

## THÈME 3 : Algorithmes de tri (Partie 2)

Nous avons étudié un premier algorithme de tri la dernière séance. Cet algorithme est le tri par sélection. Toutefois, cet algorithme est très inefficace : il nous faut donc trouver un algorithme plus rapide.

Le but de ce TP est d'étudier un deuxième algorithme de tri classique appelé le tri par insertion.

### 1) Algorithme "avancé" : tri par insertion

L'algorithme de tri par insertion est plus avancé et permet d'obtenir de meilleures performances de tri. On vous donne ci-dessous l'algorithme de tri par insertion.

```
DEBUT
n ← longueur(L)
pour i allant de 1 à n-1:
    valeurTraitee ← L[i]
    j = i-1
    tant que j >= 0 et valeurTraitee < L[j]:
        L[j+1] = L[j]
        j = j-1
    L[j+1] = valeurTraitee
renvoyer L
FIN
```

Pour comprendre cet algorithme, il est **CRITIQUE** de le dérouler à la main.

#### Question 1

À l'aide du papier et d'un crayon et en utilisant le tableau algorithmique ci-dessous, appliquez cet algorithme sur la liste  $L=[10, 3, 7, 5, 6, 1]$ . En particulier, on suivra **l'évolution de L** à chaque passage dans les différentes boucles.

i	valeurTraitee	j	L[j]	j >= 0 et valeurTraitee < L[j]	L[j+1]	L

#### Question 2

- Montrez que la propriété "à l'étape k, les valeurs comprises entre 0 et k du tableau L sont triées dans l'ordre croissant" est un invariant de boucle.
- En déduire que cet algorithme réalise bien un tri d'un tableau dans l'ordre croissant.

#### Question 3

Reprenez le programme "theme3.py" déjà écrit au cours précédent.

Traduisez cet algorithme en Python en complétant la fonction `triInsertion`. Vous pourrez tester votre programme sur [PythonTutor.com](https://www.python-tutor.com/) pour observer le fonctionnement du programme.

#### Question 4

En vous rappelant du principe du tri par sélection et en utilisant la questions 1, expliquez le fonctionnement de l'algorithme de tri par insertion.

### Question 5

Testez votre programme sur les tableaux :

- ❖ L = [1, 5, 12, 8]
- ❖ L = [-3, -5, -12, -1]
- ❖ L = []

## 2) Comparaison d'algorithmes

### Question 1 — Complexité

Quelle sera la complexité de notre algorithme dans les cas suivants :

- ❖ L = [4, 2, 8, 7, 9, 3, 11]
- ❖ L = [11, 8, 6, 4, 2, 1]
- ❖ L = [1, 4, 5, 6, 9, 13]

### Question 2 — Complexité

Déduire de la question 2, la complexité du tri par insertion dans le meilleur et le pire des cas.

La librairie `timeit` permet de faire des mesures de vitesse d'exécution (voir Partie 1 pour une explication plus détaillée).

Python possède une fonction de tri interne. Cette fonction s'appelle `sorted` et s'utilise ainsi :

```
sorted([10, 3, 7, 5, 6, 1])    renvoie    [1, 3, 5, 6, 7, 10]
```

### Question 3

Affichez avec `print` les mesures de temps d'exécution des algorithmes de tri par sélection sur des tableaux de taille 100, 1000, 10000.

Vous pourrez vous inspirer de ce qui a déjà été écrit dans le code `theme3.py` pour le tri par sélection.

### Question 4

On dispose d'un tableau avec les tailles de nos tableaux.

Modifiez votre programme de manière à **enregistrer** vos mesures de temps dans un tableau appelé `tempsSelection`.

*Exemple :*

`tailleTableau = [ 10, 100, 1000, 10000 ]`

`tempsInsertion = [ 0,0001 , 0,01, 0,1, 1,2 ]`

`tempsSelection = [ 0,0001 , 0,001, 0,01, 0,2 ]`

`tempsSorted = [ 0,0001 , 0,002, 0,008, 0,1 ]`

*Cela se lit : pour un tableau de 10000 éléments, le tri par sélection a pris 0,2 seconde.*

### Question 5

À l'aide de la bibliothèque graphique `matplotlib`, définissez `x` et `y` correctement puis représentez l'évolution du temps de calcul en fonction de la taille du tableau à trier. La complexité est-elle conforme à vos attentes ?

On utilisera une échelle logarithmique<sup>1</sup> en remplaçant `plot` par `semilogx`, `semilogy` ou `loglog`.

### Question 6

Pensez-vous que le tri interne de Python soit le tri par sélection, par insertion ou un autre ? Pourquoi ?

---

<sup>1</sup> [https://fr.wikipedia.org/wiki/Échelle\\_logarithmique](https://fr.wikipedia.org/wiki/Échelle_logarithmique)