

TP 09

Tri

Savoirs et compétences :

☐ Th. 8 : Tris.

Proposition de corrigé

Activité 1 – Tri pas comptage d'une liste

Question 1 Ecrire la fonction on pourra (au choix) utiliser l'une des signatures suivante: `tri_comptage(L:list,k:int)` -> None ou `tri_comptage(L:list,k:int)` -> list permettant de réaliser un tri par comptage (avec ou sans effet de bord).

```
def tri_comptage(L:list,k:int):
    C=k*[0]
    a=[]
    for x in L:
        for i in range(k):
            if x==i:
                C[i]+=1
    for i in range(k):
        a+=C[i]*[i]
    return a

def tri_comptage2(L,k):
    C=[0]*k
    for i in range(len(L)):
        C[L[i]]=C[L[i]]+1
    p=0
    for i in range(k):
        for j in range(C[i]):
            L[p]=i
            p+=1
```

Question 2 La fonction proposée agit-elle avec effet de bord? Sans effet de bord?

Selon la version du code la méthode se fait avec ou sans effet de bord. La première méthode proposée est sans effet de bord alors que la deuxième oui.

Question 3 La fonction proposée réalise-t-elle un tri stable? un tri en place?

Activité 2 – Classement de l'étape 18 Embrun – Valloire – 208 km du tour de France

Question 4 Quelle méthode de tri vous semble la mieux adaptée au tri du classement général?

La méthode par insertion qui semble naïve est sans doute adaptée ici car les listes triées n'évolueront peu d'une étape à l'autre et dans ce cas la méthode peut s'approcher d'une complexité linéaire.

Question 5 Modifier un algorithme de tri au choix pour pouvoir trier la liste obtenue en sortie de la fonction ajoutTemps(L1,L2) selon la dernière colonne. On l'appellera tri_modifie(liste)->list.

```
####Tri par insertion modifié
def tri_insertion_modifie(T):
    n=len(T)
    for i in range(1,n):
        j=i
        v=T[i][-1]
        v2=T[i]
        while j>0 and v<T[j-1][-1]:
            T[j]=T[j-1]
            j=j-1
        T[j]=v2
    return T

####Tri rapide modifié
def partition_modifie(a,g,d):
    assert g<d
    v=a[g][-1]
    v2=a[g]
    ainf=[]
    asup=[]
    for x in a[g+1:d]:
        if x[-1]<=v:
            ainf.append(x)
        else:
            asup.append(x)
    a=a[0:g]+ainf+[v2]+asup+a[d:len(a)]
    m=len(ainf)+g
    return m,a

def tri_rapide_modifie(a,g,d):
    if g>=d-1:
        return
    else:
        m,a=partition_modifie(a,g,d)
        tri_rapide_modifie(a,g,m)
        tri_rapide_modifie(a,m+1,d)

####Tri par fusion modifié
def fusion_modifie(a0,a,g,m,d):
    i,j=g,m
    for k in range(g,d):
        if i<m and (j==d or a0[i][-1]<=a0[j][-1]):
            a[k]=a0[i]
            i=i+1
        else:
            a[k]=a0[j]
            j=j+1

def tri_fusion_modifie(a,g,d):
    a0=a[:]
    if g>=d-1:
        return
    else:
        m=(g+d)//2
        tri_fusion_modifie(a,g,m)
        tri_fusion_modifie(a,m,d)
        a0[g:d]=a[g:d]
        fusion_modifie(a0,a,g,m,d)
```

Question 6 Ecrire une fonction update_classement_general(liste1:list,liste2:list)->list

qui à partir des deux listes du classement de l'étape 18 et du classement général renvoie la liste de la forme [[Nom_1, Dossard_1, Temps_1], [Nom_2, Dossard_2, Temps_2], ...] triée dans l'ordre du nouveau classement général.

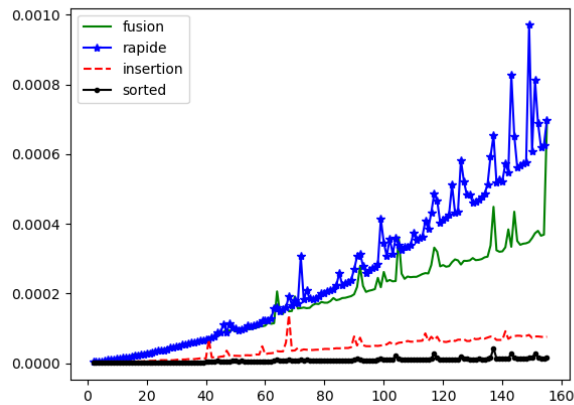
```
def update_classement_general(liste1:list,liste2:list)->list:
    LGN=ajoutTemps(LG,L18)
    sorted(LGN,key=lambda colonnes:colonnes[2])
    return LGN
```

Question 7 Utiliser le module `time` pour comparer les temps de traitement pour trier le classement à l'issu de la dernière étape selon votre algorithme de tri modifié et selon la fonction `sorted`.

```
def comparer_tri_liste(LGN,nom_de_fichier):
    temps_insertion=[]
    temps_rapide=[]
    temps_fusion=[]
    temps_sorted=[]
    for k in range(2,len(LGN)):
        LGT1=LGN[:k+1]
        LGT2=LGN[:k+1]
        LGT3=LGN[:k+1]
        LGT4=LGN[:k+1]
        tic=t.time()
        tri_fusion_modifie(LGT1,0,len(LGT1))
        toc=t.time()
        temps_fusion.append(toc-tic)
        tic=t.time()
        tri_insertion_modifie(LGT2)
        toc=t.time()
        temps_insertion.append(toc-tic)
        tic=t.time()
        tri_rapide_modifie(LGT3,0,len(LGT3))
        toc=t.time()
        temps_rapide.append(toc-tic)
        tic=t.time()
        sorted(LGT4,key=lambda colonnes:colonnes[2])
        toc=t.time()
        temps_sorted.append(toc-tic)

    plt.clf()
    plt.plot(list(range(2,len(LGN))),temps_fusion,'g-',label='fusion')
    plt.plot(list(range(2,len(LGN))),temps_rapide,'b*-',label='rapide')
    plt.plot(list(range(2,len(LGN))),temps_insertion,'r--',label='insertion')
    plt.plot(list(range(2,len(LGN))),temps_sorted,'k.-',label='sorted')
    plt.legend()
    plt.savefig(nom_de_fichier)

LGN=update_classement_general(LG,L18)
comparer_tri_liste(LGN,'tp09_durif_compare_tri1.png')
```



Question 8 Comparer les différentes méthodes de tri en traçant en fonction de la taille du tableau à traiter le temps nécessaire au tri des données. On pourra pour cela utiliser un tranchage. Vous pourrez renvoyer la figure tracée à vos enseignants.

On vérifie bien que dans ce cas la méthode par insertion est la plus adaptée.
La méthode sorted semble encore plus efficace.

Question 9 Facultatif : comparer les différentes méthodes de tri vues en cours appliquée sur une liste aléatoire (comme donnée précédemment) en traçant en fonction de la taille du tableau à traiter le temps nécessaire au tri des données. On pourra pour cela utiliser un tranchage. Vous pourrez renvoyer la figure tracée à vos enseignants.

Si on compare les méthodes de tri pour des listes de nombre aléatoires, voici les résultats obtenus.

