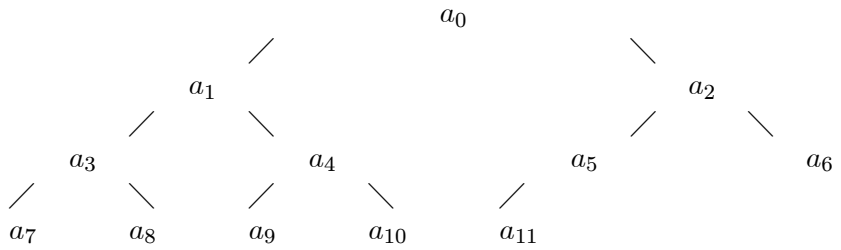
Interrogation d'informatique n°1. Tri par tas. *Barème sur 23 pts*

En informatique, un tas (*heap* en anglais) est une structure de données arborescente qui permet de retrouver directement le plus petit élément.

Un tas est un arbre binaire étiqueté (ici par des **entiers**) et presque complet à gauche. Un arbre binaire est dit presque complet à gauche si tous ses niveaux sont remplis, sauf éventuellement le dernier, qui doit être rempli sur la gauche. Ainsi, il existe au plus un sommet admettant un unique fils.

Dans un tas, **les sommets sont numérotés par hauteur croissante et de gauche à droite** :



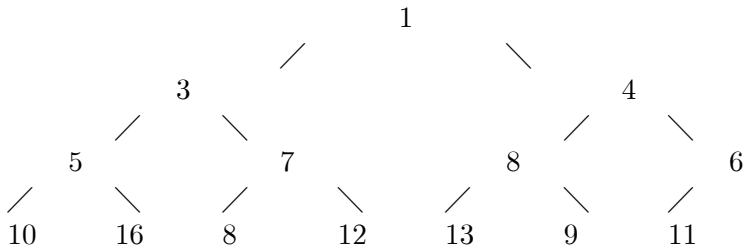
On peut en déduire (admis ici) que les deux fils du sommet d'indice i, s'ils existent, sont les sommets d'indices $2i+1$ et $2i+2$.

De plus, dans un tas, les étiquettes a_i sont ordonnées par ordre croissant le long des branches : on a $x \leq y$ si y est un fils de x. **Ainsi, un tas est codé informatique par une liste d'entiers $L = [a_0, ..., a_{n-1}]$ qui vérifie les propriétés suivantes :**

$$\forall i, \quad \begin{cases} a_i \leq a_{2i+1} \text{ lorsque } 2i+1 < n \\ a_i \leq a_{2i+2} \text{ lorsque } 2i+2 < n \end{cases}$$

En particulier, la racine du tas (sommet d'indice 0 et d'étiquette a_0) est l'élément minimal.

Exemple de tas d'entiers (représenté ici par un arbre binaire étiqueté par les éléments du tas) :



Ce tas est donc codé par la liste $L = [1, 3, 4, 5, 7, 8, 6, 10, 16, 8, 12, 13, 9, 11]$ de longueur $n = 14$.

Les opérations usuelles sur un tas sont les suivantes :

- *suppression de la racine* (et reconstitution de la structure de tas)

- *ajout d'un nouvel élément* (et reconstitution de la structure de tas)

La hauteur d'un sommet de l'arbre est sa distance à la racine (c'est-à-dire la longueur de la branche qui le relie à la racine). Par exemple, a_1 et a_2 (cf schéma ci-dessous) sont les sommets de hauteur 1.

La hauteur p d'un l'arbre est la hauteur maximale de ses sommets, c'est-à-dire à dire la longueur maximale de ses

1) a) [0.5 pt] Considérons un arbre binaire complet (de hauteur assez grande) où les sommets sont numérotés par hauteur croissante et de gauche à droite (comme dans un tas). Donner le nombre de sommets de hauteur $\leq p$.

Par exemple, il y a 3 sommets de hauteur ≤ 1.

b) [1.5 pt] En déduire que dans un tas à n sommets et de hauteur p, on a $p = \lfloor \log n \rfloor$, où log désigne le logarithme en base 2.

