Complexité du tri par Sélection

Algorithme du tri par sélection :

```
void triSelection(int tab[], int const taille)
{       int i, min, j;
        for (i = 0; i < n- 1; ++i)
        {
        min = i;
        for (j = i + 1; j < n; ++j)
        {
                if (tab[j] < tab[min])
                {
                        min = j;
                }
        }
                if (min != i)
                {
                        échanger(tab[i], tab[min]);
                }
        }
}
```

Complexité du tri par sélection :

Le principe du tri **par sélection** est de **rechercher le plus petit élément** du tableau et de le placer au début de ce tableau. On commence en considérant le tableau initial non trié. Au fur et à mesure nous déplaçons au début du tableau le plus petit élément de ce tableau. De cette manière, la partie du tableau non triée diminue d'une case à chaque itération.
Pour le calcul du tri par insertion nous nous intéressons au nombres de comparaisons qui permettent de calculer le minimum du tableau à chaque itération :

Le calcul du minimum se répète n-1 fois **(boucle : for (i = 0; i < n - 1; ++i))**

Pour la première itération i=0 : pour calculer le min nous réalisons n-1 comparaisons
Pour la deuxième itération i=1 : pour calculer le min nous réalisons n-2 comparaisons
.
.
Pour la n-2 itération i=n-3: pour calculer le min nous réalisons 2 comparaisons
Pour la n-1 itération i=n-2 : pour calculer le min nous réalisons 1 comparaisons

Soit C le nombre de comparaisons total. **C** est la somme des compassions pour toutes les itérations (n-1 termes, car le calcul du min se répète n-1 fois).

C= n-1 + n-2 + ……+ 2 + 1

Soit C' = C = 1 + 2 + ……n-1 + n-2

C+C' = n + n + ……..+n + n (n-1 fois)

C+C' = n (n-1)

2C = n^2-n

C = (n^2-n)/2 = n^2/2 – n/2

Donc la complexité du tri par sélection par approximation est de l'ordre de **O(n^2)**

Complexité du tri par insertion

Le principe du tri par insertion est de diviser le tableau à trier en deux sous-ensembles : un trié et l'autre nom. A chaque itération un élément de l'ensemble non trié est déplacé vers l'ensemble trié tout en veillant à ce que celui-ci reste trié. A chaque itération les cases de la partie du tableau non triée diminuent et celles de la partie triée augmentent.

Concrètement l'algorithme commence par une partie du tableau triée contenant une seule case : la première case du tableau tab[0]. Nous somme alors certain que cette partie du tableau composée d'une seule case est triée. L'autre partie du tableau est quant à elle non triée (de tab[1] à tab[n-1]).

Voici la version simple du tri par insertion :

```
void tri-insertion(int tab[], int n)
{
	for (i = 1; i < n; i++)
	{
		elementInsere = tab[i];

		j = i;

		while ((j > 0) && (tab[j - 1] > elementInsere))
		{
			tab[j] = tab[j - 1];
			j--;
		}
		tab[j] = elementInsere;
	}
}
```

L'algorithme se répète **n-1** fois (la boucle : for (i = 1; i < n; i++)). A chaque itération de cette boucle **for** un élément de la partie non triée est déplacé vers la partie triée en utilisant des décalages successives.

La boucle **while ((j > 0) && (tab[j - 1] > elementInsere))** a pour rôle de comparer l'élément du tableau qui est juste à droite de la zone déjà triée (on l'appel **élément à insérer**) avec les éléments de la partie triée du tableau, et cela afin de déplacer cet élément vers une position qui garde toujours la première partie du tableau triée.

Tant que les cases qui sont à gauche de l'élément à insérer lui sont supérieures on les décale vers la droite afin de lui faire de la place avant ces éléments, et cela dans le but de garder la partie gauche du tableau toujours triée.

On quitte la boucle **while ((j > 0) && (tab[j - 1] > elementInsere))** soit quand j==0 (c.a.d tous les élément de la partie triée sont supérieures à l'élément à insérer) ou si tab[j-1]<= **elementInsere** (c.a.d on arrive à une case du tableau triée qui est inférieur **à l'élément à insérer**). L'insertion dans cette position est réalisée grâce aux décalages (tab[j] = tab[j - 1]) réalisés à chaque itération de cette boucle while(…).

Calcul de la complexité :

Dans ce programme à chaque itération de la première boucle (**boucle for**) on réalise une séries de décalages (garce à la boucle interne while) qui permettent de placer un élément dans la partie du tableau non triée. Pour calculer la complexité de l'algorithme il suffit alors de compter le nombre de ces décalages.

