L16 : Les tris de tableaux - COURS

En informatique, on traite d'énormes quantités de données. Une fois ces données stockées, on souhaite y accéder et ce travail est facilité lorsque les données sont triées, comme dans un dictionnaire où les mots sont rangés dans un ordre logique qui permet de ne pas devoir parcourir tout l'ouvrage pour retrouver une définition. Cette problématique permet d'introduire la notion d'**algorithme de tri.**

1. Les algorithmes de tri

Qu'est-ce qu'un algorithme de tri ?

C'est un algorithme qui, partant d'une liste, renvoie une version ordonnée (croissante ou décroissante) de la liste.

```
[5,1,4,8,3,13,7,2]  →  [1,2,3,4,5,7,8,13]
```

Pourquoi trier ?

Le tri permet essentiellement d'accélérer les recherches. On remarquera qu'on peut trier à peu près tout type de données :

- ❏ Fichiers par taille
- ❏ Livres par ordre alphabétique
- ❏ Dossier par profondeur dans une arborescence
- ❏ Joueurs par score
- ❏ Fichiers par date d'édition, …, etc.

Les algorithmes de tris sont fondamentaux.

Il en existe plusieurs :

- ❏ Le tri par sélection
- ❏ Le tri par insertion
- ❏ Le tri bulle
- ❏ Le tri fusion
- ❏ Le tri rapide, …, etc.

Ils ont tous tous la même finalité : **trier un tableau de nombres**.

Voici une simulation visuelle de quelques algorithmes de tris :

https://drive.google.com/file/d/1SYKUYmo6TIqwzn3g9d7stvLR9HjqVsiI/view?usp=sharing

Pourquoi a-t-on besoin de plusieurs tris ?

Les situations exigeant de trier des données sont très variées.

Par exemple, on peut être amené à trier de très nombreuses données (recensement de population). Dans ce cas, on va essayer de limiter la complexité calculatoire.

On peut parfois gérer des données qui occupent beaucoup d'espace (images). On est alors tenté de limiter la complexité spatiale.

Cette année, nous étudierons deux types de tris en particulier :

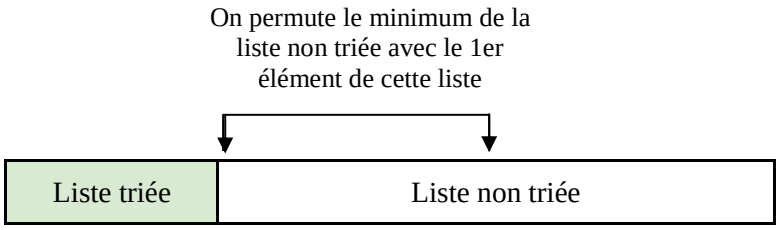
- **le tri sélection**
- **le tri par insertion.**

L'idée est de comparer leur efficacité et leur complexité.

2. Tri par sélection

a. Principe de l'algorithme

On parcourt les éléments d'une liste et on cherche le minimum, on le permute avec le premier élément de la liste. Puis on parcourt le reste des éléments de la liste et on cherche le nouveau minimum, on le permute avec le nouveau premier élément de la liste non triée, …, etc.



Le tri par sélection se décompose donc en deux étapes :

- ❏ Sélectionner un élément (d'où son nom) ;
- ❏ Le placer à sa bonne place.

b. Pseudo-code

En langage naturel, pour le tri par sélection, on obtient donc le **pseudo-code** suivant :

```
tri_Selection :
   Pour chaque élément
      Pour chaque élément de la partie non triée
         Mettre à jour le minimum du tableau rencontré jusqu'ici
      Échanger l'élément actuel avec le minimum
```

c. Complexité

Le tri par sélection a une complexité en $O(n^2)$. En effet :

- La première boucle parcourt n tours.
- La deuxième boucle parcourt n-i tours (i variant de 0 à n).

Sa complexité est donc légèrement inférieure à n^2, cependant cette différence est mineure et sa complexité est considérée comme étant en $O(n^2)$.

