TP TRIS

Partie A : TRAVAUX PRELIMINAIRES

A1.1 **Explication de code**

```
1  L=[1,5,8,12,14,16,53]
2  res=True
3  for i in range(1,len(L)):
4       if L[i-1]>L[i]:
5            res=False
6  print(res)
```

Question 1 : Que fait l'algorithme ci-dessus ?

Question 2 : Que se passerait-il si on avait `L=[4,6,9,11]` ?

A2 Minimum et sous liste d'une liste

Question 3 : Ecrire une fonction mini ayant comme argument une liste L qui détermine le plus petit élément de cette liste. Tester la fonction lorsque $L = [5, 4,7,12,2,6]$.

Question 4 : Ecrire une fonction `creerPairImp` ayant comme argument une liste d'entiers L qui renvoie deux listes :
- La liste des pairs notée `Pa`
- La liste des impairs notée `Imp`

Partie B : TRI PAR INSERTION ET COMPLEXITE

B1 Tri par insertion

On donne ci-dessous l'algorithme de tri par insertion pour une liste L de n éléments :

```
1  Pour i allant de 2 à n
2       Insérer l'élément L[i] à sa place dans L[0,i-1]
3  Fin pour
```

Question 5 : Ecrire une fonction `triInsertion` prenant en argument la liste `L1` et qui renvoie la liste `L2` contenant les éléments de `L1` triés par ordre croissant.

Question 6 : Modifier la fonction triInsertion pour qu'elle renvoie également le nombre de comparaisons d'entiers réalisées.

B2 Complexité du tri par insertion

Soit `n` la longueur de la liste `L1`. On souhaite tracer l'évolution de la complexité temporelle (évaluée ici par le nombre de comparaisons à effectuer pour réaliser le tri de la liste `L1`) en fonction de la longueur `n` de la liste `L1`.
Afin que la complexité obtenue pour un `n` donné soit fiable il est nécessaire de répéter plusieurs fois la même mesure et d'utiliser la valeur moyenne des résultats obtenus. On fera donc varier `n` de 1000 à 10000 par pas de 1000 et on répètera chaque mesure 10 fois.

On peut montrer que le nombre de comparaisons en moyenne est équivalent à $\frac{n^2}{4}$.

Question 7 : Ecrire un script qui permet en traçant un graphe de vérifier cette propriété. On pourra construire des listes aléatoires avec la fonction `randint` du module `random`. Il est possible que vous soyez obligés de modifier la valeur maximale de `n` si le temps de calcul est trop important.

Partie C : TRI FUSION

C1 Tri fusion

Soient deux listes `L1` et `L2` préalablement triées par ordre croissant que l'on souhaite fusionner dans la liste `L`.

```
Tant qu'une des deux listes n'est pas vide
     Si le premier élément de L1 est plus petit que le premier élément de L2.
          Retirer le premier élément de L1 et l'ajouter à L
          Sinon, retirer le premier élément de L2 et l'ajouter à L
Si L1 n'est pas vide
     Elle est ajoutée à L
     Sinon L2 est ajoutée à L
Renvoyer L
```

Question 8 : Ecrire une fonction `fusion` qui prend en argument les deux listes `L1` et `L2` et qui renvoie une liste `L3`, fusion des deux listes `L1` et `L2` et dont les éléments sont rangés par ordre croissant.

Question 9 : Ecrire une fonction `triFusion` qui prend en argument une liste d'entiers `L` non ordonnée et qui renvoie cette liste triée par la méthode du tri fusion.

C2 Complexité du tri fusion

Question 10 : Modifier la fonction `triFusion` pour qu'elle renvoie également le nombre de comparaisons nécessaires pour effectuer le tri.

Question 11 : Modifier le script de la question 7 pour que celui trace sur le même graphe l'évolution des complexités des deux méthodes de tri. Commenter la complexité du tri fusion et la comparer à celle du tri par insertion.

