Le tri par tas ou heapsort

Tri basé sur l’interprétation d’un tableau comme un arbre binaire complet

Rappel : représentation contiguë d'un arbre binaire complet

- Séquence des **n** nœuds : mémorisée par niveaux dans un tableau de taille **n**

  t : tableau sur [**1**..n] d'Élément

  Racine en position 1

  X en position **i**

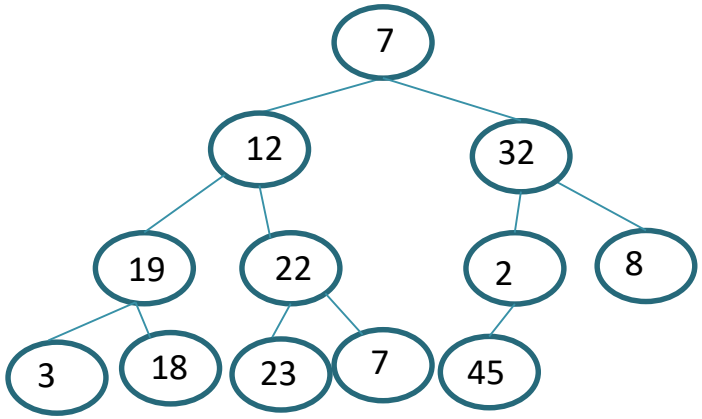
  Gauche(X) en position **i*2**

  Droit(X) en position **i*2+1**

  Père(X) en position **i div 2**

Correspondance Arbre /Tableau

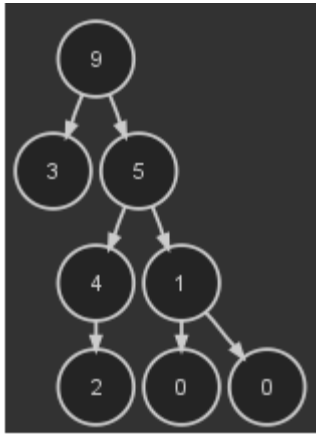
a : Arbre	**a : entier sur 1..n**
a↑.el	**T[a]**
a=nil	**a > n**
a↑.g	**a*2**
a↑.d	**a*2+1**
a↑.g ≠ nil	**2*a ≤ n**
a↑.d ≠ nil	**2*a+1 ≤ n**
feuille(a)	**a*2 > n**



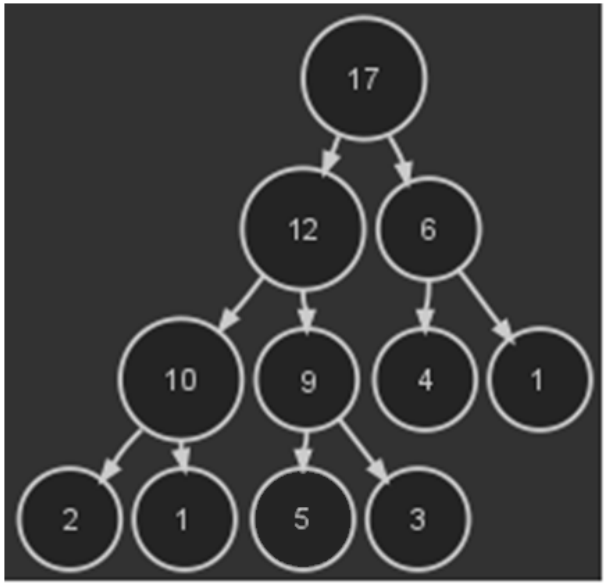
	1	2	3	4	5	6	7	8	9	10	11	12
T	7	12	32	19	22	2	8	3	18	23	7	45

Utilisation d'un arbre binaire complet : Arbre partiellement ordonné

- Un arbre binaire partiellement ordonné est un arbre tel que chaque nœud est supérieur à ses fils.
- Exemples :



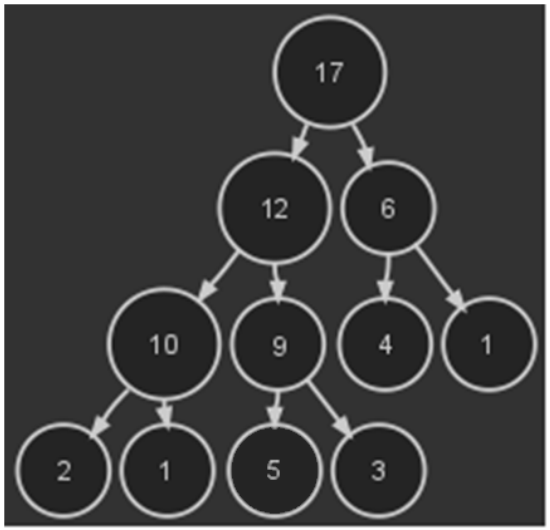
Arbre partiellement ordonné



Arbre binaire complet partiellement ordonné

Tas : définition

- Un tableau peut être interprété comme un arbre binaire complet
- Un tas est un tableau dont l'arbre correspondant est un arbre partiellement ordonné



