Deux algorithmes simples de tri

Dans tout ce TD, il est interdit d'utiliser la fonction `sorted(tableau)` ou la méthode `tableau.sort()`.

Préparation

On se donne un entier n. Construire par compréhension :

1. Un tableau `tab1` composé des entiers de 1 à n dans l'ordre croissant.
2. Un tableau `tab2` composé des entiers de 1 à n dans l'ordre décroissant.
3. Un tableau `tab3` de longueur n d'entiers aléatoire compris entre 1 et $100 \times n$

In [ ]:
```
from random import randint
n = 10

tab1 = []
tab2 = []
tab3 = []
print(tab1)
print(tab2)
print(tab3)
```

Tri par sélection

Reprendre l'algorithme du cours et le programmer ci-dessous

In [ ]:
```
def triSelection(tab):
    """
    Donnée : prend un tableau tab de longueur n
    Résultat : renvoie le tableau trié par ordre croissant
    """

    return tab


print(triSelection(tab1))
print(triSelection(tab2))
print(triSelection(tab3))
```

Tri par insertion

Reprendre l'algorithme du cours et le programmer ci-dessous

In [ ]:
```
def triInsertion(tab):
    """
    Donnée : prend un tableau tab de longeur n
    Résultat : renvoie le tableau trié par ordre croissant
    """

    return tab

print(triSelection(tab1))
print(triSelection(tab2))
print(triSelection(tab3))
```

Complexité des tris par sélection et par insertion

On va rajouter une instruction permettant de mesurer le temps d'exécution de la fonction, calculé en moyenne sur un grand nombre de fois. Attention cela prend quelques secondes supplémentaires -voire quelques minutes-. Par défaut, la fonction est exécutée 10000 fois. On peut se limiter à 100 ou 1000 exécutions. Un exemple :

In [ ]:
```
import timeit

def test():
    triSelection(tab3)
    return

nombre = 10000
temps_total = timeit.timeit(stmt = test, number = nombre)
print("temps moyen : ",temps_total / nombre)
```

Remarque : une autre instruction donnant des résultats plus détaillés est :

```
%timeit (triSelection(tab3), number = 1000)
```

Dans l'affichage du temps, "mean" signifie moyenne et std. dev. signifie "écart-type"

Ecrire une fonction permettant de mesurer expérimentalement la complexité des algorithmes de tri par sélection et par insertion. Cette fonction renverra deux tableaux des temps moyens, suivant la taille du tableau à trier (un tableaupour le tri par sélection et un tableau pour le tri par insertion). On commencera par un tableau de taille 100, et on ajoutera 100 éléments de plus 10 fois (taille 200, 300 etc. jusqu'à 1100). On se limitera à 100 exécutions dans `timeit` .
Si nécessaire :

- on limitera la taille du tableau (par exemple on ne peut faire que 8 fois l'augmentation de la taille du tableau).
- faire à l'inverse une boucle plus ambitieuse pour le tri par insertion, dont les résultats expérimentaux sur le temps d'exécution sont très capricieux.

Instrumentez le programme : un `print` de chaque temps obtenu permet de voir s'il rame vraiment trop. Dans ce cas, on peut interrompre le noyau (Kernel -> interrupt dans le menu)

In [ ]:
```
def testSel():
    triSelection(tab3)
    return
def testInser():
    triInsertion(tab3)
    return

def mesure_complx():
    global tab3
    nombre = 100
    n = 100
    tempsSel = []
    tempsInser = []

    return tempsSel, tempsInser

tempsSel, tempsInser = mesure_complx()
print(tempsSel)
print(tempsInser)
```

Représentation et interprétation des résultats

Le programme ci-après donne les courbes obtenues. Conclure sur le lien entre la taille n des données du tableau, et le temps d'exécution.
On testera les deux tris, avec les trois tableaux. On fera un partage intelligent du travail avec ses voisins de droite et de gauche pour ne traiter qu'un seul cas par algorithme. Et on rédigera la réponse ci-après :

1. Tri par sélection :
   - Cas moyen (liste aléatoire)
   - Meilleur des cas (liste déjà triée)
   - Pire des cas (liste triée dans l'ordre décroissant)
2. Tri par insertion :
   - Cas moyen (liste aléatoire)
   - Meilleur des cas (liste déjà triée)
   - Pire des cas (liste triée dans l'ordre décroissant)

In [ ]:
```
import matplotlib.pyplot as plt

X = [100*(i + 1) for i in range(len(tempsSel))]
plt.plot(X,tempsSel, label = "tri par sélection, en moyenne")

plt.show()
```

<hr style="color:black; height:1px />