NFP 136 – ALGORITHMES DE TRI

- **Le problème du tri**
- **Le tri par insertion**
- **Le tri fusion**
- **Le tri par tas**
- **Le tri rapide**

LE PROBLEME DU TRI : RAPPELS

Éléments à trier par ordre (dé)croissant

- SOLDES
- BONS PLANS
- VENTES FLASH
- OFFRES ADHÉRENTS
- OFFRES REMBOURSEMENT
- PACKS MICRO

TOUTES NOS OFFRES

- **TABLETTE TACTILE**
- TOUTES LES TABLETTES
- KOBO BY
- MICROSOFT SURFACE
- IPAD
- SAMSUNG GALAXY
- TABLETTE ACER
- TABLETTE ASUS, NEXUS
- ACCESSOIRE TABLETTE
- **ORDINATEUR**
- **ORDINATEUR PORTABLE**
- Espace

Résultats : **1** - **10** sur un total de **42**

1 - 2 - 3 - 4 - 5 › »

	Modèle ▲▼	Ecran ▲▼	Processeur ▲▼	Système d'exploitation ▲▼	Note : ▲▼	Expédié ▲▼	Prix ▲▼
BON PLAN	Tablette Microsoft Surface Pro 3 12" 256 Go - Intel Core i5	12 "	Intel Core i5*	Windows 8 Pro**	4,5/5	En Stock / En stock	BON PLAN 1229,90€ ~~1299,90€~~ » Ajouter au panier » 6 neufs à partir de 1229,90€
BON PLAN	Tablette Microsoft Surface Pro 3 12" 128 Go	12 "	Intel Core i5*	Windows 8 Pro**	4,5/5	En Stock / En stock	BON PLAN 929,90€ ~~999,90€~~ » Ajouter au panier » 6 neufs à partir de 929,90€
SOLDE	Pack Tablette Asus T100TA-DK090H 10.1" Tactile, 1 To + Housse + Microsoft Office 2013	10,1 "	Intel Quad-Core Atom Bay Trail-T	Windows 8 32 Bits**	4/5	En Stock	SOLDE 399,98€ ~~499,90€~~ » Ajouter au panier
BON PLAN	Tablette Microsoft Surface Pro 3 12" 64 Go	12 "	Intel Core i3*	Windows 8 Pro**	3,5/5	En Stock / En stock	BON PLAN 749,90€ ~~799,90€~~ » Ajouter au panier » 5 neufs à partir de 749,90€

INTERET DU TRI ?

- **Problème simple à énoncer, intuitif et courant : étant données n valeurs, les classer par ordre (dé)croissant**

- **Nombreux algorithmes avec structures de données variées**

- **Sous-problème usuel de problèmes plus complexes (répartition de tâches, etc.)**

IMPLÉMENTATION

==> Quel algorithme ? Non précisé dans la spécification !

Rappel : **tri par sélection** (cf Complexité) en **O(n^2)**
(Il existe une variante appelée **tri à bulles**, en **O(n^2)**.)

On va en étudier d'autres, dont certains sont plus efficaces !

Un premier exemple, dit « tri par insertion » :

- Principe : éléments mis 1 par 1 « directement » à leur place,
- Deux variantes : insertion séquentielle ou dichotomique.

TRI PAR INSERTION SÉQUENTIELLE

EXEMPLE (en vert : éléments triés)

7 15 8 10 5 17

7 15 8 10 5 17 //échange de 8 et 15

7 8 15 10 5 17 //échange de 10 et 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

Implémentation (en JAVA)

```
void  triInsertionSequentielle(int[] tab) {
        for(int j = 1 ; j <= tab.length-1 ; j++) {
                int cle = tab[j];
                int i = j-1;
                while ((i >= 0) && (tab[i] > cle)) {
                        //décalage vers la droite
                        tab[i+1] = tab[i];
                        i = i-1;
                }
                //clé mise définitivement à sa place
                tab[i+1]=cle;
        }
}
```

Complexité du tri par insertion

- **Pire des cas**
  - **boucle pour : j de 1 à n-1 (= tab.length-1)**
  - **itération j, boucle tant que : au pire j fois**

