

# Langage et algorithmique

Rodéric Moitié et A. Malek TOUMI

ENSTA Bretagne

# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- Tri par insertion
- Tri shell
- Autres tris

# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- Tri par insertion
- Tri shell
- Autres tris

# Les tableaux

## Définition (Tableau)

Structure de données contenant plusieurs éléments du même type (primitif ou composite).

# Les tableaux

## Définition (Tableau)

Structure de données contenant plusieurs éléments du même type (primitif ou composite).

## Exemple (Déclaration de tableaux)

```
int idEtudiant[];  
char[] notes;  
double coordonnees[][];
```

## Création d'un tableau

- Nécessité de définir la taille du tableau  
⇒ utilisation de **new**

## Création d'un tableau

- Nécessité de définir la taille du tableau  
⇒ utilisation de **new**

## Exemple (Création de tableaux)

```
int idEtudiant[] = new int [20];
char[] notes = new char [20];
double coordonnees[][] = new double [10] [5];
```

# Création d'un tableau

- Nécessité de définir la taille du tableau  
⇒ utilisation de **new**

## Exemple (Création de tableaux)

```
int idEtudiant[] = new int [20];
char[] notes = new char [20];
double coordonnees[][] = new double [10] [5];
```

## Exemple (Création et initialisation de tableau)

```
char[] notes = {'A', 'B', 'C', 'D', 'F'};
double coordonnees[][] = {{0.0, 0.1}, {0.2, 0.3}};
```

# Accès aux éléments des tableaux

- Accéder à un élément = accéder à une case du tableau
- Opérateur []
- Première case : 0
- Tableaux multidimensionnels : plusieurs indices
- Taille du tableau : **tab.length**, **tab[0].length**, ...

## Exemple (Accès à un tableau)

```
char note1; note1 = notes[0];  
notes[4] = 'Z';  
double ligne2Col1 = coordonnees[1][0];
```

# Tableaux et boucles

Utiliser les boucles facilite le parcours des tableaux.

## Exemple

```
tab[0]=1 ; tab[1]=2 ; ... tab[9]=10;
```

Remplacé par :

```
for (int i=0 ; i<tab.length ; i++)  
    tab[i] = i+1;
```

# Récapitulatif

- Déclaration d'un tableau

## Exemple

```
int tableau[] ;
```

# Récapitulatif

- Déclaration d'un tableau
- Création d'un tableau

## Exemple

```
int tableau[] = new int[10];
```

# Récapitulatif

- Déclaration d'un tableau
- Création d'un tableau
- Remplissage du tableau

## Exemple

```
int tableau[] = new int[10];
for (int i=0 ; i<tableau.length ; i++)
    tableau[i] = i;
```

# Récapitulatif

- Déclaration d'un tableau
- Création d'un tableau
- Remplissage du tableau
- Utilisation du tableau

## Exemple

```
int tableau[] = new int[10];
for (int i=0; i<tableau.length; i++)
    tableau[i] = i;
System.out.println(tableau[0]);
```

# Affichage du contenu d'un tableau

- Méthode d'affichage « naïve » : afficher la variable

# Affichage du contenu d'un tableau

- Méthode d'affichage « naïve » : afficher la variable

## Incorrect

```
int tab[] = new int[10];  
System.out.println(tab);
```

## Affichage

```
[I@187c6c7
```

# Affichage du contenu d'un tableau

- Méthode d'affichage « naïve » : afficher la variable
- Méthode correcte : afficher les cases une à une

## Correct

```
int tab[] = new int[10];
for (int i=0 ; i<tab.length ; i++) {
    System.out.print(tab[i]);
}
```

## Affichage

0 0 0 0 0 0 0 0 0 0

# Affichage du contenu d'un tableau

- Méthode d'affichage « naïve » : afficher la variable
- Méthode correcte : afficher les cases une à une

