

Sujet BAC 8 : Diviser pour régner



1. France 2021

Sujet n°1 : France 2021

Cet exercice porte sur l'algorithme de tri fusion, qui s'appuie sur la méthode dite de « diviser pour régner ».

Question 1

Enoncé

- Quel est l'ordre de grandeur du coût, en nombre de comparaisons, de l'algorithme de tri fusion pour une liste de longueur ?
- 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. Aucune justification n'est attendue.

Solution

- $O(n \log_2(n))$
- 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.

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 2

Enoncé

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

 Script Python

```
tri_fusion([7, 4, 2, 1, 8, 5, 6, 3])
```

Solution

Voici l'affichage obtenu :

 Script Python

```
[7, 4, 2, 1, 8, 5, 6, 3]
[7, 4, 2, 1]
[7, 4]
[2, 1]
[8, 5, 6, 3]
[8, 5]
[6, 3]
```

résultat renvoyé par la fonction : [1, 2, 3, 4, 5, 6, 7, 8]

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

 Question 3

Enoncé

Ecrire la fonction moitie_droite.

Solution

 Script Python

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

Question 4

Enoncé

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

Script Python

```

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
18
19
20    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 des lignes 17 à 22 permettant d'insérer dans la liste L les éléments des listes L1 et L2 par ordre croissant.

Solution

Script Python

```

def fusion(L1, L2):
    L=[]

```

```
n1 = len(L1)
n2 = len(L2)
i1 = 0
i2 = 0
while i1<n1 or i2<n2:
    if i1>=n1:
        L.append(L2[i2])
        i2 = i2+1
    elif i2>=n2:
        L.append(L1[i1])
        i1=i1+1
    else :
        e1 = L1[i1]
        e2 = L2[i2]
        if e1 > e2:
            L.append(e2)
            i2 = i2 + 1
        else :
            L.append(e1)
            i1 = i1 + 1
return L
```

2. Polynésie 2021

Sujet n°2 : BAC Polynésie 2021

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 »).

2.1. Partie A : Manipulation d'une liste en Python

Question A.1

Enoncé

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

🐍 Script Python

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

Solution

🐍 Script Python

```
>>> 8
>>> [8, 7, 18, 16, 12, 9, 17, 3]
```

```!!! fabquestion "Question A.2" === "Enoncé" Écrire un code Python permettant d'afficher les éléments d'indice 2 à 4 de la liste notes.

#### ⌚ Texte

```
==== "Solution"

```python
for i in range(2,5):
    print(notes[i])
```
```

## 2.2. 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.

## Question B.1

### Enoncé

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

#### Script Python

```

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

```

### Solution

#### 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 B.2

### Enoncé

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

### Solution

#### Script Python

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

## Question B.3

### Enoncé

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

### Solution

#### Script Python

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

## 2.3. 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

### Question C.1

#### Enoncé

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

#### Solution

récursif : à l'étape (3) l'algorithme de tri fusion s'appelle lui-même.

### Question C.2

#### Enoncé

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.

#### Solution

1. Comparer les cartes du haut des 2 tas.
2. Placer la carte de valeur plus faible dans la main.
3. Recommencer l'étape 1 jusqu'à épuisement des 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 sousliste, et l'indice où se termine la sous-liste de droite.

### Question C.3

#### Enoncé

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

#### Script Python

```

1 from math import floor
2
3 def tri_fusion (liste, i_debut, i_fin):
4 if i_debut < i_fin:
5 i_partage = floor((i_debut + i_fin) / 2)
6 tri_fusion(liste, i_debut,)
7 tri_fusion(liste, , i_fin)
8 fusionner(liste, , ,
.....)

```

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

#### Solution

#### Script Python

```

from math import floor

def tri_fusion (liste, i_debut, i_fin):
 if i_debut < i_fin:
 i_partage = floor((i_debut + i_fin) / 2) # milieu pour diviser le tableau en
deux moitiés
 tri_fusion(liste, i_debut, i_partage) # Appel tri_fusion pour 1ère moitié du
tableau
 tri_fusion(liste, i_partage + 1, i_fin) # Appel tri_fusion pour 2ème moitié du
tableau
 fusionner(liste, i_debut, i_fin, i_partage) # Fusion des deux moitiés triées

