Elements d'Algorithmique
CMTD2: Tri par sélection

Le problème de tri

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

Algorithme : De nombreuses techniques et algorithmes ont été élaborées

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

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.

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

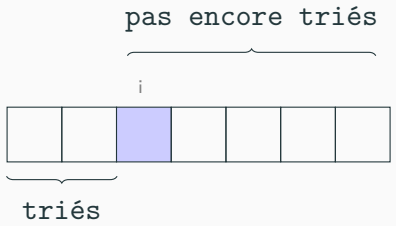
Le tri par sélection résout le problème de tri d'une séquence de nombres en ordre croissant. L'algorithme recherche le minimum parmi les éléments non triés pour le placer à la suite des éléments déjà triés.

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

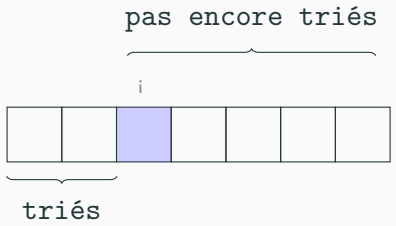
Tri par Sélection : Le principe

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

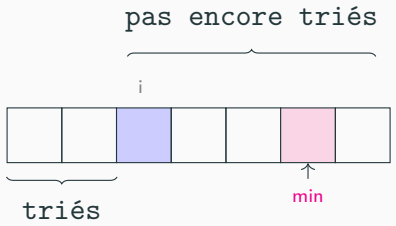
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.



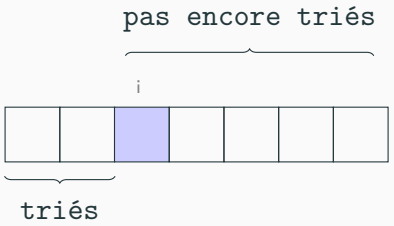
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.



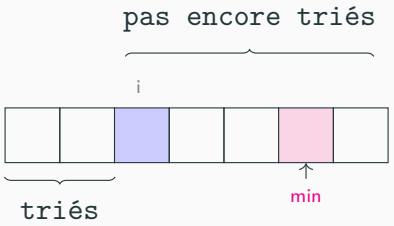
3. Trouver le plus petit élément du sous-tableau non encore trié.



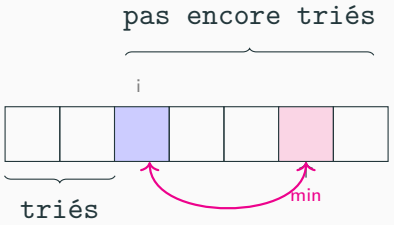
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.



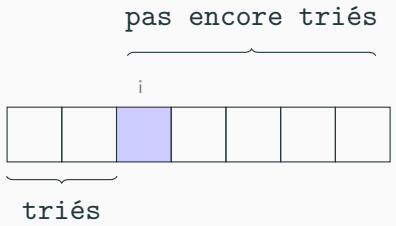
3. Trouver le plus petit élément du sous-tableau non encore trié.



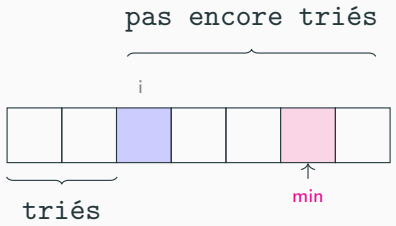
4. Echanger cet élément avec `T`$[i]$.



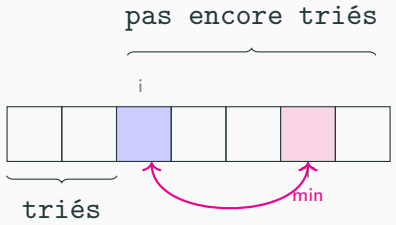
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

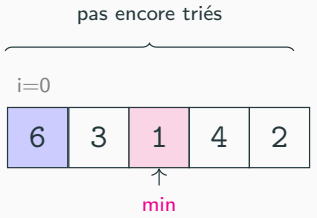


3. Trouver le plus petit élément du sous-tableau non encore trié.

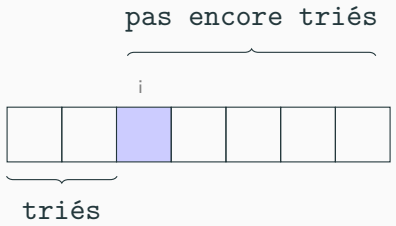


4. Echanger cet élément avec `T`$[i]$.

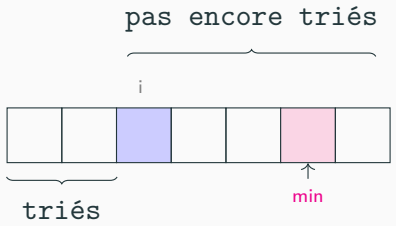




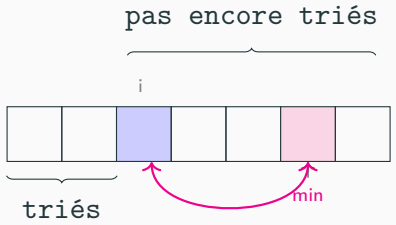
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

