Les algorithmes de tri - correction des scripts

Tri rapide (ou QuickSort)

La fonction auxiliaire permettant de partitionner un tableau en deux tableaux autour d'un pivot :

```
def partition(a):
    '''partitionne le tableau a en deux,le premier tableau renvoyé contient les éléments inférieurs
           ou égaux au pivot (sans le pivot), le deuxième tableau contient les éléments strictement supérieurs'''

    pivot=a[0]  #on choisit toujours le premier élément de la liste comme pivot
    n=len(a)
    b=[]   #b contiendra les éléments plus petits que le pivot
    c=[]   #c contiendra les éléments plus grands que le pivot
    for x in a[1:n]:
        if x<=pivot:
            b.append(x)
        else:
            c.append(x)

    return b,pivot,c
```

L'algorithme du tri rapide :

```
def tri_rapide(t):

    if len(t)==1 or len(t)==0:  # si la liste est vide ou contient un seul élément, elle est déjà triée
        return(t)

    b,pivot,c=partition(t)            # on partionne la liste autour du pivot
    t=tri_rapide(b)+[pivot]+tri_rapide(c) # on fait un appel récursif
                                      #en appliquant la fonction aux deux sous-tableaux
    return t
```

Tri fusion(ou MergeSort)

La fonction auxiliaire permettant de fusionner deux tableaux déjà triés :

```
def fusion(a,b):
    '''réalise la fusion de deux tableaux a et b déjà triés'''
    t=[]
    i=0;j=0

    while i<len(a) and j<len(b):  #la boucle tourne tant qu'au moins une des listes n'est pas parcourue entièrement
        if a[i]<b[j]:
            t.append(a[i])
            i=i+1
        else:
            t.append(b[j])
            j=j+1

    if i==len(a):       # s'il s'agit de la liste a qui est épuisée, on complète t avec les éléments restants de b
        t=t+b[j:len(b)]
    if j==len(b):       # s'il s'agit de la liste b qui est épuisée, on complète t avec les éléments restants de a
        t=t+a[i:len(a)]

    return t
```

L'algorithme du tri fusion :

```
def tri_fusion(t):

    if len(t)==0 or len(t)==1:  # si la liste est vide ou contient un seul élément,
                                #elle est déjà triée
        return t

    n=len(t)//2                 # on repère la position moitié (ou presque) dans la liste
    t1=tri_fusion(t[0:n])       # appel récursif pour trier la première moitié du tableau

    t2=tri_fusion(t[n:len(t)])  # appel récursif pour trier la deuxième moitié du tableau

    return fusion(t1,t2)      # on retourne la fusion des deux tableaux triés
```

Tri par insertion

```
def tri_insertion(a):
    '''retourne un tableau avec les mêmes éléments triés par ordre croissant'''

    n=len(a)

    for i in range(1,n):  #on parcourt un par un tous les éléments de la liste
        x=a[i]              #x est l'élément à insérer dans la partie de liste déjà triée
        j=i-1                   #on va parcourir la partie de liste déjà triée ,
                                #de la droite vers la gauche
        while j>=0 and a[j]>x:
            a[j+1]=a[j]         #on décale vers la droite les éléments tant qu'ils sont plus grands que x
            j=j-1
        a[j+1]=x   #on insère x à la bonne place
    return a
```

Tri par sélection (version non récursive)

```
def indice_min(t,g,d):
    '''renvoie l'indice du plus petit élément de la partie de t comprise entre g inclus et d inclus'''
    i_min=g
    min=t[g]
    for i in range(g+1,d+1):
        if t[i]<min:
            min=t[i]
            i_min=i
    return i_min

def tri_selection_iteratif(t):
    n=len(t)
    for k in range(n-1):
        j=indice_min(t,k,n-1)  #position du minimum dans le tableau à partir de l'indice k
        t[k],t[j]=t[j],t[k]
    return t
```