VARI
4 LES TRIS

PLAN

- Le problème du tri
- Le tri par insertion
- Le tri fusion
- Le tri par tas
- Le tri rapide
- Comparaison des tris
- Le tri par dénombrement

4.1 LE PROBLEME DU TRI

4-1-1 DÉFINITION

entrée: suite de n éléments $e_1, e_2, .., e_n$

sortie: permutation de la suite

$$e'_1, e'_2, .., e'_n$$

$$e'_1 \leq e'_2 \leq .. \leq e'_n$$

condition: ensemble totalement ordonné

EXEMPLE: clés, $\mathbb{N}$, alphabet,..

- **problème simple, connu et courant**

- **nombreuses solutions avec structures de données variées**

- **sous-problème de problèmes complexes**

4-1-2 IMPLÉMENTATION

rappel: tri par sélection (cours 1) **en $O(n^2)$**

tri à bulle: (variante) **en $O(n^2)$**

faire

parcours à partir de la fin de l

comparaison de e_i avec e_{i-1}

échange si $e_{i-1} \geq e_i$

jusqu'à **aucun échange possible**

Etude dans la suite d'algorithmes plus efficaces

Si on veut trier des éléments (classe Elt), le tri doit se faire sur les clés des éléments. Pour simplifier la présentation, dans ce chapitre, on trie des entiers et on utilise donc des tableaux d'entiers. On pourrait aussi utiliser des tableaux de clés (CClé) ou des tableaux d'éléments (Elt).

```
classe CTab{
        entier taille;          //taille du tableau
        entier [ ] tab;         //ce tableau contient des entiers
        entier long=0;          //nombre d'éléments dans le tableau

        CTab( entier t){
        this.taille=t;
        this.tab=new entier[t];
        }
}
```

Dans tout ce chapitre nous supposerons que les tableaux sont indicés de 1 à n   (et non de 0 à n-1) Classe CIndice

4-2 LE TRI PAR INSERTION

principe: **éléments placés un à un « directement » à leur place**

4-2-1 INSERTION SÉQUENTIELLE

EXEMPLE

7 15 8 10 5 17

7 15 8 10 5 17 //décalage à droite de 15

7 8 15 10 5 17 //décalage à droite de 15

7 8 10 15 5 17 //décalage à droite de 7, 8, 10 et 15

5 7 8 10 15 17

5 7 8 10 15 17

<u>implémentation</u>

```
void  tri_inser_seq  ( )     //tri d'un tableau d'entiers ou de clés
CIndice i,j; entier  clé;
début
pour  j = 2 à long  faire
        clé = tab[j] ; //clé à placer
        i = j-1 ;
        tant que  i > 0  et tab[i] > clé faire
                //décalage vers la droite
                tab[i+1] = tab[i] ;
                i = i-1 ;
        fait;
        tab[i+1] = clé ; //mise de clé définitivement à sa place
fait;
fin
```

complexité

- **pire des cas**

-boucle pour : j de 2 à n (=long)

-itération j,

boucle tant que : j fois

$$\rightarrow O(2 + 3 + .. + n) \text{ opérations}$$

insertion séquentielle en $O(n^2)$

complexité

- **en moyenne**

-boucle pour : j de 2 à n

-itération j,

boucle tant que : j/2 fois

$\rightarrow$ **O(2 + 3 +..+ n)/2 opérations**

en moyenne en $O(n^2)$

4-2-2 INSERTION DICHOTOMIQUE

recherche dichotomique de la place où insérer l'élément j (dans la première partie triée de la liste)

(cf cours 1)

à chaque itération:

recherche de la place en O(log n)

mais déplacements en O(n)

complexité en $O(n^2)$

4-2-3 COMPLEXITÉ EN MÉMOIRE

tris par insertion et sélection:

tris "sur place"

complexité mémoire en O(1)

4-3 LE TRI FUSION

4-3-1 LA FUSION

fusion de 2 listes ordonnées

sélection et retrait du plus petit des 2 premiers éléments jusqu'à avoir parcouru les 2 listes

T1	1 4 5 6 9
T2	2 3 4
T	

T1	1 4 5 6 9
T2	2 3 4
T	1

T1	1 4 5 6 9
T2	2 3 4
T	1 2

T1	1 4 5 6 9
T2	2 3 4
T	1 2 3

T1	1 4 5 6 9
T2	2 3 4
T	1 2 3 4

T1	1 4 5 6 9
T2	2 3 4
T	1 2 3 4

T1	1 4 5 6 9
T2	2 3 4
T	1 2 3 4 4

T1	1 4 5 6 9
T2	2 3 4
T	1 2 3 4 4 5 6 9

UN EXEMPLE

T1	**1 4 5 6 9**
T2	**2 3 4**

n1 termes

n2 termes

Tant qu'il reste des éléments dans les deux tableaux
on sélectionne le plus petit

