

# Introduction à l'algorithmique

---

Marie Asté

Marie.Aste@sophia.inria.fr

Ignasi Sau

Ignasi.Sau@sophia.inria.fr

<http://www-sop.inria.fr/mascotte/personnel/Ignasi.Sauvalls>

LPSIL

Année 2007-2008

## Objectifs

- Quels sont les critères pour caractériser un « bon algorithme »
- Notions élémentaires de complexité algorithmique
- Récursivité
- Algorithmes de base sur les listes
- Algorithmes de base sur les arbres
- Algorithmes de base sur les graphes

## Exemple 1 : Recherche d'un élément dans une séquence

- Données : une séquence de  $n$  entiers distincts  $e_0, e_1, \dots, e_{n-1}$   
un entier  $x$
- Résultat :  $-1$  si  $x$  n'est pas dans la séquence  $e_0, \dots, e_{n-1}$   
 $j$  si  $x = e_j$

Recherche de 6 dans  $\{3, 10, 4, 1, 6, 3\}$   $\rightarrow 4$

Recherche de 31 dans  $\{3, 10, 4, 1, 6, 3\}$   $\rightarrow -1$

- Principe de l'algorithme : parcourir la séquence en comparant  $x$  à  $e_0$  puis à  $e_1$ , puis à  $e_2$ , ....  
Si l'on trouve un  $i$  tel que  $e_i=x$ , le résultat est  $i$ ,  
Si l'on parcourt les  $e_i$  jusqu'à  $i=n$ , le résultat est  $-1$

### Traduction en Java :

```
public class TableauEntier {  
  
    /** méthode pour rechercher l'indice d'un élément  
    ** dans un tableau */  
  
    public static int recherche(int[] tab, int x) {  
        int i=0;  
        while(i<tab.length && x!=tab[i])  
            i++;  
        if (i==tab.length) return -1;  
        else return i;  
    }  
}
```

- Question 1 : l'algorithme est-il correct ?
- Question 2 : l'algorithme termine-t-il ?
- Question 3 : quelle est la place utilisée en mémoire ?
- Question 4 : quel est le temps d'exécution ?

### Question 1 : l'algorithme est-il correct ?

```
public static int recherche(int[] tab, int x) {
    /** méthode pour rechercher l'indice d'un élément
     * antécédent : tab est un tableau d'entiers distincts,
     *             x est un entier
     * conséquent : renvoie i si tab[i] == x,
     *             -1 si x n'est pas dans le tableau
     */
    int i=0;
    while(i<tab.length && x!=tab[i])
        // A1 :  $\forall 0 \leq j \leq i, \text{tab}[j] \neq x$ 
        i++;
    if (i==tab.length)
        // A2 : x n'est pas dans le tableau
        return -1;
    else return i; // A3 : x est à l'indice i
}
```

- A1 est vrai en rentrant dans la boucle (pour  $i = j = 0$ )
- A1 est vrai à chaque passage dans la boucle  
Puisque on entre dans la boucle quand  $\text{tab}[i] \neq x$  et que l'on a incrémenté  $i$
- A2 est vrai  
Puisque A1 est vrai à chaque étape de la boucle, si  $i = \text{tab.length}$  alors  
 $\forall 0 \leq j < \text{tab.length} \text{ tab}[j] \neq x$   
donc  $x$  n'est pas dans le tableau
- A3 est vrai  
Si  $i < \text{tab.length}$  alors on est sorti du while avec la condition  $x == \text{tab}[i]$   
donc  $x$  est à l'indice  $i$

Question 2 : l'algorithme termine-t-il ?

```
int i=0;
while(i<tab.length && x!=tab[i])
    i++;
if (i==tab.length) return -1;
else return i;
```

au pire  $i$  croît de 0 à  $\text{tab.length}$  qui est une valeur finie

Question 3 : quelle est la place utilisée en mémoire ?

