TP 11 : Algorithmes de tri (partie 2)

Correction

Exercice 1 (Tri par dénombrement) 1. a) On parcourt une fois la liste par compréhension et on compte les apparitions de l'élément recherché.

```
N=15  # On donne une valeur pour N avant de commencer !!

def nbapparitions(L,i):
    c=0      # compteur
    for e in L:
        if e==i:
            c=c+1
    return c
```

b) Directement, par compréhension, on écrit :

```
def comptage(L):
    return [nbapparitions(L,i) for i in range(N)]
```

c) L'exécution de `nbapparitions(L,i)` réalise `len(L)` tests d'égalité (un pour comparer chaque élément de `L` à `i`). Celles-ci sont exécutées pour `i` variant de `0` à `N-1` donc il y a `N x len(L)` tests d'égalité réalisés lors de l'exécution de `comptage(L)`.

2. La fonction ci-après ne parcourt qu'une seule fois la liste `L`. Il y a `len(L)` tests d'égalité réalisés.

```
def comptage2(L):
    r=[0 for i in range(N)]  # liste de N+1 compteurs
    for e in L:
        r[e]=r[e]+1
    return r
```

3. Trois méthodes proposées.

```
def tri_denombrement(L):
    apparition = comptage2(L)
    liste_triee = []
    for i in range(N):
        for nb in range(apparition[i]):          # boucle for
            liste_triee.append(i)                # on utilise .append()
    return liste_triee
```

```
def tri_denombrement2(L):
    apparition = comptage2(L)
    liste_triee = []
    for i in range(N):
        liste_triee = liste_triee + [i for nb in range(apparition[i])]
        # concaténation de listes
    return liste_triee
```

```
def tri_denombrement3(L):
    apparition =comptage2(L)
     liste_triee =[]
    for i in range(N):
         liste_triee .extend([i for nb in range( apparition [i]) ])
    return liste_triee
```

Exercice 2 (Tri par fusion) 1. Si `n=len(L)//2`, il suffit d'exécuter `L[:n]` et `L[n:]`.

2. Version itérative :

```
def fusion_ite (L1,L2):
    resultat =[]
    n1=len(L1)
    n2=len(L2)
    i=0 # Nombre d'éléments pris dans L1, L1[i] est le premier élément visible
    j=0 # Nombre d'éléments pris dans L2, L2[j] est le premier élément visible
    while i != n1 and j != n2:
        if L1[i] < L2[j]:
            resultat .append(L1[i])
            i=i+1
        else :
            resultat .append(L2[j])
            j=j+1
    return resultat +L1[i:]+L2[j :] # L1[i :] ou L2[j :] étant vide
```

3. Version récursive :

```
def fusion_rec (L1,L2):
    if len(L1) == 0:
        return L2
    if len(L2) == 0:
        return L1
    else :
        if L1[0] <= L2[0]:
            return [L1[0]] + fusion_rec(L1[1:], L2)      # concaténation de listes
        else:
            return [L2[0]] + fusion_rec(L1,L2[1:])       # concaténation de listes
```

4. On écrit :

```
def tri_fusion (L):
    if len(L)==1 or len(L)==0 :
        return(L)
    else :
        n=len(L)//2
        return fusion( tri_fusion (L[:n]), tri_fusion (L[n:]) )
```

Exercice 3 (Tri rapide) On écrit :

```
def tri_rapide (L):
    if len(L)==0 :
        return L
    else :
        pivot=L[0]
        Linf=[]  # partie inférieure
        Lsup=[]  # partie supérieure
        for x in L[1:] :
            if x < pivot :
                Linf.append(x)
            else :
                Lsup.append(x)
        return tri_rapide (Linf)+[pivot]+ tri_rapide (Lsup)
```

Exercice 4 (Version en place du tri rapide) a) Partitionnement.

```
def partition (L):
    n=len(L
    pivot=t[0]        # 1er élément de L
    m = 0             # compteur d'éléments inférieurs au pivot (=position du pivot)
    for i in range(1,n): # On parcourt tous les éléments de L pour les comparer au pivot.
        if L[i]< pivot :   # Si L[i] doit aller dans la partie inférieure :
            m = m+1                # on acutalise m,
            (L[i],L[m])=(L[m],L[i]) # on déplace L[i] dans la partie inférieure.
    (L[0],L[m])=(L[m],L[0]) # On met le pivot à sa place dans la liste triée.
    return m                # On retourne l'emplacement du pivot.
```

b) Version en place du tri rapide.

```
def tri_rapide_en_place (L):
    if len(L)==0:
        return L
    else :
        m=partition(L)
        return tri_rapide2 (L[:m])+[L[m]]+tri_rapide2(L[m+1:])
```