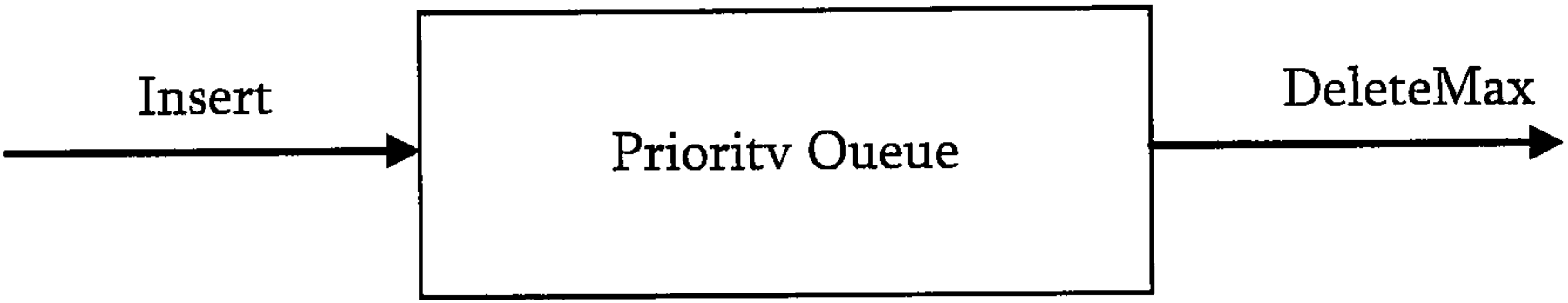
Files de priorité et Tas

Qu'est ce qu'une File de Priorité?

- La file d'attente de priorité est une structure de données qui prend en charge l'Insertion et la Suppression de Min renvoie et supprime l'élément min) ou suppression de Max.

- Ces opérations sont équivalentes aux opérations Enfiler et Défiler de la file d'attente. La différence est que, dans la file d'attente de priorité, l'ordre dans lequel les éléments entrent dans la file d'attente peut ne pas être le même dans lequel ils ont été traités...



Opérations Principales des FP

- **Insérer ()**: insérer une donnée dans la FP
- **Supprimer Min/ Supprimer Max**: retirer de la file et retourner l'élément avec la plus petite/grande clé
- **Trouve Min/ Trouve Max**: retourner l'élément avec la plus petite/ grande clé sans le supprimer
- Et bien d'autres …

Implémentation

- **Il existe plusieurs implémentation pour une FP**

Implémentation	Insertion	Suppression (Max/Min)	Trouve (Min/Max)
Tableau	1	N	N
Liste	1	N	N
Tableau ordonné	N	1	1
Liste ordonné	N	1	1
ABR	Log N (moy)	Log N (moy)	Log N (moy)
ABR équilibré	Log N (Pire)	Log N (Pire)	Log N (Pire)
Tas Binaire	Log N	Log N	1

Complexité des ABR

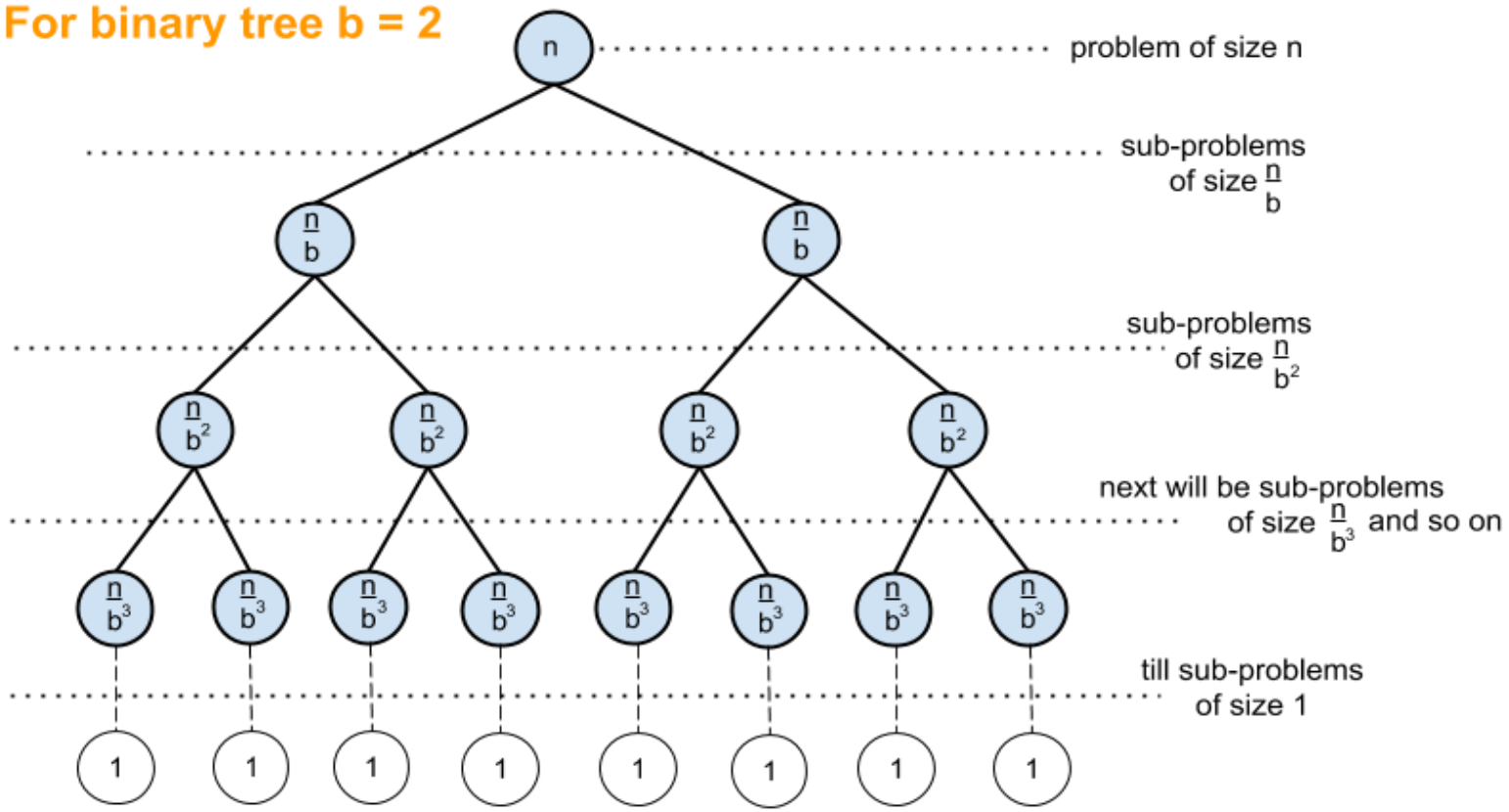
- **Parcours Infixé prend un temps de O(N) au pire cas pour parcourir an ABR de N nœuds, comme après l'appel initial, la procédure fait des appelles récursifs 2 fois (FG et FD)**

- **Recherche, Insertion, Suppression d'un nœud dans un ABR prend un temps de O(h) au pire cas.**
  « *h* » est la hauteur de l'arbre.

