Chapitre 2 : Tris par insertion et par sélection

Dans ce chapitre on considère un tableau T d'entiers que l'on veut trier par ordre croissant. Les algorithmes permettant d'effectuer cette tâche s'appellent des **algorithmes de tri** et ont une place fondamentale en algorithmique et en informatique car il est souvent nécessaire des trier des éléments (des idées ?). On va étudier deux algorithmes de tri dans ce chapitre : le tri par insertion et le tri par sélection.

Spécification d'un algorithme de tri

- *Entrée/Sortie* : tableau `T` (de taille `n`, constitué d'entiers, les indices variant de `0` à `n-1`)
- *Rôle* : trier `T` par ordre croissant
- *Précondition* : `T` non vide
- *Postcondition* : `T` est trié c'est-à-dire que chaque élément de T est inférieur ou égal à l'élément suivant : pour tout entier `i` compris entre `0` et `n-2`, `T[i]≤T[i+1]`

Remarque : on ne doit pas parler dans la spécification de *comment* trier !

Tri par insertion

Présentation de l'algorithme

Voici une vidéo illustrant l'algorithme du tri par insertion : https://youtu.be/bRPHvWgc6YM. Et voici l'algorithme :

Tri par insertion

```
pour i de 1 à n-1 faire
    x ← T[i]
    j ← i
    tant que j > 0 et x < T[j-1] faire
        T[j] ← T[j-1]
        j ← j - 1
    fin tant que
    T[j] ← x
fin pour
```

On applique cet algorithme au tableau `T = [8, 3, 11, 7, 2]`. Compléter le tableau suivant pour suivre l'état des variables et du tableau au cours de l'algorithme.

i n°ité	**x**	**j**	**j > 0 et x < T[j-1]** V ou F ? (on entre ou pas dans la boucle tant que)	**T[j] ← T[j-1]** **j ← j - 1** (intérieur boucle tant que)	**T[j]**	Etat du tableau. **T**	Nb de compa de 2 éléments de **T** pour chaque itération du `pour i`	
1	x=3 (T[1])	1	1>0 et 3<8 VRAI → oui (1>0 et 3<T[0])	T[1]=8 j = 1-1=0				
			0>0 et 3<T[-1] FAUX → non		T[0]=3	[3, 8, 11, 7, 2]	1 (T[1] comparé à T[0])	
2	x=11 (T[2])	2	2>0 et 11<8 FAUX → non (2>0 et 11<T[1])		T[2]=11	[3, 8, 11, 7, 2]	1 (T[2] comparé à T[1])	

Germain BECKER & Sébastien POINT, Lycée Mounier, ANGERS

Moralité

On peut traduire l'algorithme de tri par insertion de la façon suivante :

- Prendre le deuxième élément du tableau et l'insérer à sa place parmi les éléments qui le précèdent
- Prendre le troisième élément du tableau et l'insérer à sa place parmi les éléments qui le précèdent
- Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exercice : On considère désormais le tableau T = [7, 5, 4, 3, 1]. Reproduire le schéma de la dernière colonne pour trier par insertion ce tableau T. Compter à chaque étape le nombre de comparaisons nécessaires.

B. Efficacité de l'algorithme

On rappelle que l'efficacité d'un algorithme correspond à son coût en terme d'opérations élémentaires *dans le pire cas*. Pour simplifier cette étude, nous n'allons étudier ici que le nombre de comparaisons du tri par insertion.

Qu'est-ce qu'un pire cas ? Un tableau trié par ordre décroissant (comme dans l'exercice ci-dessus) puisqu'il faut aller insérer chaque élément en première position : cela implique de le comparer à tous les éléments qui le précèdent.

