Tris récursifs

I3 - Algorithmique et programmation

Nicolas Delestre

Architecture des Systèmes d'Information

Plan. . .

Architecture des Systèmes d'Information

Rappels sur les tris

Objectif

procédure trier **(E/S** t : **Tableau[**1..MAX**] d'**Element,**E** nbElements : **Naturel)**

Les tris vus jusqu'à présent

- Tri à bulles ($O(n^2)$)
- Tri par sélection ($O(n^2)$)
- Tri par insertion ($O(n^2)$)

Principe des tris récursifs

Principe

L'algorithme des tris récursifs est basé sur le principe :

- Diviser : On divise le tableau en deux
- Régner : On trie ces deux tableaux
- Combiner : On combine ces deux tableaux

Il existe deux tris récursifs

- **Le tri rapide** : "l'intelligence" du tri se trouve au niveau de la division du tableau (partitionnement)
- **Le tri par fusion** : "l'intelligence" du tri se trouve au niveau la combinaison des deux tableaux

Architecture des Systèmes d'Information

Le tri rapide (Quick sort) 1 / 9

Principe

1. Partitionner le tableau afin que tous les éléments du sous-tableau gauche soient plus petits ou égaux à un élément (le pivot)
2. Trier le sous-tableau gauche et le sous-tableau droit

algorithme

procédure triRapide **(E/S** t :**Tableau[**1..MAX**] d'Entier**,**E** nb :**Naturel)**
debut
    triRapideRecursif(t,1,nb)
fin
procédure triRapideRecursif **(E/S** t :**Tableau[**1..MAX**] d'Entier**,**E** d,f :**Naturel)**
    Déclaration indicePivot : **Naturel**
debut
    si d<f **alors**
        partionner(t,d,f,indicePivot)
        triRapideRecursif(t,d,indicePivot-1)
        triRapideRecursif(t,indicePivot+1,f)
    finsi
fin

Le tri rapide (Quick sort) 2 / 9

Exemple de partitionnement (pivot = premier élément)

