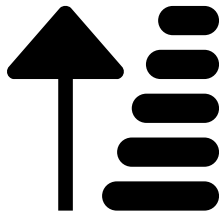


Activité 14 : Algorithmes de tri



Dans cette activité nous allons étudier deux algorithmes de tri, ce qui est surtout un prétexte pour nous interroger sur la complexité des algorithmes.

Nous présentons le tri par sélection et le tri par insertion, et nous nous interrogeons sur l'efficacité de ces deux algorithmes, en évaluant leur temps d'exécution.

1. Rôle du tri en informatique

La recherche d'un élément dans une table est plus rapide quand celle-ci est ordonnée. C'est une chose que nous savons depuis qu'il existe des bibliothèques et des bibliothécaires : même si cela est long et fatigant, il vaut mieux ranger les livres d'une bibliothèque une fois pour toutes, par exemple dans l'ordre alphabétique, plutôt que les laisser en vrac et arpenter des kilomètres de rayonnages à chaque fois que l'on cherche un volume.

L'importance de ce problème fait que plusieurs dizaines d'algorithmes différents ont été proposés pour trier des objets. Cette diversité d'algorithmes se justifie par le fait qu'un même algorithme sera plus ou moins performant selon l'ordre de départ des objets à trier. Cette activité est consacrée à deux d'entre eux : le tri par sélection et le tri par insertion.

L'ensemble des travaux portera sur le tri de tableaux de nombres entiers positifs organisés sous forme de listes. Cependant les algorithmes développés pourront tout aussi bien être appliqués à des nombres réels ou des caractères

2. Le tri par sélection

Principe de tri

Le tri par sélection parcourt le tableau de gauche à droite, en maintenant sur la gauche une partie de la liste triée et à sa place définitive :

2	4	8	10	3	7	20	6
Liste triée				Liste non triée			

A chaque étape, on cherche le plus petit élément de la partie droite (non triée) puis on l'échange avec l'élément le plus à gauche de la partie non triée. Une illustration de ce processus est disponible sur le lien suivant :

http://lwh.free.fr/pages/algo/tri/tri_selection.html

Prenons pour exemple la liste d'entiers désordonnée suivante :

3	4	1	7	2	6
---	---	---	---	---	---



Q1. Compléter sur le schéma 1, la séquence à effectuer pour trier cette liste selon la méthode du tri par sélection. Pour cela pour chaque étape de tri :

→ **Ecrire** à leur place les éléments

→ **Repérer** en rouge les deux valeurs qui seront échangées

→ **Repérer** avec une flèche rouge le premier élément de la liste non triée

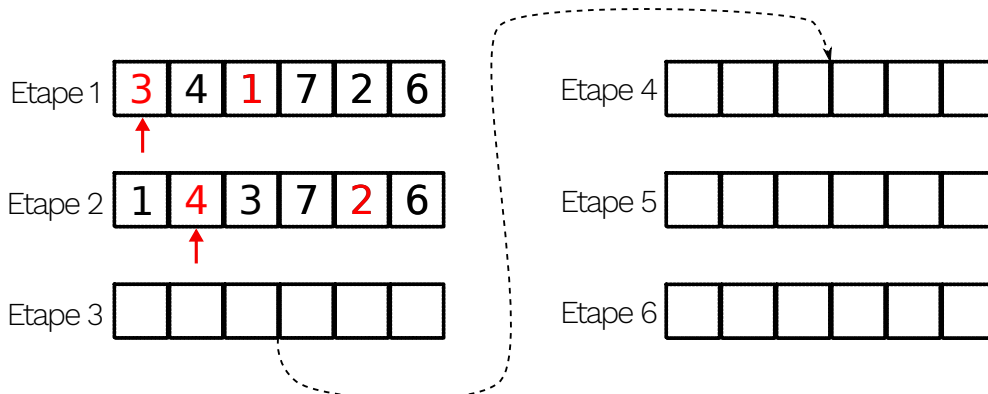


Schéma 1: Séquence de tri lors d'un tri par sélection

Programme de tri

Le programme Python de ce tri par sélection est donc le suivant :

```
def tri_par_selection(t):
    '''trie le tableau t dans l'ordre croissant'''
    for debut_nonTrie in range(len(t)) :
        indice_valMini = debut_nonTrie
        for i in range(debut_nonTrie + 1 , len(t)):
            if t[i] < t[indice_valMini] :
                indice_valMini = i
        echange(t, debut_nonTrie, indice_valMini)
```

Durant son fonctionnement cet algorithme échange de position deux valeurs positionnées à des indices différents de la liste.