Arbre binaire complet partiellement ordonné

Le tas correspondant à cet arbre est le tableau :

t = [17,12,6,10,9,4,1,2,1,5,3]

Tri par tas d'un tableau (heapsort)

Principe

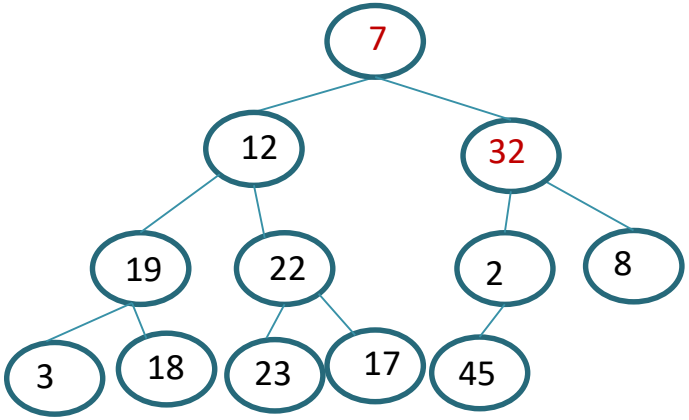
- On interprète le tableau comme un arbre binaire complet
- On procède en deux étapes
  1. On transforme cet arbre binaire en un arbre partiellement ordonné (tas), par permutation de nœuds $\Rightarrow$ la plus grande valeur sera la racine
  2. On procède au tri par échange de la racine avec la feuille la plus à droite (supprimée de l'arbre); le tableau est divisé en 2 parties :
     - La partie droite triée
     - La partie gauche : le tas restant à trier

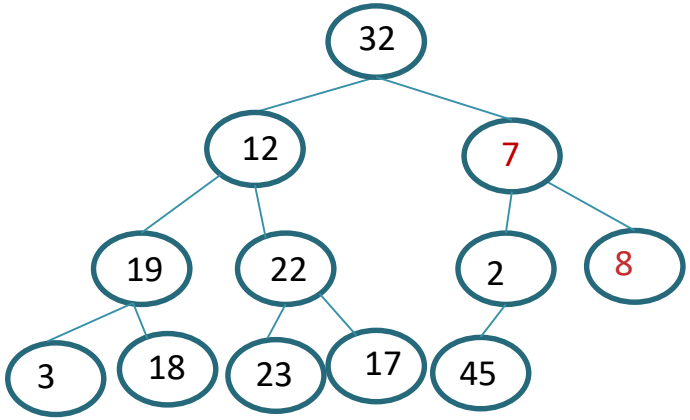
Tri par tas d'un tableau (heapsort)

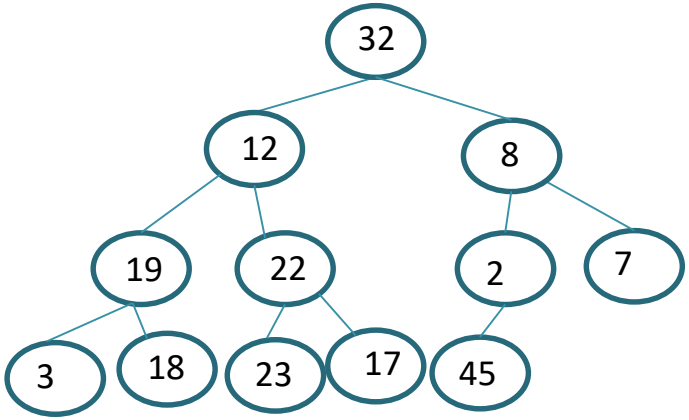
1 - Transformation de l'arbre en tas

- On définit l'action placer qui place la racine de cet arbre binaire de sorte que l'arbre soit un tas.
- On procède de la manière suivante :
  - Si la valeur de la racine est inférieure à la valeur de l'un de ses fils, on échange la valeur de la racine avec celle du fils de valeur maximale
  - On poursuit récursivement en appliquant placer au sous-arbre fils concerné par l'échange

