TP : Arbres
Tri par tas

Le but de ce TP est d'écrire un algorithme de tri de tableaux basé sur la notion de **tas**.

Notion de tas :

Définition 1

> Un ***tas*** est un arbre **binaire** tel que l'information contenue dans tout nœud est supérieure à celle contenue dans ses fils.

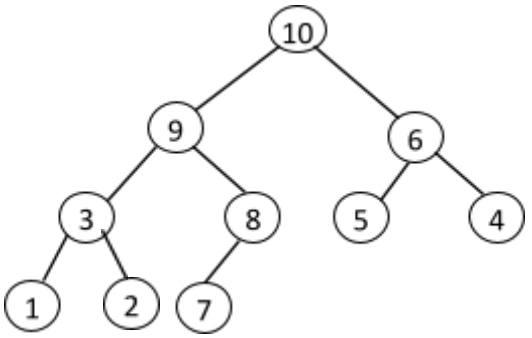


Figure 1 : Tas (maximier *ou tas-max*)

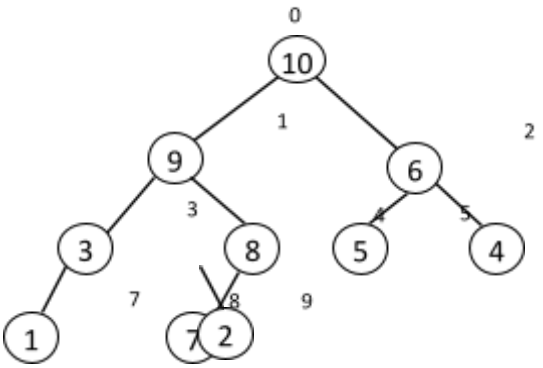
Remarque :

Dans le cas où la clé de chaque nœud est inférieure ou égale aux clés de chacun de ses fils, on parle de *tas-min*

Représentation du tas

On peut représenter ce tas par le tableau:

Indices :	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
T :	10	9	6	3	8	5	4	1	2	7



- Un élément d'indice ***i*** du tableau (***0*** à ***n*** (exclu)) est considéré comme un ***nœud*** de l'arbre ;
- son ***fils gauche*** s'il existe se trouve à l'indice **2*i+1**
- son ***fils droit*** s'il existe se trouve à l'indice **2*i+2**.
- Ce *nœud* est une ***feuille*** si **2*i+1** est supérieur ou égal à ***n***.

A. Fonctions `filsG(i)`, `filsD(i)` et `pere(i)`

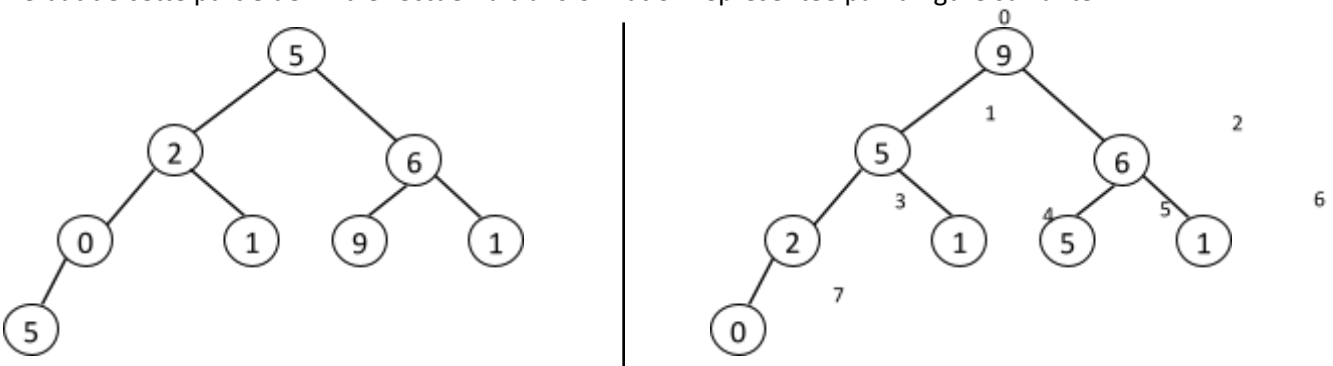
1. Ecrire la fonction ***`filsG(i)`*** qui retourne l'indice du fils gauche d'un nœud d'indice i.	**2.** Ecrire la fonction ***`filsD(i)`*** qui retourne l'indice du fils droit d'un nœud d'indice i.	**3.** Ecrire la fonction ***`pere(i)`*** qui retourne l'indice du père d'un nœud d'indice i.
`def filsG(i) :`	***`def filsD(i) :`***	***`def pere(i) :`***

