Algorithmes de tris

Stéphane Grandcolas

stephane.grandcolas@univ-amu.fr

Tris.

- *Arbre de décision* : tri par insertion.
- Complexité des tris par comparaison dans le pire des cas : borne minimale.
- *Tas* et *tri par tas*.
- *Tri par dénombrement*, *tri par base*.

Tris.

organiser un ensemble d'objets selon un ordre déterminé

relation d'ordre : comparaison de clés

dans nos exemple nous confondrons les objets avec leurs clés

- tri par comparaison versus tri par indexation
- tri sur place : espace mémoire de taille constante
- tri stable : préserve l'ordre initial en cas d'égalité

Tris.

1. Tris en $O(n^2)$.

- tri à bulles,
- tri par insertion,
- tri par sélection.

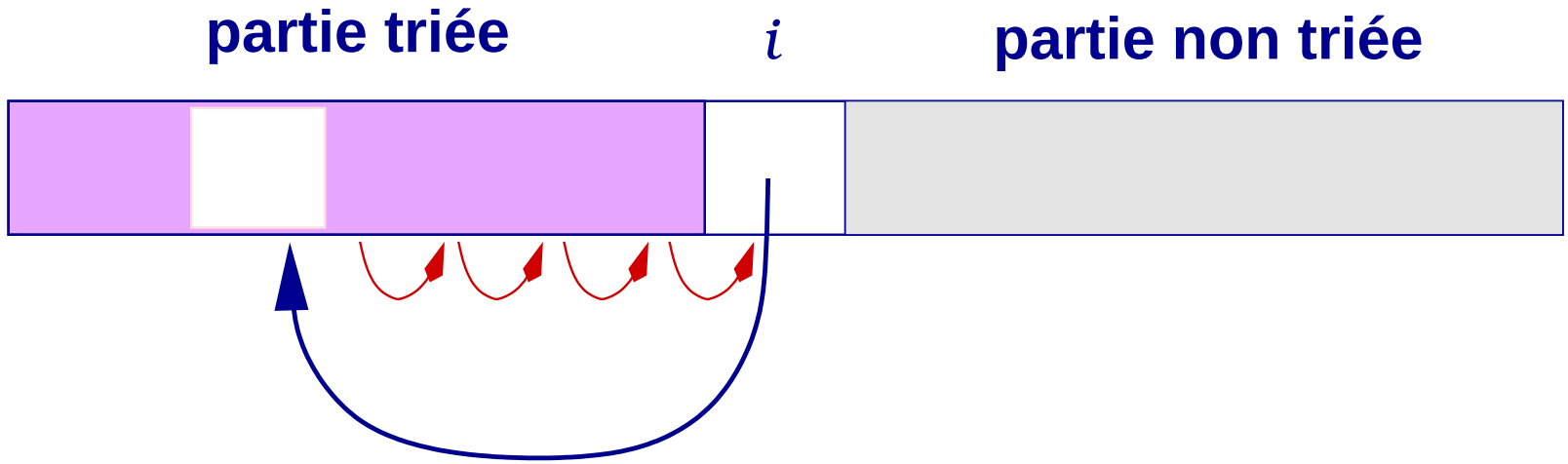
2. Tris en $O(n \times \log n)$.

- tri par fusion,
- tri par tas,
- tri rapide (mais en $O(n^2)$ dans le pire des cas).

3. Tris spéciaux.

- tri shell (probablement $O(n^{1.25})$),
- tri par dénombrement ($O(n)$).

Tri par insertion



Principe : insérer les éléments les uns après les autres dans la partie triée

Insertion : recopie des éléments plus grands vers la droite

Tri par insertion

procédure TRI_PAR_INSERTION($T[1, \ldots, n]$)
début
    pour $i := 2$ **jusqu'à** n **faire**
        $j := i$,
        tant que $((j > 1)$ **et** $(T[j] < T[j-1]))$ **faire**
            PERMUTER(T, j, $j-1$),
            $j := j - 1$,
        fin faire
    fin faire
fin procédure

Tri par insertion

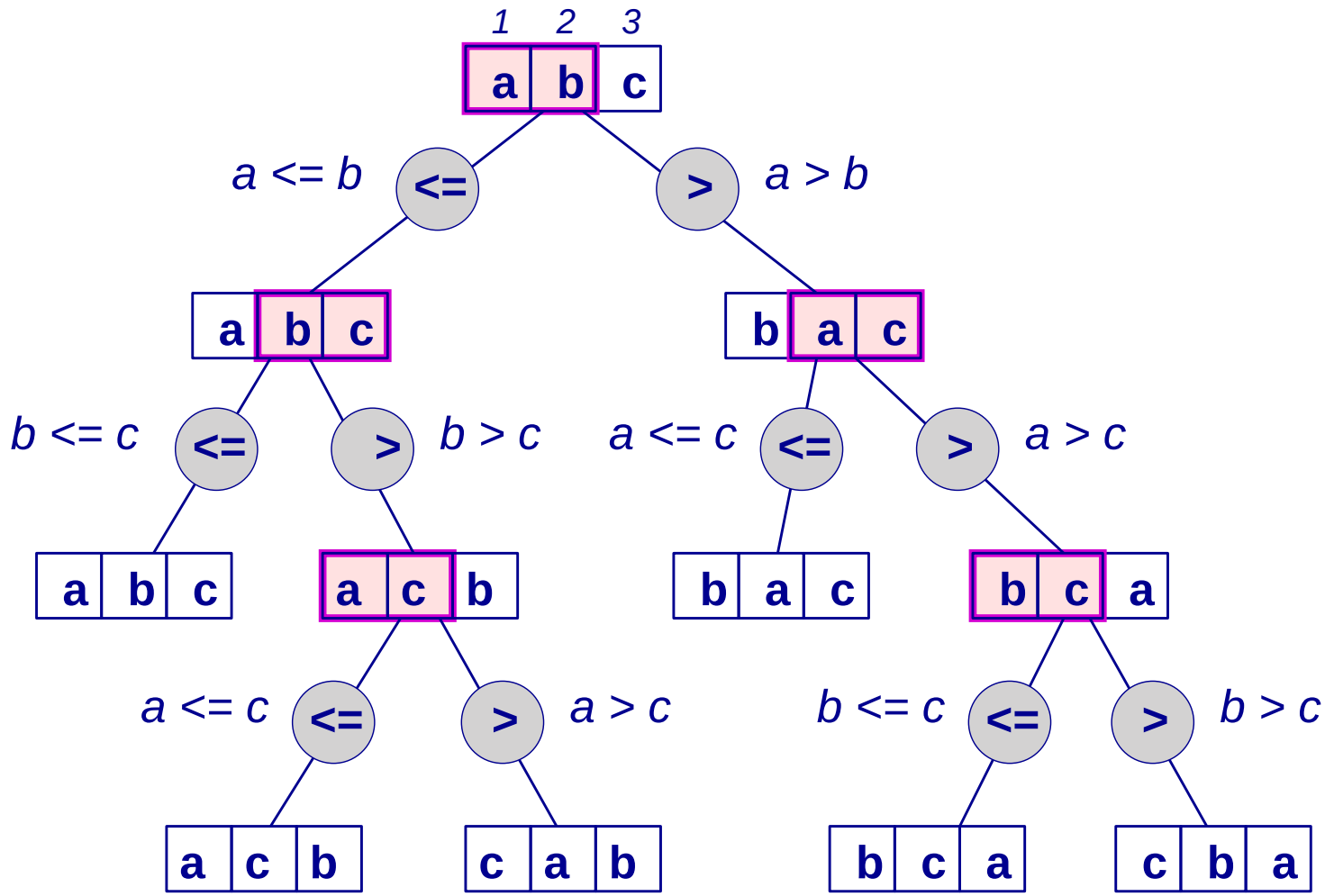
Pire des cas : $i - 1$ permutations à chaque passage.

$$T(n) = \sum_{i=2}^{n} i = O(n^2)$$

Meilleur des cas : aucune permutation (mais une comparaison).

$$T(n) = \sum_{i=2}^{n} 1 = O(n)$$

Arbre de décision du tri par insertion.



Complexité des tris par comparaison

Nombre de feuilles : $n!$ (le nombre de permutations de n éléments)

(une permutation est une bijection, si les éléments sont tous différents, deux suites distinctes produisent des résultats différents avec la même permutation)

Complexité des tris par comparaison

Nombre de feuilles : $n!$ (le nombre de permutations de n éléments)

une permutation est une bijection, si les éléments sont tous différents, deux suites distinctes produisent des résultats différents avec la même permutation

Complexité du tri dans le pire des cas = hauteur de l'arbre de décision

$$h \geq \log(n!) \geq \log\left(\frac{n}{e}\right)^n$$

(formule de Stirling) or

$$\log\left(\frac{n}{e}\right)^n = n \times \log n - n \times \log e$$

et la hauteur de l'arbre

$$h = O(n \times \log n)$$

Tas (files de priorité)

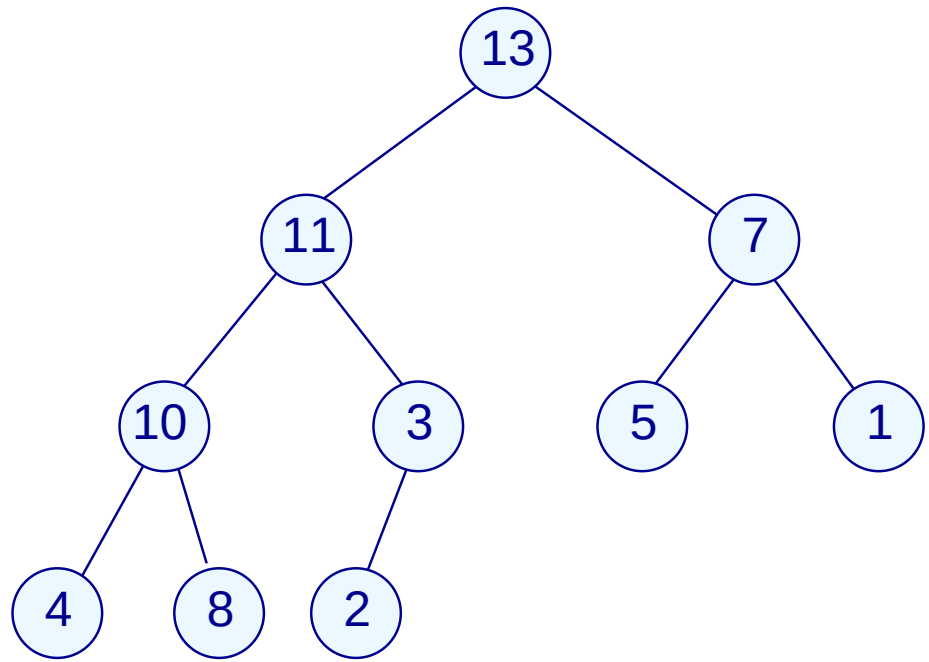
Arbre binaire représenté dans un tableau.

Opérations :

- *insertion*,
- *extraction du max* (resp. *extraction du min*),
- *augmentation de la clé* (resp. *diminution de la clé*),
- *construction d'un tas.*

Tas (files de priorité)

Tas : arbre binaire, chaque noeud a une valeur supérieure (resp. inférieure) aux valeurs figurant dans ses sous-arbres



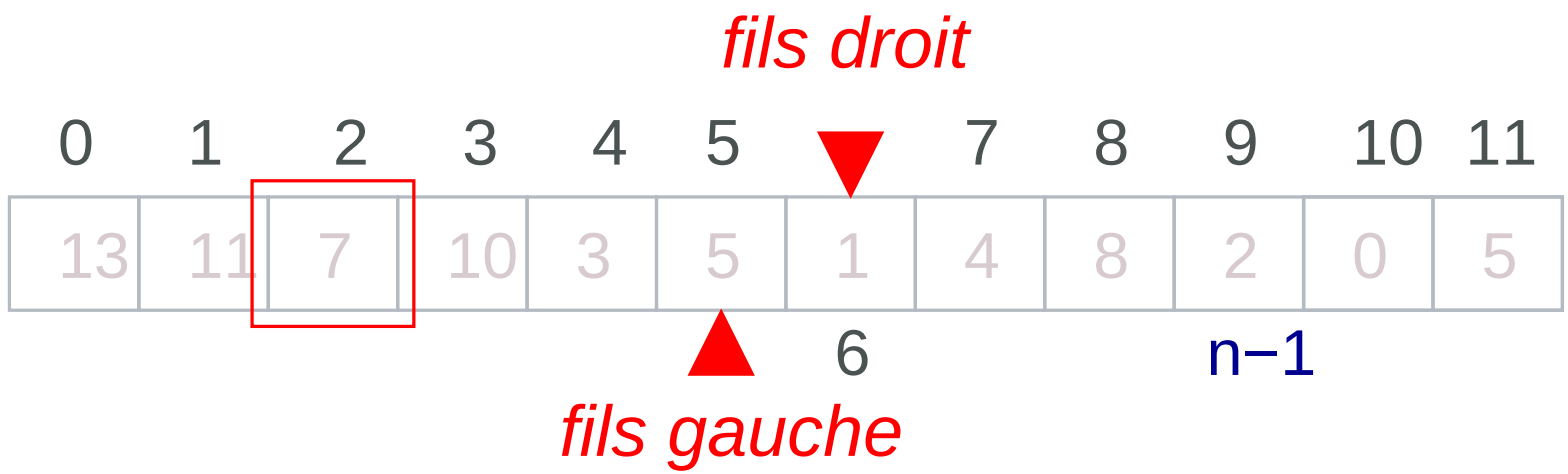
Tas : représentation

Tableau indexé : les n éléments du tas sont $T[0]$, $T[1]$,. . ., $T[n-1]$.

