Complexité algorithmique de la construction d'un monceau :

Problème :

Un monceau est un arbre binaire complet ordonné en tas, c'est-à-dire un arbre binaire complet obéissant à la définition récursive voulant que tout nœud dans l'arbre possède une clé dont la valeur est inférieur ou égale à l'ensemble des clés de ses enfants.

La construction d'un monceau peut s'effectuer au moins de deux manières :

- En insérant les éléments un à un dans le monceau.
- En insérant l'ensemble des éléments à la fois ; autrement dit, partant d'un tableau, ordonner en tas les éléments dans le tableau.

Dans ce qui suit, nous considérons la complexité algorithmique de la construction d'un monceau suivant l'une et l'autre approche.

Analyses de complexité:

1. Première approche

Considérons dans un premier temps la complexité algorithmique de la construction d'un monceau de n éléments. Au départ, le monceau est vide. Chaque élément est inséré dans le monceau à tour de rôle.

L'insertion d'un élément dans un monceau a une complexité $\mathcal{O}(1)$[1] en cas moyen et une complexité $\mathcal{O}(\log(n))$ en pire cas. Le pire cas survient lorsque l'insertion induit une série de percolations jusqu'à la racine (autrement dit, l'élément inséré est le plus petit parmi l'ensemble des éléments déjà présents[2]).

Ainsi, la construction d'un monceau élément par élément prend un temps $\mathcal{O}(n)$ en temps moyen et $\mathcal{O}(n\log(n))$ en pire cas. Le pire cas survient lorsque les éléments sont insérés en ordre décroissant (ex. 7, 6, 5, 4, 3, 2 puis 1).

Preuve:

La preuve du cas moyen est évidente. En admettant que l'insertion prend en moyenne $\mathcal{O}(1)$, l'insertion de n éléments prend $n\mathcal{O}(1) = \mathcal{O}(n)$. La preuve de la complexité $\mathcal{O}(n\log(n))$ en pire cas est plus difficile.

Rappel : Un monceau comportant n éléments a une hauteur h de valeur $h = \lfloor \log_2(n) \rfloor$, où $\lfloor x \rfloor$ signifie x arrondi au plus proche entier plus petit ou égal à x, x étant un réel. Rappelons que la

[1] La preuve de la complexité en $\mathcal{O}(1)$ de l'insertion n'est pas aisée. Elle implique des notions de probabilité et des hypothèses réalistes quant à la nature des applications. Mark Weiss mentionne à la page 206 de la seconde édition de Data Structures and Algorithm Analysis in Java qu'il a été démontré qu'une insertion implique au plus 1.607 percolations.

[2] On entend ici que l'on insère un élément dont la clé est plus petite ou égale que toutes les clés des nœuds de l'arbre. Nous admettrons ici ce type d'abus de langage ici dont le but est d'alléger le texte.

hauteur d'un arbre est le nombre de niveaux qu'il comporte moins 1. Notons aussi que : $2^h \leq n < 2^{h+1}$

Lorsqu'un élément est inséré dans un monceau de taille $n-1$ contenant des éléments tous plus grands où égaux à l'élément inséré, il faut effectuer $h = \lfloor \log_2(n) \rfloor$ percolations pour rétablir l'ordre en tas. Lorsque cette opération est répétée n fois pour l'insertion des éléments du monceau en ordre décroissant, nous avons un total d'opérations égal à :

$$T(n) = \sum_{i=1}^{n} \lfloor \log_2(i) \rfloor$$

$$\sum_{i=1}^{n} (\log_2(i) - 1) < T(n) \leq \sum_{i=1}^{n} \log_2(i)$$

$$\log_2(\prod_{i=1}^{n} i) - n < T(n) \leq \log_2(\prod_{i=1}^{n} i)$$

$$\log_2(n!) - n < T(n) \leq \log_2(n!)$$

Or,

$$(\frac{n}{2})^{(\frac{n}{2})} < n! < n^n \text{ , pour } n > 1$$

D'où il ressort,

$$\frac{n}{2} \cdot log\left(\frac{n}{2}\right) < \log(n!) < n \cdot log(n) \text{ , pour } n > 1$$

Et on trouve que $T(n)$ est en $\mathcal{O}(n\log(n))$.

En résumé, la construction d'un monceau en insérant les éléments un à la fois a une complexité:

Meilleur cas	Moyen cas	Pire cas
$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot \log(n))$

2. Seconde approche

Dans le cas où l'on insère tous les éléments à la fois, la construction du monceau s'effectue par une série de percolations vers le bas de $2^h - 1$ nœuds. Dans le meilleur cas, aucun élément n'est percolé et la procédure prend un temps $\mathcal{O}(n)$ (temps de visite des nœuds). Dans le pire cas, chaque nœud visité est percolé tout en bas de l'arbre. Ainsi, un nœud à la hauteur i est percolé h-i niveaux plus bas. Nous aurons donc :

$$T(n) = \sum_{i=0}^{h-1} 2^i(h-i) = h + \sum_{i=1}^{h-1} 2^i(h-i)$$

En notant que :

$$2T(n) = \sum_{i=0}^{h-1} 2^{i+1}(h-i) = \sum_{i=1}^{h} 2^{i}(h-i+1)$$

$$= \sum_{i=1}^{h-1} 2^{i}(h-i+1) + 2^{h}$$

En soustrayant les deux termes, on trouve :

$$T(n) = 2^{h} + \sum_{i=1}^{h-1} 2^{i} - h$$

$$= 2^{h} + (2^{h} - 1) - h = 2^{h+1} - (h+1)$$

Il en ressort que $T(n)$ est en $\mathcal{O}(n)$. En résumé, la construction d'un monceau en insérant tous les éléments d'un trait garantit un temps $\mathcal{O}(n)$ en tout cas en temps.