	---	---	---	---	----	---

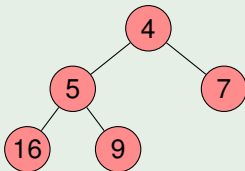
Exemple: evolució dels vectors (operació **treure\_min**)



3	4	7	5	9	16	2	1
---	---	---	---	---	----	---	---

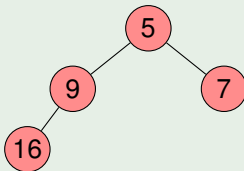


Exemple: evolució dels vectors (operació **treure\_min**)



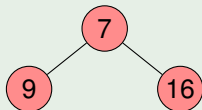
4	5	7	16	9	3	2	1
---	---	---	----	---	---	---	---

Exemple: evolució dels vectors (operació **treure\_min**)



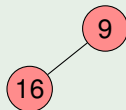
5	9	7	16	4	3	2	1
---	---	---	----	---	---	---	---

Exemple: evolució dels vectors (operació **treure\_min**)



7	9	16	5	4	3	2	1
---	---	----	---	---	---	---	---

Exemple: evolució dels vectors (operació **treure\_min**)



9	16	7	5	4	3	2	1
---	----	---	---	---	---	---	---

Exemple: evolució dels vectors (operació **treure\_min**)

16

16	9	7	5	4	3	2	1
----	---	---	---	---	---	---	---

Exemple: evolució dels vectors (operació **treure\_min**)



16	9	7	5	4	3	2	1
----	---	---	---	---	---	---	---

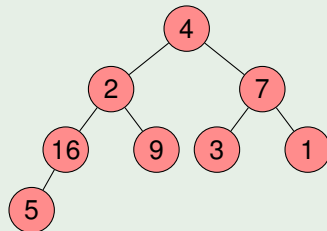
## Segona millora

**Construir el heap en temps  $\Theta(n)$**  en lloc de  $\Theta(n \log n)$  seguint els passos següents:

- 1 Introduir els elements en el heap en qualsevol ordre (i temps lineal).
- 2 Per cada node  $x$  que no sigui una fulla, en ordre decreixent de posició **enfonsar**  $x$

(De tota manera, el cost de l'algorisme continua sent  $\Theta(n \log n)$ .)

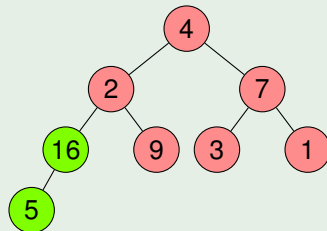
## Exemple



4	2	7	16	9	3	1	5
---	---	---	----	---	---	---	---

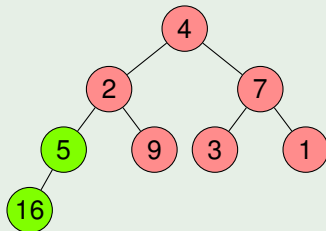


## Exemple



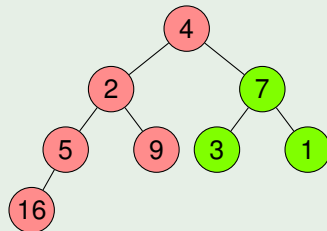
4	2	7	16	9	3	1	5
---	---	---	----	---	---	---	---

## Exemple



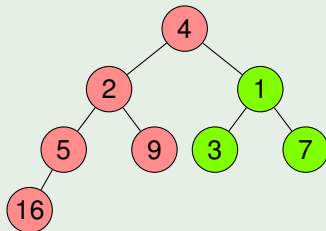
4	2	7	5	9	3	1	16
---	---	---	---	---	---	---	----

## Exemple



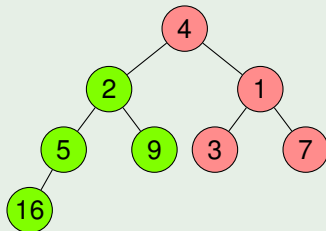
4	2	7	5	9	3	1	16
---	---	---	---	---	---	---	----