B. Construction d'un tas à partir d'un tableau.

Définition2 :

Un tas est un tableau d'entiers tel que pour tous les indices ***i*** strictement positifs, la valeur de ***T[i]*** est inférieure ou égale à celle de ***T[pere(i)].***

Le but de cette partie de TP d'effectuer la transformation représentée par la figure suivante :



(a) *Arbre initiale* ***(b)*** *Tas obtenu par construction*

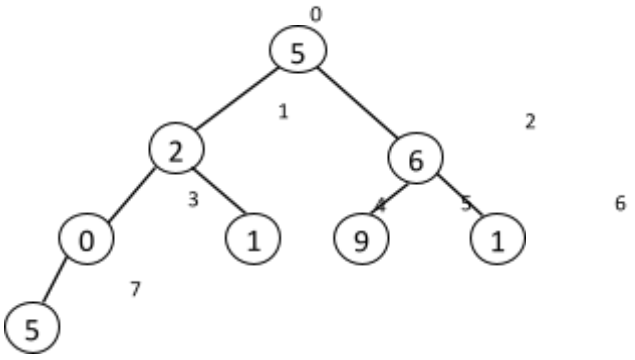
Figure 2 : Construction d'un Tas

4. Ecrire la fonction ***estUnTas(T)*** qui retourne ***True*** si le tableau T est un tas, ***False*** sinon.

Exemple : si ***T =***

Indices :	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	5	2	6	0	1	9	1	5

```
>>> estUnTas(T)
False
```




```
def estunTas(T) :
```

5. Avec les hypothèses **0≤i<limite et limite≤longueur(T)**, écrire la fonction ***maximum(T,i,limite)*** qui retourne :

$$le\ plus\ petit\ entier\ iMax\ tel\ que: \{\quad (iMax\ <\ limite) ET\ (iMax \in \{i, filsG(i), filsD(i)\})\ T[iMax] \geq T[i]$$

En d'autres termes, ***maximum(T,i,limite)*** retourne l'indice (inférieur à limite) de la plus grande des Trois valeurs ***T[i], T[filsG(i)]*** et ***T[filsD(i)].*** En cas de valeurs égales, le plus petit indice est retourné. Par exemple sur la figure 2a, ***maximum(T,0,8)=2, maximum(T,2,8)=5, maximum(T,3,8)=7*** et ***maximum(T,3,7)=3.***

```
def maximum(T, i, limite):
```

6. Soit la fonction récursive Python suivante :

```
def entasserRecursif(T,i,limite):
 iMax=maximum(T,i,limite)
 if iMax!=i:
       T[i],T[iMax]=T[iMax], T[i]
       entasserRecursif(T,iMax,limite)
```

a- Compléter l'arborescence avec les valeurs du tableau après l'appel **`entasserRecursif(T,0,8)`**

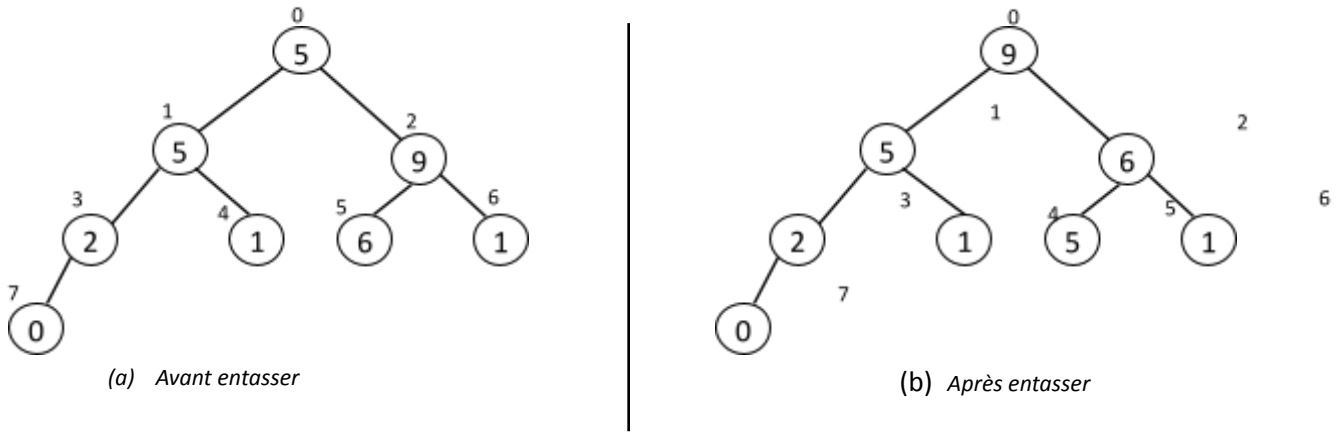


Figure 3 - Entasser(T,0,8)

