

# UEF 1 : Informatique & Programmation

Faculté des Sciences de Nice

DEUG 2001-2002

Jérôme DURAND-LOSE

Jean-Paul ROY

**COURS 12**

## Premier tri

Premiertri

ePmriertri

emPriertri

eimPrertri

eeimPrtrtri

eeimPrtrtri

eeimPrtrtri

eeimPrtrtri

eeimPrtrtri

→ conserver une collection triée

→ trier par insertion

12-2

## Collection triée

Avoir une collection (ou un tableau) triée  
simplifie la recherche  
(dichotomie sur annuaire, dictionnaire...)

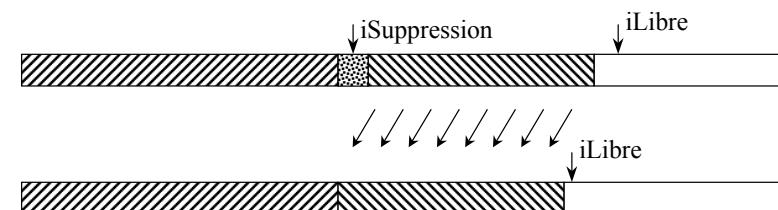
Comment garder une collection triée  
quand on enlève ou ajoute un élément ?

Comment trier un tableau ?

12-3

## Suppression dans une collection triée

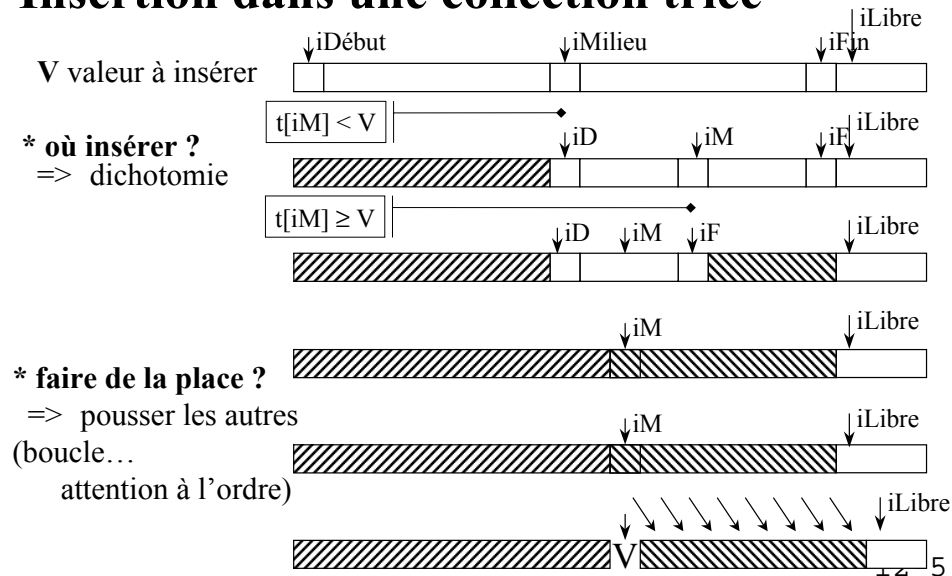
Décaler pour « écraser » la valeur à supprimer et éviter un trou



**parcourir** le tableau de *iSuppression* inclus à *iLibre* exclu  
recopier le contenu de la case suivante dans la case courante  
diminuer *iLibre* de 1

12-4

## Insertion dans une collection triée



## Insertion dans une collection triée

### Type Collection triée

#### Attributs

table : tableau

iLibre : entier indiquant la première case libre de table

#### Donnée

élément : élément à insérer

#### Variable

iInsertion : entier

#### Insertion d'un élément à sa place

iInsertion  $\leftarrow$  position où élément doit être inséré (trouvée par dichotomie)

**pour** i variant de iLibre-1 à iInsertion **faire**  
 recopier le contenu de la case i dans la case suivante

**fin faire pour**  
 mettre élément dans la case d'indice iInsertion  
 augmenter iLibre de 1

12-6

## Coût / durée de l'insertion

*durée* (insertion d'un élément dans un tableau de  $n$  cases)

= *durée* (recherche dichotomique)

+ *durée* (déplacer entre 0 et  $n$  cases)

=  $O(\log n) + O(n) = O(n)$

Ce qui dure le plus longtemps est donc de faire de la place, trouver l'endroit où insérer est relativement rapide

On aurait eu la même « complexité » avec une recherche séquentielle à la place d'une recherche dichotomique

12-7

## Simplification de l'algorithme

Faire de la place prend tout le temps

- peut-on l'améliorer ?

- peut-on en tirer partie pour la recherche ?

