Chapitre 3 - Tri par insertion

On s'intéresse dans ce chapitre à un autre algorithme classique, utilisé pour trier les éléments d'une liste : l'algorithme du tri pas insertion.

1- TRI PAR INSERTION D'UNE LISTE

Principe du TRI **par INSERTION** pour une liste de n valeurs :

1 ⇨ On commence le process en s'intéressant au 2ième élément d'index 1. On mémorise sa valeur dans une variable et on compare celle-ci à celles des éléments situés à gauche et donc d'index inférieur. Si ces éléments ont une valeur supérieure, ils sont décalés vers la droite (leur index augmente de 1). La valeur mémorisée est ensuite insérée avant le dernier élément décalé.

2 ⇨ On continue le process en s'intéressant au 2ième élément d'index 2, puis au 3ième et ceci jusqu'au dernier élément.

Exemple : Tri par insertion de la liste `[6, 5, 3, 1, 8, 7, 2, 4 ]`

	Etat de la liste							
Etape 1		5						
			3	1	8	7	2	4
Etape 2			3					
				1	**8**	**7**	**2**	**4**
Etape 3				1				
					8	**7**	**2**	**4**
Etape 4					8			
						7	**2**	**4**
Etape 4						7		
							2	**4**
Etape 4							2	
								4
Etape 4								4

Exercice : ⇨ Ecrire le script d'une fonction nommée *triInsertion()* qui trie la liste mise en argument en utilisant le principe du tri par insertion.
L'exécution du programme principal ci-contre, donnera dans la console :

```
# programme principal
l = [6, 5, 3, 1, 8, 7, 2, 4 ]
triInsertion(l)
print('Insertion :',l)
```

```
>>> (executing file "triInsertion.py")
Insertion : [1, 2, 3, 4, 5, 6, 7, 8]
```

⇨ Sauvegarder le code crée dans un fichier nommé *insertion.py* .

2- COMPLEXITE DE CET ALGORITHME :

Point Cours :

Pour trier une liste de taille n , on estime que le nombre d'opérations élémentaires réalisées dans le pire des cas est égal à

On dit que cet algorithme a une complexité de classe $\mathcal{O}(n^2)$ ou aussi que la complexité de cet algorithme est *quadratique*. Si on multiplie la taille de la liste à traiter par 10 par exemple, les temps de calcul seront « *à peu près* » multipliés par $10^2 = 100$.

3- VERSION ONLINE DE CET ALGORITHME DE TRI :

L'algorithme a la particularité d'être « *online* », c'est-à-dire qu'il peut recevoir la liste à trier, élément par élément pour les insérer au fur et à mesure. Parmi les différents algorithmes de tri existants, c'est le seul capable de faire cela.

On se propose ici d'écrire le script d'une fonction nommée *triInsertionOnline()* . Elle n'a aucun paramètre. Elle demande à l'utilisateur de saisir à chaque fois une valeur numérique qui est aussitôt insérée une liste. Le contenu de la liste est affiché dans la console, après chaque saisie. Si l'utilisateur saisi le string 'fin', le processus s'arrête et la fonction renvoie la liste constituée.

Cela donne par exemple :

```
>>> triInsertionOnline()
terme à saisir : 5
Insertion : [5]
terme à saisir : 9
Insertion : [5, 9]
terme à saisir : 1
Insertion : [1, 5, 9]
terme à saisir : -8
Insertion : [-8, 1, 5, 9]
terme à saisir : 25
Insertion : [-8, 1, 5, 9, 25]
terme à saisir : 6
Insertion : [-8, 1, 5, 6, 9, 25]
terme à saisir : fin
[-8, 1, 5, 6, 9, 25]
```

4- VERSION ONLINE AVEC SAISIE DE MOTS :

On écrit à présent le code d'une fonction nommée *triInsertionOnlineMots()* . Elle n'a également aucun paramètre. Elle demande à l'utilisateur de saisir à chaque fois un string qui est aussitôt inséré une liste. Le contenu de la liste est affiché dans la console, après chaque saisie. Si l'utilisateur saisi le string 'FIN', le processus s'arrête et la fonction renvoie la liste constituée.

Cela donne par exemple :

```
>>> triInsertionOnlineMots()
terme à saisir : soleil
Insertion : ['soleil']
terme à saisir : bien
Insertion : ['bien', 'soleil']
terme à saisir : chaleur
Insertion : ['bien', 'chaleur', 'soleil']
terme à saisir : zozotte
Insertion : ['bien', 'chaleur', 'soleil', 'zozotte']
terme à saisir : FIN
['bien', 'chaleur', 'soleil', 'zozotte']
```