## Correct (depuis Java 5.0)

```
int tab[] = new int[10];  
for (int val: tab) {  
    System.out.print(val);  
}
```

## Affichage

```
0 0 0 0 0 0 0 0 0 0
```

# Affichage du contenu d'un tableau

- Méthode d'affichage « naïve » : afficher la variable
- Méthode correcte : afficher les cases une à une

## Autre écriture (depuis Java 5.0)

```
int tab[] = new int[10];  
System.out.print(Arrays.toString(val));
```

## Affichage

```
0 0 0 0 0 0 0 0 0 0
```

# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- Tri par insertion
- Tri shell
- Autres tris

# Remplissage aléatoire d'un tableau

- Utilisation de la classe **Random**

## Exemple

```
Random alea = new Random();
int tab[] = new int[10];
for (int i=0 ; i<tab.length ; i++)
    tab[i] = alea.nextInt(10);
```

# Mé lange d'un tableau

---

## Algorithme 1: Mé lange()

---

**Entrées :** tab : entier[]

**Données :** i, n : entier

$n \leftarrow \text{taille}(\text{tab}) - 1 ;$

**tant que**  $n > 0$  **faire**

$i \leftarrow$  nombre aléatoire entre 0 et  $n$  inclus ;

permuter  $\text{tab}[n]$  et  $\text{tab}[i]$  ;

décrémenter  $n$  ;

**fin**

---

# Mé lange d'un tableau

```
// no max des elements
int n = tableau.length-1;
int tmp;
while (n>0) {
    int i = alea.nextInt(n+1);
    tmp = tableau[i];
    tableau[i] = tableau[n];
    tableau[n] = tmp;
    n--;
}
```



# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- Tri par insertion
- Tri shell
- Autres tris

oooooooooooo

●oooooooooooo

Tri par sélection

# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- Tri par insertion
- Tri shell
- Autres tris

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

7	2	1	8	4
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

7	2	1	8	4
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

1	2	7	8	4
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

1	2	7	8	4
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

1	2	7	8	4
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

1	2	7	8	4
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

1	2	4	8	7
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencer entre le deuxième plus petit et le deuxième élément

## Exemple

1	2	4	8	7
---	---	---	---	---

# Principe général

- Rechercher le plus petit élément du tableau
- Le permutez avec le premier élément
- Recommencez entre le deuxième plus petit et le deuxième élément

## Exemple

1	2	4	7	8
---	---	---	---	---

## Algorithme tri par sélection

---

**Algorithme 2:** triSel

## Entrées : tab : entier

**Données** : indMin, i, j, n : entier

*n*  $\leftarrow$  *taille(tab)* ;

**pour**  $i \in [0, n - 2]$  **faire**

## Algorithme tri par sélection

---

**Algorithm 3:** triSel

## Entrées : tab : entier

**Données** : indMin, i, j, n : entier

*n*  $\leftarrow$  *taille(tab)* ;

**pour**  $i \in [0, n - 2]$  **faire**

*indMin*  $\leftarrow i$  ;

**pour**  $j \in [i + 1, n - 1]$  **faire**

**si**  $tab[j] < tab[indMin]$  **alors**

*indMin*  $\leftarrow j$ ;

## Algorithme tri par sélection

---

**Algorithme 4:** triSel

## Entrées : tab : entier

**Données** : indMin, i, j, n : entier

*n*  $\leftarrow$  *taille(tab)* ;

**pour**  $i \in [0, n - 2]$  **faire**

*indMin*  $\leftarrow i$  ;

**pour**  $j \in [i + 1, n - 1]$  **faire**

si  $tab[j] < tab[indMin]$  alors

| *indMin*  $\leftarrow j$  ;

fin

fin

**inverser(tab[indMin], tab[i]);**

fin

# Algorithme tri par sélection

```
public void triSel(int tab[]) {  
    int indMin; // indice du plus petit élément  
    int elemTmp; // élément temporaire  
    for (int i=0 ; i<tab.length ; i++) {  
        indMin = i;  
        for (int j=i+1 ; j<tab.length ; j++)  
            if (tab[j] < tab[indMin])  
                indMin = j;  
        elemTmp = tab[indMin];  
        tab[indMin] = tab[i];  
        tab[i] = elemTmp;  
    }  
}
```

# Étude de la complexité du tri

Complexité  $C(n)$  : ordre de grandeur du nombre d'opérations d'un algorithme.