- **Pour un ABR Complet, les opérations de base prend un temps de O(log N) au pire cas.**

(si l'arbre est une liste chainé, la complexité est O(N))

ABR le temps de la plupart des opérations est O (log N) en moyen



The height of the above tree is answer to the following question: How many times we divide problem of size n by b until we get down to problem of size 1?

The other way of asking same question:

when $\frac{n}{b^x} = 1$ *[in binary tree b = 2]*

i.e. $n = b^x$ which is $\log_b n$ *[by definition of logarithm]*

Tas et Tas Binaire
(Heaps & Binary Heaps)

- L'implémentation qu'on va utiliser pour FP est le Tas Binaire

  - Comme l'ABR, le Tas a 2 propriétés:
    - Structure
    - Ordre

Qu'est ce qu'un Tas ?

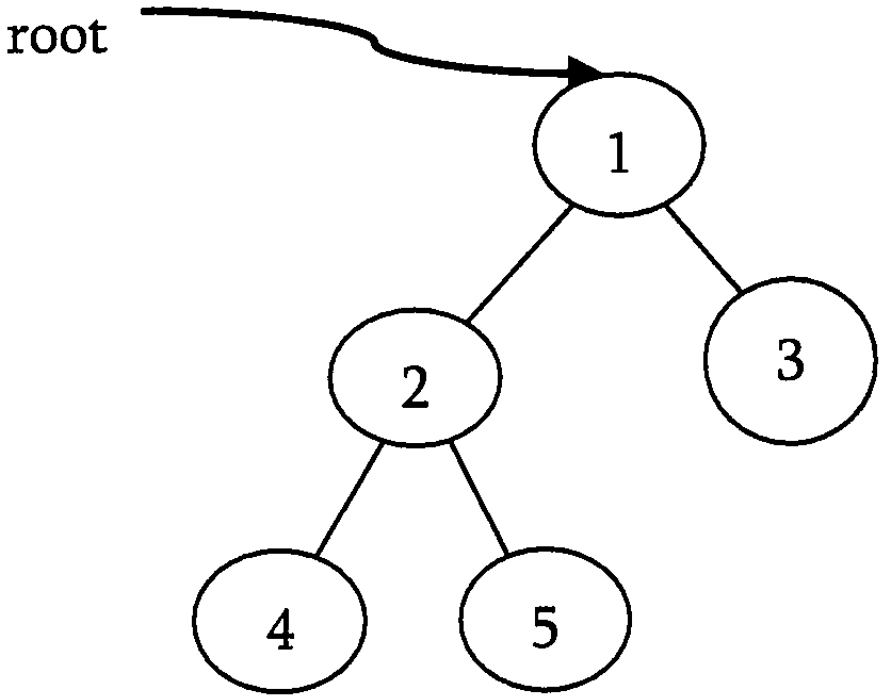
- **Tas est un Arbre avec propriétés:**

1- A.clé >= A.Fils.clé (A.clé <= A.Fils.clé)

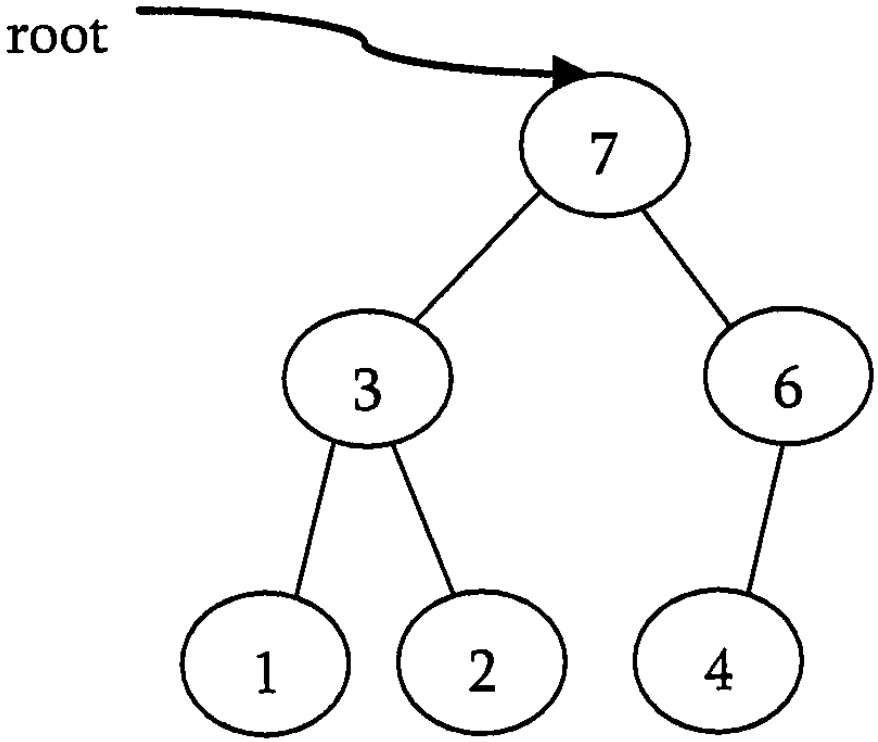
2- Toutes les feuilles existent au niveau h ou h-1

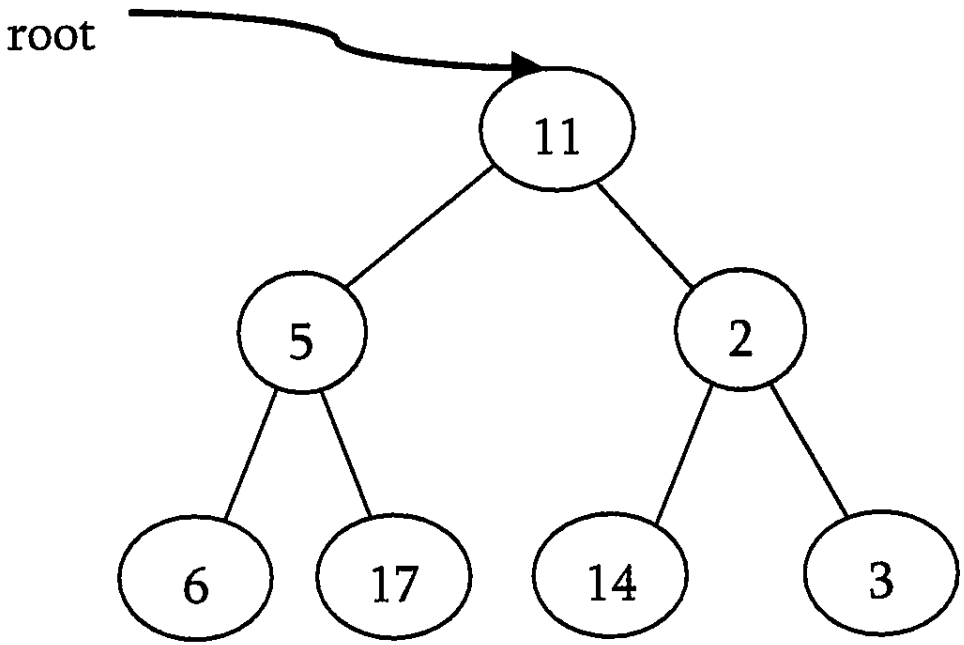
(Tas est un Arbre Binaire Complet)

