Algorithmique de tris
Tris par sélection

Chap. 17,01

Nous avons vu, au chapitre 16, que *la recherche d'un élément dans un tableau est beaucoup plus efficace si ce tableau est ordonné*[1]. La question que se propose d'aborder ce chapitre est : « comment classer les éléments d'un tableau ? ». *De façon plus générale, le tri des listes permet de trouver rapidement les choses, et il facilite la recherche des valeurs extrêmes.*

Cette question est suffisamment importante pour que de nombreux chercheurs se soient penchés sur le problème et aient proposé plusieurs dizaines d'algorithmes différents. Certains sont *spécifiques au données numériques*, certains *copient les données dans une nouvelle structure* — ce qui réclame de l'espace en mémoire —, certains *effectuent un tri sur place* — c'est à dire dans le même espace mémoire, ...

Ce chapitre aborde l'étude de deux tris : le *tri par sélection* et le *tri par insertion*.

1 Du plus léger au plus lourd

Pour donner une idée de la difficulté du problème de tri, on peut utiliser le petit jeu à cette adresse : http://lwh.free.fr/pages/algo/tri/tri.htm. Il s'agit de trier quelques tonneaux (entre 3 et 10) par ordre de masse croissante. La masse de chaque tonneau est attribuée aléatoirement. On peut utiliser le « Glisser – Déposer » pour déplacer les tonneaux.
On ne dispose que d'une balance non étalonnée permettant de comparer les masses des tonneaux 2 à 2 et d'étagères pouvant vous servir de stockage intermédiaire.

Ce sont exactement les mêmes éléments que ceux dont dispose un ordinateur : une ***fonction de comparaison*** *et des* ***zones de stockage****.*

Le but ultime du jeu est évidemment de trier ces tonneaux en faisant le moins de comparaisons et d'échanges possibles.

1) Limiter l'animation à trois tonneaux et trouver le tonneau le plus léger. Décrire la méthode employée. Combien de comparaisons est-il nécessaire de faire ?

2) Toujours à partir de trois tonneaux, les trier du plus léger au plus lourd. Décrire la méthode employée (on peut schématiser toutes les étapes sur sa feuille). Combien de comparaisons est-il nécessaire de faire ?

3) Choisir maintenant cinq tonneaux. Les trier du plus léger au plus lourd. Décrire la méthode employée et indiquer combien de comparaisons ont été effectuées.

4) (Facultatif) Toujours à partir de cinq tonneaux, essayer de trouver une autre méthode permettant de les trier. La décrire.

1. À vrai dire, ce n'est pas en cours d'informatique que vous avez découvert ceci : dans toutes les bibliothèques les livres sont classés de façon à rendre leur recherche plus rapide !

2 Algorithme de tri

Un **algorithme de tri** est, en informatique ou en mathématiques, *un algorithme qui permet d'organiser une collection d'objets selon un ordre déterminé.*

Les objets à trier font donc partie d'un *ensemble muni d'une relation d'ordre*. Les ordres les plus utilisés sont l'*ordre numérique* et l'*ordre lexicographique* (dictionnaire).

3 Tri par sélection

3.1 Introduction

La méthode du tri par insertion est expliquée à cette adresse : https://youtu.be/AgFR0kO05RM (ou, de façon plus folklorique.... à cette adresse : `https://www.youtube.com/watch?t=12&v=Ns4TPTC8whw`).

5) Visualiser la première vidéo (s'arrêter au bout de 4min40). Essayer de bien comprendre la méthode.

6) Appliquer la méthode dans le jeu des tonneaux et trier cinq tonneaux du plus léger au plus lourd. Cette méthode était-elle celle que vous aviez mise en œuvre dans la section 1 ?

7) La méthode étant bien comprise, choisir sept cartes à jouer (avec les valeurs numériques, pas des figures). Les placer en ligne au hasard sur une table et les trier en appliquant le tri par sélection. **Se filmer pendant toute l'opération en commentant chacune des étapes !**

3.2 Algorithme

Idée. À chaque étape du processus, le tableau est divisé en deux sous-tableaux : le premier (indices plus petits que l'indice courant) est trié, le second (indices plus grands que l'indice courant n'est pas encore trié). On cherche, à partir de la position courante, la plus petite valeur dans le sous-tableau non trié (à droite) et une fois cette dernière trouvée, on permute (si nécessaire) la valeur courante et la valeur la plus petite.