Transformation de l'arbre en tas
Exemple de placement







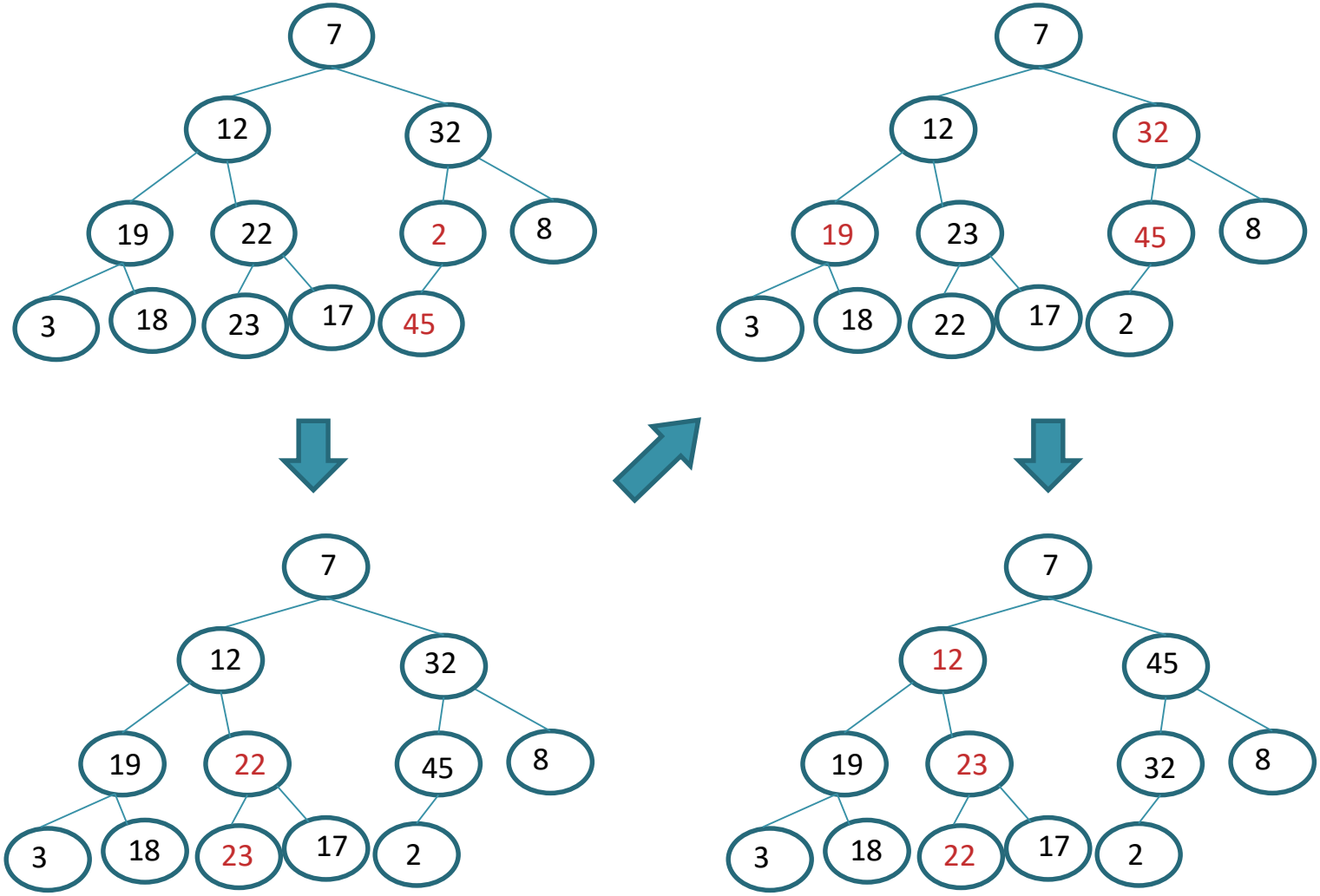
Réalisation de l'action placer

```
action placer(modifié t : tableau sur [1..nmax] d'entiers
                    consulté a : entier sur 1..nmax, n : entier > 0)
// Effet: place la valeur de la racine a dans l'arbre de sorte que
//         l'arbre soit partiellement ordonné (modifie le tableau t[1..n] de taille n)
Lexique
 // paramètre a : entier sur 1..n  : racine de l'arbre à ordonner
// paramètre  t : tableau sur [1..nmax] d'entiers  : tableau représentant l'arbre
 // paramètre  n : entier > 0                    : taille de l'arbre / tableau
 k : entier sur 1..n  // fils maximum de a
 x : entier           // intermédiaire pour échange
Algorithme
k ← a*2               // k = indice du fils gauche de a
si k ≤ n
alors  // a n'est pas une feuille
   si k < n etpuis t[k] < t[k+1]  // comparaison fils gauche et fils droit
   alors k ← k+1   // fils droit > fils gauche => k = indice du fils droit
   fsi
   si t[a] < t[k]
   alors // on échange les valeurs de t[a] et t[k]
       x ← t[a] ; t[a] ← t[k] ; t[k] ← x
      placer(t, k, n)
   fsi
fsi
```