Tri par sélection

# Étude de la complexité du tri

Complexité  $C(n)$  : ordre de grandeur du nombre d'opérations d'un algorithme. Notation  $\Theta$  :

$$\Theta(g(n)) = \left\{ \begin{array}{l} f(n) : \exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N} / \\ \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \end{array} \right\}$$

# Étude de la complexité du tri

Complexité  $C(n)$  : ordre de grandeur du nombre d'opérations d'un algorithme. Notation  $\Theta$  :

$$\Theta(g(n)) = \left\{ \begin{array}{l} f(n) : \exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N} / \\ \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \end{array} \right\}$$

$\Theta(g(n))$  représente l'ensemble des fonctions de même ordre que  $g(n)$ .

## Tri par sélection :

- Boucle externe :  $i = n - 2$  fois
- Boucle interne :  $n - i$  fois
- Somme :

$$\sum_{i=0}^{n-2} (n - i) = \sum_{j=1}^{n-1} j = \frac{n(n - 1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

$$C(n) = \frac{1}{2}n^2 - \frac{1}{2}n$$

Tri par sélection :

- Boucle externe :  $i = n - 2$  fois
- Boucle interne :  $n - i$  fois
- Somme :

$$\sum_{i=0}^{n-2} (n - i) = \sum_{j=1}^{n-1} j = \frac{n(n - 1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

$$C(n) \simeq \frac{1}{2}n^2$$

## Tri par sélection

Tri par sélection :

- Boucle externe :  $i = n - 2$  fois
- Boucle interne :  $n - i$  fois
- Somme :

$$\sum_{i=0}^{n-2} (n - i) = \sum_{j=1}^{n-1} j = \frac{n(n - 1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

$$C(n) \simeq n^2$$

## Tri par sélection

Tri par sélection :

- Boucle externe :  $i = n - 2$  fois
- Boucle interne :  $n - i$  fois
- Somme :

$$\sum_{i=0}^{n-2} (n - i) = \sum_{j=1}^{n-1} j = \frac{n(n - 1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

$$C(n) \in \Theta(n^2)$$

## Tri par sélection

Tri par sélection :

- Boucle externe :  $i = n - 2$  fois
- Boucle interne :  $n - i$  fois
- Somme :

$$\sum_{i=0}^{n-2} (n - i) = \sum_{j=1}^{n-1} j = \frac{n(n - 1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

$$C(n) = \Theta(n^2)$$

oooooooooooo

oooooooo●○oooooooo

Tri bulles

# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- **Tri bulles**
- Tri par insertion
- Tri shell
- Autres tris

# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

7	2	1	8	4
---	---	---	---	---

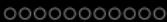


# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

7	2	1	8	4
---	---	---	---	---

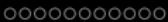


# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	7	1	8	4
---	---	---	---	---



# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	7	1	8	4
---	---	---	---	---



# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	1	7	8	4
---	---	---	---	---



# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	1	7	8	4
---	---	---	---	---

# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	1	7	8	4
---	---	---	---	---



# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	1	7	8	4
---	---	---	---	---

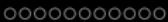


# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	1	7	4	8
---	---	---	---	---



# Principe du tri bulles

- Très simple à mettre en oeuvre
- Comparaisons locales : un élément et son successeur
- Faire remonter l'élément le plus grand, puis le deuxième plus grand,  
...

