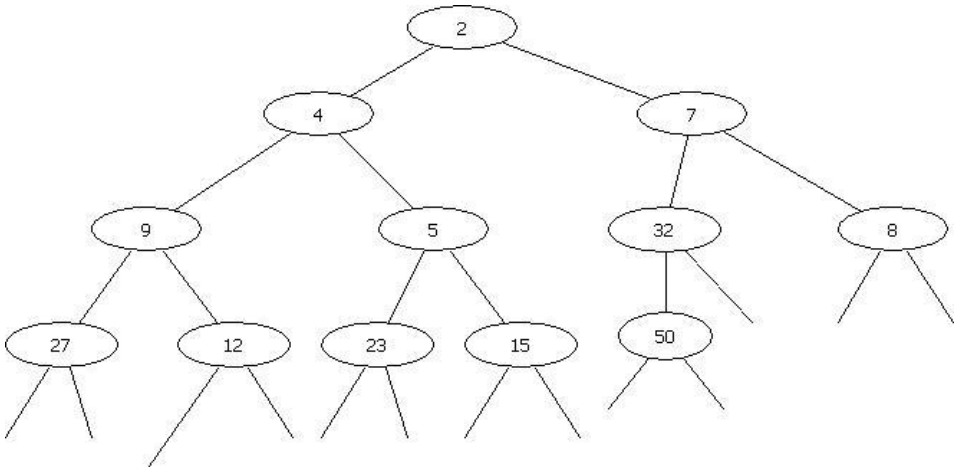
ALGORITHMIQUE

LES TAS

TAS (HEAP EN ANGLAIS)

Arbre binaire valué aux nœuds internes (valeurs aux nœuds mais pas aux feuilles) par des entier (ou autre chose tant qu'il y a un ordre total dessus) avec deux propriétés :

- L'arbre est parfait. Si pert la profondeur de l'arbre, toutes les feuilles sont à profondeur p ou p-1 celles à p-1 sont à droite de celles à p. « remplis niveaux par niveau de gauche à droite »
- Si x est père de y, alors la x≤y.
  => si x est assendant de y alors x≤y



On veut gérer un ensemble E avec deux opérations de base :

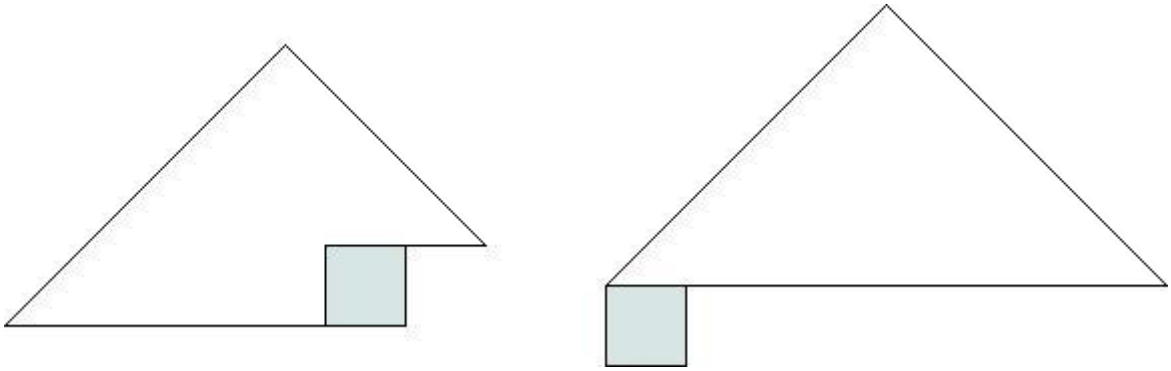
- ⇨ ajouter un élément
- ⇨ sortir le minimum

Coùt	liste	Liste triée	Tas
Ajout	Θ (1)	Θ (n)	Θ (lnn)
Sortir le min	Θ (n)	Θ (1)	Θ (lnn)

