CHAPITRE 29

Les algorithmes de tri

La mort a une curieuse façon de faire le tri parmi les priorités.

Jack Sparrow [1]

Thème : Algorithmique

Contenus	**Capacités attendues**
Tris par insertion, par sélection	Écrire un algorithme de tri. Décrire un invariant de boucle qui prouve la correction des tris par insertion, par sélection. La terminaison de ces algorithmes est à justifier. On montre que leur coût est quadratique dans le pire cas.

I. Introduction

Vers 1951	**Betty Holberton** (l'une des six programmatrice de l'ENIAC) développe la première routine de tri et la première application logicielle.

Trier des données permet généralement de faciliter leur exploitation [2].

- ▷ Ordre alphabétique des mots d'un dictionnaire.
- ▷ Classement des cartes d'un joueur.
- ▷ ...

En informatique, on l'utilise par exemple :

- ▷ Lorsque l'on consulte sur Internet des listes de produits que l'on souhaite afficher par prix croissant ou décroissant, par popularité ...
- ▷ Pour exploiter des données (cf. chapitre sur les données en table).

1. Pirates des Caraïbes
2. `https://interstices.info/les-algorithmes-de-tri/`

▷ Pour accélérer les recherches d'une donnée (cf. chapitre sur la recherche dichotomique).

▷ ...

Dans ce cours, nous étudierons deux algorithmes de tri (sélection et insertion) et discuterons de leur efficacité.

On suppose que les éléments à trier sont des entiers (mais les algorithmes proposés sont valables pour n'importe quel type d'éléments) contenu dans un tableau. On supposera également qu'on trie ces tableaux par ordre croissant.

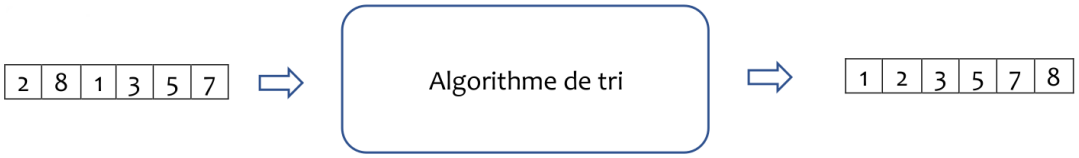

FIGURE 29.1 – Présentation synthétique des méthodes de tri

II. Tri par sélection (du minimum)

https://www.youtube.com/watch?v=DVj4nAXRsI4

II. 1. Principe du tri par sélection

▷ Première étape : à partir d'un tableau non trié de n éléments, on parcourt le tableau, et on permute le plus petit élément avec le premier ;

▷ deuxième étape : on parcourt le tableau à partir du deuxième élément (d'indice 1), et on permute le plus petit élément restant avec le deuxième ;

▷ ...

▷ i^e étape : on parcourt le tableau à partir du i^e élément, et on permute le plus petit élément restant avec le $(i\text{-}1)^e$;

▷ ...

▷ $(n - 1)^e$ et dernière étape : on compare les 2 derniers éléments du tableau, et on les permute si besoin.

Activité 1.
En indiquant ci-dessous les listes intermédiaires obtenues, triez le tableau $[5, -27, -14, 10, 7, 26]$ par sélection.

Étape 1	
Étape 2	
Étape 3	
Étape 4	
Étape 5	

II. 2. Algorithme du tri par sélection

Chaque étape de l'algorithme du tri par sélection se décompose en deux phases :

- ▷ Recherche de la valeur minimum à droite de la position courante du tableau (le sous-tableau droit) ;
- ▷ permutation de la valeur minimum trouvée avec la valeur de la position courante du tableau.

