Elements d'Algorithmique
CMTD3: Tri par sélection

Christine Tasson
Université de Paris, IRIF

Algorithmes de tris

Données : Tableau `T` contenant des éléments que l'on peut ordonner (entiers, paires d'entiers, chaînes de caractères,...)

Algorithme : Plusieurs possibilités (Sélection, Insertion, Fusion, ...)

Sortie : Un tableau **ordonné** et qui contient exactement les **mêmes éléments** que ceux de `T`

Efficacité des algorithmes de tri

Des **simulations** de différents tris peuvent être visualisées à l'adresse suivante :

Simulations des Algorithmes de Tris

Ces simulations donnent une indication sur l'efficacité des tris. Nous allons maintenant calculer leur **complexité théorique**.

Dans ce cours : Complexité des Tris

La complexité d'un algorithme est le nombre d'opérations élémentaires pour résoudre le problème.

Opérations élémentaires pour un tableau `t`

- Affectations : `t[i] = 5`
- Comparaisons : `t[i] < t[j]`

Comparaison de deux algorithmes

- Tri par sélection
- Tri par insertion

Tri par sélection

Tri par sélection - Principe

Le tri par sélection consiste à échanger le minimum des éléments non triés pour le placer à la suite des éléments triés.

10, **1**, 5, 19, 3, 3

1, 10, 5, 19, **3**, 3

1, 3, 5, 19, 10, **3**

1, 3, 3, 19, 10, **5**

1, 3, 3, 5, **10**, 19

1, 3, 3, 5, 10, 19

Tri par sélection : Pseudocode/Complexité

Entrée : tableau `T`

```
fonction triParSélection(T)
    n ← longueur de T
    pour i ← 0 à n − 2 faire
        trouver l'indice min du plus
            petit élément parmi T[i+1,..., n-1]
        échanger T[i] et T[min]
```

Nombres d'opérations élémentaires = nombre de comparaisons

Tri par sélection : Pseudocode/Complexité

Entrée : tableau `T`

fonction TRIPARSÉLECTION(`T`)
    $n \leftarrow$ longueur de `T`
    pour $i \leftarrow 0$ à $n-2$ **faire**
        trouver l'indice `min` du plus petit élément parmi `T`[i+1,. . . , n-1]
        échanger `T`[i] et `T`[`min`]

Nombres d'opérations élémentaires = nombre de comparaisons

- Trouver l'indice du `min` dans un tableau de longueur `n-i-1`.

Tri par sélection : Pseudocode/Complexité

Entrée : tableau `T`

```
fonction triParSélection(T)
    n ← longueur de T
    pour i ← 0 à n − 2 faire
        trouver l'indice min du plus
            petit élément parmi T[i+1,. . . , n-1]
        échanger T[i] et T[min]
```

Nombres d'opérations élémentaires = nombre de comparaisons

- Trouver l'indice du `min` dans un tableau de longueur `n-i-1`. Nécessite `n-i-1` comparaisons

Tri par sélection : Pseudocode/Complexité

Entrée : tableau `T`

```
fonction triParSélection(T)
    n ← longueur de T
    pour i ← 0 à n − 2 faire
        trouver l'indice min du plus
            petit élément parmi T[i+1,. . . , n-1]
        échanger T[i] et T[min]
```

Nombres d'opérations élémentaires = nombre de comparaisons

- Trouver l'indice du `min` dans un tableau de longueur `n-i-1`. Nécessite `n-i-1` comparaisons

On pose, $m = n - 2$, $$\sum_{i=0}^{m}(m-i) =$$

Tri par sélection : Pseudocode/Complexité

Entrée : tableau `T`

```
fonction triParSélection(T)
    n ← longueur de T
    pour i ← 0 à n − 2 faire
        trouver l'indice min du plus
          petit élément parmi T[i+1,. . . , n-1]
        échanger T[i] et T[min]
```

Nombres d'opérations élémentaires = nombre de comparaisons

- Trouver l'indice du `min` dans un tableau de longueur `n-i-1`. Nécessite `n-i-1` comparaisons

On pose, $m = n - 2$, $$\sum_{i=0}^{m}(m-i) = \sum_{\ell=0}^{m}\ell$$

Tri par sélection : Pseudocode/Complexité

Entrée : tableau `T`

```
fonction triParSélection(T)
    n ← longueur de T
    pour i ← 0 à n − 2 faire
        trouver l'indice min du plus
            petit élément parmi T[i+1,..., n-1]
        échanger T[i] et T[min]
```

Nombres d'opérations élémentaires = nombre de comparaisons

- Trouver l'indice du `min` dans un tableau de longueur `n-i-1`. Nécessite `n-i-1` comparaisons

On pose, $m = n - 2$, $$\sum_{i=0}^{m}(m-i) = \sum_{\ell=0}^{m}\ell = \frac{(m-1)m}{2}$$

Avec un nombre d'opérations élémentaires proportionnel à n^2, la complexité du tri par sélection **quadratique** et ne dépend pas du tableau de taille n passé en argument.

Tri par insertion

Tri par insertion - Principe

Le tri par insertion consiste à

- garder le début du tableau trié
- y insérer successivement, à leur place, les éléments non-triés.

