Algorithmes de tri

Tri

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

Clés comparables :

```
public interface Comparable<T>
{
      int compareTo(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 : 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. . .

Tri interne

Entrée : tableau `A[]` de données comparables (interface `Comparable` en Java)

On veut trier le tableau

Solutions :

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

Tri (cont)

tableau $A[1..n]$

Tri par sélection :
pour $i \leftarrow 1, \ldots, n-1$ faire $A[1..i]$ contient les i éléments les plus petits de A, triés

Tri par insertion :
pour $i \leftarrow 2, \ldots, n$ faire $A[1..i]$ contient les i premiers éléments de A, triés

Tri par sélection

```
     Algo TRI-SELECTION(A[1 ... n])
S1   pour i ← 1, 2, ..., n − 1 faire
S2       minidx ← i
S3       pour j ← i + 1, ..., n faire
S4           si A[j] < A[minidx] alors minidx ← j
         // (maintenant A[minidx] = min{A[i], ..., A[n]})
S5       si i ≠ minidx alors échanger A[i] ↔ A[minidx]
```

Complexité

comparaison d'éléments [ligne S4] $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$ fois ;
échange d'éléments [ligne S5] $\leq (n-1)$ fois

Temps de calcul : $\Theta(n^2)$ pour tout A
mais pas nécessairement une mauvaise idée si l'échange est beaucoup plus couteux que la comparaison

Tri par insertion

```
    Algo TRI-INSERTION(A[1..n])
I1  pour i ← 2, ..., n faire
I2      j ← i − 1
I3      tandis que j ≥ 1 et A[j] < A[i] faire
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[1]` (sentinelle) — pas de test $j \geq 1$ en I3

remplacer échange par décalage en I4

Tri par tas

```
      HeapSort(A) // vecteur non-trié A[1..n]
H1    heapify(A)
H2    pour i ← |A|, ... 2 faire
H3        échanger A[1] ↔ A[i]
H4        Sombrer(A[1], 1, A[1..i − 1])
```

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

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

Temps $O(n \log n)$ dans le pire des cas, sans espace additionnelle !

quicksort : $O(n^2)$ dans le pire des cas
mergesort : $O(n \log n)$ dans le pire des cas mais utilise un espace auxiliaire de taille n

Tri par fusion

Algo MERGESORT($A[1..n]$)

M1 **si** $n < 2$ **alors retourner** A

M2 $B_1 \leftarrow$ TRI-FUSION($A[1..\lfloor n/2 \rfloor]$)

M3 $B_2 \leftarrow$ TRI-FUSION($A[\lfloor n/2 \rfloor + 1..n]$)

M4 $B \leftarrow$ FUSION(B_1, B_2)

M5 **retourner** B

Fusion de deux tableaux

```
1  Entrée A[1..n], B[1..m] (de type Comparable[])
2  intialiser C[1..n + m]
3  i ← 1; j ← 1
4  pour k ← 1, 2, . . . , n + m faire
5      si j > m ou A[i] ≤ B[j] alors C[k] ← A[i]; i ← i + 1
6      sinon C[k] ← B[j]; j ← j + 1
```

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

Tri par fusion — analyse

Thm. En tri par fusion, le nombre de comparaisons est inférieur à $C(n) = n\lceil \lg n \rceil$. ($n > 0$)

Preuve. Nombre total de comparaisons sur chaque niveau de récurrences est $\leq n$

Nombre total de niveaux de récurrences : $\lceil \lg(n) \rceil$. □

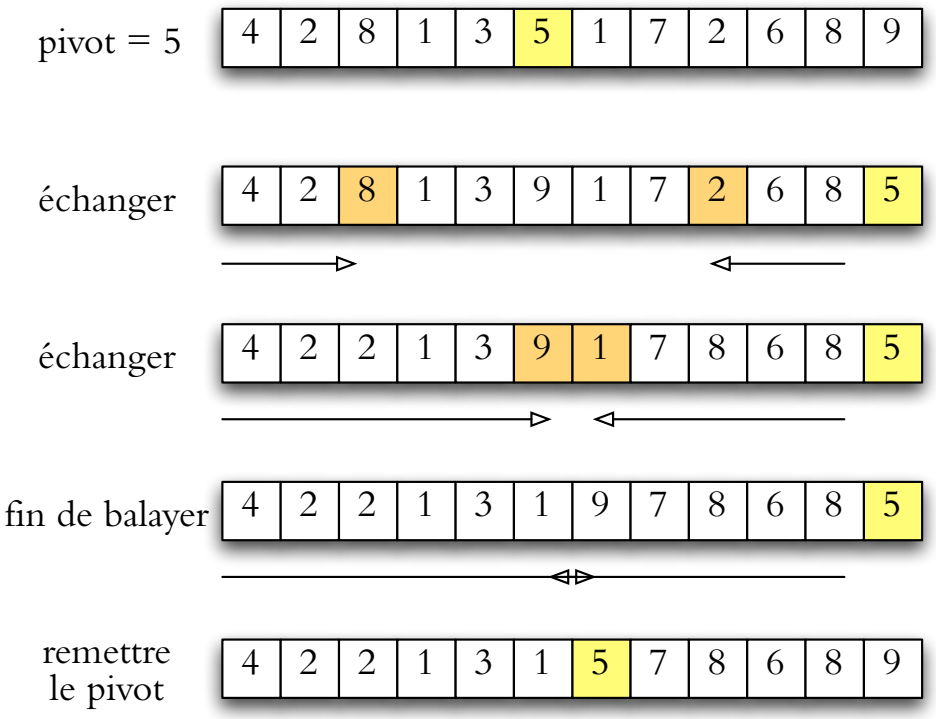
Nombre d'affectations : $\leq n\lceil \lg n \rceil$.

Complexité $\Theta(n \log n)$ mais on a besoin d'un espace auxiliaire de taille n pour la fusion

(quand le tableau est «presque trié», tri par insertion est plus rapide !)

Tri rapide

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



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

Tri rapide (cont)

Temps de calcul : $\Theta(n \log n)$ en moyen si le pivot est bien choisi ; $\Theta(n^2)$ en pire cas

Choix du pivot : médiane de trois ($A[1]$, $A[n-1]$ et $A[n]$) ou aléatoire (uniforme)

génie algorithmique :

1. tri par insertion quand n est petit ($\leq 5..20$)

2. ne pas exécuter les récursions sur les petits sous-tableaux, mais plutôt faire un tri par insertion une fois 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

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

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. □

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,

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

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 $D(n)/n < 2 \cdot \ln n \approx 1.39 \lg n$.