

Elements d'Algorithmique

CMTD2: Tri par sélection

Daniela Petrişan
Université de Paris, IRIF

Le problème de tri

Le problème de tri est un problème fondamental en informatique. Le tri est souvent utilisé en pré-traitement dans de nombreux algorithmes. Par exemple :

Le problème de tri

Le problème de tri est un problème fondamental en informatique. Le tri est souvent utilisé en pré-traitement dans de nombreux algorithmes. Par exemple :

- Pour rechercher des informations, nous pouvons effectuer une recherche dichotomique sur un **tableau déjà trié**.

Le problème de tri

Le problème de tri est un problème fondamental en informatique. Le tri est souvent utilisé en pré-traitement dans de nombreux algorithmes. Par exemple :

- Pour rechercher des informations, nous pouvons effectuer une recherche dichotomique sur un **tableau déjà trié**.
- L'algorithme de Prim et l'algorithme de Dijkstra sont des algorithmes classiques qui traitent les graphes. Les files de priorité jouent un rôle fondamental permettant des algorithmes efficaces.

Le problème de tri

Le problème de tri est un problème fondamental en informatique. Le tri est souvent utilisé en pré-traitement dans de nombreux algorithmes. Par exemple :

- Pour rechercher des informations, nous pouvons effectuer une recherche dichotomique sur un **tableau déjà trié**.
- L'algorithme de Prim et l'algorithme de Dijkstra sont des algorithmes classiques qui traitent les graphes. Les files de priorité jouent un rôle fondamental permettant des algorithmes efficaces.
- Les algorithmes de traitement des chaînes de caractères sont souvent basés sur le tri.

Le problème de tri

Le problème de tri est un problème fondamental en informatique. Le tri est souvent utilisé en pré-traitement dans de nombreux algorithmes. Par exemple :

- Pour rechercher des informations, nous pouvons effectuer une recherche dichotomique sur un **tableau déjà trié**.
- L'algorithme de Prim et l'algorithme de Dijkstra sont des algorithmes classiques qui traitent les graphes. Les files de priorité jouent un rôle fondamental permettant des algorithmes efficaces.
- Les algorithmes de traitement des chaînes de caractères sont souvent basés sur le tri.
- Le calcul scientifique se préoccupe souvent de la précision (à quel point sommes-nous proches de la réponse correcte?). Certains algorithmes numériques utilisent des files de priorité pour contrôler la précision des calculs.

Le problème de tri

De nombreuses techniques et algorithmes ont été élaborées pour optimiser le tri.



A word cloud of various sorting algorithms. The words are arranged in a roughly triangular shape, with 'Shell cocktail' at the top, 'sélection' in the middle, and 'arborescent' at the bottom. The words are in different colors (orange, brown, blue) and sizes, indicating their relative frequency or importance. The algorithms listed include: Shell cocktail, sélection, Introsort, bulles, pair-impair, rapide, insertion, tas, fusion, Timsort, Smoothsort, and arborescent.

Shell cocktail
sélection
Introsort bulles
pair-impair
rapide insertion tas
fusion
Timsort
Smoothsort
arborescent

Dans ce cours : L'algorithme de tri par sélection

Le **tri par sélection** résout le problème de tri d'une séquence de nombres en ordre croissant. L'algorithme recherche le minimum parmi les éléments non triés pour le placer à la suite des éléments déjà triés.

Dans ce cours : L'algorithme de tri par sélection

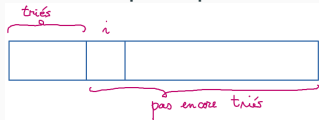
Le **tri par sélection** résout le problème de tri d'une séquence de nombres en ordre croissant. L'algorithme recherche le minimum parmi les éléments non triés pour le placer à la suite des éléments déjà triés.