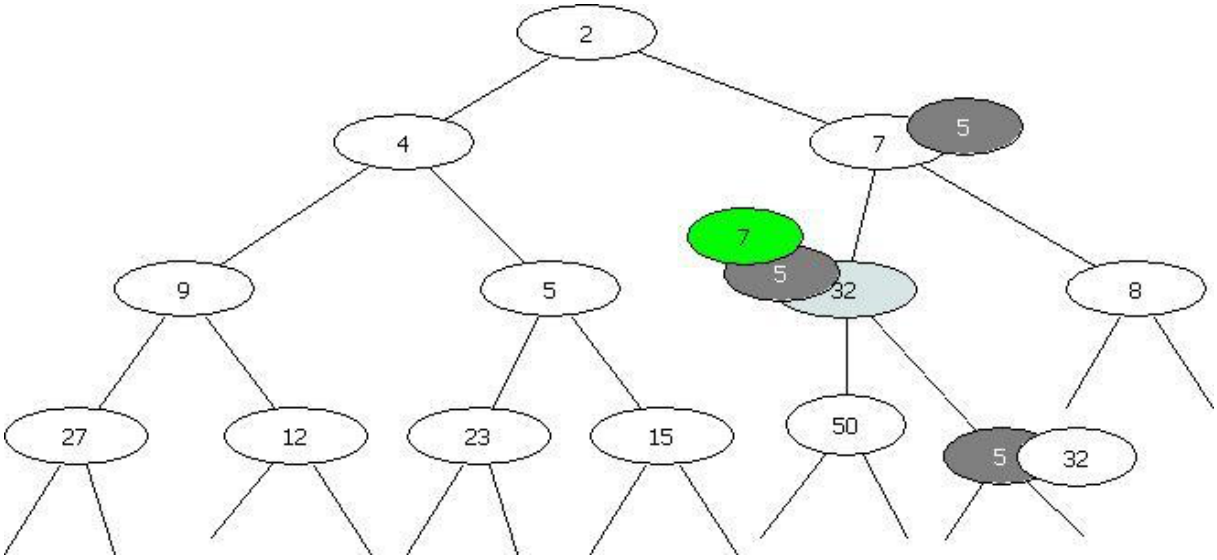
#élements.

rapport entre h (hauteur de l'arbre) et n (nombre d'élément)

AJOUT D'UN ELEMENT A UN TAS



Il faut créer une nouvelle place : un seul lieu possible, la prochaine case du dernier niveau ou la première du niveau suivant si le niveau precedent est plein.



L'élément n'a aucune raison d'être correctement placé. (insertion de 5)

Tant que l'élément un un père et que ce père est plus grand, échanger l'élément avec son père.

Complexité au pire :

- $\ln_2 n$ comparaisons
- $\ln_2 n$ échanges

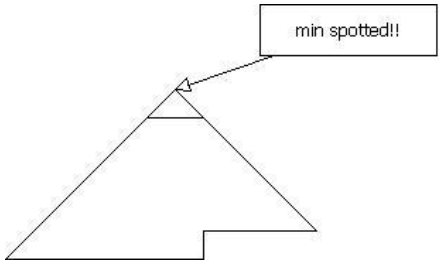
Remarque : Preuve que le procédé est correct :

Lors de la montée, on a à tout moment si x est père de u et u≠ element inséré alors

Et si z est père du nouvel élément et w est fils du nouvel élément, alors

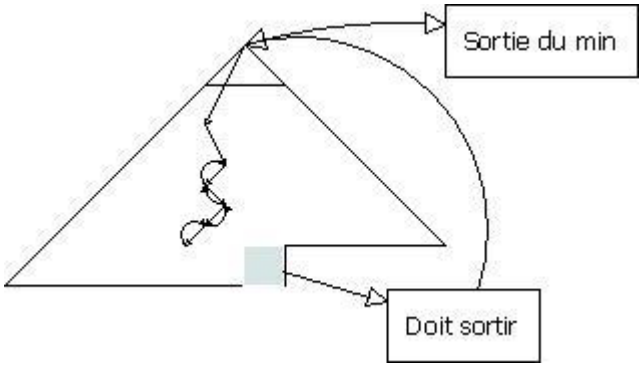
SORTIR LE MIN

Le min est à la cîme de l'arbre



Le min est à la racine, la place libérée ne peut être que la dernière case du dernier niveau.

- On sort le min , trouvé à la racine
- On place à la racine l'élément e le dernier du dernier niveau
- Tant que e a au moins un fils et qu'il est plus grand qu'au moins un des fils :
  - Echanger(e , leplus petit de ses fils)



2 comparaisons par niveaux :

- On compare les fils de e
- e avec le plus petit des deux fils.