(Exception possible de dernier niveau de G à D)



Tas



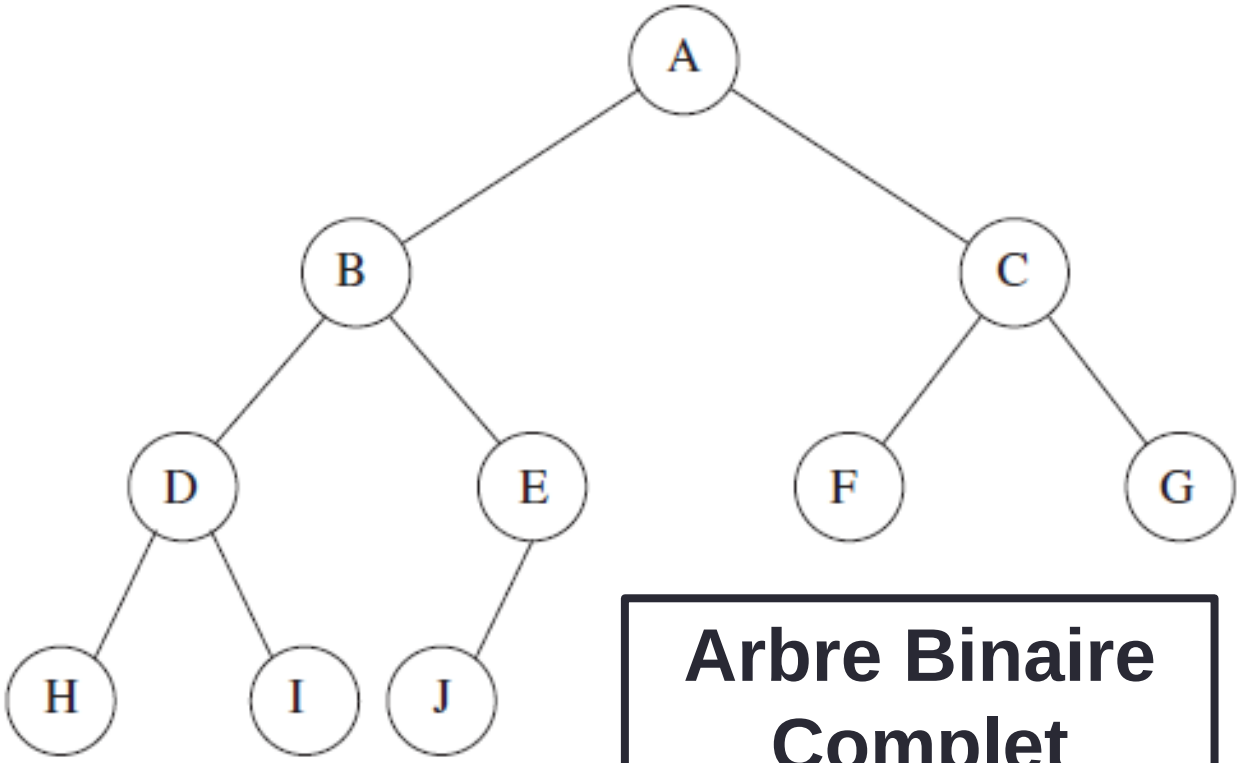


Tas

N’est pas un Tas

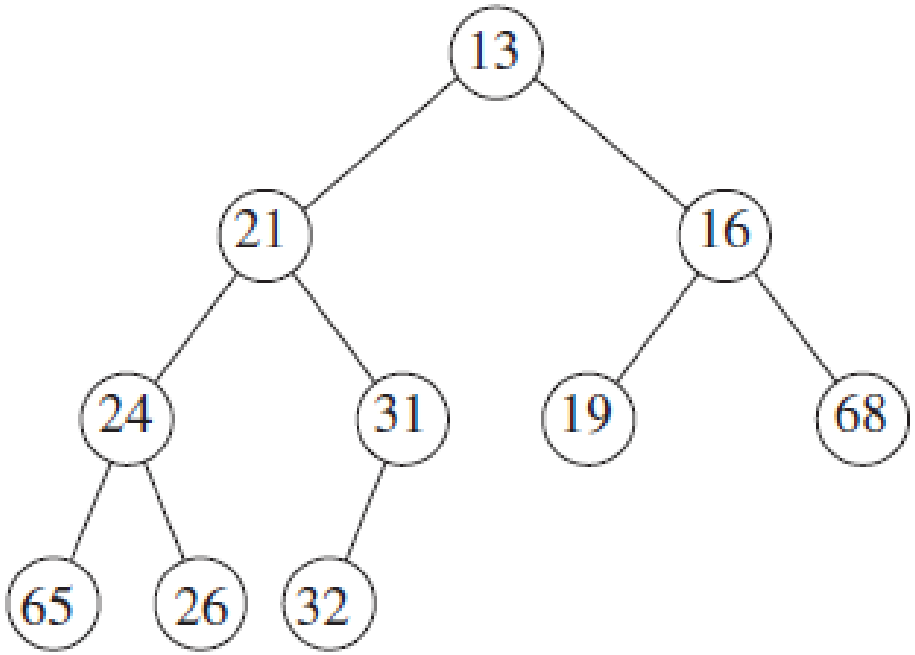
Tas

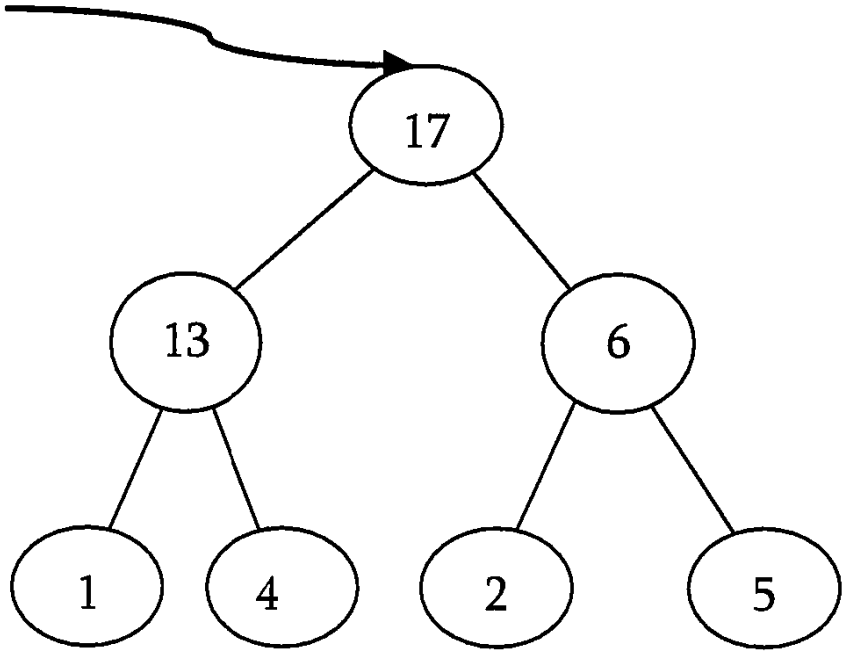
- Un Arbre binaire complet d'hauteur h a un nombre de nœuds entre 2^h et 2^{h+1}-1
- Donc la hauteur d'un Arbre Binaire Complet est [Log N] qui est clairement O(Log N)



Arbre Binaire Complet

Types de Tas

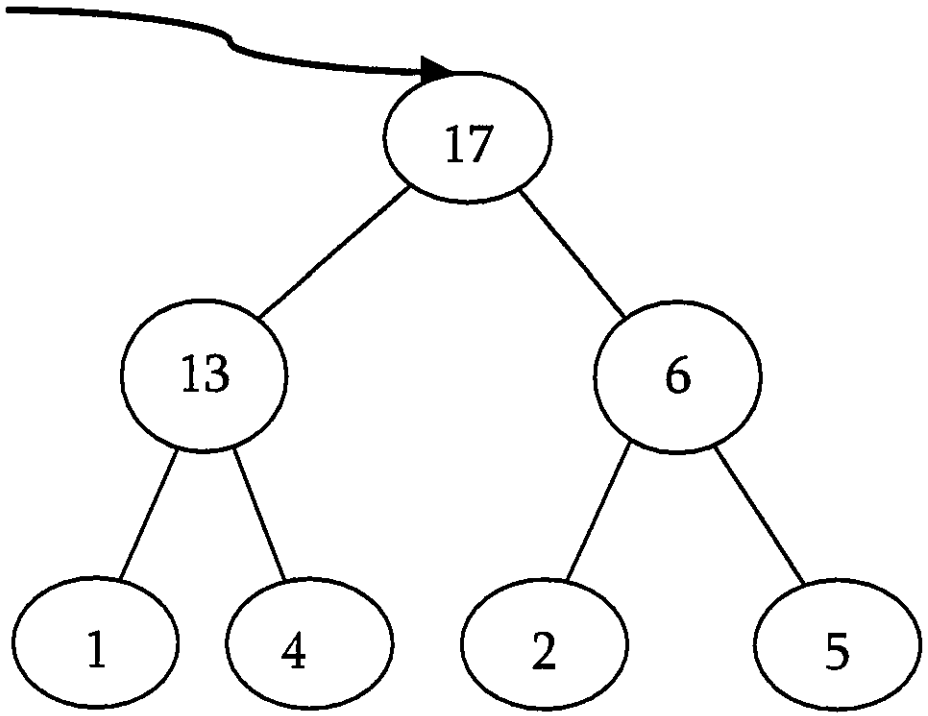




1- Tas Min

2- Tas Max

Représentation de Tas -par- Tableau



- **x** dans case **[i]**
- **FG** de x dans la case **[2*i+1]**
- **FD** de x dans la case **[2*i+2]**
- **Père** de x dans la case **[i-1/2]**

17	13	6	1	4	2	5
0	1	2	3	4	5	6

Déclaration de Tas

```
Typedef struct  Tas
