

Algorithmes de tri

Nous allons détailler 2 méthodes de tri parmi les nombreuses qui existent.

Rappel : un invariant de boucle est une propriété qui est vraie avant chaque exécution d'une boucle et qui reste vérifiée à chaque itération de la boucle. Elle est donc vraie *avant*, *pendant* et *après* l'exécution de la boucle.

I Tri par sélection :


- Principe : pour toutes les positions i de la première à l'avant dernière, on cherche le plus petit élément de la liste qui commence à i et qui va au bout de la liste : on l'échange avec la valeur en i .

Exemple : on considère la liste suivante en cours de tri

	Partie triée					Partie non triée									
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	3	5	8	9	15	36	23	24	11	29	25	32	48	65	17

- La prochaine étape consiste à trouver la plus petite valeur du tableau qui commence à $i=4$ et à l'échanger avec la valeur en 4.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	3	5	8	9	15	36	23	24	11	29	25	32	48	65	17



- La partie triée est donc triée jusqu'à $i=4$ et on recommence avec $i=5$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	3	5	8	9	11	36	23	24	15	29	25	32	48	65	17

- Python :

```
def tri_selection(liste):
    for i in range(len(liste)-1):
        min=i
        for j in range(i+1,len(liste)):
            if liste[j]<liste[min]:
                min=j
        liste[i],liste[min]=liste[min],liste[i]
```

on parcourt la liste de i=0 à l'avant dernière position
on crée la variable min qui doit contenir l'indice de la plus petite valeur
on parcourt la moitié non-triée de la liste à la recherche du minimum
dès qu'on trouve une valeur inférieure à liste[min], min prend la valeur
de l'indice correspondant
lorsque la partie droite est parcourue, on échange les valeurs

- Invariant de boucle :

A l'entrée de la boucle i , la liste qui va de 0 à $i-1$ est déjà triée **ET** tous les éléments de la liste d'indice supérieur ou égal à i sont supérieurs à l'élément en $i-1$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	3	5	8	9	11	36	23	24	15	29	25	32	48	65	17

A l'entrée de la boucle $i=5$, la liste de $i=0$ à $i=4$ est bien triée et tous les éléments à partir de $i=5$ sont supérieurs à l'élément en $i=4$.

- On en déduit que, lors de la dernière boucle, i étant égal à l'avant dernière position de la liste (ici $i=13$) la liste est triée entre les positions 0 et $m-2$ (ici 12), m étant l'indice max de la liste. Il reste donc les positions $m-1$ et m à trier.
Si la valeur en m est inférieure à celle en $m-1$, elles seront échangées et la liste sera alors complètement triée. L'invariant de boucle est encore vérifié.
Nous avons prouvé la **correction** de l'algorithme.

- Avec la liste donnée en exemple : avant la boucle $i=13$, la liste est dans l'état suivant :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	3	5	8	9	11	15	17	23	24	25	29	32	36	65	48

L'algorithme compare `liste[13]` et `liste[14]` et les échange. On obtient alors :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	3	5	8	9	11	15	17	23	24	25	29	32	36	48	65

- Terminaison : il faut prouver que l'algorithme termine, c'est-à-dire qu'il s'arrête.
L'algorithme est constitué de deux boucles bornées qui vont donc forcément se terminer : il ne peut pas y avoir de boucle infinie.
- Complexité :
 n étant la longueur de la liste, la première boucle est effectuée de 0 à $n-2$ donc $n-1$ fois.
Pour chaque itération de la première boucle, on a une boucle qui va de $i+1$ à $n-1$, donc exécutée $n-1-i$ fois.
Il y a donc $(n-1) + (n-2) + (n-3) + \dots + 1$ comparaisons à effectuer.
Cette somme est égale à $\frac{n(n-1)}{2}$ ce qui équivaut à $\frac{n^2}{2}$.

On dit que la complexité de cet algorithme est *quadratique* car elle dépend du carré de la taille de la liste.

On la note **$O(n^2)$**

II Tri par insertion :

- Principe : pour toutes les positions i de $i=1$ à la dernière, on enregistre la valeur en i ($liste[i]$) et on décale chaque élément de la liste $(0, i-1)$ vers le haut jusqu'à ce qu'on rencontre une valeur inférieure à la valeur en $[i]$.

On insère alors la valeur $liste[i]$ juste au-dessus de cette valeur.

