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

non précisée dans la spécification

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**

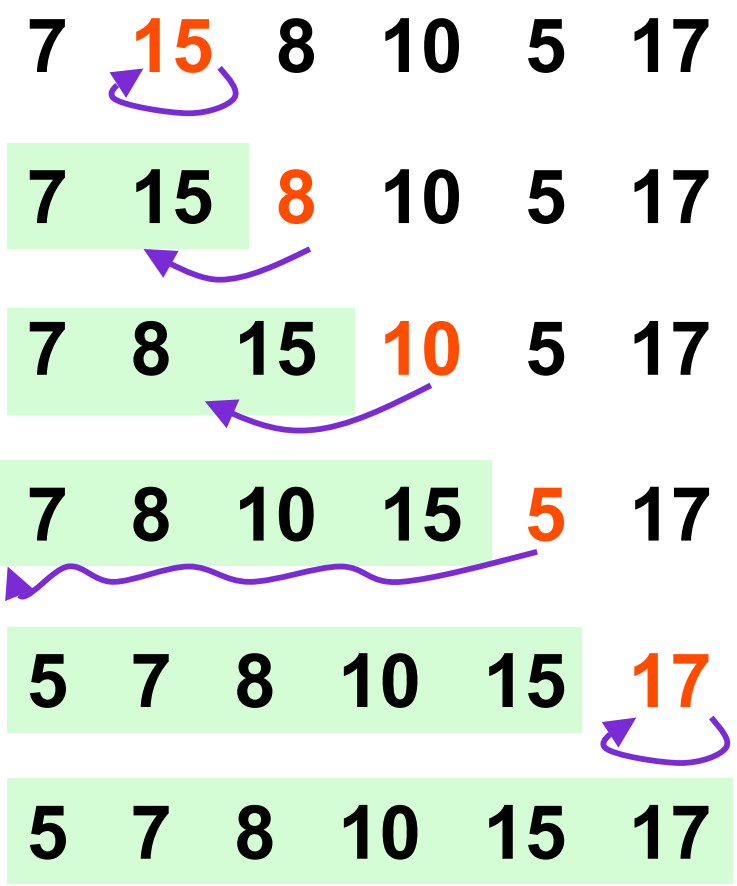
Etude dans la suite d'algorithmes plus efficaces

4-2 LE TRI PAR INSERTION

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

4-2-1 INSERTION SÉQUENTIELLE

EXEMPLE



//échange de 8 et 15

//échange de 10 et 15

//décalage à droite de 7, 8, 10 et 15

implémentation

```
void tri_inser_seq ( )      //tri d'un tableau d'entiers ou de clés
Indice i,j; entier clé;
début
pour j = 2 à long faire
        clé = tab[j] ;
        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

→ O(2 + 3 +..+ n) 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

→ O(2 + 3 +..+ n)/2 opérations

en moyenne en $O(n^2)$

4-2-2INSERTION 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	4	

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	4	

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

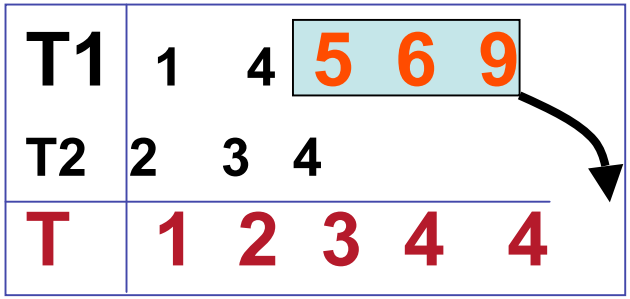
UN EXEMPLE

T1	1	4	5	6	9
T2	2	3	4		

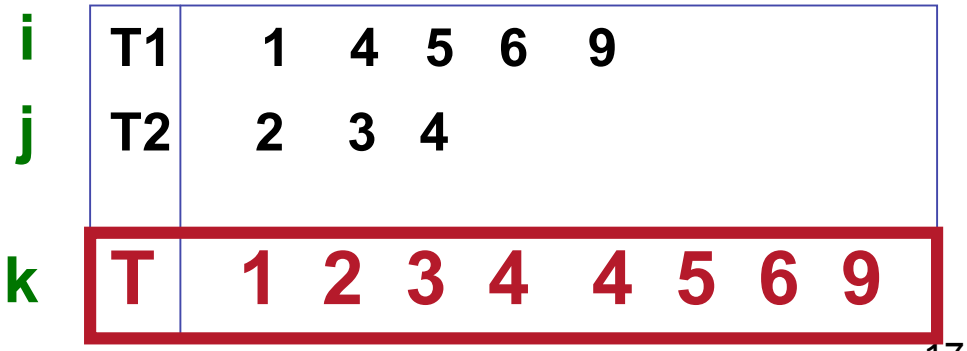
n1 termes

n2 termes

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



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



UN EXEMPLE suite

