Exercice 1

Analyse de l'algorithme `tri1`

1.1 Appliquer `tri1`

On donne le script d'une fonction de tri, dont la classe de complexité est quadratique :

```
1 def tri1(L):
2     for j in range(len(L)):
3         temp = L[j]
4         i = j
5         while i>0 and L[i-1]>temp:
6             L[i]=L[i-1]
7             i-=1
8         L[i]=temp
```

a. Comment s'appelle cet algorithme de tri ?

b. Comment execute t-on le tri sur la liste `tab = [8, 5, 7, 3, 6, 2, 11]` en utilisant `tri1` ? Selectionner les instructions possibles :

- `tri1([8, 5, 7, 3, 6, 2, 11])`
- `L = tri1([8, 5, 7, 3, 6, 2, 11])`
- `L = [8, 5, 7, 3, 6, 2, 11]` puis `tri1(L)`
- `L = [8, 5, 7, 3, 6, 2, 11]` puis `L = tri1(L)`

c. Que vaut la liste `tab` après le premier passage dans la boucle externe `for` ?

d. Montrer l'évolution de la liste lors de son tri à l'aide de la fonction `tri1`.

1.2 Analyse

a. Montrer que pour `j = 1`, après le premier passage dans la boucle externe `for`, la liste est triée jusqu'au rang 1 inclus.

b. Supposons qu'à la fin du tour `j-1`, les valeurs sont triées jusqu'au rang `j-1` inclus. Montrer alors que, lors du tour `j`, la case `temp = L[j]` sera insérée correctement et que la liste sera alors triée jusqu'au rang j. On considèrera le cas : `L[j] >= L[0]`

La situation suivante pourra fournir les explications necessaires (`j = 4`).

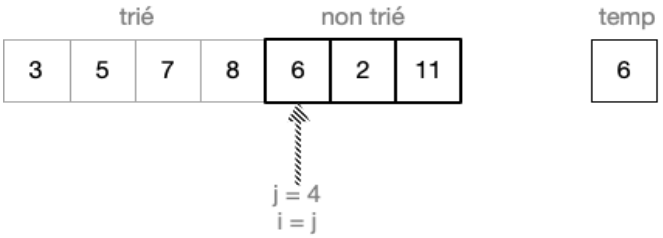

FIGURE 1 – liste triée jusqu'au rang j = 4

1.3 Complexité de l'algorithme de tri par insertion

On va dénombrer le nombre d'**affectations** réalisées pour trier la liste. Puis établir une loi recursive sur `n`.

a. Avec la liste triée jusqu'au rang `j-1 = 3`, combien d'affectations sont réalisées pour placer correctement la valeur 6 ?

b. Combien faudra-t-il d'affectations pour placer correctement la valeur 2 ?

c. On dispose maintenant d'une liste L de dimension `n`, qui est triée en sens inverse. On lui applique l'algorithme de tri par insertion. Il s'agit du *pire des cas* pour cet algorithme : Combien d'affectations sont réalisées pour trier toute la liste ?

1.3.1 Conclure que la complexité O(g(n)) est O(n^2)

Exercice 2

Tri par selection du plus grand élément

```
def tri2(T):
    for j in range(0,len(T)-1) :
        indiceDuMin = j
        for k in range(debut+1,len(T)) :
            if T[k]< T[indiceDuMin] :
                indiceDuMin=k
        if indiceDuMin ! = j :
            T[j],T[indiceDuMin]=T[indiceDuMin],T[j]
```

La fonction `tri2` est celle du tri par sélection du plus petit élément. Il est possible aussi de faire un tri par *sélection du plus grand élément*. On place alors systématiquement l'élément le plus grand en debut de liste. Celle-ci apparait alors triée à l'envers, du plus grand au plus petit.

2.1 Recopier et modifier le script de `tri2` pour réaliser le tri par selection du plus grand élement.

2.2 Pour le tableau suivant : Donner les états successifs du tableau à la fin de chaque étape du tri par *sélection du plus grand élément*. Préciser à chaque fois le nombre de **comparaisons** effectuées.

$t =$

0	1	2	3	4	5	6	7
T	I	M	O	L	E	O	N

FIGURE 2 – tableau à trier

2.3 Trier à l'aide d'une clé

On donne maintenant une table à trier. (prenom, age). Le tri se fera à partir d'une clé. Ici, ce sera l'âge des personnes. On veut trier cette liste par age décroissant. Ecrire le nouveau script qui devra réaliser ce tri, en s'inspirant du script `tri2`.

Exemple de table :

```
[('Kevin', 25), ('Marley', 18), ('Martine', 64), ('Thierry', 55), ('
   Felix', 13), ...]
```