Algorithme : Fonction de tri par sélection du minimum de T

Entrées : T : un tableau d'entiers non triés
Sorties : T : le même tableau d'entiers triés

```
1  début
2      pour 0 ⩽ position courante < (nombre d'éléments dans T - 1) pas de 1 faire
3          position du minimum ← recherche du minimum du sous-tableau droit de T
4          si T[position du minimum] < T[position courante] alors
5              échanger les valeurs de la position courante et de la position du minimum
6          fin si
7      fin pour
8  fin
```

Algorithme : Fonction de recherche du minimum

Entrées : T : un tableau d'entiers non triés
i : l'indice à partir duquel on va chercher
Sorties : *position* : la position du minimum de T, à droite de i

```
1  début
2      position ← i
3      minimum ← T[i]
4      pour i + 1 ⩽ j < nombre d'éléments dans T par pas de 1 faire
5          si T[j] < minimum alors
6              position ← j
7              minimum ← T[j]
8          fin si
9      fin pour
10     renvoyer position
11 fin
```

Activité 2.
Complétez le tableau d'exécution de l'algorithme **tri par sélection** ci-dessous lorsqu'on souhaite trier le tableau [6, 5, 2] par sélection.

Numéro de ligne	T	position courante	position du minimum	Test
Initialisation	[6, 5, 2]			

La suite page suivante

Suite de l'activité 2.

Numéro de ligne	T	position courante	position du minimum	Test

II. 3. Complexité de l'algorithme du tri par sélection

Activité 3.

1. De façon à pouvoir déterminer sa complexité, indiquez à partir de l'algorithme du **tri par sélection** la ligne qui va être exécutée le plus souvent.
2. En reprenant l'exemple de l'activité 2, indiquez ci-dessous le nombre de comparaisons réalisées à chaque étape de l'algorithme **de recherche du minimum** dans cet exemple (le tableau initial vaut $[5, -27, 14, 10, 7, 26]$).

		Comparaisons réalisées
Étape 1	[-27, 5, -14, 10, 7, 26]	
Étape 2	[-27, -14, 5, 10, 7, 26]	
Étape 3	[-27, -14, 5, 10, 7, 26]	
Étape 4	[-27, -14, 5, 7, 10, 26]	
Étape 5	[-27, -14, 5, 7, 10, 26]	

La suite page suivante

Suite de l'activité 3.

3. Quel est le nombre total de comparaisons réalisées par l'algorithme **de tri par sélection**
4. Donnez la formule mathématique permettant de calculer le nombre de comparaisons réalisées si le tableau est de longueur n.
5. Un calcul mathématique nous donne $1 + 2 + 3 + \ldots + (n-1) = \dfrac{n^2}{2} - \dfrac{n}{2}$
   Calculez le nombre de comparaisons effectuées par l'algorithme pour un tableau de 10^2 valeurs ? de 10^3 valeurs ? de 10^6 valeurs ?
6. De quel type est la complexité de l'algorithme du tri par sélection ?

Coût en temps du tri par sélection

Le coût en temps du tri par sélection est **quadratique**.

II. 4. Validité de l'algorithme

II. 4. a. Terminaison de l'algorithme de tri par sélection

Pour s'assurer de la terminaison globale du tri par sélection, il faut s'assurer de la terminaison de chaque algorithme.

▷ Dans l'algorithme de **recherche du minimum**, il s'agit d'une boucle bornée. Le nombre d'éléments du tableau n'est pas modifié, **ET** le pas d'incrémentation est de 1.
C'est une preuve de terminaison. L'algorithme ne réalise pas de boucle infinie.

▷ Dans l'algorithme de **tri par sélection**, il s'agit d'une boucle bornée. Le nombre d'éléments du tableau n'est pas modifié, **ET** le pas d'incrémentation est de 1.
C'est une preuve de terminaison. L'algorithme ne réalise pas de boucle infinie.

II. 4. b. Correction de l'algorithme de tri par sélection

L'algorithme du tri par sélection conserve les éléments d'un tableau T et les trie dans l'ordre croissant.

Montrons que la proposition

> *Pour chaque valeur de $i \geqslant 0$, le sous-tableau gauche `T[0 ... i]` est trié **ET** toutes les valeurs du sous-tableau droit `T[i+1 ...]` sont plus grandes que `T[i]`*

est un invariant de boucle pour l'algorithme **tri par sélection**.