La place nécessaire pour stocker  $x$  et  $\text{tab}$

Si  $\text{tab}$  contient  $n$  éléments on dit :

*Complexité en espace =  $\theta(n)$*

#### Question 4 : quel est le temps d'exécution ?

Temps CPU : dépend de la machine

« Temps de l'algorithme » : on fait au plus `tab.length` passages dans la boucle `for`

Si leTableau contient  $n$  éléments :

*Complexité en temps dans le pire des cas =  $n$*

*Complexité en temps dans le meilleur des cas =  $1$*

*Complexité en temps en moyenne =  $p (n+1)/2 + n (1-p)$*

*(où  $p$  est la probabilité pour que  $x$  soit dans le tableau)*

#### Algorithmique en Java : du java bien commenté !

- **Antécédent** : conditions d'entrée
  - quels types de données sont traités ?  
entiers, chaînes de caractères, réels, tableau d'entiers, ...
  - conditions sur les données ?  
entiers positifs, non nuls, tableau d'entiers triés par ordre croissant, décroissant, ...
- **Conséquent** : que renvoie la méthode (return) ? quelles modifications ont été apportées sur les données (void) ?  
renvoie l'indice de l'élément  
renvoie le maximum des éléments du tableau,  
ordonne les éléments du tableau par ordre croissant, ...
- **Assertions** : propriétés liant les données d'entrée et les variables de la méthode.  
Permettent de justifier la correction : si l'antécédent est vérifié, après exécution de la méthode, le conséquent est vérifié  
Permettent de justifier la terminaison : si l'antécédent est vérifié, le calcul s'arrêtera
- **Complexité en temps dans le pire des cas**

## Exemple 2 : Recherche d'un élément dans une séquence ordonnée

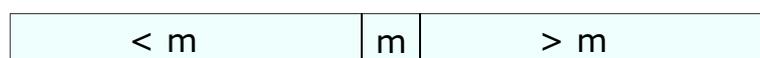
- Données : une séquence de  $n$  entiers distincts  $e_0, e_1, \dots, e_{n-1}$  ordonnés par ordre croissant un entier  $x$
- Résultat :  $-1$  si  $x$  n'est pas dans la séquence  $e_0, e_1, \dots, e_{n-1}$   
 $i$  si  $x = e_i$

Recherche de 6 dans  $\{3, 4, 6, 10, 34\}$   $\rightarrow 2$

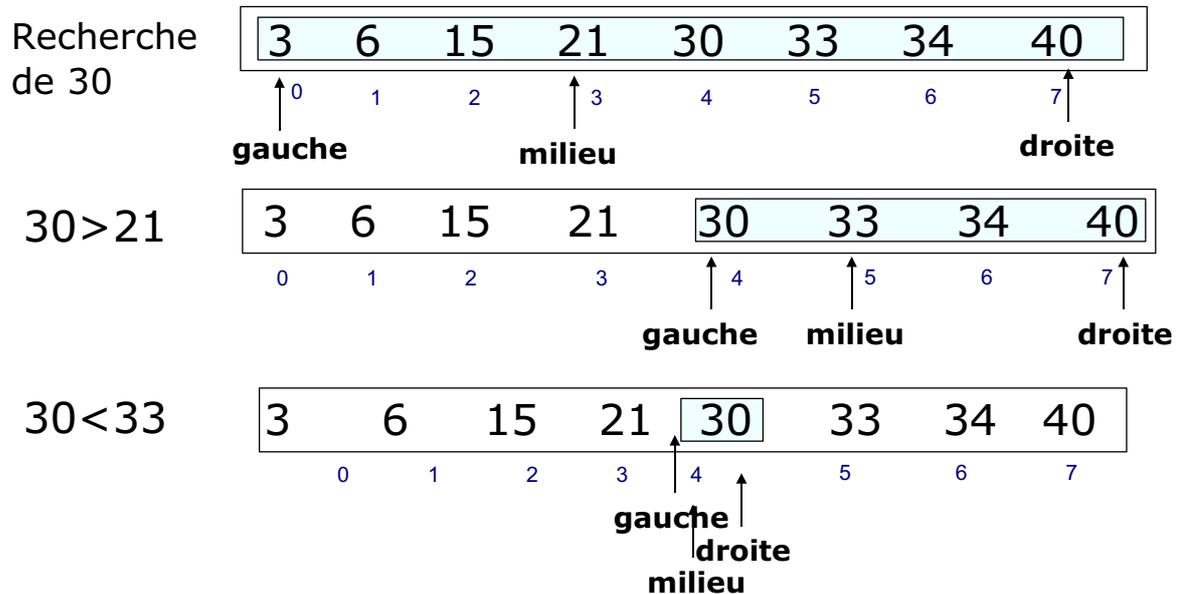
Recherche de 31 dans  $\{3, 4, 6, 10, 34\}$   $\rightarrow -1$

- Solution 1 : le même algorithme que dans le cas où les éléments ne sont pas ordonnés.