Convention :

- indice du fils gauche de i : $2 \times i + 1$,
- indice du fils droit de i : $2 \times i + 2$.

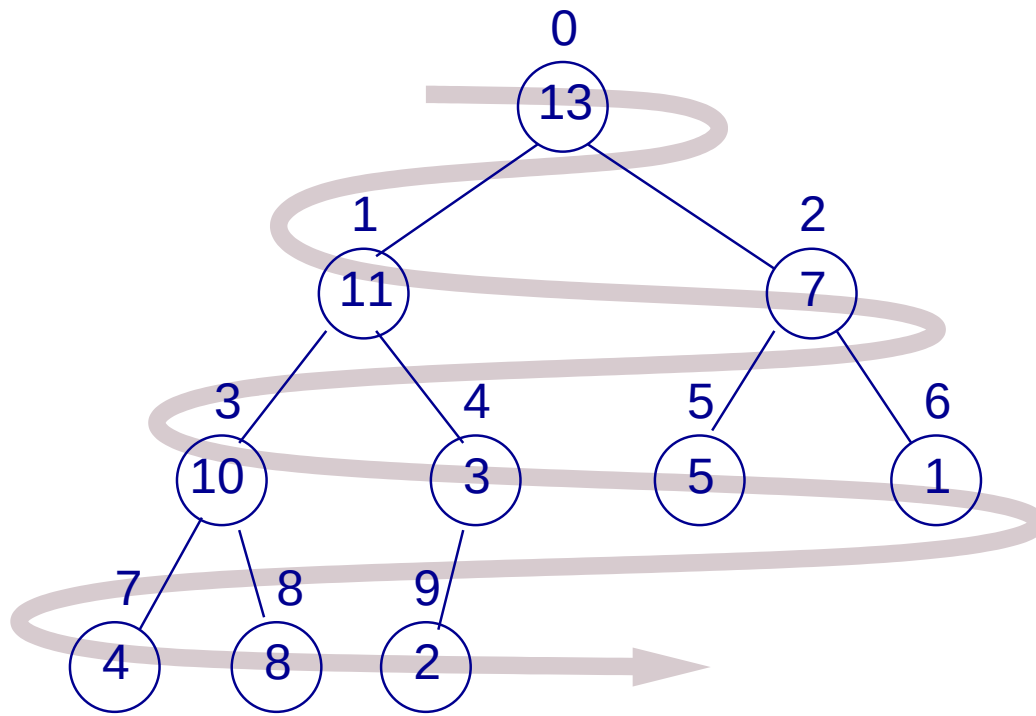
L'arbre est *naturellement* équilibré (toutes les feuilles ont la même profondeur à un niveau près)



Tas (files de priorité)

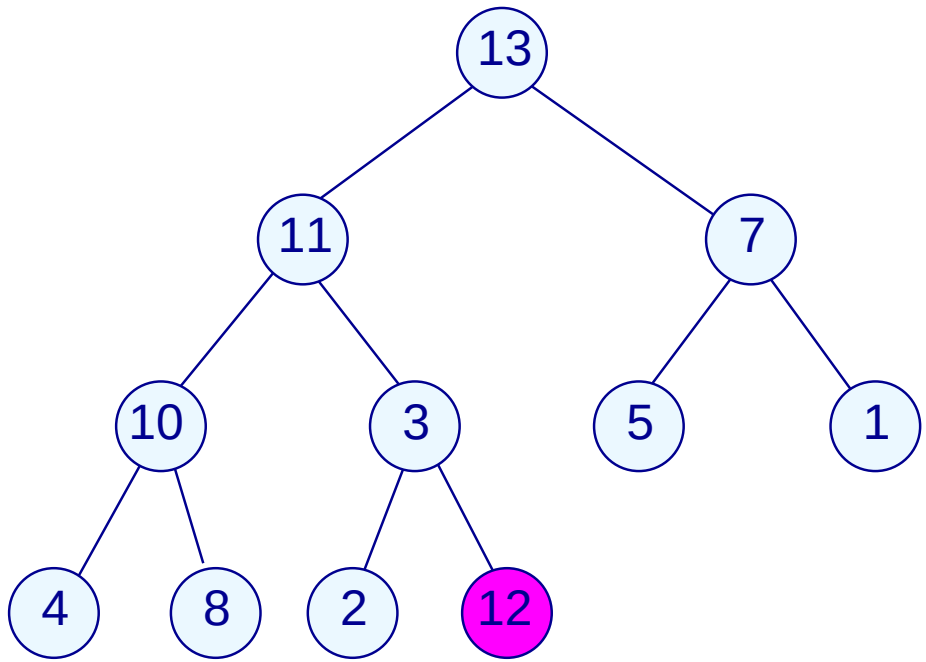
Tas : le parcours des éléments du tableau correspond au parcours de l'arbre *par niveaux* (parcours en largeur).

hauteur $\log_2 n$



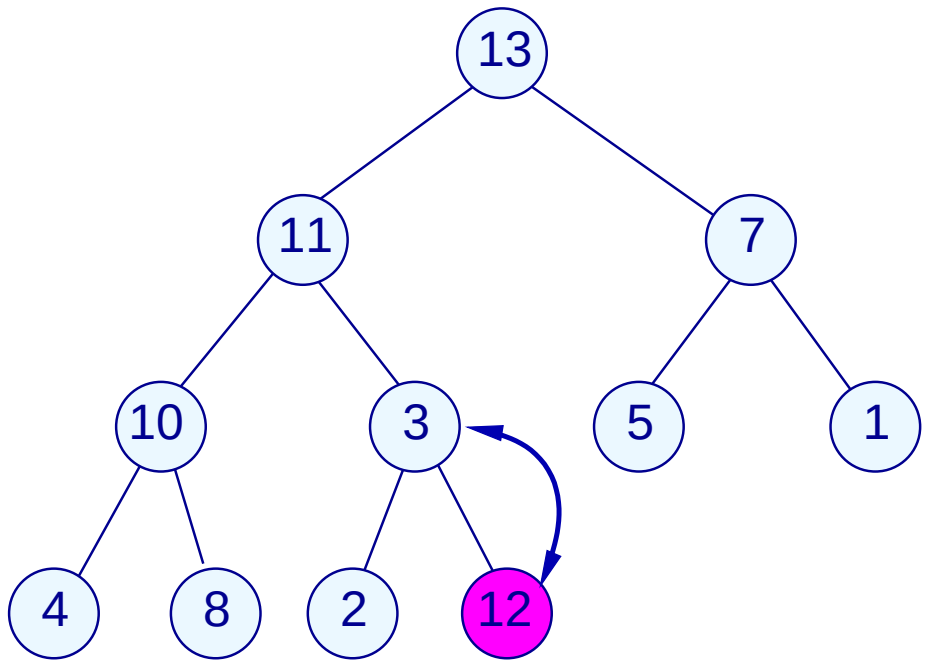
Tas : insérer un élément

Placer le nouvel élément à la fin du tableau.



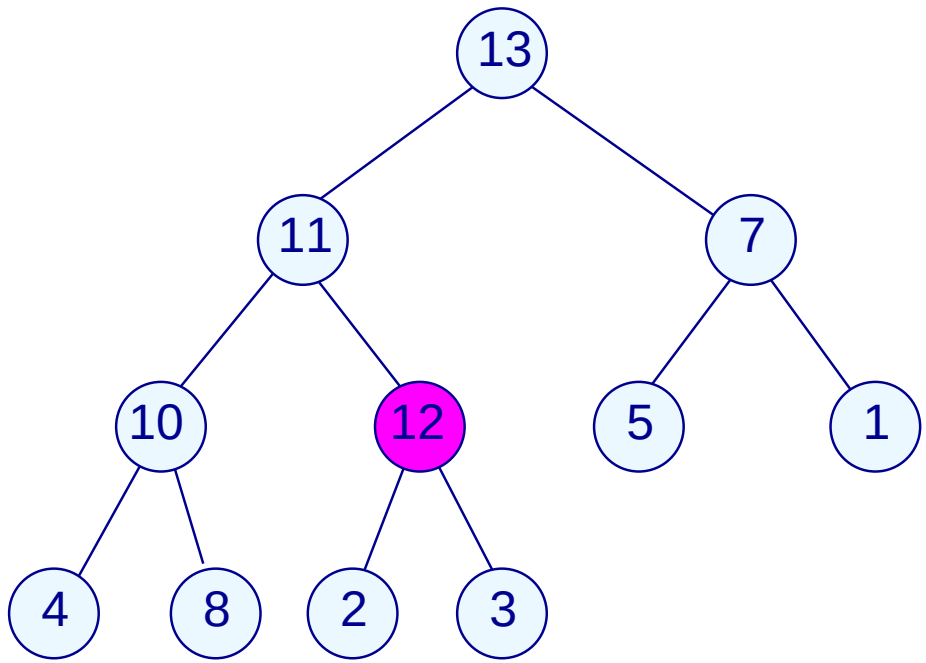
Tas : insérer un élément

Comparer le nouvel élément avec son père.



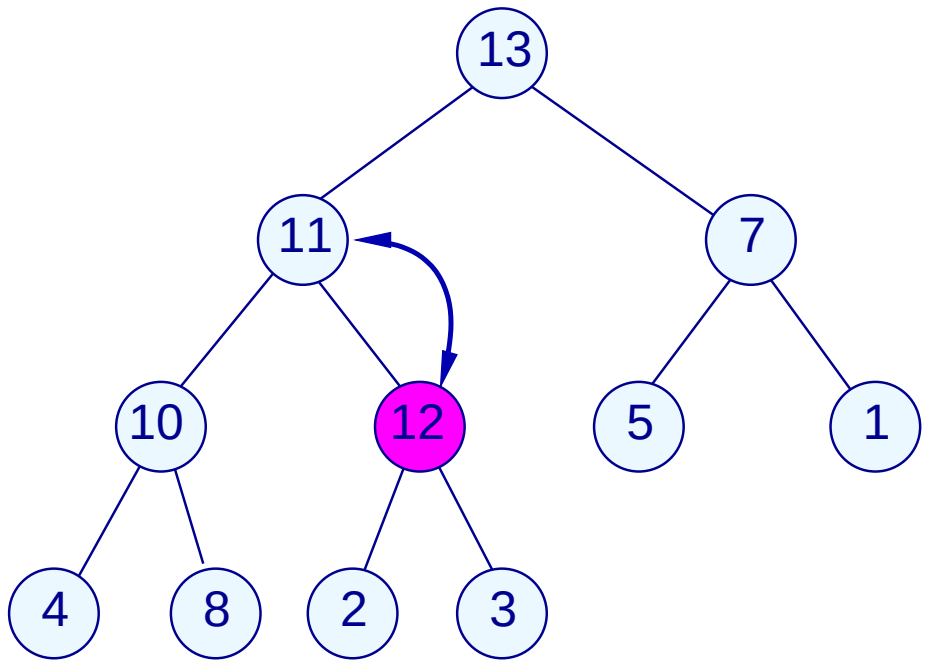
Tas : insérer un élément

Permuter avec le père si la propriété d'infériorité n'est pas satisfaite.