```
void fusion (int[] t1, int[] t2 , int[] t)
//fusionne les deux tableaux ordonnés t1 et t2 en un seul tableau ordonné t
Indice  i, j, k = 0;
début
n1=t1.long-1; n2=t2.long-1;
tant que  i ≤ n1 et j ≤ n2  faire  //jusqu'à "avoir vidé" l'un des 2 tableaux
    si  t1 [i] ≤ t2 [j]  alors  //sélection dans tab du plus petit des 2 tableaux
                t [k] = t1 [i] ;
                i = i+1 ;
    sinon       t [k] = t2 [j] ;
                j = j+1 ;
    finsi;
    k := k+1 ;
fait;
```

```
    si  i ≤ n1  alors   //ajouter les éléments restants de tab1
          tant que  i ≤ n1  faire
                t [k]=t1 [i]; i=i+1;  k = k+1;
          fait;
    sinon                    //ajouter les éléments restants de tab2
          tant que  j ≤ n2  faire
                t [k]=t2 [j]; j=j+1;
                k = k+1;
          fait;
    finsi;
fin
```

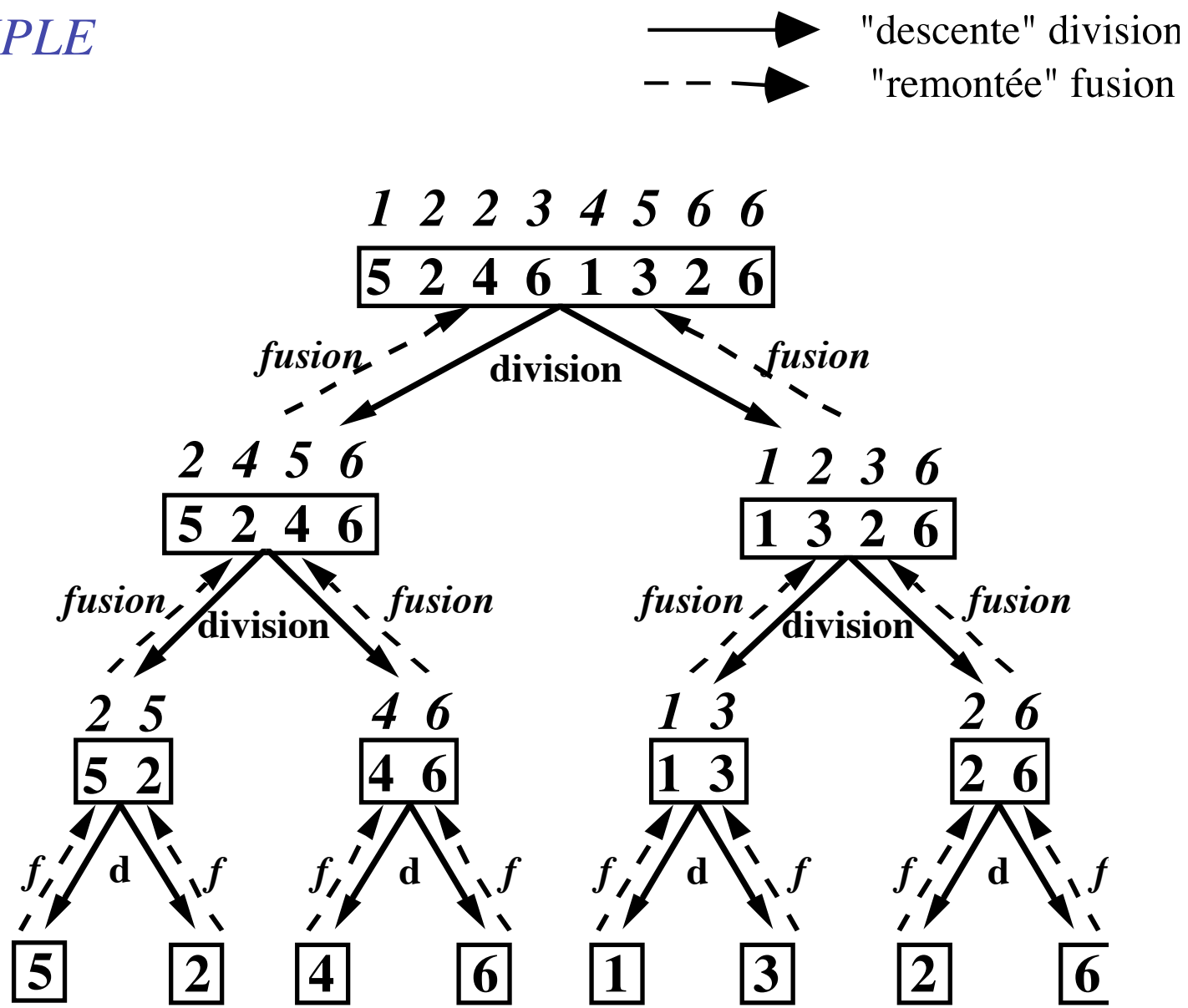
complexité: $O(n_1+n_2)$

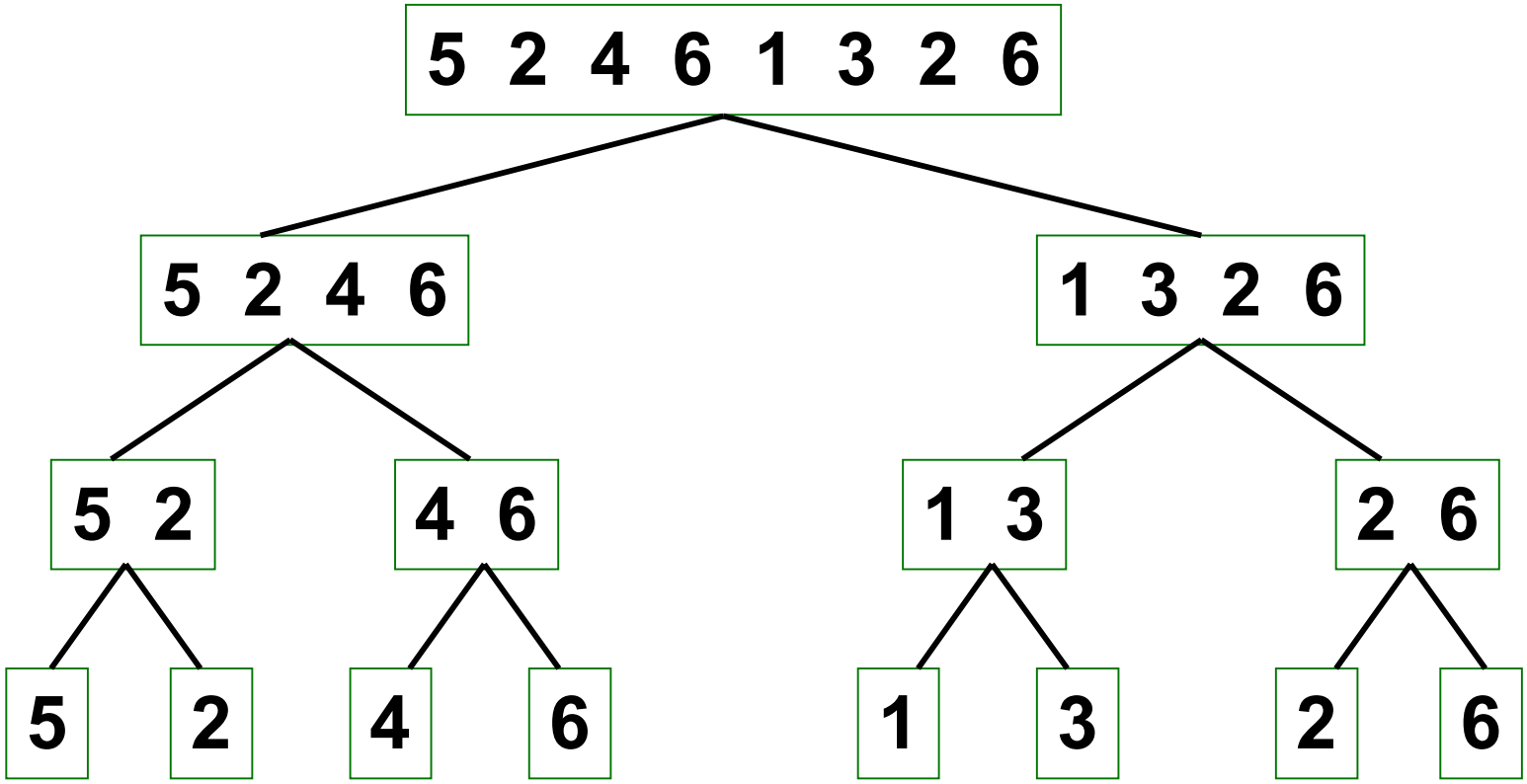
4-2-2 TRI FUSION

procédure récursive:

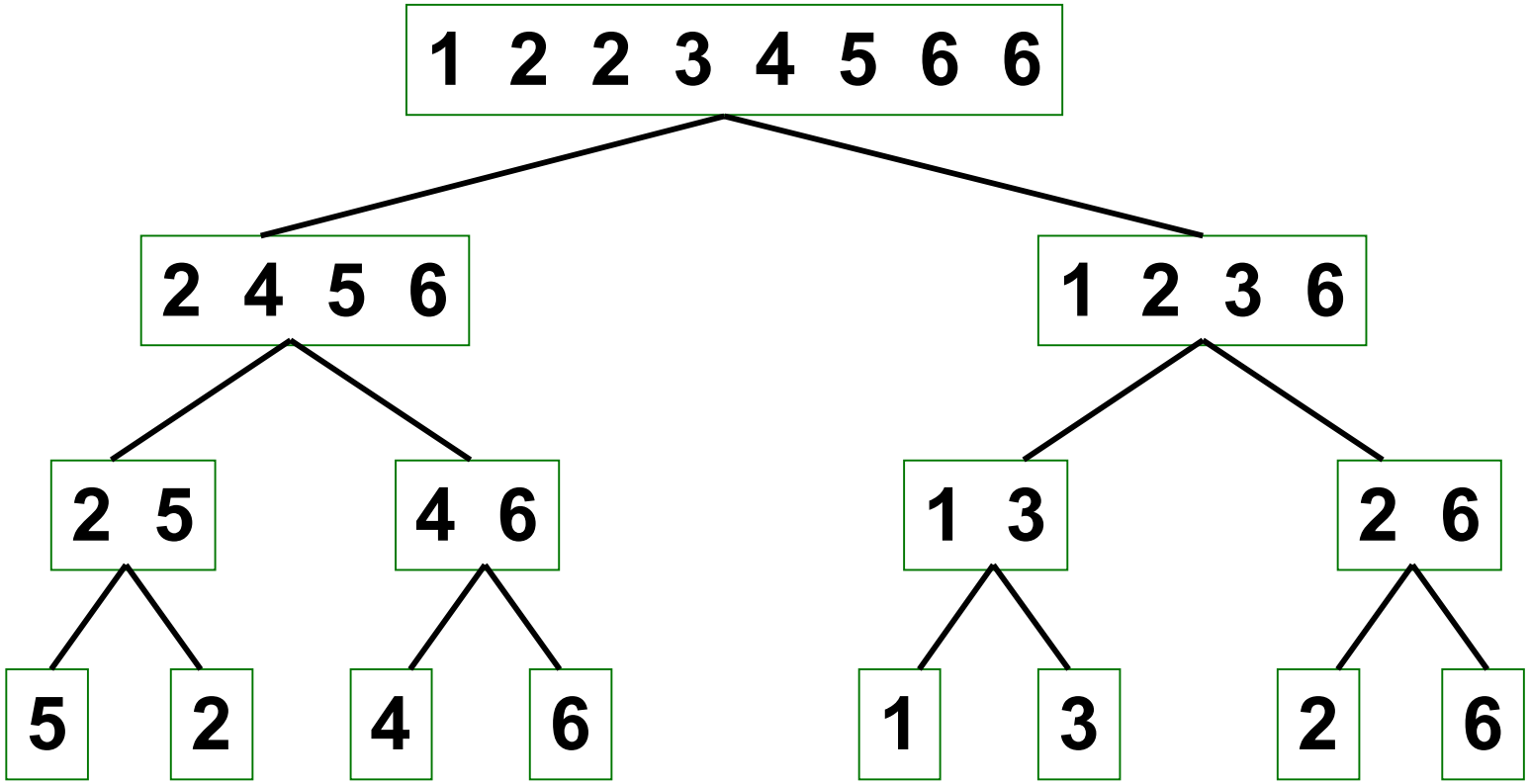