- Le principe
- Le pseudocode
- Correction de l'algorithme
- Propriétés

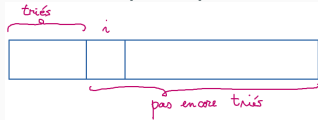
Le principe de l'algorithme de Tri par Sélection

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

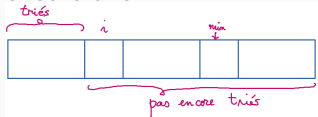
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



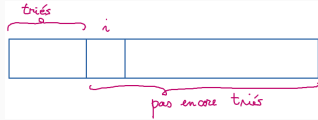
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



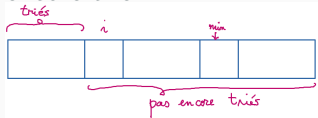
3. Trouver le plus petit élément du sous-tableau non encore trié.



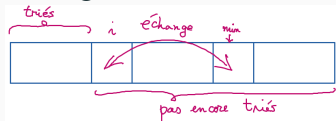
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



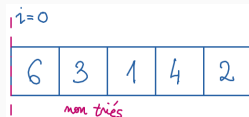
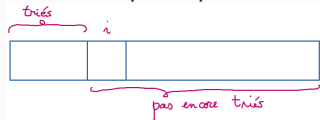
3. Trouver le plus petit élément du sous-tableau non encore trié.



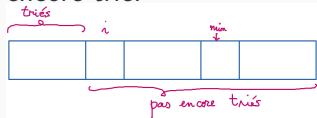
4. Echanger ce élément avec $T[i]$.



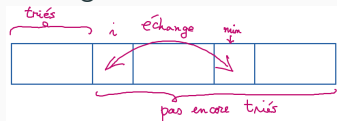
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



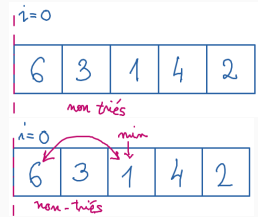
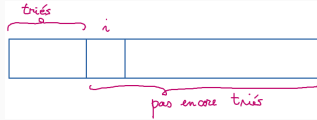
3. Trouver le plus petit élément du sous-tableau non encore trié.



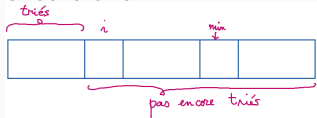
4. Echanger ce élément avec $T[i]$.



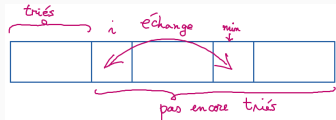
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



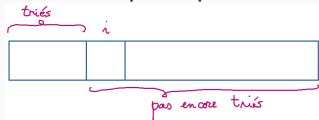
3. Trouver le plus petit élément du sous-tableau non encore trié.



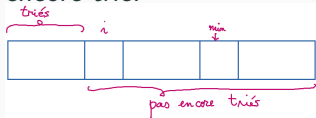
4. Echanger ce élément avec $T[i]$.



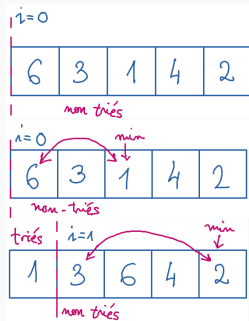
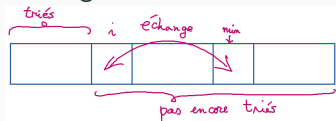
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



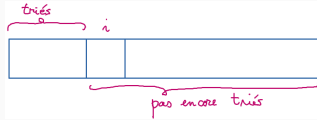
3. Trouver le plus petit élément du sous-tableau non encore trié.



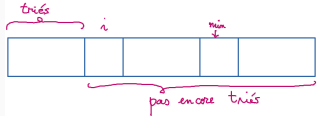
4. Echanger ce élément avec $T[i]$.