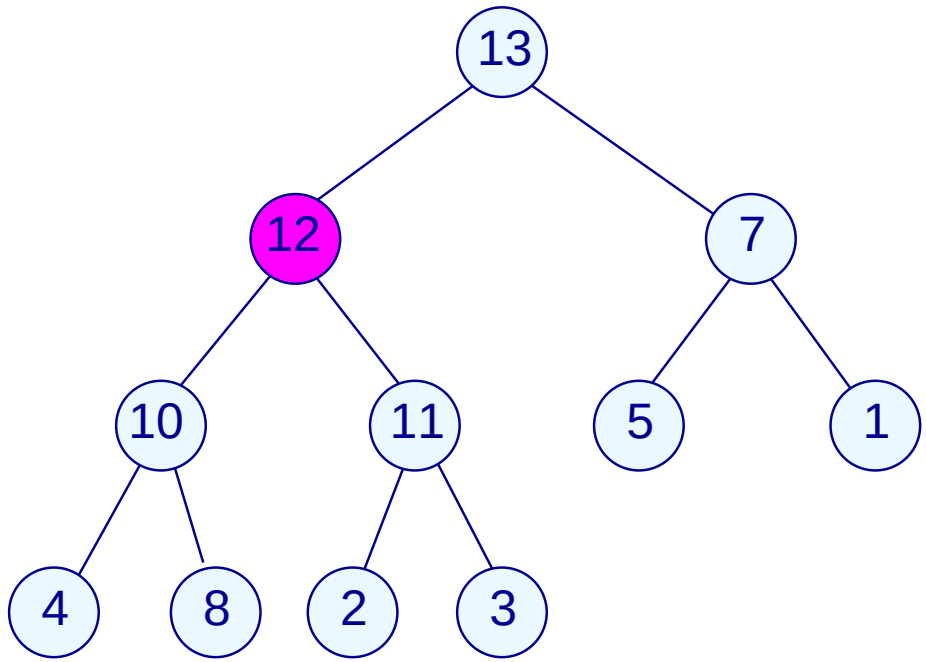
Tas : insérer un élément

Comparer avec le père.



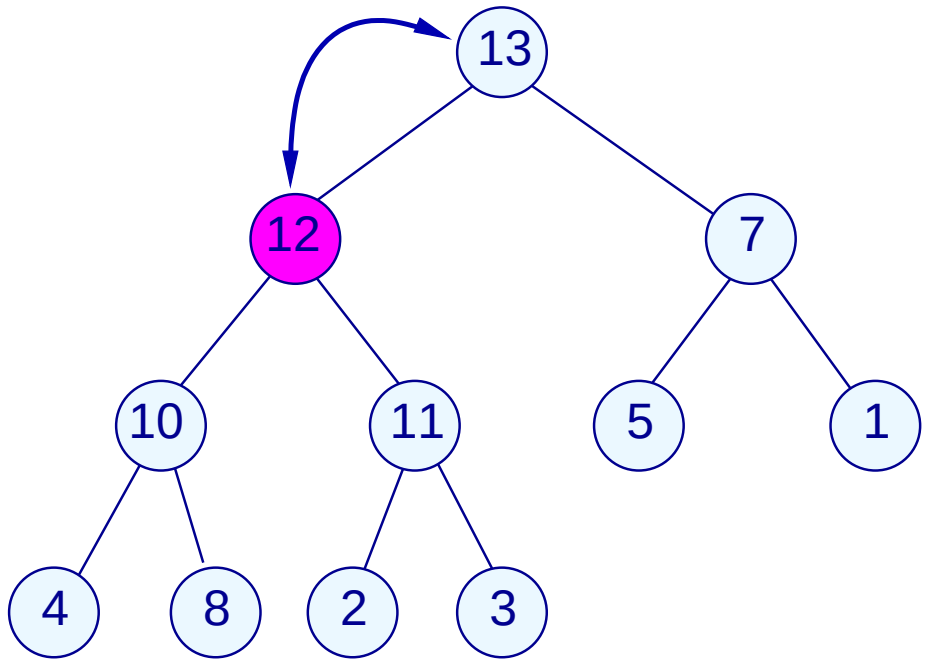
Tas : insérer un élément

Permuter.



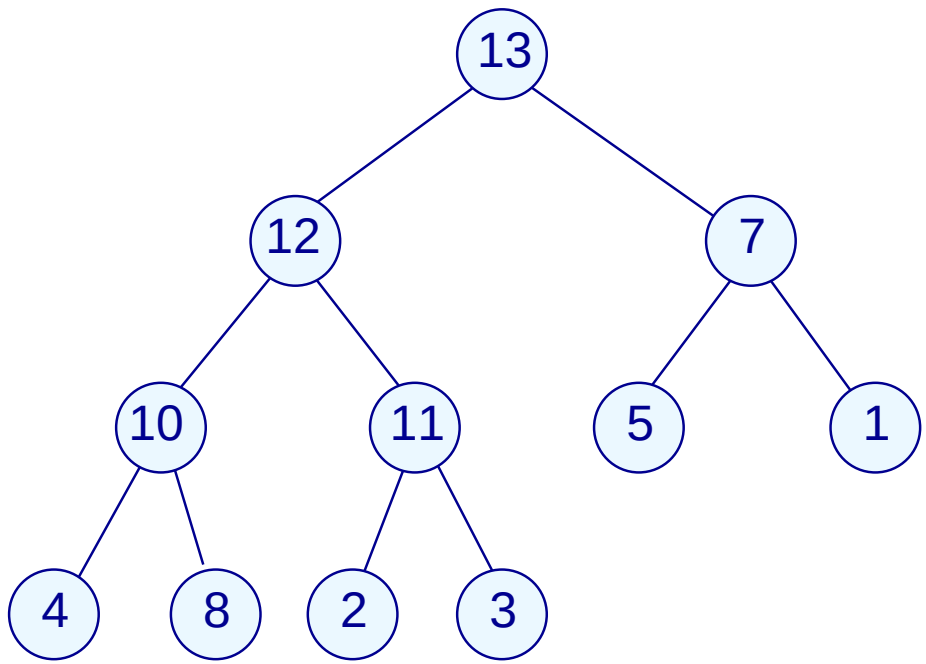
Tas : insérer un élément

Le processus prends fin dès que le père a une clé supérieure.



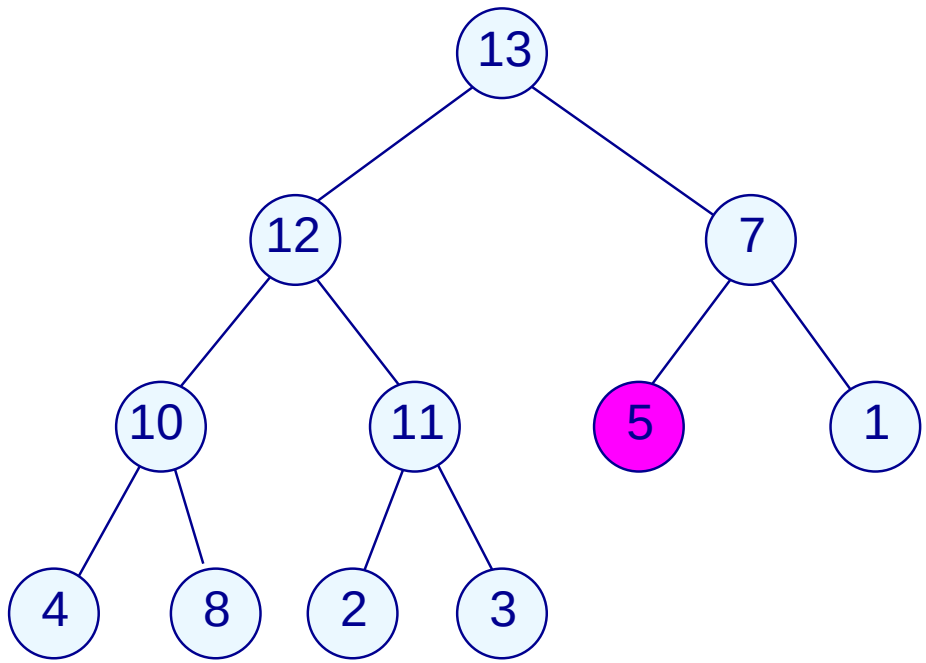
Tas : insérer un élément

Coût : $O(\log n)$, le parcours d'une branche.



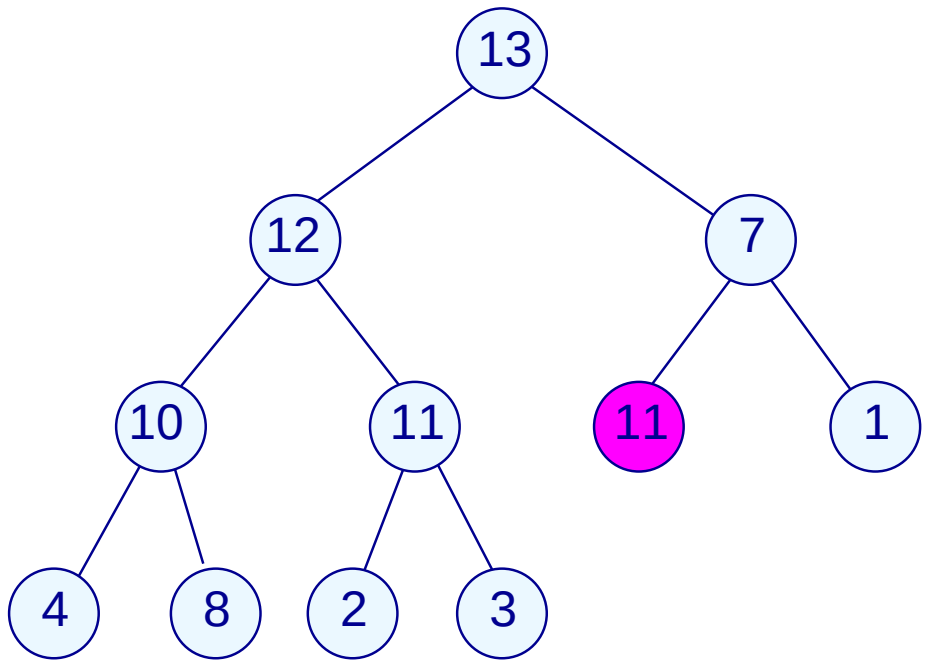
Tas : augmenter la clé

Principe : faire remonter l'élément comme pour l'insertion.



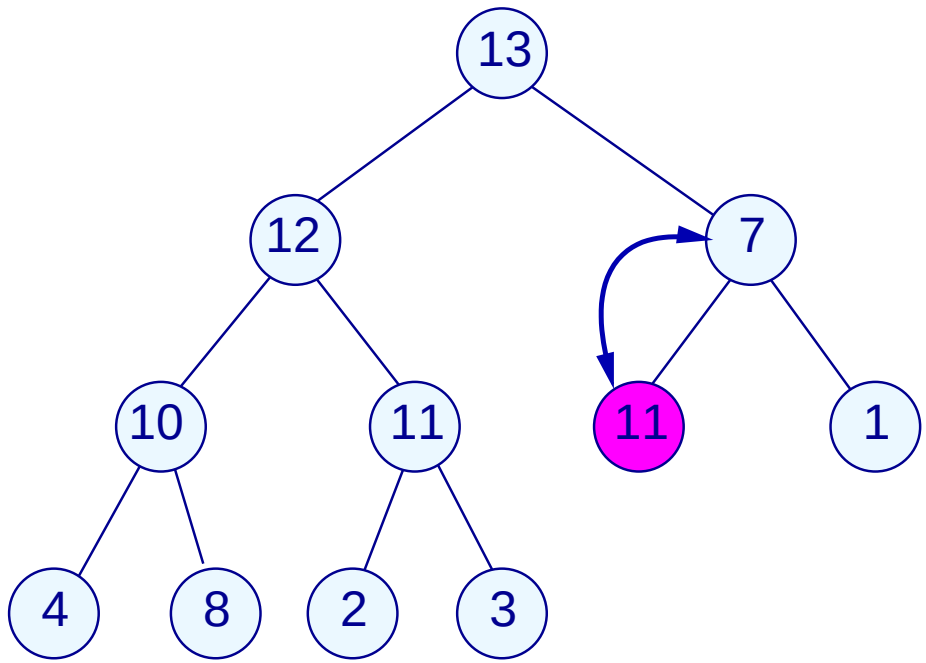
Tas : augmenter la clé

Principe : faire remonter l'élément comme pour l'insertion.