T1	**1 4 5 6 9**
T2	**2 3 4**
T	**1 2 3 4 4**

Quand on est au bout de l'un des tableaux
on recopie le reste de l'autre.

i	**T1**	**1 4 5 6 9**
j	**T2**	**2 3 4**
k	**T**	**1 2 3 4 4 5 6 9**

UN EXEMPLE suite

```
void fusion (CTab t1; CTab t2; CTab t)
//fusionne les deux tableaux ordonnés t1 et t2 en un seul tableau ordonné t
CIndice        i, j, k = 1;
début
n1=t1.long; n2=t2.long;
tant que  i ≤ n1 et j ≤ n2  faire  //jusqu'à "avoir vidé" l'un des 2 tableaux
    si   t1.tab[i] ≤ t2.tab[j]  alors   //sélection dans tab du plus petit des 2
    tableaux
              t.tab[k] = t1.tab[i] ;
              i = i+1 ;
    sinon     t.tab[k] = t2.tab[j] ;
              j = j+1 ;
    finsi;
    k := k+1 ;
fait;
```

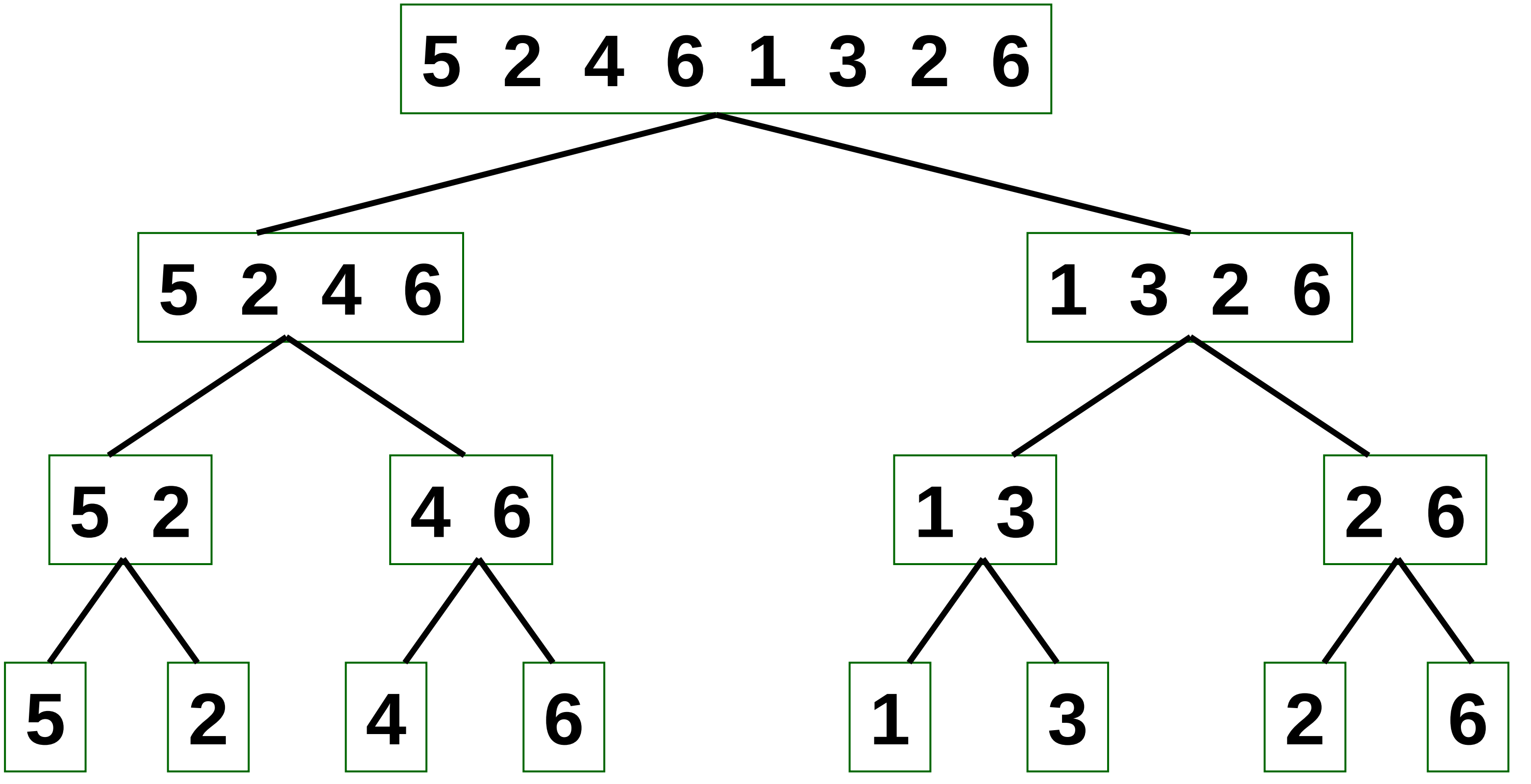
```
si  i ≤ n₁  alors    //ajouter les éléments restants de t₁.tab
      tant que  i ≤ n₁  faire
            t.tab[k]=t₁.tab[i]; i=i+1;  k = k+1;
      fait;
sinon                    //ajouter les éléments restants de t₂.tab
      tant que  j ≤ n₂  faire
            t.tab[k]=t₂.tab[j]; j=j+1;
            k = k+1;
      fait;
finsi;
fin
```