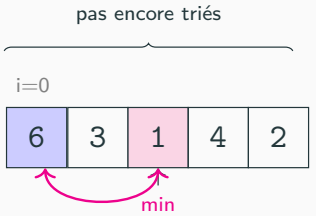


3. Trouver le plus petit élément du sous-tableau non encore trié.

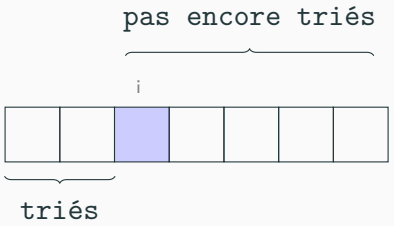


4. Echanger cet élément avec `T`$[i]$.

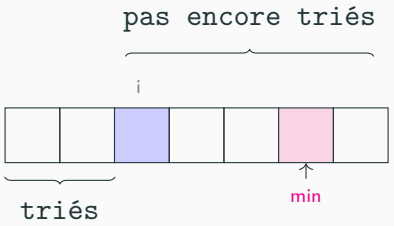




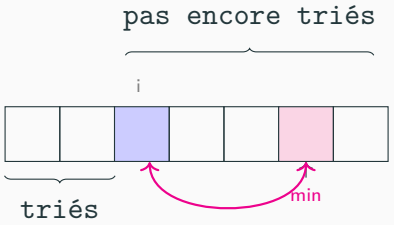
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

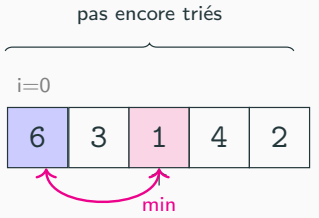


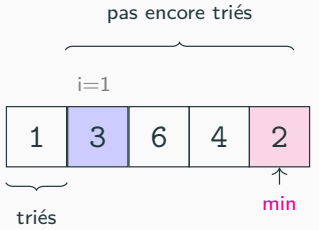
3. Trouver le plus petit élément du sous-tableau non encore trié.



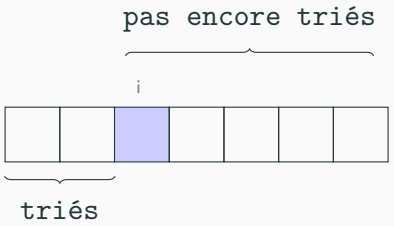
4. Echanger cet élément avec `T`$[i]$.

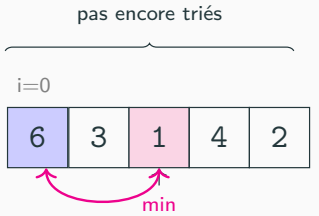


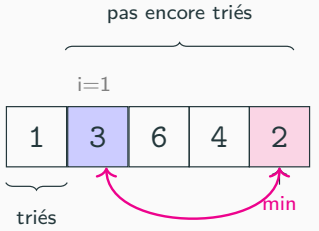




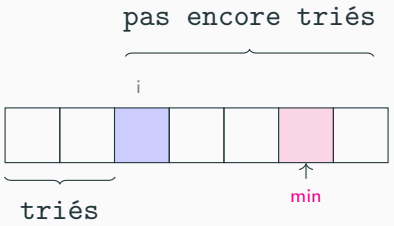
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.



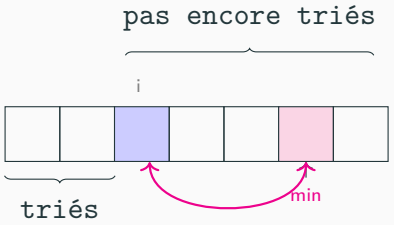




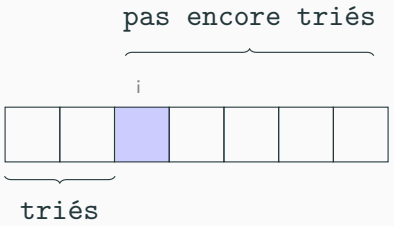
3. Trouver le plus petit élément du sous-tableau non encore trié.



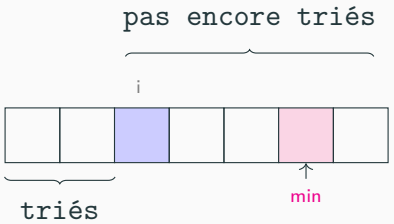
4. Echanger cet élément avec `T`$[i]$.



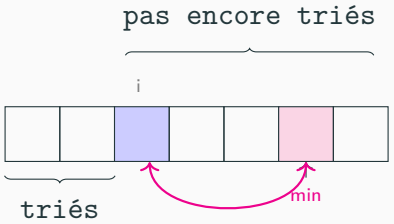
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

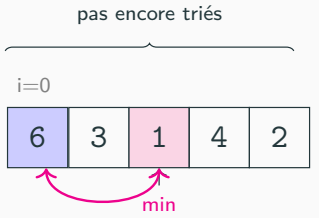


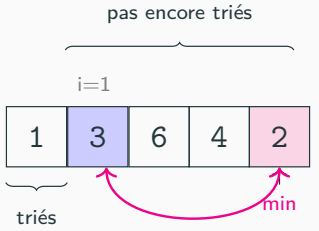
3. Trouver le plus petit élément du sous-tableau non encore trié.