- **diviser la séquence de n éléments en 2 sous-séquences de n/2 éléments** (ou n/2 et n/2+1)
- **trier chaque sous-séquence avec tri-fusion**
- **fusionner les 2 sous-séquences triées**

EXEMPLE

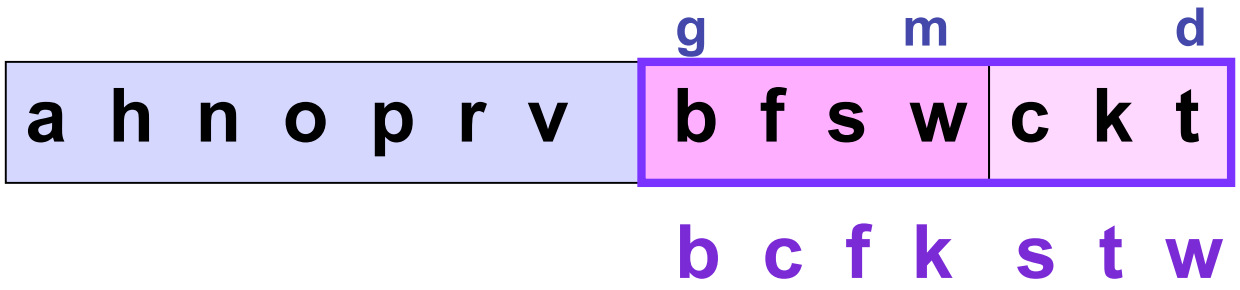

"descente" division
"remontée" fusion
1 2 2 3 4 5 6 6
5 2 4 6 1 3 2 6
fusion
division
fusion
2 4 5 6
5 2 4 6
1 2 3 6
1 3 2 6
fusion
division
fusion
fusion
division
fusion
2 5
5 2
4 6
4 6
1 3
1 3
2 6
2 6
f
d
f
f
d
f
f
d
f
f
d
f
5
2
4
6
1
3
2
6



Division



Fusion



tri "sur place" → *un seul tableau*

void fusionner (int[] t, Indice g, Indice d)