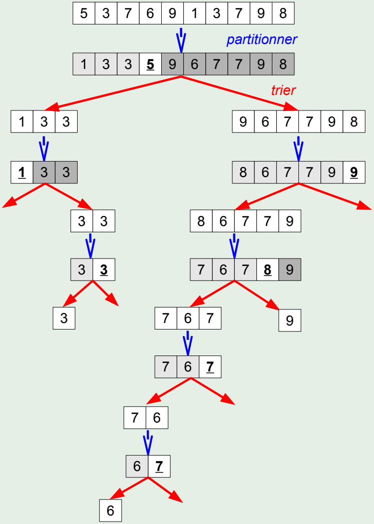
d									f
5	3	7	6	9	1	3	7	9	8

$\downarrow$

d									f
1	3	3	5	9	6	7	7	9	8
			I_p						

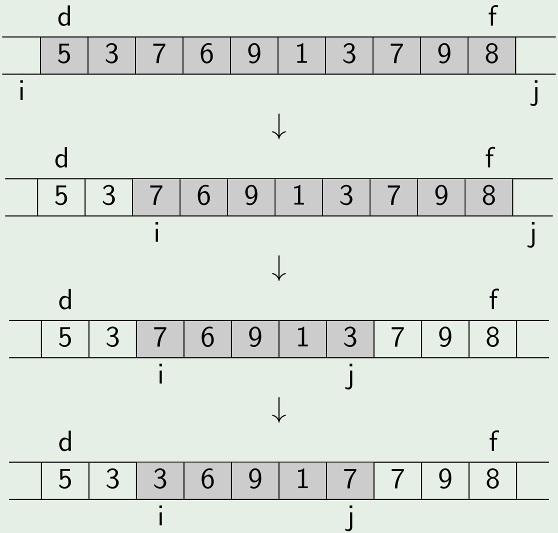
Le tri rapide (Quick sort) 3 / 9

Arbre des appels de procédures



Le tri rapide (Quick sort) 4 / 9

Exemple de fonctionnement du partitionnement



Le tri rapide (Quick sort) 5 / 9

Exemple de fonctionnement du partitionnement

↓

d									f
5	3	3	6	9	1	7	7	9	8
			i			j			

↓

d									f
5	3	3	6	9	1	7	7	9	8
			i		j				

↓

d									f
5	3	3	1	9	6	7	7	9	8
			i		j				

Architecture des Systèmes d'Information

Le tri rapide (Quick sort) 6 / 9

Exemple de fonctionnement du partitionnement

↓

d									f
5	3	3	1	9	6	7	7	9	8
				i	j				

↓

d									f
5	3	3	1	9	6	7	7	9	8
			j	i					

↓

d									f
1	3	3	5	9	6	7	7	9	8
			l_p						

Le tri rapide (Quick sort) 7 / 9

procédure *partitionner* (première version)

procédure partitionner **(E/S** t : **Tableau[**1..MAX**] d'Entier** ; **E** debut,fin : **Naturel** ; **S** indicePivot : **Naturel)**

Déclaration i,j,pivot : **Naturel**

```
debut
    pivot ← t[debut]
    i ← debut
    j ← fin
    tant que i≤j faire
        tant que t[i]≤pivot et i≤j faire
            i ← i+1
        fintantque
        tant que t[j]>pivot et i≤j faire
            j ← j-1
        fintantque
        si i≤j alors
            echanger(t[i],t[j])
        finsi
    fintantque
    indicePivot ← j
    echanger(t[debut],t[j])
fin
```

Le tri rapide (Quick sort) 8 / 9

procédure *partitionner* (deuxième version)

procédure partitionner **(E/S** t : **Tableau[**1..MAX**] d'Entier** ; **E** debut,fin : **Naturel** ; **S** indicePivot : **Naturel)**

Déclaration i,j,pivot : **Naturel**

```
debut
    pivot ← t[debut]
    i ← debut
    j ← fin
    tant que i≤j faire
        si t[i]≤pivot alors
            i ← i+1
        sinon
            si t[j]>pivot alors
                j ← j-1
            sinon
                echanger(t[i],t[j])
            finsi
        finsi
    fintantque
    indicePivot ← j
    echanger(t[debut],t[j])
fin
```

Le tri rapide (Quick sort) 9 / 9

Calcul de la complexité

La procédure de partionnement a une complexité en n. La complexité du tri rapide dépend donc du nombre d'appels récursifs (hauteur h de l'arbre de récursion)

- Dans le meilleur des cas, le partitionnement coupe le tableau en deux parties de même longueur (à plus ou moins 1 près)
  On a : $n = 2^h$, donc $h = log_2 n$
  Donc on a $\Omega(nlog_2 n)$
- Dans le pire des cas, le partionnement coupe le tableau en deux sous tableaux, l'un de longueur 1 et l'autre $n - 1$
  Dans ce cas $h = n$
  Et donc on a $O(n^2)$
- En moyenne on a $\Theta(nlog_2 n)$

Complexité

$\Omega(nlog_2 n)$	$\Theta(nlog_2 n)$	$O(n^2)$

Le tri par fusion 1 / 6

Principe

1. Diviser le tableau en deux sous-tableaux de même longueur (à plus ou moins 1 près)
2. Trier le sous-tableau gauche et le sous-tableau droit
3. Fusionner les deux sous-tableaux

algorithme

procédure triFusion **(E/S** t :**Tableau[**1..MAX**] d'Entier**,**E** nb :**Naturel)**
debut
    triFusionRecursif(t,1,nb)
fin
procédure triFusionRecursif **(E/S** t :**Tableau[**1..MAX**] d'Entier**,**E** d,f :**Naturel)**
debut
    si d<f **alors**
        triFusionRecursif(t,d,(d+f) div 2)
        triFusionRecursif(t,((d+f) div 2)+1,f)
        fusionnner(t,d,(d+f) div 2,f)
    finsi
fin