{ int *Tab;
  int   Nombre;    // Nombre des éléments dans le Tas
  int   Taille;    // La taille du Tas
  int   TypeTas;   // Tas Min ou Tas Max
}
```

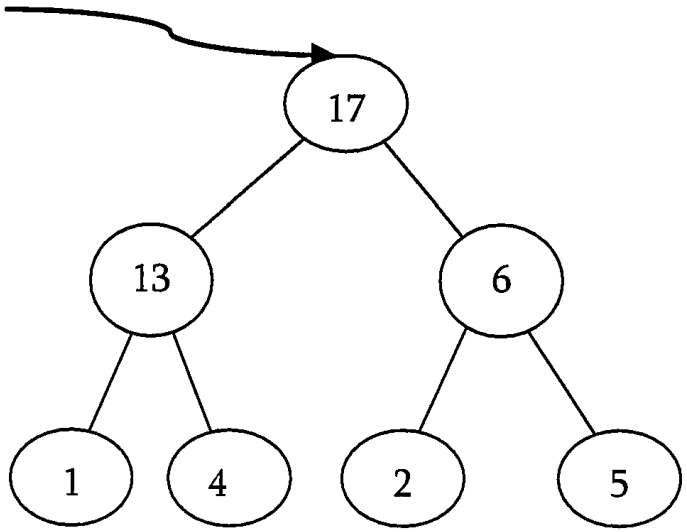
Création de Tas

```
Typedef struct  Tas
{ int *Tab;
  int   Nombre;
  int   Taille;
  int   TypeTas; }
```

```
Fonction CreerTas (Capacité: entier, Type: entier): Tas
Début
{  Tas *T= Allouer ( Tas );
     T.Nombre = 0;
     T.Taille = Capacité;
     T.TypeTas = Type;
     T.Tab= Allouer ( entier * T.Taille);
 retourner (T);
} Fin
```

Le Parent du nœud dans Tas

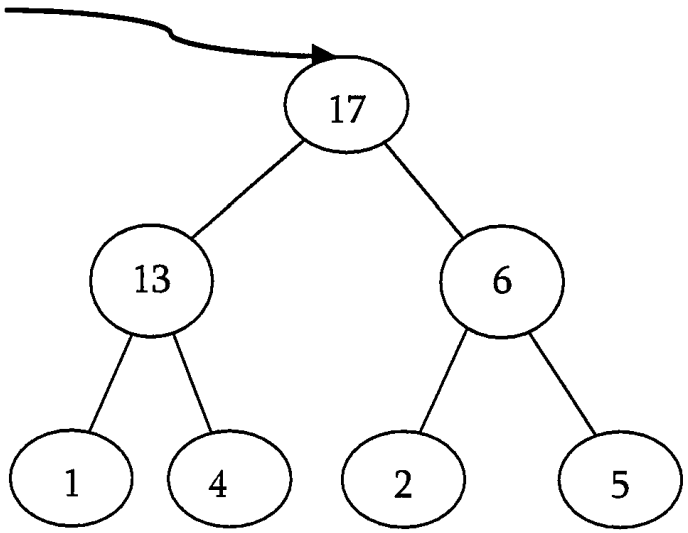
```
Fonction ParentTas (T: Tas, i: entier): entier
Début
{Si (i<= 0) Alors retourner (-1);
 Sinon retourner (i-1/2);
} Fin
```