Echange de deux valeurs

Pour échanger deux valeurs il faut utiliser une troisième pour mémoriser l'une des deux. Par exemple si l'on souhaite échanger les valeurs de x et y (voir figure 1) dans le cas où x=2 et y=3, il faut :

- 1) Affecter la valeur de x dans une variable temporaire temp
- 2) Affecter la valeur de y dans x
- 3) Affecter la valeur de temp dans y

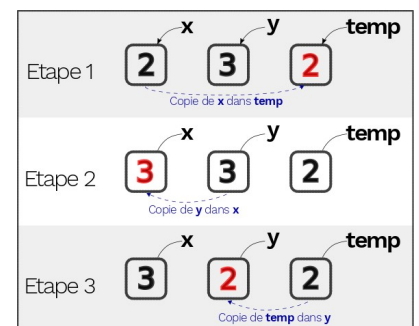


Figure 1: Echange de 2 valeurs

Q2. Ecrire un programme Python effectuant cette opération d'échange de valeurs entre deux variables x et y. **Valider** son fonctionnement.

Q3. Compléter sur le fichier tri.py, la fonction echange(table,indice1,indice2) afin qu'elle respecte la spécification associée. **Valider** son fonctionnement.



Q4. Dérouler à la main l'exécution attendue de la fonction `tri_selection(t)`, `t` étant la table à trier initialisée avec `t=[3, 4, 1, 7, 2, 6]`. **Compléter** la table d'exécution suivante afin de détailler les différentes étapes de la fonction `tri_selection` lors de l'appel `tri_par_selection(t)`

Table d'exécution

Liste t	debut_nonTrie	Valeur Mini dans t[debut_nonTrie :]	indice_valMin	Echange effectué
3 4 1 7 2 6	0	1	2	echange(t, 0, 2)

Dans la toute dernière étape du tri par sélection, il n’y a qu’une seule valeur dans la partie droite et il est donc inutile d’y chercher la plus petite valeur ni d’échanger avec elle-même.

- Q5. Modifier** le programme pour éviter cette étape inutile.
Q6. Valider sur un IDE Python l'appel de la fonction `tri_selection(t)` pour `t=[3, 4, 1, 7, 2, 6]`.
Décrire la démarche employée

3.Le tri par insertion

Principe de tri

Un autre tri, souvent utilisé par les joueurs de carte est le tri par insertion. Il suit le même principe que le tri par sélection en parcourant le tableau de gauche à droite et en maintenant une partie déjà triée sur la gauche.
 Mais plutôt que de chercher la valeur la plus petite dans la partie de droite, le tri par insertion va insérer la première valeur de la liste non triée (partie droite) dans la liste triée (partie gauche).
 Pour cela, on va décaler d’une case vers la droite tous les éléments déjà triés qui sont plus grands que la valeur à insérer, puis déposer cette dernière dans la case libérée. Une illustration de ce processus est disponible sur le lien suivant :

http://lwh.free.fr/pages/algo/tri/tri_insertion.html

Reprenons pour exemple la liste d’entiers désordonnée suivante :

3	4	1	7	2	6
---	---	---	---	---	---



Q7. Compléter sur le schéma 2, la séquence à effectuer pour trier cette liste selon la méthode du tri par sélection. Pour cela pour chaque étape de tri :

- **Ecrire** à leur place les éléments
- **Repérer** en rouge la valeur qui sera insérée dans la liste triée
- **Repérer** avec une flèche rouge l'emplacement de la liste triée où sera insérée cette valeur

