Algorithmes de tri interne

Clés comparables

On a un fichier d'éléments avec clés comparables
— on veut les ranger selon l'ordre les clés

Clés comparables en Java :

```
public interface Comparable<T>
{
	int compareTo(T autre_objet);
}

public interface Comparator<T>
{
	int compare(T un_objet, T autre_objet);
}
```

`x.compareTo(y)` est négatif si x précède y, ou positif si x suit y dans l'ordre «naturel» des éléments

Tri interne

Tri interne : tout le fichier est en mémoire
(représenté par un tableau ou une liste chaînée)

Tri externe : fichier stocké partiellement ou entièrement en mémoire externe (disque)
— accès à mémoire externe est couteux. . .

Entrée : tableau `A[]` de données — on veut le trier

Solutions :

* tri par sélection (*selection sort*)
* tri par insertion (*insertion sort*)
* tri par fusion (*Mergesort*)
* tri par tas (*Heapsort*)
* tri rapide (*Quicksort*)

Tri par sélection

Algo TRI-SELECTION$(A[0..n-1])$

S1 **for** $i \leftarrow 0, 1, \ldots, n-2$ **do**

S2   minidx $\leftarrow i$

S3   **for** $j \leftarrow i+1, \ldots, n-1$ **do**

S4   **if** $A[j] < A[\mathsf{minidx}]$ **then** minidx $\leftarrow j$

    // *(maintenant* $A[\mathsf{minidx}] = \min\{A[i], \ldots, A[n]\}$*)*

S5   **if** $i \neq \mathsf{minidx}$ **then** échanger $A[i] \leftrightarrow A[\mathsf{minidx}]$

Complexité :

⋆ comparaison d'éléments $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$ fois ;

⋆ échange d'éléments [ligne S5] $\leq (n-1)$ fois

Temps de calcul : toujours $\Theta(n^2)$

mais pas une mauvaise idée si l'échange est beaucoup plus cher que la comparaison

Tri par insertion

```
    Algo TRI-INSERTION(A[0..n − 1])
I1  for i ← 1, . . . , n − 1 do
I2      j ← i − 1
I3      while j ≥ 0 et A[j] < A[i] do
I4          échanger A[i] ↔ A[j]
I5          j ← j − 1
```

Complexité — dépend de l'ordre des éléments au début

meilleur cas (déjà trié) : $n-1$ comparaisons et aucun échange

pire cas (trié en ordre décroissant) : $\frac{n(n-1)}{2}$ comparaisons et échanges

moyen cas : $\Theta(n^2)$

☞ très utile si A est «presque trié» au début

génie algorithmique : min en `A[0]` (sentinelle) — pas de test $j \geq 0$ en I3

remplacer échange par décalage en I4

Tri par insertion

```
public void tri(double[] T)
{
    for (int i=1; i<T.length; i++)
    {
        // T[0]<=T[1]<=...<=T[i-1]
        double x = T[i];
        int j=i;
        while (j>0 && T[j-1]>x) { T[j]=T[j-1]; --j; } // décalage
        T[j] = x;
    }
}
```

avec sentinelle à T[0]

```
    int m=0; for (int i=1; i<T.length; i++) if (T[i]<T[m]) m=i;
    double v = T[0]; T[0]=T[m]; T[m]=v; // minimum dans T[0]
    for (int i=1; i<T.length; i++)
    {
        double x = T[i];
        int j=i-1;
        while (T[j]>x) { T[j+1]=T[j]; --j; } // décalage
        T[j+1] = x;
    }
```

Tri par tas

```
    HEAPSORT(A) // tableau non-trié A[1..n]
H1  heapify(A)
H2  for i ← |A|, ... 2 do
H3      échanger A[1] ↔ A[i]
H4      sink(A[1], 1, A, i − 1)
```

$A[1..n]$ est dans l'ordre décroissant à la fin

(pour l'ordre croissant, utiliser un max-tas)

Temps $O(n \log n)$ au pire, sans espace additionnelle !

