TP n° 07 – Algorithmes de tri

I Tri par sélection

Le tri par sélection (ou tri par extraction) est un algorithme de tri par comparaison. Cet algorithme est simple, mais considéré comme inefficace car il s'exécute en temps quadratique en le nombre d'éléments à trier, et non en temps pseudo linéaire.

I.1 Pseudo-code

Sur un tableau de n éléments, le principe du tri par sélection est le suivant :

1. rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0,
2. rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1,
3. continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exercice 1. Proposer une fonction `tri_selection(liste)` qui permet de trier `liste`. Attention, la fonction sera utilisée comme suit :

```
print(liste) # la liste n'est pas triée
tri_selection(liste)
print(liste) # la liste est triée
```

Il n'y a donc pas besoin de mettre un `return` dans la fonction.

Solution 1.

```
def tri_selection(liste):
    # Parcours de 1 à la taille de la liste
    for i in range(len(liste)-1):
        # Initialiser le min
        min=liste[i]
        jmin=i
        for j in range(i, len(liste)):
            # Chercher le min
            if liste[j]<min:
                jmin=j
                min=liste[j]
        # Permuter le min et l'élément i
        liste[i],liste[jmin]=liste[jmin],liste[i]
```

Exercice 2. Tester cette fonction sur la liste `[6,3,4,2,7,1,5]`.

Solution 2.

```
liste=[6,3,4,2,7,1,5]
tri_selection(liste):
print(liste)
```

II Tri par insertion

En informatique, le tri par insertion est un algorithme de tri classique. La plupart des personnes l'utilisent naturellement pour trier des cartes à jouer.
En général, le tri par insertion est lui aussi assez lent car sa complexité asymptotique est quadratique.
Le tri par insertion considère chaque élément du tableau et l'insère à la bonne place parmi les éléments déjà triés. Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés.

II.1 Pseudo-code

Sur un tableau de n éléments, le principe du tri par insertion est le suivant :

1. pour tout i des n éléments de la liste,
2. on décale l'élément i vers la gauche tant que l'élément précédent est plus petit que lui.

Exercice 3. Proposer une fonction `tri_insertion(liste)` qui permet de trier `liste`. Attention, la fonction sera utilisée comme suit :

```
print(liste) # la liste n'est pas triée
tri_selection(liste)
print(liste) # la liste est triée
```

Il n'y a donc pas besoin de mettre un `return` dans la fonction.

Solution 3.

```
def tri_insertion(liste):
    # Parcours de 1 à la taille de la liste
    for i in range(1, len(liste)):
        # On mémorise la position initiale de l'élément
        k = liste[i]
        # On décale cet élément vers la gauche tant que
        # l'élément précédent est plus grand que lui
        j = i-1
        while j >= 0 and k < liste[j] :
                liste[j + 1] = liste[j]
                j -= 1
        liste[j + 1] = k
```

Exercice 4. Tester cette fonction sur la liste `[6,3,4,2,7,1,5]`.

Solution 4.

```
liste=[6,3,4,2,7,1,5]
tri_insertion(liste):
print(liste)
```

III Tri à bulles

Le tri à bulles ou tri par propagation consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés.
Il doit son nom au fait qu'il déplace rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide.

III.1 Pseudo-code

On donne les états successifs de la liste [6, 3, 4, 2, 7, 1, 5] traitée par l'algorithme de tri à bulles.

```
[6, 3, 4, 2, 7, 1, 5]
[3, 6, 4, 2, 7, 1, 5]
[3, 4, 6, 2, 7, 1, 5]
[3, 4, 2, 6, 7, 1, 5]
[3, 4, 2, 6, 1, 7, 5]
[3, 4, 2, 6, 1, 5, 7]
[3, 2, 4, 6, 1, 5, 7]
[3, 2, 4, 1, 6, 5, 7]
[3, 2, 4, 1, 5, 6, 7]
[2, 3, 4, 1, 5, 6, 7]
[2, 3, 1, 4, 5, 6, 7]
[2, 1, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
```

Exercice 5. Expliquer l'algorithme et donner le pseudo code qui a été suivit pour réaliser ces étapes.

