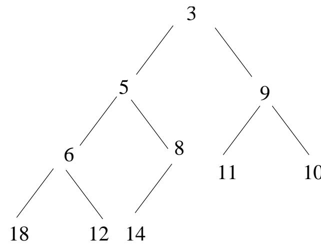
Université de Provence
Master 2. Génie Statistique et Informatique. 2004 – 2005
Solange COUPET-GRIMAL

# Examen d'Algorithmique

## Le Tri par Tas

### Arbres binaires parfaits partiellement ordonnés (Tas)

Un arbre binaire est dit *parfait* lorsque tous ses niveaux sont complètement remplis, sauf éventuellement le dernier niveau et dans ce cas les nœuds (feuilles) du dernier niveau sont groupés le plus à gauche possible. Ceci est illustré par la figure ci-dessous.




Un arbre binaire parfait de taille $n$ peut se représenter de façon très compacte dans un tableau $T$ de taille au moins égale à $n+1$ (la première cellule de $T$, de rang 0, n'est pas utilisée). La représentation est la suivante:

**La racine est en T[1] et si un nœud est en T[i], son fils gauche est en T[2i] et son fils droit en T[2i+1]**.

**Question 1** Donner la représentation dans le cas de l'arbre ci-dessus.

L'arbre et chacun de ses sous-arbres sont repérés par un indice $i$, $1 \leq i \leq n$. Les primitives relatives à l'ensemble des sous-arbres $\{1, \ldots, n\}$ de l'arbre parfait de taille $n$ sont alors :

**racine(i)** : i
**etiquette(i)** : T[i]
**arbre-vide(i)** : i = 0
**sous-arbre-gauche(i)** : si $2i \leq n$ alors $2i$ sinon 0
**sous-arbre-droit(i)** : si $2i+1 \leq n$ alors $2i+1$ sinon 0.
**pere(i)** : $i/2$.

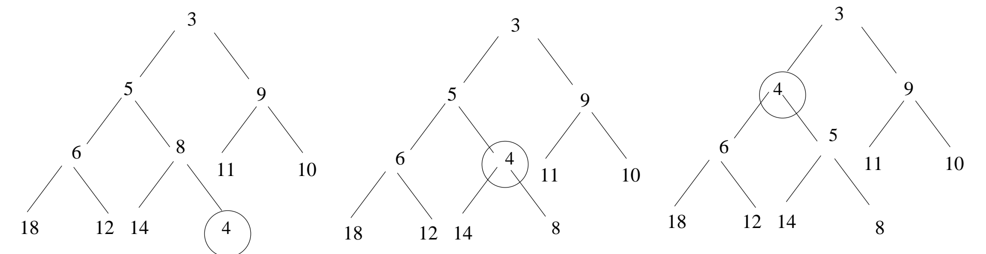
Un arbre binaire parfait (étiqueté sur un ensemble ordonné) est dit partiellement ordonné si l'étiquette de tout nœud est inférieure à celle de ses fils. C'est le cas de l'exemple ci-dessus. Un arbre parfait partiellement ordonné est aussi appelé *tas*. Le minimum de l'ensemble représenté par un tas est donc à la racine, d'où la facilité d'accès annoncée.

**Question 2** Définir une classe Java *Heap* permettant de manipuler cette structure de données. Elle comprendra notamment un constructeur pour le tas vide et une méthode renvoyant un booléen indiquant si le tas est vide ou non.

Le problème est donc maintenant d'ajouter ou de supprimer un élément $x$ à un tas, représenté comme indiqué dans un tableau $T$, tout en conservant une structure de tas.

### Adjonction d'un élément à un tas

On commence par rajouter l'élément $x$ comme feuille au dernier niveau de l'arbre puis on échange l'étiquette du nœud portant $x$ avec celui de son père jusqu'à ce que, pour respecter la structure de tas, $x$ ne soit pas inférieur à l'étiquette du père du nœud qui le porte. Par exemple, pour rajouter 4 au tas donné dans l'exemple ci-dessus, on obtient les étapes suivantes :



