Chapitre 4

Algorithmes de tri

On désigne par "tri" l'opération consistant à ordonner un ensemble d'éléments en fonction de clés sur lesquelles est définie une relation d'ordre.

Les algorithmes de tri ont une grande importance pratique. Ils sont fondamentaux dans pratiquement tous les domaines et c'est rare qu'on voit une application informatique sans besoin de trier des éléments.

On trouve dans la littérature plusieurs algorithmes de tri qui se diffèrent dans la complexité :

4.1 Méthodes simples

4.1.1 Tri par bulle

On balaye la liste en échangeant deux éléments consécutifs s'ils sont dans le mauvais ordre. Le plus grand élément de la liste est donc repoussé à la fin.

On recommence à partir du début, avec les n - 1 premiers éléments et ainsi de suite.

```
Procédure TriParBulles( T : tableau[1..n] de entier);
Var  i,j,temp : entier ;
Début
    Pour i de 1 à n − 1 faire
        Pour j de 1 à n − i faire
            Si (T[j] > T[j + 1]) Alors
                temp ← T[j] ;
                T[j] ← T[j + 1] ;
                T[j + 1] ← temp ;
            Fin Si;
        Fin Pour;
    Fin Pour;
Fin;
```

L'exemple suivant montre une trace de cet algorithme pour trier un tableau de quatre éléments :

i	*j*	*T = [8,4,11,1]*			*(n=4)*
1	1	**4** ↔	**8**	**11**	**1**
	2	**4**	**8** ↔	**11**	**1**
	3	**4**	**8**	**1** ↔	**11**
2	1	**4** ↔	**8**	**1**	**11**
	2	**4**	**1** ↔	**8**	**11**
3	1	**1** ↔	**4**	**8**	**11**

Le nombre d'itérations faite par la procédure en fonction deux n est :

$\sum^{n-1}\sum^{n-i} = (n-1) + (n-2) + ... + (n-(n-1)) = n(n-1) - n(n-1)/2$

$= n^2 - n - \frac{n^2}{2} + \frac{n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$

Donc la complexité de la procédure est de $O(n^2)$.

4.1.2 Tri par insertion

On suppose que les i-1 premiers éléments de la liste sont déjà classés. On insère à sa place le i^{eme} élément parmi les i-1 premiers ; de la sorte, les i premiers éléments sont triés. On itère jusqu'à insérer le n^{eme}.

```
Procédure TriInsertion( T : tableau[1..n] de entier);
Var  i,j,temp : entier ;
Début
    Pour i de 2 à n faire
        k ← i-1 ;
        temp ← T[i] ;
        Tant que (k ≥ 1 et temp < T[k] ) faire
            T[k + 1] ← T[k] ;
            k ← k − 1 ;
        Fin TQ;
        T[k + 1] ← temp ;
    Fin Pour;
Fin;
```

i	*k*	*temp*	*T = [8,4,11,1]*			*(n=4)*
2	1	4	**8**	**4**	**11**	**1**
			8	**8**	**11**	**1**
			4	**8**	**11**	**1**
3	2	11	**4**	**8**	**11**	**1**
4	3	1	**4**	**8**	**11**	**1**
	2		**4**	**8**	**11**	**11**
	1		**4**	**8**	**8**	**11**
			4	**4**	**8**	**11**
			1	**4**	**8**	**11**

La complexité de l'algorithme au pire des cas est $O(n^2)$

4.1.3 Tri par sélection

Le tri par sélection consiste simplement à sélectionner l'élément le plus petit du tableau à trier, à le mettre au début, et à répéter itérativement le processus tant qu ?il reste des éléments dans le tableau.

```
Procédure TriParSelection( T : tableau[1..n] de entier);
Var  i,j,IndMin,temp : entier ;
Début
    Pour i de 1 à n − 1 faire
        IndMin ← i ;
        Pour j de i+1 à n faire
            Si (T[j] < T[IndMin]) Alors
                IndMin ← j ;
            Fin Si;
        Fin Pour;
        temp ← T[i] ;
        T[i] ← T[IndMin] ;
        T[IndMin] ← temp ;
    Fin Pour;
Fin;
```

La figure suivante illustre une trace de cet algorithme :