- *à partir de* **fusion(t_1 ,t_2 ,t)**
- m=(g+d)/2
- *t: de g à d* t_1 *: de g à m* t_2*: 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)

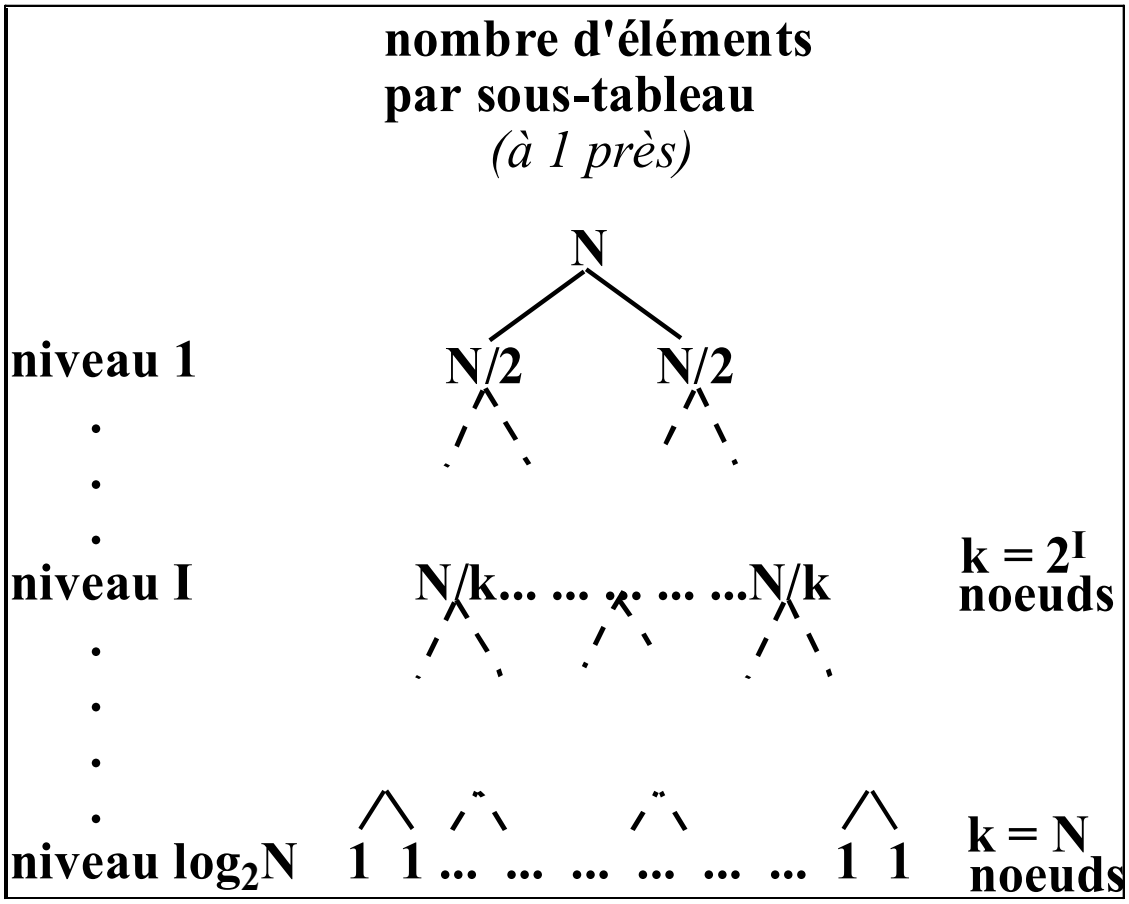
```
void tri-fusion (int[] t , Indice i, Indice j)
//tri-fusion du sous-tableau de t.tab allant de i à j
Indice   d, m, g ;
début
  si   i > j   alors "erreur";
  si   i < j   alors                    //il y a plus d'un élément dans le sous-tableau
          g = i ; d = j ;
          m = i+j / 2 ;  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