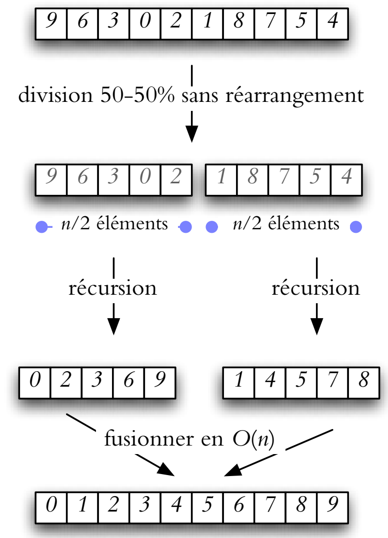
quicksort : $\Theta(n^2)$ au pire (mais très rarement)
mergesort : $O(n \log n)$ toujours mais utilise un espace auxiliaire de taille n

Fusion de deux tableaux

Algo FUSION$(A[0..n-1], B[0..m-1])$ *// A, B : tableaux triés*
initialiser $C[0..n+m-1]$ *// on place le résultat dans C*
$i \leftarrow 0; j \leftarrow 0; k \leftarrow 0$ *// i est l'indice dans A ; j est l'indice dans B*
while $i < n$ && $j < m$ **do**
  if $A[i] \leq B[j]$ **then** $C[k] \leftarrow A[i]; i \leftarrow i+1$
  else $C[k] \leftarrow B[j]; j \leftarrow j+1$
  $k \leftarrow k+1$
while $i < n$ **do** $C[k] \leftarrow A[i]; i \leftarrow i+1; k \leftarrow k+1$
while $j < m$ **do** $C[k] \leftarrow B[j]; j \leftarrow j+1; k \leftarrow k+1$

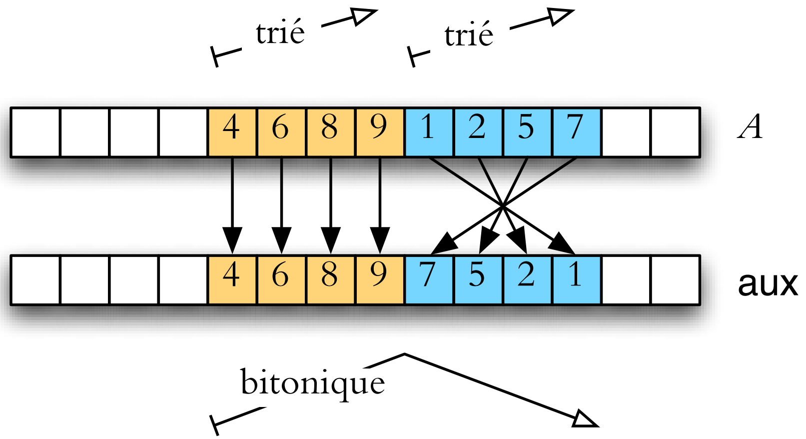
Temps de calcul : $(n+m)$ affectations, et $(n+m-1)$ comparaisons (au pire)

Tri par fusion



	Algo MERGESORT$(A[0..n-1], g, d)$	// *appel initiel avec* $g = 0, d = n$
	// *récursion pour trier le sous-tableau* $A[g..d-1]$	
M1	**if** $d - g < 2$ **then return**	// *cas de base : tableau vide ou un seul élément*
M2	$m \leftarrow \lfloor (d+g)/2 \rfloor$	// m *est au milieu*
M3	MERGESORT(A, g, m)	// *trier partie gauche*
M4	MERGESORT(A, m, d)	// *trier partie droite*
M5	FUSION(A, g, m, d)	// *fusion des résultats*

Fusion — arrangement bitonique



Algo FUSION$(A[\,], g, m, d)$ *// fusion «en place» pour* $A[g..m-1]$ *et* $A[m..d-1]$

F1 **for** $i \leftarrow g, g+1, \ldots, m-1$ **do** $\mathsf{aux}[i] \leftarrow A[i]$

F2 **for** $j \leftarrow m, m+1, \ldots, d-1$ **do** $\mathsf{aux}[m+d-1-j] \leftarrow A[j]$