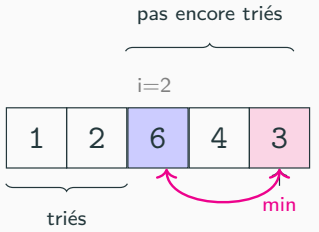


4. Echanger cet élément avec `T`$[i]$.

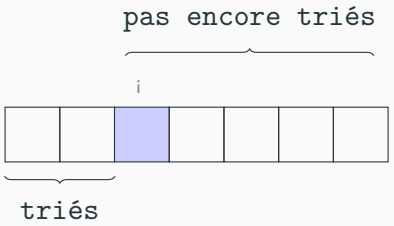




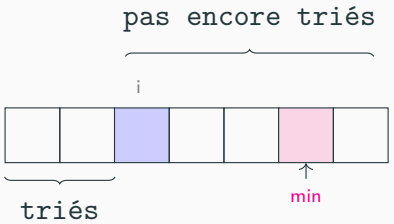




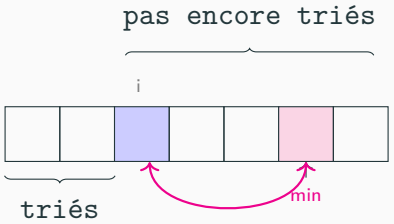
1. On parcourt le tableau `T` à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

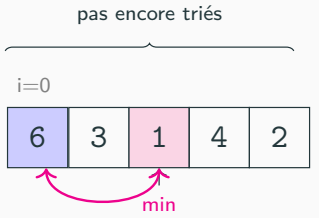


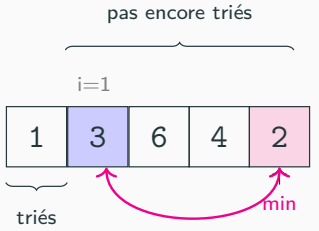
3. Trouver le plus petit élément du sous-tableau non encore trié.

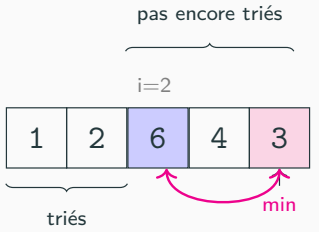


4. Echanger cet élément avec `T`$[i]$.

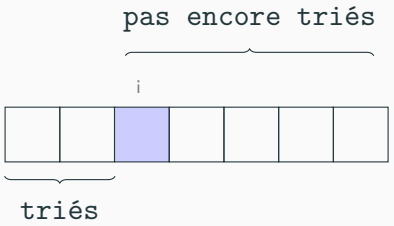




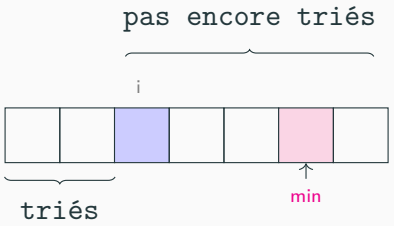




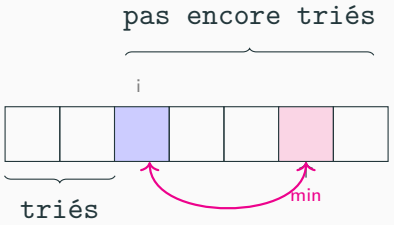
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

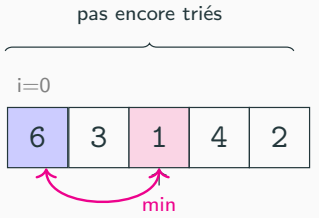


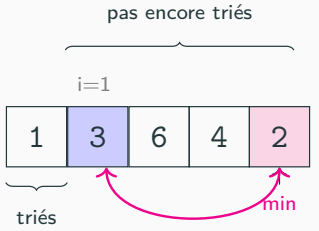
3. Trouver le plus petit élément du sous-tableau non encore trié.

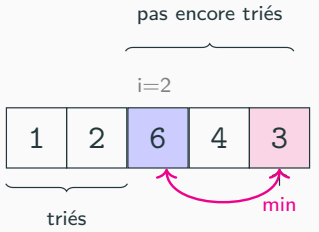


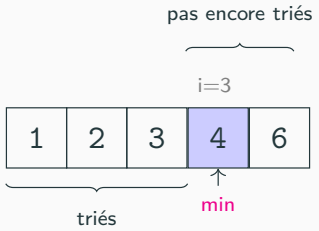
4. Echanger cet élément avec T[i].



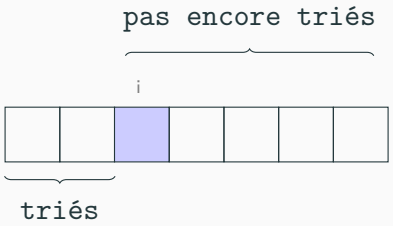




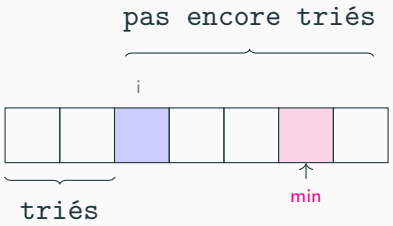




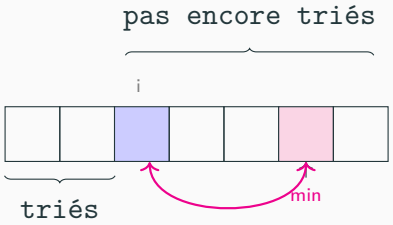
1. On parcourt le tableau T à trier de gauche à droite.
2. Au moment où on considère l'élément d'indice i, les éléments qui le précèdent sont déjà triés.