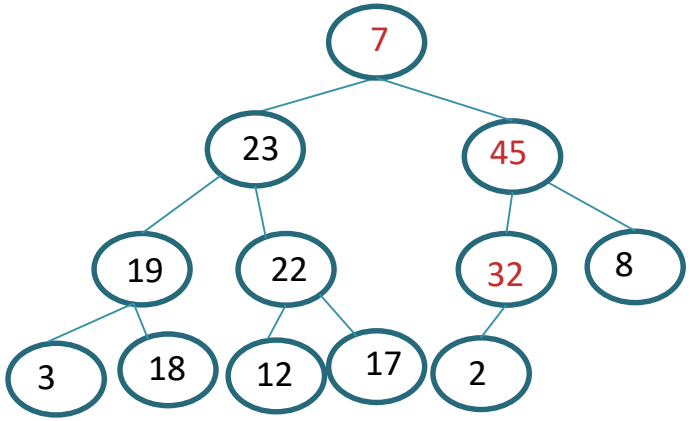
Version itérative de l'action placer

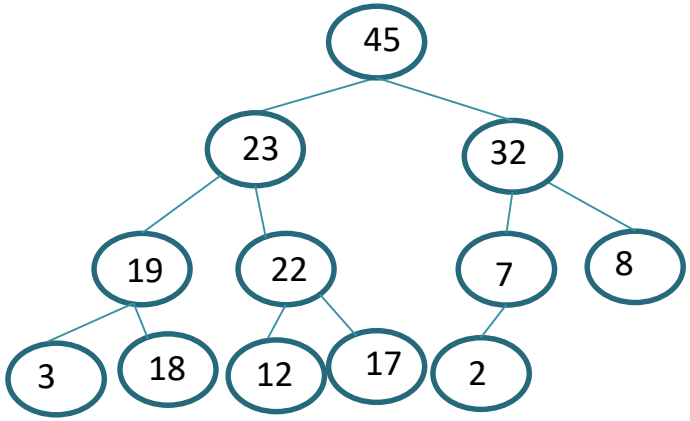
```
action placer(consulté a : entier sur 1..n)
// Effet: place la valeur de la racine a dans l'arbre de sorte que
//          l'arbre soit partiellement ordonné (modifie le tableau t[1..n])
Lexique
 // paramètre a : entier sur 1..n  racine de l'arbre à ordonner
 r : entier sur 1..n  // racine courante
 k : entier sur 1..n  // fils maximum de a
 x : entier           // intermédiaire pour échange
Algorithme
r ← a ; k ← r+r     // r*2
tantque k ≤ n
   // r n'est pas une feuille
   si k < n etpuis t[k] < t[k+1] alors k ← k+1 fsi // fils droit > fils gauche
   si t[r] < t[k]
   alors // on échange les valeurs de t[r] et t[k]
      x ← t[r] ; t[r] ← t[k] ; t[k] ← x
      r ← k ; k ← r+r
   sinon // on a fini : plutôt que de gérer un booléen …
     k ← n+1
   fsi
fsi
```

Pour transformer l'arbre en tas on commence par les nœuds non feuilles les plus bas dans l'arbre



Pour transformer l'arbre en tas on commence par les nœuds non feuilles les plus bas dans l'arbre





L'arbre est transformé en tas

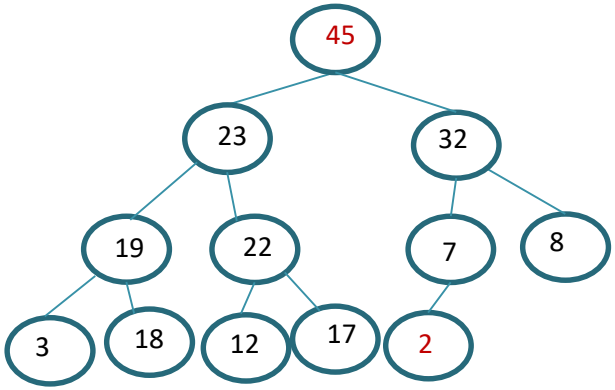
T	45	23	32	19	22	7	8	3	18	12	17	2	
	1	2	3	4	5	6	7	8	9	10	11	12	

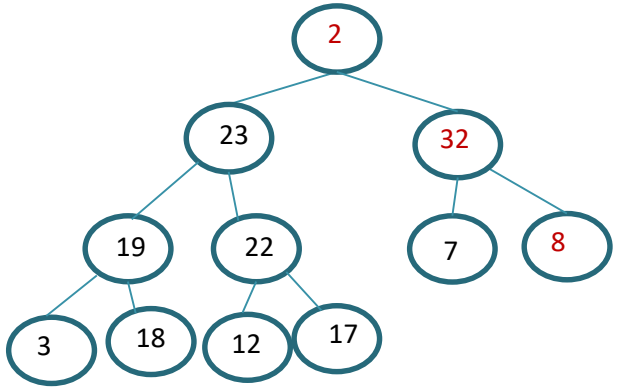
Tri par tas d'un tableau (heapsort)