## Exemple

2	1	7	4	8
---	---	---	---	---

# Algorithme du tri bulles

---

## Algorithme 5: Algorithme du tri bulles

---

**Entrées :** tab : entier[]

**Données :** i, j, n : entier

$n \leftarrow \text{taille}(\text{tab});$

**pour**  $i \in [n - 1, 1]$  **faire**

|

# Algorithme du tri bulles

---

## Algorithme 6: Algorithme du tri bulles

---

**Entrées :** tab : entier[]

**Données :** i, j, n : entier

$n \leftarrow \text{taille(tab)};$

**pour**  $i \in [n - 1, 1]$  **faire**

**pour**  $j \in [0, i - 1]$  **faire**

        |

        |

# Algorithme du tri bulles

---

## Algorithme 7: Algorithme du tri bulles

---

**Entrées :** tab : entier[]

**Données :** i, j, n : entier

$n \leftarrow \text{taille}(\text{tab});$

**pour**  $i \in [n - 1, 1]$  **faire**

**pour**  $j \in [0, i - 1]$  **faire**

**si**  $\text{tab}[j] > \text{tab}[j + 1]$  **alors**

            |

# Algorithme du tri bulles

---

## Algorithme 8: Algorithme du tri bulles

---

**Entrées :** tab : entier[]

**Données :** i, j, n : entier

$n \leftarrow \text{taille(tab)};$

**pour**  $i \in [n - 1, 1]$  **faire**

**pour**  $j \in [0, i - 1]$  **faire**

**si**  $\text{tab}[j] > \text{tab}[j + 1]$  **alors**

            inverser( $\text{tab}[j], \text{tab}[j + 1]$ );

**fin**

**fin**

**fin**

---

# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- **Tri par insertion**
- Tri shell
- Autres tris

# Principe du tri par insertion

- Principe : découper le tableau en deux parties

5	2	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée

5	2	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée

5	2	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé)

5	2	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée

5	2	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée

2	5	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée
- Continuer en agrandissant la partie triée

2	5	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée
- Continuer en agrandissant la partie triée

2	5	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée
- Continuer en agrandissant la partie triée

2	5	6	3	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée
- Continuer en agrandissant la partie triée

2	3	5	6	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée
- Continuer en agrandissant la partie triée

2	3	5	6	1
---	---	---	---	---

# Principe du tri par insertion

- Principe : découper le tableau en deux parties
- Une partie triée
- Une partie non triée
- Insérer le premier nombre non trié (clé) dans la partie triée
- Continuer en agrandissant la partie triée

1	2	3	5	6
---	---	---	---	---

# algorithme

---

## Algorithme 9: Algorithme du tri par insertion

---

**Entrées** : tab : entier[]

**Données** : i, j, n, cle : entier

$n \leftarrow \text{taille(tab)}$  ;

**pour**  $i \in [1, n - 1]$  **faire**

# algorithme

---

## Algorithme 10: Algorithme du tri par insertion

---

**Entrées** : tab : entier[]

**Données** : i, j, n, cle : entier

$n \leftarrow \text{taille}(\text{tab})$  ;

**pour**  $i \in [1, n - 1]$  **faire**

$\text{cle} \leftarrow \text{tab}[i]$  // valeur à insérer dans la partie triée

$j \leftarrow i;$

# algorithme

---

## Algorithme 11: Algorithme du tri par insertion

---

**Entrées** : tab : entier[]

**Données** : i, j, n, cle : entier

$n \leftarrow \text{taille(tab)}$  ;

**pour**  $i \in [1, n - 1]$  **faire**

$\text{cle} \leftarrow \text{tab}[i]$  // valeur à insérer dans la partie triée  
 $j \leftarrow i;$

**tant que**  $j > 0$  **et**  $\text{tab}[j - 1] > \text{cle}$  **faire**

$\text{tab}[j] \leftarrow \text{tab}[j - 1]$  // décaler l'élément j d'un cran à  
droite

décrémenter  $j$  ;

