I Tris quadratiques

1.1 Tri par insertion

Exercice 1
Ecrire une fonction **insertion(L, x)** qui prend une liste triée L et un élément x et insert x dans L à la bonne place (de manière à ce que la liste obtenue soit encore triée).
On pourra utiliser

```
L.insert(indice, element)
```

qui fait exactement ce qu'elle prétend, en modifiant au passage la liste L.
Votre fonction n'a pas de besoin de valeur de retour, on modifie l'argument L passé (qui est mutable, donc les modifications seront visibles après la fin de l'appel).

Exercice 2
Ecrire une fonction **tri_insert1(L)** qui implémente le principe du tri par insertion et utilise la fonction précédente.

1.2 Mesurer le temps d'exécution

```
import time
help(time.time)
```

Cette fonction va nous aider à mesurer le temps d'exécution de nos différentes fonctions de tri.

Exercice 3
Créer une fonction **temps_tri(n, fonc)** qui prend un entier n comme argument et une fonction $fonc$ et mesure le temps moyen (50 essais, valeur à adapter) pour trier une liste aléatoire de longueur n mis par la fonction $fonc$.
Indication

```
import numpy as np
np.random.randint(0, M+1, n)
```

Exercice 4
Tracer l'évolution du temps de calcul en fonction de la longueur de la liste. On prendra des longueurs de liste de 50 à 500 par pas de 50.
Adapter ensuite votre fonction **temps_tri** pour obtenir la courbe du temps de calcul en fonction du temps de la méthode **.sort()** des listes.

1.3 Améliorons

Voici l'implémentation proposée en cours

```
def tri_inser2(L):
        n = len(L)
        for i in range(1, n):
                j = i - 1
                while j >= 0 and L[j] > L[j + 1]:
                        L[j], L[j + 1] = L[j + 1], L[j]
                        j = j - 1
```

Exercice 5
Tracer le graphique pour cette fonction. Comparer les temps d'exécution des deux tris par insertion.

Exercice 6
Trouver expérimentalement une constante α telle que la courbe de $y = \alpha x^2$ soit le plus proche possible du tracé de l'exercice précédent.
Question subsidiaire délicate : comment faire trouver α à python ?

Exercice 7
Implémenter l'amélioration vue en cours qui consiste non pas à échanger L[J] et L[$j+1$] mais à stocker L[i] avant la boucle while et faire simplement "descendre" L[j] puis à placer L[i] au bon endroit à la fin du while.

Exercice 8
Modifier les deux implémentations du tri par insertion pour obtenir comme valeur de retour (c, a) où c est le nombre de comparaisons entre éléments de L effectuées et a le nombre d'affectations dans L.
Tracer ensuite l'évolution de ces nombres en fonction de la taille de la liste triée pour les deux versions.

II Tri rapide

2.1 Algorithme

L'algorithme se décompose en plusieurs étapes :

1. Une liste de longueur 1 est toujours triée.
2. Pour une liste de longueur > 1 :
   (a) On choisi L[0] comme élément de référence appelé *pivot*.
   (b) On cré L_1 et L_2 qui sont composées respectivement des éléments $\leq$ pivot et $>$ pivot.
   (c) On tri récursivement L_1 et L_2.
   (d) On retourne $L_1 + [\text{pivot}] + L_2$.

2.2 Implémentation

Exercice 9
Créer en python la fonction **tri_rapide(L)**. On aura sûrement besoin d'une fonction intermédiaire.

Exercice 10
Tracer l'évolution du temps d'exécution en fonction de la taille des listes.

Exercice 11
A la place de choisir L[0] comme pivot, on prend la médiane des éléments de début, milieu et fin de la liste. Est-ce un meilleur choix en moyenne ?

Exercice 12
Trouver expérimentalement le moment ou le tri rapide devient plus efficace que le tri par insertion. Remplacer alors la condition de fin de récursion par un tri par insertion. Est-ce plus efficace ?