TP : Tris

I) Préparation

Exercice 1 (Lire et comprendre un programme)
Que fait le programme suivant :

```
1 import time as t
2
3 def Mystere(fonction, arg_fonc):
4     debut = t.time()
5     fonction(arg_fonc)
6     fin = t.time()
7     return fin - debut
```

Vérifier que les résultats sont cohérent à l'aide de la fonction `sleep` de la bibliothèque `time`.

Exercice 2
À l'aide de la fonction `randint` du package `random` de la bibliothèque NumPy, écrire une fonction qui prend en argument un entier n et un entier $maxi$, et qui retourne une liste de taille n d'entiers tirés au hasard entre 0 et une valeur maximale $maxi - 1$.
On pourra donner la valeur par défaut 100 à $maxi$ via la syntaxe `(n, maxi=100)` dans la déclaration de la fonction.

II) Tri

Exercice 3 (Tri par insertion)

1) Écrire une fonction de tri par insertion.

2) Tester sur différentes listes d'entiers (toujours tester !).

3) Écrire une nouvelle fonction de tri par insertion qui affiche tous les états intermédiaires des variables. Avec, si possible, une présentation lisible et agréable ! Le tester pour une taille de liste et une valeur de $maxi$ qui rende le résultat lisible.

Rappel : `print(var1, var2)` affichera les contenus de `var1` et `var2` séparés par un espace. Par exemple `print("n =", n)`.

4) Tester le temps d'exécution de la fonction du 1) sur un échantillon de listes (tirées au hasard) en faisant varier n.

5) Écrire une fonction `echantillon(n_max, nb_n, nb_listes)` où les arguments représentent respectivement la valeur maximale de n, le nombre de valeurs de n testées (par exemple 10), et le nombre de listes testées pour chaque valeur de n.

Cette fonction retournera deux listes, d'une part les n testés, et d'autre part les temps trouvés pour les n correspondants.

6) A l'aide de la bibliothèque `matplotlib`, afficher les temps d'exécution en fonction de la taille n (si vous êtes en avance, et que vous ne vous souvenez plus de la syntaxe, cherchez sur le web).

7) Comparer avec le résultat de `L.sort` (graphiquement).

Exercice 4 (Tri rapide – Quicksort)
L'idée du quicksort est de choisir un pivot (par exemple le premier élément de la liste), puis de casser le reste de la liste en deux listes : ceux qui sont plus grand que le pivot, et ceux qui sont plus petit. Puis on réitère le procédé.

1) Écrire l'algorithme récursif correspondant.

2) Tester. Comparer les temps d'exécution avec ceux du tri par insertion, pour de petites ou de grandes listes. Afficher graphiquement à l'aide de la fonction de l'exercice précédent les deux courbes : pour le tri par insertion et pour le quicksort.

3) Rendre aléatoire le choix du pivot. Comparer la performance par rapport à un pivot non aléatoire. Dans quels cas est-ce intéressant ?

Exercice 5 (Complexité en temps avec un compteur)

1) Reprendre les algorithmes précédent en rajoutant un compteur, pour déterminer le nombre de comparaisons.

2) Comparer les courbes de ce compteur et les courbes du temps d'exécution. Est-il légitime de calculer la complexité temporelle en regardant ce compteur ?

3) Proposer une méthode pour vérifier expérimentalement (graphiquement ou non) que la complexité moyenne du tri par insertion (resp. quicksort) est en $O(n^2)$ (resp $O(n \ln n)$). Les plus motivés écriront sous forme d'une fonction qui puisse servir pour n'importe quel tri et n'importe quelle complexité conjecturée.

Exercice 6 (Tri fusion)

Ce tri se base sur la même idée que le tri rapide (quicksort), mais au lieu de séparer la liste à trier en 2, on fusionne 2 liste triée : on divise la liste de départ en 2 récursivement jusqu'à obtenir n listes à 1 éléments (qui sont donc triées), puis on remonte en fusionnant deux à deux.

1) Écrire une fonction qui fusionne deux listes *déjà triées*.

2) Écrire un algorithme de tri fusion, qui fait appel à la fonction précédente. On peut écrire un algorithme récursif ou non récursif, au choix.

3) Tester. Faire afficher le contenu des variables à chaque étape.

4) Comparer les temps d'exécution avec ceux des tris précédents. Conjecturer la complexité.

5) Écrire une version « en place » du tri : on se contente de permuter les éléments de la liste `L` sans créer de nouvelles listes.