```

### Question C.4

#### Enoncé

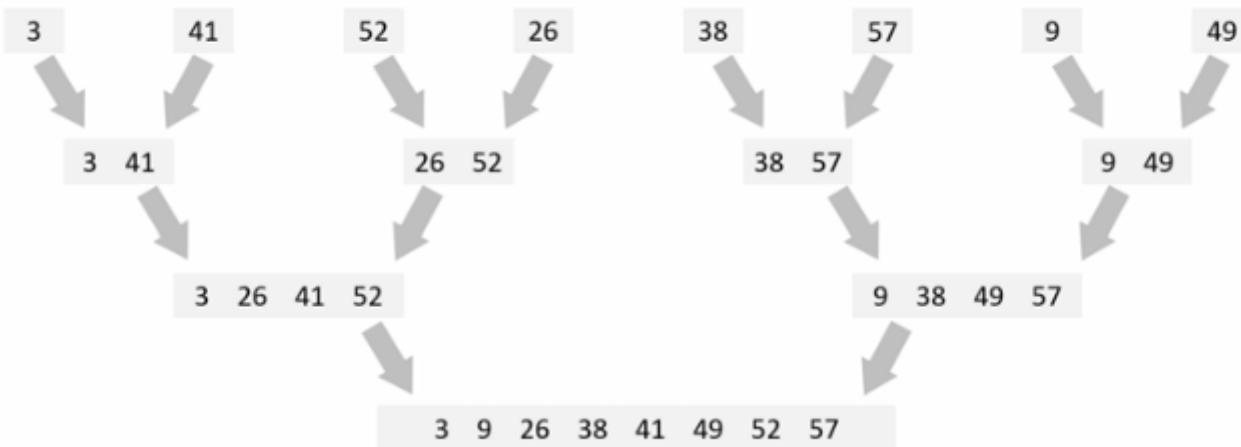
Expliquer le rôle de la première ligne du code de la question 3.

#### Solution

permet d'importer la fonction floor() du module math utilisée à la ligne 7

## 2.4. 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].



[:center]}

### Question D.1

#### Enoncé

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

#### Solution

tri par fusion : à chaque étape, le tri se fait par fusion de 2 tas déjà triés

## Question D.2

### Enoncé

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.

### Solution

- tri par insertion :  $O(n^2)$
- tri par fusion :  $O(n \cdot \log_2 n)$

## Question D.3

### Enoncé

Justifier brièvement ces deux complexités.

### Solution

- Le tri par insertion utilise 2 boucles imbriquées, soit dans le pire des cas  $\sim \frac{1}{2}n(n + 1)$  opérations soit  $O(n^2)$ .
- Le tri par fusion, divise la liste à trier par 2 pour chaque itérations, soit  $\sim \log_2 n$  opérations d'où une complexité global  $O(n \cdot \log_2 n)$ .

### Inversions dans une liste : FRANCE CANDIDAT LIBRE SUJET 1

Cet exercice traite de manipulation de tableaux, de récursivité et du paradigme « diviser pour régner ».

Dans un tableau Python d'entiers tab, on dit que le couple d'indices ( $i, j$ ) forme une inversion lorsque  $i < j$  et  $tab[i] > tab[j]$ . On donne ci-dessous quelques exemples.

- Dans le tableau [1, 5, 3, 7], le couple d'indices (1,2) forme une inversion car  $5 > 3$ . Par contre, le couple (1,3) ne forme pas d'inversion car  $5 < 7$ . Il n'y a qu'une inversion dans ce tableau.
- Il y a trois inversions dans le tableau [1, 6, 2, 7, 3], à savoir les couples d'indices (1, 2), (1, 4) et (3, 4).

- On peut compter six inversions dans le tableau [7, 6, 5, 3] : les couples d'indices (0, 1), (0, 2), (0, 3), (1, 2), (1, 3) et (2, 3).

On se propose dans cet exercice de déterminer le nombre d'inversions dans un tableau quelconque.

Questions préliminaires

### Question 1

Enoncé

Expliquer pourquoi le couple (1, 3) est une inversion dans le tableau [4, 8, 3, 7].

Solution

À l'indice 1 du tableau on trouve 8, à l'indice 3 on trouve 7.

Nous avons  $1 < 3$  alors que  $8 > 7$ , nous avons donc bien une inversion

### Question2

Enoncé

Justifier que le couple (2, 3) n'en est pas une.

Solution

À l'indice 2 du tableau on trouve 3, à l'indice 3 on trouve 7.

Nous avons  $2 < 3$  et  $3 < 7$ , nous n'avons donc pas d'inversion

## 2.5. Partie A : Méthode itérative

Le but de cette partie est d'écrire une fonction itérative `nombre_inversion` qui renvoie le nombre d'inversions dans un tableau. Pour cela, on commence par écrire une fonction `fonction1` qui sera ensuite utilisée pour écrire la fonction `nombre_inversion`.

## Question A.1

### Enoncé

On donne la fonction suivante.

#### Script Python

```
def fonction1(tab, i):
 nb_elem = len(tab)
 cpt = 0
 for j in range(i+1, nb_elem):
 if tab[j] < tab[i]:
 cpt += 1
 return cpt
```

a. Indiquer ce que renvoie la fonction1(tab, i) dans les cas suivants.