complexité: $O(n_1+n_2)$

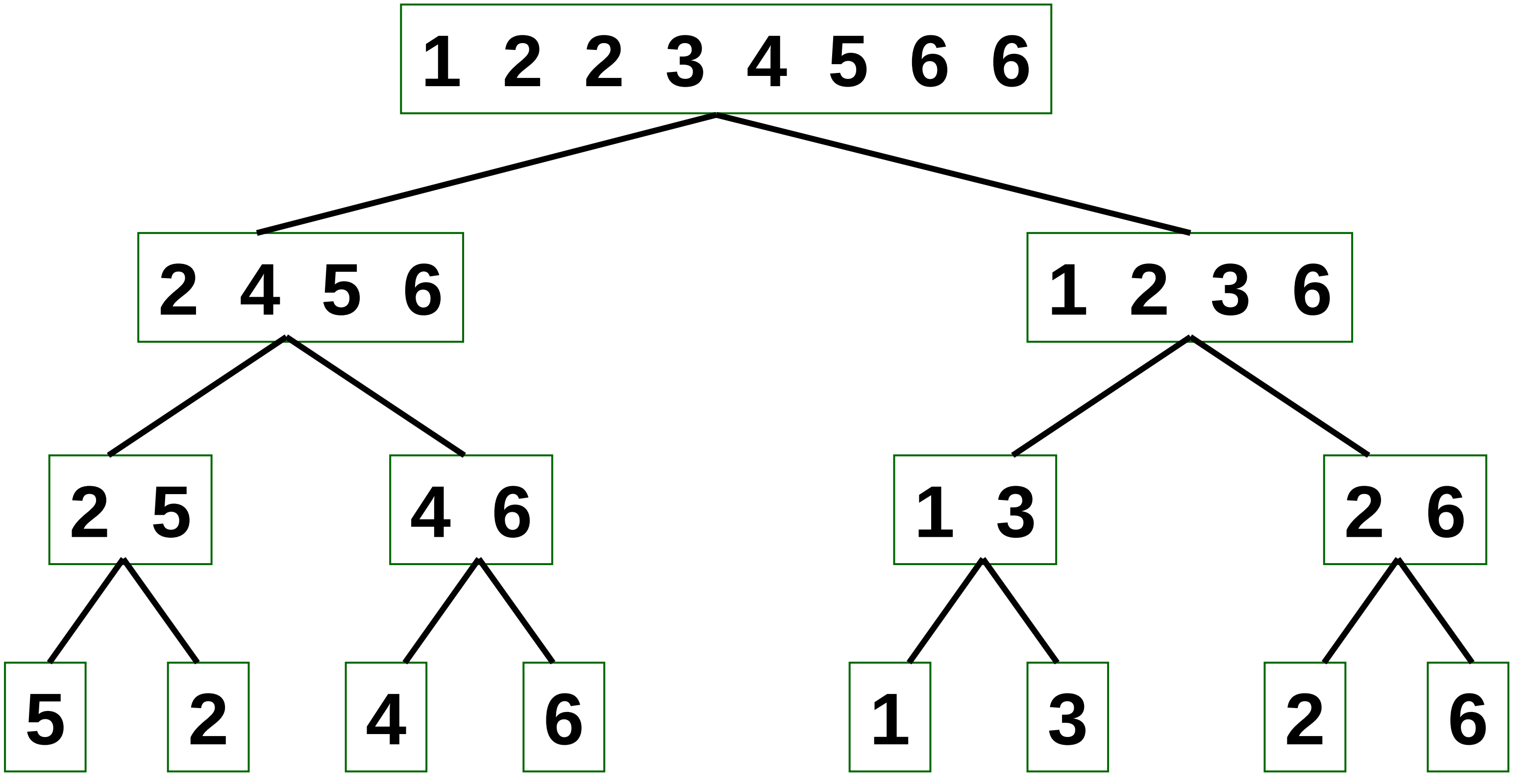
4-2-2 TRI FUSION

procédure récursive:

- **diviser la séquence de n éléments en 2 sous-séquences de n/2 éléments** (ou n/2 et n/2+1)
- **trier chaque sous-séquence avec tri-fusion**
- **fusionner les 2 sous-séquences triées**