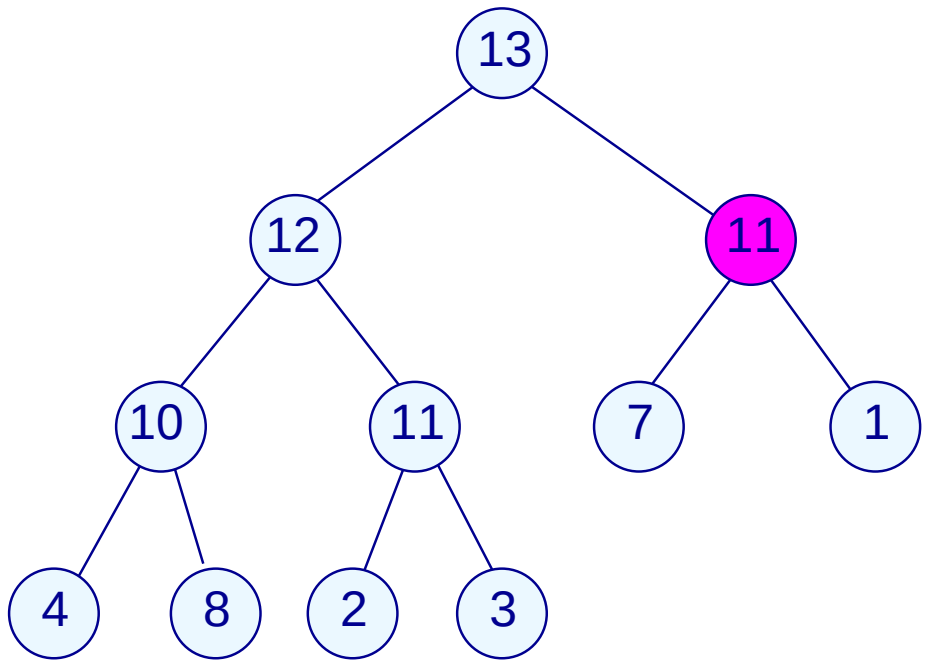
Tas : augmenter la clé

Principe : faire remonter l'élément comme pour l'insertion.



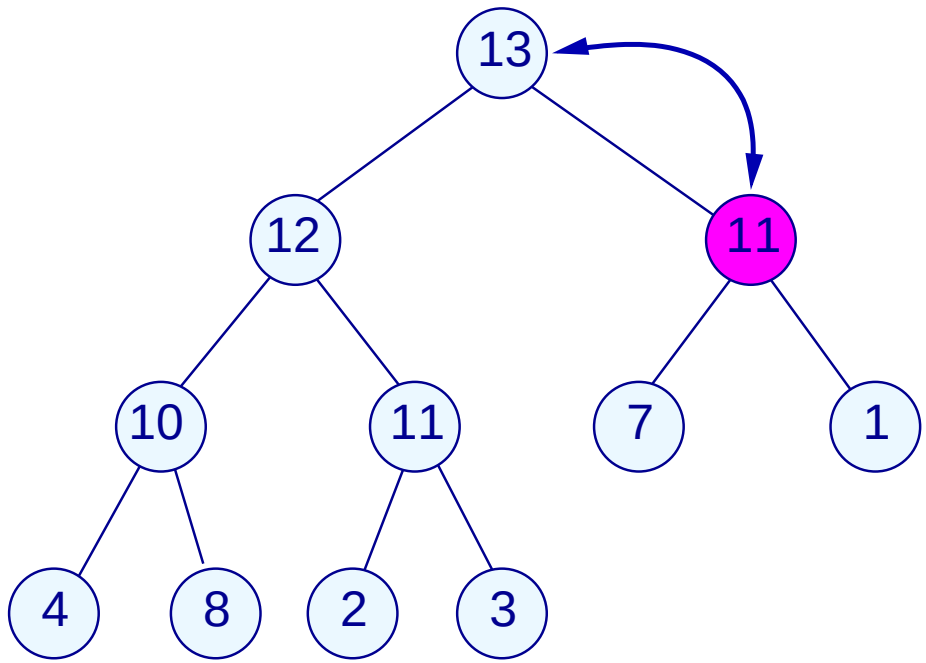
Tas : augmenter la clé

Principe : faire remonter l'élément comme pour l'insertion.



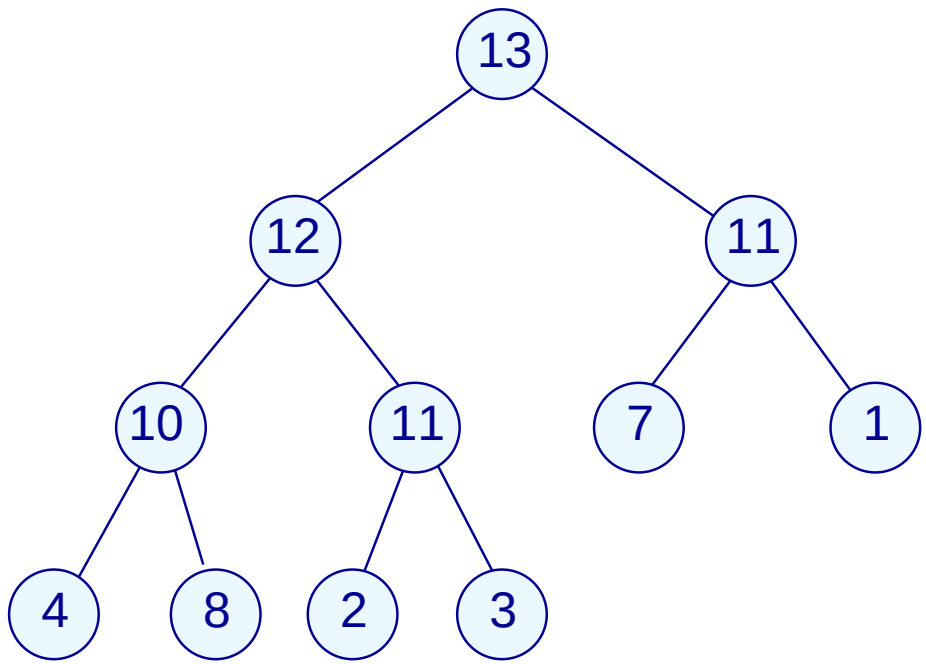
Tas : augmenter la clé

Principe : faire remonter l'élément comme pour l'insertion.



Tas : augmenter la clé

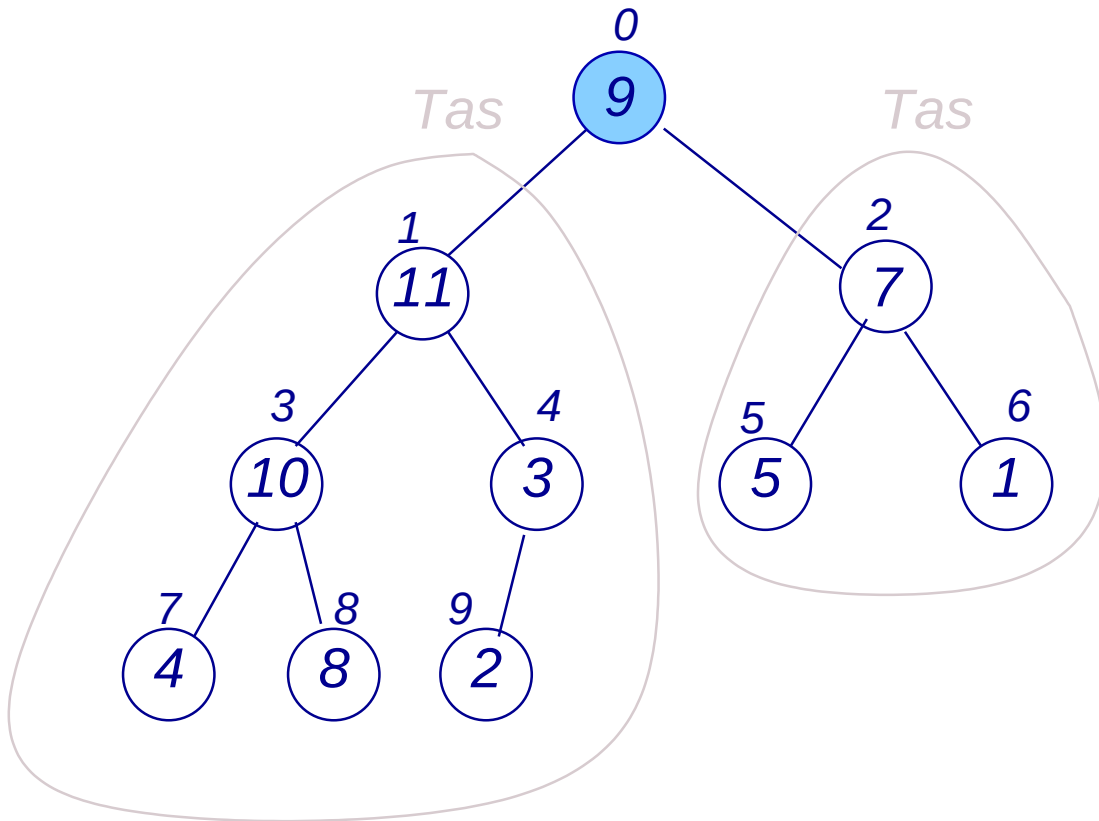
Coût : $O(\log n)$, le parcours d'une branche.



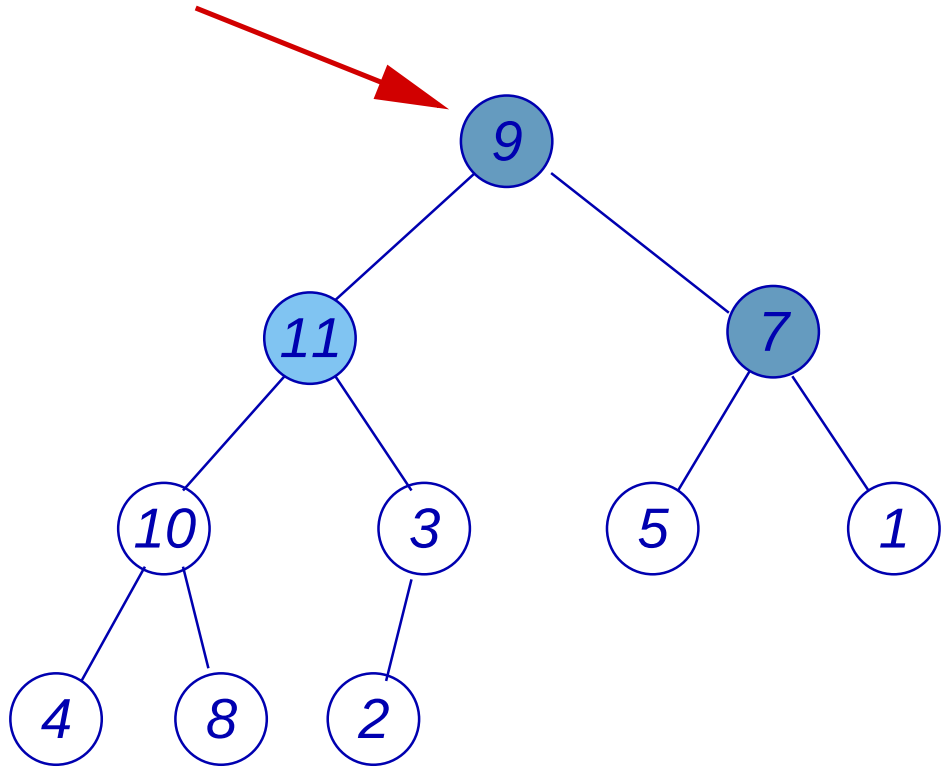
Tas : extraire le max

- supprimer le *dernier* noeud de l'arbre et recopier sa valeur à la racine,
- *entasser à la racine*.