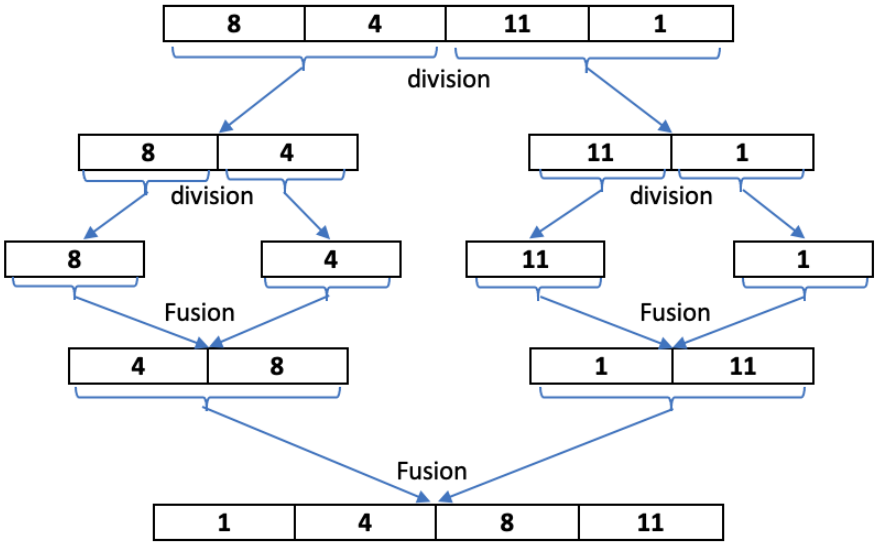
i	*IndMin*	*j*	*T = [8,4,11,1]*		*(n=4)*	
1	1		**8**	**4**	**11**	**1**
		2	**8**	**4**	**11**	**1**
		3	**8**	**4**	**11**	**1**
	4	4	**8**	**4**	**11**	**1**
			1	**4**	**11**	**8**
2	2		**1**	**4**	**11**	**8**
		3	**1**	**4**	**11**	**8**
	4	4	**1**	**4**	**11**	**8**
			1	**4**	**11**	**8**
3	3		**1**	**4**	**11**	**8**
	4	4	**1**	**4**	**11**	**8**
			1	**4**	**8**	**11**

L'algorithme fait n-1 itérations, à chaque itération i il fait n-i-1 itération d'où sa complexité est $O(n^2)$

4.2 Tri par fusion

Le tri par fusion (merge sort en anglais) est une illustration du principe "diviser pour résoudre" : le tableau à trier est tout d ?abord scindé (divisé) en deux suites de longueurs égales. Ces deux suite sont ensuite triées séparément avant d'être fusionnées (ou interclassées).

Exemple :



Dans cet exemple, le tableau de quatre éléments est scindé en deux tableaux de deux éléments chacun. Chaque tableau est trié de la même façon que le tableau initial. Après le tri des deux tableaux, ils sont fusionnés pour donner le tableaux entier trié. La division s'arrête lorsqu'on obtient un tableau d'une seule case qui est naturellement trié. L'algorithme détaillé est le suivant :

```
Algorithme TriParFusion;
Var T : tableau[1..n] de entier ;
Procédure Fusion( Debut,Fin : entier);
Début
    // Vu en TP
Fin;

Procédure TriParFusion( Debut,Fin : entier);
Début
    Si (Debut<Fin) Alors
        TriParFusion(Debut,(Debut+Fin) div 2) ;
        TriParFusion((Debut+Fin) div 2 + 1,Fin) ;
        Fusion(Debut,Fin) ;
    Fin Si;
Fin;

Début
  Pour i de 1 à n faire
      Lire(T[i]) ;
  Fin Pour;
  TriParFusion(1,n) ;
  Pour i de 1 à n faire
      Ecrire(T[i]) ;
  Fin Pour;
Fin.
```

La complexité de l'algorithme peut être calculée comme suit :

$$
\begin{array}{lll}
\mathrm{T(n)} & = 2\mathrm{T(n/2)} + \mathrm{n} & 1 \\
& = 2\ [2(\mathrm{T(n/4)} + \mathrm{n/2}] + \mathrm{n}) \quad = 4\ \mathrm{T(n/4)} + 2\mathrm{n} & 2 \\
& = 4\ [2\mathrm{T(n/8)} + \mathrm{n/4}\ ] + 2\mathrm{n} \quad = 8\mathrm{T(n/8)} + 3\mathrm{n} & 3 \\
& = 8[2\mathrm{T(n/16)} + \mathrm{n/8}] + 3\mathrm{n} \quad 16\mathrm{T(n/16)} + 4\mathrm{n} & 4 \\
& \quad \vdots & \\
& = 2^i\mathrm{T(n/2^i)} + i\mathrm{n} & i \\
& \quad \vdots & \\
& = \mathrm{n}\ \mathrm{T(1)} + log_2(\mathrm{n})\mathrm{n} & log_2(\mathrm{n}) \\
& = \mathrm{n}(log_2(\mathrm{n}) + 1) &
\end{array}
$$

Donc la complexité de l'algorithme est O(nlog_2(n))