10, **1**, 5, 19, 3, 3
1, 10, 5, 19, 3, 3
1, 10, **5**, 19, 3, 3
1, **5**, 10, 19, 3, 3
1, 5, 10, **19**, 3, 3
1, 5, 10, 19, **3**, 3
1, 5, 10, **3**, 19, 3
1, 5, **3**, 10, 19, 3
1, **3**, 5, 10, 19, 3
1, 3, 5, 10, 19, **3**
1, 3, 5, 10, **3**, 19
1, 3, 5, **3**, 10, 19
1, 3, **3**, 5, 10, 19

Tri par insertion : Pseudocode/Complexité

Entrée : tableau T

fonction TRIPARTIEL(T, i)
    $j \leftarrow i$
    tant que $j > 0$ et $T[j] < T[j-1]$ **faire**
        échanger $T[j-1]$ et $T[j]$
fonction TRIPARINSERTION(T)
    $n \leftarrow$ longueur de T
    pour $i \leftarrow 1$ à $n-1$ **faire**
        TRIPARTIEL(T,i)

Nombres d'opérations élémentaires= nombre de comparaisons

Tri par insertion : Pseudocode/Complexité

Entrée : tableau T

```
fonction triPartiel(T, i)
    j ← i
    tant que j > 0 et T[j] < T[j − 1] faire
        échanger T[j − 1] et T[j]
fonction triParInsertion(T)
    n ← longueur de T
    pour i ← 1 à n − 1 faire
        triPartiel(T,i)
```

Nombres d'opérations élémentaires= nombre de comparaisons

Appliquer triPartiel(T,i)

Tri par insertion : Pseudocode/Complexité

Entrée : tableau T

fonction TRIPARTIEL(T, i)
  $j \leftarrow i$
  tant que $j > 0$ et $T[j] < T[j-1]$ **faire**
    échanger $T[j-1]$ et $T[j]$
fonction TRIPARINSERTION(T)
  $n \leftarrow$ longueur de T
  pour $i \leftarrow 1$ à $n-1$ **faire**
    TRIPARTIEL(T,i)

Nombres d'opérations élémentaires= nombre de comparaisons

Appliquer TRIPARTIEL(T,i)

- Si $T[i-1] < T[i]$, alors 1 comparaison (meilleur des cas)

Tri par insertion : Pseudocode/Complexité

Entrée : tableau T

fonction TRIPARTIEL(T, i)
    $j \leftarrow i$
    tant que $j > 0$ et $T[j] < T[j-1]$ **faire**
        échanger $T[j-1]$ et $T[j]$
fonction TRIPARINSERTION(T)
    $n \leftarrow$ longueur de T
    pour $i \leftarrow 1$ à $n-1$ **faire**
        TRIPARTIEL(T,i)

Nombres d'opérations élémentaires= nombre de comparaisons

Appliquer TRIPARTIEL(T,i)

- Si $T[i-1] < T[i]$, alors 1 comparaison (meilleur des cas)
- Si $T[i] < T[0]$, alors i comparaisons (pire des cas)

Tri par insertion : Pseudocode/Complexité

Entrée : tableau $\mathtt{T}$

```
fonction triPartiel(T, i)
    j ← i
    tant que j > 0 et T[j] < T[j − 1] faire
        échanger T[j − 1] et T[j]
fonction triParInsertion(T)
    n ← longueur de T
    pour i ← 1 à n − 1 faire
        triPartiel(T,i)
```

Nombres d'opérations élémentaires= nombre de comparaisons

Appliquer $\textsc{triPartiel}(\mathtt{T},i)$

- Si $\mathtt{T}[i-1] < \mathtt{T}[i]$, alors 1 comparaison (meilleur des cas)
- Si $\mathtt{T}[i] < \mathtt{T}[0]$, alors i comparaisons (pire des cas)

Pour la boucle, on somme :

- Dans le **pire des cas**, la complexité est quadratique : $\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$

Tri par insertion : Pseudocode/Complexité

Entrée : tableau T

```
fonction TRIPARTIEL(T, i)
    j ← i
    tant que j > 0 et T[j] < T[j − 1] faire
        échanger T[j − 1] et T[j]
fonction TRIPARINSERTION(T)
    n ← longueur de T
    pour i ← 1 à n − 1 faire
        TRIPARTIEL(T,i)
```

Nombres d'opérations élémentaires= nombre de comparaisons

Appliquer TRIPARTIEL(T,i)

- Si $\mathtt{T}[i-1] < \mathtt{T}[i]$, alors 1 comparaison (meilleur des cas)
- Si $\mathtt{T}[i] < \mathtt{T}[0]$, alors i comparaisons (pire des cas)

Pour la boucle, on somme :

- Dans le **pire des cas**, la complexité est quadratique : $\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$
- Dans le **meilleur des cas**, la complexité est linéaire.

En résumé

Complexité des algorithmes de tris

- Le tri par sélection est quadratique
- Le tri par insertion est quadratique dans le pire des cas, mais linéaire dans le meilleur des cas
- Il existe des algorithmes de tri (tri fusion) en $n \log(n)$
- On ne peut pas résoudre le problème du tri avec une meilleur complexité que $n \log(n)$.