Exemple : on considère la liste suivante au début du tri :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	8	5	2	25	11	36	23	24	15	29	28	32	48	65	17

- La première étape consiste à enregistrer $liste[1]$ dans une variable, ici $liste[1]=5$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	8		2	25	11	36	23	24	15	29	28	32	48	65	17

- On décale vers le haut les éléments précédents jusqu'à trouver une valeur inférieure à 5.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur		8	2	25	11	36	23	24	15	29	28	32	48	65	17

- Il n'y en a pas, on remet la valeur de $liste[1]$ dans la case libérée.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	5	8	2	25	11	36	23	24	15	29	28	32	48	65	17

- Etape suivante : $i=2$, on enregistre $liste[2] = 2$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	5	8	2	25	11	36	23	24	15	29	28	32	48	65	17

- On décale :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	5		8	25	11	36	23	24	15	29	28	32	48	65	17

- On décale :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur		5	8	25	11	36	23	24	15	29	28	32	48	65	17

- On remplace 2 car on n'a pas trouvé de valeur plus faible :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	25	11	36	23	24	15	29	28	32	48	65	17

- On accélère jusqu'à $i=6$. Voici la liste avant la boucle.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11	25	36	23	24	15	29	28	32	48	65	17

- On enregistre liste[6]=23.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11	25	36		24	15	29	28	32	48	65	17

- On décale jusqu'à trouver un élément plus petit, c'est liste[3] :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11		25	36	24	15	29	28	32	48	65	17

- On place liste[6] au-dessus de liste[3] donc en position 4.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11	23	25	36	24	15	29	28	32	48	65	17

Partie triée
Partie non triée

- Python :

```
def tri_insertion(liste):
    for i in range(1, len(liste)):
        valTest = liste[i]                # on enregistre liste[i]
        j = i - 1
        while j >= 0 and liste[j] > valTest:    # on décale en partant de i-1 et en descendant vers 0 chaque
            liste[j+1] = liste[j]              # élément vers la droite tant que les valeurs sont supérieures
            j = j - 1                          # à liste[i]
        liste[j+1] = valTest                  # dès qu'on en trouve une inférieure, on place la valeur de liste[i]
```

- Invariant de boucle :

A l'entrée de la boucle i, la liste qui va de 0 à i-1 est déjà triée.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11	23	25	36	24	15	29	28	32	48	65	17

A l'entrée de la boucle i=7, la liste de i=0 à i=6 est bien triée.

- On en déduit que, lors de la dernière boucle, i étant égal à la dernière position de la liste (ici i=14) la liste est triée entre les positions 0 et m-1 (ici 13), m étant l'indice max de la liste. Il reste donc la dernière valeur à positionner. Elle va être positionnée juste au-dessus de la première valeur qui lui sera inférieure. La liste sera alors triée.

- Avec la liste donnée en exemple : avant la boucle $i=14$, la liste est dans l'état suivant :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11	15	23	24	25	28	29	32	36	48	65	17

On enregistre 17 et on décale jusqu'à trouver une valeur inférieure à 17 :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11	15		23	24	25	28	29	32	36	48	65

On place 17 au-dessus :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
valeur	2	5	8	11	15	17	23	24	25	28	29	32	36	48	65

- Terminaison :

La première boucle est une boucle for qui termine toujours.

La deuxième boucle est une boucle qui termine si j devient négatif, or il est positif au départ et décroît régulièrement : il devient donc forcément négatif, il ne peut pas y avoir de boucle infinie.

- Complexité :

L'estimation est la même que pour le tri par sélection.

Dans le cas d'une liste triée à l'envers (le pire des cas), il y a

$(n-1) + (n-2) + (n-3) + \dots + 1$ comparaisons à effectuer.

C'est encore quadratique et donc $O(n^2)$

Dans le meilleur des cas :

Si la liste est déjà triée, la boucle while ne se déclenche jamais et on ne parcourt la liste que n fois.

On a donc une complexité en $O(n)$

En moyenne : la complexité est en $O(n^2)$ mais elle se rapproche de $O(n)$ lorsque la liste est presque déjà triée.

Exercices :

1. Ecrire une fonction qui renvoie True si une liste passée en argument est triée et False si elle ne l'est pas.
2. La fonction tri_insertion proposée dans ce cours modifie la liste passée en argument. Ecrire une fonction qui, au lieu de la modifier, renvoie une nouvelle liste triée.
3. Ecrire une fonction qui renvoie les n plus petits éléments d'une liste qui en contient plus que n .
4. Ecrire une fonction qui, à partir du tri par insertion, prend comme argument une liste d'entiers et renvoie la valeur la plus fréquente dans cette liste.
5. Ecrire une fonction *occurrences* prenant en argument une liste d'entiers triée et qui renvoie une liste de tuple indiquant le nombre d'occurrences de chaque nombre.

Exemple :

```
liste=[0,0,1,1,1,1,2,3,3,4,4,4,4,5,5,5,6]
```

```
print (occurrences(liste))
```

sortie : [(0,2),(1,5),(2,1),(3,2),(4,4),(5,3),(6,1)]