Corrigé

Code de partage avec Capytale : 8661-2397891

Exercice 1 - faire le tri par recherche de minimum (retour sur le TP précédent)

1. a) Exécuter et tester à nouveau les programmes suivants

```
def minimum(L) :
    min=L[0]
    indice=0
    for i in range(1,len(L)):
        if L[i]<min :
            min=L[i]
            indice=i
    return [indice, min]

def echange(L):
    indice_min=minimum(L)[0]
    m=L[indice_min]
    L[indice_min]=L[0]
    L[0]=m
    return L
```

2. a) A l'aide de la fonction `echange`, écrire un programme qui trie les valeurs d'une liste (numérique) par ordre croissant.

On va répéter le processus d'échange en plaçant dans le premier temps le minimum global de la liste en première position, puis on applique le même processus au $n-1$ éléments restant de la liste (on place en deuxième position le « deuxième minimum » de liste
La commande `L[i:]` permet de ne sélectionner que les éléments de la liste à partir du $i^{\text{ème}}$ (inclus).
Il faut ensuite recoller les morceaux : on redéfinit `L` avec le début de la liste déjà trié au préalable et le reste de la liste qui contient le minimum de la liste restante puis les éléments non triés. On peut tester avec `tri([1,3,2,5,4,0])`

```
def tri(L):
    for i in range(0,len(L)-1):
        L=L[:i]+echange(L[i:])
    return L
```

b) Combien d'opérations ou tests effectue ce dernier programme ?

En arrondissant, et en considérant que la liste est de taille n, le programme effectue n opérations (pour trouver le minimum de la liste entière), puis $n-1$ pour trouver le minimum des $n-1$ restants, puis $n-2$...
donc au total $n+(n-1)+(n-2)+\cdots+2+1=\dfrac{n(n+1)}{2}$, on dit dans ce cas que la complexité du programme est en n^2. Pour illustrer cela, on comprend qu'avec une liste de 1 000 éléments, le programme va engendrer de l'ordre de 1 000 000 d'opérations.

Exercice 2 - tri à bulles

1. Ecrire une fonction qui prend en entrée une liste et renvoie la liste où l'ordre de ses valeurs a été modifié de telle sorte qu'en parcourant toute la liste initiale, une valeur est permutée avec la suivante si elle est plus grande (cf. illustration avec les amidakujis).

On parcourt la liste et on compare chaque valeur avec la suivante. On effectue la permutation dès lors que les deux valeurs ne sont pas rangés dans l'ordre croissant (comme pour la fonction `echange` on créée une variable auxiliaire pour ne pas écraser la valeur dans `L[i]`

```
def bulle(L):
  for i in range(0,len(L)-1):
    if L[i]>L[i+1]:
      a=L[i]
      L[i]=L[i+1]
      L[i+1]=a
  return L
```

2. A l'aide de la fonction précédente, écrire une fonction qui prend une liste en entrée et renvoie la liste triée par ordre croissant.

   Comme on sait que dès le premier passage, la plus grande valeur va se placer à la fin, au deuxième, la deuxième plus grande valeur va se placer en avant-dernière position, à chaque passage on trie une valeur et donc en n « remontées de bulles », la liste sera triée.

```
def tri(L):
    for i in range(0,len(L)):
        L=bulle(L)
    return L
```

3. Combien d'opérations ou tests effectue ce dernier programme ?

   On applique la fonction `bulle` autant de fois qu'il y a de termes dans la liste, donc n fois si n est la longueur de la liste. Mais la fonction bulle effectue également n opérations ($n-1$ en fait), donc finalement ce tri effectue un nombre d'opérations (ou de tests) de l'ordre de n^2, son efficacité est donc similaire au tri par recherche de minimum.