# TRI PAR TAS

Leçons : 901, 903, 927, 931

Références : Cormen

**Théorème**

Le tri par tas trie un tableau en place en $O(n \log n)$

**Résumé**

I - Définition d'une procédure ENTASSER_MAX permettant de placer correctement un élément du tas si le reste qui le suit est correct.

II - Construction d'un tas max à partir d'un tableau.

III - Algorithme du tri par tas

Remarque : un tas est représenté sous la forme d'un tableau, mais il peut être vu comme un arbre binaire presque complet, avec les relations

GAUCHE(i) = 2i

DROITE(i) = 2i+1

PARENT(i) = $\lfloor i/2 \rfloor$.

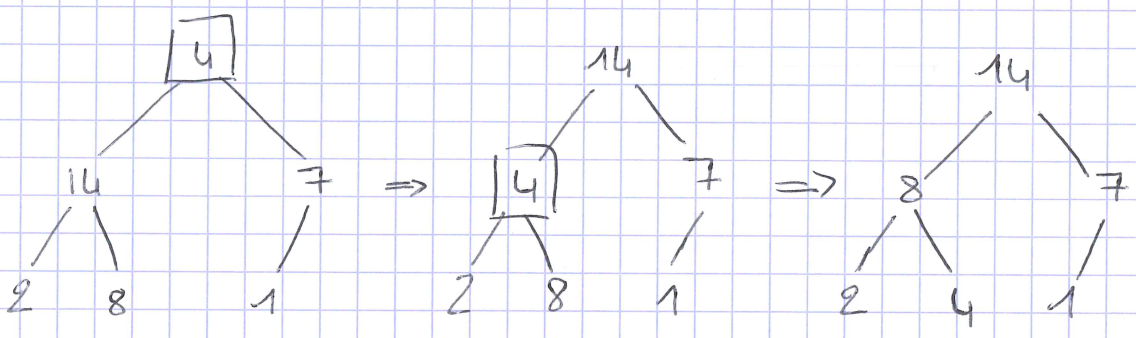
+ un tas max a la propriété, pour i différent de la racine :

A[PARENT[i]] ≥ A[i].

I - On suppose que les tas enracinés en GAUCHE[i] et DROITE[i] sont max. La procédure suivante fait descendre A[i] autant qu'il faut :

```
ENTASSER_MAX(A,i):
    max = argmax(A[i], A[GAUCHE[i]], A[DROITE[i]])
    si max ≠ i
        ECHANGER A[i] et A[max]
        ENTASSER_MAX(A, max).
```

Exemple :




À chaque étape, le tas non modifié est max et son parent est plus grand, et on continue de même récursivement jusqu'à ce que ce soit bon.

Complexité : à chaque appel récursif de ENTASSER_MAX, la hauteur de l'arbre considéré, h, diminue de 1. L'algorithme s'exécute donc en $O(h)$

Comme l'arbre est presque complet, $O(h) = O(\log n)$.

II • En partant d'un tableau A de taille n, on peut construire en place un tas max en entassant à partir de l'étage juste au dessus des feuilles.

• Entre $\lfloor n/2 \rfloor + 1$ et n, l'arbre binaire ne contient que des feuilles, on peut donc commencer à entasser en $\lfloor n/2 \rfloor$ :

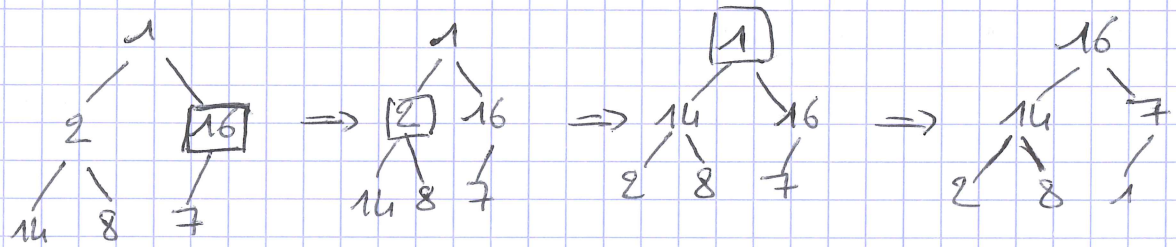
```
CONSTRUIRE_TAS_MAX
    n = taille[A]
    Pour i = ⌊n/2⌋ à 1,
        ENTASSER_MAX(A,i)
```

Correction On utilise l'invariant de boucle suivant :

"Au début de chaque itération, les noeuds i+1, i+2, ..., n sont racines d'un tas max."

Initialisation : pour $i = \lfloor n/2 \rfloor$, les noeuds i+1 à n sont des feuilles, donc des racines de tas max à un élément.

Conservation : les indices des enfants de $i$ sont strictement plus grands que lui : ce sont donc des racines de tas max.
L'appel d'ENTASSER_MAX construit donc un tas max enraciné en $i$, et les noeuds d'indices plus grands restent des racines de tas max.




Complexité : la complexité d'ENTASSER_MAX sur un tas de hauteur $h$ est $O(h)$, et il y a au plus $\lceil n/2^{h+1} \rceil$ noeuds à hauteur $h$.
Le coût en temps est donc

$$\sum_{h=1}^{\lfloor \log_2 n \rfloor} \lceil n/2^{h+1} \rceil O(h) = O\left(n \sum_{h=1}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h}\right) = O(n) \quad \text{car} \sum_{h=1}^{+\infty} \frac{h}{2^h} = 2.$$

III • On a presque l'algorithme de tri :
- on construit déjà un tas max dans le tableau $A$.
- on peut échanger $A[1]$ (le plus grand élément) et $A[n]$ (le dernier), dès lors $A$ privé de son dernier élément est un tas max à l'exception de $A[1]$, qu'il suffit d'entasser, en ignorant le dernier élément du tableau :

```
TRI_PAR_TAS(A)
    CONSTRUIRE_TAS_MAX
    Pour i = longueur(A) jusqu'à 2 :
        Echanger A[1] ↔ A[i]
        Taille(A) = Taille(A) - 1.
        ENTASSER_MAX(A, 1)   (on ignore dans cet appel la fin de A)
```

Correction découle directement de celles de ENTASSER_MAX et de CONSTRUIRE_TAS_MAX : on a un tas max donc T[1] est toujours le plus grand élément → et c'est celui qu'on met à la fin !

Complexité

- Construction du tas max : $O(n)$
- $O(\log n)$ pour chaque appel à ENTASSER_MAX

→ d'où une complexité en temps en $O(n \log n)$.