2) [1.5 pt] Définir trois fonctions `G(i)` , `D(i)` et `P(i)` qui étant donné un entier i renvoie les indices du fils gauche, du fils droit et du père du sommet d'indice i dans un tas.

On suppose ici que ces sommets existent. Par exemple, `G(i)` renvoie $2i + 1$.

Pour définir $P(i)$, utiliser l'opérateur de division euclidienne //. On prend par convention $P(0) = 0$.

3) Ajout d'un élément dans un tas par "percolation ascendante"

On commence par ajouter le nouvel élément x **à la fin de la liste** (en position n).

Puis on compare x à son père y et on les échange si $x < y$.

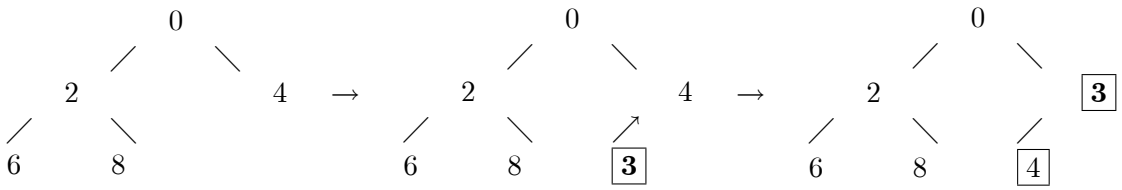
Et on itère le procédé :

Tant que x n'est pas positionné à la racine et que x est inférieur à son père y, on échange x et y.

On admet que le procédé aboutit bien à un tas.

[2.5 pts] Ecrire une procédure `ajout(L,x)` qui, étant donnés un tas représenté par la liste L et un entier x modifie L de sorte à ajouter x au tas L et à conserver la structure de tas.

Ainsi, si $L = [0, 2, 4, 6, 8]$, l'instruction `ajout(L,3)` modifie L en $[0, 2, 3, 6, 8, 4]$:



4) Suppression de la racine dans un tas L par "percolation descendante"

a) [1.5 pt] Définir une fonction `minFils(i,L)` qui étant donné un tas de taille n (codé par L) et un entier i vérifiant $0 \leq i < n$, renvoie l'indice du fils du sommet d'indice i ayant la plus petite étiquette.

Si le sommet i n'a pas de fils dans le tas, la fonction renvoie i.

Ainsi, `minFils(i,L)` renvoie toujours une des trois valeurs : i, $2i + 1$ ou $2i + 2$.

b) [3 pts] L'algorithme de suppression de la racine est le suivant :

- Si le tas est composé d'un unique élément, le tas L devient vide.

Sinon :

- On commence par remplacer la racine par le dernier élément x de la liste (la liste perd donc un élément).

- Tant que x n'est pas une feuille et que x est supérieur à moins un de ses fils, on échange x avec le plus petit de ses deux fils (ou bien avec son unique fils s'il n'en admet qu'un).

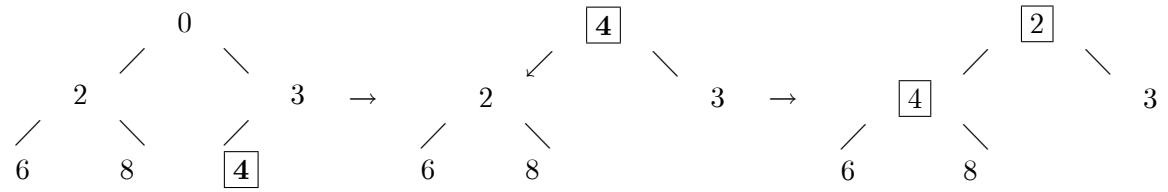
On admet que le procédé aboutit bien à un tas.

Définir une procédure-fonction `supprime(L)` qui :

- d'une part modifie le tas en supprimant la racine r (et en conservant la structure de tas)

- d'autre part renvoie r (la valeur supprimée).

Par exemple, si $L = [0, 2, 3, 6, 8, 4]$, `supprime(L)` modifie L en `[2,4,3,6,8]` et renvoie 0 :



5) [1 pt] Exprimer la complexité de `ajout` et `supprime` en fonction du nombre n d'éléments du tas.