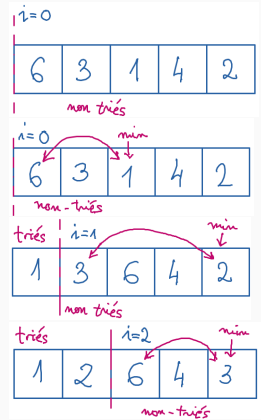
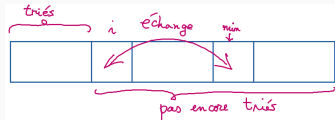
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



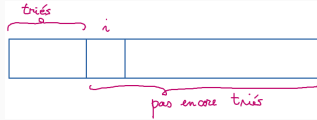
3. Trouver le plus petit élément du sous-tableau non encore trié.



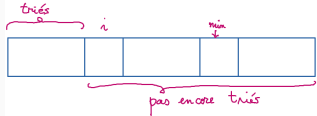
4. Echanger ce élément avec $T[i]$.



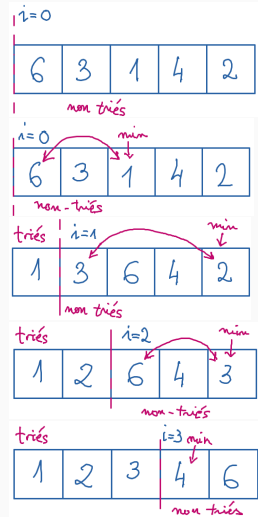
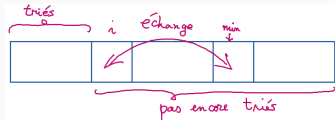
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



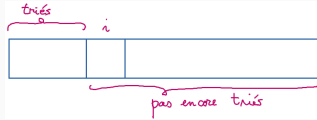
3. Trouver le plus petit élément du sous-tableau non encore trié.



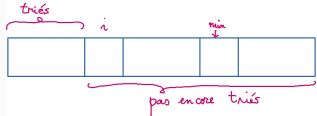
4. Echanger ce élément avec $T[i]$.



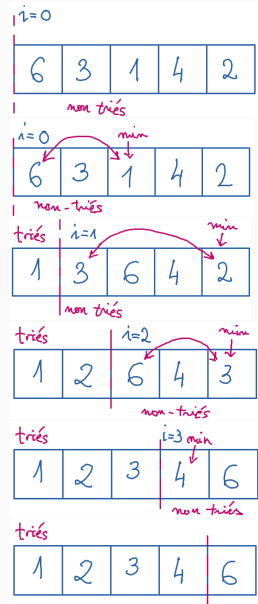
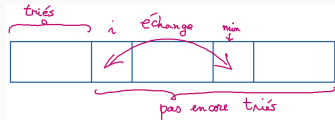
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i , les éléments qui le précèdent sont déjà triés.



3. Trouver le plus petit élément du sous-tableau non encore trié.



4. Echanger ce élément avec $T[i]$.



Tri par Sélection

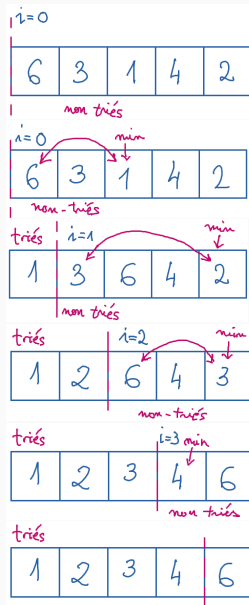
pseudocode

Entrée : tableau T

1: **fonction** TRIPARSÉLECTION(T)

2: $n \leftarrow$ longueur de T

3: **pour** $i \leftarrow 0$ à $n - 2$ **faire**



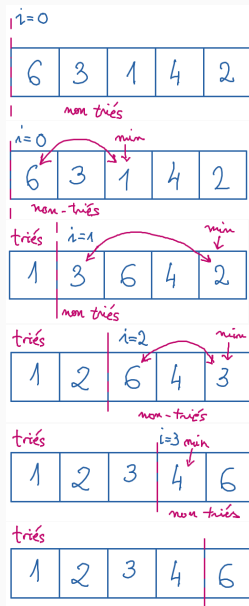
Entrée : tableau T

1: **fonction** TRIPARSÉLECTION(T)