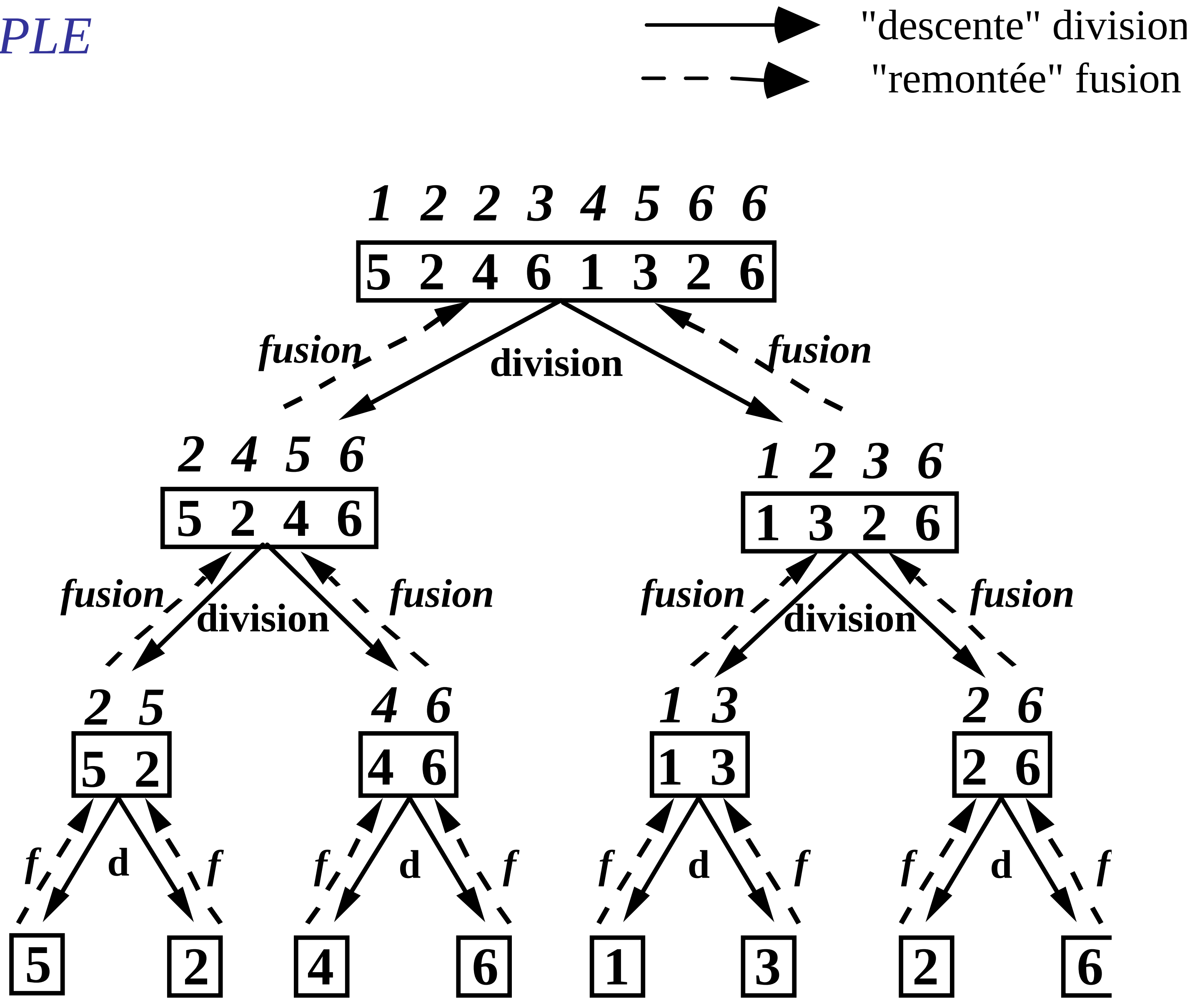


Division



Fusion

EXEMPLE



"descente" division
"remontée" fusion
1 2 2 3 4 5 6 6
5 2 4 6 1 3 2 6
fusion
division
fusion
2 4 5 6
5 2 4 6
1 2 3 6
1 3 2 6
fusion
division
fusion
fusion
division
fusion
2 5
5 2
4 6
4 6
1 3
1 3
2 6
2 6
f
d
f
5
2
4
6
1
3
2
6

	g m d	
a h n o p r v	b f s w	c k t

b c f k s t w

tri "sur place" $\rightarrow$ ***un seul tableau***

void **fusionner** **(CTab t, Indice g, Indice d)**

- *à partir de* **fusion(t_1 ,t_2 ,t)**
- m=(g+d)/2
- *t.tab: de g à d* *t_1.tab : de g à m* *t_2.tab : de m+1 à d*
  suppose que les éléments de g à m (et de m+1 à d) sont ordonnés