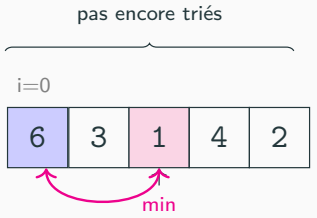


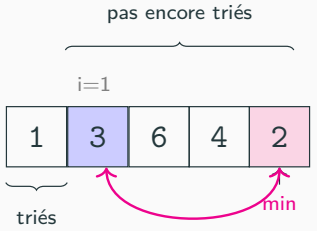
3. Trouver le plus petit élément du sous-tableau non encore trié.

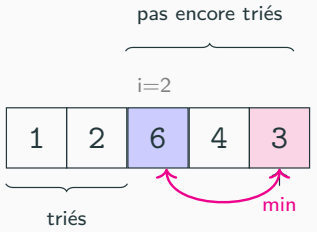


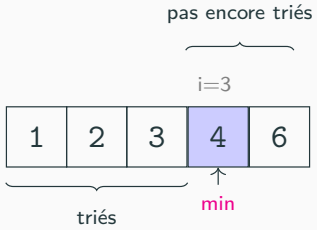
4. Echanger cet élément avec T[i].

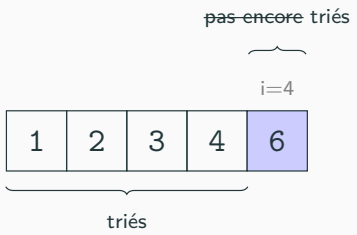












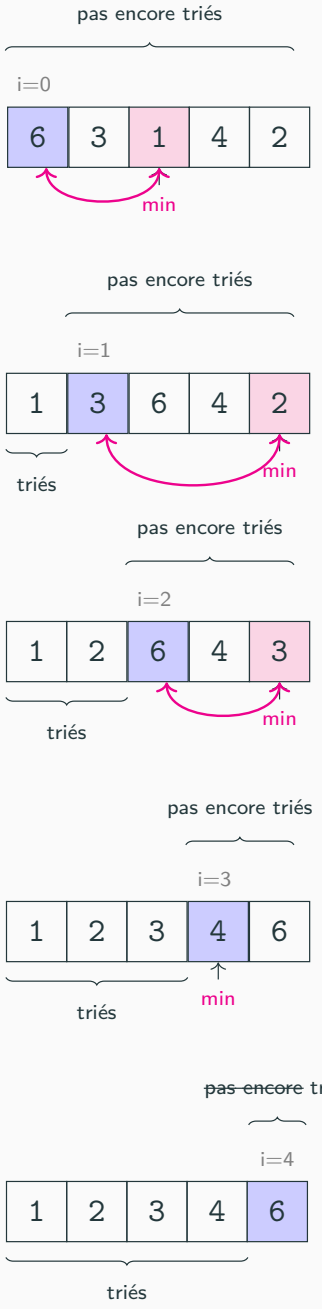
Tri par Sélection : Pseudocode

Entrée : tableau T

1: **fonction** TRIPARSÉLECTION(T)

2: $n \leftarrow$ longueur de T

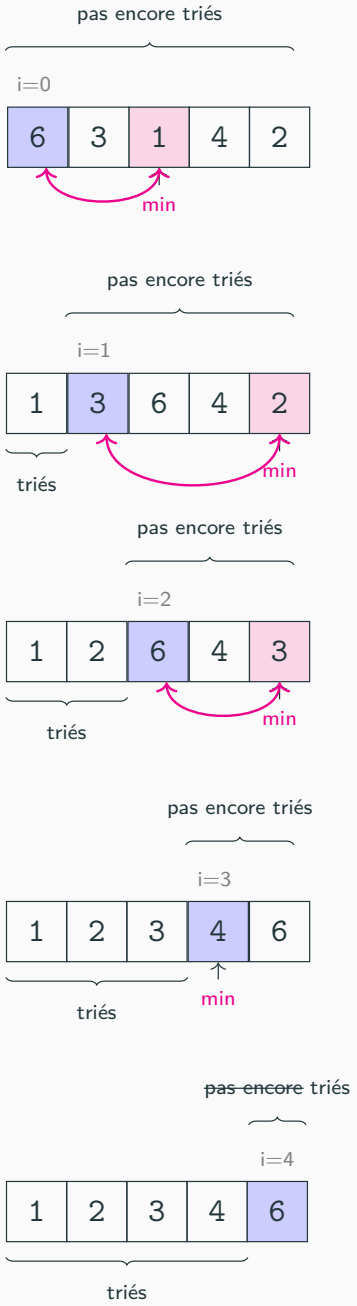
3: **pour** $i \leftarrow 0$ à $n - 2$ **faire**



Entrée : tableau T

1: **fonction** TRIPARSÉLECTION(T)
2: $n \leftarrow$ longueur de T
3: **pour** $i \leftarrow 0$ à $n - 2$ **faire**

