Thème 3 : 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 de programmes permettant de trier rapidement des listes d'objets (si possible en complexité logarithmique). Mais cela est violemment hors-programme… Pour vous donner une idée, allez voir le nombre d'algorithmes de tri existant en tapant : "algorithmes de tri wikipedia".

La problématique du tri est encore aujourd'hui largement étudiée en recherche informatique. Toute personne travaillant dans le domaine de l'informatique est amenée à travailler sur ce type d'algorithme. 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 bouillotvincent.github.io afin de comprendre son fonctionnement dans le cas où on recherche l'objet le plus lourd.

a) À partir de cette vidéo et de l'aide de votre professeur, écrivez l'algorithme du tri par sélection en langage naturel. On adaptera la recherche de la vidéo à l'objet le plus léger.

b) Transcrivez cet algorithme en langage Python. Pour se faire, on complétera la fonction `triSelection` du programme "theme3.py" présent sur le mon site web.

c) Modifier votre code afin qu'il renvoie la liste triée ainsi que le nombre de passages dans la boucle interne. Quel serait le cas le plus pénalisant (pire des cas) ? En déduire la complexité de cet algorithme dans le pire des cas.

2) 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 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.

a) À l'aide du papier et d'un crayon et en utilisant un tableau, 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.

b) Dans le programme "theme3.py", traduire cet algorithme en Python en complétant la fonction `triInsertion`. Vous pourrez tester votre programme sur PythonTutor.com pour observer le fonctionnement du programme.

c) En utilisant les questions a) et b), expliquez le fonctionnement de l'algorithme de tri par insertion.

d) 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**.
Quelle sera la complexité de cet algorithme dans le pire des cas ? dans le meilleur des cas ?
Conclusion ?

3) Comparaison d'algorithmes

La librairie `timeit` permet de faire des mesures de vitesse d'exécution (voir Thème 2 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]

a) Réaliser et afficher avec print les mesures de temps d'exécution des algorithmes de tri par sélection, insertion et sorted sur des tableaux de taille 100, 1000, 10000.

b) Modifier votre programme de manière à **enregistrer** ces mesures de temps dans trois variables : `tempsSelection`, `tempsInsertion` et `tempsSorted` en fonction de la variable tailleTableau.

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]

c) Réaliser un graphique représentant vos mesures de temps en fonction de tailleTableau. On pourra utiliser une échelle logarithmique double avec l'instruction loglog.

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

Exemple d’organisation possible :

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

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