Chap-5 : *Algorithmes de tri: analyse et estimation de leur complexité*

Année universitaire :2020/2021

1. Tri par sélection

Principe

- Le tri par sélection est basé sur l'idée suivante : Sélectionner le minimum ou le maximum et le déplacer au début ou à la fin du tableau.

Algorithme

```
For I := 1 To N-1 Do
   K := I
   For J = I+1 To N Do
            If ( A(j) < A(k)) Then
                        K := J
            Endif
   Endfor
   Y := A(I)           ; A(I) := A(k)          ; A(k) := Y
Endfor
```

Quelle est la complexité de cet algorithme ?

2. Tri par insertion linéaire

Principe

- Le tri par insertion est basé sur les principes suivants : dans le tableau à insérer, on suppose qu'une partie a été triée et qu'il reste à trier l'autre partie. La partie triée est appelée "séquence destination" et la partie qui reste à trier est appelée "séquence source"

a_1 , a_2 , a_3 , ….. a_{i-1} , a_i, a_{i+1}, ….. a_n

séquence destination   séquence source

(triée)   (non triée)

2. Tri par insertion linéaire

Algorithme

```
For I := 2 To N Do
   X := A(I)   ;   J := I
   While ( X < A (j-1)  and j > 1) Do
            A(j) := A(j-1) ;  j := J-1
   Endwhile
   A(j) := X
Endfor
```

Quelle est la complexité de cet algorithme ?

3. Tri par insertion binaire

Principe

- La séquence destination étant ordonnée, nous pouvons donc appliquer une recherche binaire ce qui accélèrera la recherche.

Algorithme

```
For I := 2 To N Do
   X := A(I)
   k=Recherche_Binaire( A , I-1, X)
   Decaler_Droite( A , k, I-1)
   A (k) := X
Endfor
```

Quelle est la complexité de cet algorithme ?

4. Tri à bulles

Principe

- Cette méthode est basée sur l'idée de comparer toute paire d'éléments adjacents et de les permuter s'ils ne sont pas dans le même ordre et répéter le processus jusqu'à ce qu'il n'y ait plus de permutations.

Algorithme

```
Repeat
    No_permutation := True
    For I := 1 To N-1 Do
            If (A(I) > A (I+1)) Then
                    X :=A(I) ; A(I) := A(I+1) ; A(I+1) := X
                    No_Permutation := False
            Endif
    Endfor
Until ( No_permutation)
```

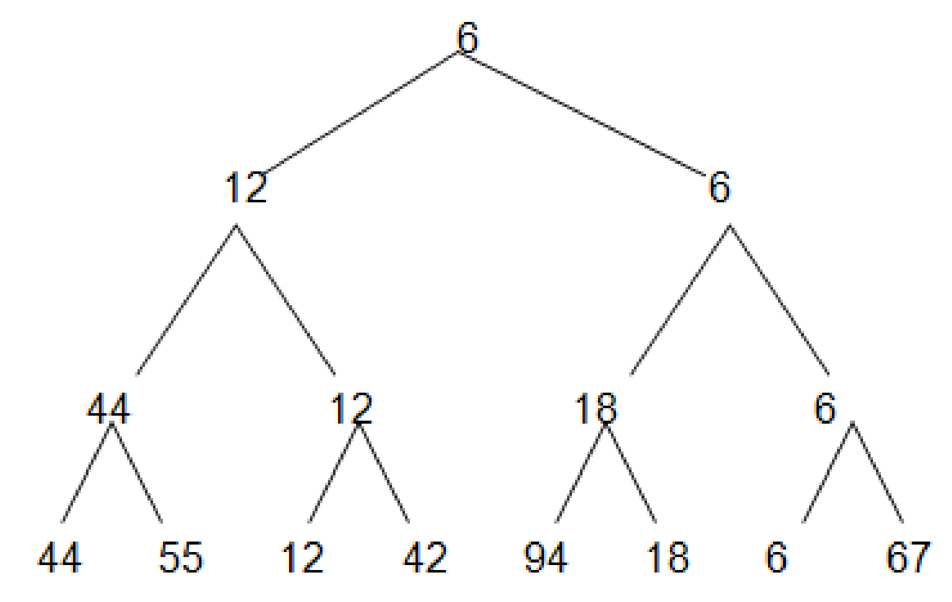
Quelle est la complexité de cet algorithme ?