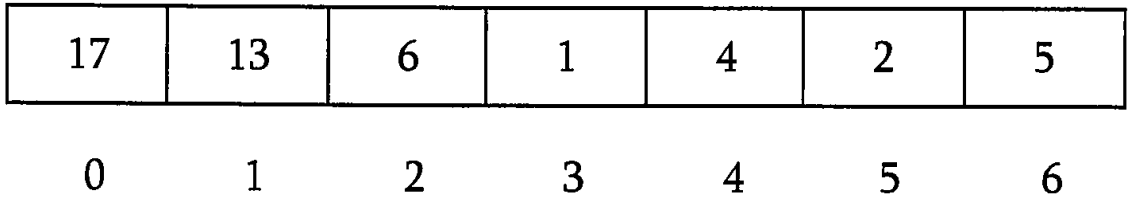


17	13	6	1	4	2	5
0	1	2	3	4	5	6

Les Fils du Nœud dans Tas

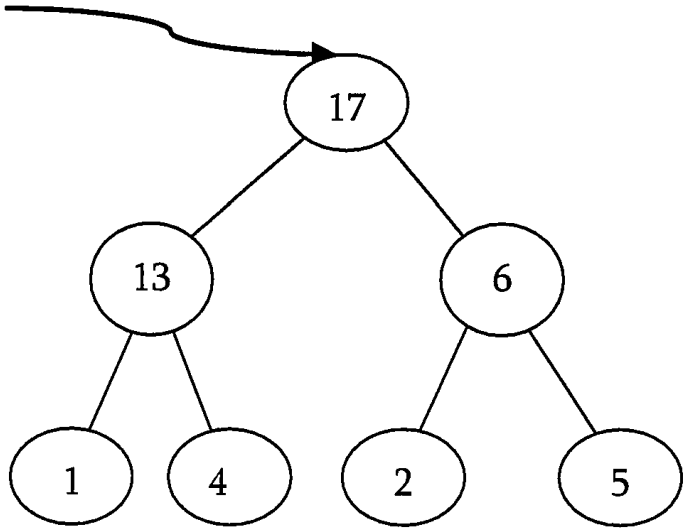
```
Fonction FGTas (T: Tas*, i: entier): entier
Début
{Gauche: entier;
 Gauche = 2*i+1;
Si (Gauche >= T.Nombre) Alors
        retourner (-1);
retourner (Gauche);
} Fin
```

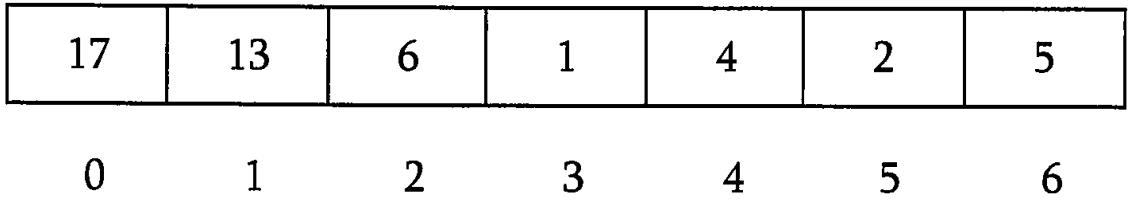




Les Fils du Nœud dans Tas

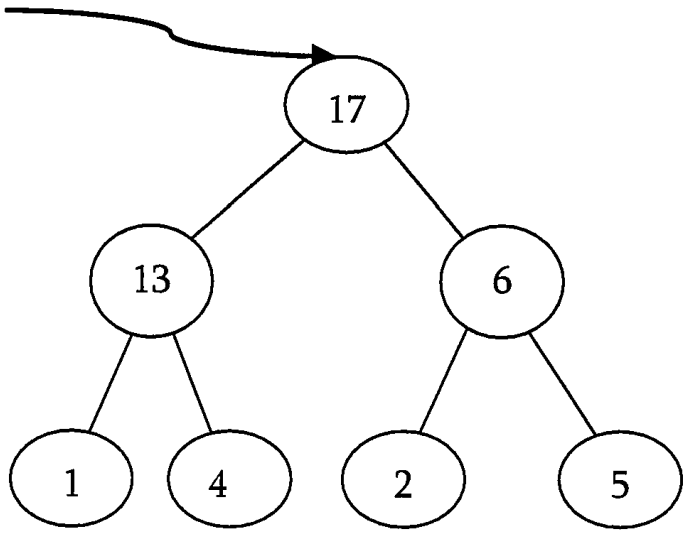
```
Fonction FDTas (T: Tas*, i: entier): entier
Début
{Droit: entier;
 Droit = 2*i+2;
Si (Droit >= T.Nombre) Alors
        retourner (-1);
retourner (Droit);
} Fin
```

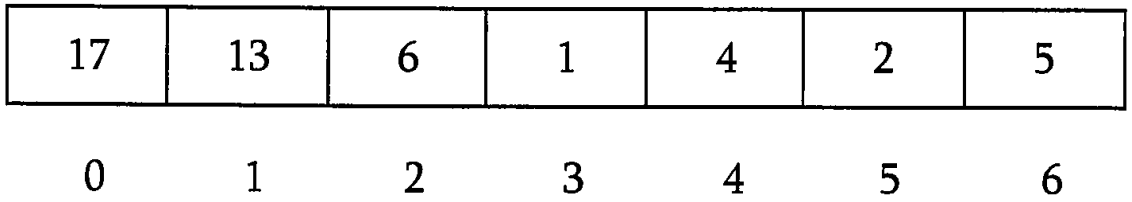




Obtenir le Max dans Tas

```
Fonction MaxTas (T: Tas*): entier
Début
{Si (T.Nombre = 0) Alors
        retourner (-1);
retourner (T.Tab[0]);
} Fin
```