(on suppose que cette procédure est fournie: implémentation non donnée ici)

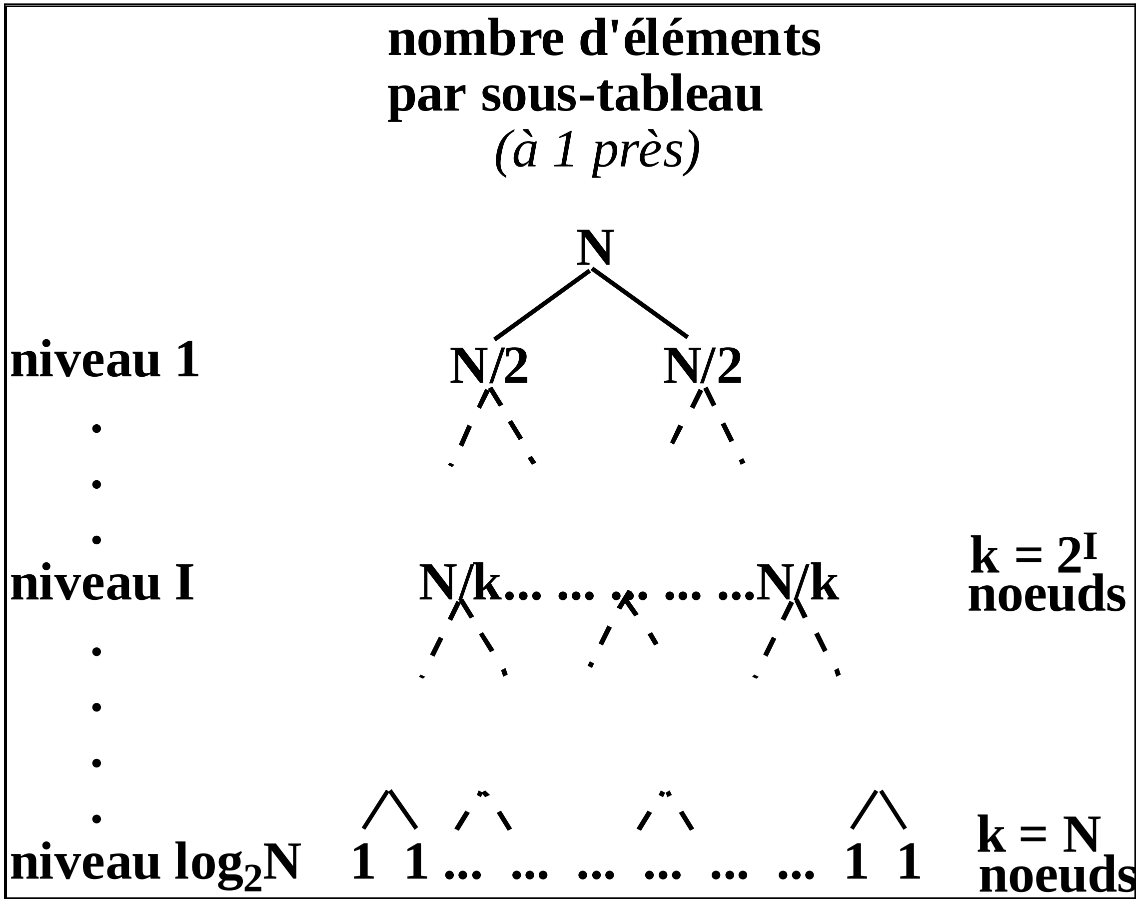
```
void tri-fusion (CTab t , Indice i, Indice j)
//tri-fusion du sous-tableau de t.tab allant de i à j
CIndice        d, m, g ;
début
  si  i > j  alors "erreur";
  si  i < j  alors                   //il y a plus d'un élément dans le sous-tableau
        g = i ; d = j ;
        m = i+j / 2 ;                //séparation en 2 "sous-sous-tableaux"
        tri-fusion (t.tab,g,m) ;
        tri-fusion (t.tab,m+1,d);
        fusionner (t.tab,g,d);
  finsi ;                            //si i = j  il n'y a qu'un élément: fin ;
fin ;
```