2: $n \leftarrow$ longueur de T

3: **pour** $i \leftarrow 0$ à $n - 2$ **faire**

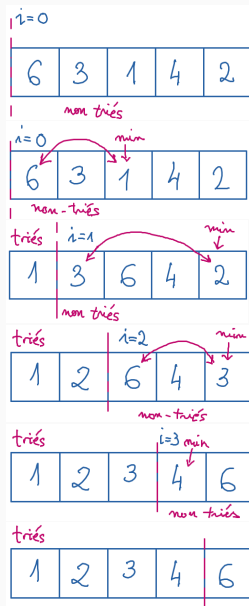
► Remarquez qu'il suffit de parcourir les premiers $n - 1$ éléments du tableau. Le dernier élément est automatiquement le maximum à la fin de l'exécution.



Entrée : tableau T

- 1: **fonction** TRIPARSÉLECTION(T)
- 2: $n \leftarrow$ longueur de T
- 3: **pour** $i \leftarrow 0$ à $n - 2$ **faire**
 trouver l'indice min du plus
 petit élément parmi $T[i+1, \dots, n-1]$
- 4:

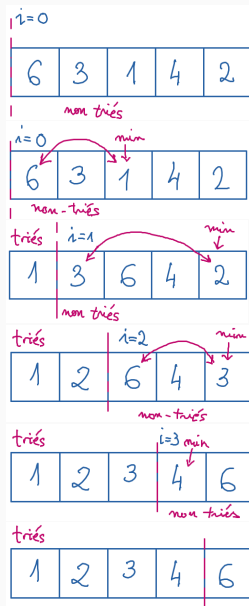
▷ Remarquez qu'il suffit de parcourir les premiers $n - 1$ éléments du tableau. Le dernier élément est automatiquement le maximum à la fin de l'exécution.



Entrée : tableau T

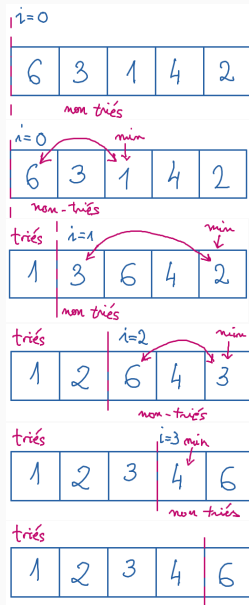
- 1: **fonction** TRIPARSÉLECTION(T)
- 2: $n \leftarrow$ longueur de T
- 3: **pour** $i \leftarrow 0$ à $n - 2$ **faire**
 trouver l'indice min du plus
 petit élément parmi $T[i+1, \dots, n-1]$
- 5: échanger $T[i]$ et $T[\text{min}]$

► Remarquez qu'il suffit de parcourir les premiers $n - 1$ éléments du tableau. Le dernier élément est automatiquement le maximum à la fin de l'exécution.



Entrée : tableau T

```
1: fonction TRIPARSÉLECTION(T)
2:    $n \leftarrow$  longueur de T
3:   pour  $i \leftarrow 0$  à  $n - 2$  faire
4:      $\text{min} \leftarrow i$ 
5:     pour  $j \leftarrow i + 1$  à  $n - 1$  faire
6:       si  $T[j] < T[\text{min}]$  alors
7:          $\text{min} \leftarrow j$ 
8:       échanger  $T[i]$  et  $T[\text{min}]$ 
```



