Algorithmique 1
Tas binaires

Hajer Akid & Sandrine Julia

Semestre 4

Tas binaire : structure de données

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

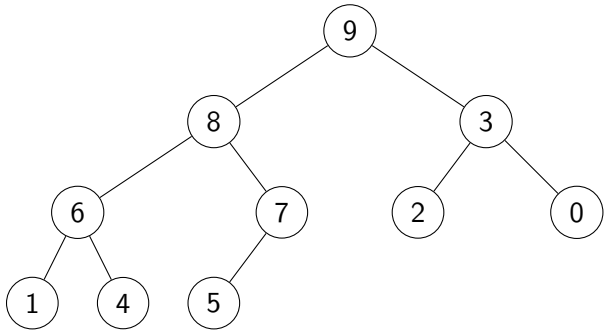
- Un **tas ascendant** ou **tas-max** est un tableau dont les cases sont numérotées de 1 à n vérifiant :

$$\text{si } 2i \leq n : \quad T[i] \geq T[2i]$$
$$\text{si } 2i+1 \leq n : \quad T[i] \geq T[2i+1]$$

- on définit de la même manière un **tas descendant** ou **tas-min**

Tas binaire : représentation arborescente

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



Tas binaire : stockage dans un tableau

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

- la valeur de la racine de l'arbre est en $T[1]$
- la valeur de la racine du fils gauche de $T[i]$ est $T[2i]$
  la valeur de la racine du fils droit de $T[i]$ est $T[2i+1]$
- le père de $T[i]$ est $T[\lfloor i/2 \rfloor]$
- le sous-tableau de $T[\lfloor n/2 \rfloor + 1]$ jusqu'à $T[n]$ correspond aux feuilles.

Tamisage

L'opération de **tamisage** s'applique à un tableau qui n'est pas un tas à cause du seul élément de sa première case.

Comme le tableau T suivant :

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

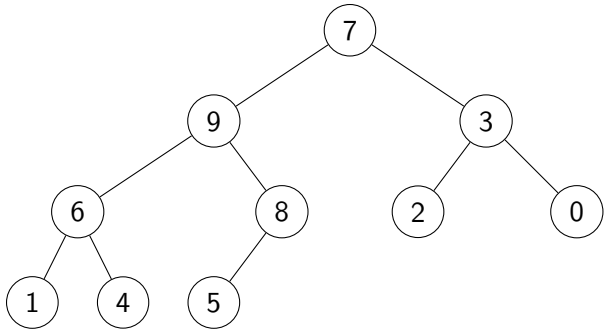
Dans l'arbre, le tamisage échange l'élément de la racine avec le plus grand de ses fils, et ainsi de suite jusqu'à ce qu'il soit à sa place (*bubble down*).

La complexité est en $O(\log(n))$. On obtient le tas :