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

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**

t(n): O(temps d'exécution pour n)

- "diviser" → O(1)

- "régner" → 2 t(n/2)

- fusionner → O(n)

$$t(n) = \begin{cases} O(1) & \text{si } n = 1 \\ 2\,t(n/2)+O(n) & \text{si } n > 1 \end{cases}$$

par substitutions on obtient:

$$t(n) = O(n \log n)$$

- **autres résultats (pour O ou Θ)**

- t: fonction croissante et t(1)=1

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

$t(n) = t(n/2)+c \quad \Rightarrow \quad t(n) = O(\log n)$

$t(n) = 2t(n/2)+cn \quad \Rightarrow \quad t(n)=O(n\log n)$

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

$t(n) = 4t(n/2)+cn^2 \quad \Rightarrow \quad t(n) = O(n^2\log n)$

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 int[])

t_tas: tas utilisé pour le tri (classe C_Tas)

```
void tri_par_tas (int[] l)
entier val; C_Tas letas= new C_Tas (l.length) ; entier n=l.long;
début
pour k=1 à n faire      //construction du tas associé à l
        val = l [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 [k]= val ;
    //suppression du min en gardant la structure de tas
    letas.supprimer_min();
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)

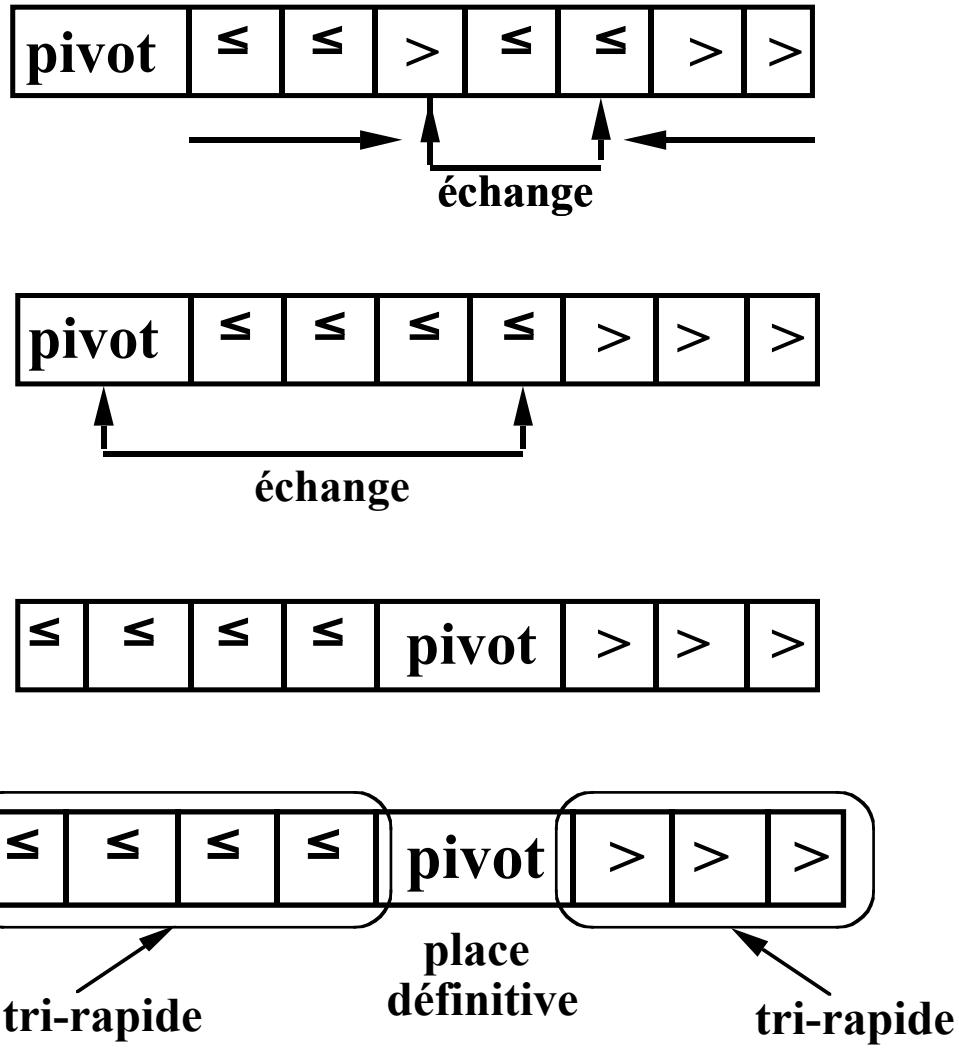
mais, tri "sur place" possible avec programmation de même principe un peu plus complexe

complexité en mémoire O(1)
tri par tas: très bon tri

4-5 LE TRI RAPIDE

(implémentation non étudiée ici)

principe



procédure récursive :

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

  à gauche du pivot, éléments ≤ 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(nlogn)

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

  → $2n^2$ 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 1: 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 **si les n nombres à trier son petits**

(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.

∃ bijection entre E et {1,..,n}

EXEMPLE **alphabet ←→ {1,..,26}**

4-7-2 PROCÉDURE

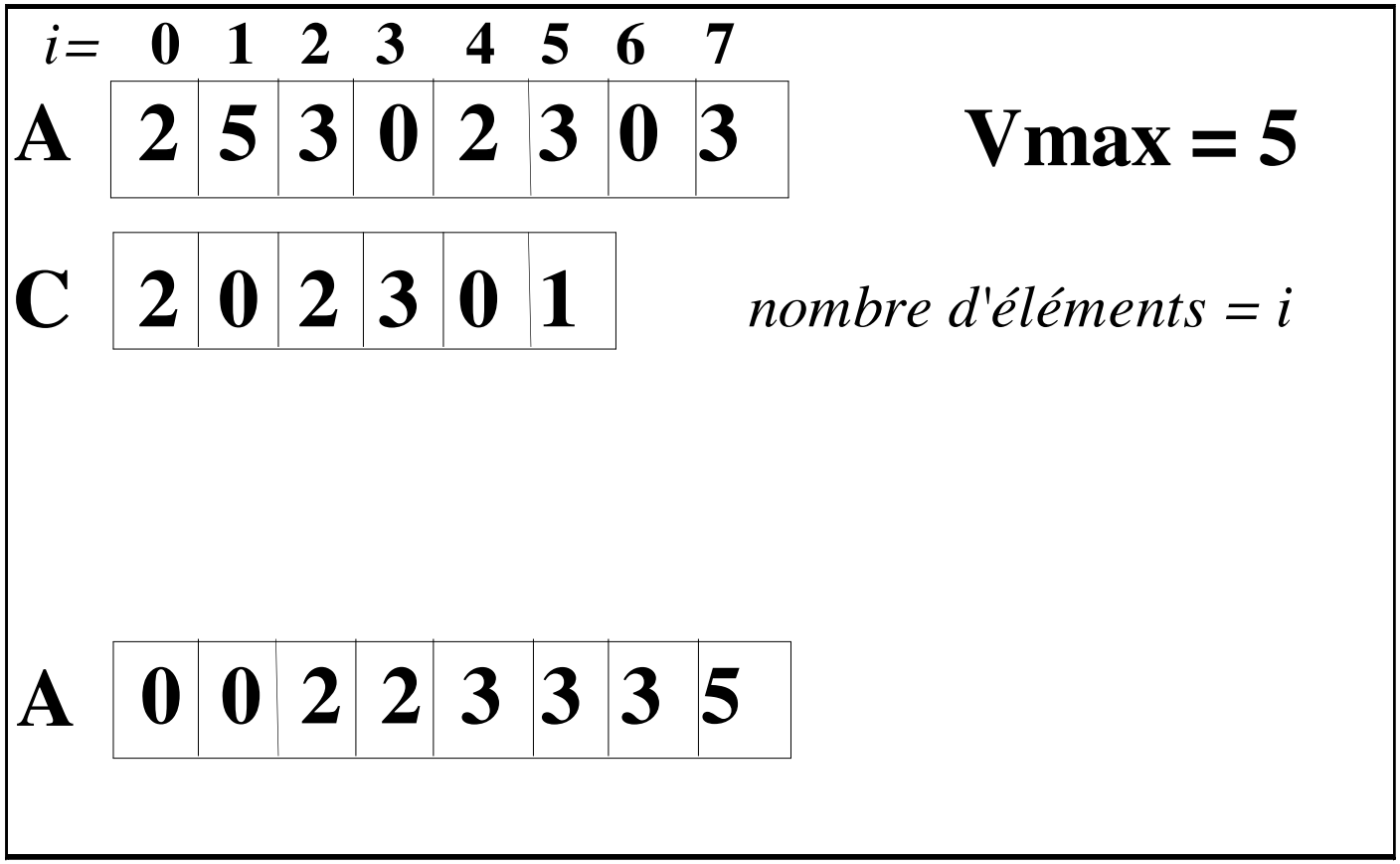
A: tableau de n entier; *à trier*

On suppose que tous les éléments de A sont compris entre 0 et Vmax