appel: **tri-fusion(t.tab,1,n);**

tri "sur place": ***complexité mémoire en O(1)***

4-2-3 COMPLEXITÉ ET RÉCURRENCE

a) complexité du tri-fusion



niveau I: **$O(k \cdot N/k) = O(N)$ opérations**

<u>complexité</u> du tri-fusion

$O(n \log n)$

tri-fusion = très bon tri

b)complexité des algorithmes récursifs *(facultatif)*

"diviser pour régner "

taille du problème: n

- **relation de récurrence**

♦(n): O(temps d'exécution pour n)

- "diviser" $\rightarrow$ O(1)
- "régner" $\rightarrow$ 2 ♦(n/2)
- fusionner $\rightarrow$ O(n)

♦(n) = | O(1) si n = 1
| 2 ♦(n/2)+O(n) si n > 1

par substitutions on obtient:

♦(n) = O(n logn)

- **autres résultats**

- ♦: fonction croissante et ♦(1)=1

- $n>1$, $n = 2^I$, c = constante

♦(n) = ♦(n/2)+c $\Rightarrow$ ♦(n) = O(logn)

♦(n) = 2♦(n/2)+cn $\Rightarrow$ ♦(n)=O(nlogn)

♦(n) = 2♦(n/2)+cn^2 $\Rightarrow$ ♦(n) = O(n^2)

♦(n) = 4♦(n/2)+cn^2 $\Rightarrow$ ♦(n) = O(n^{2}logn)

4-4 LE TRI PAR TAS

4-4-1 L'ALGORITHME

tas: (cf cours 3) **arbre parfait tel que tout noeud a une valeur ≤ à celle de tous ses descendants**

méthodes: **estvide, min_tas,**
insérer, supprimer_min

principe du tri par tas:

- transformer la séquence initiale en tas
- extraire un à un les éléments min (racines) en conservant la structure de tas

l: liste à trier de n éléments (classe CTab)

letas: tas utilisé pour le tri (classe C_Tas)

Dans ces deux classes on utilise une représentation par un tableau dont les indices sont de la classe CIndice (entiers allant de 1 à la taille du tableau)

```
void  tri_par_tas  (CTab  l)
entier val; C_Tas  letas= new C_Tas (l.long) ; entier n=l.long;
début
pour  k=1   à  n faire      //construction du tas associé à l
            val = l.tab[k];
            letas.insérer (val) ;
fait ;
pour  k=1 à  n faire
    //sélection  successive des min du tas qui sont reportés à leur place dans l
    val = letas.min_tas;
    l.tab[k]= val ;
    //suppression du min en gardant la structure de tas
    letas.supprimer_min;
fait ;
fin ;
```

```
void tri_par_tas (CTab l)
entier val; C_Tas letas= new C_Tas (l.long) ; entier n=l.long;
début
pour k=1 à n faire
        val = l.tab[k];
        letas.insérer (val) ;      | O(log n)
fait ;                                          n fois
pour k=1 à n faire
    val = letas.min_tas;   | O(1)
    l.tab[k]= val ;                             n fois
    letas.supprimer_min;   | O(log n)
fait ;
fin ;
```

4-4-2 COMPLEXITÉ

-n itérations pour chaque boucle

-insérer et supprimer en O(log n)

complexité du tri par tas
O(n log n)

espace mémoire: ici O(n)

mais, tri "sur place" possible avec programmation de même principe un peu plus complexe