# algorithme

---

## Algorithme 12: Algorithme du tri par insertion

---

**Entrées** : tab : entier[]

**Données** : i, j, n, cle : entier

$n \leftarrow \text{taille}(\text{tab})$  ;

**pour**  $i \in [1, n - 1]$  **faire**

$\text{cle} \leftarrow \text{tab}[i]$  // valeur à insérer dans la partie triée  
 $j \leftarrow i$ ;

**tant que**  $j > 0$  **et**  $\text{tab}[j - 1] > \text{cle}$  **faire**

$\text{tab}[j] \leftarrow \text{tab}[j - 1]$  // décaler l'élément j d'un cran à  
droite

décrémente  $j$  ;

**fin**

$\text{tab}[j] \leftarrow \text{cle}$  // insérer  $\text{cle}$  à sa place

**fin**

---

oooooooooooo

oooooooooooo●oooo

Tri shell

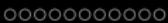
# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- Tri par insertion
- Tri shell**
- Autres tris



Tri shell

# Principe du tri shell

- Proposé en 1959 par Donald L. Shell



Tri shell

# Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé



# Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau



# Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau
- Idée : faire un “pré-tri” avec un pas grossier  $h$



# Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau
- Idée : faire un “pré-tri” avec un pas grossier  $h$
- puis affiner le tri en diminuant  $h$



# Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau
- Idée : faire un “pré-tri” avec un pas grossier  $h$
- puis affiner le tri en diminuant  $h$
- finir avec  $h = 1$  (tri par insertion)



Tri shell

## Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau
- Idée : faire un “pré-tri” avec un pas grossier  $h$
- puis affiner le tri en diminuant  $h$
- finir avec  $h = 1$  (tri par insertion)
- Choix de la suite  $h_n$  important



# Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau
- Idée : faire un “pré-tri” avec un pas grossier  $h$
- puis affiner le tri en diminuant  $h$
- finir avec  $h = 1$  (tri par insertion)
- Choix de la suite  $h_n$  important
- exemple de suite :  $u_0 = 1; u_{n+1} = 3u_n + 1$ ,  $h_n$  défini par



## Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau
- Idée : faire un “pré-tri” avec un pas grossier  $h$
- puis affiner le tri en diminuant  $h$
- finir avec  $h = 1$  (tri par insertion)
- Choix de la suite  $h_n$  important
- exemple de suite :  $u_0 = 1; u_{n+1} = 3u_n + 1$ ,  $h_n$  défini par
  - $h_0 =$  le plus grand  $u_n$  tel que  $u_n < N$  ( $N$  : taille du tableau)



# Principe du tri shell

- Proposé en 1959 par Donald L. Shell
- Tri par insertion optimisé
- Inconvénient du tri par insertion : insérer une clé en début de tableau
- Idée : faire un “pré-tri” avec un pas grossier  $h$
- puis affiner le tri en diminuant  $h$
- finir avec  $h = 1$  (tri par insertion)
- Choix de la suite  $h_n$  important
- exemple de suite :  $u_0 = 1; u_{n+1} = 3u_n + 1$ ,  $h_n$  défini par
  - $h_0 =$  le plus grand  $u_n$  tel que  $u_n < N$  ( $N$  : taille du tableau)
  - $h_{n+1} = \frac{h_n - 1}{3}$

# Algorithme du tri shell

---

## Algorithme 13: Tri Shell

---

**Entrées :** tab : entier[]

**Données :** i, j, n, cle,h : entier

$n \leftarrow \text{taille(tab)} ;$

$h \leftarrow 1 ;$

**tant que**  $3h + 1 < n // \text{calcul de } h_0$

**faire**

    |  $h \leftarrow 3h + 1 ;$

**fin**

...