5. Tri arborescent

Principe

- La méthode du tri arborescent veut mémoriser toute information obtenue à l'issu des comparaisons pour l'exploiter dans l'établissement de l'ordre final. Pour ce faire elle construit une arborescence qui traduit la relation qui existe entre tous les éléments.
- Soit le tableau suivant à ordonner :

44	55	12	42	94	18	6	67



Quelle est la complexité de cet algorithme ?

5. Tri par fusion

Principe

- L'algorithme de tri par fusion est construit suivant le paradigme « diviser pour régner » :

1. Il divise la séquence de n nombres à trier en deux sous-séquences de taille $n/2$.
2. Il trie récursivement les deux sous-séquences.
3. Il fusionne les deux sous-séquences triées pour produire la séquence complète triée.

La récurrence se termine quand la sous-séquence à trier est de longueur 1

5. Tri par fusion

Algorithme

- La principale action de l'algorithme du tri par fusion est justement la fusion des deux listes triées.

La fusion

- Le principe de cette fusion est simple: à chaque étape, on compare les éléments minimaux des deux sous-listes triées, le plus petit des deux étant l'élément minimal de l'ensemble on le met de côté et on recommence. On conçoit ainsi un algorithme « Fusionner » qui prend en entrée un tableau A et trois entiers, p, q et r, tels que $p<=q < r$ et tels que les tableaux $A[p..q]$ et $A[q+1..r]$ sont triés.

5. Tri par fusion

Le tri

Fusionner (A, p, q, r) // Fusionner les 2 sous-tableaux A(p→q) et A(q+1→r)
i=p // les 2 sous tableaux sont supposés triés
j=q+1 ; k=1
while (i<=q et j<=r) do
if (A[i] < A[j]) then
C[k] = A[i] ; i=i+1
else
C[k]=A[j] ; j=j+1
endif
k=k+1
Endwhile
	While i<=q do
	C[k]=A[i] ; i=i+1 ; k=k+1
	Endwhile
	While j<=r do
	C[k]=A[j] ; j=j+1 ; k=k+1
	Endwhile
	For k=1 to r-p+1 do // Recopier le tableau C dans le tableau original
	A(p+k-1)= C(k)
	Endfor
	EndFusionner

5. Tri par fusion

```
Tri_Fusion(A, p, r)
If (p<r ) then
q= (p+r)/2
Tri_Fusion(A, p, q)
Tri_Fusion(A,q+1,r)
Fusionner(A,p,q,r)
   Endif
End_Tri_Fusion
```

5. Tri par fusion

Quelle est la complexité de cet algorithme ?

Complexité de la fusion

Études les différentes étapes de l'algorithme :

– les initialisations ont un coût constant Q(1) ;

– la boucle *While* de fusion s'exécute au plus *r*-*p* fois, chacune de ses itérations étant de coût constant, d'où un coût total en *O*(*r*-*p*) ;

– les deux boucles *While* complétant *C* ont une complexité respective au pire de *q*-*p*+1 et de *r*-*q*, ces deux complexités étant en *O*(*r*-*p*) ;

– la recopie finale coûte Q(*r*-*p*+1).

Par conséquent, l'algorithme de fusion a une complexité en Q(*r*-*p*).

5. Tri par fusion

Complexité de l’algorithme de tri par fusion

l’algorithme Tri_Fusion est de type « diviser pour régner ». Il faut donc étudier ses trois phases:

Diviser : cette étape se réduit au calcul du milieu de l’intervalle [*p*,*r*], sa complexité est donc en Q(1).