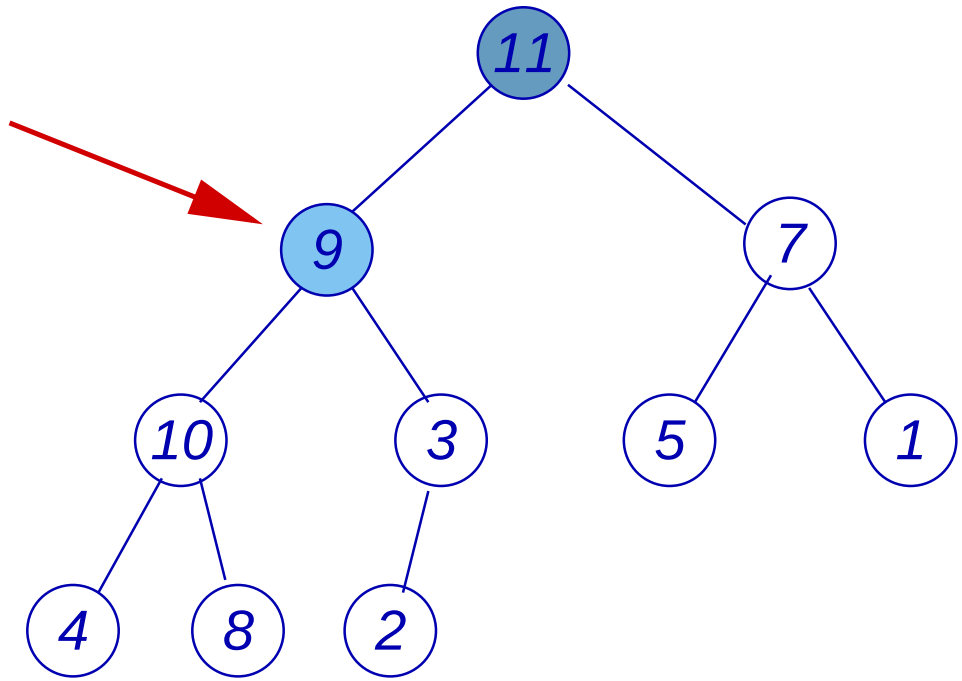
Tas : entasser.



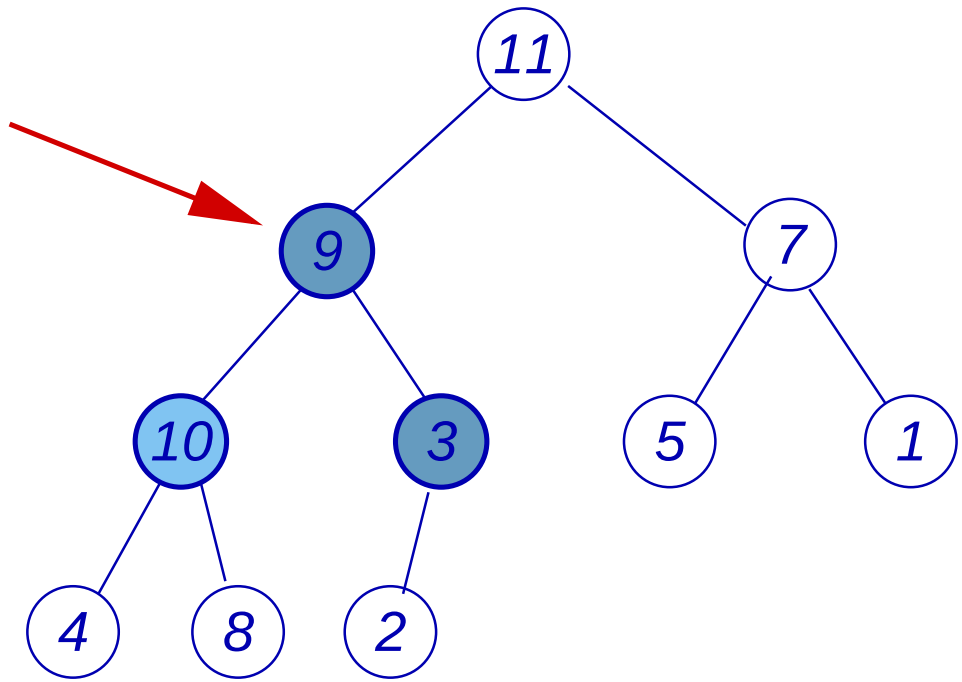
Tas : entasser.



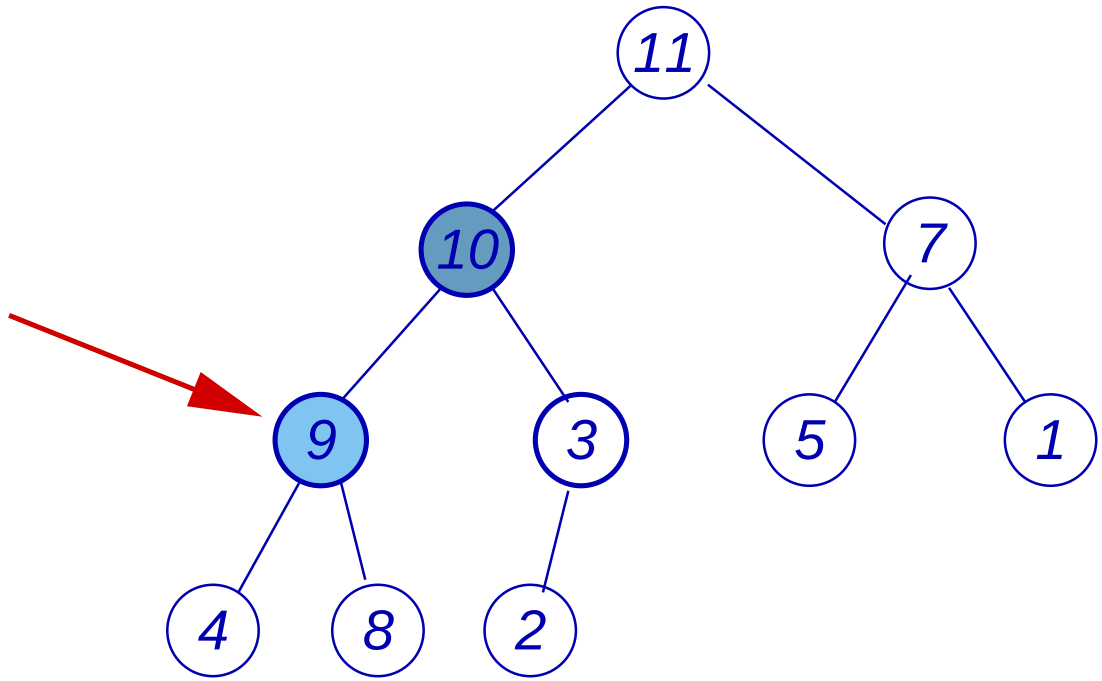
Tas : entasser.



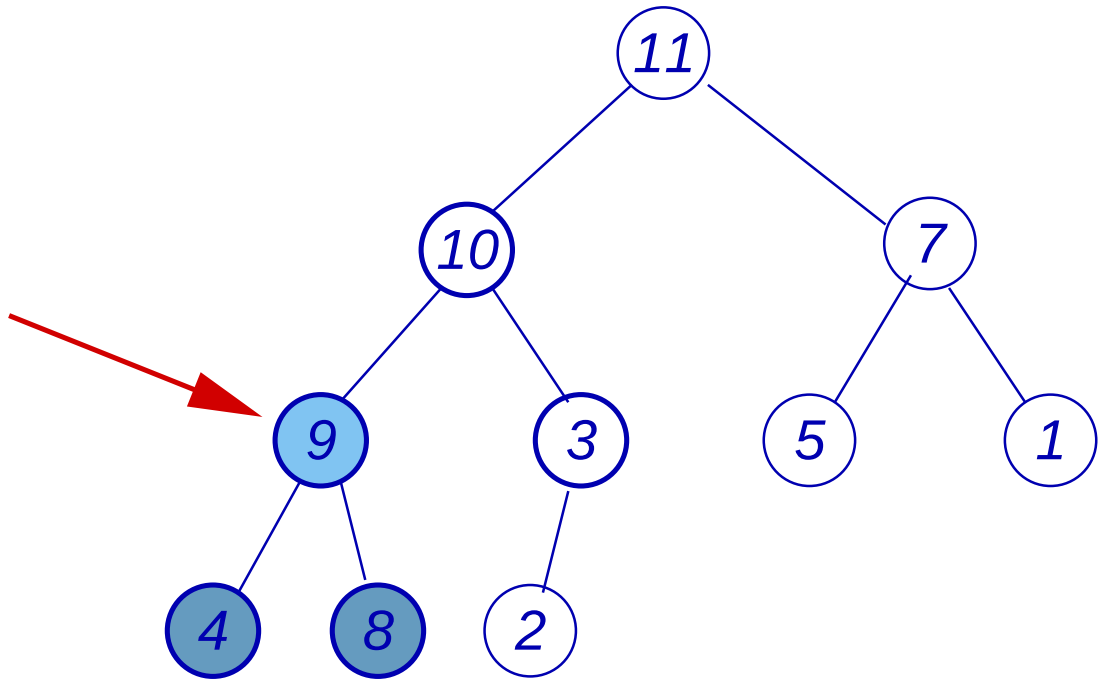
Tas : entasser.



Tas : entasser.



Tas : entasser.



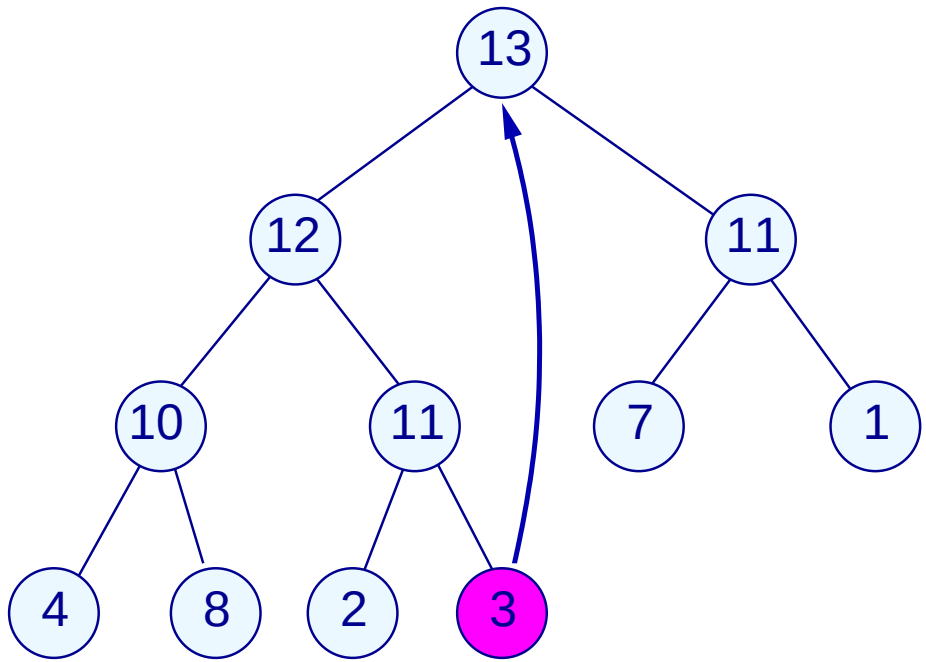
Tas : entasser.

procédure ENTASSER(i, T, n)
{FILSG(i) *et* FILSD(i) *sont des tas,* i *peut-être pas*}
début
    $iMax := i$,
    si (FILSG$(i) < n$) et $(T[$FILSG$(i)] > T[iMax])$ **alors**
        $iMax :=$FILSG(i),
    si (FILSD$(i) < n$) et $(T[$FILSD$(i)] > T[iMax])$ **alors**
        $iMax :=$ FILSD(i),
    si $(iMax \neq i)$ **alors**
        ECHANGER$(T, i, iMax)$,
        ENTASSER$(iMax, T, n)$,
    fin si
fin procédure

Pire des cas : parcours de la plus longue branche $O(\log_2 n)$.