- Cas n°1 : tab = [1, 5, 3, 7] et i = 0.
- Cas n°2 : tab = [1, 5, 3, 7] et i = 1.
- Cas n°3 : tab = [1, 5, 2, 6, 4] et i = 1.

b. Expliquer ce que permet de déterminer cette fonction.

### Solution

a.

- cas n°1 : 0
- cas n°2 : 1
- cas n°3 : 2

b.

## Question A.2

### Enoncé

En utilisant la fonction précédente, écrire une fonction nombre\_inversion(tab) qui prend en argument un tableau et renvoie le nombre d'inversions dans ce tableau.

On donne ci-dessous les résultats attendus pour certains appels.

### Texte

```
>>> nombre_inversions([1, 5, 7])
0
>>> nombre_inversions([1, 6, 2, 7, 3])
3
>>> nombre_inversions([7, 6, 5, 3])
6
```

### Solution

#### Script Python

```
def nombre_inversions(tab):
 nb_inv = 0
 n = len(tab)
 for i in range(n-1):
 nb_inv = nb_inv + fonction1(tab, i)
 return nb_inv
```

## Question A.3

### Enoncé

Quelle est l'ordre de grandeur de la complexité en temps de l'algorithme obtenu ?

Aucune justification n'est attendue.

### Solution

L'ordre de grandeur de la complexité en temps de l'algorithme est  $O(n^2)$

## 2.6. Partie B : Méthode récursive

Le but de cette partie est de concevoir une version récursive de la fonction `nombre_inversion`.  
On définit pour cela des fonctions auxiliaires.

### Question B.1

#### Enoncé

Donner le nom d'un algorithme de tri ayant une complexité meilleure que quadratique.

Dans la suite de cet exercice, on suppose qu'on dispose d'une fonction `tri(tab)` qui prend en argument un tableau et renvoie un tableau contenant les mêmes éléments rangés dans l'ordre croissant.

#### Solution

Le tri fusion a une complexité en  $O(n \cdot \log_2(n))$

## Question B.2

### Enoncé

Écrire une fonction moitie\_gauche(tab) qui prend en argument un tableau tab et renvoie un nouveau tableau contenant la moitié gauche de tab. Si le nombre d'éléments de tab est impair, l'élément du centre se trouve dans cette partie gauche.

On donne ci-dessous les résultats attendus pour certains appels.

#### Script Python

```
>>> moitie_gauche([])
[]
>>> moitie_gauche([4, 8, 3])
[4, 8]
>>> moitie_gauche ([4, 8, 3, 7])
[4, 8]
```

### Solution

#### Script Python

```
def moitie_gauche(tab):
 n = len(tab)
 nvx_tab = []
 if n==0:
 return []
 mil = n//2
 if n%2 == 0:
 lim = mil
 else :
 lim = mil+1
 for i in range(lim):
 nvx_tab.append(tab[i])
 return nvx_tab
```

une autre possibilité un peu plus concise :

#### Script Python

```
def moitie_gauche(tab):
 return [tab[i] for i in range(len(tab)//2+len(tab)%2)]
```

Dans la suite, on suppose qu'on dispose de la fonction moitie\_droite(tab) qui renvoie la moitié droite sans l'élément du milieu.

### Question B.3

#### Enoncé

On suppose qu'une fonction `nb_inv_tab(tab1, tab2)` a été écrite. Cette fonction renvoie le nombre d'inversions du tableau obtenu en mettant bout à bout les tableaux `tab1` et `tab2`, à condition que `tab1` et `tab2` soient triés dans l'ordre croissant.

On donne ci-dessous deux exemples d'appel de cette fonction :

#### Script Python

```
>>> nb_inv_tab([3, 7, 9], [2, 10])
3
>>> nb_inv_tab([7, 9, 13], [7, 10, 14])
3
```

En utilisant la fonction `nb_inv_tab` et les questions précédentes, écrire une fonction récursive `nb_inversions_rec(tab)` qui permet de calculer le nombre d'inversions dans un tableau. \* Cette fonction renverra le même nombre que `nombre_inversions(tab)` de la partie A. On procédera de la façon suivante :

- Séparer le tableau en deux tableaux de tailles égales (à une unité près).
- Appeler récursivement la fonction `nb_inversions_rec` pour compter le nombre d'inversions dans chacun des deux tableaux.
- Trier les deux tableaux (on rappelle qu'une fonction de tri est déjà définie).
- Ajouter au nombre d'inversions précédemment comptées le nombre renvoyé par la fonction `nb_inv_tab` avec pour arguments les deux tableaux triés.

