TP 10

Quelques algorithmes de tris

Activité 1 – Tri par comptage

On suppose que la liste à trier `L` est constituée d'entiers de l'intervalle $[\![0;k[\![$. L'algorithme fonctionne suivant le principe suivant :

1. on parcourt une fois la liste et on compte le nombre d'éléments de la liste égaux à 0, 1, ..., $k-1$. Pour ce faire on utilise une liste `C` de taille k ;
2. on procéde alors à une réécriture de la liste initiale, de sorte qu'en sortie elle soit constituée des mêmes éléments, mais triés dans l'ordre croissant.

Ainsi, si on cherche à trier la liste `L=[2,1,4,1]`. Dans un premier temps on crée une liste `C` contenant cinq fois la valeur 0 : `C=[0,0,0,0,0]`. Une fois le comptage terminé on obtient la liste suivante : `C=[0,2,1,0,1]`. La liste triée sera donc constituée de 2 fois la valeur 1 puis 1 fois la valeur 2 puis une fois la valeur 4 soit `C=[0,0,1,1,2]`.

L'algorithme prend en entrée la liste L à trier, ainsi qu'un entier k tel que tous les éléments de la liste soient des entiers de l'intervalle $[\![0;k[\![$. On procède en deux étapes : d'abord compter les éléments de chaque type, ensuite réécrire la liste L.

Question 1 *Ecrire la fonction* `tri_comptage(L:list,k:int) -> None` *permettant de réaliser un tri par comptage avec effet de bord.*

Question 2 *Ecrire la fonction* `tri_comptage_02(L:list,k:int) -> list` *permettant de réaliser un tri par comptage sans effet de bord.*

Définition Tri en place – Wikipedia Un tri est dit en place s'il n'utilise qu'un nombre très limité de variables et qu'il modifie directement la structure qu'il est en train de trier. Ceci nécessite l'utilisation d'une structure de donnée adaptée (un tableau par exemple). Ce caractère peut être très important si on ne dispose pas de beaucoup de mémoire.

Définition Tri stable – Wikipedia Un tri est dit stable s'il préserve l'ordonnancement initial des éléments que l'ordre considère comme égaux.

Question 3 *Les fonctions proposées réalisent-t-elles un tri stable ? un tri en place ?*

```
def tri_comptage(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

# Il s'agit ici d'un tri avec effet de bord et non stable.
```

Activité 2 – Tri fusion

Le tri fusion d'une liste se base sur le principe suivant :

1. on sépare la liste en 2 listes de longueurs quasi-égales (à un élément près) ;

2. on trie ces deux listes en utilisant le tri fusion (par un appel récursif) ;
3. à partir de deux listes triées, on les fusionne en une seule liste en conservant l'ordre croissant.

Question 4 *Implémenter la fonction* `separe(L : list) -> tuple[list,list]` *renvoyant deux listes de longueurs quasi-égales.*

```
def separe(L: list) -> tuple[list,list]:
    return L[:len(L) // 2], L[len(L) // 2:]
```

Question 5 *Proposer 3 tests unitaires permettant de vérifier le bon fonctionnement de votre fonction.*

Question 6 *Implémenter la fonction* `fusion(L1 : list, L2 : list) -> list` *réalisant la fusion de deux listes triées. Pour cela on procèdera ainsi :*

1. *si une des listes est vides on renvoie l'autre ;*
2. *sinon :*
   (a) *si* `L1[0] <L2[0]` *on fusionne, par récursivité,* `L1[0]`, `L[1:]` *et* `L2` *;*
   (b) *sinon, on réfléchit et on fusionne ce qu'il y a fusionner ;).*

```
def fusion(L1: list, L2: list) -> list:
    if not L1 or not L2: # si l'une des listes est vide (éventuellement les 2)
        return L1 or L2 # alors on renvoie l'autre (éventuellement vide aussi)
    else:
        a, b = L1[0], L2[0]
        if a < b : # sinon on compare leurs premiers éléments
            return [a] + fusion(L1[1:], L2) # on place le plus petit en tête et on fusionne le ↙
                reste
        else:
            return [b] + fusion(L1, L2[1:])
```

Question 7 *Proposer 3 tests unitaires permettant de vérifier le bon fonctionnement de votre fonction.*

Question 8 *Implémenter la fonction* `tri_fusion(L : list) -> list` *réalisant par récursivité le tri de la liste* `L`.

```
def tri_fusion(L: list) -> list:
    if len(L) < 2: # cas d'arrêt
        return L
    L1, L2 = separe(L) # sinon on sépare
    return fusion(tri_fusion(L1), tri_fusion(L2)) # et on fusionne les sous-listes triées
```

Activité 3 – Comparaison des tris

Objectif L'objectif est de comparer les différents tris en comptant le nombre d'opérations élémentaires réalisées.

Question 9 *Copier – coller les algorithmes de tri par sélection et de tri rapide réaliser avec Capytale.*

```
def compteur():
    global C
    C = C+1
```

Question 10 *Copier – coller la fonction ci-dessus dans votre script et ajouter l'appel* `compteur()` *une fois pour chacun des tris, au niveau de l'instruction étant réalisée le plus grand nombre de fois.*

Soit les instructions suivantes.

```
import matplotlib.pyplot as plt
import random as rd

les_i = []
les_selection = []
```

```
for i in range(1,100,10):
    les_i.append(i)
    L = [rd.randrange(0,i) for x in range(i)]

    C = 0
    tri_comptage(L.copy())
    les_comptage.append(C)

plt.plot(les_i,les_comptage,label='Comptage')
plt.grid()
plt.legend()
plt.show()
```

Question 11 *Vérifier que ces lignes sont fonctionnelles. Commenter ces lignes par blocs d'instructions.*

Question 12 *Adapter ces lignes pour comparer chacune des méthodes de tri. Conclure*