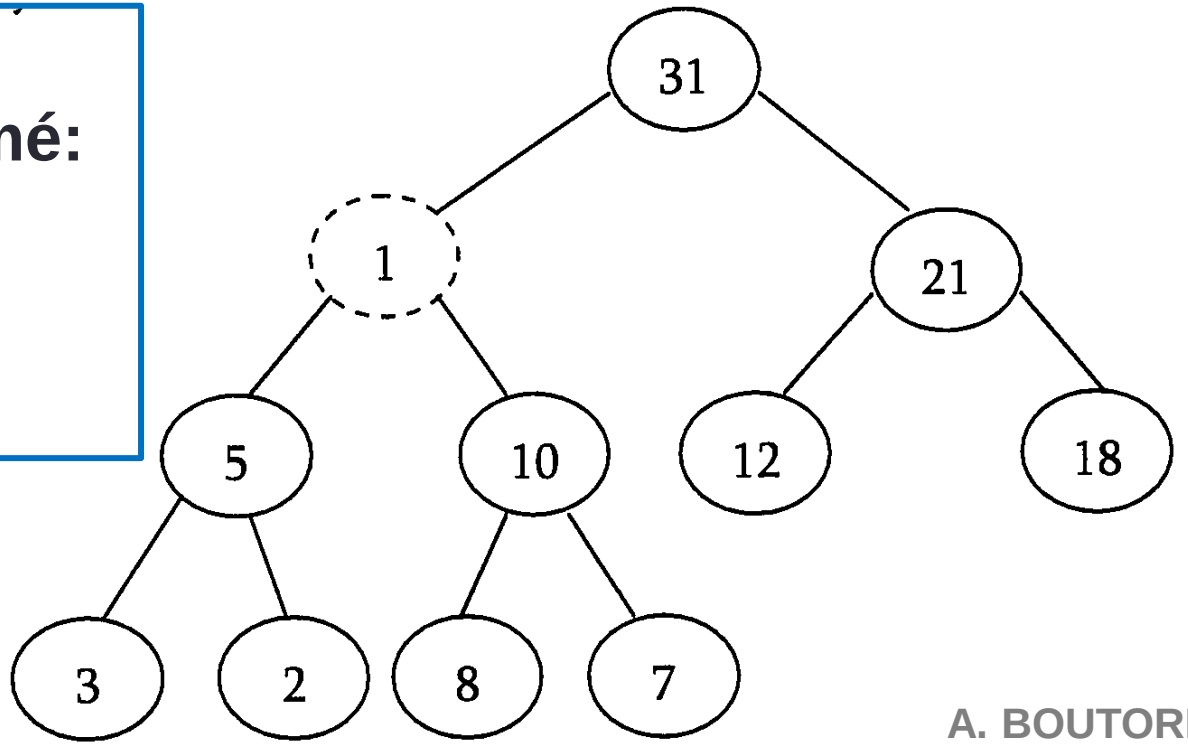
Rendre Tas à Nouveau

- Après une opération dans le Tas, il se peut que les propriétés du tas soient violées.
- Dans ce cas on a besoin de le rendre un Tas à nouveau.
- Dans le Tas Max, pour rendre Tas à nouveau on a besoin de trouver le Max de ses fils et le permuter avec l'élément.
- Continuer le processus jusqu'à que les propriétés de Tas seront vérifiées pour tous les nœuds.

De haut en bas, ce processus est nommé:

Percolate Down

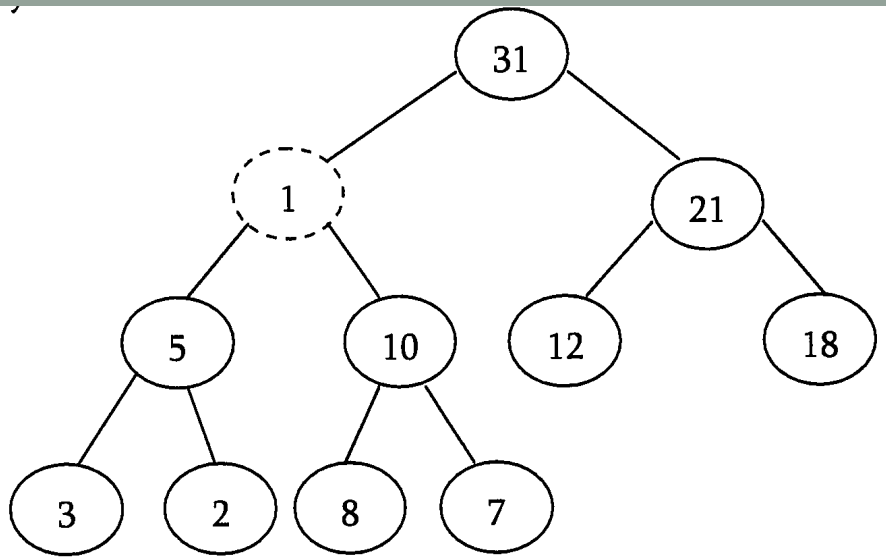
Percoler vers le bas

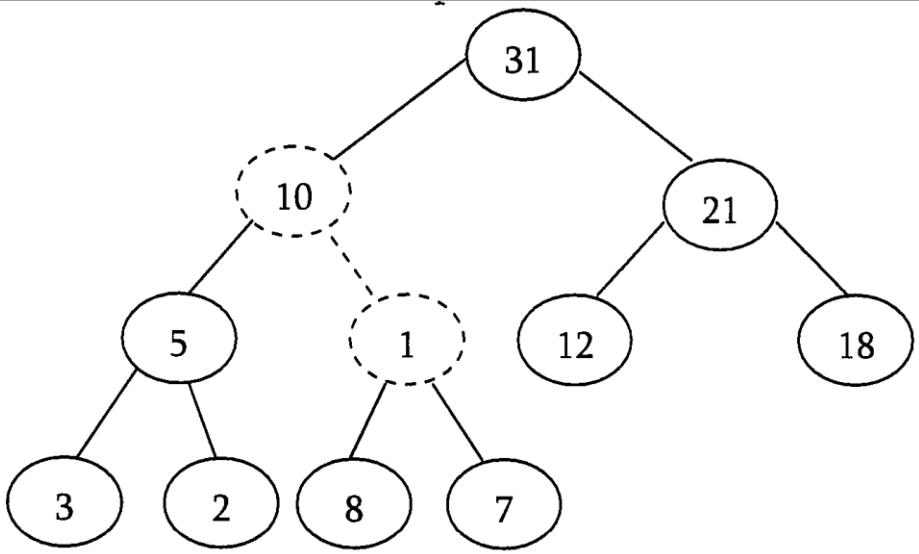


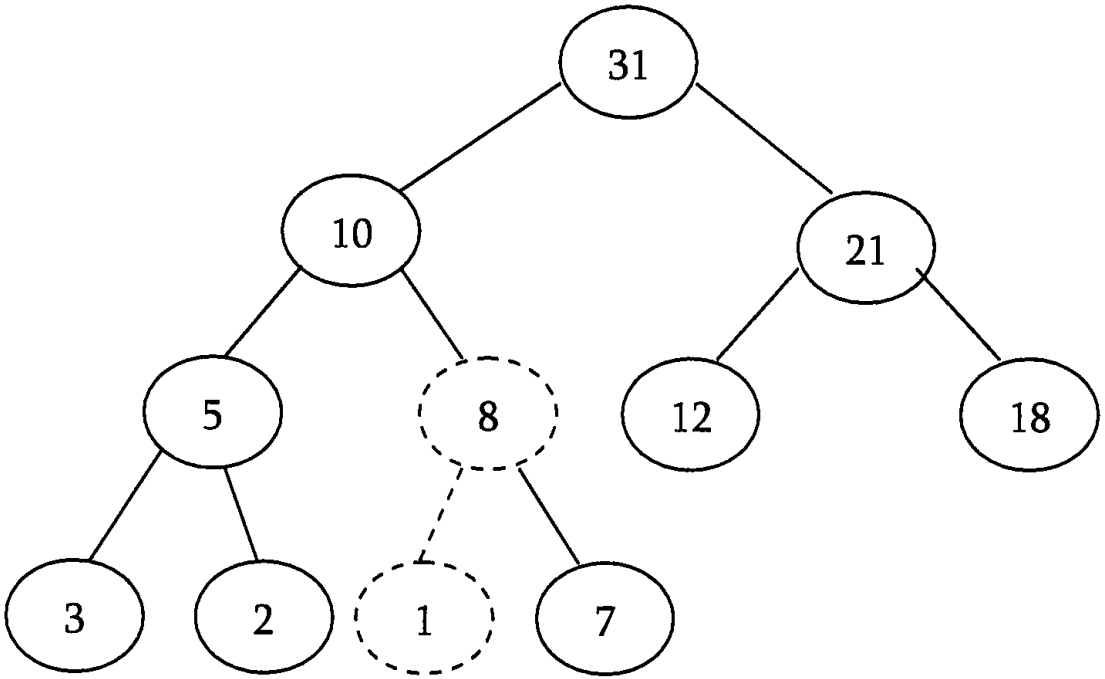
- **Une propriété importante des Tas: Si un élément ne satisfait pas les propriétés du Tas, Alors tous les éléments de cet élément à la racine ont aussi le même problème.**

- **Egalement, si on corrige le Tas pour cet élément alors tous les éléments de cet élément à la Racine vont assurer les propriétés de Tas automatiquement.**

- **Exemple:**







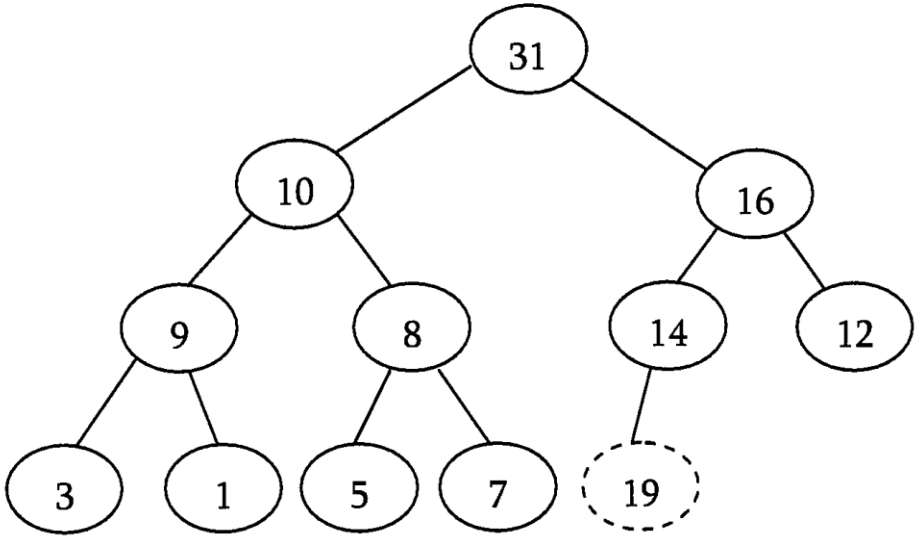
Complexité temporelle O(log N). Un Tas est un Arbre Binaire Complet.

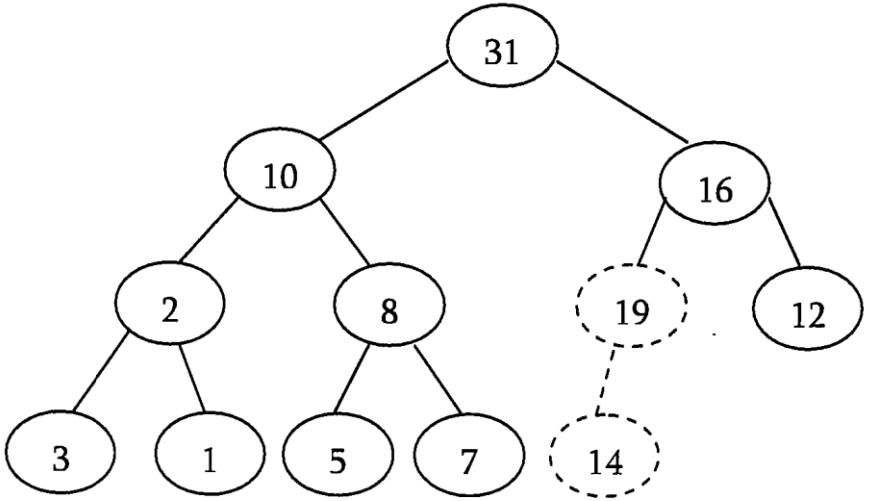
Au pire cas on commence de la racine jusqu'à la dernière feuille = la hauteur de l'Arbre Binaire complet