Pour un tableau de taille n trié par ordre décroissante (pire cas), on a :

- Au premier passage dans la boucle : il y a 1 comparaison à faire (T[1] est comparé à T[0]) ;
- Au deuxième passage : il y a 2 comparaisons à faire (T[2] est comparé successivement à T[1] et T[0]) ;
- Ainsi de suite ;
- Au dernier passage (i = n – 1) : il y a n – 1 comparaisons à faire (T[n-1] est comparé successivement à T[n-2], T[n-3], …, T[0]).

Il y a donc en tout $1 + 2 + 3 + \cdots + (n-2) + (n-1)$ comparaisons dans le pire cas.

Propriété

Pour tout entier naturel n, on a :

$$1 + 2 + 3 + \cdots + (n-2) + (n-1) = \frac{n(n-1)}{2}$$

Germain BECKER & Sébastien POINT, Lycée Mounier, ANGERS

En effet : Notons $S = 1 + 2 + \cdots + (n-2) + (n-1)$. Calculons $2S$:

$$\begin{array}{c|c} & 1 + 2 + \ldots + (n-2) + (n-1) \\ + & (n-1) + (n-2) + \ldots + 2 + 1 \\ \hline = & \underbrace{n + n + \ldots + n + n}_{n-1 \text{ fois}} \end{array}$$

On obtient alors : $2S = (n-1) \times n$ et donc $S = \frac{n(n-1)}{2}$

Moralité

Il y a donc en tout : $1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2}$ comparaisons à faire dans le pire cas. Comme $\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$, on peut alors dire que le coût de l'algorithme est de l'ordre de n^2 (on prend la plus grande puissance de n).
On dit que l'algorithme de tri par insertion a un **coût quadratique**.
Cela signifie que si l'on double la taille du tableau T, alors le temps de calcul (théorique) n'est pas doublé mais est multiplié par 4. Si on triple la taille de T, le temps de calcul (théorique) est multiplié par $3^2 = 9$, etc.

Tri par sélection

Présentation de l'algorithme

Voici une vidéo illustrant l'algorithme du tri par sélection : https://youtu.be/8u3Yq-5DTN8. Et voici l'algorithme :

Tri par sélection

```
pour i de 0 à n-2 faire
    ind_min ← i
    pour j de i+1 à n-1 faire
        si T[j] < T[ind_min] alors
            ind_min ← j
        fin si
    fin pour
    echange(T, i, ind_min)   # échange T[i] et T[ind_min]
fin pour
```

On applique cet algorithme au tableau `T = [8, 3, 11, 7, 2]`. Compléter le tableau suivant pour suivre l'état des variables et du tableau au cours de l'algorithme.

i n°ité	**ind_min**	**j**	**T[j] < T[ind_min]** V ou F ?	**ind_min ← j** (si condition vraie)	Etat du tableau **T** après **echange(T,i,ind_min)**	Nb de compa de 2 élts de **T** pour chaque itération du pour i	
0	0	1	3<8 VRAI (T[1]<T[0])	ind_min = 1			
		2	11<3 FAUX (T[2]<T[1])	X			
		3	4<3 FAUX (T[3]<T[1])	X			
		4	2<3 VRAI (T[4]<T[1])	ind_min = 4	[2,3,11,7,8] échange T[0] et T[4]	4	

Moralité

On peut traduire l'algorithme de tri par sélection de la façon suivante :

- Rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
- Rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exercice : On considère désormais le tableau T = [7, 5, 4, 3, 1]. Reproduire le schéma de la dernière colonne pour trier par sélection ce tableau T. Compter à chaque étape le nombre de comparaisons nécessaires.

B. Efficacité de l'algorithme

Pour simplifier et pour pouvoir comparer avec l'autre algorithme de tri, nous n'allons étudier ici que le nombre de comparaisons du tri par sélection.

