TD4
Algorithmes de Tri

On veut trier une séquence de nombres entiers par ordre croissant in situ. Les fonctions de tri que vous allez écrire doivent être toutes enregistrées dans un fichier nommé `mesFonctionsTri.py` qui jouera le rôle de module externe. On rappelle que pour importer et utiliser les fonctions de `mesFonctionsTri.py` dans un autre fichier (tel que `TD4.py` par exemple), il suffit que les deux fichiers se trouvent dans le même dossier et d'écrire au début de `TD4.py` :

```
from mesFonctionsTri import *
```

Exercice 1: Tri par sélection

Le tri par sélection consiste à rechercher le plus petit élément de la séquence et à le mettre en position initiale (en l'échangeant avec le 1er élément), puis à réitérer en recherchant le 2eme plus petit élément pour le mettre en 2eme position....

1. Dérouler l'algorithme sur la séquence : (9 3 12 5 1)
2. Ecrire l'algorithme en pseudo-code.
3. Traduire cet algorithme en une fonction triSelection() qui prend une liste en argument et ne renvoie rien. Cette fonction va permettre de trier les éléments de la liste par ordre croissant.

Exercice 2 : Tri par insertion

Le tri par insertion consiste à trier les éléments d'une séquence au fur et à mesure de la lecture de cette séquence. Ainsi, à l'issue de l'itération k, les k premiers éléments de la séquence sont triés. On considère alors l'élément $k+1$ de la séquence en l'insérant à la bonne place parmi les k précédents. Cette insertion doit se faire par échanges successifs.

1. Dérouler l'algorithme sur la séquence : (9 3 12 5 1)
2. Traduire cet algorithme en une fonction triInsertion() qui prend une liste en argument et ne renvoie rien. Cette fonction va permettre de trier les éléments de la liste par ordre croissant.

Exercice 3 : Tri rapide ou quick sort

Le tri rapide est un tri récursif dont l'idée est la suivante : on commence par choisir aléatoirement un élément quelconque de la séquence (appelé pivot), on parcourt une fois le tableau de sorte à séparer les éléments inférieurs, égaux, et supérieurs au pivot. Cela fournit un tableau se découpant en trois parties : une première sous-séquence contenant les éléments inférieurs au pivot, une deuxième sous-séquence contenant les éléments égaux au pivot (dont le pivot lui-même), et une troisième sous-séquence contenant tous les éléments supérieurs au pivot. On recommence cette opération récursivement sur la première et la troisième sous-séquences jusqu'à ce qu'elles soient de longueur 1.

1. Écrire une fonction qui prend en argument une liste et deux indices `d` et `f`. Cette fonction choisit aléatoirement un indice de pivot entre les indices `d` et `f`, et réorganise la liste indicée de `d` à `f` de façon à classer en premier les éléments inférieurs, puis les éléments égaux, puis les éléments supérieurs au pivot. Cette fonction renvoie deux indices délimitant la séquence des éléments égaux au pivot.

2. Écrire une fonction récursive triRapide() qui prend en argument une liste et deux indices `d` et `f` et qui trie les éléments de la liste allant de d à f en utilisant la fonction écrite à la question précédente.

3. Tester votre fonction pour trier des listes générées aléatoirement de taille 1000, 10000 et 100000 contenant des entiers entre -50 et 50. Que constatez-vous ?

Exercice 4 : comparaison expérimentale des algorithmes

1. Créer le module `mesFonctionsTri.py` regroupant toutes vos fonctions de tri, que vous pourrez ainsi importer dans vos futurs programmes.

2. De retour dans Pyzo, écrire une fonction qui prend en argument une valeur entière n et renvoie une liste de taille n. Les éléments de la liste sont générés aléatoirement dans l'intervalle [-1000,1000].

3. Générer aléatoirement des listes de taille 1000 jusqu'à 10000 par pas de 1000. Pour chaque taille, générer 10 listes et calculer les temps moyens d'exécution de chacune des fonctions : `triSelection()`, `triInsertion()`, `triRapide()` et la méthode `sort` de Python (il sera difficile de faire mieux !). Stocker les résultats dans un fichier. L'affichage des résultats dans le fichier doit être de la forme :

Taille	TS	TI	TR	PS
1000	tps en s	tps en s	tps en s	tps en s
⋮				
10000	tps en s	tps en s	tps en s	tps en s

Figure 1: Temps de calcul moyen sur 10 instances/taille