F3 $i \leftarrow g$; $j \leftarrow d-1$; $k \leftarrow g$

F4 **while** $k < d$ **do** *// meilleure condition :* $i < m$

F5   **if** $\mathsf{aux}[i] \le \mathsf{aux}[j]$ **then** $A[k] \leftarrow \mathsf{aux}[i]$; $i \leftarrow i+1$; $k \leftarrow k+1$

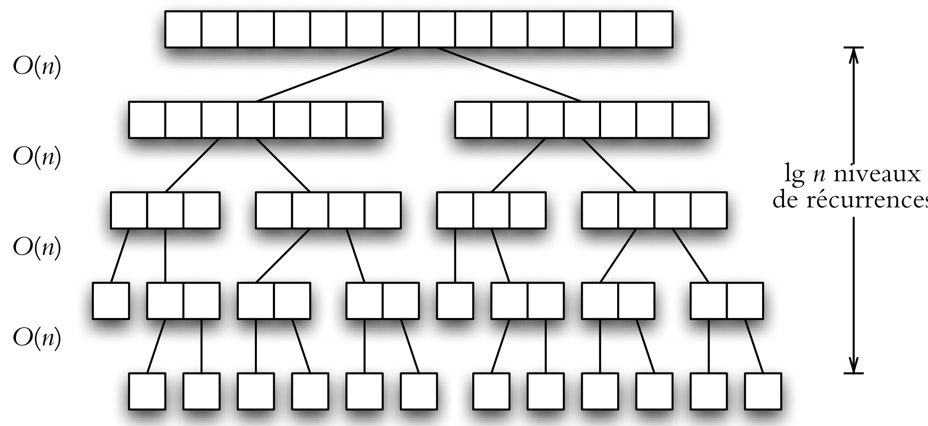
F6   **else** $A[k] \leftarrow \mathsf{aux}[j]$; $j \leftarrow j-1$; $k \leftarrow k+1$

Mergesort — temps de calcul

Nombre de comparaisons $C(n)$ ou temps de calcul $T(n)$:

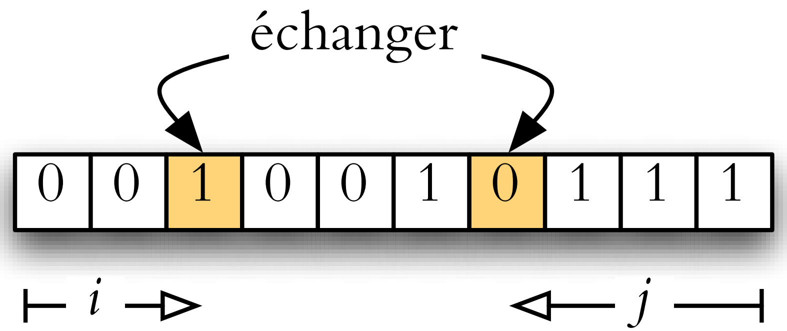
$$C(n) \leq C\Big(\lfloor n/2 \rfloor\Big) + C\Big(\lceil n/2 \rceil\Big) + n - 1$$
$$T(n) = T\Big(\lfloor n/2 \rfloor\Big) + T\Big(\lceil n/2 \rceil\Big) + \Theta(n)$$