    ==> O(1 + 2 + … + n-1) opérations

D'où : insertion séquentielle en $O(n^2)$

Complexité du tri par insertion

- **En moyenne (avec équiprobabilité)**
  - boucle pour : j de 1 à n-1 (= tab.length-1)
  - itération j, boucle tant que : au pire j/2 fois

    ==> O((1 + 2 + … + n-1)/2) opérations

D'où : insertion séquentielle en $O(n^2)$

INSERTION DICHOTOMIQUE

Idée : rechercher par dichotomie la case où insérer l'élément j

(dans la première partie <u>triée</u> de la liste)

(cf chapitre Complexité)

A chaque itération :

recherche de la place en O(log n)

mais décalages en O(n)

==> gain potentiel en pratique, mais complexités au pire cas et en moyenne restent en $O(n^2)$

COMPLEXITÉ EN MÉMOIRE

Tris par insertion et sélection :

tris « sur place »

(pratiquement aucune recopie de données)

==> complexité mémoire en O(1)

TRI FUSION

OPERATION DE BASE : LA FUSION

Fusion de 2 tableaux triés :

- Sélection et retrait du plus petit des 2 premiers éléments de chaque tableaux,

- Jusqu'à avoir parcouru les 2 tableaux en entier.

UN EXEMPLE

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	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	

EXEMPLE (SUITE)

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	

En pratique, T1 et T2 sont des « morceaux » de T, chacun compris entre deux indices : les paramètres nécessaires sont donc T, et les indices délimitant T1 et T2 dans T

```
procédure fusion(tableau d'entiers tab, entier g, entier d)
/* fusionne les deux parties de tab comprises entre g et (g+d)/2 et entre (g+d)/2+1 et
    d, qui elles sont supposées déjà triées */
Tableau[d-g+1] d'entiers : temp; entiers : i=0, j=1+(d-g)/2, k=g, l;
début
pour l=0 à d-g faire temp[l]=tab[l+g]; fait;
tant que i+g ≤ (g+d)/2 et j+g ≤ d faire //tant qu'une des 2 parties n'est pas vide
    si temp[i] ≤ temp[j] alors //tab[k] doit valoir le plus petit entier des 2 parties
                    tab[k] := temp[i];
                    i := i+1;
    sinon
                    tab[k] := temp[j];
                    j := j+1;
    finsi
    k := k+1;
fait;
```

```
tant que i+g ≤ (g+d)/2 faire //ajouter le reste de la partie gauche
        tab[k] := temp[i];
        i := i+1;
        k := k+1;
fait;
tant que j+g ≤ d faire //ajouter les éléments restants de la partie droite
        tab[k] := temp[j];
        j := j+1;
        k := k+1;
fait;
fin
```