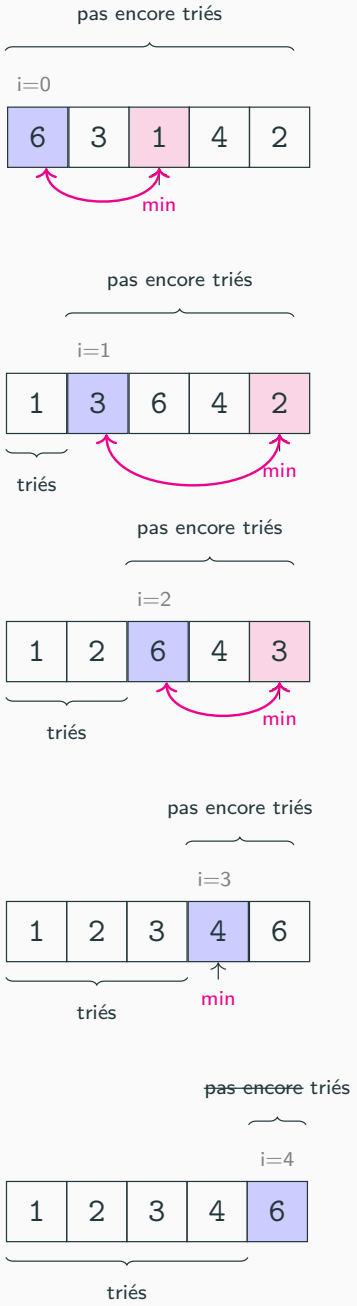
Remarquez qu'il suffit de parcourir les premiers $n - 1$ éléments du tableau. Le dernier élément est automatiquement le maximum à la fin de l'exécution.



Entrée : tableau `T`

1: **fonction** TRIPARSÉLECTION(`T`)
2: $n \leftarrow$ longueur de `T`
3: **pour** $i \leftarrow 0$ à $n - 2$ **faire**
4: trouver l'indice `min` du plus
5: petit élément parmi `T[i,. . . , n-1]`
6:
7:

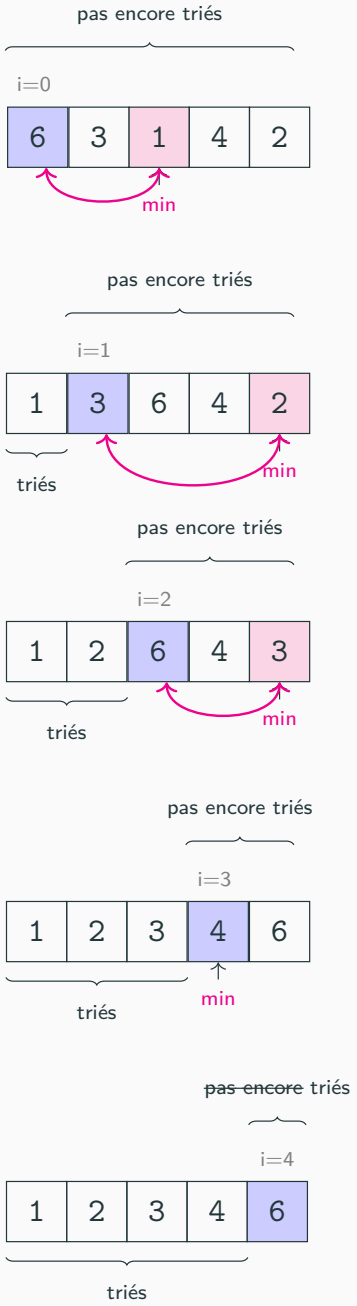
Remarquez qu'il suffit de parcourir les premiers $n - 1$ éléments du tableau. Le dernier élément est automatiquement le maximum à la fin de l'exécution.



Entrée : tableau `T`

```
1: fonction triParSélection(T)
2:     n ← longueur de T
3:     pour i ← 0 à n – 2 faire
4:         trouver l'indice min du plus
5:         petit élément parmi T[i,. . . , n-1]
6:
7:
8:         échanger T[i] et T[min]
```

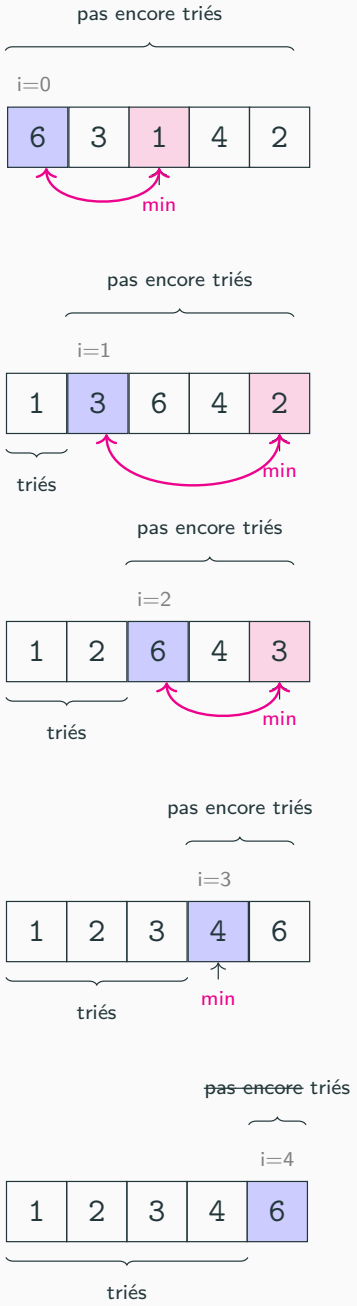
Remarquez qu'il suffit de parcourir les premiers $n - 1$ éléments du tableau. Le dernier élément est automatiquement le maximum à la fin de l'exécution.



