Devoir 4 : Les tris - Diviser pour régner

1. Q.C.M.

Exo

1. On applique l'algorithme du tri par sélection à la liste `[8,12,6,19]`, après la première étape, le contenu de la liste sera :

a) `[12,8,6,19]`

b) `[6,12,8,19]`

c) `[19,12,6,9]`

d) Aucune des propositions ci-dessus

2. On applique l'algorithme du tri par insertion à la liste `[9,11,7,16]`, quel sera le contenu de la liste après le premier échange ?

a) `[11,9,7,16]`

b) `[9,11,16,7]`

c) `[9,7,11,16]`

d) Aucune des propositions ci-dessus

3. L'algorithme du tri par sélection a une complexité :

a) logarithmique

b) linéaire

c) quadratique

d) exponentielle

4. Un programme de tri par insertion prend environ 1/10 seconde pour trier une liste de \(1\,000\) éléments, combien de temps prendra-t-il environ pour trier une liste de \(100\,000\) éléments ?

a) 1 seconde

b) 10 secondes

c) 100 secondes

d) 1000 secondes

2. D'aprés exercice BAC

Exo

2.1. Partie A : Généralités - Cours

Question A.1

Quel est l'ordre de grandeur du coût, en nombre de comparaisons, de l'algorithme de tri fusion pour une liste de longueur \(n\) ?

Réponse

\(O(nlog_2(n))\)

Question A.2

Citer le nom d'un autre algorithme de tri. Donner l'ordre de grandeur de son coût, en nombre de comparaisons, pour une liste de longueur .
Comparer ce coût à celui du tri fusion.

Réponse

L'algorithme de tri par insertion a une complexité en temps dans le pire des cas en \(O(n^2)\).
L'algorithme du tri par insertion est moins efficace que l'algorithme de tri fusion.

2.2. Partie B : Tri fusion

L'algorithme de tri fusion utilise deux fonctions `moitie_gauche` et `moitie_droite` qui prennent en argument une liste L et renvoient respectivement :

- la sous-liste de L formée des éléments d'indice strictement inférieur à `len(L)//2` ;
- la sous-liste de L formée des éléments d'indice supérieur ou égal à `len(L)//2`.

On rappelle que la syntaxe a//b désigne la division entière de a par b.

Par exemple,

Script Python

```
>>> L = [3, 5, 2, 7, 1, 9, 0]
>>> moitie_gauche(L)
[3, 5, 2]
>>> moitie_droite(L)
[7, 1, 9, 0]
>>> M = [4, 1, 11, 7]
>>> moitie_gauche(M)
[4, 1]
>>> moitie_droite(M)
[11, 7]
```

L'algorithme utilise aussi une fonction `fusion` qui prend en argument deux listes triées `L1` et `L2` et renvoie une liste `L` triée et composée des éléments de `L1` et `L2`.
On donne ci-dessous le code python d'une fonction récursive `tri_fusion` qui prend en argument une liste `L` et renvoie une nouvelle liste triée formée des éléments de `L`.

Script Python

```
def tri_fusion(L):
    n = len(L)
    if n<=1 :
        return L
    print(L)
    mg = moitie_gauche(L)
    md = moitie_droite(L)
    L1 = tri_fusion(mg)
    L2 = tri_fusion(md)
    return fusion(L1, L2)
```

Question B.1

Donner la liste des affichages produits par l'appel suivant.

Script Python

```
tri_fusion([9, 5, 3, 1, 7, 6, 10, 3])
```

Réponse

Script Python

```
[9, 5, 3, 1, 7, 6, 10, 3]
[9, 5, 3, 1]
[9, 5]
[3, 1]
[7, 6, 10, 3]
[7, 6]
[10, 3]
```

On s'intéresse désormais à différentes fonctions appelées par `tri_fusion`, à savoir `moitie_gauche` et `fusion`.

Question B.2

Écrire une fonction `moitie_gauche(tab)` qui prend en argument un tableau `tab` et renvoie un nouveau tableau contenant la moitié gauche de `tab`.

Réponse

Script Python

```
def moitie_gauche(L):
    n = len(L)
    fin = n//2
    tab = []
    for i in range(fin):
        tab.append(L[i])
    return tab
```

Question B.3

On donne ci-dessous une version incomplète de la fonction `fusion`.

```
1   def fusion(L1, L2):
2       L = []
3       n1 = len(L1)
4       n2 = len(L2)
5       i1 = 0
6       i2 = 0
7       while .... :
8           if i1 >= n1:
9               ...
10              ...
11          elif i2 >= n2:
12              L.append(L1[i1])
13              i1 = i1 + 1
14          else:
15              e1 = L1[i1]
16              e2 = L2[i2]
17
18
19
20
21
22
23
24      return L
```

Dans cette fonction, les entiers i1 et i2 représentent respectivement les indices des éléments des listes L1 et L2 que l'on souhaite comparer :

- Si aucun des deux indices n'est valide, la boucle while est interrompue ;
- Si i1 n'est plus un indice valide, on va ajouter à L les éléments de L2 à partir de l'indice i2 ;
- Si i2 n'est plus un indice valide, on va ajouter à L les éléments de L1 à partir de l'indice i1 ;
- Sinon, le plus petit élément non encore traité est ajouté à L et on décale l'indice correspondant.

Écrire sur la copie les instructions manquantes de la ligne 7, des lignes 9 à 10 puis des lignes de 17 à 22..

Réponse

```
1   def fusion(L1, L2):
2       L=[]
3       n1 = len(L1)
4       n2 = len(L2)
5       i1 = 0
6       i2 = 0
7       while i1<n1 or i2<n2:
8           if i1>=n1:
9               L.append(L2[i2])
10              i2 = i2+1
11          elif i2>=n2:
12              L.append(L1[i1])
13              i1=i1+1
14          else :
15              e1 = L1[i1]
16              e2 = L2[i2]
17              if e1 > e2:
18                  L.append(e2)
19                  i2 = i2 + 1
20              else :
21                  L.append(e1)
22                  i1 = i1 + 1
23      return L
```

2.3. Partie C : 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.

Question C.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
```

Réponse

🐍 **Script Python**

```
def tri_insertion(liste):
for indice_courant in range(1, len(liste)):
    element_a_inserer = liste[indice_courant]
    i = indice_courant - 1
    while i >= 0 and liste[i] > element_a_inserer:
        liste[i+1] = liste[i]
        i=i-1
    liste[i + 1] = element_a_inserer
```

On a écrit dans la console les instructions suivantes :

🐍 **Script Python**

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

On a obtenu l'affichage suivant :

🐍 **Script Python**

```
[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)`.

Question C.2

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

Réponse

`Passage 1 : [7, 8, 18, 14, 12, 9, 17, 3]`

Question C.3

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

Réponse

`Passage 3 : [7, 8, 14, 18, 12, 9, 17, 3]`

Question C.4

Donner le contenu de la liste notes étape par étape lors de l'éxecution de la fontion `tri_insertion` .

Réponse

🐍 Script Python

```
Passage 1 : [7, 8, 18, 14, 12, 9, 17, 3]
Passage 2 : [7, 8, 18, 14, 12, 9, 17, 3]
Passage 3 : [7, 8, 14, 18, 12, 9, 17, 3]
Passage 4 : [7, 8, 12, 14, 18, 9, 17, 3]
Passage 5 : [7, 8, 9, 12, 14, 18, 17, 3]
Passage 6 : [7, 8, 9, 12, 14, 17, 18, 3]
Passage 7 : [3, 7, 8, 9, 12, 14, 17, 18]
```