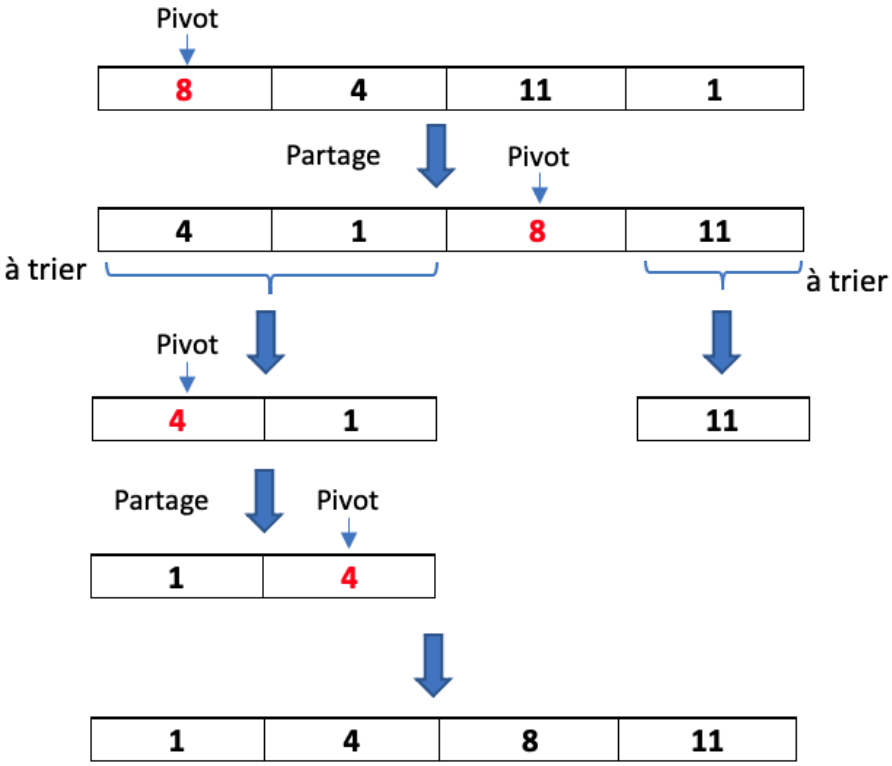
4.3 Tri rapide

Le tri rapide (quicksort) consiste à partitionner le tableau T à trier en deux sous-tableaux T1 et T2 contenant respectivement les plus petits et les plus grands éléments de T, puis à appliquer récursivement l'algorithme sur les tableaux T1 et T2.

Il existe plusieurs méthodes pour partitionner un tableau. Le principe général consiste à utiliser un élément particulier, le pivot, comme valeur de partage.

On peut prendre pour le pivot le premier élément du tableau, comme on peut choisir aléatoirement un élément quelconque du tableau.

Exemple :



La méthode de tri rapide permet d'éviter l'opération de fusion et ainsi gagner en temps de calcul.

L'algorithme détaillé est le suivant :

```
Algorithme TriRapide;
Var T : tableau[1..n] de entier ;
Procédure Partirion( Debut,Fin : entier ; var IPivot : entier);
var Pivot,i,temp : entier ;
Début
    Pivot ← T[Debut] ;
    i ← Debut + 1 ;
    Pour j de Debut+1 à Fin faire
        Si (T[j] ≤ Pivot) Alors
            // Permuter T[i] et T[j]
            temp ← T[i] ;
            T[i] ← T[j] ;
            T[j] ← temp ;
            i ← i + 1 ;
        Fin Si;
    Fin Pour;

    // Permuter T[Debut] et T[i - 1]
    temp ← T[Debut] ;
    T[Debut] ← T[i − 1] ;
    T[i − 1] ← temp ;
    IPivot ← i - 1 ;
Fin;

Procédure TriRapide( Debut,Fin : entier);
var IPivot : entier ;
Début
    Si (Debut<Fin) Alors
        Partition(Debut,Fin,IPivot) ;
        TriRapide(Debut,IPivot-1) ;
        TriRapide(IPivot+1,Fin) ;
    Fin Si;
Fin;

Début
    TriRapide(1,n) ;
Fin.
```

Dans le pire des cas le tableau est déjà trié. Dans ce cas la procédure *Partition* parcourt le tableau jusqu'à la fin et retourne un pivot au début du tableau. La procédure *TriRapide* est relancée alors sur les derniers n-1 éléments. Le temps de calcul peut être donnée par :

$T(n) = n + (n-1) + (n-2) + ...(n-n) = \frac{1}{2}n^2 - \frac{2}{2}n$

La complexité est alors en $O(n^2)$.

Dans le cas moyen le pivot se met à chaque fois au milieu du tableau. La procédure *Partition* parcours le tableau en O(n) est les deux appels de la procédure *TriRapide* sont effectué sur une moitié du tableau chacun. Le temps de calcul peut être trouvé comme dans le cas du tri par fusion :

$T(n) = n + 2T(n/2)$ et la complexité est alors de O(nlogn).

En pratique, c'est l'algorithme le plus utilisé et très souvent, le plus rapide.