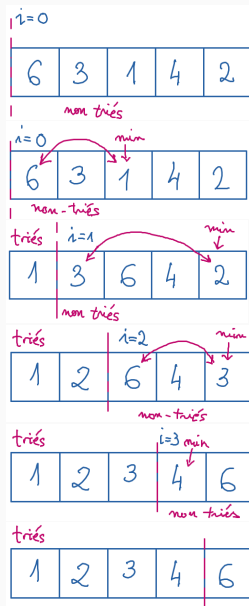
Correction

Entrée : tableau T

```
1: fonction TRIPARSÉLECTION(T)
2:    $n \leftarrow$  longueur de T
3:   pour  $i \leftarrow 0$  à  $n - 2$  faire
4:      $\text{min} \leftarrow i$ 
5:     pour  $j \leftarrow i + 1$  à  $n - 1$  faire
6:       si  $T[i] > T[j]$  alors
7:          $\text{min} \leftarrow j$ 
8:       échanger  $T[i]$  et  $T[\text{min}]$ 
```

Preuve de correction :

Idée : On trouve un **invariant de boucle**, c.-à-d., une propriété qui est vraie avant de commencer la boucle et reste correcte après chaque itération de la boucle.

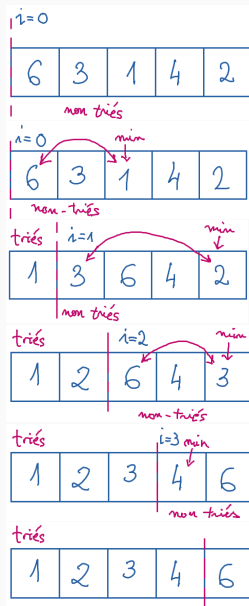


Entrée : tableau T

```
1: fonction TRIPARSÉLECTION( $T$ )  
2:    $n \leftarrow$  longueur de  $T$   
3:   pour  $i \leftarrow 0$  à  $n - 2$  faire  
4:      $\text{min} \leftarrow i$   
5:     pour  $j \leftarrow i + 1$  à  $n - 1$  faire  
6:       si  $T[i] > T[j]$  alors  
7:          $\text{min} \leftarrow j$   
8:       échanger  $T[i]$  et  $T[\text{min}]$ 
```

Preuve de correction :

Pour tout i de 0 à $n-2$, à la fin de l'étape i , le tableau est une permutation du tableau initial et les $i+1$ premiers éléments du tableau sont triés et inférieurs ou égaux aux éléments restants.



Preuve de correction (au tableau) :

Invariant de boucle : Pour tout i de 0 à $n - 2$, à la fin de l'étape i ,

1. le tableau est une permutation du tableau initial
2. les $i + 1$ premiers éléments du tableau sont triés
3. les $i + 1$ premiers éléments du tableau sont inférieurs ou égaux aux éléments restants.

La preuve est par induction sur i .

Cas de base $i = 0$:

1. est vrai, car à la fin de cette étape, le nouveau tableau est obtenu simplement en appliquant la transposition de deux éléments.
2. est vrai, car nous n'avons qu'un seul élément.
3. est vrai, car nous avons choisi le minimum du tableau original.

Preuve de correction (au tableau) :

Cas inductif : On suppose la propriété vrai pour $i = k$ et nous la démontrons pour $i = k + 1$.

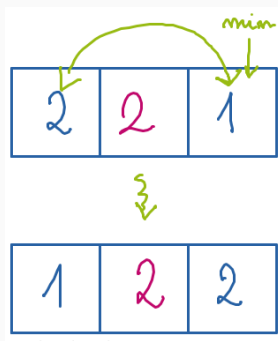
Nous notons par T le tableau à la fin de l'étape k et par T' le tableau à l'étape $k + 1$.

1. est vrai car T' est obtenu à partir du tableau initial en composant la permutation nécessaire pour obtenir T avec une transposition de deux éléments.
2. est vrai car d'un côté $T'[0, \dots, k]$ coïncide avec $T[0, \dots, k]$. Ces éléments sont donc triés en utilisant la partie 2 de l'hypothèse inductive. De l'autre côté, par la partie 3 de l'hypothèse inductive, on sait que tous les éléments de $T[0, \dots, k]$ sont inférieurs ou égaux au reste des éléments de ce tableau, notamment à $T'[k + 1]$. Donc, $T'[0, \dots, k + 1]$ est trié.
3. Pour montrer que les éléments de $T'[0, \dots, k]$ sont inférieurs ou égaux aux éléments de $T'[k + 2, \dots, n - 1]$, nous utilisons la partie 3 de l'hypothèse inductive. Nous avons également que $T'[k + 1]$ est inférieur ou égal aux éléments de $T'[k + 2, \dots, n - 1]$ parce que nous l'avons choisi comme le minimum parmi $T[k + 1, \dots, n - 1]$.