#### Solution

#### Script Python

```
def nb_inversions_rec(tab):
 if len(tab) > 1:
 tab_g = moitie_gauche(tab)
 tab_d = moitie_droite(tab)
 return nb_inv_tab(tri(tab_g), tri(tab_d))
 nb_inversions_rec(tab_g) + nb_inversions_rec(tab_d)
 else:
 return 0
```

ou

#### Script Python

```

def nb_inversions_rec(tab : list, n : int = 0) -> int:
 if len(tab) <= 1:
 return 0
 else:
 #Séparer le tableau en deux tableaux de tailles égales (à une unité près).
 gauche = moitie_gauche(tab)
 droite = moitie_droite(tab)
 #Compter le nombre d'inversions dans chacun des deux tableaux.
 n = nb_inv_tab(sorted(gauche), sorted(droite))
 #Appeler récursivement la fonction nb_inversions_rec
 return n + nb_inversions_rec(gauche, n) + nb_inversions_rec(droite, n)

```

### Quart de tour d'une image

1. Pour faire tourner une image carré de côté  $2^n$  pixels d'un quart de tour à gauche, on propose la méthode suivante :

- Diviser l'image en quatre quarts Q1,Q2,Q3,Q4

|    |    |
|----|----|
| Q1 | Q2 |
| Q3 | Q4 |

- Faire tourner chacun des quarts d'un quart de tour à gauche

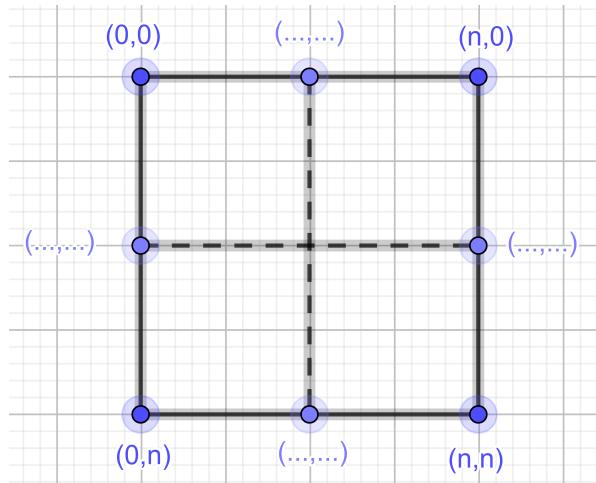
|    |    |
|----|----|
| Q1 | Q2 |
| Q3 | Q4 |

- Permuter chaque quart afin de le placer correctement

|    |    |
|----|----|
| Q2 | Q4 |
| Q1 | Q3 |

Expliquer pourquoi cette méthode est une illustration de la technique diviser pour régner.

2. C'est algorithme est-il du type itératif ou récursif ? Justifier.
3. Découpage de l'image en quatres quart à l'aide du module `PIL` de manipulation d'images
- a. On a représenté une image carré de  $n$  pixels de côté avec le système de coordonnées d'une image dans le module `PIL`. Quelles sont les coordonnées manquantes ?



- b. La méthode `crop` du module `PIL` permet d'extraire une portion rectangulaire d'une image en donnant les coordonnées des coins supérieur gauche et inférieur droit du rectangle. Compléter la fonction Python suivante qui prend en entrée une image et retourne les quatre quart de cette image.

#### Script Python

```
from PIL import Image
def partage_quart(image):
 n = image.width
 if n > 1:
 q1 = image.crop((0,0,n//2,n//2))
 q2 = image.crop((.....,.....,.....,.....))
 q3 = image.crop((.....,.....,.....,.....))
 q4 = image.crop((.....,.....,.....,.....))
 return q1,q2,q3,q4
```

- c. Tester cette fonction (on pourra utiliser [cette image carré](#))

 Aide

- La création d'une image dans PIL à partir d'un fichier s'effectue à l'aide de :

 Script Python

```
img_test = Image.open("mettre ici le nom du fichier")
```

- La visualisation d'une image s'effectue à l'aide de :

 Script Python

```
img_test.show()
```

d. Ajouter une instruction `assert` permettant de vérifier que l'image est carré (c'est à dire `image.width==image.height`)

e. Ajouter une instruction `assert` permettant de vérifier que `n` est pair.

4. Compléter puis tester la fonction python qui implémente l'algorithme décrit à la question 1.

 Script Python

```
def quart_tour(image):
 n = image.width
 # Partage de l'image en quatre quarts
 if n>1:
 q1,q2,q3,q4 = partage_quart(image)
 # Rotation de chacun des quarts
 rq1 = quart_tour(q1)
 rq2 = quart_tour(q2)
 rq3 = quart_tour(q3)
 rq4 = quart_tour(q4)
 # Reconstruction de l'image
 resultat = Image.new('RGB',image.size)
 resultat.paste(rq2,(0,0))
 resultat.paste(....,(n//2,0))
 resultat.paste(rq1,(...,...))
 resultat.paste(....,(...,...))
 return resultat
 else:
 return image
```