Éléments d'Algorithmique
CMTD3: Complexité des Tris par sélection et par insertion

Université de Paris

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`)
    $n \leftarrow$ longueur de `T`
    pour $i \leftarrow 0$ à $n-2$ **faire**
        trouver l'indice `min` du plus
            petit élément parmi `T`[i,. . . , 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 ← longueur de T
    pour i ← 0 à n − 2 faire
        trouver l'indice min du plus
            petit élément parmi T[i,..., 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`.

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,..., 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`. Nécessite `n-i-1` 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,. . . , n-1]
        échanger `T`[i] et `T`[`min`]

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

Nombres d'opérations élémentaires = nombre de comparaisons

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

$$\sum_{i=0}^{n-2}(n-i-1) = (n-1) + \ldots + 1 = \sum_{\ell=1}^{n-1} \ell$$

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,..., 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`. Nécessite `n-i-1` comparaisons

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

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,..., 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`. Nécessite `n-i-1` comparaisons

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

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

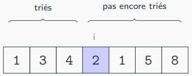
Tri par insertion

Tri par insertion - Principe

1. On parcourt le tableau T à trier de gauche à droite.

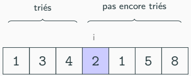
Tri par insertion - Principe

1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

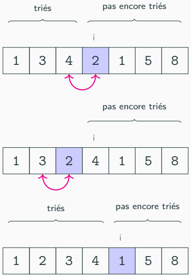


Tri par insertion - Principe

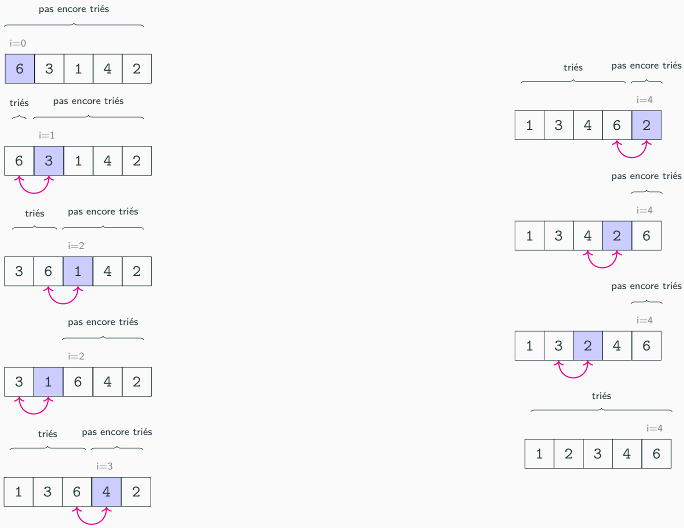
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.



3. On insère l'élément d'indice i dans le sous-tableau trié en l'échangeant avec l'élément précédent tant que celui-ci est plus grand.



Exemple



pas encore triés
i=0
6 3 1 4 2
triés
pas encore triés
i=1
6 3 1 4 2
triés
pas encore triés
i=2
3 6 1 4 2
pas encore triés
i=2
3 1 6 4 2
triés
pas encore triés
i=3
1 3 6 4 2
triés
pas encore triés
i=4
1 3 4 6 2
pas encore triés
i=4
1 3 4 2 6
pas encore triés
i=4
1 3 2 4 6
triés
i=4
1 2 3 4 6

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]
        j ← j − 1
fonction triParInsertion(T)
    n ← longueur de T
    pour i ← 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)
  $j \leftarrow i$
  tant que $j > 0$ et T[j] $<$ T[$j-1$] **faire**
    échanger T[$j-1$] et T[j]
    $j \leftarrow j - 1$
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)

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]
        j ← j − 1
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 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 $T[j]$
        $j \leftarrow j-1$
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 T

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

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

Tri Partiel - Correction

Entrée : tableau $\mathtt{T}$

fonction $\textsc{triPartiel}(\mathtt{T}, i)$
    $j \leftarrow i$
    tant que $j > 0$ et $\mathtt{T}[j] < \mathtt{T}[j-1]$ **faire**
        échanger $\mathtt{T}[j-1]$ et $\mathtt{T}[j]$
        $j \leftarrow j - 1$

Tri Partiel - Correction

Entrée : tableau $\mathtt{T}$ trié sur $T[0 \ldots i-1]$

fonction $\textsc{triPartiel}(\mathtt{T}, i)$
    $j \leftarrow i$
    tant que $j > 0$ et $\mathtt{T}[j] < \mathtt{T}[j-1]$ **faire**
        échanger $\mathtt{T}[j-1]$ et $\mathtt{T}[j]$
        $j \leftarrow j - 1$

Tri Partiel - Correction

Entrée : tableau `T` trié sur $T[0 \ldots i-1]$

```
fonction triPartiel(T, i)
    j ← i
    tant que j > 0 et T[j] < T[j − 1] faire
        échanger T[j − 1] et T[j]
        j ← j − 1
```

But : Prouver qu'à la fin de `triPartiel`, le tableau $T[0 \ldots i]$ est trié.