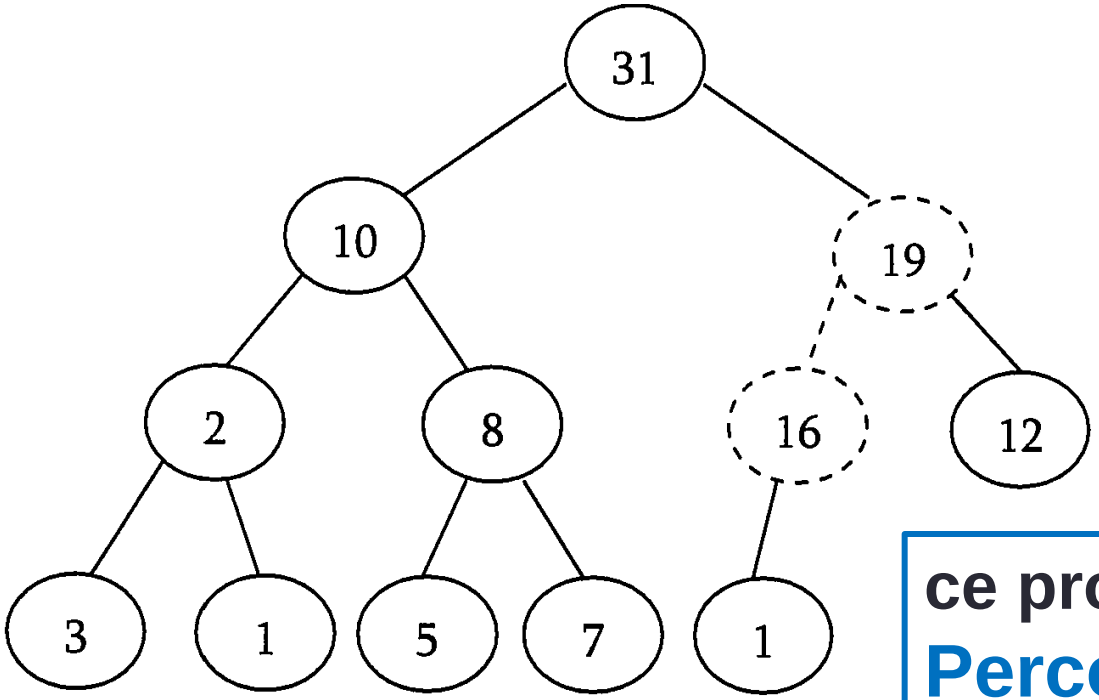
Insertion d'un élément dans Tas

- Augmenter la taille du Tas
- Insérer le nouveau élément à la fin du Tas
- Rendre Tas à nouveau si la propriété d'ordre était violée

- Exemple: Insertion de 19 ne satisfait pas la propriété d'ordre dans le Tas







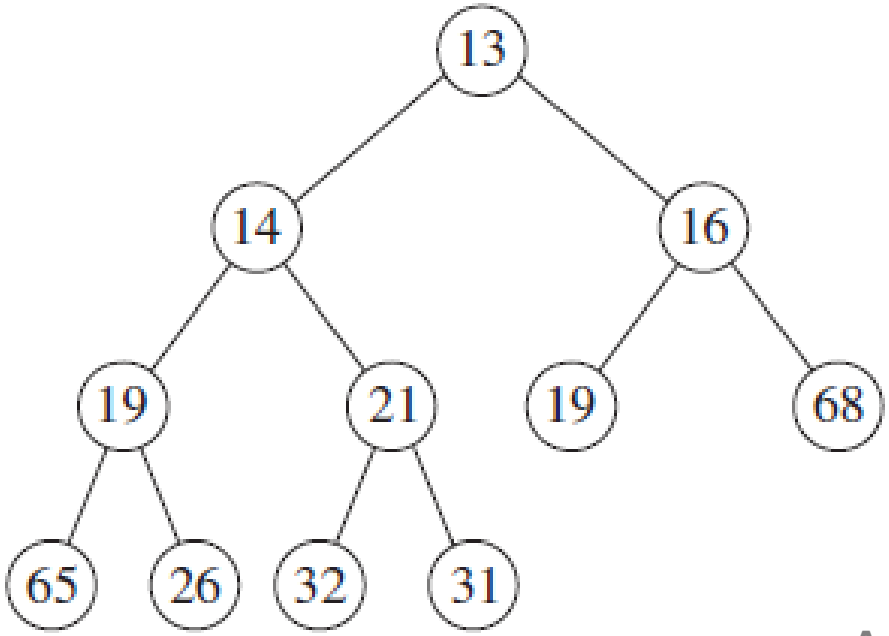
ce processus est nommé:

Percolate Up

Percoler vers le haut.

Suppression d'un élément du Tas

- **Pour supprimer un élément du Tas, on a juste besoin de supprimer l'élément de la racine (le Max) de Tas**
- **Après la suppression de la Racine, copier le dernier élément dans la racine et supprimer cet élément.**
- **L'arbre peut ne pas satisfaire les propriétés du Tas, alors rendre l'arbre Tas à nouveau avec le processus Percolate Down**
- **(Percoler vers le bas).**



Les Algorithmes

```
Procédure PercolerBas (T: Tas*, i:entier)
 Début
{ L, D, Max, Tmp: entier
  L = FGTas (T, i);   D = FDTas (T, i);
Si (L <> -1  et T.Tab[L] > T.Tab[i]) Alors Max= L;
  Sinon Max = i;
Fsi;
Si (D <> -1  et T.Tab[D] > T.Tab[Max]) Alors Max= D;
Si (Max <> i) Alors
     Tmp= T.Tab[i];
     T.Tab[i] = T.Tab[Max];
     T.Tab[Max] = Tmp;
Fsi;
PercolerBas (T, Max); } Fin
```

O (log n) Tas est AB complet et au pire cas on commence de la racine jusqu'à la feuille, qui égale h

Les Algorithmes

```
Fonction SupprimeMaxTas (T: Tas*) :entier
 Début
{ Data: entier
 Si (T.Nombre = 0) Alors retourner (-1);
  Data = T.Tab[0];
  T.Tab[0] = T.Tab[T.Nombre-1];
  T.Nombre = T.Nombre -1 // Réduire le nombre
 PercolerBas (T, 0);
Retourner (Data);
 } Fin
```

O (log n)

Les Algorithmes

```
Procédure InsertTas (T: Tas*, Data: entier)
 Début
{ i: entier
 Si (T.Nombre = T.Taille) Alors ResizeTas (T); Fsi;
 T.Nombre = T.Nombre + 1;
 i = T.Nombre -1;
Tant que (i>=0 et Data > T.Tab[(i-1)/2]) Faire
        T.Tab[i] = T.Tab[(i-1)/2];
         i= (i-1)/2;
Fait;
T.Tab[i] = Data;
} Fin
```

Les Algorithmes

```
Procédure ResizeTas (T: Tas*)
 Début
{ Tb_old :  Tableau[T.Nombre]: d'entier;
  Tb_old = T.Tab;
 T.Tab = Allouer ( entier * T.Taille *2);
Pour (i =0 à T.Taille) Faire
        T.Tab[i] = Tb_old [i];
Fait;
T.Taille = T.Taille*2;
Liberer (Tb_old);
} Fin
```

O (log n)

Les Algorithmes

```
Procédure DétruireTas (T: Tas*)
 Début
{ Si  ( T = Nill) Alors Retourner ( Nill);
  Libérer ( T.Tab);
  Libérer (T);
} Fin
```

Les Algorithmes

```
Fonction ConstruireTas (T: Tas*, A[]: entier, N: entier): Tas
 Début
{ Si  ( T = Nill) Alors Retourner ( Nill);

Tant que (N > T.Taille) Faire ResizeTas (T);  Fait;

 Pour (i = 0 à N-1 ) Faire  T.Tab[i] = A[i];  Fait;

T.Taille = N;
```

O (n) par l'application de PercolerBas dans l'ordre de niveau inverse

```
 Pour (i = (N-1)/2 à 0;  i=i-1 ) Faire PercolerBas (T, i); Fait;
Retourner (T);
} Fin
```

FIN