Entrée : tableau T

```
1: fonction triParSélection(T)
2:     n ← longueur de T
3:     pour i ← 0 à n − 2 faire
4:         min ← i
5:         pour j ← i + 1 à n − 1 faire
6:             si T[j]<T[min] alors
7:                 min ← j
8:         échanger T[i] et T[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.



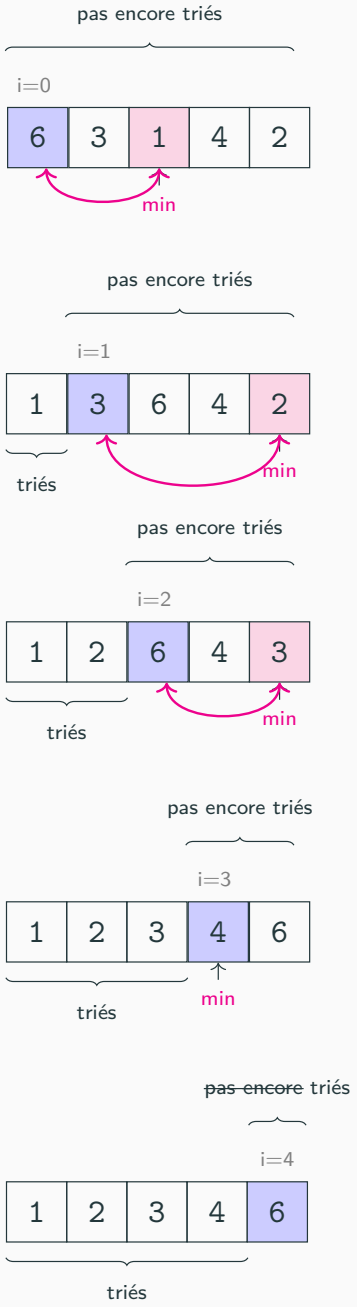
Tri par Sélection : Correction

Entrée : tableau T

```
1: fonction triParSélection(T)
2:     n ← longueur de T
3:     pour i ← 0 à n − 2 faire
4:         min ← i
5:         pour j ← i + 1 à n − 1 faire
6:             si T[j]<T[min] alors
7:                 min ← j
8:         échanger T[i] et T[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.