2 - Algorithme de tri

- On procède de la manière suivante :
  - On échange la racine avec la feuille la plus à droite qui est le dernier élément du tableau
  - On place ainsi la plus grande valeur à sa place dans le futur tableau trié
- Le tableau est donc divisé en 2 parties :
  - La partie droite triée
  - La partie gauche : le tas restant à trier
- La nouvelle valeur de la racine est replacée dans l'arbre de façon à obtenir à nouveau un arbre partiellement ordonné (action placer)
- On poursuit le processus pour le tas restant à trier

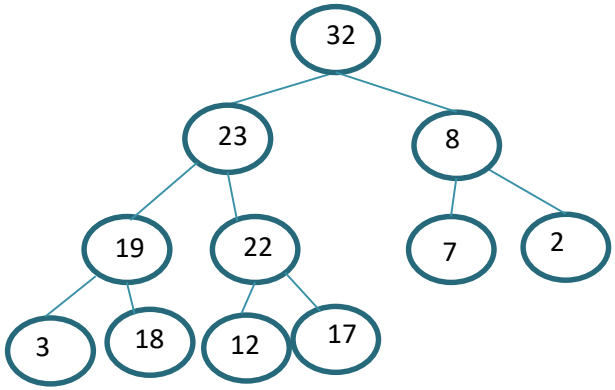
Application sur l'exemple





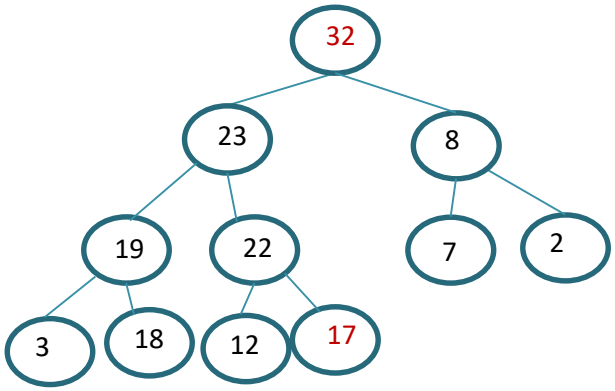
t	45	23	32	19	22	7	8	3	18	12	17	2	
	1	2	3	4	5	6	7	8	9	10	11	12	

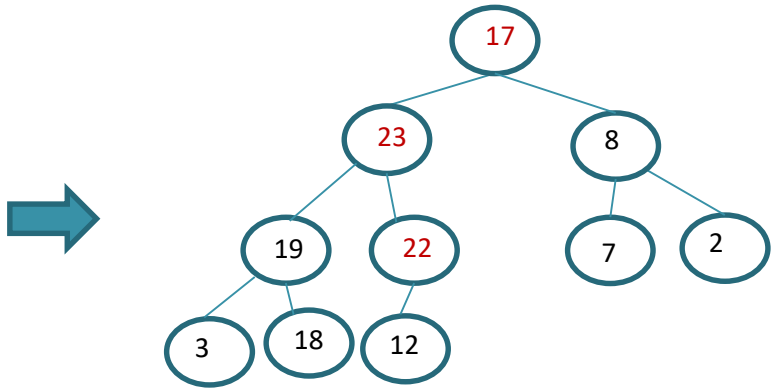
t	2	23	32	19	22	7	8	3	18	12	17	45	
	1	2	3	4	5	6	7	8	9	10	11	12	



t	32	23	8	19	22	7	2	3	18	12	17	45	
	1	2	3	4	5	6	7	8	9	10	11	12	

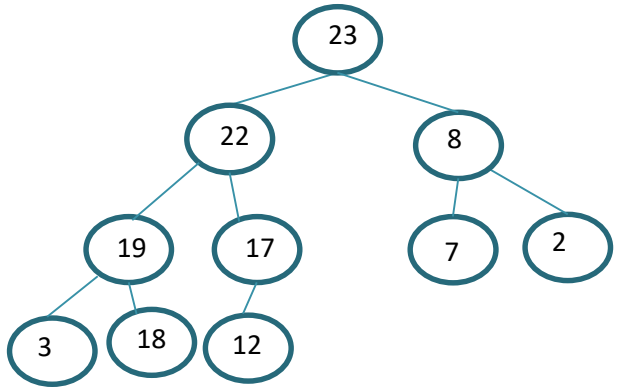
Application sur l'exemple





t

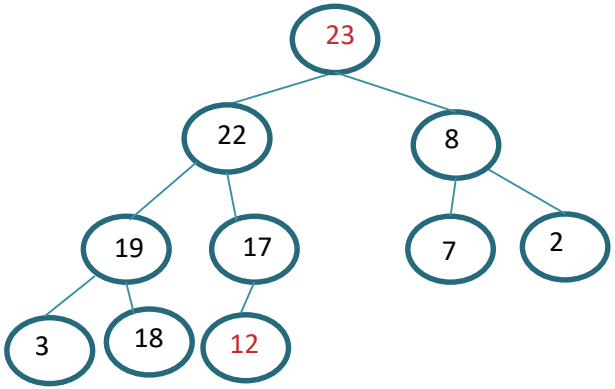
32	23	8	19	22	7	2	3	18	12	17	45	
1	2	3	4	5	6	7	8	9	10	11	12	

