Travaux dirigés n^o 5
Tas binaires

Un tas binaire ascendant (*max-heap*) est un tableau que l'on interprète comme un arbre binaire où chaque élément d'un nœud est **supérieur ou égal** à ceux de ses fils. Notez au passage la différence entre cet arbre-là et un Arbre Binaire de Recherche (ABR). Bien sûr, on peut aussi définir les tas binaires descendants (*min-heap*) de façon symétrique.

Exercice 5.1 — Primitives

Donnez les fonctions de base qui prennent en entrée l'indice i d'un élément d'un tas et qui permettent de *se déplacer* dans sa représentation sous forme d'arbre binaire (on ne se souciera pas ici des éventuels débordements) :

1. `gauche(...)`
2. `droit(...)`
3. `père(...)`

Exercice 5.2 — Tamisage

L'algorithme de tamisage, dont vous avez vu la version récursive `max_Heapify(T,n,i)` en cours, prend en entrée un tableau qui est un tas à sa première case près. Il retourne le tableau devenu tas-max après une application du tamisage.

1. Faites la trace de cet algorithme de tamisage sur le tableau suivant :

5	11	9	8	10	4	0	6	3	2	7	1

2. Donnez à présent une version itérative `max_Heapify(T)` de cette fonction ainsi que sa complexité.

Exercice 5.3 — Construction d'un tas-max

La fonction `build_max_Heap(T)` prend en entrée un tableau quelconque et en fait très efficacement un tas binaire.

1. Trouver une idée afin de transformer un tableau en tas-max sans avoir recours à la fonction `insertion_max_Heap(T,elm)`. Vous l'appliquerez par exemple au tableau suivant :

1	0	7	8	2	5	3	4	9	6

2. Ecrivez le pseudo-code de l'algorithme `build_max_Heap(T)` puis analysez sa complexité.

Exercice 5.4 — Tri par tas

La notion de tas mène à un algorithme de tri efficace appelé le tri par tas (*heapsort*).

1. Ecrivez l'algorithme du tri par tas `max_Heapsort(T)`. Pour cela, assemblez des fonctions vues précédemment afin de trier les éléments d'un tableau quelconque par ordre croissant.
2. Ce tri s'effectue-t-il *sur place* ? Indiquez sa complexité.

Exercice 5.5 — Insertion

Vous avez vu en cours la fonction `insertion_min_Heap(T,elm)` qui constitue une version itérative de l'insertion d'un élément dans un tas-min.
Cette fois, écrivez une fonction principale `insert_min_Heap(T,elm)` qui lance une sous-fonction récursive qui doit effectuer le même travail.