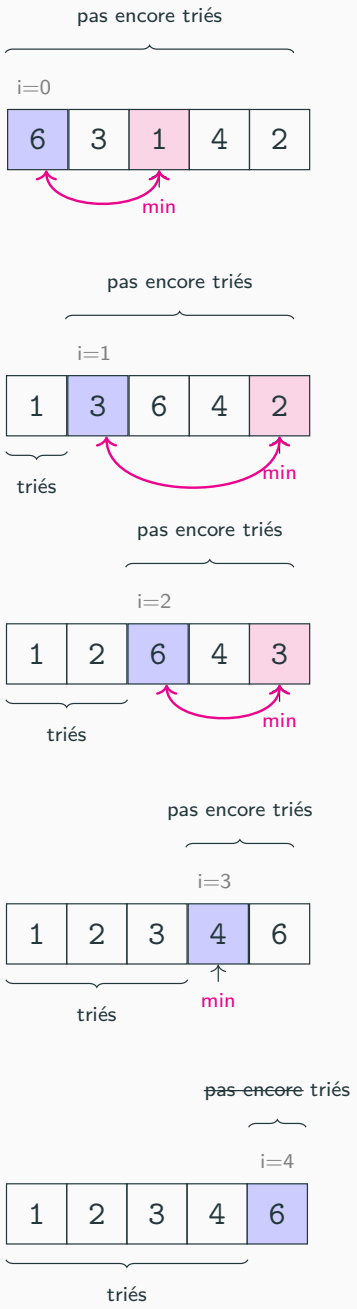


Entrée : tableau T

```
1: fonction triParSélection(T)
2:     n ← longueur de T
3:     pour i ← 0 à n − 2 faire
4:         min ← i
5:         pour j ← i + 1 à n − 1 faire
6:             si T[j]<T[min] alors
7:                 min ← j
8:         échanger T[i] et T[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) :

Nous notons par $\mathtt{T}_i$ le tableau obtenu à la fin de l'itération i et par $\mathtt{T}[k, \ldots, l]$ le sous-tableu obtenu à partir de $\mathtt{T}$ en ne considérant que les éléments de la position k à la position l.

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

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

La preuve est par récurrence sur i.

Cas de base $i = 0$:

1. est vrai, car $\mathtt{T}_0$ est obtenu en échangeant deux éléments de $\mathtt{T}$.

2. est vrai, car nous n'avons qu'un seul élément.

3. est vrai, car $\mathtt{T}_0[0]$ est le minimum du tableau original.

Preuve de correction (au tableau) :

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

1. est vrai car T_{k+1} est obtenu à partir du tableau initial en composant la permutation nécessaire pour obtenir T_k avec une transposition (échange) de deux éléments.

2. Observez que $\mathrm{T}_{k+1}[0, \ldots, k]$ coïncide avec $\mathrm{T}_k[0, \ldots, k]$, donc ces éléments sont 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 $\mathrm{T}_k[0, \ldots, k]$ sont inférieurs ou égaux au reste des éléments de ce tableau, notamment à $\mathrm{T}_{k+1}[k + 1]$. Donc, $\mathrm{T}_{k+1}[0, \ldots, k + 1]$ est trié.

3. Pour montrer que les éléments de $\mathrm{T}_{k+1}[0, \ldots, k]$ sont inférieurs ou égaux aux éléments de $\mathrm{T}_{k+1}[k + 2, \ldots, n - 1]$, nous utilisons la partie 3 de l'hypothèse inductive. Nous avons également que $\mathrm{T}_{k+1}[k + 1]$ est inférieur ou égal aux éléments de $\mathrm{T}_{k+1}[k + 2, \ldots, n - 1]$ parce que nous l'avons choisi comme le minimum parmi $\mathrm{T}_k[k + 1, \ldots, n - 1]$.

Preuve de correction (au tableau) :

