THÈME II : Algorithmes de tri

Nous avons vu que la recherche dichotomique permettait de rechercher de manière très efficace un élément dans une liste triée. Cet algorithme va donc être utile si nous disposons d'algorithmes permettant de trier rapidement des listes, si possible en complexité logarithmique. Mais cela est violemment hors-programme… Pour vous donner une idée, aller voir le nombre d'algorithmes de tri existant en tapant : "algorithmes de tri wikipedia".

Nous verrons cette année deux algorithmes de tri :

- le tri par sélection ;
- le tri par insertion.

1) Algorithme "naïf" : tri par sélection

Regardez la vidéo **SortingAlgorithms.mp3** que vous trouverez sur le GitHub afin de comprendre son fonctionnement dans le cas où on recherche l'élément le plus lourd.

- À partir de cette vidéo et de l'aide de votre professeur, écrivez l'algorithme du tri par sélection en langage naturel. On fera une recherche suivant l'élément le plus léger.
- Transcrivez votre algorithme en langage Python. Pour se faire, on complétera la fonction *triSelection* du programme "triListe.py" présent sur le GitHub.
- Quelle sera la complexité de cet algorithme dans le pire des cas ? Modifier votre code afin qu'il renvoie la liste triée ainsi que le nombre de passages dans la boucle interne.

2) Algorithme "avancé" : tri par insertion

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

```
VARIABLE
L : liste de nombres à trier

DEBUT
n ← longueur(L)
pour i allant de 1 à n-1:
  valeurTraitee ← L[i]
  j = i-1
  tant que j>=0 and 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.

- ❖ À l'aide du papier et d'un crayon, appliquez cet algorithme sur la liste L=[10, 3, 7 ,5 ,6, 1]. À l'aide d'un tableau contenant L, vous écrirez l'évolution de L à chaque passage dans les différentes boucles.

- ❖ Expliquez avec vos propres mots le fonctionnement de l'algorithme de tri par insertion.

- ❖ Dans le programme "triListe.py", traduire cet algorithme en Python en complétant la fonction `triInsertion`.

- ❖ Quelle sera la complexité de cet algorithme dans le pire des cas ? dans le meilleur des cas ? Conclusion ? À l'aide d'un compteur, modifier votre code afin qu'il renvoie la liste triée ainsi que le nombre de passages dans la boucle interne.

3) Comparaison d'algorithmes

La librairie time permet de faire des mesures de vitesse d'exécution.

a. En faisant varier nVal, réaliser quelques mesures de temps d'exécution de *triSelection*, *triInsertion* et de *sorted()*, la fonction de tri interne de Python.

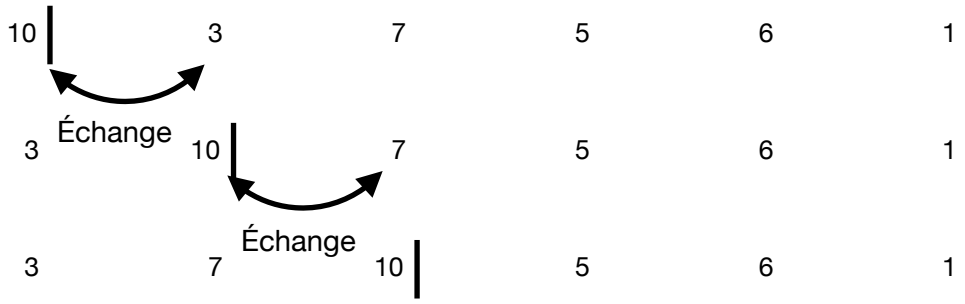
b. Pensez-vous que Python utilise le tri par sélection ou le tri par insertion ? Pourquoi ?

c. Modifier votre programme de manière à enregistrer vos mesures de temps dans quatre variables : `xVal`, `tInsertion`, `tSelection` et `tSorted`.

   Exemple :
   xVal = [10, 100, 1000, 10000]
   tInsertion = [0.0001 , 0.01, 0.1, 1.2]

d. En important matplotlib.pyplot, réaliser un graphique représentant vos mesures de temps en fonction de nVal. On pourra utiliser une échelle logarithmique en ordonnées.

Exemple d'organisation possible :



10 |	3	7	5	6	1
3	10 |	7	5	6	1
3	7	10 |	5	6	1