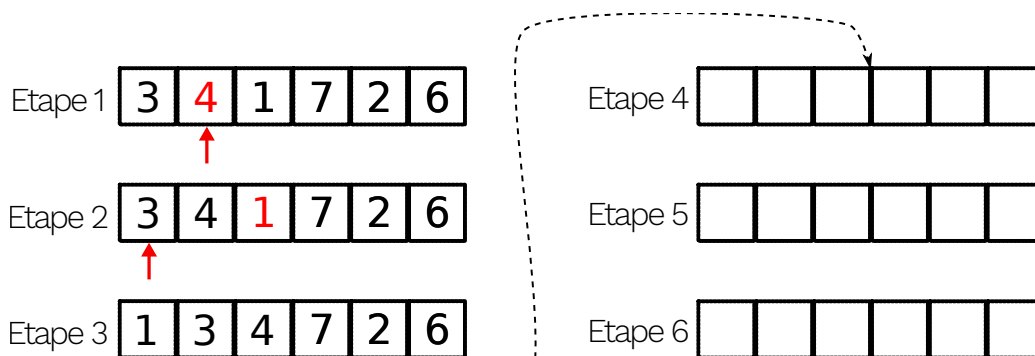


Schéma 2 : Séquence de tri lors d'un tri par insertion

Fonction d'insertion

Commençons par définir une fonction capable d'insérer un nouvel élément `val` dans une liste partiellement triée. Cette fonction s'écrit :

```
def insere(t, i, val) :
    ''' insère val dans la liste triée t[0...i] supposée triée'''
    indice_val = i
    while indice_val > 0 and t[indice_val-1] > val :
        t[indice_val] = t[indice_val-1]
        indice_val = indice_val - 1
    t[indice_val] = val
```

Q8. Dérouler à la main l'exécution attendue de la fonction `insere(t, 4, 2)`, `t` étant la table initialisée avec `t=[1, 4, 5, 7, 2]`. **Compléter** la table d'exécution suivante afin de détailler les valeurs des variables indiquées à chaque entrée de la boucle `while` lors de l'appel de la fonction `insere(t, 4, 2)`

Liste t	indice_val	t[indice_val - 1]	t[indice_val]
1 4 5 7 2	4	7	2
1 4 5 7 7	3		

← Etat de la liste t à la sortie de la fonction insere



Avec cette fonction, le plus dur est fait. Pour écrire le tri par insertion, il suffit d'insérer successivement toutes les valeurs du tableau avec la fonction `insere`, en procédant de la gauche vers la droite avec une boucle `for` :

```
def tri_insertion(t) :  
    '''trie le tableau t dans ordre croissant'''  
    for i in range(1, len(t)) :  
        insere(t, i, t[i])
```

Q9. Valider sur un IDE Python l'appel de la fonction `tri_insertion(t)` pour `t=[1, 4, 5, 7, 2]`.
Décrire la démarche employée

Correction de l'algorithme de tri

Un algorithme est correct s'il fait ce qu'on attend de lui. Démontrer la correction de l'algorithme consiste à démontrer que l'algorithme retourne les résultats escomptés dans toute circonstance.

Les invariants de boucle servent, entre autres, à prouver la validité d'un algorithme comportant une ou plusieurs boucles `FOR` ou `WHILE`.

Un invariant de boucle est une propriété qui est vraie pour tous les passages dans la boucle. **L'invariant doit être vrai avant d'entrer dans la boucle (Initialisation) et rester vrai jusqu'à la fin de celle-ci (Conservation et Terminaison).** Trouver et prouver que l'on a un invariant de boucle permet très généralement de prouver la correction (validité) d'un algorithme.

Dans notre cas de tri par sélection, l'invariant de boucle peut être :

x pour chaque valeur de `debut_nonTrie`, la liste `t[0 : debut_nonTrie - 1]` est triée. Cette affirmation est vraie avant d'entrer dans la boucle et reste vraie jusqu'à la fin de celle-ci. La présence de cet invariant prouve la correction du programme de tri

Q10. En déduire l'invariant de boucle de l'algorithme de tri par insertion

