TP 7 : Algorithmes de tri (partie 1)

Correction

Exercice 1 1. Après avoir importé la bibliothèque `random`, par compréhension, il suffit d'écrire :

```
1 L = [randint(0, 100) for i in range(10)]
```

On prendra garde au fait que `randint(a,b)` retourne un entier compris entre a et b inclus.

2. La fonction `sorted()` retourne une copie triée de la liste donnée en argument sans modifier la liste initiale.

```
>>> L= [12,43,5]
>>> M=sorted(L)
>>> M
[5,12,43]
>>> L
[12,43,5]
```

La méthode `sort()` modifie en place la liste donnée en argument, ie la liste est modifiée, et ne retourne rien.

```
>>> L=[12,43,5]
>>> L.sort()
>>> L
[5,12,43]
```

La fonction `sorted()` peut également prendre en argument d'autres listes que des listes d'entiers ou flottants. Par exemple :

```
>>> Lstr=['Truc','Chose','Machin','Bidule']
>>> Mstr=sorted(Lstr)
>>> Mstr
 ['Bidule','Chose','Machin','Truc']
```

Exercice 2 On propose plusieurs versions itératives (avec un ou plusieurs return) et une version récursive (à l'utilité pratique douteuse).

```
1 def croissante(L):
2     for i in range(len(L)-1):
3         if L[i]>L[i+1]:
4             return False
5     return True
6
7
```

```
8  def croissante2(L) : # un seul return
9      k=0
10     while k < len(L)-1 and L[k] <= L[k+1] :
11         k=k+1
12     return k==len(L)-1 # retourne un booléen
13
14 def croissante3(L): # un seul return
15     rep=True
16     for i in range(len(L)-1):
17         if L[i]>L[i+1]:
18             rep = False
19     return rep
20
21 def croissante_rec(L) : # Une version récursive qui joue avec les booléens.
22     if len(L)==1:
23         return True
24     else :
25         return L[0] <=L[1] and croissante(L[1:])
```

Exercice 3 (Tri par sélection) 1. On compare chaque nouvel élément de `L` au plus petit déjà rencontré.

```
1  def pos_mini(L):
2      m=L[0]  # minimum glissant
3      pos=0  # indice du minimum glissant
4      for i in range(len(L)):
5          if L[i]<m:  # Si on trouve un élément plus petit que le plus petit rencontré,
6              m=L[i]  # on actualise m,
7              pos=i  # on actualise pos.
8      return pos
```

2. On peut effectuer l'échange simultané sans introduire de nouvelle variable.

```
1  def echange(L,i,j):
2      (L[i],L[j])=(L[j],L[i])
```

3. On écrit :

```
1  def  tri_selection (L):
2      for i in range(len(L)):
3          m=pos_mini(L[i:])
4          echange(L,i,m+i)
5      return L
```

4. L'exécution de `pos_mini(L[i:])` réalise autant de comparaisons qu'il y a d'éléments dans la sous-liste `L[i:]`, à savoir `len(L)-i` comparaisons. Lors de l'exécution de tri_selection(L), sont exécutées successivement `len(L)`, `len(L)-1`, `len(L)-2`,..., 1 comparaisons, ie $\frac{n(n+1)}{2} = O(n^2)$ comparaisons où $n =$`len(L)`. La complexité du tri par sélection est quadratique en le nombre d'éléments de la liste à trier.

Exercice 4 Voici deux versions pour le tri par insertion.

```
def tri_insertion (L):
    for i in range(0,len(L)):
        j=i
        aplacer=L[i]        # On sauvegarde l'élément à insérer.
        while j > 0 and L[j-1]>aplacer:
            L[j]=L[j-1]     # On décale vers la droite.
            j=j-1           # On passe à l'élément précédent.
        L[j]=aplacer        # On insère l'élément.
    return(L)

def tri_insertion2 (L):
    for i in range(1, len(L)):      # On insère L[i] dans L[0:i].
        j = i
        while j > 0 and L[j-1] > L[j]:
            (L[j], L[j-1]) = (L[j-1], L[j])
            j = j-1
    return L
```

Exercice 5 (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 nbapparition(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 [nbapparition(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 6 1. On écrit :

```
def chiffres(n):
    L=[]
    m=n
    while m!=0:
        L.append(m%10)
        m=m//10
    return L
```

2. On écrit :

```
def cdr(n,r):
    L=chiffres(n)
    p=len(L)
    if r < p:
        return L[r]
    else :
        return 0
```

3. a) On écrit :

```
def emboite(L,r):
    Rep=[[] for i in range(10)]
    for x in L:
        a=cdr(x,r)
        Rep[a].append(x)
    return Rep
```

b) Trois méthodes sensiblement identiques :

```
def deboite(L,r):
    Rep=[]
    L1=emboite(L,r)
    for l in L1:
        Rep.extend(l)
    return Rep

def deboite2(L, r):
    Rep = []
    L1 = emboite(L, r)
    for l in L1:
        for x in l:
            Rep.append(x)   # On utilise la méthode append.
    return Rep

def deboite3(L, r):
    Rep = []
    L1 = emboite(L, r)
    for l in L1:
        Rep = Rep + l       # concaténation de listes
    return Rep

```

4. On écrit :

```
def tri(L):
    Rep=L
    M=max(L)
    n=len(str(M)) # Pour savoir combien de fois itérer
    for i in range(n): # on commence par déboiter les unités, puis les dizaines, etc
        Rep=deboite(Rep,i)
    return Rep
```

5. a) On écrit :

```
import random
testaleatoire =[random.randint(0,10000) for i in range(5000)]
```

b) On exécute :

```
import time
# chrono de tri (Pour mémoire, 0.468s avec mon vieux PC)
t0=time.time()
tri( testaleatoire )
t1=time.time()
print(t1-t0)
# chrono de sorted (Pour mémoire, 0.0156s avec vieux mon PC)
t2=time.time()
sorted( testaleatoire )
t3=time.time()
print(t3-t2)
```

On obtient 0.468s avec un vieux PC / 0.03s à 0.05s avec un PC plus récent pour t1-t0.
On obtient 0.0156s avec un vieux PC / 0.0s à 0.01s avec un PC plus récent pour t3-t2.