TRIER UN TAS

```
Tas : t
  ← Ø
Pour tous les éléments   de E
       Ajoute   à                          //
Tant que le tas n'est pas vide
       Sortir le min et le ranger dans une pile     //
```

Tri par tas est en Θ(nlnn).

IMPLEMENTATION

IDEES 1 : LA STRUCTURE CHAINEES.

Celle-ci necessite 5 pointeurs par nœuds. Beurk ! ☹

IDEE 2 : LE TABLEAU

1	7	12	19	9	15	…
racine					Dernire case de tas	Prochaine case libre

- racine accessible (première case)
- dernière valeur du dernier niveau accessible (dernière case)
- prochaine case libre (à tab[max+1]

Ou est mon père ?

* on prend des tableaux qui commencent en T[1] */

e est en T[i], ou sont …

- le père (P)

- le Fils gauche (FG)

- Le Fils Droit (FD)

Le <u>fils gauche</u> de T[i] est en T[2]

Si le FG de T[i] est en T[j] alors le FG de T[i+1] est en T[j+2]
➔par récurrence, le FG de T[i] est en T[2i]
Le FD de T[i] est en T[2i+1]
le père de T[i] est eb T[i/2]

```
Monter(T[], t, i)
/*
       T : taille du tas
       1≤i≤t
       X← T[i]
       Tant que x a un père plus grand, on le monte
*/
Pos ← i
Tant que pos≠ 1 et T[pos div 2]> T[pos] alors
       Achanger(T[pos div 2@ et T [pos]
       Pos← pos div 2
ftq
```

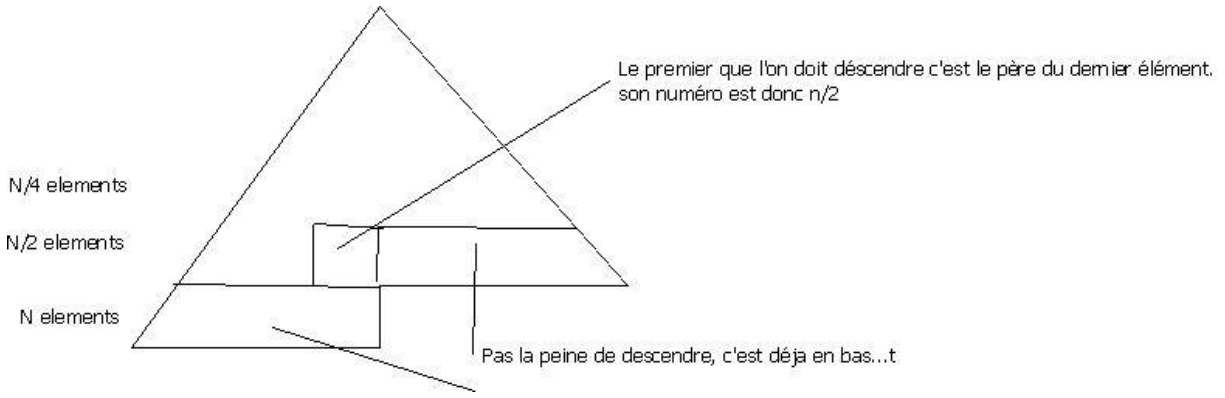
```
Descendre(T[], t, i)
Pos ← i
Stop ← faux
Tant que 2*pos+1≤t faire
      Si T[2pos] ≤ T[2pos+1] alors
            Si T[2pos]<T[pos] alors
                  échanger T[2pos] et T[pos]
            sinon
                  stop← vrai
```

```
            fsi
      sinon
            … 2p ←→ 2p+1
      Fsi
      Si 2pos=t et T[2pos]<T[pos]
            Echange T[2pos]et T[pos]
Ftq
```

```
Tripartas(T[], N)
      Pour k de 2 à N faire
            Monter (T[], k, k)
      Fpour
      Pour k de N à 2 avec pas de -1 faire
            Echanger T[i] et T[k]
            Descendre(T[], k-1,1)
      Fpour
Fin
```

Autre méthode de construction du tas :

```
Construire()
      Pour i decroissant à 1
            Descendre (T[], i, N)
      Fin pour
```



Nombre d'élément du paquet	Coût par élément	Coût du paquet
		____
		____
__	2.3	____

_	2.2	____
_	2	____
_	0	0

__ __ __ __ __

__

Construit le tas de manière linéaire !