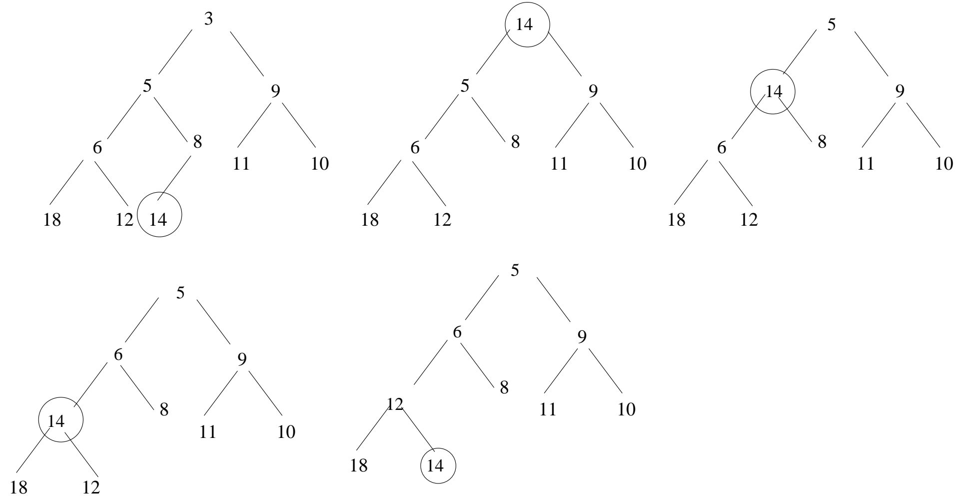
**Question 3**

a/Définir une méthode *adjonction* pour la classe *Heap*, ajoutant un nouvel élément $x$ au tas courant. On pourra utiliser $T[0]$ comme sentinelle.

b/ Evaluer l'algorithme dans le pire des cas.

### Suppression du minimum d'un tas

L'idée est de supprimer la dernière feuille du dernier niveau après avoir recopié son étiquette à la racine, puis, pour respecter la structure de tas, de faire redescendre cette valeur en la comparant au contenu de ses fils, par un processus inverse de celui de l'adjonction. La figure au verso illustre ce processus dans le cas de l'exemple.




**Question 4**

a/Définir une méthode $ExtractMin$ pour la classe $Heap$, supprimant le minimum du tas courant et retournant la valeur de ce minimum. On pourra utiliser $T[0]$ comme sentinelle.

b/ Evaluer l'algorithme dans le pire des cas.

On se propose maintenant d'utiliser cette classe pour trier des tableaux de nombres entiers.

## Transformation d'une liste en tas

*Dans ce qui suit,* $T[i\ ..\ j]$ *désigne la partie d'un tableau* $T$ *correspondant aux cellules dont l'indice est compris entre* $i$ *et* $j$*.*

**Question 5**

a/ Dans une nouvelle classe $HeapSort$, écrire une méthode transformant en tas une liste de taille $n$. Cette liste est supposée mémorisée à partir de la cellule d'indice 1 dans un tableau $T$ de taille $n + 1$ passé en paramètre à la méthode. L'algorithme consistera, en supposant qu'à la i$^{ieme}$ itération $T[1\ ..\ i]$ a déjà une structure de tas, à adjoindre à ce tas l'élément $T[i + 1]$.

b/ Calculer la complexité en temps dans le pire des cas de l'algorithme et montrer avec précision qu'il est en $\Theta(nlog(n))$.

## Tri par tas

Le tri par tas d'une liste à $n$ éléments consiste, après l'avoir transformée en tas, à supprimer le minimum du tas (le tas n'occupe plus alors que $T[1\ ..\ n-1]$), à recopier ce minimum en $T[n]$ et à recommencer l'opération sur $T[1\ ..\ n-2]$, $T[1\ ..\ n-3]$, . . . . La liste se trouve ainsi triée par ordre décroissant.

**Question 6**

a/ Ecrire une méthode *sort* qui réalise cet algorithme.

b/ Calculer la complexité en temps de cet algorithme.

c/ Qu'aurait-on dû faire pour obtenir une liste triée par ordre croissant?

d/ Il existe un algorithme permettant de transformer une liste en tas dont la complexité est en $\Theta(n)$. Cela change-t-il le comportement asymptotique de la complexité du tri par tas?

**Question 7** Définir une classe $HeapSortProg$ permettant de tester le programme précédent.