Remarque. La permutation des valeurs contenues dans deux variables est souvent employée en informatique, elle nécessite une troisième variable pour un stockage temporaire, comme nous l'avons vu au début de l'année).

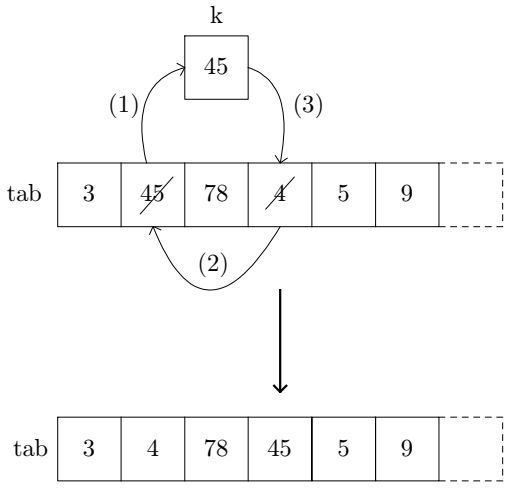

Figure 1. *Inversion des valeurs de deux cases d'un tableau.*

Le *tri par sélection* est basé sur l'utilisation de deux boucles `POUR` imbriquées :

- La première boucle parcourt la liste des valeurs, de la première à la l'avant dernière ;
- La seconde boucle recherche la plus petite valeur, de la position courante (compteur de la première boucle) à la fin du tableau, puis l'échange avec la position courante.

La partie gauche de la liste est donc triée au fur et à mesure de l'avancement de la première boucle.

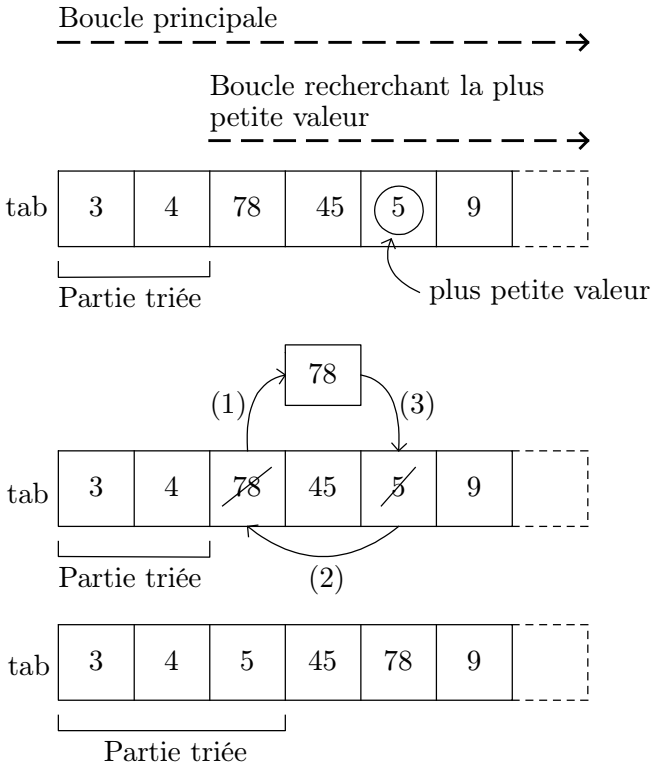

Figure 2. *Illustration du tri par sélection.*

Remarque. Dans l'algorithme ci-dessous, le comportement « Python » a été utilisé : `nb` *étant le nombre de valeurs dans le tableau, les indices de ce derniers varient de 0 à* $nb-1$

Algorithme 1

```
Fonction tri_sélection(tab, nb)
Déclarations
        Paramètre tab           tableau[20] d'Entiers
        Paramètre nb            Entier, longueur du tableau
        Variables i, j, k, min  Entiers
Début
        Pour i variant de 0 à nb-2 Faire
                min = i
                // localisation du minimum
                Pour j variant de i+1 à nb-1 Faire
```

```
                    Si (tab[j] < tab[min]) Alors
                        min = j
                    FinSi
                FinPour
                // inversion
                k = tab[i]          // étape 1
                tab[i] = tab[min]   // étape 2
                tab[min] = k        // étape 3
        FinPour
Fin
```

Remarque. L'algorithme du tri par sélection est un *algorithme de tri en place*. La réorganisation du tableau ne nécessite *pas la création d'un nouveau tableau*, ce qui économise de la place en mémoire.