6) a) [1.5 pt] Ecrire une fonction `creerTas(M)` qui étant donnée une liste M d'entiers crée un tas contenant les éléments de M. Préciser sa complexité en fonction de la longueur n de M.

b) [1.5 pt] En déduire une fonction `tri(M)` qui étant donnée une liste M d'entiers naturels, renvoie la liste triée de ses éléments en $O(n \log n)$ opérations.

Les deux questions suivantes sont des compléments et sont indépendantes l'une de l'autre..

7) Dictionnaire associé à un tas

On suppose ici que dans les tas considérés, toutes les étiquettes sont distinctes.

Pour certaines opérations supplémentaires, il est intéressant de disposer, pour chaque étiquette présente dans le tas, de l'indice du tableau qui lui correspond et de pouvoir accéder *en temps constant amorti* $O(1)$ à cet indice.

a) [1.5 pt] Définir une fonction `dictionnaire(L)` qui étant donné un tas L renvoie un dictionnaire D de sorte que pour toute étiquette x du tas, $D[x]$ renvoie l'entier i tel que $L[i] = x$.

Dans la suite, on dira que D est le dictionnaire associé à L.

On demande une fonction de complexité $O(n)$, où n est la longueur de L.

b) [2 pts] Ecrire une procédure `ajout(x,L,D)` qui étant donnés :

- un tas L
- le dictionnaire D associé à L
- un entier x

modifie le tas **et** le dictionnaire de sorte à ajouter x au tas s'il n'est pas déjà une étiquette de L, et ne fait rien

c) *Question supplémentaire*

Ecrire une procédure `modifier(x,y,L,D)` qui étant donnés :

- un tas L
- le dictionnaire D associé à L
- deux entiers x et y, où x est une étiquette du tas

modifie le tas et le dictionnaire de sorte à remplacer x par y dans le tas.

On demande une complexité en $O(\log n)$.

8) (★) On se propose dans la suite d'améliorer la fonction du 6) a).

Le principe est de construire la structure du tas en procédant par dichotomie.

Pour simplifier la description, on note $M\langle i\rangle$ le sous-arbre issu du sommet i : il s'agit donc de la partie de la liste M correspondant aux descendants du sommet i, y compris le sommet i lui-même.

a) [2 pts] Définir une procédure **récursive** `percol(M,i)` qui :

- suppose que dans la liste M, les sous-arbres $M\langle 2i+1\rangle$ et $M\langle 2i+2\rangle$ sont des tas (s'ils existent)
- modifie M par **percolation descendante** de sorte à transformer $M\langle i\rangle$ en un tas (ayant les mêmes éléments).

La procédure utilise $O(\log m)$ opérations, où m est le nombre d'éléments du sous-arbre $M\langle i\rangle$.

Remarque : Lorsque le sommet d'indice i est une feuille, `percolation(M,i)` ne modifie rien.

b) [1.5 pt] En déduire une procédure `tasPartiel(M,i)` qui étant donnés une liste M de longueur n et un entier $i < n$ modifie les éléments du sous-arbre $M\langle i\rangle$ de sorte à obtenir (uniquement) pour $M\langle i\rangle$ une structure de tas.

Donner pour conclure la nouvelle fonction `creerTas` en utilisant `tasPartiel`.

c) [1.5 pt] On note $c(n)$ le nombre d'opérations effectuées par la nouvelle fonction `creerTas`.

On se limite désormais aux cas où $n = n_p = 2^p - 1$ (arbre binaire complet).

A chaque étape, on effectue la construction du tas sur les deux sous-arbres (complets eux-aussi), puis on obtient la structure de tas de l'arbre complet en utilisant une percolation descendante en $O(p)$ opérations.

On a donc $c(0) = 1$ et $c(n_p) \leq 2c(n_{p-1}) + Kp$ pour tout entier $p \in \mathbb{N}^*$, où $n_p = 2^p - 1$ et K constante.

Trouver une constante L telle que $c(n) \leq Ln$ pour tout entier n de la forme n_p.