COMPÉTENCE 10 : Tri par insertion

❖ Exercice 10.1 : Trions un peu ! (⋆)

Écrire une fonction `génerer_liste_alea(n,max)` qui renvoie une liste de taille n composée d'entiers pris aléatoirement entre 0 et max.

Écrire une fonction `tri_insertion(liste)` qui reçoit en argument une liste désordonnée de 10 entiers et retourne la liste triée.

❖ Exercice 10.2 : Petits comptages (⋆)

On souhaite faire le comptage expérimental du nombre de comparaisons du tri par insertion. Une astuce classique pour dénombrer les opérations d'un certain type (par exemple les comparaisons entre éléments de tableau) durant l'exécution d'un programme consiste à initialiser à 0 une variable globale, disons nb, puis à incrémenter nb à chaque opération devant être décomptée. Pour cela, il suffit de remplacer dans le programme le test de comparaison par un appel à une fonction qui renverra le même résultat, mais en incrémentant au passage la variable globale nb.

Comment mettre en place un compteur ?

1) **Implémenter le constructeur compteur() qui va renvoyer un objet inc**

```
def compteur():
    def inc(r):
        inc.count += 1
        return r
    inc.count = 0
    return inc
```

2) Créer un objet compteur dans la fonction de tri : `compteur1 = compteur()`
3) Appeler la fonction compteur1 à chaque comparaison en remplaçant

`tableau[j-1]>x` par `compteur1(tableau[j-1]>x)`

Mettre en œuvre cette idée pour tester le programme de tri par insertion de l'exercice 10.1. On comptera

- le nombre de comparaisons entre deux éléments du tableau, avec le compteur cptComp
- le nombre d'échanges, avec le compteur cptEch.

❖ Exercice 10.3 : Top chrono ! (⋆)

Comment mettre en place un chronomètre précis?

On peut aussi chronométrer le temps d'exécution d'un tri, en utilisant par exemple la nouvelle méthode (depuis Python 3.3) time.perf_counter (plus précis que time.process_time). Un appel à perf_counter() renvoie un flottant correspondant à l'instant dudit appel.

```
import time
# returned value is in fractional seconds
start = time.perf_counter()
# let's time this calculation
y = 2**6
end = time.perf_counter()
elapsed = end - start
print("elapsed time = {:.12f} seconds".format(elapsed))
```

Effectuer les comptages et mesurer le temps d'exécution sur un tableau de 10000 entiers compris entre 0 et 100. Attention, si vous souhaitez mesurer des durées, l'appel à la fonction de comptage ralentit le processus d'exécution !

❖ Exercice 10.4 : Complexité et tracé de graphes avec matplotLib (⋆)

1. On se place dans le meilleur des cas. Montrer que la complexité de l'algorithme de tri par insertion est en O(n), en considérant
   - uniquement le nombre de comparaisons entre deux éléments du tableau à trier,
   - uniquement le nombre d'affectations.
2. Même question en se plaçant dans le pire cas. Montrer que la complexité de l'algorithme de tri par insertion est en $O(n^2)$.
3. Proposer un exemple de tableau pour lequel le tri par insertion a un coût linéaire (meilleur cas).
4. Proposer un exemple de tableau pour lequel le tri par insertion a un coût quadratique (pire cas).
5. Écrire une fonction `genererMeilleurCas(n,max)` qui retourne un tableau de taille n comportant des entiers majorés par max.
6. Écrire une fonction `genererPireCas(n,max)` qui retourne un tableau de taille n comportant des entiers majorés par max. On pourra pour cela écrire une fonction `inverserTableau(tab)`.
7. Modifier la fonction `triInsertion(L)` de l'exercice 1 pour effectuer un comptage des comparaisons. On écrira ainsi une fonction `triInsertionAvecComptage(L)`.
8. Écrire une fonction `simulMeilleurCas(nbTab,n,max)` qui retourne deux listes ListeX et listeY telles que :
   En entrée :
   nbTab :nombre de tableaux générés,
   n : taille maximum des tableaux,
   max : valeur maximale des entiers à trier.
   En sortie :
   ListeX est une liste contenant la taille aléatoire des tableaux générés,
   ListeY est une liste contenant le résultat du comptage des comparaisons.

9. Écrire de la même manière une fonction `simulPireCas(nbTab,n,max)`.

10. Proposer un tracé matplotlib dans les deux configurations pour un ensemble de 100 tableaux d'entiers de taille aléatoire (de 0 à 100 éléments). Le résultat devrait être :

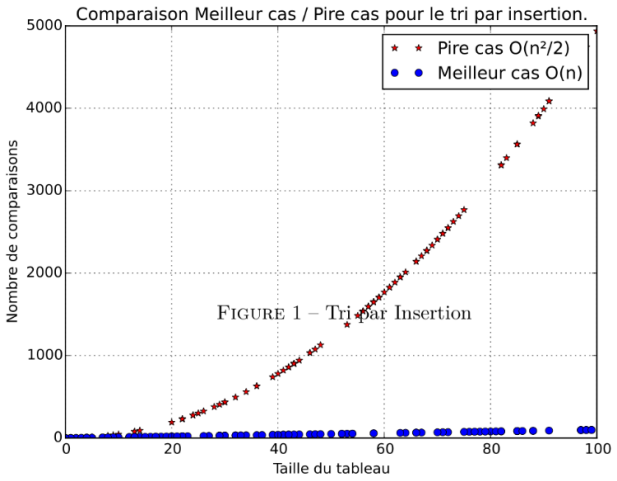

FIGURE 1 – Tri par Insertion