**Idee :** faire de la place et chercher en même temps !

On part de la fin de la collection et on déplace tant que l'on n'a pas encore trouvé où insérer

On a la même complexité, mais la réalisation du programme est plus simple

12-8

## Insertion dans une collection triée

### Type Collection triée

#### Attributs

table : tableau

iLibre : entier indiquant la première case libre de table

#### Donnée

élément : élément à insérer

#### Variable

i : entier

#### Insertion d'un élément à sa place

i ← iLibre

**tant que** i > 0 et table[i - 1] > élément **faire**

table[i] ← table[i - 1]

i ← i - 1

**fin tant que**

mettre élément dans la case d'indice i

augmenter iLibre de 1

À chaque fois,  
la case i est libre

12-9

## Tri par insertion

### Problème (fréquent en informatique)

on a des données dans un tableau et on voudrait les avoir dans un certain ordre.

### Idée

Tirer partie des collections triées et de leurs possibilité d'insertion

### Stratégie

Partir d'une collection vide (donc triée)

Insérer les éléments les uns après les autres

Recopier la collection triée dans le tableau de départ

12-10

## Trace du tri par insertion (première version)

tableau à trier	insérer	tableau annexe
17 7 23 11 53 13		17
17 7 23 11 53 13	7	7 17
17 7 23 11 53 13	7	7 17 23
17 7 23 11 53 13	7	7 11 17 23
17 7 23 11 53 13	7	7 11 17 23 53
17 7 23 11 53 13	7	7 11 13 17 23 53
recopier		7 11 13 17 23 53

12-11

## Algorithme

Créer une collection triée

parcourir le tableau

Insérer la valeur courante dans la collection triée

Recopier la collection triée dans le tableau de départ

### Donnée

table : tableau (à trier)

### Variables

tableAnnexe : tableau de même taille que table

iLibre : entier (indice de la première case vide dans tableAnnexe) = 0

i : entier

### Algorithme pour le tri par insertion

**parcourir** table **faire**

i ← iLibre

**tant que** i > 0 et tableAnnexe[i - 1] > case courante de table **faire**

tableAnnexe[i] ← tableAnnexe[i - 1]

diminuer i de 1

**fin tant que**

tableAnnexe[i] ← case courante de table

augmenter iLibre de 1

**fin parcourir**

recopier tableAnnexe dans table

12-12

## Un tableau de moins

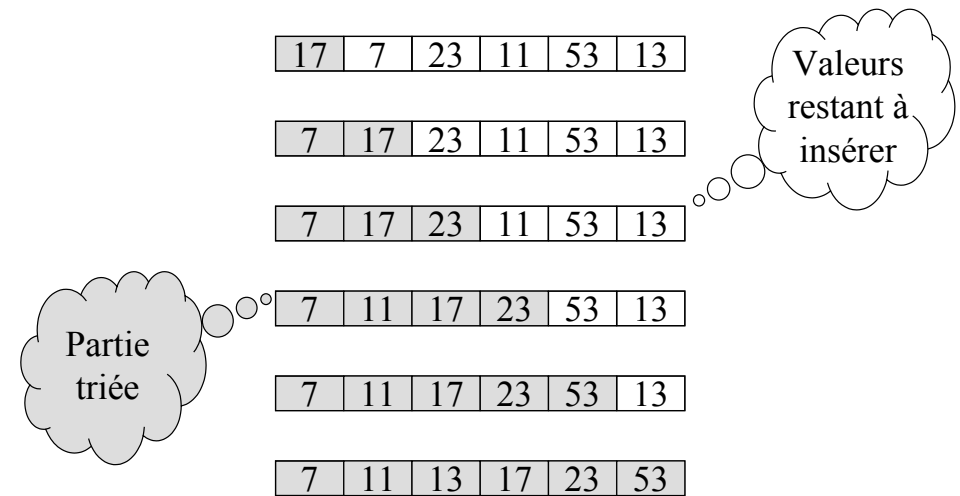
Dans la trace,  
la partie grisée (valeurs déjà rangée)  
correspondent à  
la partie utile de la collection

Peut-on utiliser un seul tableau ?

Oui, il faut « sortir » la valeur à insérer, cela fait une place libre,  
on décale ensuite jusqu'à insérer la valeur

12-13

## Trace du tri par insertion



12-14

## Programme

```
static void triInsertion ( double [] table ) {  
    for ( int i = 0; i < table.length ; i++ ) {  
        int iCher = i;  
        double val = table[ i ];  
        while ( ( 0 < iCher ) && ( table[ iCher - 1 ] > val ) ) {  
            table[ iCher ] = table[ iCher - 1 ];  
            iCher --;  
        }  
        table[ iCher ] = val;  
    }  
}
```

12-15

## Durée / complexité du tri par insertion

*durée* ( décaler et trouver la position pour la valeur en  $i$  )  
= entre 0 et  $i$   
=  $O(i)$

$durée ( tri ) = \sum O(i) \quad \{ \text{pour } i \text{ variant de } 1 \text{ à } n \}$   
=  $O(1 + 2 + 3 + \dots + n)$   
=  $O(n^2)$

L'algorithme de tri par insertion est « *quadratique* », i.e.,  
le nombre d'opérations faites est de l'ordre du carré du nombre  
d'éléments du tableau.

Question peut-on faire mieux ? *Oui mais chut*

12-16