1 Problématique

Trier efficacement est une tâche fondamentale en algorithmique.

Peut-on trier un tableau efficacement en utilisant les propriétés des arbres binaires ?

2 Un tas

2.1 Définition

Un **arbre partiellement ordonné** est tel que la valeur de chaque nœud fils est inférieur au nœud père (figure 8).

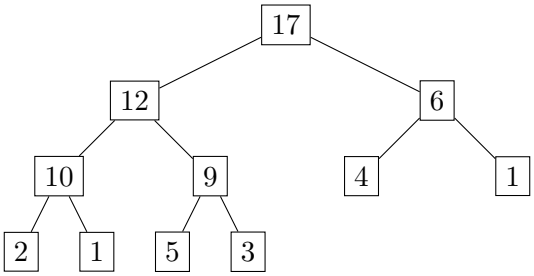

FIGURE 1 – Arbre partiellement ordonné

Un arbre partiellement ordonné n'implique pas que tous les éléments soient ordonnés.

Un **tas** est un tableau dont l'arbre associé est un arbre partiellement ordonné (code 1).

```
1 tas = [17, 12, 6, 10, 9, 4, 1, 2, 1, 5, 3]
```

Code 1 – Tas associé à l'arbre 8

Chaque nœud fils est accessible en respectant la convention suivante :

— l'indice de la racine est 0,

— l'indice du fils gauche est $2 * i + 1$,

— l'indice du fils droit est $2 * i + 2$.

Activité 1 : Écrire la fonction **echanger(t : list, i1 : int, i2 : int) → None** qui inverse les éléments d'indice *i1* et *i2* du tableau *t*.

2.2 Tamiser un élément du tableau

Un tableau n'est pas nécessairement un tas (figure 2).

Christophe Viroulaud

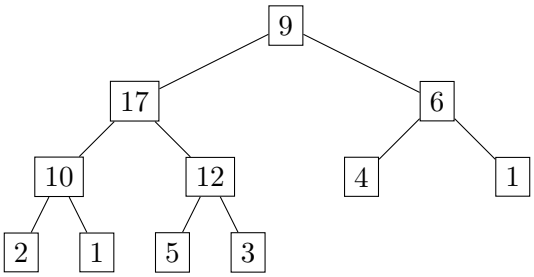

FIGURE 2 – La racine ne respecte pas les propriétés du tas

Tamiser un élément du tableau consiste à faire respecter la propriété d'un arbre partiellement ordonné pour ce nœud. Prenons l'exemple du nœud racine contenant la valeur 9. Pour respecter la propriété il faut échanger la valeur du nœud père (9) avec la valeur de son plus grand fils (17).

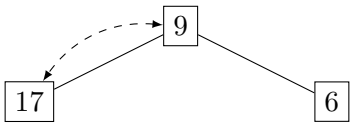

FIGURE 3 – Échange du père avec son plus grand fils.

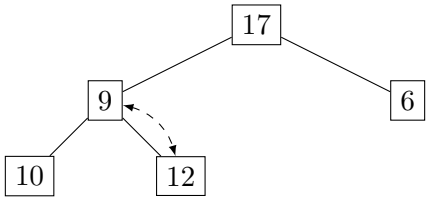

FIGURE 4 – Propagation récursive

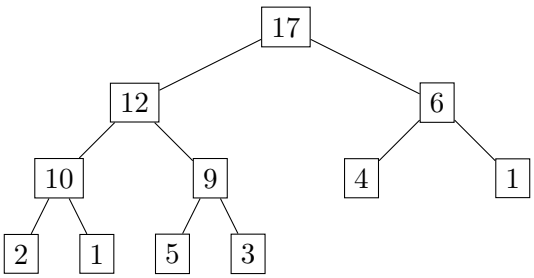

FIGURE 5 – La valeur 9 est à sa place

Activité 2 :

1. Écrire la fonction *récursive* **tamiser(t : list, i_pere : int) → None** qui positionne correctement le nœud d'indice *i_pere*.
2. Tester avec le tableau associé à la figure 2.
3. Modifier la signature de la fonction tel que **tamiser(t : list, i_pere : int, i_max : int) → None** afin qu'elle ne tamise que jusqu'à l'indice *i_max* (inclus) et non plus jusqu'à la fin du tableau. Cette propriété nous sera utile lors du tri.

2.3 Entasser le tableau

En tamisant chaque élément du tableau, on obtient un tas. Une bonne approche est de commencer par le bas de l'arbre. Ainsi, les sous-arbres droit et gauche respectent toujours la propriété d'un arbre

Christophe Viroulaud

partiellement ordonné. De plus, une feuille respectant nécessairement la propriété, il est judicieux de ne commencer qu'au premier nœud qui n'est pas une feuille.

Activité 3 : Écrire la fonction **entasser(t : list) → None** qui transforme le tableau t en tas, en tamisant chaque nœud (qui n'est pas une feuille).

3 Principe du tri par tas

Dans un tas la racine contient la plus grande valeur. Il suffit alors d'inverser cette valeur avec la dernière du tableau pour la placer définitivement. Il suffit ensuite d'appliquer le même principe au reste du tableau.

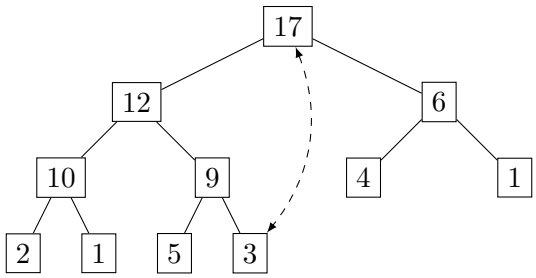

FIGURE 6 – Placer le plus grand élément en fin.

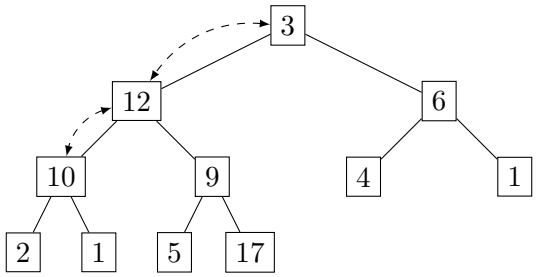

FIGURE 7 – Tamiser.

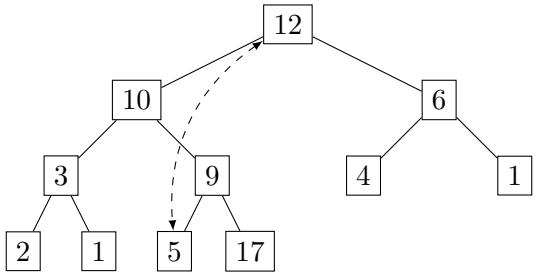

FIGURE 8 – Recommencer.

Activité 4 : Écrire la fonction **tri_par_tas(t : list) → None** qui implémente l'algorithme ci-après :

- Entasser le tableau.
- Initialiser indice_dernier.
- Tant que indice_dernier ≥0
  - Échanger le premier élément avec l'élément d'indice indice_dernier.

Christophe Viroulaud

- Tamiser le tableau de la racine jusqu'à l'élément d'indice indice_dernier.
- Décrémenter indice_dernier.

Ce tri a une complexité temporelle en $O(n.log(n))$

Christophe Viroulaud