C'est un tri assez lent.

d. Démo :

http://lwh.free.fr/pages/algo/tri/tri_selection.html

3. Le tri par insertion

Le tri par insertion est le tri que la majorité des joueurs de cartes pratiquent intuitivement.

a. Principe de l'algorithme

Le principe du tri par insertion est de trier les éléments du tableau comme avec des cartes :

- ❏ On prend nos cartes mélangées dans notre main.
- ❏ On crée deux ensembles de cartes, l'un correspond à l'ensemble des cartes triées, l'autre contient l'ensemble des cartes restantes (non triées).
- ❏ On prend au fur et à mesure, une carte dans l'ensemble non trié et on l'insère à sa bonne place dans l'ensemble de carte triée.
- ❏ On répète cette opération tant qu'il y a des cartes dans l'ensemble non trié.

b. Pseudo-code

En langage naturel, pour le tri par insertion, on obtient donc le pseudo-code suivant :

```
tri_Insertion :
   Pour chaque élément non trié du tableau
      Décaler vers la droite dans la partie triée, les éléments supérieurs à
      celui que l'on souhaite insérer
      Placer notre élément à sa place dans le trou ainsi créé
```

c. Complexité

L'algorithme du tri par insertion a une complexité de $\boldsymbol{O(n^2)}$.

En effet :

- La première boucle parcourt $n-1$ tours, ici on notera plutôt n tours car le -1 n'est pas très important.
- Décaler les éléments de la partie triée prend i tours (avec i variant de 0 à n).

Dans le pire des cas on parcourt n^2 tours, donc le tri par insertion a une complexité en temps de $\boldsymbol{O(n^2)}$.

Ainsi, le tri par insertion est aussi un tri assez lent mais qui est relativement efficace lorsqu'il s'agit de trier de courtes listes.

d. Démo :

http://lwh.free.fr/pages/algo/tri/tri_insertion.html

4. Exercices

Exercice 1

a. Sur une feuille, appliquer l'algorithme de tri par sélection et dérouler les étapes à la main pour trier la liste d'entier suivant par ordre croissant : *L = [3, 1, 7, 9, 4, 12, 5]*

Étape 0	3	1	7	9	4	12	5
Étape 1	...						
Étape 2	...						

b. Sur une feuille, appliquer l'algorithme de tri par insertion et dérouler les étapes à la main pour trier la liste d'entier suivant par ordre croissant : `L = [5, 14, 2, 7, 6, 3, 8]`

Étape 0	5	14	2	7	6	3	8
Étape 1	...						
Étape 2	...						

Exercice 2

a. Écrire une fonction `tri_par_insertion(L)` qui retourne une liste *L* triée par ordre croissant en utilisant l'algorithme de tri par insertion où L est une liste passée en paramètre. Tester la fonction avec le tableau de l'exercice 1.

b. La complexité de cet algorithme est-elle linéaire ou quadratique ?

Exercice 3

a. Écrire une fonction `tri_par_selection(L)` qui retourne une liste L triée en utilisant l'algorithme de tri par sélection où L est une liste passée en paramètre. Tester la fonction avec le tableau de l'exercice 1.

b. La complexité de cet algorithme est-elle linéaire ou quadratique ?

Exercice 4

Écrire une fonction `min_liste(L)` qui retourne la valeur minimale de la liste L passée en paramètre.

Exercice 5

Écrire une fonction `moyenne_liste(L)` qui retourne la moyenne de la liste `L` passée en paramètre.

Exercice 6

a. Écrire une fonction `rech_seq(t,n,x)` qui effectue une recherche séquentielle de *x* (entier) parmi les *n* éléments d'un tableau *t* non trié et qui renvoie l'indice de sa première occurrence.

Exécuter l'algorithme avec les données suivantes :

```
n = 10
t = [11,7,31,9,15,8,22,30,18,13]
x = 18
```

b. Comment faut-il modifier l'algorithme si l'on n'est pas sûr que la valeur *x* appartienne au tableau ?