---

# Algorithme du tri shell

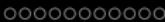
---

## Algorithme 14: Tri Shell

---

tant que  $h > 0$  faire





# Algorithme du tri shell

## Algorithme 15: Tri Shell

```
tant que h > 0 faire
    pour i ∈ [h, n – 1] faire
        cle ← tab[i] // valeur à insérer dans la partie triée
        j ← i;
        tant que j ≥ h et tab[j – h] > cle faire
            tab[j] ← tab[j – h] // décaler de h crans à droite
            j ← j – h ;
        fin
```



# Algorithme du tri shell

---

## Algorithme 16: Tri Shell

---

```
tant que h > 0 faire
    pour i ∈ [h, n – 1] faire
        cle ← tab[i] // valeur à insérer dans la partie triée
        j ← i;
        tant que j ≥ h et tab[j – h] > cle faire
            tab[j] ← tab[j – h] // décaler de h crans à droite
            j ← j – h ;
        fin
        tab[j] ← cle // insérer cle à sa place
    fin
```



# Algorithme du tri shell

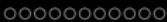
---

## Algorithme 17: Tri Shell

---

```
tant que h > 0 faire
    pour i ∈ [h, n – 1] faire
        cle ← tab[i] // valeur à insérer dans la partie triée
        j ← i;
        tant que j ≥ h et tab[j – h] > cle faire
            tab[j] ← tab[j – h] // décaler de h crans à droite
            j ← j – h ;
        fin
        tab[j] ← cle // insérer cle à sa place
    fin
    h ← h/3 // équivalent à (h-1)/3
fin
```

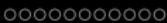
---



Tri shell

## Étude du tri shell

- Complexité dépend de la suite  $h_n$  utilisée



## Étude du tri shell

- Complexité dépend de la suite  $h_n$  utilisée
- Meilleure suite inconnue



## Étude du tri shell

- Complexité dépend de la suite  $h_n$  utilisée
- Meilleure suite inconnue
- Complexités dans le pire des cas connues :



Tri shell

# Étude du tri shell

- Complexité dépend de la suite  $h_n$  utilisée
- Meilleure suite inconnue
- Complexités dans le pire des cas connues :
  - $h_i = 2^{i+1} - 1 \rightsquigarrow \Theta(n^{\frac{3}{2}})$



Tri shell

## Étude du tri shell

- Complexité dépend de la suite  $h_n$  utilisée
- Meilleure suite inconnue
- Complexités dans le pire des cas connues :
  - $h_i = 2^{i+1} - 1 \rightsquigarrow \Theta(n^{\frac{3}{2}})$
- Complexités moyennes connues :



# Étude du tri shell

- Complexité dépend de la suite  $h_n$  utilisée
- Meilleure suite inconnue
- Complexités dans le pire des cas connues :
  - $h_i = 2^{i+1} - 1 \rightsquigarrow \Theta(n^{\frac{3}{2}})$
- Complexités moyennes connues :
  - $h_i = 2^{i+1} - 1 \rightsquigarrow \Theta(n^{\frac{3}{2}})$



# Étude du tri shell

- Complexité dépend de la suite  $h_n$  utilisée
- Meilleure suite inconnue
- Complexités dans le pire des cas connues :
  - $h_i = 2^{i+1} - 1 \rightsquigarrow \Theta(n^{\frac{3}{2}})$
- Complexités moyennes connues :
  - $h_i = 2^{i+1} - 1 \rightsquigarrow \Theta(n^{\frac{3}{2}})$
  - $h_i = 2^p 3^q \rightsquigarrow \Theta(n \log^2 n)$

# Sommaire

## 1 Les tableaux

- Déclaration
- Remplissage, mélange d'un tableau

## 2 Algorithmes de tris

- Tri par sélection
- Tri bulles
- Tri par insertion
- Tri shell
- Autres tris



Autres tris

## Autres tris

Il existe d'autres tris plus performants mais plus complexes.

- tri par segmentation (quicksort)
- tri par partition/fusion
- tri par arbre binaire de recherche équilibré
- tri par tas

Exemples :

<https://moodle.ensieta.fr/course/view.php?id=548>