- Solution 2 : utiliser le fait que les éléments sont ordonnés pour appliquer le paradigme « diviser pour régner »
  - *Principe* : comparer  $x$  à l'élément  $m$  qui est au milieu de la partie du tableau considérée.  
Si  $x = m$  renvoyer l'indice de  $m$   
Si  $x < m$  chercher  $x$  dans la partie du tableau à gauche de  $m$   
Si  $x > m$  chercher  $x$  dans la partie du tableau à droite de  $m$   
Si la partie considérée est vide, renvoyer  $-1$



- *Avantage* : meilleure complexité en temps (i.e en  $O(\log n)$ )



30 est trouvé; il est à l'indice 4

```

/** pour rechercher l'indice d'un élément
 * antécédent : tab est un tableau d'entiers
 *               distincts ordonné par ordre croissant,
 *               x est un entier
 * conséquent : renvoie i si tab[i] == x,
 *               -1 si x n'est pas dans le tableau
 * complexité : O(log n)
 */
public int rechercheVite(int[] tab, int x) {
    int gauche = 0;
    int droite = tab.length - 1;
    int milieu;

```

```
while (gauche <= droite) {
    // A1 :  $\forall j < gauche \text{ tab}[j] \neq x$ 
    //       $\forall j > droite \text{ tab}[j] \neq x$ 
    milieu = (gauche + droite) / 2 ;
    if (x==tab[milieu])
        // A2 : x est à l'indice milieu
        return milieu;
    if (x<tab[milieu]) droite = milieu - 1;
    else gauche = milieu + 1;
}
// A3 : x n'est pas dans le tableau
return -1;
}
```

## A VOUS

- Que se passe-t-il si les éléments du tableau ne sont pas distincts dans le cas de la recherche simple ?

21	6	2	21	21	33	21	40
0	1	2	3	4	5	6	7

- Que se passe-t-il si les éléments du tableau ne sont pas distincts dans le cas de la recherche dichotomique ?

3	6	21	21	21	33	34	40
0	1	2	3	4	5	6	7

- Comment peut-on prendre en compte le fait que les éléments sont triés dans la recherche simple ?

## A VOUS : tri par insertion

---

- Données : un tableau d'entiers
- Résultat : le tableau est trié par ordre croissant
- Algorithme : respecter l'assertion de boucle A1

6	3	21	62	68	30	33	4	1	40	70
0	1	2	3							tab.length-1

```
public static void triInsertion(int[] tab) {  
    for (int i=1; i<tab.length;i++) {
```

3	6	21	30	62	68	33	4	1	40	70
0	1	2		i-1	i					tab.length-1

```
        // A1: les éléments de tab de 0 à i-1 sont triés par ordre croissant  
        //     tab contient les mêmes valeurs qu'à l'état initial
```

```
    }  
}
```

## Plan

- Complexité des algorithmes
  - Bases mathématiques
    - sommations
    - logarithmes
  - Notations asymptotiques
  - Evaluation de la complexité d'algorithmes itératifs
- Induction et récursivité
  - Exemple 1 : palindrome
  - Exemple 2 : recherche d'un élément dans une liste ordonnée
  - Exemple 3 : factorielle
  - schémas d'induction
    - les entiers positifs
    - les listes
    - les arbres binaires
  - correction et terminaison des algorithmes récursifs
  - relations de récurrence et évaluation de la complexité d'algorithmes récursifs

- Listes
  - les listes chaînées
  - les listes chaînées ordonnées
  
- Les arbres binaires
  - les arbres binaires de recherche
  - les AVL
  
- Introduction aux graphes
  - Définition et exemples d'utilisation
  - Algorithmes de parcours élémentaire

## Bibliographie

- T. Cormen, C. Leiserson, R. Rivest : « Introduction à l'algorithmique », Dunod.
- S. Baase, A. V. Gelder : « Computers algorithms : Introduction to Design & Analysis », Addison Wesley
- U. Mander : « Introduction to algorithms, A Creative Approach », Addison Wesley