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

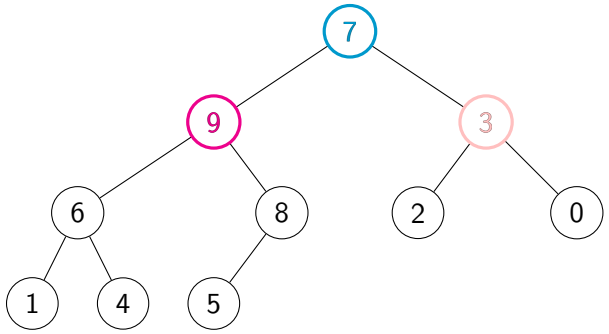
Exemple

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



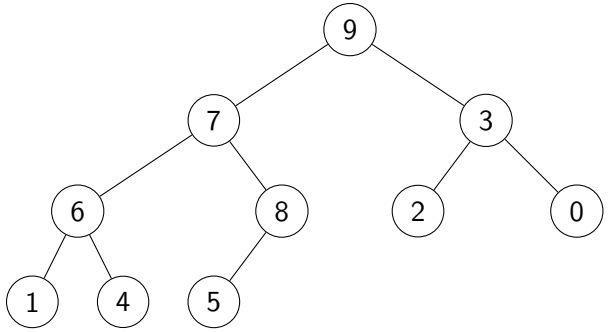
Exemple

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



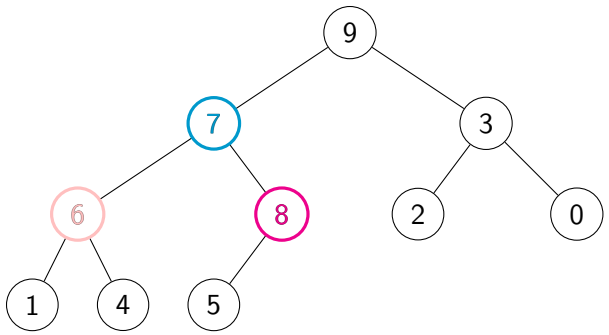
Exemple

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



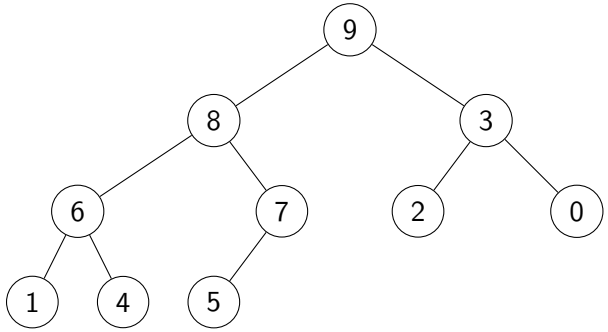
Exemple

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



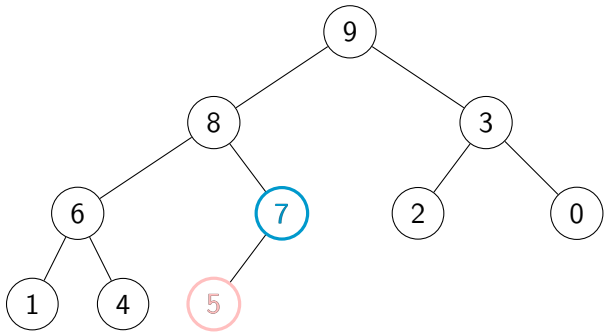
Exemple

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



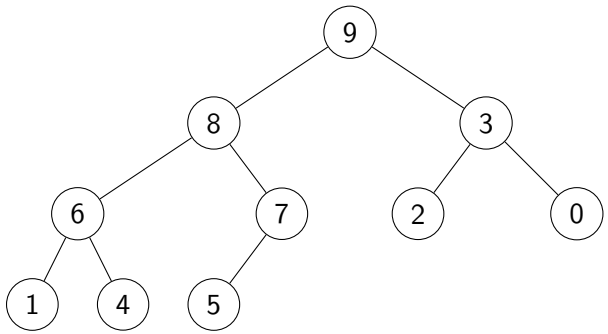
Exemple

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



Exemple

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



Tamisage en tas-max

```
max_Heapify(entier T[], entier n, entier i) {
  //version fonction récursive
  g ← gauche(i)
  d ← droit(i)
  si (g ≤ n et T[g] > T[i])
    imax ← g
  sinon
    imax ← i
  si (d ≤ n et T[d] > T[imax])
    imax ← d
  si (imax ≠ i) {
    échanger(T[i],T[imax])
    T ← max_Heapify(T, n, imax)
  }
  retourner T
}
```


Tamisage en tas-max

```
max_Heapify_p(var entier T[], entier n, entier i) {
  //version procédure récursive -> sur place
  g ← gauche(i)
  d ← droit(i)
  si (g ≤ n et T[g] > T[i])
    imax ← g
  sinon
    imax ← i
  si (d ≤ n et T[d] > T[imax])
    imax ← d
  si (imax ≠ i) {
    échanger(T[i],T[imax])
    max_Heapify_p(T, n, imax)
  }
}
```


Extraction

Cette opération consiste à :

- retirer l'élément à la racine
- mettre à la racine le dernier élément en diminuant la taille du tas
- re-tamiser le tas