## Exemple



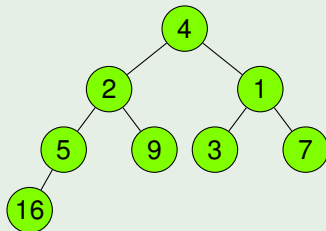
4	2	1	5	9	3	7	16
---	---	---	---	---	---	---	----

## Exemple



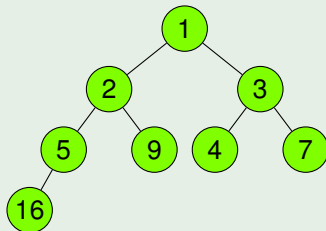
4	2	1	5	9	3	7	16
---	---	---	---	---	---	---	----

## Exemple



4	2	1	5	9	3	7	16
---	---	---	---	---	---	---	----

## Exemple



1	2	3	5	9	4	7	16
---	---	---	---	---	---	---	----

# Millors de l'algorisme bàsic

Com que molts dels subheaps són petits, **enfonsar** fa  $\Theta(n)$  intercanvis

## Exemple

Per a un heap de 127 nodes, hi ha:

- 1 heap de mida 127 i alçada 6
- 2 heaps de mida 63 i alçada 5
- ...
- 32 heaps de mida 3 i alçada 1

## Intercanvis en arbres perfectes

Un arbre perfecte d'alçada  $h$  té  $2^i$  nodes a nivell  $i$  per cada  $0 \leq i \leq h-1$ .  
Cada node a nivell  $i$  és arrel d'un subheap amb alçada  $h-i$ .

Com a molt es fan  $\sum_{0 \leq i \leq h-1} 2^i \cdot (h-i) = 2^{h+1} - h - 2 < n$  intercanvis

ja que un arbre perfecte d'alçada  $h$  té  $2^{h+1} - 1$  nodes.

(Per arbres complets, es demostra la mateixa fita.)



# Tema 4. Cues amb prioritats

## 1 Preliminars matemàtics

## 2 Cues amb prioritats

- Introducció
- Heaps
- Operacions bàsiques
- Implementació recursiva
- Implementació iterativa

## 3 Heapsort

- Algorisme bàsic
- Millores de l'algorisme bàsic

## 4 Altres aplicacions

- El problema de selecció

# El problema de selecció

## Problema de selecció

Donat un vector  $S$  de naturals i un  $k \in \mathbb{N}$ ,  
determinar el  $k$ -èsim element més petit de  $S$ .

Fent servir els heaps, podem trobar un nou algorisme:

- 1 Construir un min-heap a partir de  $S$ .  $\Theta(n)$
- 2 Efectuar  $k$  operacions **treure\_min** del min-heap.  $\Theta(k \log n)$
- 3 Retornar l'últim element extret.  $\Theta(1)$

Cost total:  $\Theta(n + k \log n)$ .

La **mediana** correspon a  $k = n/2$ . Cost:  $\Theta(n \log n)$ .

En el cas  $k = O(n / \log n)$ , el cost és  $\Theta(n)$ .

# El problema de selecció

## Problema de selecció

Donat un vector  $S$  de naturals i un  $k \in \mathbb{N}$ ,  
determinar el  $k$ -èsim element més petit de  $S$ .

Fent servir els heaps, podem trobar un nou algorisme:

- 1 Construir un min-heap a partir de  $S$ .  $\Theta(n)$
- 2 Efectuar  $k$  operacions **treure\_min** del min-heap.  $\Theta(k \log n)$
- 3 Retornar l'últim element extret.  $\Theta(1)$

Cost total:  $\Theta(n + k \log n)$ .

La **mediana** correspon a  $k = n/2$ . Cost:  $\Theta(n \log n)$ .

En el cas  $k = O(n / \log n)$ , el cost és  $\Theta(n)$ .