solution : $C(n) \sim n \lg n$, $T(n) = \Theta(n \log n)$



Tri binaire

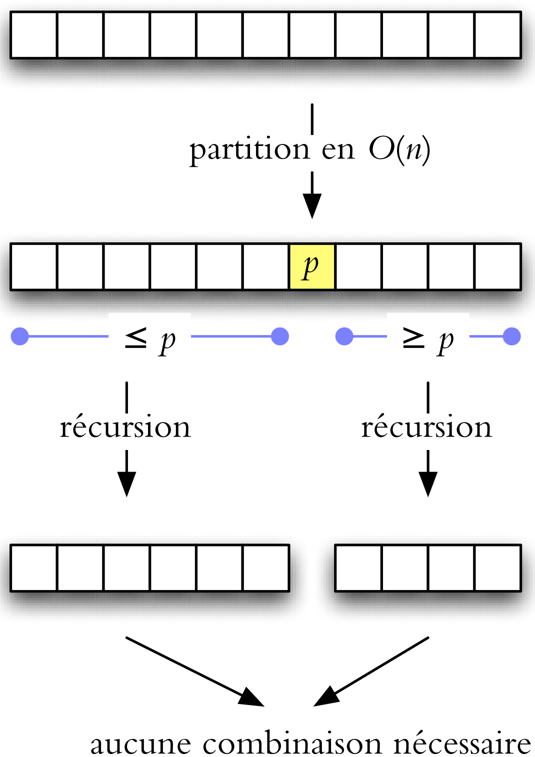
Clés : 0 ou 1



	TRI01$(A[0..n-1])$ *// tri binaire*
B1	$i \leftarrow 0\,;\, j \leftarrow n-1$
B2	**loop**
B3	**while** $i < j$ && $A[i] = 0$ **do** $i \leftarrow i+1$
B4	**while** $i < j$ && $A[j] = 1$ **do** $j \leftarrow j-1$
B5	**if** $i < j$ **then** échanger $A[i] \leftrightarrow A[j]$
B6	**else return**

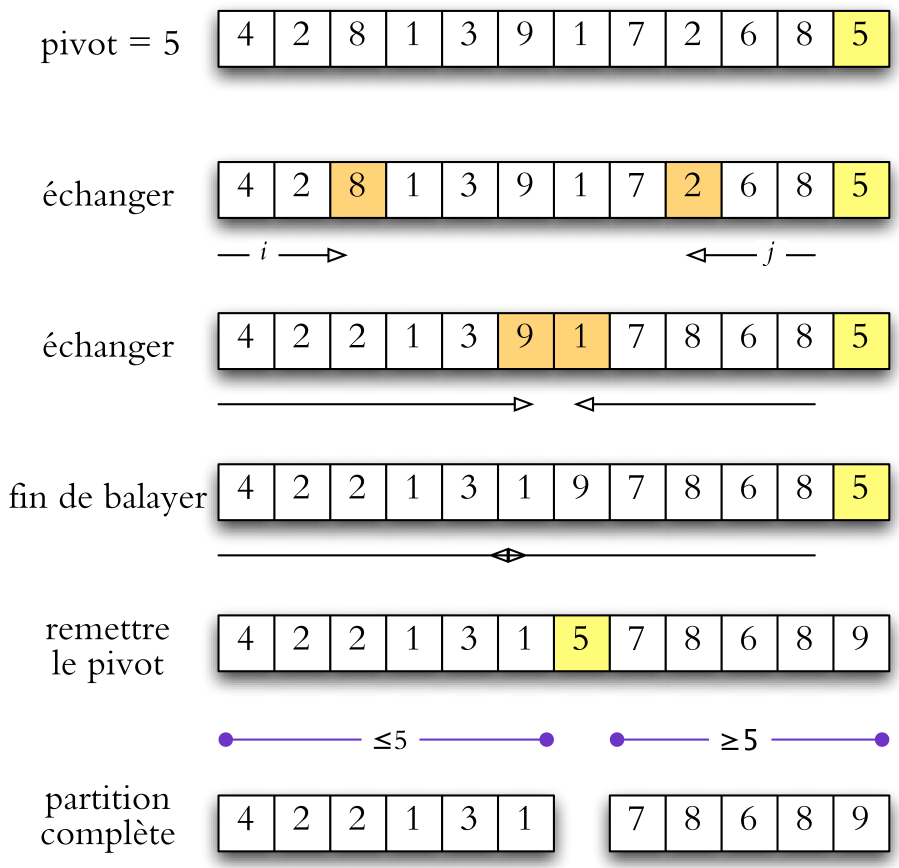
Tri rapide

1. choisir un élément (pivot), le placer dans sa case finale i, mettre tous les éléments inférieurs en $A[0..i-1]$ et tous les éléments supérieurs en $A[i+1..n-1]$