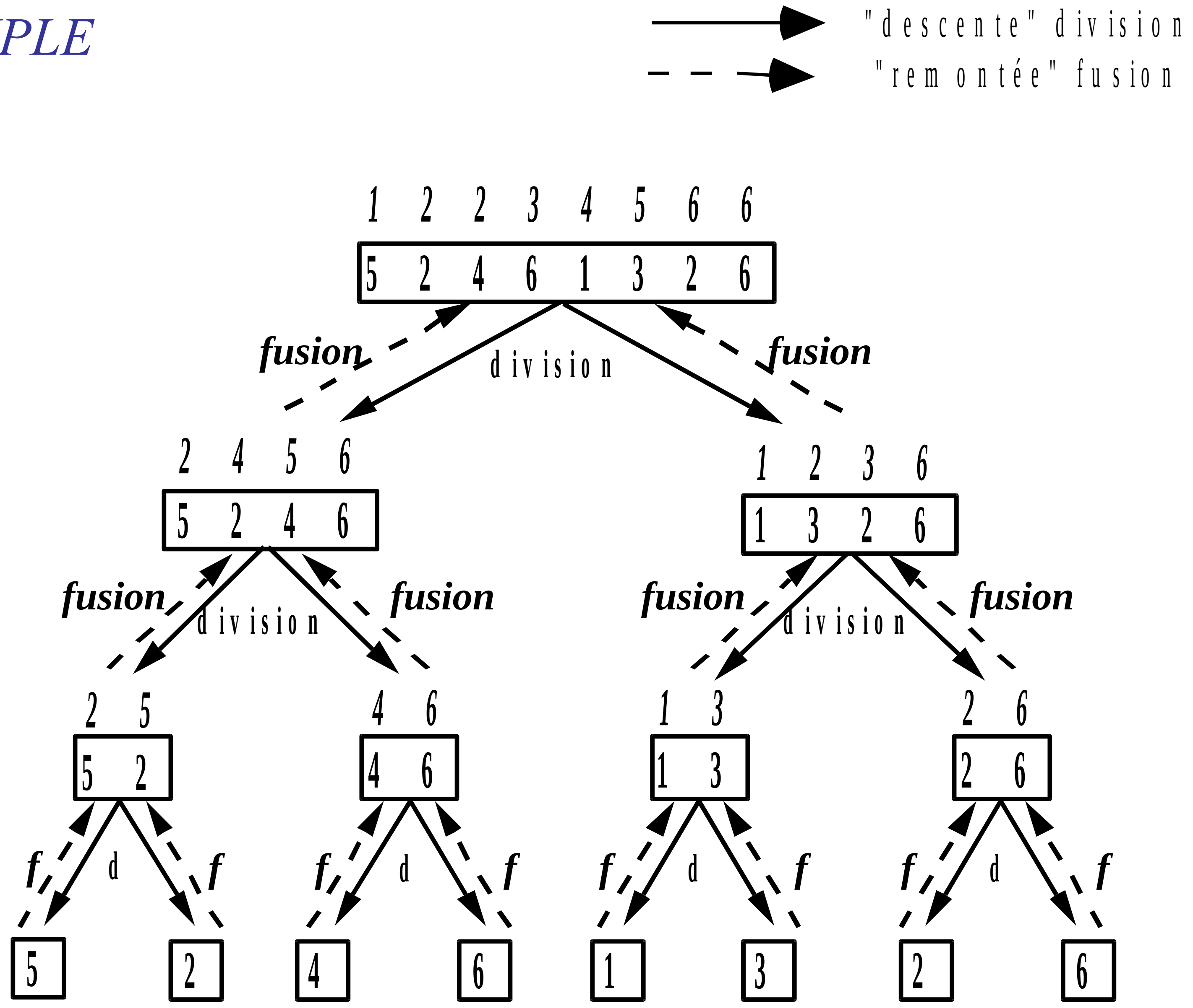
<u>Complexité en temps et en mémoire = O(d-g)</u>