1. **Initialisation** : avant d'entrer dans la boucle, le sous-tableau gauche est vide. Il est donc nécessairement trié **ET** toutes les valeurs sont supérieures à un tableau vide. La proposition est vérifiée.
2. **Conservation** :
   ▷ On suppose qu'au début de l'itération i cette proposition est vérifiée. Le sous-tableau `T[0, ... , i-1]` est déjà trié. Dans ce cas, toutes les valeurs du sous-tableau droit sont supérieures à `T[i-1]`.
   On exécute alors la fonction *recherche du minimum*.
   ▷ **Premier cas** : *T[position du minimum]* est supérieur à *T[i]*. Dans ce cas, on ne modifie pas le tableau. La proposition est donc toujours vérifiée.

- ▷ **Deuxième cas** : *T[position du minimum]* est inférieur à *T[i]*. Dans ce cas, on modifie le tableau en remplacant *T[i]* par une valeur plus petite que celle existante dans le sous-tableau gauche **ET** plus grande que celle de `T[i-1]`. La proposition est donc toujours vérifiée.
- ▷ **Dans tous les cas**, la proposition est vérifiée.

L'invariant de boucle est vérifié. L'algorithme fait bien ce qu'on attend de lui.

III. Tri par insertion

`https://www.youtube.com/watch?v=wNhikYt2C18`

Cette méthode de tri est souvent utilisée pour trier des cartes à jouer :

- ▷ Je trie les 2 premières cartes.
- ▷ Je regarde la troisième et l'insère à sa bonne place (les valeurs les plus grandes sont décalées vers la droite).
- ▷ Je recommence avec la carte suivante.

III. 1. Principe du tri par insertion

- ▷ Première étape : à partir d'un tableau non trié de n éléments, on compare les 2 premiers éléments et on les range par ordre croissant ;
- ▷ Deuxième étape : on insère le 3 ^{e} élément de telle sorte que les trois premiers éléments du tableau soient triés.
- ▷ ...
- ▷ i ^{e} étape : on insère le (i + 1) ^{e} élément de telle sorte que les (i + 1) premiers éléments du tableau soient triés.
- ▷ ...
- ▷ (n - 1) ^{e} étape : on insère le n ^{e} élément de telle sorte que le tableau soit trié.

Activité 4.
En indiquant ci-dessous les listes intermédiaires obtenues, trier le tableau [5, 2, 6, 3, 8, 1] par insertion.

Étape 1	
Étape 2	
Étape 3	
Étape 4	
Étape 5	

III. 2. Algorithme du tri par insertion

Algorithme : Fonction tri par insertion

`/* Tri d'un tableau par insertion */`
Entrées : T : un tableau d'entiers non triés
Sorties : T : le même tableau d'entiers triés

```
1 début
2 |  pour 1 ⩽ position courante < nombre d'éléments dans T par pas de 1 faire
3 |  |  insérer T[position courante] à gauche de position courante
4 |  fin pour
5 fin
```

Algorithme : Fonction insérer

`/* insertion d'un élément à la bonne place dans un tableau */`
Entrées : T : un tableau d'entiers non triés
i : la position à laquelle insérer
v : la valeur à insérer
Sorties : T : le même tableau, avec v en position i

```
1 début
2 |  tant que (i > 0) ET (T[i-1] > v) faire
3 |  |  décaler T[i-1] d'un cran vers la droite
4 |  |  i ← i − 1
5 |  fin tq
6 |  T[i] ← v
7 fin
```

Activité 5.
Compléter le tableau d'exécution ci-dessous de l'algorithme **tri par insertion** lorsqu'on souhaite trier le tableau suivant $[6, 5, 2]$.

Numéro de ligne	T	position courante
Initialisation	[6, 5, 2]	

Activité 6.
Dans l'algorithme **insérer**,

1. Quel est le rôle de la ligne 3 ?

2. Quelle ligne permet d'insérer la valeur lorsque sa place est trouvée ?

III. 3. Complexité de l'algorithme du tri par insertion

Activité 7.

