EXERCICE 1 : Algorithmes de tri (4 points)

Cet exercice traite principalement du thème « algorithmique, langages et programmation ». Le but est de comparer le tri par insertion (l'un des algorithmes étudiés en 1ère NSI pour trier un tableau) avec le tri fusion (un algorithme qui applique le principe de « diviser pour régner »).

Partie A : Manipulation d'une liste en Python

1. Donner les affichages obtenus après l'exécution du code Python suivant.

```
notes = [8, 7, 18, 14, 12, 9, 17, 3]
notes[3] = 16
print(len(notes))
print(notes)
```

2. Écrire un code Python permettant d'afficher les éléments d'indice 2 à 4 de la liste `notes`.

Partie B : Tri par insertion

Le tri par insertion est un algorithme efficace qui s'inspire de la façon dont on peut trier une poignée de cartes. On commence avec une seule carte dans la main gauche (les autres cartes sont en tas sur la table) puis on pioche la carte suivante et on l'insère au bon endroit dans la main gauche.

1. Voici une implémentation en Python de cet algorithme. Recopier et compléter les lignes 6 et 7 surlignées (uniquement celles-ci).

```
1  def tri_insertion(liste):
2      """ trie par insertion la liste en paramètre """
3      for indice_courant in range(1,len(liste)):
4          element_a_inserer = liste[indice_courant]
5          i = indice_courant - 1
6          while i >= 0 and liste[i] > ..... :
7              liste[.....] = liste[.....]
8              i = i - 1
9          liste[i + 1] = element_a_inserer
```

On a écrit dans la console les instructions suivantes :

```
notes = [8, 7, 18, 14, 12, 9, 17, 3]
tri_insertion(notes)
print(notes)
```

On a obtenu l'affichage suivant : `[3, 7, 8, 9, 12, 14, 17, 18]`

On s'interroge sur ce qui s'est passé lors de l'exécution de `tri_insertion(notes)`.

2. Donner le contenu de la liste `notes` après le premier passage dans la boucle `for`.
3. Donner le contenu de la liste `notes` après le troisième passage dans la boucle `for`.

Partie C : Tri fusion

L'algorithme de tri fusion suit le principe de « diviser pour régner ».

> (1) Si le tableau à trier n'a qu'un élément, il est déjà trié.
>
> (2) Sinon, séparer le tableau en deux parties à peu près égales.
>
> (3) Trier les deux parties avec l'algorithme de tri fusion.
>
> (4) Fusionner les deux tableaux triés en un seul tableau. *source : Wikipedia*

1. Cet algorithme est-il itératif ou récursif ? Justifier en une phrase.

2. Expliquer en trois lignes comment faire pour rassembler dans une main deux tas déjà triés de cartes, la carte en haut d'un tas étant la plus petite de ce même tas ; la deuxième carte d'un tas n'étant visible qu'après avoir retiré la première carte de ce tas.
   À la fin du procédé, les cartes en main doivent être triées par ordre croissant.

Une fonction `fusionner` a été implémentée en Python en s'inspirant du procédé de la question précédente. Elle prend quatre arguments : la liste qui est en train d'être triée, l'indice où commence la sous-liste de gauche à fusionner, l'indice où termine cette sous-liste, et l'indice où se termine la sous-liste de droite.

3. Voici une implémentation de l'algorithme de tri fusion. Recopier et compléter les lignes 8, 9 et 10 surlignées (uniquement celles-ci).

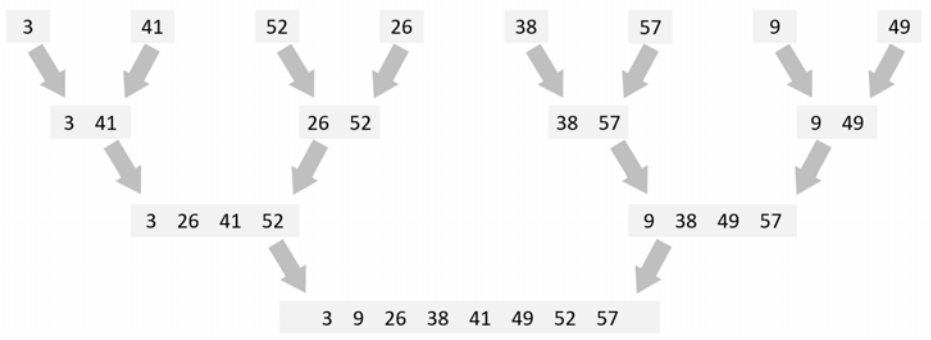
```
1   from math import floor
2
3   def tri_fusion (liste, i_debut, i_fin):
4       """ trie par fusion la liste en paramètre depuis
5       i_debut jusqu'à i_fin """
6       if i_debut < i_fin:
7           i_partage = floor((i_debut + i_fin) / 2)
8           tri_fusion(liste, i_debut, ..........)
9           tri_fusion(liste, ..........., i_fin)
10          fusionner(liste, ..........., ..........., ..........)
```

Remarque : la fonction `floor` renvoie la partie entière du nombre passé en paramètre.

4. Expliquer le rôle de la première ligne du code de la question 3.

Partie D : Comparaison du tri par insertion et du tri fusion

Voici une illustration des étapes d'un tri effectué sur la liste [3, 41, 52, 26, 38, 57, 9, 49].



1. Quel algorithme a été utilisé : le tri par insertion ou le tri fusion ? Justifier.

2. Identifier le tri qui a une complexité, dans le pire des cas, en $O(n^2)$ et identifier le tri qui a une complexité, dans le pire des cas, en $O(n \log_2 n)$.
   Remarque : n représente la longueur de la liste à trier.

3. Justifier brièvement ces deux complexités.