TRI FUSION

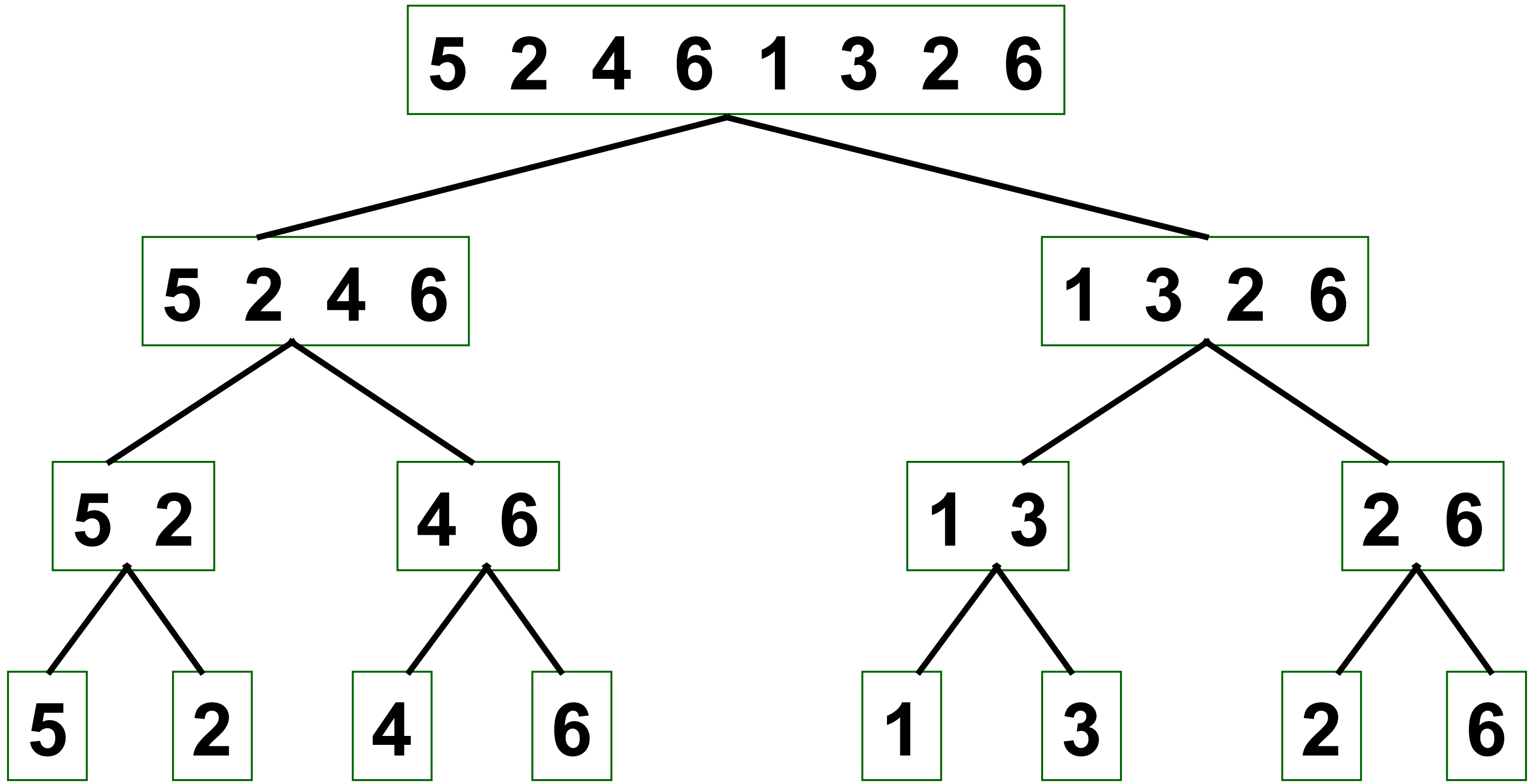
Procédure récursive :

- **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 triFusion,**
- **fusionner les 2 sous-séquences triées (avec la procédure fusion).**