On constate sur les deux exemples que, quel que soit le tableau de taille 5 à trier par sélection, il y a $4+3+2+1=15$ comparaisons. Ainsi, pour une taille de tableau donnée, il y a toujours le même nombre de comparaisons, il n'y a donc pas de pire ou meilleur cas (tous les tableaux sont des pires cas). Cela vient du fait que l'on connaît toujours à l'avance le nombre de tours de boucle puisqu'il s'agit de deux boucles pour. Plus précisément, pour un tableau de taille n :

- Au 1er passage dans la boucle (pour i = 0), il y a n-1 comparaisons dans la recherche du plus petit élément ;
- Au 2ème passage (pour i=1), il y en a n-2 comparaisons ;
- Ainsi de suite ;
- Au dernier passage (pour i = n-2), il n'y a plus que 1 comparaison (les deux dernières cases).

Il y a donc en tout (comme pour le tri par insertion) $(n-1)+(n-2)+\cdots+2+1=\frac{n(n-1)}{2}$ comparaisons à faire dans le pire cas (dans tous les cas !). On retrouve donc aussi un coût de l'ordre de n^2 (puisque $\frac{n(n-1)}{2}=\frac{n^2}{2}-\frac{n}{2}$).

Moralité

Le tri par sélection a également un **coût quadratique** dans le pire cas, c'est-à-dire de l'ordre de n² (où n = taille du tableau). Donc si on multiplie par k la taille du tableau, alors le temps de calcul pour faire le tri est multiplié par k². Contrairement au tri par insertion, il y a dans tous les cas $\frac{n(n-1)}{2}$ comparaisons à faire, quel que soit le tableau de départ.

Germain BECKER & Sébastien POINT, Lycée Mounier, ANGERS

Implémentation des algorithmes en Python

Programmez deux fonctions Python `tri_insertion(T)` et `tri_selection(T)` qui implémentent ces deux algorithmes puis recopiez ci-dessous leurs codes pour pouvoir les apprendre.

Les tris fournis par Python

Il existe d'autres algorithmes de tris et certains sont (beaucoup) plus efficaces (leur coût est de l'ordre de n log(n) qui est inférieur à n^2).
Python fournit notamment des fonctions permettant de trier de manière plus efficace un tableau. Elles se présentent de deux façons différentes, selon que l'on veuille obtenir une copie triée du tableau, sans le modifier (`sorted`), ou au contraire modifier le tableau pour le trier (`sort`).

La fonction `sorted` prend en argument un tableau et renvoie un *nouveau tableau*, trié, contenant les mêmes éléments.

```
>>> t = [12, 5, 3, 6, 8, 10]
>>> sorted(t)
[3, 5, 6, 8, 10, 12]
>>> t
[12, 5, 3, 6, 8, 10]  # le tableau t de départ n'a pas été modifié !
```

La construction `t.sort()`, en revanche, modifie le tableau `t` pour le trier sans rien renvoyer.

```
>>> t.sort()
>>> t
[3, 5, 6, 8, 10, 12]
```

Ces deux fonctions sont largement plus efficaces que les tris par sélection et par insertion. Trier un tableau d'un million d'éléments, par exemple, est instantané (ce qui n'est pas le cas pour les deux tris étudiés dans ce chapitre).

Bilan

- Le **tri par sélection** et le **tri par insertion** sont deux algorithmes de tri élémentaires, qui peuvent être utilisés pour trier des tableaux.
- Ces deux algorithmes ont un **coût quadratique** dans le pire cas (et même en moyenne) : cela signifie que si on multiplie la taille du tableau par k alors le temps de calcul (dans le pire cas) est multiplié par k^2.
- Les deux restent peu efficaces dès que les tableaux contiennent plusieurs milliers d'éléments. Il existe de meilleurs algorithmes de tri, plus complexes, dont celui offert par Python avec les fonctions `sort` et `sorted`.
- Nous reviendrons sur cela lorsque nous aborderons les tris de tables de données.

Références

- Équipe éducative du DIU EIL, Université de Nantes.
- Livre *Numérique et Sciences Informatiques*, niveau Première, T. BALABONSKI, S. CONCHON, J.-C. FILLIATRE, K. NGUYEN, éditions Ellipses.