Régner : l’algorithme résout récursivement deux sous-problèmes de tailles respectives (n/2) , d’où une complexité en 2*T*(n/2).

Combiner : la complexité de cette étape est celle de l’algorithme de fusion qui est de Q(*n*) pour la construction d’un tableau solution de taille *n*.

Par conséquent, la complexité du tri par fusion est donnée par la récurrence :

$$T(n)=\begin{cases}\Theta(1) & si \quad n=1\\ 2T(n/2)+\Theta(n)\sin on & \end{cases}$$

5. Tri par fusion

- Pour déterminer la complexité du tri par fusion, nous utilisons le théorème de résolution des récurrences avec : $a = 2$ et $b = 2$, donc log$b$ a = 1 et nous nous trouvons dans le deuxième cas du théorème : $f(n)$ = Q($n$log$b$ a) = Q(n). Par conséquent :

$$T(n) = \Theta(n \log n)$$

- Pour des valeurs de n suffisamment grandes, le tri par fusion avec son temps d'exécution en Q($n$log$n$) est nettement plus efficace que le tri par insertion dont le temps d'exécution est en Q(n^2).

6. Tri rapide (Quicksort)

Principe

- Le tri rapide est fondé sur le paradigme « diviser pour régner », tout comme le tri fusion, il se décompose donc en trois étapes :

Diviser : Le tableau $A[p..r]$ est partitionné (et réarrangé) en deux sous-tableaux non vides, $A[p..q]$ et $A[q+1..r]$ tels que chaque élément de $A[p..q]$ soit inférieur ou égal à chaque élément de $A[q+1..r]$.

L'indice q est calculé pendant la procédure de partitionnement.

Régner : Les deux sous-tableaux $A[p..q]$ et $A[q+1..r]$ sont triés par des appels récursifs.

Combiner : Comme les sous-tableaux sont triés sur place, aucun travail n'est nécessaire pour les recombiner, le tableau $A[p..r]$ est déjà trié !

6. Tri rapide (Quicksort)

Algorithme

```
Tri_Rapide (A, p, r)
        If (p < r ) then
                q =Partionner  (A, p, r)
                Tri_Rapide(A, p, q)
                Tri_Rapide (A, q+1, r)
        Endif
End_Tri_Rapide
```

```
Partionner (A, p, r)
        x = A(p)
        i = p-1
        j= r+1
        while (1)
                repeat  { j=j-1 } until A(j) <= x
                repeat { i =i+1 } until A(i) >= x
                if ( i < j )
                        permuter (A(i), A(j))
                        else return j
End_Partionner
```

6. Tri rapide (Quicksort)

Complexité

Pire cas

Le cas pire intervient quand le partitionnement produit une région à *n*-1 éléments et une à 1 élément.

Comme le partitionnement coûte Q(*n*) et que *T*(1) = Q(1), la récurrence pour le temps d'exécution est :

$$T(n) = T(n-1) + \Theta(n)$$

et par sommation on obtient :

$$T(n) = \sum_{k=1}^{n} \Theta(k) = \Theta(\sum_{k=1}^{n} k) = \Theta(n^2)$$

6. Tri rapide (Quicksort)

Complexité

Meilleur cas

Le meilleur cas intervient quand le partitionnement produit deux régions de longueur n/2.

La récurrence est alors définie par :

$$T(n) = 2T(n/2) + \Theta(n)$$

ce qui donne d'après le théorème de résolution des récurrences :

$$T(n) = \Theta(n \log n)$$

6. Tri rapide (Quicksort)

Complexité

Complexité moyenne

Pour avoir une complexité moyenne, on tire au hasard l'indice de départ de partitionnement. Et on démontre que la complexité moyenne est aussi égale à :

$$T(n) = \Theta(n \log n)$$