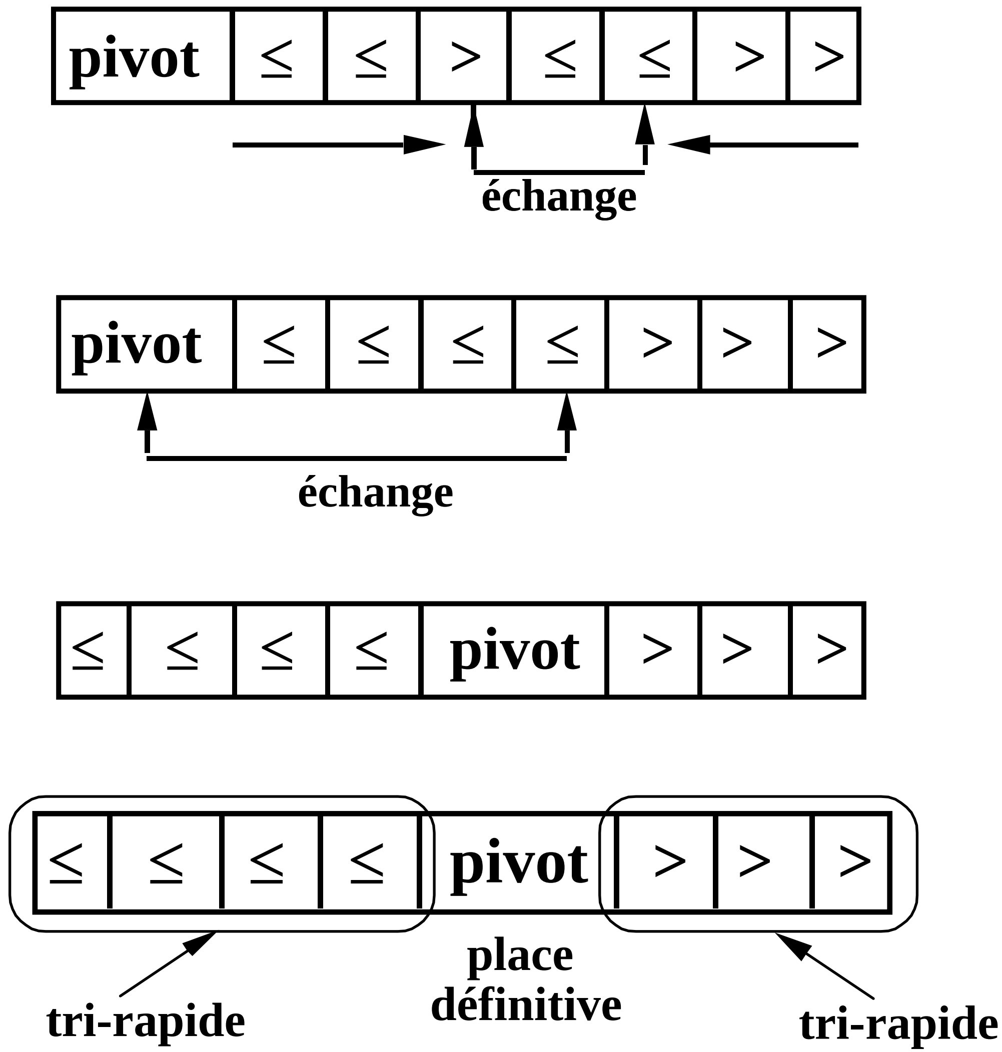
complexité en mémoire O(1)

tri par tas: très bon tri

4-5 LE TRI RAPIDE

(implémentation non étudiée ici)

principe



LE TRI RAPIDE

7	10	3	12	4	6	18	15

7	6	3	12	4	10	18	15

7	6	3	4	12	10	18	15

4	6	3	7	12	10	18	15

LE TRI RAPIDE

4	6	3	7	12	10	18	15

4	6	3

7

12	10	18	15

3	4	6

10	12	15	18

3	4	6	7	10	12	15	18

procédure récursive :

- **sélectionner un élément pivot p**
- **partitionner la liste à trier en 2 sous-listes:**

