Thème 3 : Algorithmes de tri (Partie 1)

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 simple

Question 1
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.

Question 2
Nous allons représenter les boites par un tableau de nombres dont les valeurs correspondent aux poids des pièces.

Complétez l'algorithme en pseudo-langage suivant :

```
Paramètre d'entrée :
boitePoids = Tableau des poids
boitePoidsTrié = Tableau des poids triés

On parcourt … … … … … … … … … … … … … … … … … … … :
     on trouve … … … … … … … … … … … … …
     on l'enregistre dans boitePoidsTrié.

On renvoie … … … … … … … … … …
```

Pour des raisons de simplicité d'écriture, nous allons chercher l'élément le plus léger.
La fonction `triSelectionSimple` du programme "theme3.py" (que vous téléchargerez depuis mon site web) permet de réaliser l'algorithme ci-dessus.

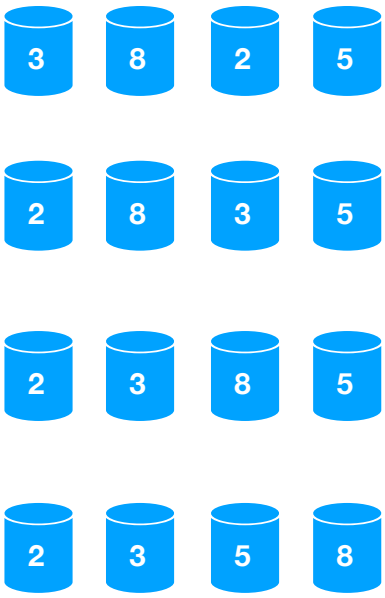
Question 3
Commentez chaque ligne de la fonction `triSelectionSimple` en la reliant à une étape de l'algorithme.

Question 4
La fonction `plusLeger` doit être programmée. Cette fonction prend en paramètres d'entrée un tableau de nombres entiers T et renvoie la position du minimum dans ce tableau.

2) Algorithme : tri par sélection

L'algorithme précédent a l'inconvénient d'utiliser deux tableaux, ce qui est inefficace en terme de cout mémoire. On se propose de trouve une méthode permettant de n'utiliser qu'**un seul tableau**.

Question 1
À partir du schéma du tri par sélection ci-dessous, expliquez ce qu'il se passe à chaque étape. En déduire quelle pourrait être la méthode nous permettant de nous passer d'un tableau.



Cas initial :
On trouve que 2 est le minimum du tableau. Il est en position 2.

Passage 1 dans la boucle :

....
....

Passage 2 dans la boucle :

....
....

Passage 3 dans la boucle :

....
....

Question 2
À partir du schéma ci-dessus, répondez à ces questions importantes :
- ❖ combien de passage(s) dans la boucle contient cet algorithme ?
- ❖ la recherche d'un minimum se fait-elle systématiquement dans tout le tableau ? Si non, à une étape quelconque k, à partir de quel position doit-on chercher le minimum ?
- ❖ quelle valeur est toujours échangée ?

Question 3
À partir de vos réponses, complétez l'algorithme ci-dessous :

```
Paramètre d'entrée :
boitePoids = Tableau des poids

On parcourt … … … … … … … … … … … … … … … … … … :
    on enregistre … … … … … … … … … … … … … … … … … … … … … … … … …
    on … … … … … … … … … … … … … … … … … … … … … … … … … … … … …
    on … … … … … … … … … … … … … … … … … … … … … … … … … … … … …

On renvoie … … … … … … … … …
```

Question 4
Pour que notre nouvel algorithme fonctionne, modifiez la fonction `plusLeger`.
Celle-ci doit maintenant trouver le minimum d'un tableau T pour des éléments compris entre la position idebut et la position len(T)-1. On utilisera le paramètre optionnel idebut.

Rappel : l'instruction range(a, b, p) permet de compter de a à b-1 par pas de p.

Question 5
À partir de l'algorithme de la question 3, du tri par Sélection simplifié de la partie 1) et en utilisant le paramètre optionnel de la fonction `plusLeger`, complétez la fonction `triSelection`.

Point technique : pour utiliser un paramètre optionnel, il suffit de lui donner une valeur.
Exemple :
`plusLeger([11,18,8,63])` trouve la plus petite valeur entre 11, 18, 8 et 63 (valeur par défaut 0) ;
`plusLeger([11,18,8,63], 2)` trouve la plus petite valeur entre 8 et 63.

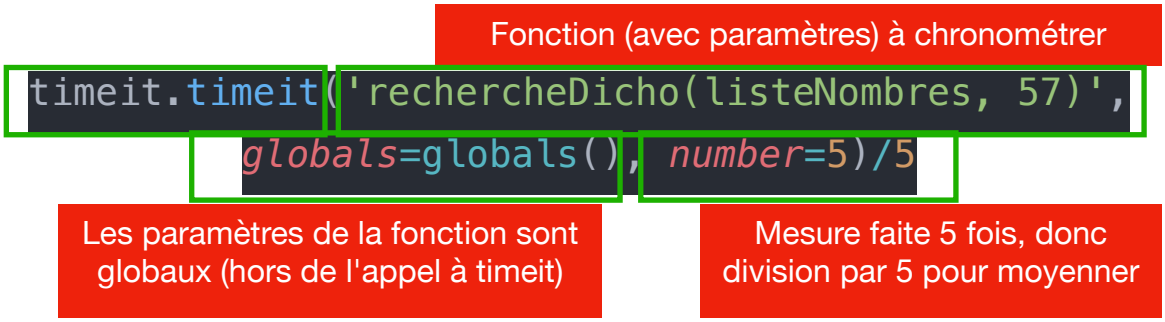
Question 6
Testez votre fonction sur quelques tableaux afin d'en vérifier le bon fonctionnement !

3) Complexité et mesures de vitesse d'exécution

Question 1 — Complexité
Quelle sera la complexité de notre algorithme dans les cas suivants :
- liste = [4, 2, 8, 7, 9, 3, 11]
- liste = [11, 8, 6, 4, 2, 1]
- liste = [1, 4, 5, 6, 9, 13]

Question 2 — Complexité
Déduire de la question précédente le pire des cas ainsi que la complexité du tri par sélection dans ce cas.

La librairie `timeit` permet de faire des mesures de vitesse d'exécution :



Question 3
Réalisez et affichez avec print les mesures de temps d'exécution des algorithmes de tri par sélection sur des tableaux de taille 100, 1000, 10000.

Rappel : pour créer un tableau d'entiers de taille quelconque n, une méthode efficace consiste à utiliser [i for i in range(n)]

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]*
> *tempsSelection = [0.0001 , 0.001, 0.01, 0.2]*
> *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 pourra utiliser une échelle logarithmique[1] ou semi-logarithmique en remplaçant plot par `semilogx`, `semilogy` ou `loglog`.

[1] https://fr.wikipedia.org/wiki/Échelle_logarithmique