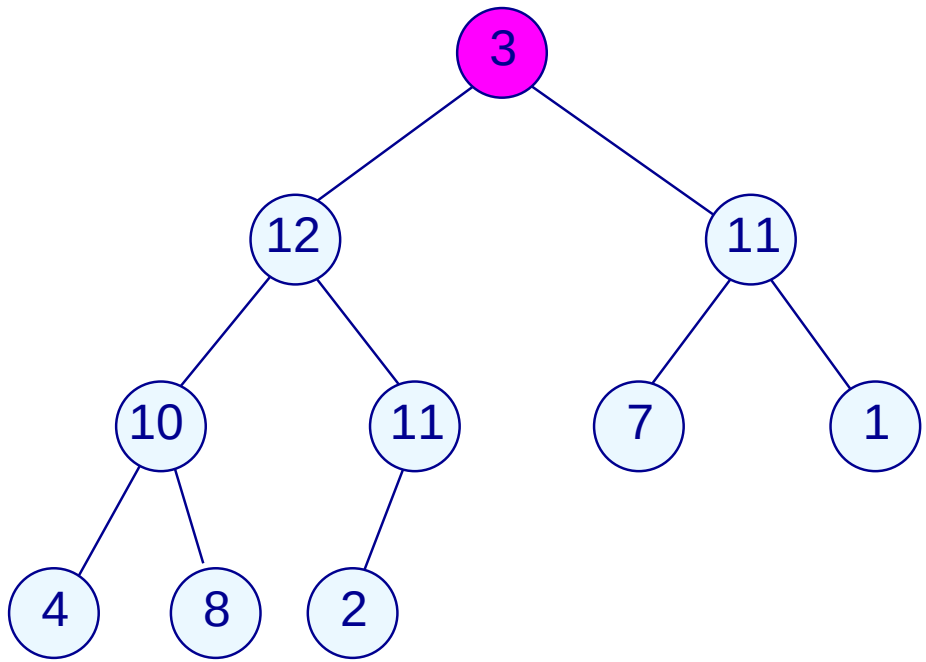
Tas : extraire le max

Supprimer la dernière valeur et la mettre à la racine :



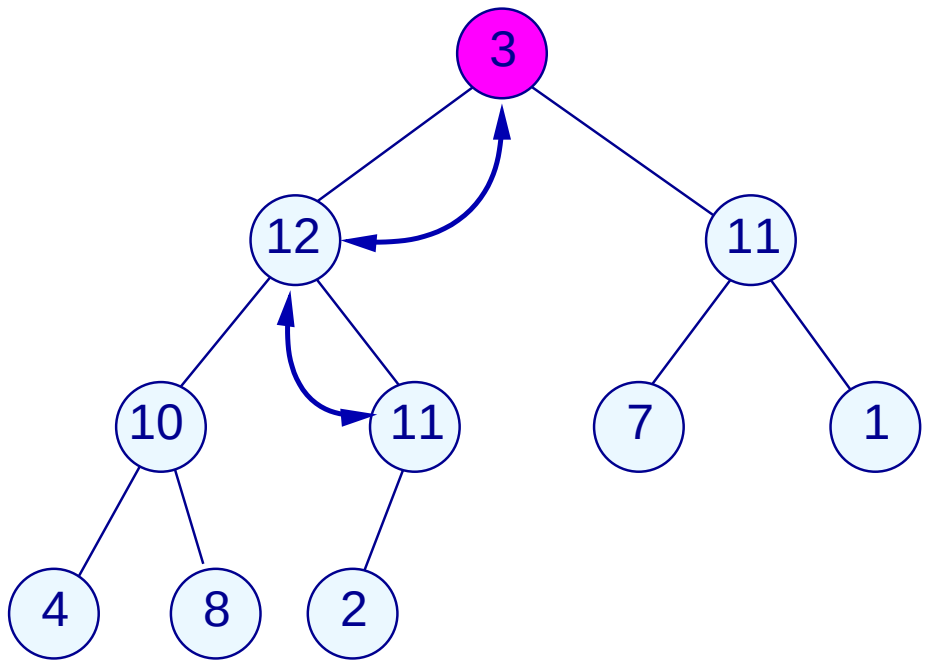
Tas : extraire le max

Supprimer la dernière valeur et la mettre à la racine :



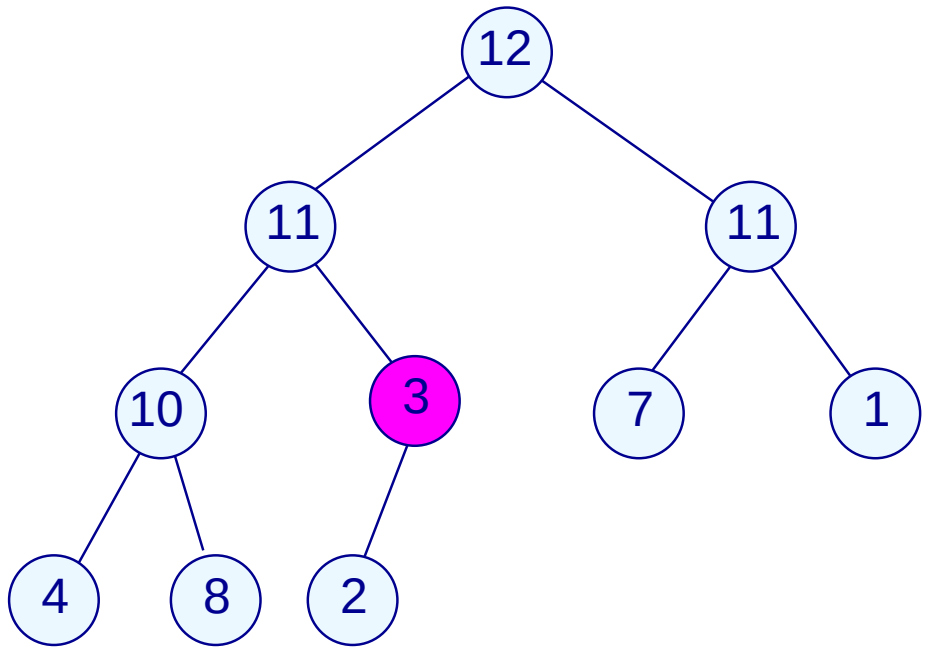
Tas : extraire le max

Entasser à la racine :



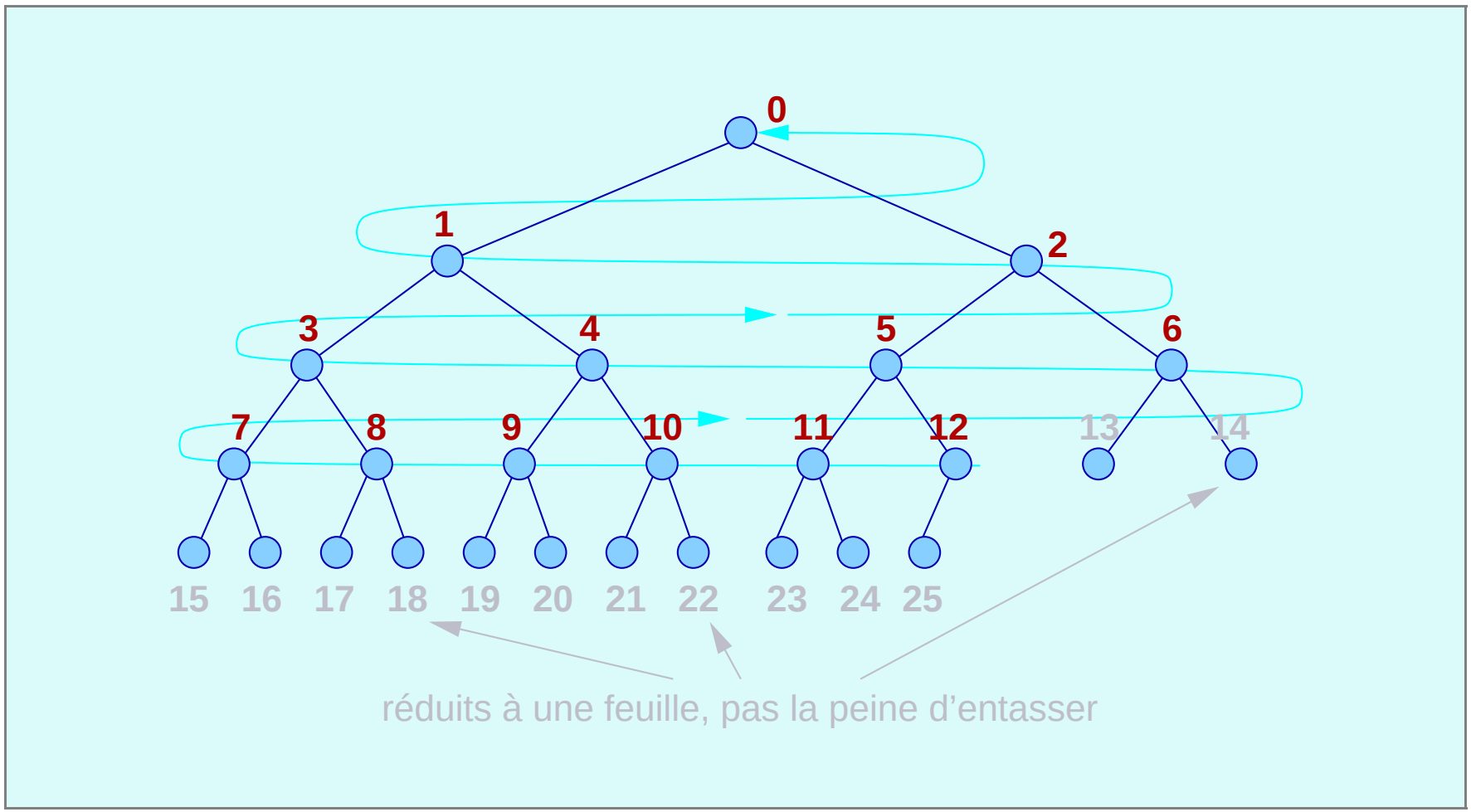
Tas : extraire le max

Coût : $O(\log n)$.



Tas : construction.

Principe : transformer en tas du bas vers le haut en entassant



Tas : construction.

Base : les $n/2$ derniers éléments du tableau sont des tas (feuilles)

Itération : $\mathrm{FilsG}(i)$ et $\mathrm{FilsD}(i)$ sont des tas, après Entasser(i, T, n), i est un tas

fonction CONSTRUIRE_TAS(T, n)
    pour $i := n/2 - 1$ **jusqu'à** 0 **faire**
        ENTASSER(i, T, n),
fin fonction

Complexité : $O(n \log n)$ de façon évidente. En fait $O(n)$.

Tas : construction.

Preuve construction du tas en $O(n)$

- à la hauteur $h = 0$ au plus $\lceil \frac{n}{2^1} \rceil$ noeuds
- à la hauteur $h = 1$ au plus $\lceil \frac{n}{2^2} \rceil$ noeuds
- à la hauteur h au plus $\lceil \frac{n}{2^{h+1}} \rceil$ noeuds

Pour un noeud à la hauteur h on entasse en au plus h échanges

Tas : construction.

Preuve construction du tas en $O(n)$

Le cumul est

$$\sum_{h=1}^{\log\ n} \lceil \frac{n}{2^{h+1}} \rceil \times h \leq n \times \sum_{h=1}^{\log\ n} \frac{h}{2^h}$$

Puisque

$$\sum_{i=0}^{\infty} i \times x^i = \frac{x}{(1-x)^2}$$

on a

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2$$

Tas : construction.

Preuve construction du tas en $O(n)$

et donc le nombre d'échanges est borné par

$$n \times \sum_{h=0}^{\log\ n} \frac{h}{2^h} \leq 2 \times n = O(n)$$

Tri par tas.

Principe : extraire successivement tous les éléments du tas.

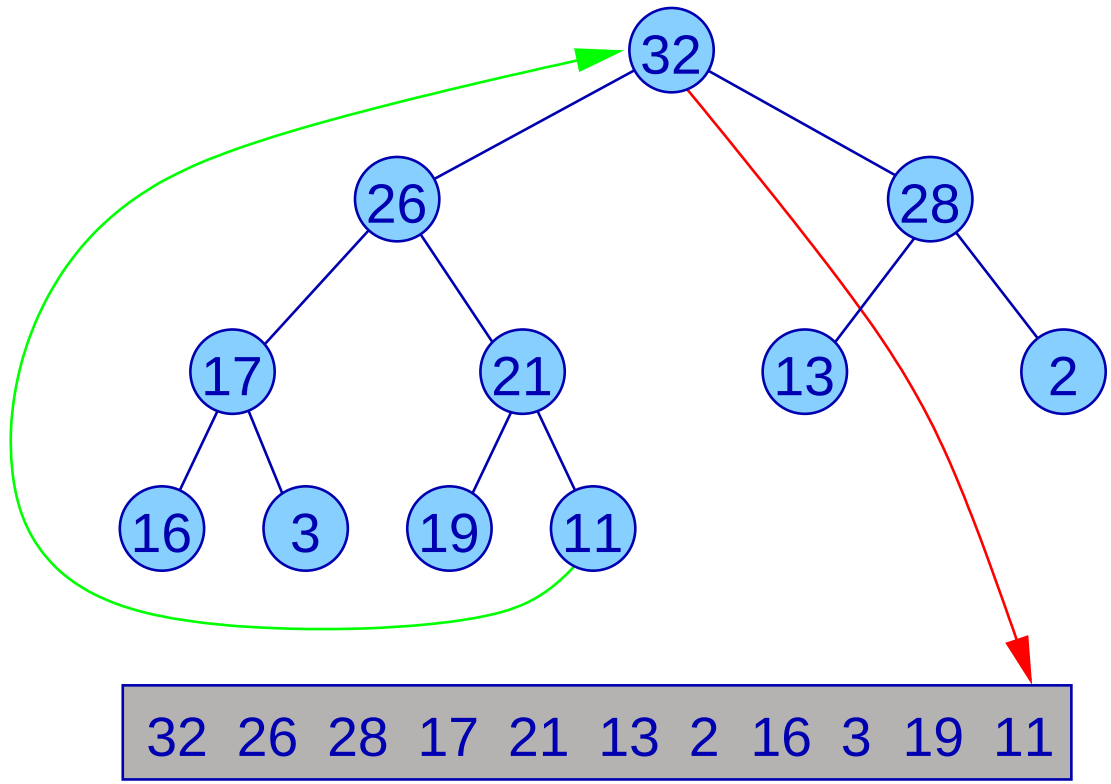
Initialisation :

- construire un tas avec les valeurs de la suite.