Solution 5.
L'algorithme parcourt le tableau et compare les éléments consécutifs. Lorsque deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.
Après un premier parcours complet du tableau, le plus grand élément est forcément en fin de tableau, à sa position définitive. En effet, aussitôt que le plus grand élément est rencontré durant le parcours, il est mal trié par rapport à tous les éléments suivants, donc échangé à chaque fois jusqu'à la fin du parcours.
Après le premier parcours, le plus grand élément étant à sa position définitive, il n'a plus à être traité. Le reste du tableau est en revanche encore en désordre. Il faut donc le parcourir à nouveau, en s'arrêtant à l'avant-dernier élément. Après ce deuxième parcours, les deux plus grands éléments sont à leur position définitive. Il faut donc répéter les parcours du tableau, jusqu'à ce que les deux plus petits éléments soient placés à leur position définitive.
Le pseudo-code est le suivant :
- pour tout i de 1 (0 n'a pas de précédent) à n,
- on parcours tous les j de 0 à i (les précédents),
- on permute l'élément j et $j+1$ si $j > j+1$.

Exercice 6. Proposer une fonction `tri_bulle(liste)` qui permet de trier `liste`. Attention, la fonction sera utilisée comme suit :

```
print(liste) # la liste n'est pas triée
tri_selection(liste)
print(liste) # la liste est triée
```

Il n'y a donc pas besoin de mettre un `return` dans la fonction.

Solution 6.
```
def tri_bulle(liste):
    # Parcours de 1 à la taille de la liste
    for i in range(1, len(liste)):
        # Parcours des éléments précédents
        for j in range(0, len(liste)-i-1):
            # On permutte les deux éléments successifs
            if liste[j] > liste[j+1] :
                liste[j], liste[j+1] = liste[j+1], liste[j]
```

Exercice 7. Tester cette fonction sur la liste `[6,3,4,2,7,1,5]`.

Solution 7.
```
liste=[6,3,4,2,7,1,5]
tri_bulle(liste):
print(liste)
```

IV Déterminer la fonction la plus efficace

Le code suivant permet de déterminer la durée d'exécution d'une instruction.

```
import time

start = time.process_time()
# instruction
end = time.process_time()
print(end - start)
```

Exercice 8. Utiliser cette fonction pour mesurer la durée du tri avec chaque fonction.

Solution 8.
```
liste=[6,3,4,2,7,1,5]
start = time.process_time()
tri_selection(liste)
end = time.process_time()
print(end - start)
```

```
liste=[6,3,4,2,7,1,5]
start = time.process_time()
tri_insertion(liste)
end = time.process_time()
print(end - start)

liste=[6,3,4,2,7,1,5]
start = time.process_time()
tri_bulle(liste)
end = time.process_time()
print(end - start)
```

Exercice 9. A quoi correspond la liste `liste0=[randrange(size) for i in range(size)]`.

Solution 9.
Cela remplit une liste de taille `size` avec des entiers aléatoires.

Exercice 10. Utiliser ce résultat afin de tester les 3 fonctions précédentes dans plusieurs cas.

Solution 10.

```
t=[[],[],[]]
fonction=[tri_selection,tri_insertion,tri_bulle]
nom_fonction=["Tri par sélection","Tri par insertion","Tri à bulles"]
for i in range(tirages):
    print(i)
    liste0=[randrange(size) for i in range(size)]
    for j in range(3):
        liste = liste0
        #print(liste)
        start = time.process_time()
        fonction[j](liste)
        end = time.process_time()
        #print(liste)
        t[j].append(end - start)

def moyenne(l):
    somme=0
    for i in l:
        somme+=i
    return somme/len(l)

for i in range(3):
    print(min(t[i]),max(t[i]),moyenne(t[i]))
```