8) Faire tourner « à la main » l'algorithme lorsque la fonction reçoit le tableau `tab = [5,2,4,6,1,3]` et la variable `nb = 6`.

	tab	nb	i	min	j	k	Tests
	[5, 2, 4, 6, 1, 3]	6	0	0	1		tab[1] = 2 < tab[0] = 5
		6	0	1	2		tab[2] = 4 > tab[1] = 2
		6	0	1	3		tab[3] = 6 > tab[1] = 2
		6	0	1	4		tab[4] = 1 < tab[1] = 2
		6	0	4	5		tab[5] = 3 > tab[4] = 1
		6	0	4		1	
	[1, 2, 4, 6, 1, 3]	6	0	4		1	
	[1, 2, 4, 6, 5, 3]	6	0	4		1	
		6	1	1	2	1	tab[2] = 4 > tab[1] = 2
		6	1	1	3	1	tab[3] = 6 > tab[1] = 2
		6	1	1	4	1	tab[4] = 5 > tab[1] = 2
		6	1	1	5	1	tab[5] = 3 > tab[1] = 2
		6	1	1		2	
	[1, 2, 4, 6, 5, 3]	6	1	1		2	
►	[1, 2, 4, 6, 5, 3]	6	1	1		2	
		6	2	2	3	2	tab[3] = 6 > tab[2] = 4
		6	2	2	4	2	tab[4] = 5 > tab[2] = 4
		6	2	2	5	2	tab[5] = 3 < tab[2] = 4
		6	2	5		3	
	[1, 2, 3, 6, 5, 3]	6	2	5		3	
	[1, 2, 3, 6, 5, 4]	6	2	5		3	
		6	3	3	4	3	tab[4] = 5 < tab[3] = 6
		6	3	4	5	3	tab[5] = 4 < tab[4] = 5
		6	3	5		6	
	[1, 2, 3, 4, 5, 4]	6	3	5		6	
	[1, 2, 3, 4, 5, 6]	6	3	5		6	
		6	4	4	5	6	tab[5] = 6 < tab[4] = 5
		6	4	4		5	
	[1, 2, 3, 4, 5, 6]	6	4	4		5	
	[1, 2, 3, 4, 5, 6]	6	4	4		5	

Vérification de la justesse du tableau à cette adresse : pythontutor.

9) Implémenter la fonction en langage Python et la tester en l'appelant avec les arguments donnés à la question précédente.

▶
```
def tri_selection(tab):
    """
    Implémentation du tri par sélection.
    La tableau est trié en place.
    En théorie, il n'est pas nécessaire de retourner le tableau.
    """
    nb = len(tab) - 1  # dernier indice

    for i in range(0, nb):  # du premier à l'avant-dernier indice
        min = i
        for j in range(i + 1, nb+1):  # de l'indice i+1 au dernier indice
            if tab[j] < tab[min]:
                min = j
        # Permutation du contenu des variables
        k = tab[i]
        tab[i] = tab[min]
        tab[min] = k
    return tab

def main():
    tab = [5, 2, 4, 6, 1, 3]
    print(tab)
    tri_selection(tab)
    print(tab)

main()
```

On faire fonctionner le programme à cette adresse : pythontutor.

10) Comment prouve-t-on, de façon générale, la terminaison d'un algorithme ?

▶ On définit un variant de boucle.

11) Est-il nécessaire de prouver la terminaison de cet algorithme ?

▶ L'algorithme fait intervenir deux boucles Pour. Il se termine donc forcément puisqu'il s'agit de boucles déterministes.

12) Comment prouve-t-on, de façon générale, la correction d'un algorithme ?

▶ On définit un invariant de boucle.

13) Définir un invariant de boucle et prouver que l'algorithme est correct.

▶ **Invariant de boucle.** La question à se poser est : « *Qu'est ce qui est constant à chaque étape de cet algorithme ?* »
Au début de chaque itération de la boucle `Pour` externe le tableau `tab[0, ..., i-1]` est un tableau trié constitué des i plus petits éléments du tableau `tab`, dans l'ordre croissant.

Rappels. Nous devons montrer trois choses, concernant un invariant de boucle :

Initialisation. Il est vrai avant la première itération de la boucle.