  à gauche du pivot, éléments $\leq$ p

  à droite du pivot, éléments > p

- **- tri-rapide des 2 sous-listes**

  - concaténation des listes triées

tri "sur place" (+ pile)

complexité du tri rapide

au pire: $O(n^2)$

en moyenne: $O(n \log n)$

tri rapide: très bon tri en général

4-6 COMPARAISON DES TRIS PAR COMPARAISON

de 4.1 à 4.5: tris avec tests comparatifs

4-6-1 TRIS ET ORDINATEURS

tri de n nombres

- **tri par insertion**

  sur super-ordinateur 100 mips

  langage machine

  programmeur champion

  $\rightarrow$ $2n^2$ instructions

- **tri fusion**

sur micro **1 mips**

compilateur peu efficace

programmeur moyen

→ 50nlogn instructions

si n = 10^6

tri du super-ordinateur (insertion)**:**

$2.(10^6)^2/10^8 = 20\,000$ sec = 5,56h

tri du micro (fusion)**:**

$50.10^6.\log 10^6 / 10^6 = 1\,000$ sec

= 16,67 mn

(cf cours 2: complexité)

4-6-2 CHOIX D'UN TRI

peu important si petite liste

sur une même machine:

tri rapide	tri par insertion séq	
0,2s	**1,6s**	**pour 250 éléments**
0,9s	**24s**	**pour 1000 éléments**
4,4s	**6,6mn**	**pour 4000 éléments**

types de tris	complexité	
	au pire	moyenne
sélection	n^2	n^2
insertion	n^2	n^2
fusion	nlogn	nlogn
par tas	nlogn	nlogn
rapide	n^2	nlogn

nlogn = borne non améliorable pour tris par comparaisons

Conclusion

- **tri par tas: meilleure complexité**
- **tri rapide et tri fusion: efficaces**
- **tri par insertion excellent si liste initiale presque triée**

et

- **tri par sélection donne le début de liste trié avant la fin du tri**
- **tri par insertion peut débuter sans liste initiale complète**

4-7 TRI PAR DÉNOMBREMENT

4-7-1 PRINCIPE

tri très efficace **<u>si les n nombres</u>**
<u>à trier son petits</u>
(au moins tous ≤ n)

aucune comparaison

pour chaque élément e:

p= nombre d'éléments < e

q= nombre d'éléments = e

places des éléments égaux à e:

p+1,p+2,..,p+q-1, p+q

valable pour tout ensemble E t.q.

$\exists$ bijection entre E et {1,..,n}

EXEMPLE **alphabet $\longleftrightarrow$ {1,..,26}**

4-7-2 PROCÉDURE

A: tableau de n entier; *à trier*

B: tableau de n entier; *résultat*

k=

- nombre d'éléments différents contenus dans A si ce nombre est connu

- plus grand élément de A sinon

CTab tri-dénombrement (CTab A, entier k)

CTab B (A.long); CTab C (k); entier n=A.long;

début *//pour simplifier on note A[i] au lieu de A.tab[i], idem pour B et C*

pour **i =1** à **k** faire

C[i] = 0;

fait;

pour **j =1** à **n** faire

C[A[j]] = C[A[j]]+1;

fait;

//C[i] contient le nombre d'éléments de A égaux à i

pour **i =2** à **k** faire

C[i] = C[i] + C[i-1];

fait;

…

…

```
//C[i] contient le nombre d'éléments égaux ou < à i
//il reste à placer les  éléments dans B
    pour  j :=n  à  1 pas -1 faire
          B[C[A[j]]] := A[j];
          //l'élément A[j] doit être mis dans B à la place C[A[j]]
          C[A[j]] := C[A[j]]-1;
          //cas d'égalité de 2 éléments: le deuxième est en C[A[j]]-1
    fait;
retourner B;
fin
```

EXEMPLE

i=	1	2	3	4	5	6	7	8
A	**3**	**6**	**4**	**1**	**3**	**4**	**1**	**4**

k = 6

C	**2**	**0**	**2**	**3**	**0**	**1**

nombre d'éléments = i

C	**2**	**2**	**4**	**7**	**7**	**8**

nombre d'éléments Š i

	1	2	3	4	5	6	7	8	
B	**1**	**1**	**3**	**3**	**4**	**4**	**4**	**6**	
	5	***2***	***8***	***4***	***6***	***3***	***1***	***7***	*ordre de remplissage*

complexité du tri par dénombrement

si k = O(n), tri en O(n)

meilleure complexité possible pour un tri de n nombres

4-7-3 REMARQUES DIVERSES

- **autres tris efficaces: tris par base, par paquets,…**

- **tous les tris étudiés supposent que tous les nombres à trier sont présents en mémoire centrale**

- **si le nombre d'objets à trier est trop grand: tris externes avec minimisation des entrées-sorties**