Sa complexité est en $O(\log(n))$

Extraction

Cette opération consiste à :

- retirer l'élément à la racine
- mettre à sa place le dernier élément en diminuant la taille du tas
- re-tamiser le tas

Sa complexité est en $O(\log(n))$

```
entier heappop_max (var entier T[], entier n) {
  max ← T[1]
  échanger(T[1], T[n])
  n ← n−1
  max_Heapify_p(T, n, 1)
  retourner max
}
```

Insertion

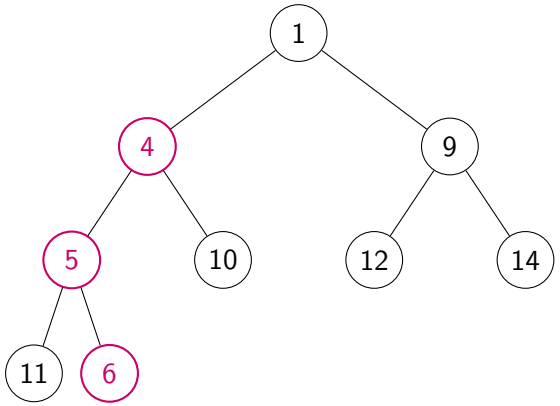
Insertion d'un élément

- On place l'élément à insérer après les autres
- Il est situé sur une feuille
- On propage la propriété de tas de cette feuille jusqu'à la racine (*bubble up !*)
- La complexité est en $O(\log(n))$

Création d'un tas à partir d'un tableau

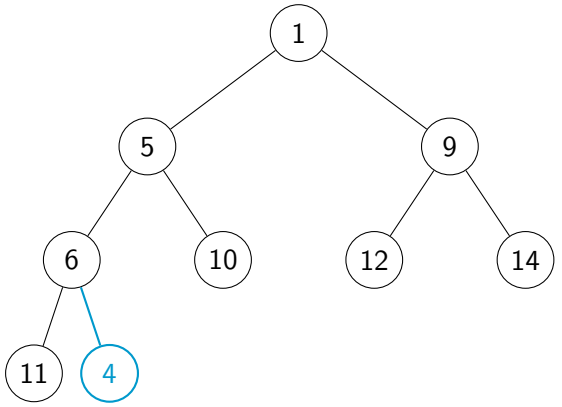
- Solution immédiate en $O(n\log(n))$
- Mais on peut transformer un tableau en tas sans procéder à l'insertion de ses éléments un à un
- C'est plus efficace !

Insertion dans un tas-min



On ajoute 4 dans un tas_min.

Insertion dans un tas-min



On ajoute 4 dans un tas_min.

Insertion

```
insertion_min_Heap(var entier T[], entier elm){
  //version itérative et sur place
  n ← longueur(T)
  n ← n+1
  T[n+1] ← elm
  père ← n div 2
  fils ← n
  tant que (père > 0 et T[père] > T[fils]) {
    échanger (T[père],T[fils])
    fils ← père
    père ← père div 2
   }
 }
```

Recherche d'un élément et suppression

Recherche

- Un tas n'est pas adapté à la recherche d'un élément donné
- Le temps de recherche est forcément linéaire

Suppression

- Avant de supprimer un élément, il faut le rechercher ...
- La suppression consisterait ensuite à le remplacer par le dernier élément, diminuer le nombre d'éléments de 1 et tamiser.

Recherche de plus grand élément et tri

Recherche du k^e plus grand élément

- On transforme un tableau en tas
- On extrait successivement les k premiers éléments d'un tas-max

Tri par tas (Heapsort)

- On peut trier le tableau **sur place** en le remplissant par la fin
- On effectue alors des tas-max avec un nombre décroissant d'éléments

File d'attente avec priorité

File d'attente

- On insère les éléments avec une certaine priorité dans un tas-max
- On les extrait successivement dans l'ordre de plus grande priorité

Modification des priorités

- On peut modifier en cours les priorités des éléments
- La recherche d'un élément n'est pas efficace.