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.

Correction

1. ```
   [1, 3, 12, 5, 9]
   [1, 3, 12, 5, 9]
   [1, 3, 5, 12, 9]
   [1, 3, 5, 9, 12]
   ```
2. On peut écrire du pseudo-code à divers niveaux de complexités. On peut par exemple résumer les fonctions essentielles de l'algorithme sans rentrer dans le détail :

   ```
   Algorithme : Tri selection
   Arguments : Liste T
   Renvoie : rien (modifie la liste en place)

   Pour cpt allant de 0 a n-2
       m <= min(T[cpt:])
       Echanger m et T[cpt]
   ```

   Ici, on voit que l'on est très proche de la description de l'algorithme donnée dans l'énoncé. On ne donne cependant aucune indication de comment le minimum est trouvé à chaque fois ni comment l'échange des valeurs dans le tableau est effectué. On peut donner plus de détails :
   ```

```
Algorithme : Tri selection
Arguments : Liste T
Renvoie : rien (modifie la liste en place)

Pour cpt allant de 0 a n-2
    m ← T[cpt]
    indMin ← cpt
    Pour k allant de cpt+1 a n-1
        Si T[k] < m alors
            m ← T[k]
            indMin ← k
        temp ← T[cpt]
        T[cpt] ← T[indMin]
        T[indMin] ← temp
```

On est ici beaucoup plus proche (voire quasi identique) à une implémentation en code (voir question suivante).

3.
```
def triSelection(T):
    for cpt in range(len(T)-1):
        # on cherche le min de T[cpt:] et sa position
        m = T[cpt]
        indMin = cpt
        for k in range(cpt+1, len(T)):
            if T[k] < m :
                m = T[k]
                indMin = k
        T[cpt], T[indMin] = T[indMin], T[cpt]   # on echange le terme a la
   position cpt et le min de T[cpt:]
```

On se rappelle qu'en Python, on peut s'affranchir de l'usage d'une variable `temp` comme on l'a écrit dans le pseudo-code.

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.

Correction

1.
```
[3, 9, 12, 5, 1]
[3, 9, 12, 5, 1]
[3, 5, 9, 12, 1]
[1, 3, 5, 9, 12]
```

2.
```
def triInsertion(T):
    for i in range(1, len(T)):
        j = i-1
        while j >= 0 and T[j+1] < T[j]:
```

```
            T[j], T[j+1] = T[j+1], T[j]    # on echange les elements aux
    positions j et j+1
            j = j-1
```

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. Dérouler l'algorithme sur la séquence (9 3 12 5 1) si on choisit comme indices de pivot 1 puis 0 à droite.
2. 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.
3. 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.
4. 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 ?

Correction

1. Le premier pivot est 3, il reste à la position d'indice 1 : `(1 3 9 12 5)`
   A gauche de 3, plus besoin de trier. A droite de 3, c'est-à-dire dans `(9 12 5)`, on choisit le pivot d'indice 0, donc 9. On réorganise pour obtenir `(5 9 12)` et la liste finale est donc `(1 3 5 9 12)`.
2. ```
   import random as rd

   def separation(S, d, f):
       # on choisit aleatoirement un pivot
       indPivot = rd.randint(d, f)
       pivot = S[indPivot]

       # on construit les listes avec les termes inferieurs, egaux et superieurs
       au pivot
       moins, egal, plus = [], [], []
       for x in S[d:f+1]:
           if x < pivot:
               moins.append(x)
           elif x == pivot:
               egal.append(x)
           else:
               plus.append(x)
       Sp = moins + egal + plus
   ```
   ```

```
    # on remplace la sous-liste entre d et f
    for k in range(len(Sp)):
        S[d+k] = Sp[k]
    return len(moins), len(moins+egal)
```

3. ```
def triRapide(S, d=0, f=-1):
    # pour eviter de donner d=0, f=len(S)-1 systematiquement
    if f == -1:
        f = len(S)-1

    # condition d'arret : la sous-liste est de longueur 0 ou 1
    if f-d < 2:
        return

    # on place le(s) pivot, on recupere son (ses) indice(s)
    m, p = separation(S, d, f)

    # on trie recursivement les 2 sous-listes separees par le(s) pivot(s)
    triRapide(S, d, d+m)
    triRapide(S, d+p, f)
```

4. ```
import time
import random as rd

for size in [1000, 10000, 100000]:
    l=[]
    for i in range(size):
        l.append(rd.randint(-50, 50))
    tps = time.time()
    triRapide(l, 0, len(l)-1)
    print('Tri rapide : ', time.time()-tps)
```

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

Correction

```
import matplotlib.pyplot as plt
import numpy as np

def listeAlea(n):
    l=[]
    for i in range(n):
        l.append(rd.randint(-1000, 1000))
    return l
    # return np.random.randint(-1000, 1000, size=n)

fich = open("tri_results.txt", 'w')
fich.write("Taille\tTS\tTI\tTR\tPS\n")

result = np.zeros((4,10))
cpt = 0
for n in range(1000, 10001, 1000):
    print(n, end=" ")
    fich.write(str(n))
    tps = np.zeros(4)

    # on repete 10 fois, on ajoute les temps de chaque algo de tri
    for i in range(10):
        l = listeAlea(n)

        t0 = time.time()
        triSelection(l[:])
        tps[0] = tps[0] + (time.time()-t0)

        t0 = time.time()
        triInsertion(l[:])
        tps[1] = tps[1] + (time.time()-t0)

        t0 = time.time()
        triRapide(l[:], 0, n-1)
        tps[2] = tps[2] + (time.time()-t0)

        t0 = time.time()
        l[:].sort()
        tps[3] = tps[3] + (time.time()-t0)

    # on moyenne les temps sur les 10 repetitions et on ecrit les resultats dans le
    fichier
    for k in range(4):
        result[k,cpt] = tps[k]/10
        fich.write("\t"+str(round(result[k,cpt], 4)))
    fich.write("\n")
    cpt += 1
fich.close()

x = range(1000, 10001, 1000)
plt.plot(x, result[0], label="Tri selection")
plt.plot(x, result[1], label="Tri insertion")
plt.plot(x, result[2], label="Tri rapide")
plt.plot(x, result[3], label="Python sort")
plt.legend()
plt.savefig('tps.jpg')
plt.show()
```

Résultats (en secondes) :

Taille	Tri selection	Tri insertion	Tri rapide	sort Python
1000	0.05	0.14	0.01	0.0
2000	0.21	0.57	0.01	0.0
3000	0.46	1.3	0.02	0.0
4000	0.92	2.38	0.02	0.0
5000	1.36	3.98	0.03	0.0
6000	1.9	5.89	0.05	0.0
7000	2.75	7.73	0.06	0.0
8000	3.48	9.73	0.06	0.0
9000	4.43	12.09	0.07	0.0
10000	5.35	15.01	0.07	0.01