LES PROCEDURES DE TRIS

1 Tri par sélection

On parle parfois de « tri naïf »parce que c'est l'une des plus intuitives.

Soit L la liste à trier de longueur n. L'idée est de commencer par chercher le plus grand élément de la liste puis de le placer à la n-ième position en l'échangeant avec le dernier élément de celle-ci.
Il suffit alors de recommencer l'opération avec les $n-1$ premiers éléments puisque le dernier élément de la liste est correctement placé. On réitère cette opération jusqu'à ce qu'il ne reste plus que deux éléments à trier.

① Écrire une fonction `maxi(L,k)` qui prend en argument une liste `L` et un entier `k` et qui retourne un indice du maximum parmi les k premiers éléments de la liste.

② En utilisant la fonction `maxi`, écrire une fonction `tri_selec` qui trie une liste selon la méthode décrite ci-dessus.

solution - tri par sélection

```
def maxi(L,k)
    indice=0
    for i in range(k):
       if L[i]>L[indice]:
          indice=i
    return indice
```

```
def tri_select(L):
    i=len(L)-1
    while i>0:
       j=maxi(L,i+1)
       if j != i:
          L[j],L[i]=L[i],L[j]
       i -= 1
    return L
```

2 Tri à bulles

Cette méthode consiste à parcourir une première fois l'ensemble du tableau et à faire « remonter » son plus grand élément en dernière place à l'aide d'échanges deux à deux.
On recommence ensuit cette opération sur les $(n-1)$ premiers termes du tableau, les $(n-2)$ premiers termes, etc. jusqu'aux deux premiers termes d'indices 0 et 1.

Exemple : On considère la liste `L=[4,6,3,7,1]`. On obtiendra successivement :

première étape = première « bulle » :
`[4,6,3,7,1]` # L[0]<L[1] : on ne fait rien
`[4,3,6,7,1]`# L[1]>=L[2] : on n'a échangé 3 et 6
`[4,3,6,7,1]` # L[2]<L[3] : on ne fait rien
`[4,3,6,1,7]` # L[3]>=L[4] : on n'a échangé 1 et 7

seconde étape = deuxième « bulle » : on s'intéresse aux 4 premiers termes
`[3,4,6,1,7]`# L[0]>=L[1] : on n'a échangé 4 et 3
`[3,4,6,1,7]`# L[1]<L[2] : on ne fait rien
`[3,4,1,6,7]`# L[2]>=L[3] : on n'a échangé 6 et 1

troisième étape = troisième « bulle » : on s'intéresse aux 3 premiers termes
`[3,4,1,6,7]`# L[0]<L[1] : on ne fait rien
`[3,1,4,6,7]`# L[1]>=L[2] : on n'a échangé 4 et 1

quatrième étape = 4-ième « bulle » : on s'intéresse aux 2 premiers termes
`[1,3,4,6,7]`# L[0]>=L[1] : on n'a échangé 3 et 1

```
def tri_bulles(L):
    n=len(L)
    for k in range(n-1,0,-1):# k de n-1 à 1 par pas de -1
        for i in range(k): # i de 0 à k-1
            if L[i]>L[i+1]:
                L[i],L[i+1]=L[i+1],L[i] # échange
    return L
```

3 Tri par insertion

Cette méthode de tri est celle du joueur de carte qui reçoit successivement les cartes qui lui sont distribuées. Une fois la première carte prise, il insère la seconde à sa place pour un tri croissant. Ensuite, à chaque nouvelle carte, il l'insère dans l'ordre en plaçant dans sa main triée chaque élément à sa bonne place.

Pour programmer un tel tri, on traite les éléments de la liste tour à tour. On les compare aux éléments précédents préalablement triés jusqu'à trouver la sa place. Il suffit alors de décaler les éléments du tableau pour insérer l'élément considéré à sa place.

Exemple : On considère la liste `L=[4,6,3,5,7,1]`. On obtiendra successivement :

```
[4, ,6,3,5,7,1]
[4,6, ,3,5,7,1]
[3,4,6, ,5,7,1]
[3,4,5,6, ,7,1]
[3,4,5,6,7, ,1]
[1,3,4,5,6,7, ]
```

① Écrire une fonction `insertion(L,i)` qui prend en argument une liste `L` dont on suppose que les $(i-1)$ premiers éléments sont triés et qui insère l'élément `L[i]` à sa place pour que les i premiers éléments de `L` soient triés.

② Écrire une fonction `tri_insert(L)` qui effectue un tri par insertion de la liste `L`.

```
def insertion(L,i):
   while i>0 and L[i]<L[i-1]:
         L[i-1],l[i]=L[i],L[i-1]
         i -= 1
   return L
```

```
def tri_insert(L):
    n=len(L)
    for i in range(1,n):
       L=insertion(L,i)
    return L
```

4 Tri rapide - quickSort

Le *tri rapide* est une méthode récursive de tri.
Elle consiste à prendre un élément au hasard (le plus souvent le premier de la liste) appelé *pivot* et de le mettre à sa place en plaçant tous les éléments plus petits à sa gauche et les plus grands à sa droite.
On recommence alors sur les deux sous-listes obtenues jusqu'à obtenir des sous-listes vides. La liste est alors triée.

Exemple :

- On suppose `L=[15,19,3,20,5,1,6,11]`
- Le pivot est 15. On place dans `L1` les éléments qui lui sont plus petits, dans `L2` sont qui lui sont plus grands.

$$[3,5,1,6,11] + \boxed{[15]} + [19,20]$$

- On répète la même opération sur les deux sous-listes `L1` et `L2` :

$$[1] + \boxed{[3]} + [5,6,11] + \boxed{[15]} + \boxed{[19]} + [20]$$

- On termine en traitant de la même manière les trois sous-listes restantes.

Exercice : Ecrire une fonction Python récursive `triRapide(L)` permettant de trier une liste `L` selon cette procédure.

```
def trirapide(L):
    if L==[]:
        return []
    else:
        n=len(L)
        L1=[]
        L2=[]
        for k in range(1,n):
            if L[k]<=L[0]:
            L1.append(L[k]) #L1 reçoit les éléments les plus petits
        else:
            L2.append(L[k]) # L2 reçoit les éléments les plus grands
    L=trirapide(L1)+[L[0]]+trirapide(L2)
    return L
```