1. De façon à pouvoir déterminer sa complexité, indiquez à partir de l'algorithme du **tri par insertion** la ligne qui va être exécutée le plus souvent dans le pire des cas.
2. En reprenant l'exemple de l'activité 2, indiquez ci-dessous le nombre de comparaisons réalisées à chaque étape de l'algorithme **insérer** (le tableau initial vaut [8, 6, 5, 3, 2, 1])

Étape 1		
Étape 2		
Étape 3		
Étape 4		
Étape 5		

3. Quel est le nombre total de comparaisons réalisées par l'algorithme de **tri par insertion** ?
4. Donnez la formule mathématique permettant de calculer le nombre de tests réalisés dans le pire des cas si le tableau est de longueur n.
5. Quelle est la complexité de l'algorithme du tri par sélection ?
6. Quel est le nombre de comparaisons réalisées par l'algorithme de **insérer** pour le tableau [5, 2, 6, 3, 8, 1].

Étape 1		
Étape 2		
Étape 3		
Étape 4		
Étape 5		

Coût en temps du tri par sélection

Le coût en temps du tri par insertion est **quadratique**.

III. 4. Terminaison de l'algorithme de tri par insertion

Pour s'assurer de la terminaison globale du tri par insertion, il saut s'assurer de la terminaison de chaque algorithme.

▷ Dans l'algorithme du **tri par insertion**, la boucle « pour » se termine car le nombre d'éléments du tableau n'est pas modifié **ET** le pas d'incrémentation est de 1.

C'est une preuve de terminaison.

▷ Dans l'algorithme **insérer**, la boucle « tant que » demande un examen plus attentif. Pour la boucle « tant que », il faut trouver un **variant de boucle**. Ici, le variant de boucle est i :

- c'est un entier positif au départ (défini à la ligne 2)
- il décroît dans la boucle (ligne 4)
- l'une de ses valeurs ($i = 0$) provoque la sortie de la boucle.

C'est une preuve de terminaison. L'algorithme ne réalise pas de boucle infinie.

III. 5. Correction de l'algorithme de tri par insertion

L'algorithme du tri par insertion conserve les éléments d'un tableau T et les trie dans l'ordre croissant. La correction de l'algorithme de **tri par insertion** revient à montrer la correction de l'algorithme **insérer**.

Montrons que la proposition :

Pour chaque valeur de $i \geqslant 0$, le sous-tableau `T[0, ..., i]` est déjà trié

est un invariant de boucle pour l'algorithme **insérer**.

1. **Initialisation** : avant d'entrer dans la boucle, le sous-tableau gauche contient un seul élément. Il est donc nécessairement trié. La proposition est vérifiée.
2. **Conservation** :
   - ▷ On suppose qu'au début de la i^e itération cette proposition est vérifiée. Le sous-tableau gauche `T[0, ..., i-1]` est déjà trié.
   - ▷ **Premier cas** : l'élément `T[i-1]` est inférieur à `T[i]`. Dans ce cas, on ne modifie pas le tableau. La proposition est donc toujours vérifiée.
   - ▷ **Deuxième cas** : l'élément `T[i-1]` est supérieur à `T[i]`. Dans ce cas, on modifie le tableau en :
     - décalant`T[i-1]` d'un cran vers la droite
     - **SI** `T[i-2]` est supérieur à `T[i]`, alors on décale `T[i-2]` **ET** `T[i-1]` d'un cran vers la droite.
     - **SI** `T[i-3]` est supérieur à `T[i]`, alors on décale `T[i-3]`, `T[i-2]` **ET** `T[i-1]` d'un cran vers la droite.
     - ...
     - On recommence jusqu'à ce qu'on trouve une valeur inférieure à `T[i]` **OU** que l'on soit en première position du tableau *($i = 0$)*.
     - On met alors une valeur plus petite que celle existante dans le sous-tableau gauche.

     La proposition est donc toujours vérifiée.
   - ▷ **Dans tous les cas**, la proposition est vérifiée.

   La proposition est bien un invariant de boucle. L'algorithme **insérer** produit le résultat attendu.