Conservation. S'il est vrai avant une itération de la boucle, il le reste avant l'itération suivante.

Terminaison. Une fois terminée la boucle, l'invariant fournit une propriété utile qui aide à montrer la validité de l'algorithme.

Si les deux premières propriétés sont vérifiées, alors l'invariant est vrai avant chaque itération de la boucle.
La troisième propriété est peut-être la plus importante : elle est utilisée pour prouver la validité de l'algorithme.

Démonstration de la correction de l'algorithme.

Initialisation. $i = 0$, tab[0] est alors un tableau à un élément, trié de façon évidente.

Conservation. Par hypothèse, pour i compris entre 1 et $\text{nb} - 2$, $\text{tab}[0, \ldots, i-1]$ est un tableau trié. Lors de l'itération i, on recherche le plus petit élément du tableau $\text{tab}[i, \ldots, \text{nb}-1]$ et on le place à la position d'indice i. À partir de l'hypothèse, cet élément est plus grand que n'importe quel élément du tableau $\text{tab}[0, \ldots, i-1]$ (sinon il appartiendrait déjà à ce tableau) ; le tableau $\text{tab}[0, \ldots, i]$ est donc trié.

Terminaison. Au début de la boucle dans laquelle $i = \text{nb} - 2$, $\text{tab}[0, \ldots, \text{nb}-3]$ est un tableau trié. À l'issue de cette boucle, on a déterminé le plus petit élément entre $\text{tab}[\text{nb}-2]$ et $\text{tab}[\text{nb}-1]$ et on a placé correctement ces éléments ; le tableau $\text{tab}[0, \ldots, \text{nb}-1]$ est donc trié et l'algorithme est correct.

14) Déterminer la complexité de l'algorithme.

▶ **Rappels.** Au chapitre précédent, la *notion de complexité* a été introduite. Nous allons rappeler, dans cette partie, quelques règles simples qui permettent de se faire une idée de l'efficacité d'un algorithme.

- Une *affectation* ou l'*évaluation d'une expression* ont un temps d'exécution petit **que l'on considère constant**. Cette durée constitue souvent l'unité de base dans laquelle on mesure le temps d'exécution d'un algorithme.

- Le temps pris pour exécuter une séquence d'instructions `p puis q` est la **somme des temps** pris pour exécuter les instructions `p` puis `q`.

- Le temps pris pour exécuter *un test* `Si (b) Alors p Sinon q FinSi` est inférieur ou égal au maximum des temps pris pour exécuter les instructions `p` et `q`, plus une unité qui correspond au temps d'évaluation de l'expression `b`.

- Le temps pris pour exécuter une boucle `Pour i variant de 1 à m par pas de 1 Faire p FinPour` est `m` fois le temps pris pour exécuter l'instruction `p` si ce temps ne dépend pas de la valeur de `i`.

  En particulier, quand deux boucles sont imbriquées, le corps de la boucle interne est répété à cause de cette boucle, mais aussi parce qu'elle-même est répétée dans son intégralité. Ainsi, si les deux boucles sont répétées respectivement `m` et `m'` fois, alors le corps de la boucle interne est exécuté `m * m'` fois en tout.

  Quand le temps d'exécution du corps de la boucle dépend de la valeur de l'indice `i`, le temps total d'exécution de la boucle est la somme des temps d'exécution du corps de la boucle pour chaque valeur de `i`.

- Le cas des boucles `Tantque` est plus complexe puisque le nombre d'itérations n'est en général pas connu à priori.

On étudie seulement la complexité du pire et on se limite à l'influence des boucles. Si on note N le nombre d'élements dans le tableau,

- pour $i = 0$, la boucle interne effectue $N - 1$ tours ;
- pour $i = 1$, la boucle interner effectue $N - 2$ tours ;
- pour $i = 3$, la boucle interne effectue $N - 3$ tours ;
- ....
- pour $i = N - 3$, la boucle interne effectue 2 tours ;
- pour $i = N - 2$, la boucle interne effectue 1 tour.

Finalement, il y a $1 + 2 + \cdots + (N-2) + (N-1) = \sum_{k=1}^{N-1} k = \frac{(N-1)\,N}{2} = \frac{N^2}{2} - \frac{N}{2}$. La complexité de l'algorithme est en $O(N^2)$.