```
int[]  tri-dénombrement  (int[] A, int Vmax)
 int[] C = new int[Vmax+1];
début
   pour  i =0  à  Vmax faire
        C[i] = 0;
   fait;
   pour  j =0  a  A.length -1 faire
        C[A[j]] = C[A[j]]+1;
   fait;
   //C[i] contient le nombre d'éléments de A
   égaux à i
```

```
int j=0;
int k;
 pour  i =0  a  Vmax faire
    pour k=1 a C[i] faire
       A[j]= i;
       j=j+1;
     fait;
 fait;
fin
```

EXEMPLE


i= 0 1 2 3 4 5 6 7
A 2 5 3 0 2 3 0 3
Vmax = 5
C 2 0 2 3 0 1
nombre d'éléments = i
A 0 0 2 2 3 3 3 5

complexité du tri par dénombrement

Si Vmax= 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**

Implémentation java de différents tris- Résultats

Tri d'un tableau de n entiers générés aléatoirement dans l'intervalle [0,1000]

n	10^2	10^3	10^4	10^5	10^6	10^7	10^8
Tri bulle	0.044 s	0.068 s	0.492 s	42.319 s	71 min 9 s	_	
Tri fusion	0.052 s	0.068 s	0.168 s	0.156 s	0.46 s	3.428 s	Out of memory
Tri par tas	0.044 s	0.056 s	0.104 s	0.152 s	0.444 s	5.016 s	Out of memory
Tri rapide	0.048 s	0.060 s	0.080 s	0.440 s	1.29 s	1min44 s	Out of memory
Tri dénomb.	0.060 s	0.052 s	0.064 s	0.144 s	0.17 s	0.72 s	Out of memory