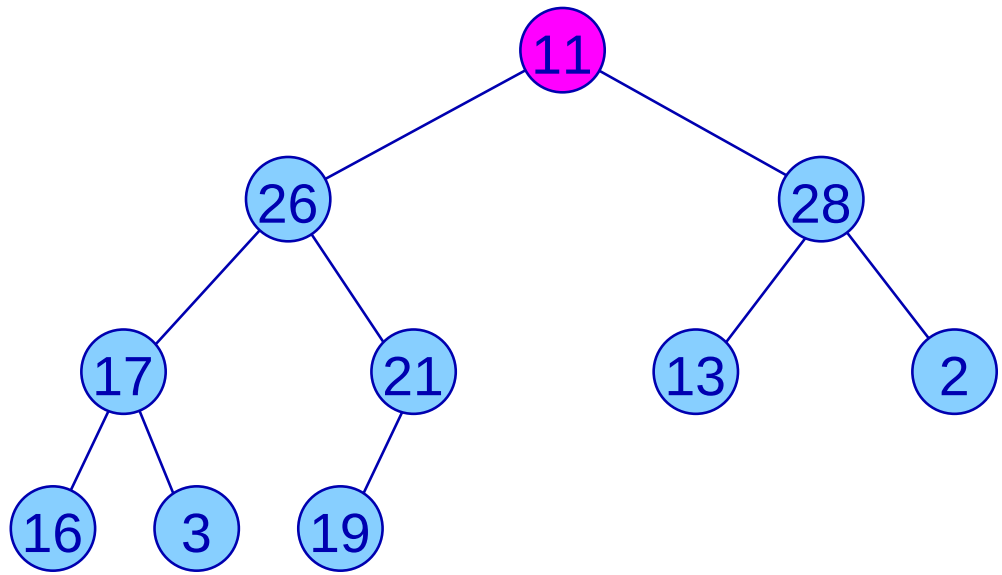
Itération :

- extraire le max et le placer dans le tableau après les éléments du tas.

Tri par tas.

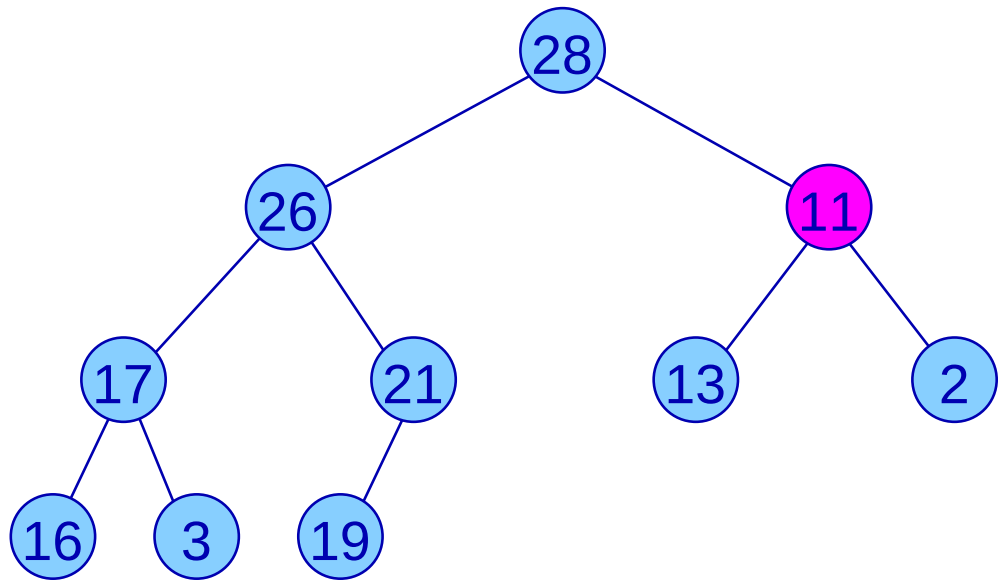


Tri par tas.



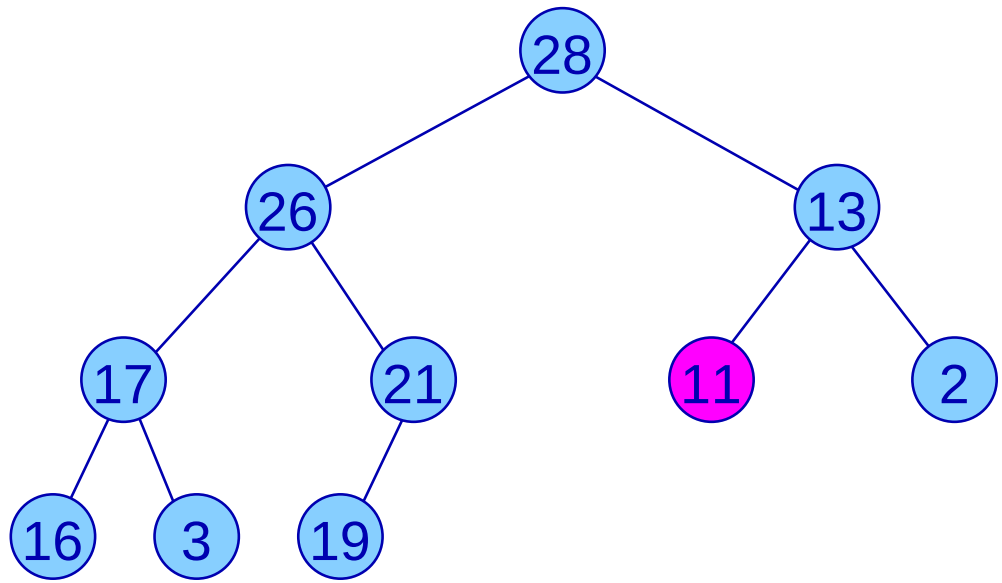
11	26	28	17	21	13	2	16	3	19	32

Tri par tas.



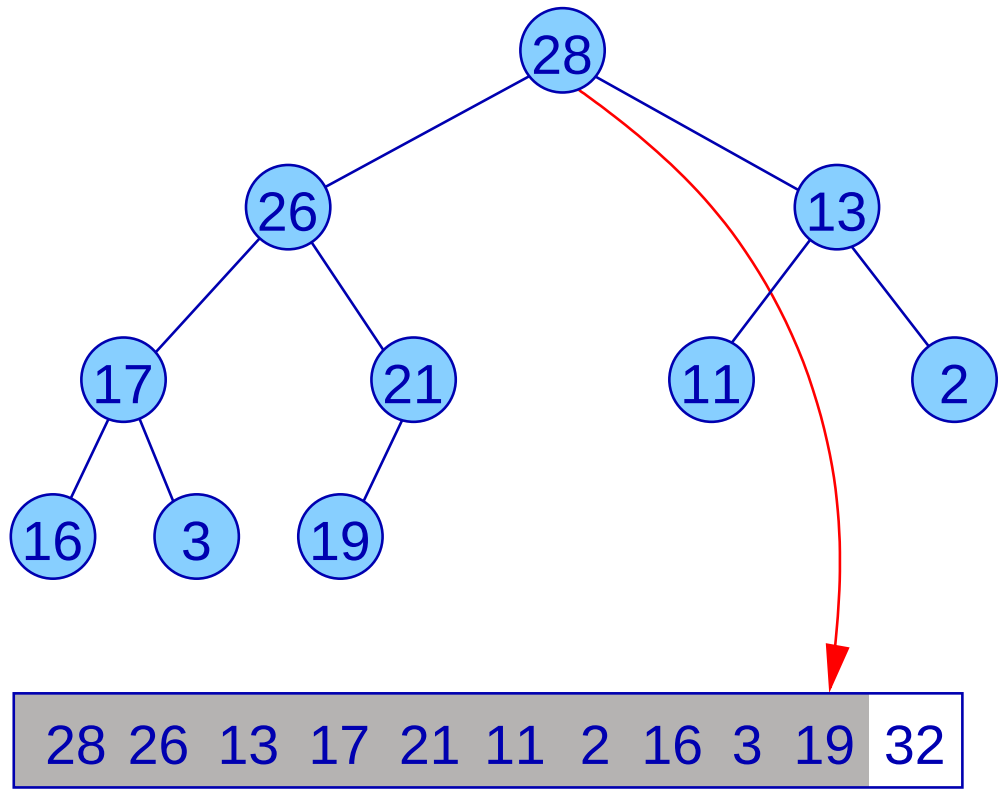
28	26	11	17	21	13	2	16	3	19	32

Tri par tas.

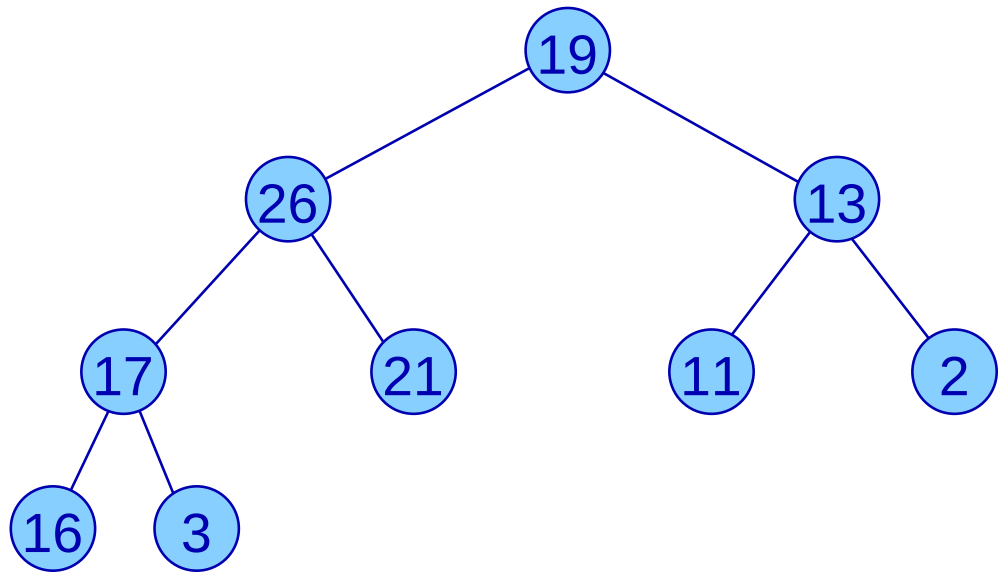


28	26	13	17	21	11	2	16	3	19	32

Tri par tas.

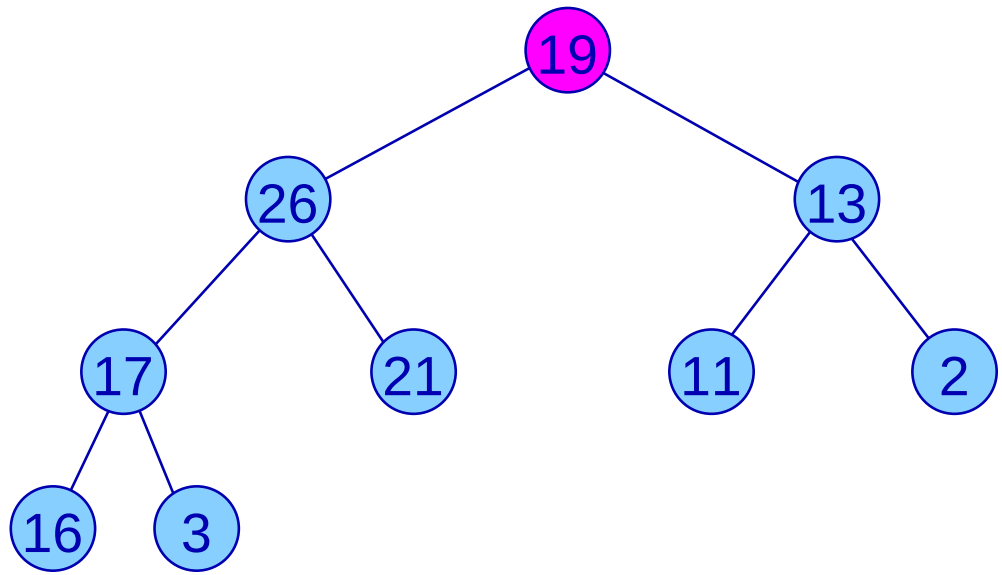


Tri par tas.