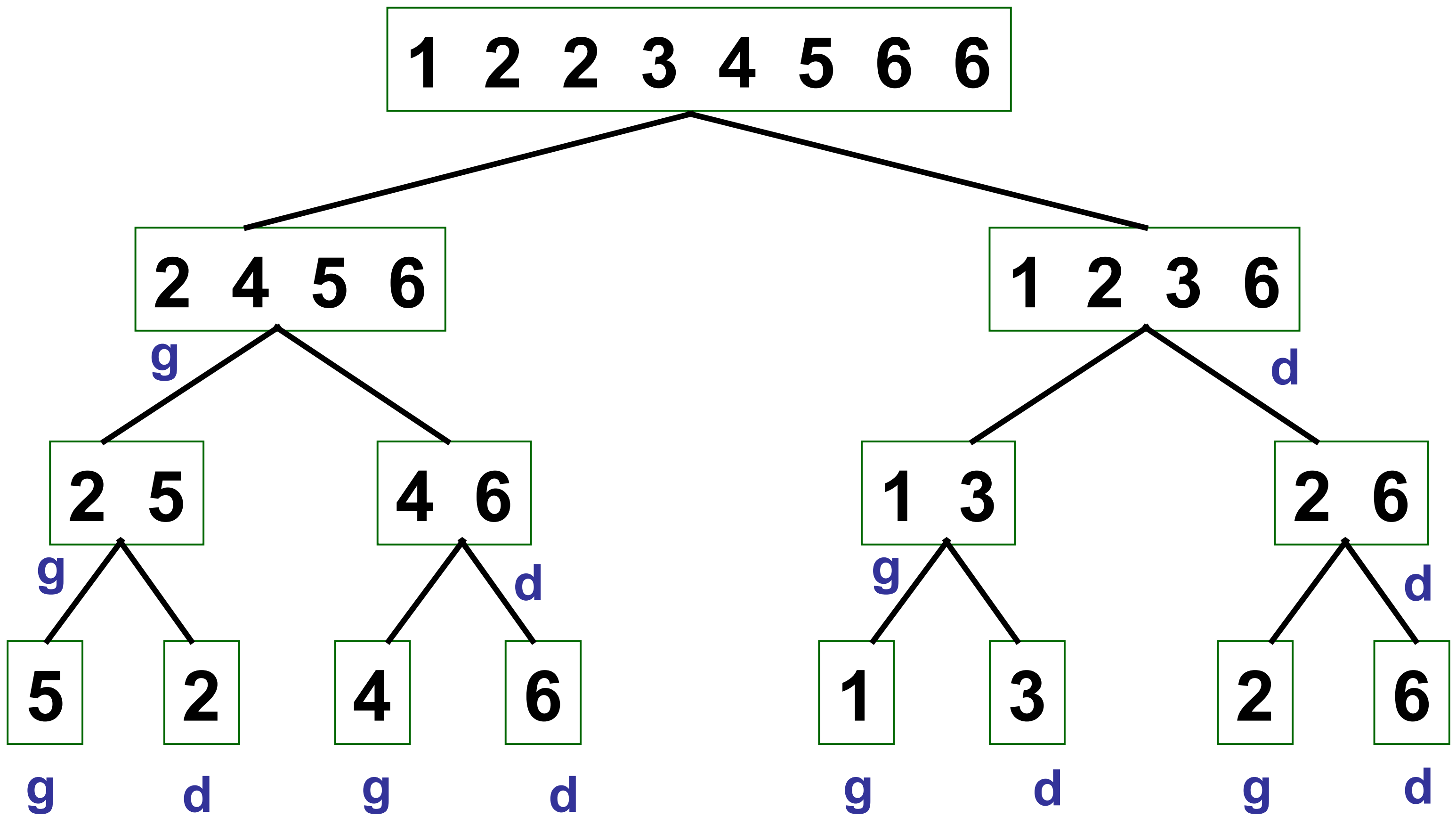
EXEMPLE



"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
f
d
f
f
d
f
f
d
f
5
2
4
6
1
3
2
6



Division (haut en bas)



Fusion (bas en haut)

```
void  triFusion(tableau d'entiers t, entier gauche, entier droite)
//triFusion du sous-tableau de t compris entre les indices gauche et droite
début
  si   gauche > droite   alors "erreur";
  si   gauche < droite   alors //il y a plus d'un élément dans le sous-tableau
         triFusion(t, gauche, (gauche+droite)/2); //tri partie gauche
         triFusion(t, (gauche+droite)/2+1, droite); //tri partie droite
         fusion(t, gauche, droite); //on fusionne les 2 parties
  fin //si i = j  il n'y a plus qu'un élément ==> STOP
fin;
```