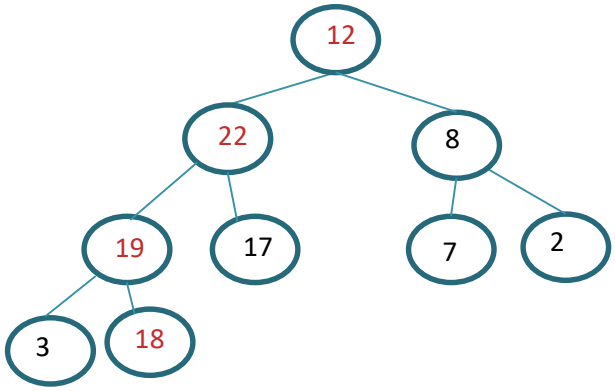


t

23	22	8	19	17	7	2	3	18	12	32	45	
1	2	3	4	5	6	7	8	9	10	11	12	

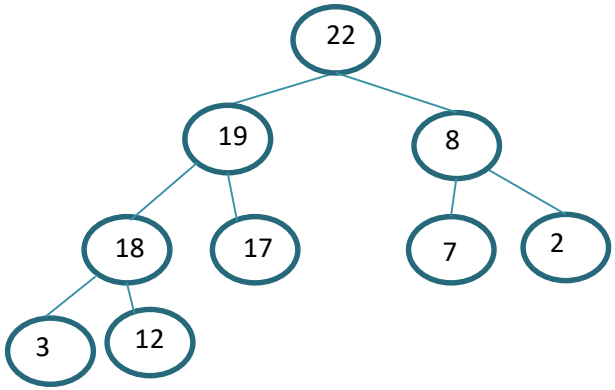
Application sur l'exemple





t

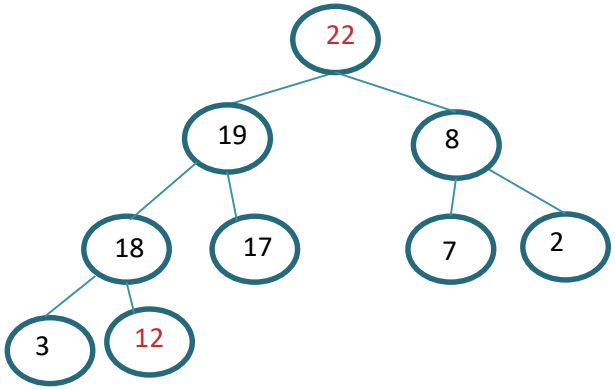
23	22	8	19	17	7	2	3	18	12	32	45	
1	2	3	4	5	6	7	8	9	10	11	12	