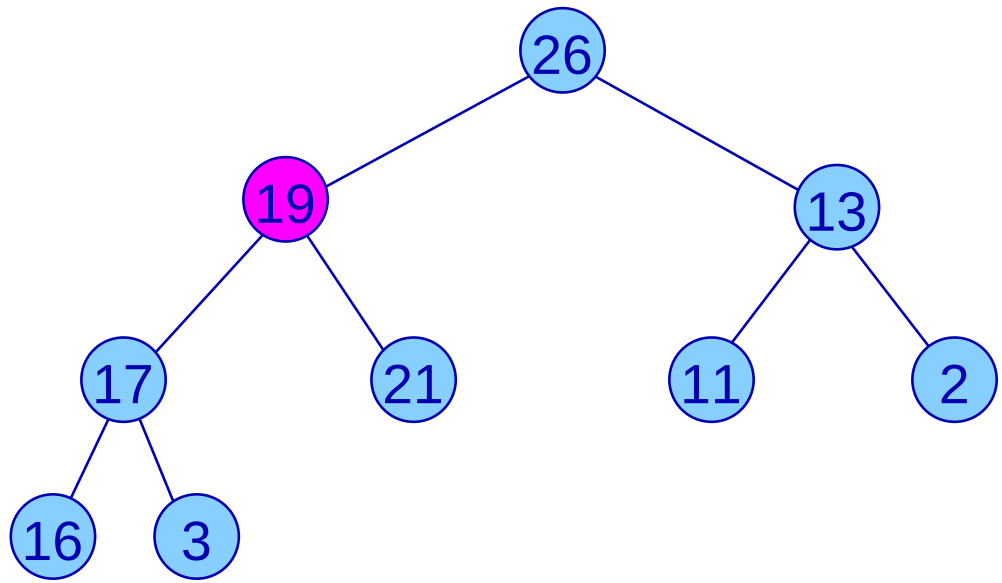
19	26	13	17	21	11	2	16	3	28	32

Tri par tas.



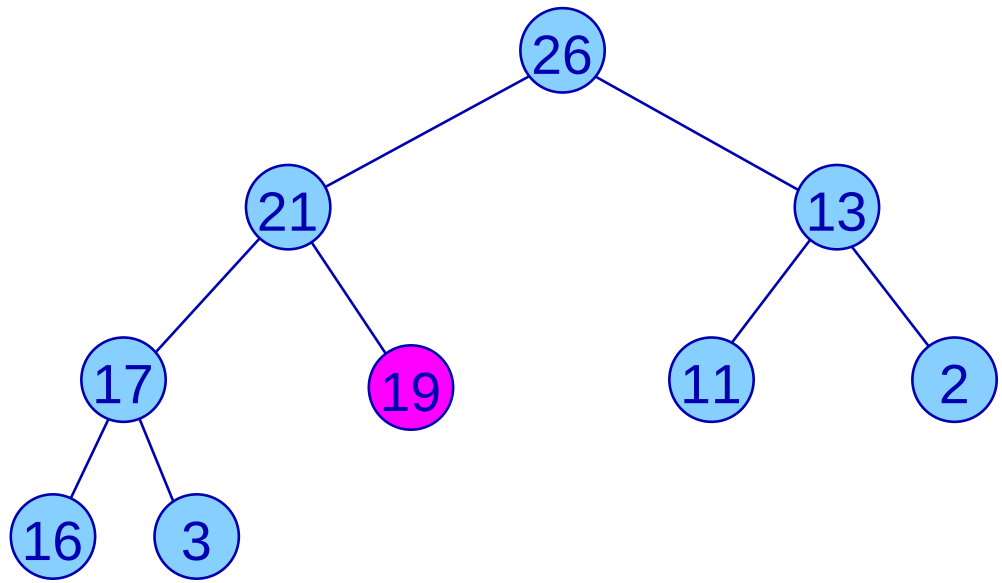
19	26	13	17	21	11	2	16	3	28	32

Tri par tas.



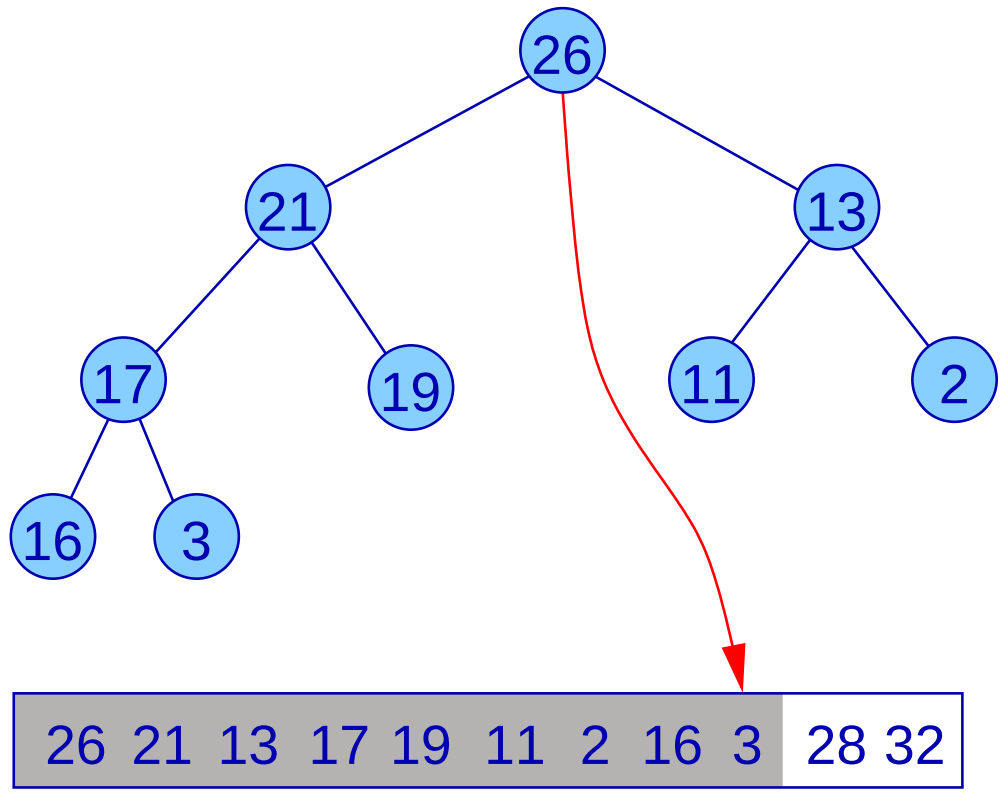
26	19	13	17	21	11	2	16	3	28	32

Tri par tas.

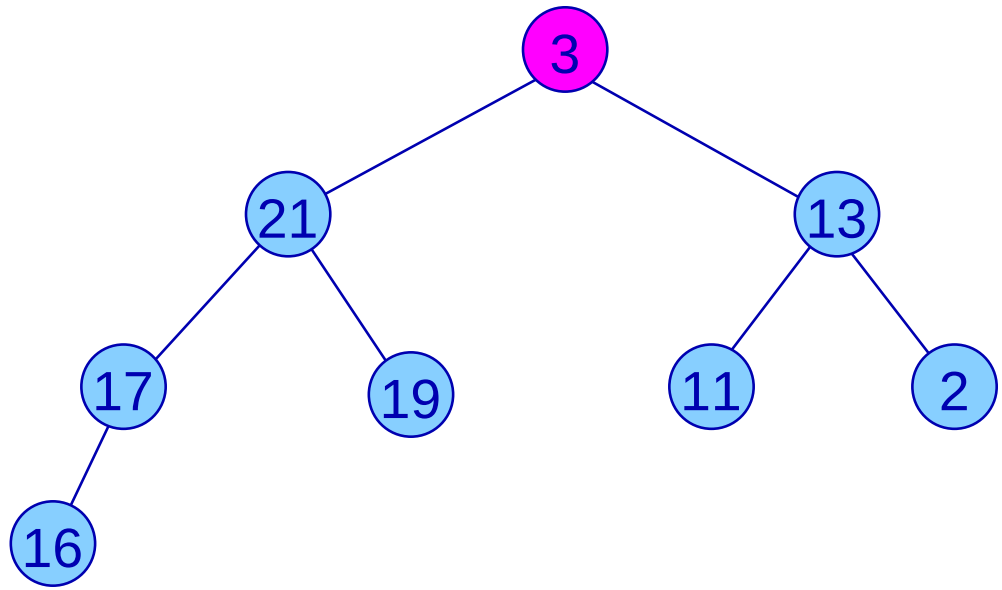


26	21	13	17	19	11	2	16	3	28	32

Tri par tas.



Tri par tas.



3	21	13	17	19	11	2	16	26	28	32

Tri par tas.

```
fonction TRI_PAR_TAS(T, n)
        CONSTRUIRE_TAS(T, n),
        pour i := n − 1 jusqu'à 1 faire
                ECHANGER(T, 0, i),
                ENTASSER(0, T, i),
        fin faire
fin fonction
```

ConstruireUnTas(T, n) est en $O(n)$.
Ensuite on fait $n - 1$ fois Entasser$(0, T, i)$, i.e. $O(n\log n)$.
La complexité du tri est donc $O(n\log n)$.

Tri par dénombrement.

- Tri sans comparaison
- Suppose que l'on sait *indexer* les éléments à trier :

  i.e. affecter à chacun un rang
  - qui dépends uniquement de sa valeur
  - qui respecte l'ordre

Exemple : tri d'une suite de valeurs entières comprises entre 1 et 1000

Tri par dénombrement.

fonction TRI_PAR_DENOMBREMENTS(T, n)
{**In :** T un tableau de n éléments}
{**Out :** R le tableau trié des éléments de T}
début

pour $i := 0$ **à** $k - 1$ **faire**	*initialisations*
$nb[i] := 0,$	
pour $i := 1$ **à** n **faire**	*calcul des nombres d'apparitions*
$nb[T[i]] := nb[T[i]] + 1,$	
$pos[0] := 0,$	*calcul des indices du premier*
pour $i := 1$ **à** $k - 1$ **faire**	*élément de chaque catégorie*
$pos[i] := pos[i - 1] + nb[i - 1],$	
pour $i := 1$ **à** n **faire**	*recopie des élément originaux*
$R[pos[T[i]]] := T[i],$	*du tableau T dans R*
$pos[T[i]] := pos[T[i]] + 1,$	
renvoyer R	

fin procédure

Complexité : $O(n + k)$. Il est donc *linéaire*.

Tri par dénombrement.

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

Nombres d'apparitions

	0	1	2
nb	3	4	3

Indices des premiers

	0	1	2
pos	0	3	7

placement de T[0] en R[0]

pos	1	3	7

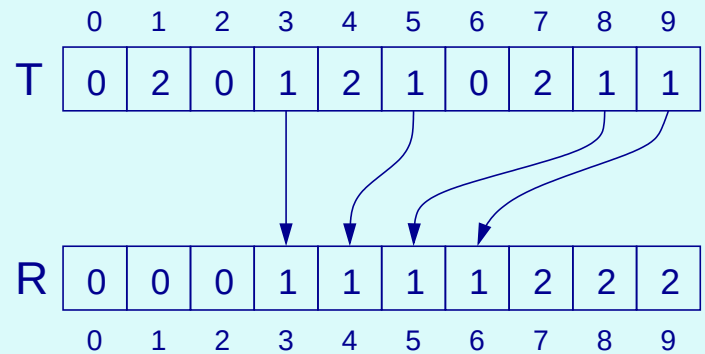
placement de T[1] en R[7]

pos	1	3	8

placement de T[2] en R[1]

pos	2	3	8

°
°
°



Tri par base.

Utilise le tri par dénombrement en plusieurs passes.

536	592	427	167
893	462	536	197
427	893	853	427
167	853	462	462
853	536	167	536
592	427	592	592
197	167	893	853
462	197	197	893

n nombres à c chiffres de k valeurs possibles. Complexité $O(c \times n + c \times k)$.

Si c est constant et si $k = O(n)$ le tri par base est linéaire ($O(n)$).