A chaque itération le décalage est réalisé par l'instruction :

```
tab[j] = tab[j - 1];
```

on peut aussi compter l'instruction j--

donc nous avons deux opérations par itération (par décalage) :

La boucle for se répète **n-1** fois : `for (i = 1; i < n; i++)`

A chaque itération de cette boucle for on réalise une suite de décalages pour déplacer une élément de droite à gauche depuis la partie non triée vers celle triée

À la première itération de la boucle (for i) : i=1, au plus on peut faire **un** décalage
À la deuxième itération de la boucle (for i) : i=2 au plus on peut faire **deux** décalages
À la troisième itération de la boucle (for i) : i=3 au plus on peut faire **trois** décalages
.
.

À la n-1 ème itération de la boucle (for i) : i=n-1 au plus on peut faire **n-1** décalages

Soit D le nombre de décalages : **D** est la somme des décalages pour toutes les itérations (n-1 termes, car la boucle (for i..) se répète n-1 fois).

Donc au plus :

D = 1 + 2 + 3 + ……+ n-1 + n-2 (n-1 termes)

Soit D' = D

D' = n-1 + n-2 + n-3 +…… + 3 + 2 + 1

D = 1 + 2 + 3 + …… + n-3 + n-2 + n-1

D+ D' = n + n + n ……..+ n + n + n (n-1 fois)

2D = n (n-1)

D = n(n-1)/2

Nous avons deux opérations par décalages

Nombre d'opérations = 2 D = n(n-1) =n^2-n

Donc la complexité du tri par insertion est par approximation **O(n^2)**

Exemple du tri par insertion, tableau tab :

11	5	7	3	2	1	14	17	20	18

Initialement on considère que la partie verte est la partie triée du tableau : i=1

11	5	7	3	2	1	14	17	20	18

élément à insérer

Première itération de la boucle for (i = 1; i < n; i++)

J =i = 1

donc j=1 // première itération

On prend la case qui suit la partie du tableau déjà triée : j=1, c'est l'élément à insérer. Et on compare cette case avec les élément triées (qui sont à sa gauche). Tant que les cases qui sont à gauche de l'élément à insérer lui sont supérieures on les décale vers la droite (**tab[j] = tab[j - 1];**) afin de faire de la place pour **l'élément à insérer** avant ces cases pour garder la partie gauche du tableau toujours triée.

Ici **elementInsere** = tab[1] = 5;

Si tab[0]>tab[1] (**tab[j - 1] > elementInsere**) on décale tab[0] vers la droite (c'est le cas car 11>5).

tab[1]=tab[0] ;

tab[1] = 11 ;

11	11	7	3	2	1	14	17	20	18

j est décrémenté donc j=0 est on sort de la boucle

Ensuite **tab[j] = elementInsere;**

c.a.d tab[0]=5 ;

5	11	7	3	2	1	14	17	20	18

On se retrouve avec une partie gauche du tableau toujours triée,

Deuxième Itération de la boucle for (i = 1; i < n; i++)

j=i = 2

On continue l'itération 2, j= 2.

5	11	7	3	2	1	14	17	20	18

élément à insérer

elementInsere = tab[2] = 7

On compare **elementInsere** avec les case de la partie gauche du tableau déjà triée

Tanque **elementInsere** est supérieur aux cases du tableau à sa gauche on décale ces éléments ;

while **(j > 0 && tab[j - 1] > elementInsere;)**

{

tab[j] = tab[j - 1];

j-- ;

}

tab[1]=11 **> elementInsere** =7

on décale la case j=1 (tab[j] = tab[j - 1])

5	11	11	3	2	1	14	17	20	18

j--

tab[0]=5 **pas > elementInsere** =7

on sort de la boucle avec j=1

tab[j] = **elementInsere** =7

tab[1]=7

5	7	11	3	2	1	14	17	20	18

Troisième Itération de la boucle for (i = 1; i < n; i++)

j=i = 3

5	7	11	3	2	1	14	17	20	18

élément à insérer

On compare 3 avec les cases vertes, à chaque fois qu'une case verte est > 3 on la décale à droite

Résultat après décalage :

5	5	7	11	2	1	14	17	20	18	5

On sort de la boucle while avec j=0 (tous les éléments en vert sont> 3)

tab[0] = **elementInsere = 3**

3	5	7	11	2	1	14	17	20	18	5

Et on continue ainsi ça jusqu'à trier tout le tableau (fin de la boucle for)