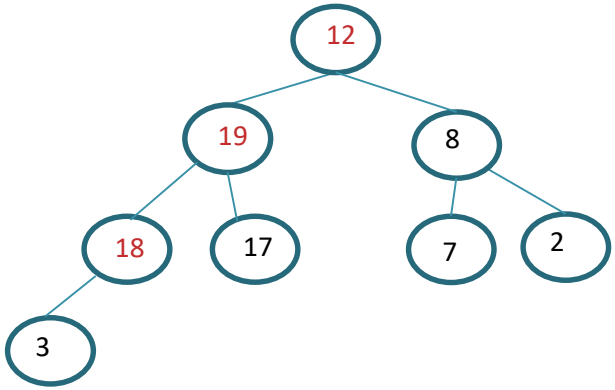


t

22	19	8	18	17	7	2	3	12	23	32	45	
1	2	3	4	5	6	7	8	9	10	11	12	

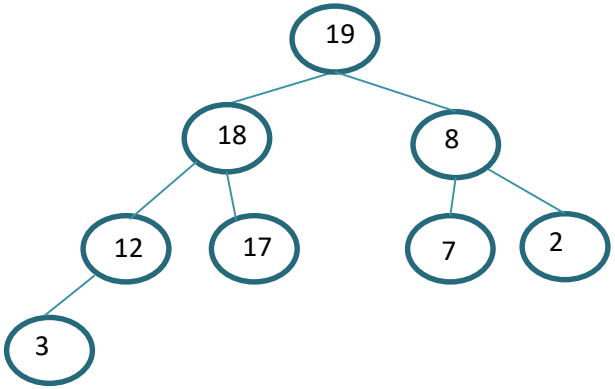
Application sur l'exemple





t

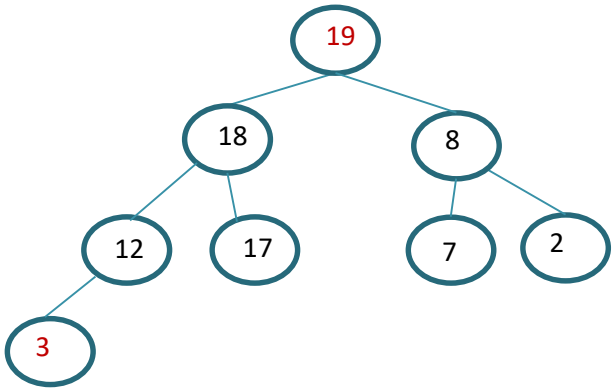
22	19	8	18	17	7	2	3	12	23	32	45	
1	2	3	4	5	6	7	8	9	10	11	12	

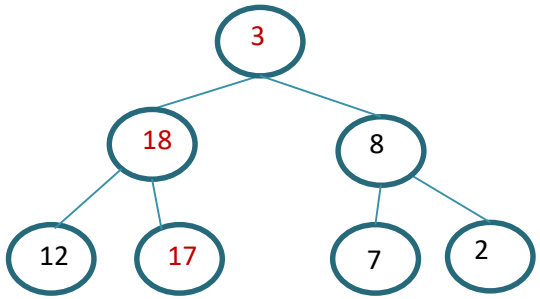


t

19	18	8	12	17	7	2	3	22	23	32	45	
1	2	3	4	5	6	7	8	9	10	11	12	

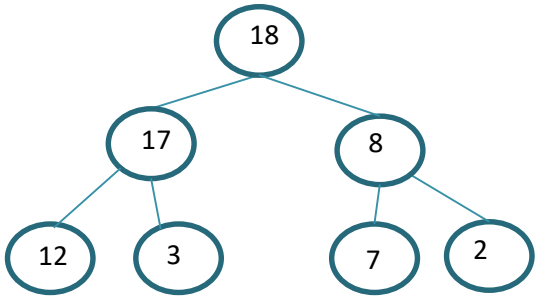
Application sur l'exemple





t

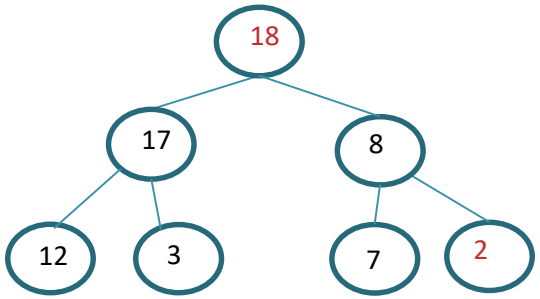
19	18	8	12	17	7	2	3	22	23	32	45	
1	2	3	4	5	6	7	8	9	10	11	12	

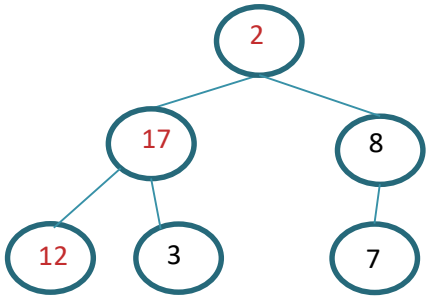


t

18	17	8	12	3	7	2	19	22	23	32	45	
1	2	3	4	5	6	7	8	9	10	11	12	

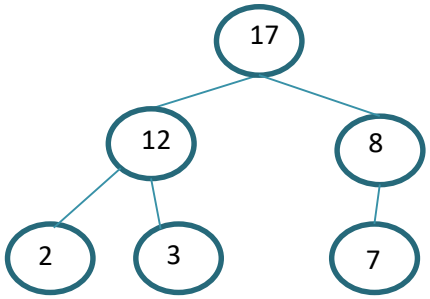
Application sur l'exemple





t

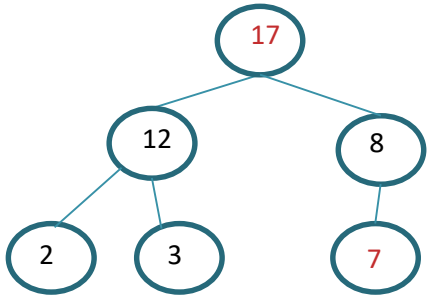
18	17	8	12	3	7	2	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