b- Ecrire la fonction **`entasser(T,i,limite)`** équivalente utilisant un traitement **itératif**.

```
def entasser(T,i,limite):
```

Complexité :

7. L'algorithme **`entasser(T,i,limite)`** échange des valeurs du tableau de haut en bas, en suivant une branche de l'arborescence. Cela a pour effet de faire *descendre* des petites valeurs, et de faire m*onter* les grandes valeurs. Il est donc possible de construire un tas, en itérant cet algorithme sur les indices décroissants du tableau.

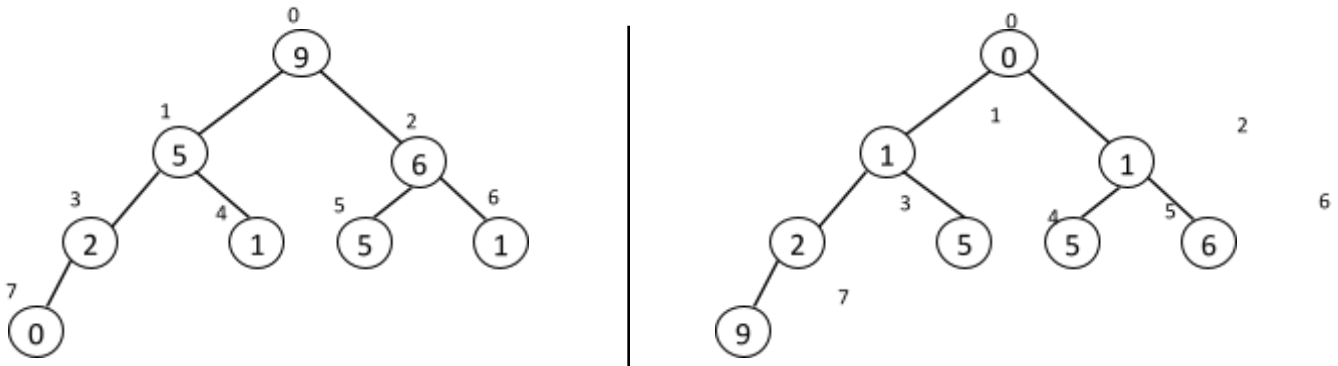
En utilisant **`entasserRecursif(T,i,limite)`** ou **`entasser(T,i,limite)`**, écrire la fonction **`construireTas(T)`** qui transforme un tableau en un tas.

```
def construireTas(T):
```

Complexité :

C. Tri d'un tas.

Le but de cette partie est d'effectuer la transformation représentée par la figure4.



(a) Tas initial

(b) *Vue arborescente du tableau trié*

Figure 4 – Tris d'un Tas

8. Dans un tas, la valeur maximale est à la racine de l'arborescence, donc en T[0]. Dans le tableau trié, cette valeur doit être en ***T[len(T)-1].***Il suffit donc d'échanger ces deux valeurs pour progresser vers la solution. Une fois cet échange fait, si l'on exclut la dernière valeur du tableau, le tas est peu changé. En fait ***entasser(T,0,len(T)-1)*** va créer un nouveau tas pour les valeurs du tableau dont les indices sont inférieurs à ***limite=len(T)-1***.Il suffit donc d'itérer ces deux étapes(échange, entasser)pour trier un tas.

Ecrire la fonction **`trierTas(T)`** qui transforme un tas en un tableau trié en ordre croissant.

```
def trierTas(T):
```

Complexité :

D. Tri d'un tableau à l'aide de tas.

9. Ecrire la fonction **`triParTas(T)`** qui trie un tableau d'entiers T en construisant d'abord un tas, puis en le triant.

Exemple :

```
>>> T=[5,2,6,0,1,9,1,5]
>>> estUnTas(T)
False
>>> triParTas(T)
>>> T
[0, 1, 1, 2, 5, 9, 6, 5]
>>> estUnTas(T)
False
```

Solution :

```
def triParTas(T):
```

Complexité :