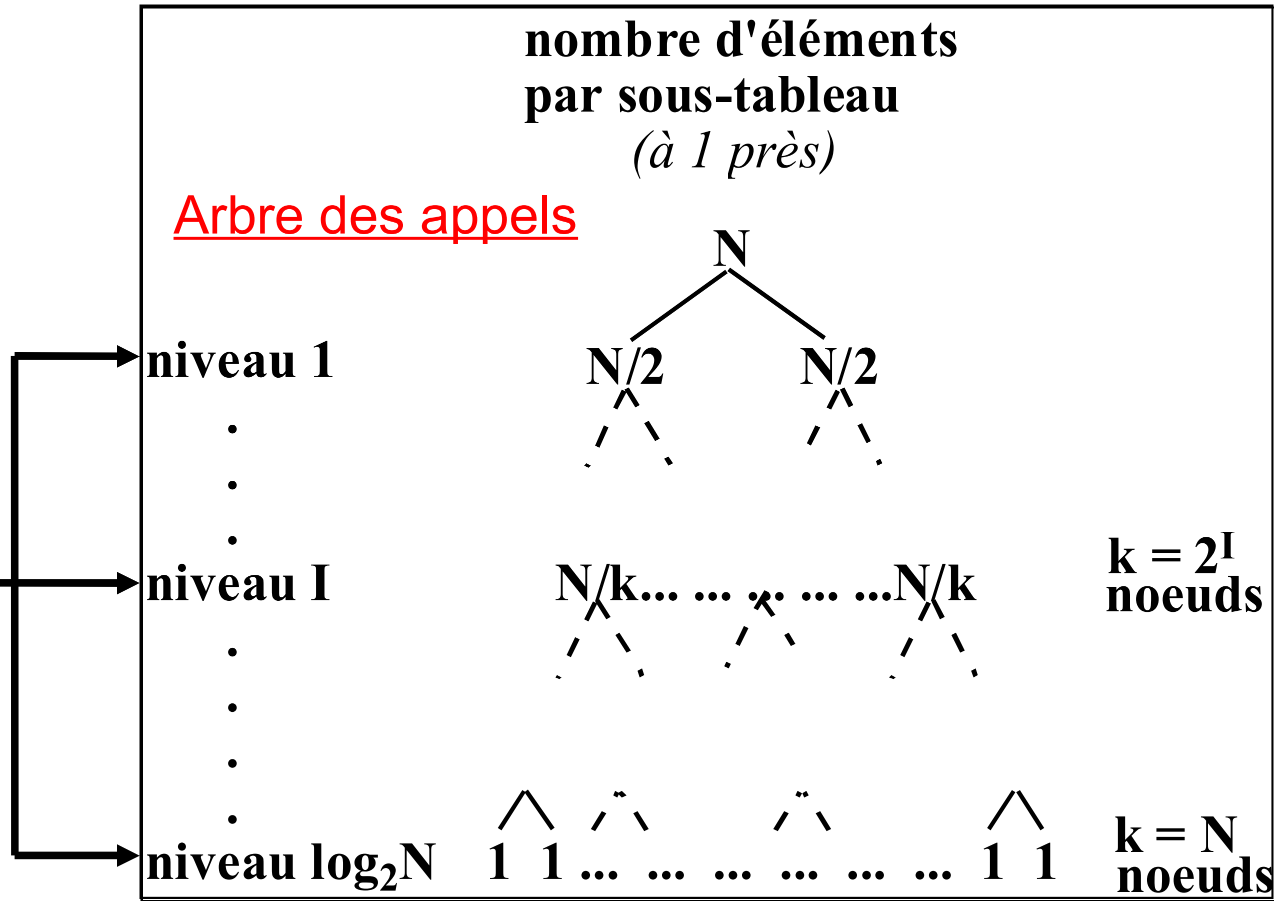
==> ***appel initial :*** **triFusion(t, 0, t.length-1);**

==> ***complexité en mémoire ?*** **Un seul appel à** ***fusion*** **exécuté en même temps, donc en O(n)**

COMPLEXITÉ ET RÉCURRENCE

a) complexité en temps au pire cas du tri fusion ?

O(k*(N/k))=O(N)
opérations, pour tout k



==> complexité en O(n log n) << O(n^2), donc <u>tri fusion = très bon tri</u> !

b) Plus généralement : complexité des algorithmes récursifs de type « diviser pour régner »

<u>**Complexité vérifiant certaines relations de récurrence**</u> **:**

t(n) = Temps d'exécution pour une donnée de taille n

- « diviser » $\rightarrow$ O(1)

- « régner » $\rightarrow$ 2 t(n/2)

- « fusionner » $\rightarrow$ O(n)

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

<u>**On peut alors montrer que**</u> **: t(n) = O(n log n) (*par récurrence sur n*)**