Pour terminer, analysons l'invariant de la boucle pour $i = n - 2$, c'est-à-dire à la fin de l'algorithme. Les trois propriétés stipulent que

1. le tableau $\mathtt{T}_{n-2}$ est une permutation du tableau initial
2. les $n - 1$ premiers éléments du tableau $\mathtt{T}_{n-2}$ sont triés
3. les $n - 1$ premiers éléments du tableau $\mathtt{T}_{n-2}$ sont inférieurs ou égaux aux éléments restants.

Preuve de correction (au tableau) :

Pour terminer, analysons l'invariant de la boucle pour $i = n - 2$, c'est-à-dire à la fin de l'algorithme. Les trois propriétés stipulent que

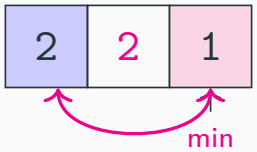
1. le tableau $\mathtt{T}_{n-2}$ est une permutation du tableau initial
2. les $n - 1$ premiers éléments du tableau $\mathtt{T}_{n-2}$ sont triés
3. les $n - 1$ premiers éléments du tableau $\mathtt{T}_{n-2}$ sont inférieurs ou égaux aux éléments restants.

Nous concluons que le tableau $\mathtt{T}_{n-2}$ est **trié** et contient exactement les **mêmes éléments** que ceux de $\mathtt{T}$.

Propriétés

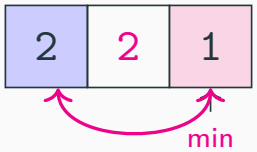
Définition. Un algorithme de tri est dit stable s'il préserve l'ordonnancement initial des éléments que l'ordre considère comme égaux.

Le tri par sélection est instable. Par exemple :

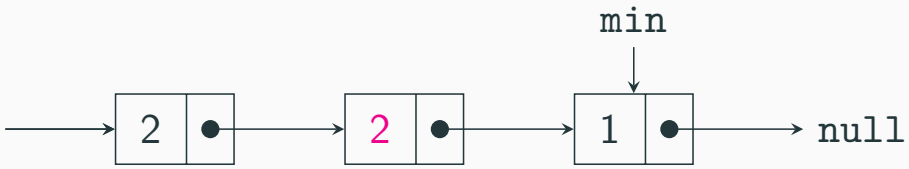


Définition. Un algorithme de tri est dit stable s'il préserve l'ordonnancement initial des éléments que l'ordre considère comme égaux.

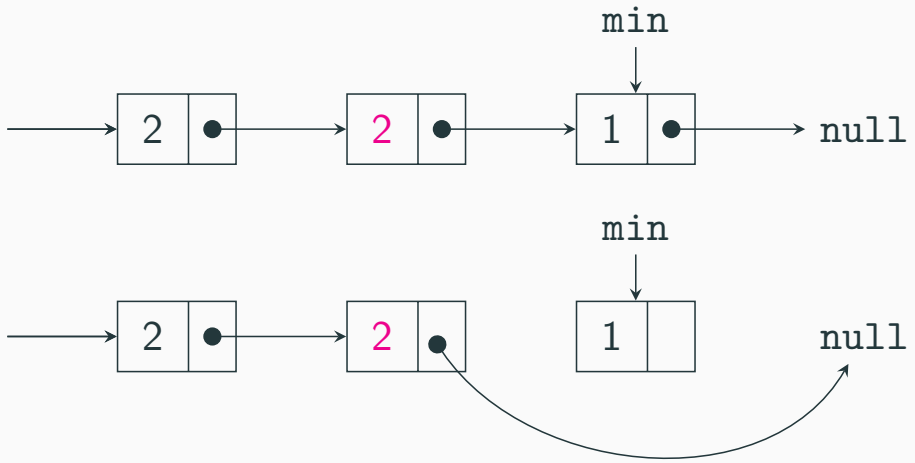
Le tri par sélection est instable. Par exemple :



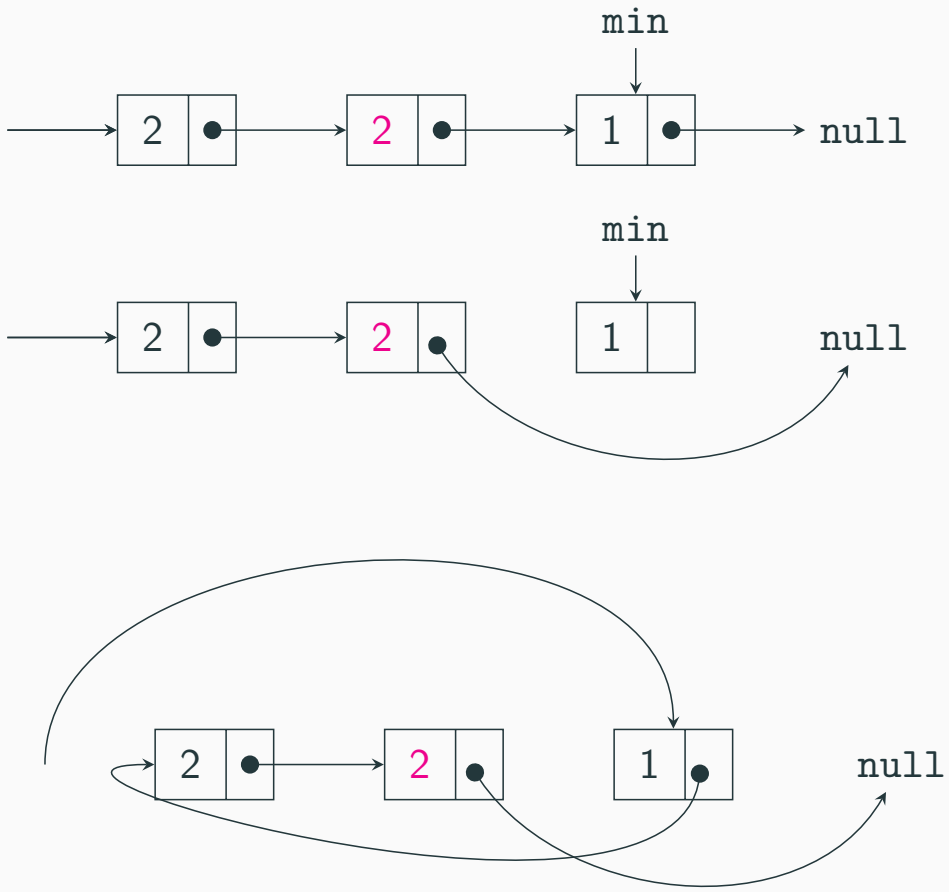
Une manière de rendre le tri par sélection stable, est de l'implémenter sur une liste. Au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.



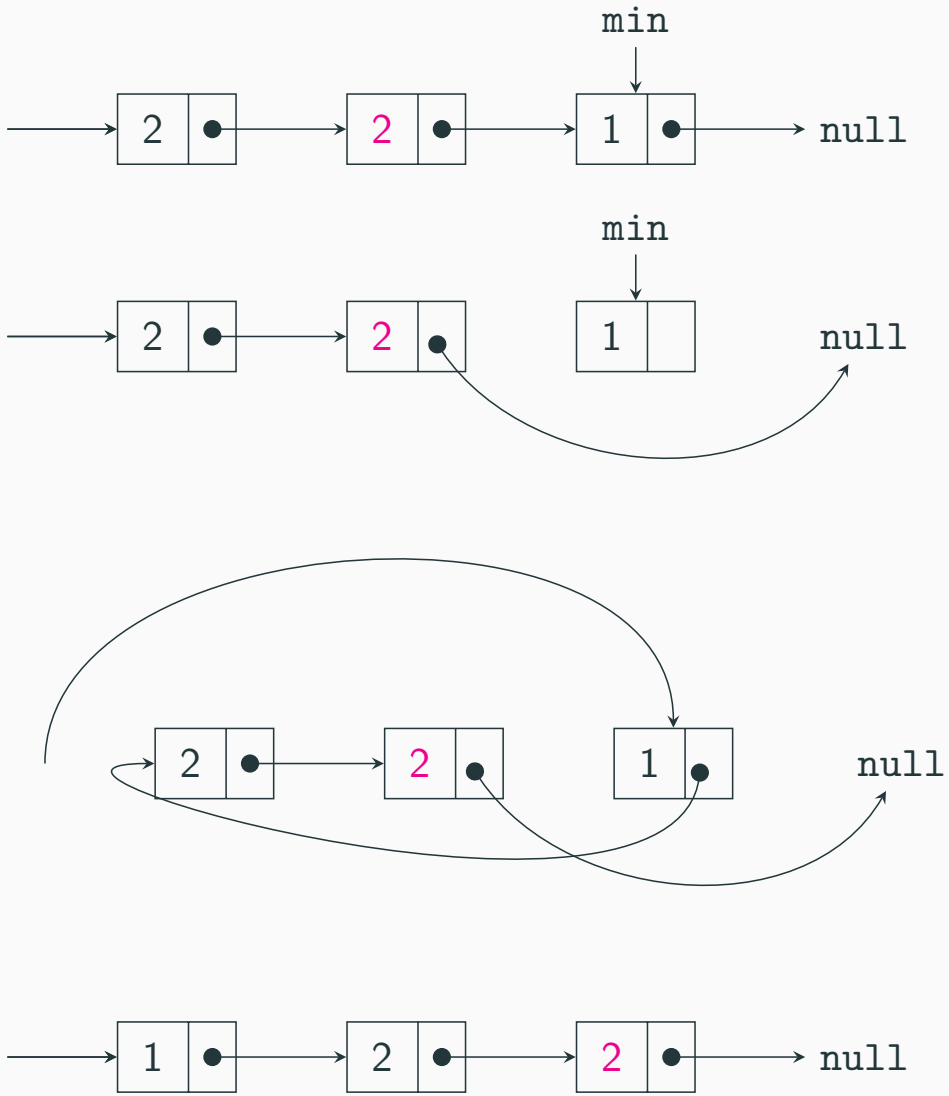
Une manière de rendre le tri par sélection stable, est de l'implémenter sur une liste. Au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.



Une manière de rendre le tri par 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 par 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 élements. (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 élements. (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 !