

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 $n = \text{len}(L)$. 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.

```

1 def tri_insertion (L):
2     for i in range(0, len(L)):
3         j=i
4         aplacer=L[i]      # On sauvegarde l'élément à insérer.
5         while j > 0 and L[j-1]>aplayer:
6             L[j]=L[j-1]   # On décale vers la droite.
7             j=j-1        # On passe à l'élément précédent.
8         L[j]=aplayer     # On insère l'élément.
9     return(L)
10
11 def tri_insertion2 (L):
12     for i in range(1, len(L)):  # On insère L[i] dans L[0:i].
13         j = i
14         while j > 0 and L[j-1] > L[j]:
15             (L[j], L[j-1]) = (L[j-1], L[j])
16             j = j-1
17     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.

```

1 N=15 # On donne une valeur pour N avant de commencer !!
2
3 def nbapparition(L,i):
4     c=0 # compteur
5     for e in L:
6         if e==i:
7             c=c+1
8     return c

```

b) Directement, par compréhension, on écrit :

```

1 def comptage(L):
2     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.

```

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

```

3. Trois méthodes proposées.

```

1 def tri_denombrement(L):
2     apparition = comptage2(L)
3     liste_triee = []
4     for i in range(N):
5         for nb in range(apparition[i]): # boucle for
6             liste_triee.append(i)      # on utilise .append()
7     return liste_triee
8
9 def tri_denombrement2(L):
10    apparition = comptage2(L)
11    liste_triee = []
12    for i in range(N):
13        liste_triee = liste_triee + [i for nb in range(apparition[i])]
14        # concaténation de listes
15    return liste_triee
16
17 def tri_denombrement3(L):
18    apparition = comptage2(L)
19    liste_triee = []
20    for i in range(N):
21        liste_triee.extend([i for nb in range(apparition[i])])
22    return liste_triee

```

Exercice 6 1. On écrit :

```

1 def chiffres(n):
2     L=[]
3     m=n
4     while m!=0:
5         L.append(m%10)
6         m=m//10
7     return L

```

2. On écrit :

```

1 def cdr(n,r):
2     L=chiffres(n)
3     p=len(L)
4     if r < p:
5         return L[r]
6     else :
7         return 0

```

3. a) On écrit :

```

1 def emboite(L,r):
2     Rep=[] for i in range(10)
3     for x in L:
4         a=cdr(x,r)
5         Rep[a].append(x)
6     return Rep

```

b) Trois méthodes sensiblement identiques :

```

1  def deboite(L,r):
2      Rep=[]
3      L1=emboite(L,r)
4      for l in L1:
5          Rep.extend(l)
6      return Rep
7
8  def deboite2(L, r):
9      Rep = []
10     L1 = emboite(L, r)
11     for l in L1:
12         for x in l:
13             Rep.append(x) # On utilise la méthode append.
14     return Rep
15
16 def deboite3(L, r):
17     Rep = []
18     L1 = emboite(L, r)
19     for l in L1:
20         Rep = Rep + l # concaténation de listes
21     return Rep
22

```

4. On écrit :

```

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

```

5. a) On écrit :

```

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

```

b) On exécute :

```

1  import time
2  # chrono de tri (Pour mémoire, 0.468s avec mon vieux PC)
3  t0=time.time()
4  tri ( testaleatoire )
5  t1=time.time()
6  print (t1-t0)
7  # chrono de sorted (Pour mémoire, 0.0156s avec vieux mon PC)
8  t2=time.time()
9  sorted( testaleatoire )
10 t3=time.time()
11 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.