Invariant de Boucle

Tri Partiel - Correction

Entrée : tableau $\mathtt{T}$ trié sur $T[0 \ldots i-1]$

```
fonction triPartiel(T, i)
    j ← i
    tant que j > 0 et T[j] < T[j − 1] faire
        échanger T[j − 1] et T[j]
        j ← j − 1
```

But : Prouver qu'à la fin de `triPartiel`, le tableau $T[0 \ldots i]$ est trié.

Invariant de Boucle (preuve au tableau) :

Soit T_0 le tableau au départ. À chaque passage dans la boucle **tant que** :

- $T[j] = T_0[i]$
- Les valeurs dans $T_0[0 \ldots i-1]$ sont dans $T[0 \ldots i]$, et leur ordre ne change pas
- $T[j]$ est plus petit que les valeurs dans $T[j+1 \ldots i]$

Tri Partiel - Correction

But : Prouver qu'à la fin de `triPartiel`, le tableau $T[0 \dots i]$ est trié.

Invariant de Boucle (preuve au tableau) :

Soit T_0 le tableau au départ. À chaque passage dans la boucle **tant que** :

- $T[j] = T_0[i]$
- Les valeurs dans $T_0[0 \dots i-1]$ sont dans $T[0 \dots i]$, et leur ordre ne change pas
- $T[j]$ est plus petit que les valeurs dans $T[j+1 \dots i]$

Conclusion, $T[0 \dots i]$ est trié car :

- On a gardé les valeurs
- Les valeurs de $T_0[0 \dots i-1]$ restent triées entre elles
- $T_0[i]$ est placé au bon endroit j dans le tableau car $T[j]$ est plus petit que les valeurs dans $T[j+1 \dots i]$ et qu'à la sortie de la boucle, soit $j = 0$ et il n'y a pas d'éléments avant $T[j]$, soit $T[j] \geq T[j-1]$ et, $T[0 \dots j-1]$ étant trié par hypothèse, cela signifie que $T[0 \dots j]$ est trié.

Tri par Insertion - Correction

Entrée : tableau `T`

```
fonction triParInsertion(T)
    n ← longueur de T
    pour i ← 1 à n − 1 faire
        triPartiel(T,i)
```

Si $T[0 \ldots i-1]$ est trié, alors `triPartiel`(T, i) trie $T[0 \ldots i]$.

Tri par Insertion - Correction

Entrée : tableau T

fonction TRIPARINSERTION(T)
    $n \leftarrow$ longueur de T
    pour $i \leftarrow 1$ à $n-1$ **faire**
        TRIPARTIEL(T,i)

Si $T[0 \ldots i-1]$ est trié, alors `triPartiel`(T, i) trie $T[0 \ldots i]$.

Par **récurrence**, après i appels de la fonction `triPartiel`, T est trié sur les cases $[0 \ldots i]$.

- **Initialisation** : Avant le premier appel, la première case est bien triée.
- **Hérédité** : Si $T[0 \ldots i]$ est trié, on sait que `triPartiel`$(T, i+1)$ trie $T[0 \ldots i+1]$.

Tri par Insertion - Correction

Entrée : tableau `T`

fonction $\textsc{triParInsertion}(\texttt{T})$
    $n \leftarrow$ longueur de `T`
    pour $i \leftarrow 1$ à $n-1$ **faire**
        $\textsc{triPartiel}(\texttt{T}, i)$

Si $T[0\ldots i-1]$ est trié, alors `triPartiel`(T, i) trie $T[0\ldots i]$.

Par **récurrence**, après i appels de la fonction `triPartiel`, T est trié sur les cases $[0\ldots i]$.

- **Initialisation** : Avant le premier appel, la première case est bien triée.
- **Hérédité** : Si $T[0\ldots i]$ est trié, on sait que `triPartiel`$(T, i+1)$ trie $T[0\ldots i+1]$.

Conclusion : `triParInsertion` trie les tableaux.

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 $n \log(n)$.