Je retiens

- ▷ Le tri par sélection consiste à rechercher le plus petit élément d'une liste, et à le permuter avec le premier, puis de recommencer en se décalant vers la droite ;
- ▷ Le tri par insertion consiste à parcourir les éléments d'une liste, puis à les insérer à la bonne place ;
- ▷ la complexité du tri par insertion et du tri par sélection est quadratique.

IV. Références bibliographiques

▷ Numérique et Sciences Informatiques, Serge Bays, ellipses, 2019, p. 288

▷ Informatique, Nicolas Audfray et al., Dunod, 2017, p.85

▷ `https://info.blaisepascal.fr/nsi-recherche-dichotomique`, 10/04/2020

▷ A. LECOMTE, lycée Maurois, Elbeuf

▷ I. KHOUJA : `https://view.genial.ly/5e8ed71d186d4e0dec349ef2/presentation-la-complexite-de`

V. Exercices

Exercice 1 (QCM).

1. Le tri par sélection consisteà placer un par un les éléments à leur place définitive.

- ☐ Vrai ;
- ☐ Faux.

2. Après quelques étapes du tri par insertion, mais avant que le tri soit terminé, nous sommes sûrs qu'au moins un élément est rangé à sa place définitive.

- ☐ Vrai ;
- ☐ Faux.

3. Dans l'algorithme du tri par insertion, nous avons deux boucles imbriquées, une boucle « Tant que » à l'intérieur d'une boucle « pour ». Comment la terminaison est-elle démontrée ?

- ☐ Avec une boucle « pour » la terminaison est toujours assurée
- ☐ On utilise un variant de la boucle « pour »
- ☐ On utilise un invariant de la boucle « Tant que »
- ☐ On utilise un variant de la boucle « Tant que »

Exercice 2.

Indiquez ci-dessous les tableaux intermédiaires obtenus, lorsque l'on souhaite trier le tableau [3, 4, 1, 7, 2, 0] par sélection et par insertion.

	Tri par sélection	Tri par insertion
Étape 1		
Étape 2		
Étape 3		
Étape 4		
Étape 5		

Exercice 3 (Complexité en temps).
Sur sa machine, Émilie a implémenté un algorithme prenant en entrée des tableaux de taille n et ayant un coût quadratique dans le pire des cas. On admet que le temps d'exécution est quadratique comme le coût et qu'il n'y a pas de problème de surcharge de la mémoire [a]

1. Sur un tableau d'entiers de taille 250 son algorithme s'est exécuté en 90 ms. Quel sera approximativement le temps d'exécution sur un tableau d'entiers de taille 50 000 ?
2. Pour quelle taille de tableau d'entiers l'exécution prendra-t-elle dans le pire des cas une journée environ ?
3. Émilie effectue un test avec un tableau d'entiers de taille 250 000. Le temps d'exécution est de 2,324 secondes. Donner une explication possible.

a. en réalité on ne passe pas aussi simplement de la complexité algorithmique au temps d'exécution sur machine

Exercice 4 (Implémentation du tri par sélection).
Ouvrez le fichier **tri_selection.py**, puis exécutez le pour vérifier l'absence d'erreurs.

1. Après avoir pris connaissance de la documentation, complétez la fonction *echanger()*.
   Vérifiez l'absence d'erreur en lancant dans la console `test_echanger()`.
2. Après avoir pris connaissance de la documentation, complétez la fonction *rechercher_minimum()*.
   Vérifiez l'absence d'erreur en lancant dans la console `test_rechercher_minimum()`.
3. Après avoir pris connaissance de la documentation, complétez la fonction *tri_selection()*.
   Vérifiez l'absence d'erreur en lancant dans la console `test_tri_selection()`.

Exercice 5 (Implémentation du tri par insertion).
Ouvrez le fichier **tri_insertion.py**, puis exécutez le pour vérifier l'absence d'erreurs.

1. Après avoir pris connaissance de la documentation, complétez la fonction *inserer()*.
   Vérifiez l'absence d'erreur en lancant dans la console `test_inserer()`.
2. Après avoir pris connaissance de la documentation, complétez la fonction *tri_insertion()*.
   Vérifiez l'absence d'erreur en lancant dans la console `test_tri_insertion()`.