Le tri par fusion 2 / 6

Fonctionnement de *fusionner*

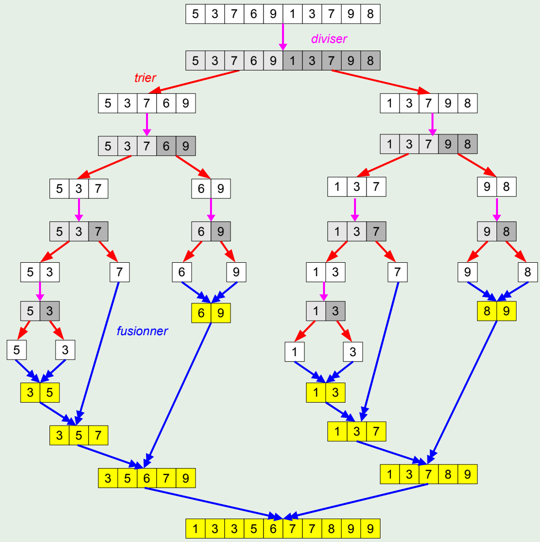
d									f
3	5	6	7	9	1	3	7	8	9

↓

1	3	3	5	6	7	7	8	9	9

Le tri par fusion 3 / 6

Arbre des appels de procédures



Le tri par fusion 4 / 6

Exemple de fonctionnement de *fusionner*

d									f
3	5	6	7	9	1	3	7	8	9

Le tableau intermédiaire :

1									
1	3								
1	3	3							
1	3	3	5						
1	3	3	5	6					
1	3	3	5	6	7				
1	3	3	5	6	7	7			
1	3	3	5	6	7	7	8		
1	3	3	5	6	7	7	8	9	

1	3	3	5	6	7	7	8	9	9

Le tri par fusion 5 / 6

Procédure *fusionner*

procédure fusionner **(E/S** t : **Tableau[**1..MAX**] d'Entier** ; **E** debut,milieu,fin : **Naturel)**

```
    Déclaration   i,j,k : Naturel,
                  temp : Tableau[1..MAX] d'Entier
debut
    i ← debut
    j ← milieu+1
    pour k ←1 à fin-debut+1 faire
        si i≤milieu et j≤fin alors
            si t[i]≤t[j] alors
                temp[k] ← t[i]
                i ← i+1
            sinon
                temp[k] ← t[j]
                j ← j+1
            finsi
        sinon
            si i≤milieu alors
                temp[k] ← t[i]
                i ← i+1
            sinon
                temp[k] ← t[j]
                j ← j+1
            finsi
        finsi
    finpour
    pour k ←1 à fin-debut+1 faire
        t[debut+k-1] ← temp[k]
    finpour
fin
```

Le tri par fusion 6 / 6

Calcul de la complexité

Soit h La hauteur de l'arbre de récursion
Ici on a toujours $n = 2^h$, donc $h = log_2 n$
Donc en temps on a $\Omega(nlog_2 n)$, $O(nlog_2 n)$ et $\Theta(nlog_2 n)$
Mais on a besoin d'un tableau intermédiaire pour fusionner

Complexité

$\Omega(nlog_2 n)$	$\Theta(nlog_2 n)$	$O(nlog_2 n)$

Conclusion

Il existe plusieurs algorithmes de tri que l'on peut classer suivant :

- les méthodes utilisées (itératifs ou récursifs)
- les performances

Ces cours ne présentent pas toutes les méthodes de tri, entre autres :

- le *shellsort*
- le *heapsort* (ou tri par tas)
  - L'un des meilleurs car en $O(nlog_2 n)$ et *itératif*
- le *radixsort*
- etc.

Architecture des Systèmes d'Information