- **Autres résultats :**

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

- n>1, c = constante

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

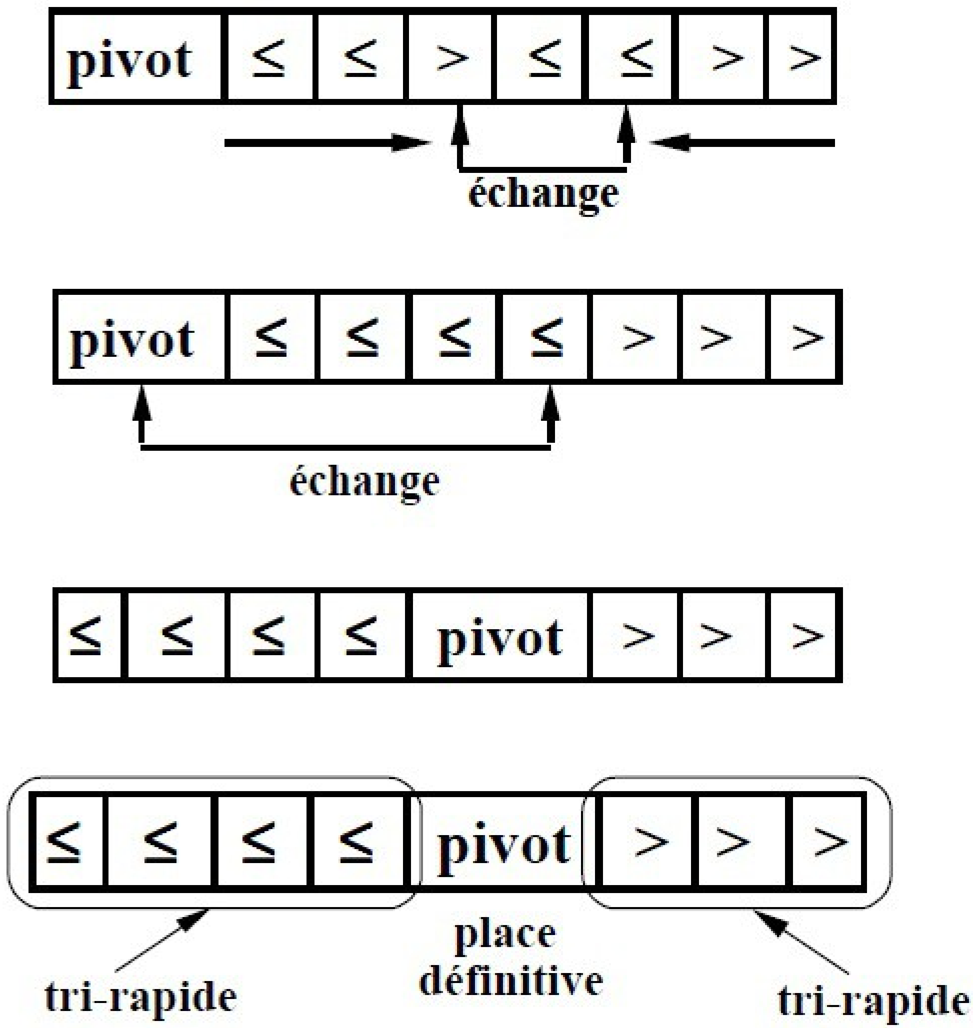
$t(n) = 2t(n/2)+cn \quad \Rightarrow \quad t(n) = O(n \log n)$

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

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

TRI RAPIDE
(« QUICKSORT »)

principe


pivot ≤ ≤ > ≤ ≤ > >
échange
pivot ≤ ≤ ≤ ≤ > > >
échange
≤ ≤ ≤ ≤ pivot > > >
≤ ≤ ≤ ≤ pivot > > >
tri-rapide
place
définitive
tri-rapide

Procédure récursive :

- **Sélectionner un élément pivot p,**

- **Partitionner les éléments à trier en 2 parties :**
  - **à gauche du pivot, éléments ≤ p,**
  - **à droite du pivot, éléments > p,**