Partie D : TRI RAPIDE (QUICKSORT)

D1 Tri rapide (en place)

IDEE

Le tri rapide a été inventé en 1960 par Tony Hoare.
C'est une méthode récursive qui s'appuie sur le principe "diviser pour régner" :

- on choisit un pivot (aléatoirement ou pas) parmi tous les éléments à trier,
- on réorganise tous les éléments de sorte que tous les éléments inférieurs soient à gauche du pivot et les éléments supérieurs à droite
- trier chaque sous-ensemble (droite et gauche) en y choisissant un nouveau pivot
- lorsque chaque sous-ensemble ne contient qu'un élément, la liste est triée

```
On choisit le premier élément de L comme pivot
On initialise le curseur à c=1 (premier élément après le pivot)
Pour tous les éléments qui suivent le pivot
     Si l'élément d'indice i est inférieur au pivot
          On l'échange avec l'élément d'indice c et on fait c=c+1
          Sinon on ne fait rien
On échange le pivot avec le terme d'indice c-1
```

Question 12 : Ecrire une fonction `partition` prenant en argument une liste `L` et deux indices `g` et `d` ($g < d$) qui définissent une sous liste à partitionner. Cette fonction renverra la position du pivot dans la liste `L` après partition.

Question 13 : Etablir le code Python d'une fonction `triRapideRec` qui prend en argument une liste d'entiers `L` et les deux indices `g` et `d` qui définissent la sous-liste à trier. Cette fonction doit effectuer le tri rapide de `L[g:d]`.

Question 14 : Ecrire le code Python de la fonction `triRapide` qui prend en argument une liste d'entiers non triés `L` et qui la renvoie triée.

D2 Influence du choix du pivot

Dans cette partie on souhaite évaluer l'influence du choix du pivot sur la rapidité du tri rapide. Pour cela on envisage trois stratégies :

- On prend, comme pivot, le premier élément de la liste à partitionner,
- On choisit l'élément le plus proche de la valeur moyenne des éléments de la liste à partitionner,
- On choisit le pivot au hasard parmi les éléments de la liste à partitionner.

En modifiant le code de la fonction partition, écrire trois fonctions de partition appliquant ces stratégies.

Question 15 : Modifier la fonction `triRapide` pour qu'elle renvoie le temps `t1` nécessaire à la réalisation du tri et le nombre `nc` de comparaisons. Le temps `t1` englobe tout (y compris le temps nécessaire au choix du pivot), en revanche, le nombre de comparaisons `nc` ne comprendra pas les éventuelles comparaisons nécessaires au choix du pivot.

Question 16 : Ecrire un script qui fait varier la longueur `n` de la liste à trier de 10000 à 100000 éléments par pas de 10000 éléments et qui trace sur un graphe l'évolution du temps `t1` en fonction de `n`, et sur un second graphe l'évolution de `nc` en fonction de `n` pour les trois stratégies de choix du pivot. Afin que les valeurs de `t1` et `nc` soient fiables on fera une moyenne sur 10 essais pour chaque valeur de `n`.

D3 Complexité du tri rapide

Question 17 : Modifier les fonctions `triInsertion` et `triFusion` pour qu'elles renvoient également le temps nécessaire pour effectuer le tri.

Question 18 : Modifier le script de la question 16 pour que celui-ci trace sur le même graphe l'évolution du temps d'exécution des 3 méthodes de tri en fonction de la longueur `n` de la liste à trier. Que peut-on dire des complexités des trois méthodes.

Partie E : TRI A BULLES (HORS PROGRAMME)

E1 Définition

DEF. Soit une liste L de `n` éléments. Le tri à bulles consiste à comparer les deux premiers termes de la liste. S'ils sont désordonnés, on les échange. On recommence ensuite avec le deuxième et le troisième terme du tableau et ainsi de suite.
Ainsi, au bout d'un premier passage, l'élément le plus grand est bien placé : il est remonté comme une bulle. On recommence l'opération jusqu'à ce que le tableau soit trié.

Pour une liste `L` de `n` éléments, on trouve l'algorithme suivant :

```
permut=vrai
Tant que permut=vrai faire :
     permut=faux
     Pour j allant de 1 à n-1
          si L[j]>L[j+1] alors :
               échanger L[j] et L[j+1]
               permut=vrai
          fin si
     fin pour
fin tant que
```

E2 Tri à bulles

Question 19 : Ecrire une fonction `triAbulles` qui prend en argument une liste d'entiers `L` et qui renvoie cette même liste triée par la méthode du tri à bulles.

E3 Complexité du tri à bulles

Question 20 : Modifier la fonction `triAbulles` pour qu'elle renvoie également le temps nécessaire et le nombre de comparaisons pour effectuer le tri.

Question 21 : Modifier le script de la question 18 pour que celui-ci trace sur le même graphe l'évolution du temps d'exécution des 4 méthodes de tri en fonction de la longueur `n` de la liste à trier. Que peut-on dire des complexités des quatre méthodes.