Feuille de TD n°1

Exercice 1. Le but de cet exercice est de montrer qu'un algorithme de tri par comparaisons et échanges a pour complexité dans le pire cas au mieux $O(n \log n)$, où n désigne la longueur du tableau à trier.

1. De combien de manières peut-on permuter les éléments d'un tableau $[u_1, u_2, \ldots, u_n]$?

2. Supposons que l'on applique un algorithme de tri donné à $[u_1, u_2, \ldots, u_n]$. Pour chaque comparaison effectuée au cours de l'algorithme, il y a deux possibilités : soit on échange les éléments comparés, soit on ne les échange pas.

Supposons que, au maximum, l'algorithme s'exécute en $C(n)$ comparaisons ($C(n)$ est donc la complexité dans le pire cas). Montrer qu'il y a au plus $2^{C(n)}$ permutations σ différentes telles qu'il existe un tableau $[u_1, u_2, \ldots, u_n]$ pour lequel le tableau ordonné est $[u_{\sigma(1)}, u_{\sigma(2)}, \ldots, u_{\sigma(n)}]$.

3. En déduire que $2^{C(n)} \geqslant n!$.

4. En utilisant la formule de Stirling

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n,$$

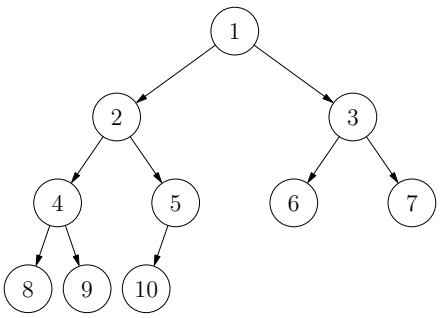
montrer qu'asymptotiquement, la complexité dans le pire cas est supérieure à un $O(n \log n)$.

Rappel. On dit que deux suites (u_n) et (v_n) sont équivalentes et on note $u_n \sim v_n$ s'il existe une suite (ε_n) qui tend vers 1 et telle que $\forall n, u_n = \varepsilon_n v_n$. On dit que (u_n) est un « grand O » de (v_n) on note $u_n = O(v_n)$ s'il existe une suite (b_n) bornée et telle que $\forall n, u_n = b_n v_n$.

Exercice 2. Déterminer la place mémoire nécessaire pour chacun des tris vus en cours :
- le tri par sélection,
- le tri par insertion,
- le tri fusion.

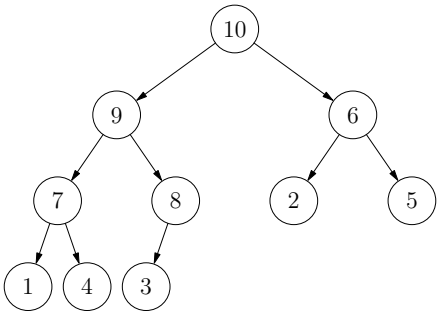
Exercice 3. Calculez la complexité du tri fusion.

Exercice 4 (tri par tas)**.** Étant donné un tableau, on peut le représenter sous forme d'un arbre binaire de la façon suivante. On place le premier élément à la racine de l'arbre, puis les deux éléments suivants sur les deux fils de la racine, puis les 4 éléments suivants sur les 4 petits-fils, etc. Si la taille du tableau n'est pas de la forme $2^k - 1$, on remplit la dernière ligne de l'arbre en commençant par la gauche. Ainsi, par exemple, le tableau $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ sera-t-il représenté comme suit :



1. Si on numérote les éléments d'un tableau à partir de 0, quels sont les numéros des fils de l'élément numéro i ?

On appelle *tas* un tableau représenté comme ci-dessus sous forme d'un arbre binaire, et tel que chaque noeud est plus grand que ses deux fils. Par exemple, le tableau $[10, 9, 6, 7, 8, 2, 5, 1, 4, 3]$ est un tas, comme on peut le vérifier sur sa représentation sous forme d'arbre :



Le tri par tas est un algorithme de tri qui utilise la structure de tas. Un des avantages de cet algorithme est qu'on peut le programmer *en place*, c'est-à-dire qu'il n'utilise pas plus de place mémoire que celle nécessaire pour stocker le tableau à trier. Un autre avantage est qu'il permet de faire un tri dynamique, c'est-à-dire que l'on peut ajouter au cours de l'algorithme de nouveaux éléments à trier.

On se donne un tableau d'entiers de taille n. Celui-ci sera modifié directement au cours de l'algorithme. On considérera un tas qui sera constitué des k premiers éléments du tableau, k pouvant varier entre 1 et n.

La première étape consiste, étant donné un tableau quelconque, à le transformer en tas. Pour cela, on construit le tas progressivement, en y ajoutant les éléments un à un (k variera de 1 à n), et en les « remontant » progressivement s'ils sont mal placés : si l'élément que l'on vient d'ajouter est plus grand que son père, on l'échange avec celui-ci, et ainsi de suite.

2. Montrer que si A est un tas, et qu'on ajoute à A un élément à la dernière place libre, alors, l'arbre obtenu en échangeant cet élément génération par génération avec ses ancêtres plus petits que lui est à nouveau un tas.

3. Montrer que si un tableau est un tas, alors son premier élément est aussi son élément maximal.

Une fois que l'on a transformé notre tableau en tas, la procédure est la suivante : on échange le premier et le dernier élément du tas, et on diminue k de 1 (ainsi, l'élément qui était en tête et que l'on vient de placer à la fin du tas n'en fait plus partie : on n'y touche plus jusqu'à la fin de l'algorithme). Les k premiers éléments du tableau ainsi obtenu ne forment en général plus un tas : l'élément que l'on vient de déplacer de la dernière à la première position est en général mal placé. On « descend » alors celui-ci en l'échangeant avec celui de ses fils qui est le plus grand, et ainsi de suite, jusqu'à ce que ses deux fils soient plus petits que lui. On recommence cette étape jusqu'à ce que k soit égal à 1.

4. Soit A un tas. Montrer que si on retire son premier élément et qu'on le remplace par le dernier, puis que l'on descend comme explicité ci-dessus, alors l'arbre obtenu est à nouveau un tas.

5. Montrer qu'à la fin de cette procédure, le tableau est trié.

6. Déterminer la complexité de l'algorithme dans le pire cas.

Remarque : on a présenté ici une version de l'algorithme utilisant des arbres binaires. On peut écrire sur le même principe un algorithme utilisant des arbres m-aires, et ce pour tout entier $m \geqslant 2$. Le plus efficace est celui utilisant des arbres ternaires ($m = 3$).