TP nº 1 - Un peu de python, d'algorithmique et de complexité...

Exercice 1. *Échauffement et rappels de* Python

✎ Écrivez les fonctions suivantes en *Python* :
- `double(n)` qui renvoie le double d'un entier passé en argument ;
- `double_tab(t)` qui renvoie un tableau contenant le double de chaque valeur du tableau passé en argument ;
- `compare(x, y)` qui affiche « Égal. » si x est égal à y, « Plus grand. » si x est plus grand que y, et « Plus petit. » sinon.

Exercice 2. *Préparation*

1. Définissez les tableaux de taille 100 suivants, qui serviront à tester les différentes fonctions que vous écrirez par la suite :
- `t1` : les entiers de 0 à 99 ;
- `t2` : les entiers de 99 à 0 ;
- `t3` : une permutation aléatoire des entiers de 0 à 99.

Pour définir le tableau `t3`, il faut importer le module `random` (« `import random` ») et utiliser soit la fonction `random.randint(a,b)` qui renvoie un nombre entier aléatoire entre `a` et `b` (inclus), soit directement la fonction `random.shuffle(tab)` qui mélange un tableau `tab` passé en argument.

Exercice 3. *Minimax*

1. Écrivez les fonctions `max(tab)` et `min(tab)` qui renvoient respectivement l'élément maximum et minimum d'un tableau d'entiers `tab`.

2. Combien de comparaisons les fonctions `max` et `min` effectuent-elles en fonction de la taille du tableau en entrée ?

3. En utilisant les fonctions précédentes, écrivez une fonction `min_max(tab)` qui renvoie le couple `(min, max)` du plus petit et plus grand élément de `tab` (ça tient en deux lignes). Comptez le nombre de comparaisons effectuées par la fonction.

4. Améliorez la fonction suivante pour qu'elle ne fasse pas plus de 150 comparaisons sur les tableaux à 100 cases :
- on parcourt les éléments du tableau deux par deux ;
- on compare les deux éléments entre eux ;
- on compare le plus grand au maximum courant et le plus petit au minimum courant.

Vérifiez le nombre de comparaisons effectuées.

Exercice 4. *Trions ensemble*

On va s'intéresser ici à différentes méthodes pour trier un tableau.

1. Écrivez une fonction `tri_permut(tab)` qui trie le tableau `tab` *en place*, c'est-à-dire qu'il modifie directement `tab`, au lieu d'en faire une copie triée, en permutant les cases deux par deux : on cherche le plus petit élément et on le met sur la première case, puis le plus petit élément des cases restantes que l'on met dans la deuxième case, etc.

2. Estimez le nombre de comparaisons effectuées en fonction de la taille du tableau en entrée.

3. Écrivez une fonction `fusion(tab1, tab2)` qui prend en argument deux tableaux **que l'on suppose déjà triés** et qui renvoie un tableau contenant les éléments de `tab1` et `tab2` dans l'ordre.

4. En utilisant la fonction `fusion` précédente, écrivez une fonction `tri_fusion(tab)` qui trie récursivement un tableau de la manière suivante :
– si le tableau est de taille 1 ou 0, on le renvoie directement (il est forcément déjà trié) ;
– sinon, on coupe le tableau en deux moitiés, on trie chacune des moitiés à l'aide de la fonction `tri_fusion` (d'où le caractère *récursif* de la fonction) et on fusionne ces deux moitiés triées en utilisant la fonction `fusion`.

5. Combien de comparaisons doit-on faire pour trier un tableau à l'aide du tri fusion ?

6. Si vous voulez vous amuser, écrivez une fonction `tri_fouillis(tab)` qui mélange un tableau (à l'aide de la fonction `random.shuffle(tab)`) jusqu'à ce qu'il soit trié... (il faut écrire la procédure de test pour vérifier que le tableau est trié). Testez cette fonctions sur des **petits** exemples (pas plus de 10 cases) et estimez le nombre de comparaisons effectuées en moyenne.