- **Tri rapide des 2 parties (appel récursif), qui sont ensuite concaténées ensemble.**

```
entier partitionner(tableau d'entiers  tab, entier gauche, entier droite)
début /* ici, on choisit tab[droite] comme pivot */
entier : i, temp, ppg:=gauche; /* ppg=indice du 1er élément plus grand que le pivot */
pour i = gauche à droite-1 faire
        si (tab[i]<=tab[droite]) alors
                temp:=tab[ppg]; /* on échange tab[i] et tab[ppg]  */
                tab[ppg]:=tab[i];
                tab[i]:=temp;
                ppg:=ppg+1; /* on met à jour ppg */
        finsi
fait;
temp:=tab[droite]; /* on échange tab[ppg] avec le pivot = tab[droite]  */
tab[droite]:=tab[ppg];
tab[ppg]:=temp;
retourner ppg; /* on renvoie l'indice du pivot après calcul de la partition */
fin
```

```
void triRapide(tableau d'entiers tabATrier, entier gauche, entier droite)
début
entier : pivot;
si (debut < fin) alors
            pivot=partitionner(tabATrier, gauche, droite);
            triRapide(tabATrier, gauche, pivot-1);
            triRapide(tabATrier, pivot+1, droite);
finsi
fin
```

==> Appel initial ? triRapide(tabATrier, 0, tabATrier.length-1);

BILAN DU TRI RAPIDE

1) Choix du pivot :

- Inutile si trop coûteux en temps (médiane),

- Par exemple, choix aléatoire ou choix de la médiane entre le 1er élément, l'élément du milieu, et le dernier.

2) Espace mémoire ? tri « sur place », donc O(1)

3) Complexité en temps du tri rapide :

- Pire cas ? $O(n^2)$

- En moyenne ? $O(n \log n)$

==> <u>Tri rapide : très bon tri en général !</u>

TRI PAR TAS

Rappels sur les tas

Un tas est (cf chapitre 5 du cours) :

- **un arbre parfait,**
- **tel que tout nœud a une valeur ≤ à celle de tous ses descendants.**

Méthodes associées :

estVide, minimum, insérer, supprimerMin

Principe du tri par tas :

- **Transformer le tableau à trier en tas,**
- **Extraire un à un les éléments min (racines) en conservant la structure de tas, et les stocker dans cet ordre dans le tableau trié.**

```
void triParTas (tableau d'entiers tabATrier)
entier n=tabATrier.length; Tas unTas = new Tas(n);
début
pour i = 0 à n-1 faire  //construction du tas associé à tabATrier
        unTas.inserer(tabATrier[i]); //O(log n) pour chaque i
fait;
pour i=0 à n-1 faire
   //on sélectionne un par un les minimums successifs du tas, qui sont
   mis à leur place dans le tableau
   tabATrier[i]=unTas.minimum(); //O(1) pour chaque i
   //suppression du minimum en gardant la structure de tas
   unTas.supprimerMin(); //O(log n) pour chaque i
fait;
fin
```

COMPLEXITÉ ET BILAN

- 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 un peu plus complexe, mais suivant le même principe.)

==> Complexité en mémoire : O(1)

<u>LE TRI PAR TAS EST DONC UN TRES BON TRI !</u>

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

<u>On peut montrer que :</u>

O(n log(n)) = « limite » non améliorable

pour tris par comparaisons !

Tri par dénombrement

- **Cette limite de O(n log n) peut être battue par un autre type de tri : le tri par dénombrement**
- **Aucune comparaison, mais un tel tri n'est efficace que si les n valeurs à trier sont « petites » (= O(n))**
- **Dans ce cas, le tri par dénombrement a une complexité en O(n) (*cf ED*) : meilleure complexité possible pour trier n nombres !**

Conclusion

- **Tri par tas et tri fusion : meilleure complexité**
- **Tri rapide : très efficace en pratique**
- **Tri par insertion excellent si liste initiale presque triée, et peut débuter sans liste initiale complète**
- **Tri par sélection donne le début de liste trié avant la fin du tri**

De plus : on peut « hybrider » les tris (ex. : tri fusion qui utilise tri par insertion en-dessous d'une certaine taille), pour plus d'efficacité !