Formation enseignement de l'Informatique– 2011

# Algorithmes de Tri

*Olivier Bournez*

## 1 Tris par comparaisons

### 1.1 Tri par remontée de bulles

L'idée consiste à comparer deux éléments adjacents, et à les échanger s'ils ne sont pas dans l'ordre. Lorsqu'on réalise un passage, on range le plus grand élément en fin de tableau (ou le plus petit au début). On a donc réalisé une partition entre l'élément le plus grand d'une part, et tous les autres éléments d'autre part.

1. Dérouler l'algorithme sur le tableau suivant.

$$[5, 8, 9, 4, 1]$$

2. En supposant que le tableau à trier est $[2, 3, 4, \ldots, n, 1]$, combien fait-on d'échanges ? Combien fait-on de comparaisons ?
3. Ecrire en Java la méthode assurant cet algorithme.

### 1.2 Tri par sélection

Cela consiste à déterminer le plus petit élément, puis le deuxième petit élément, et ainsi de suite.

1. Dérouler l'algorithme sur le tableau suivant.

$$[5, 8, 9, 4, 1]$$

2. Écrire en Java les méthodes assurant cet algorithme.
3. Quelle est la complexité du tri par sélection ?

### 1.3 Tri rapide

Inventé par HOARE en 1961, ce tri s'appelle aussi parfois tri par segmentation, ou quicksort. Le principe du tri est de créer une partition de la liste $E$ de départ deux sous-listes $E_1$ et $E_2$, où $E_1$ sont les éléments plus petits qu'un élément, appelé "pivot", et $E_2$ ceux qui sont plus grands. La qualité de la partition dépend du choix du pivot. L'idéal serait de choisir comme pivot la médiane des $n$ éléments de l'ensemble $E$ de départ. Une solution simple et relativement efficace est de choisir la médiane comme le premier élément (ou celui du milieu, ou le dernier).

1. Dérouler l'algorithme sur le tableau suivant.

$$[8, 5, 9, 4, 1]$$

2. Écrire en java les méthodes nécessaires au tri rapide.

# 2 Utilisation des tableaux triés

## 2.1 Chercher un élément dans un tableau

Le but est d' écrire une fonction de recherche d'une valeur dans un tableau d'entiers.

1. Écrire une méthode `recherche(int[] t, int x)` qui prend un quelconque tableau $t$ et une valeur entière $x$ et qui retourne une position de $x$ dans $t$. Si $x$ n'est pas dans $t$, la méthode retourne $-1$.

   Donner le nombre de comparaisons dans le pire des cas.

2. Dans le cas où le tableau est trié par ordre croissant, il est possible de retrouver plus rapidement la valeur recherchée $x$ par dichotomie. Le principe est de comparer $x$ à la valeur stockée au milieu du tableau :

   si elle est plus grande que $x$, il faut chercher dans la moitié inférieur du tableau, et sinon dans la moitié supérieure. et sinon dans la moitié supérieure.

   On recommence alors sur la portion du tableau de taille moitié et ainsi de suite jusqu'à trouver la valeur. Écrire en Java la méthode `rechercheDichotomie(int[] t, int x)` basée sur cette approche.

   Donner le nombre de comparaisons dans le pire des cas pour le cas où le nombre d'éléments du tableau est égal à $2^k$.

3. Comparer ces deux méthodes pour le cas où le nombre d'éléments du tableau est égal à $2^k$.

## 2.2 Comparaison des tableaux

On dit que deux tableaux sont
- égaux si ils contiennent les mêmes éléments aux mêmes positions
- similaires si ils contiennent les mêmes éléments mais pas forcément aux mêmes positions
- comparables si l'ensemble des valeurs qu'ils contiennent est le même.

Par exemple,
- les tableaux $[5, 2, 4, 1, 2, 1]$ et $[1, 1, 2, 2, 4, 5]$ sont similaires mais pas égaux.
- les tableaux $[5, 2, 4, 1, 2, 1]$ et $[1, 2, 4, 5]$ sont similaires mais pas égaux.

1. Écrire une méthode qui teste si deux tableaux sont égaux.
2. Écrire une méthode qui teste si deux tableaux sont similaires. Donner la complexité de cet algorithme.
3. Écrire une méthode qui teste si deux tableaux sont comparables.

# 3 Accélération du tri par sélection : le tri arbre.

Imaginé par Williams en 1964, ce tri est aussi appelé tri en tas ou tri maximier. Ce tri est une accélération du tri par sélection et échange : une organisation préalable de l'ensemble de départ permet d'obtenir une sélection du maximum en un temps qui est dans l'ordre de $log_2 n$, ce qui permet à l'algorithme de redevenir optimum. Le vecteur est considéré comme ayant une structure d'arbre binaire. On fera l'hypothèse que les éléments du tableau ont des indices qui vont de 1 à $f$. Un élément d'indice $i$ du vecteur est considéré comme un nœud de l'arbre ; son fils gauche s'il existe se trouve à l'indice $2 * i + 1$ et son fils droit s'il existe se trouve à l'indice $2 * i + 2$. Ce nœud est une feuille si $2 * i + 1$ est strictement supérieur à $f$

1. Ecrire une fonction `descendre` permettant de réorganiser un tas dont seule la racine n'est pas à sa place. La racine se trouve à l'indice $d$.
2. Ecrire une fonction `remonter` permettant de remonter le dernier élément à sa place dans le tas formés par les éléments précédent.

3. L'organisation d'un tableau peut se faire de deux manières :
   – remonter tous les éléments, en partant du second, jusqu'au dernier.
   – descendre tous les éléments, en partant du milieu, vers le début.
   Ecrire ces deux fonctions.
4. En utilisant les fonctions précédentes, proposer une procédure `triTas` assurant le tri par Tas.