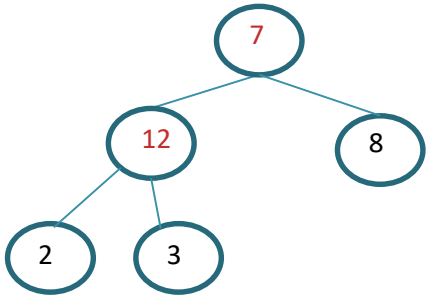


t

17	12	8	2	3	7	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

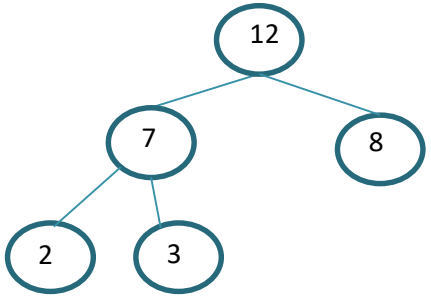
Application sur l'exemple





t

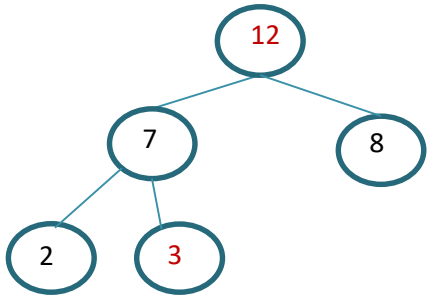
17	12	8	2	3	7	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

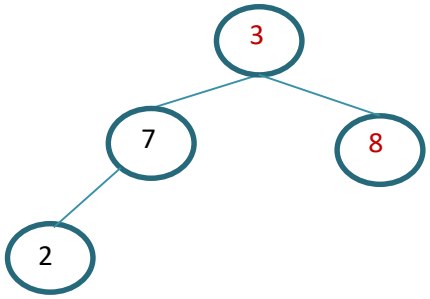


t

12	7	8	2	3	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

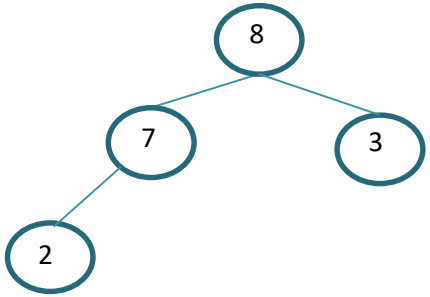
Application sur l'exemple





t

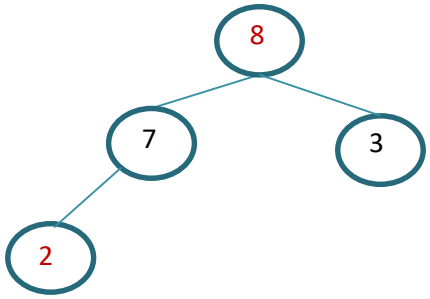
12	7	8	2	3	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

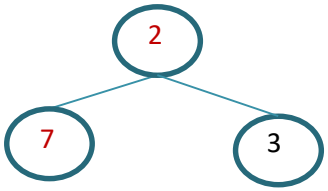


t

8	7	3	2	12	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

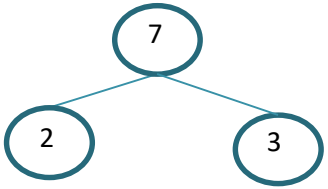
Application sur l'exemple





t

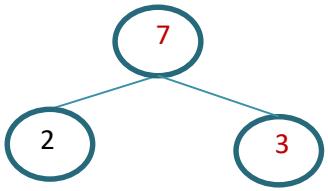
8	7	3	2	12	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

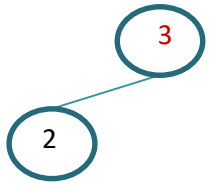


t

7	2	3	8	12	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

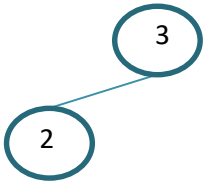
Application sur l'exemple





t

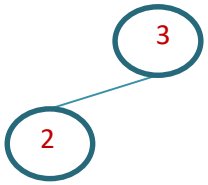
7	2	3	8	12	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	



t

3	2	7	8	12	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

Application sur l'exemple



t

2	3	7	8	12	17	18	19	22	23	32	**45**	
1	2	3	4	5	6	7	8	9	10	11	12	

Le tableau est trié !

Réalisation de l'action de tri

```
action triParTas(modifié t : tableau sur [1..nmax] d'entiers,  consulté n : entier > 0)
// Effet: trie le tableau t de n éléments selon le tri par tas
Lexique
 i : entier sur 1..n  // indice de parcours de t
x : entier            // intermédiaire pour échange
 action utilisée : placer
Algorithme
// Etape 1 : transformation du tableau en un tas
i ← n div 2
tantque i ≥ 1 faire
   placer(t, i, n)
   i ← i - 1
ftq
// Etape 2 : phase de tri
i ← n
tantque i > 1
      x ← t[1] ; t[1] ← t[i] ; t[i] ← x   // échange racine t[1] avec dernière feuille t[i]
      placer(t,1, i-1)
      i ← i - 1
ftq
```