2. trier $A[0..i-1]$ et $A[i+1..n-1]$ (récurrence)

Partition



pivot = 5
4 2 8 1 3 9 1 7 2 6 8 5
échanger
4 2 8 1 3 9 1 7 2 6 8 5
i
j
échanger
4 2 2 1 3 9 1 7 8 6 8 5
fin de balayer
4 2 2 1 3 1 9 7 8 6 8 5
remettre le pivot
4 2 2 1 3 1 5 7 8 6 8 9
≤5
≥5
partition complète
4 2 2 1 3 1
7 8 6 8 9

Quicksort

Algo Quicksort$(A[0..n-1], g, d)$ // *tri de* $A[g..d-1]$
Q1 **if** $d - g \le 1$ **then return** // *cas de base*
Q2 $i \leftarrow$ Partition(A, g, d)
Q3 Quicksort(A, g, i)
Q4 Quicksort$(A, i+1, d)$

Algo Partition(A, g, d) // *partition de* $A[g..d-1]$
P1 chosir le pivot $p \leftarrow A[d-1]$
P2 $i \leftarrow g-1\,;\, j \leftarrow d-1$
P3 **loop** // *condition terminale à P6*
P4   **do** $i \leftarrow i+1$ **while** $A[i] < p$
P5   **do** $j \leftarrow j-1$ **while** $j > i$ et $A[j] > p$
P6   **if** $i \ge j$ **then** sortir de la boucle (à P8)
P7   échanger $A[i] \leftrightarrow A[j]$
P8 échanger $A[i] \leftrightarrow A[d-1]$
P9 **return** i

Tri rapide — performances

Temps de calcul

$$T(m) = \Theta(m) + T(i) + T(m-1-i).$$

La récurrence dépend de l'indice i du pivot.

	pivot i	**récurrence** $T(n)$	**solution** $T(n)$
Meilleur cas	$(n-1)/2$	$2 \cdot T\big((n-1)/2\big) + \Theta(n)$	$\Theta(n \log n)$
Pire cas	$0, n-1$	$T(n-1) + \Theta(n)$	$\Theta(n^2)$
Moyen cas	aléatoire	$\mathbb{E}T(n) = 2\mathbb{E}T(i) + \Theta(n)$	$\Theta(n \log n)$

Le pire cas arrive (entre autres) quand on a un tableau trié au début !

Choix du pivot

Deux choix performent très bien en pratique : médiane ou aléatoire.

P1.1 $k \leftarrow g + \mathsf{rnd}(d - g)$
P1.2 $p \leftarrow A[k]$
P1.3 **if** $k \neq d - 1$ **then**
P1.4 $\quad A[k] \leftarrow A[d - 1]$
P1.5 $\quad A[d - 1] \leftarrow p$

Médiane de trois

P1.1 **if** $d \geq g + 2$ **then**
P1.2   **if** $A[g] > A[d-2]$ **then** échanger $A[g] \leftrightarrow A[d-2]$
P1.3   **if** $A[d-1] > A[d-2]$ **then** échanger $A[d-1] \leftrightarrow A[d-2]$
P1.4   **if** $A[g] > A[d-1]$ **then** échanger $A[g] \leftrightarrow A[d-1]$
P1.5 $p \leftarrow A[d-1]$ // $A[g] \leq A[d-1] \leq A[d-2]$

et on se sert des **sentinelles** qui sont maintenant en place à $A[g]$, $A[d-2]$:

P2' $i \leftarrow g$; $j \leftarrow d-2$
P5'   **do** $j \leftarrow j-1$ **while** $A[j] > p$

Profondeur de la pile

Algo QUICKSORT_ITER$(A[0..n-1], g, d)$ // *tri de* $A[g..d-1]$

QI1 **while** $d - g > 1$ **do**

QI2 $\quad i \leftarrow$ PARTITION(A, g, d)

QI3 $\quad$ QUICKSORT_ITER(A, g, i)

QI4 $\quad g \leftarrow i + 1$ // *boucler au lieu de l'appel récursif*

$\Rightarrow$ profondeur $\leq \lg n$ sûrement

Sélection

chercher k-ème plus petit élément dans $A[0..n-1]$

Idée de clé : après avoir appellé $i \leftarrow \textsc{Partition}(A, 0, n)$, on trouve le k-ème élément en $A[0..i-1]$ si $k < i$ ou en $A[i+1..n-1]$ si $k > i$. En même temps, on réorganise le tableau pour que $A[k]$ soit le k-ème plus petit élément.