Propriétés

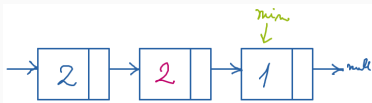
Définition. Un algorithme de tri est dit **stable** s'il préserve l'ordonnancement initial des éléments que l'ordre considère comme égaux.

Le tri par sélection est **instable**. Par exemple :

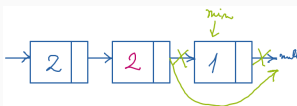
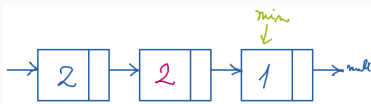


Une manière de rendre le tri de sélection **stable**, est de l'implémenter sur une liste. Au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.

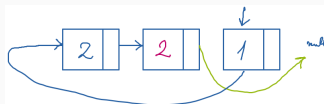
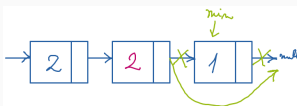
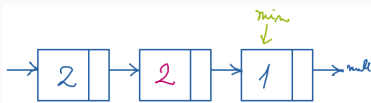
Une manière de rendre le tri de sélection **stable**, est de l'implémenter sur une liste. Au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.



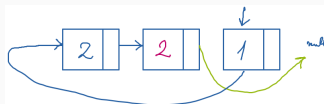
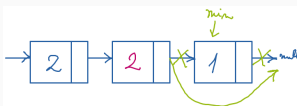
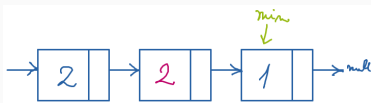
Une manière de rendre le tri de sélection **stable**, est de l'implémenter sur une liste. Au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.



Une manière de rendre le tri de sélection **stable**, est de l'implémenter sur une liste. Au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.



Une manière de rendre le tri de sélection **stable**, est de l'implémenter sur une liste. Au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.



Définition. Un tri est dit **en place** s'il est effectué directement dans la structure de donnée initiale et l'espace supplémentaire nécessaire à l'algorithme ne dépend pas de la taille du tableau.

Définition. Un tri est dit **en place** s'il est effectué directement dans la structure de donnée initiale et l'espace supplémentaire nécessaire à l'algorithme ne dépend pas de la taille du tableau.

Le **tri par sélection** est un tri **en place**.

L'espace supplémentaire nécessaire n'est que l'espace d'un élément du tableau, qui est utilisé pour effectuer l'échange des éléments. (Bien sûr, nous avons besoin d'un petit espace supplémentaire pour les opérations sur les entiers.)

Définition. Un tri est dit **en place** s'il est effectué directement dans la structure de donnée initiale et l'espace supplémentaire nécessaire à l'algorithme ne dépend pas de la taille du tableau.

Le **tri par sélection** est un tri **en place**.

L'espace supplémentaire nécessaire n'est que l'espace d'un élément du tableau, qui est utilisé pour effectuer l'échange des éléments. (Bien sûr, nous avons besoin d'un petit espace supplémentaire pour les opérations sur les entiers.)

Le **tri par sélection** n'effectue que $n - 1$ échanges, donc un nombre d'échanges inférieur à celui d'autres algorithmes tels que le **tri par bulles**; par conséquent, même si les deux méthodes de tri ont la même complexité, le tri par sélection est plus rapide et plus efficace.

Mais il est surpassé par d'autres algorithmes, dont nous parlerons dans les prochains cours !