Algo $\textsc{Selection}(A[0..n-1], g, d, k)$

S1 **if** $d - g \leq 2$ **then** *// cas de base : 1 ou 2 éléments*

S2 **if** $d = g + 2$ et $A[d-1] < A[g]$ **then** échanger $A[g] \leftrightarrow A[d-1]$ *// 2 éléments*

S3 **return** $A[k]$

S4 $i \leftarrow \textsc{Partition}(A, g, d)$

S5 **if** $k = i$ **then return** $A[k]$ *// on l'a trouvé*

S6 **if** $k < i$ **then return** $\textsc{Selection}(A, g, i, k)$ *// continuer à la gauche*

S7 **if** $k > i$ **then return** $\textsc{Selection}(A, i+1, d, k)$ *// continuer à la droite*

Tri hybride

Tri par insertion est rapide quand n est petit :

Mergesort — approche hybride : utiliser insertion sort dans les récursions quand le sous-tableau devient petit

Quicksort — on peut retourner de la récurrence immédiatement + tri par insertion à la fin

Tri rapide – analyse

Déf. Soit $D(n)$ le nombre moyen de comparaisons avec un pivot aléatoire, où n est le nombre d'éléments dans un tableau $A[1..n]$.

On va démontrer que $\frac{D(n)}{n} \in O(\log n)$.

Lemme. On a $D(0) = D(1) = 0$, et

$$\begin{aligned} D(n) &= n - 1 + \frac{1}{n} \sum_{i=0}^{n-1} \Bigl(D(i) + D(n-1-i) \Bigr) \\ &= n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} D(i). \end{aligned}$$

Preuve. Supposons que le pivot est le i-ème plus grand élément de A. Le pivot est comparé à $(n-1)$ autre éléments pour la partition. Les deux partitions sont de tailles $(i-1)$ et $(n-1-i)$. Or, i prend les valeurs $1, 2, \ldots, n$ avec la même probabilité. $\square$

Performance moyenne (cont.)

Par le lemme précédent,

$$\begin{aligned} nD(n) - (n-1)D(n-1) &= \left(n(n-1) + 2\sum_{i=0}^{n-1} D(i)\right) \\ &\quad - \left((n-1)(n-2) + 2\sum_{i=0}^{n-2} D(i)\right) \\ &= 2(n-1) + 2D(n-1). \end{aligned}$$

D'où on a

$$\frac{D(n)}{n+1} = \frac{D(n-1)}{n} + \frac{2n-2}{n(n+1)} = \frac{D(n-1)}{n} + \frac{4}{n+1} - \frac{2}{n}.$$

Avec $E(n) = \frac{D(n)-2}{n+1}$, on peut écrire

$$E(n) = E(n-1) + \frac{2}{n+1}.$$

Performance moyenne (cont.)

Donc,

$$E(n) = E(0) + \frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{n+1}$$
$$= \frac{D(0)-2}{1} + 2(H_{n+1} - 1) = 2H_{n+1} - 4$$

où $H_n = \sum_{i=1}^{n} 1/i$ est le n-ième nombre harmonique.

En retournant à $D(n) = 2 + (n+1)E(n)$, on a alors

$$D(n) = 2(n+1)H_{n+1} - 4n - 2 < 2nH_{n+1}$$

Donc le nombre de comparaisons en moyenne est tel que

$$\frac{D(n)}{n} < 2H_{n+1} \in O(\log n).$$

En fait, on a $D(n)/n < 2 \cdot \ln n \approx 1.39 \lg n$.