“

Depuis les débuts de l’informatique, les algorithmes de tri ont fait l’objet de nombreuses recherches.

Betty Holberton, qui a travaillé sur les premiers ordinateurs, compte parmi les auteurs des premiers algorithmes de tri (vers 1951). De nouveaux algorithmes sont toujours en cours d’invention, comme le Timsort assez récent puisqu’il date de 2002.

”

LES METHODES DE TRI :

- *Tri par insertion*
- *Tri par sélection*
- *Complexité*

Numérique et Sciences Informatiques
1ère

Support de cours :
Jean-Christophe BONNEFOY

Objectifs :

- Ecrire un algorithme de tri.
- Décrire un invariant de boucle qui prouve les algorithmes de tri par insertion et sélection.
- Connaître que le coût temporel est quadratique pour ces 2 méthodes de tri dans le pire des cas.

1. Le problème du tri

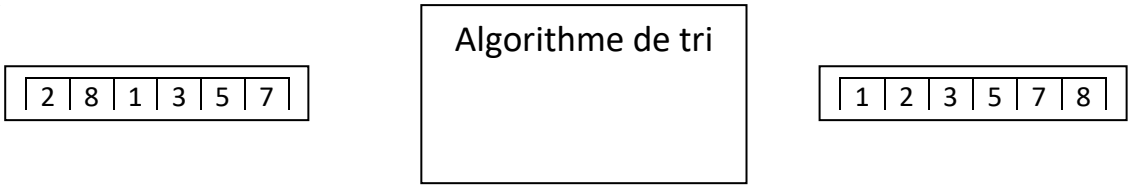
Comment ranger des données afin de faciliter leur accès futur ?

C'est par exemple l'ordre alphabétique du dictionnaire, ou les mots sont rangés dans un ordre logique qui permet de ne pas devoir parcourir tout l'ouvrage pour retrouver une définition.

Ce peut être aussi l'ordre intuitif dans lequel un joueur de cartes va ranger son jeu afin de limiter le temps de recherche pendant le déroulement de la partie. Cette problématique permet d'introduire la notion de tri (avec plusieurs sens distincts : séparer, ordonner, choisir), puis d'étudier différents algorithmes de tri.

Le tri permet essentiellement **d'accélérer les recherches**, grâce à l'algorithme de recherche **dichotomique**.

Exemple :



Le tri est historiquement un problème majeur en informatique pour plusieurs raisons :

• On a souvent besoin de trier des données (notes, noms, photos ...)

• Les algorithmes de tri sont des sous-programmes indispensables à de nombreuses applications (gestionnaires de fenêtres graphiques ...) ou programmes (compilateurs)

• La diversité des algorithmes de tri qui ont été développés, présente un intérêt pédagogique dans l'apprentissage de l'algorithmique

2. Le tri par insertion

2.1. Le principe

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 carte, l'un correspond à l'ensemble de carte triée, 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.

Prenons comme exemple la suite de nombre suivante : 9, 2, 7, 1 que l'on veut trier en ordre croissant avec l'algorithme du tri par insertion :

1er tour :

9 | **2**, 7, 1 : à gauche la partie triée du tableau (le premier élément est considéré comme trié puisqu'il est seul dans cette partie), à droite la partie non triée. On prend le premier élément de la partie non triée, 2, et on l'insère à sa place dans la partie triée, c'est-à-dire à gauche de 9.

2ème tour :

2, 9 | **7**, 1 : on prend 7, et on le place entre 2 et 9 dans la partie triée.

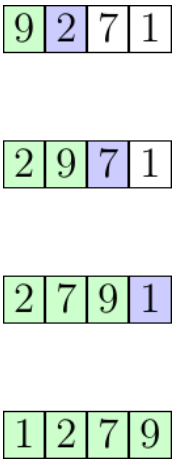
3ème tour :

2, 7, 9 | **1** : on continue avec 1 que l'on place au début de la première partie.

Résultat : 1, 2, 7, 9

Pour insérer un élément dans la partie triée, on parcourt de droite à gauche tant que l'élément est plus grand que celui que l'on souhaite insérer.

Pour résumer l'idée de l'algorithme :



Exemple de tri par insertion

La partie verte du tableau est la partie triée, l'élément en bleu est le prochain élément non trié à placer et la partie blanche est la partie non triée.

2.2. L'algorithme

L'algorithme de tri par insertion sur une liste L de n éléments peut être le suivant :

```
POUR i allant de 1 à n-1 :
        x ← L[i]
        j ← i
        TANT QUE j > 0 et L[j-1] > x :
                L[j] ← L[j-1]
                j ← j-1
        L[j] ← x
```

2.3. Complexité

La notion de complexité d'un algorithme va rendre compte de l'efficacité de cet algorithme. Pour un même problème, par exemple trier un tableau, il existe plusieurs algorithmes, certains algorithmes sont plus efficaces que d'autres (par exemple un algorithme A mettra moins de temps qu'un algorithme B pour résoudre exactement le même problème, sur la même machine).

Il existe 2 types de complexité : une complexité en temps et une complexité en mémoire. Nous nous intéresserons ici uniquement à la **complexité en temps**. La complexité en temps est directement liée au nombre d'opérations élémentaires qui doivent être exécutées afin de résoudre un problème donné. L'évaluation de ce nombre d'opérations élémentaires n'est pas chose facile, on rencontre souvent des cas litigieux.

- *À quoi correspond le meilleur des cas pour un algorithme de tri ?* Tout simplement quand le tableau initial est déjà trié comme dans cet exemple : t = [1, 2, 7, 9].
- *À quoi correspond le pire des cas pour un algorithme de tri ?* Tout simplement quand le tableau initial est "trié à l'envers" (les entiers sont classés du plus grand au plus petit), comme dans cet exemple : t = [9, 7, 2, 1].

Etudions le meilleur des cas : La boucle POUR effectue $(n-1)$ itérations, et pour chaque itération, il y a 3 affectations, soit un total de $3(n-1)$ opérations. Cependant, on considère que $n \to \infty$ et donc que $3n \gg -3$. La complexité en temps, dans ce cas est de la forme $\boldsymbol{kn}$, On dit que **le coût en temps est linéaire** et est noté **O(n)**.

Etudions le pire des cas : La boucle POUR effectue (n-1) itérations, et pour chaque itération, il y a d'abord 2 affectations, puis n itérations dans la boucle TANT QUE, dans cette dernière il y a 2 affectations, soit donc un total de $2n$ et enfin 1 dernière affectation soit un total de $2n+3$ opérations dans la boucle FOR. Soit un nombre d'opérations total de $(n-1)(2n+3) = 2n^2 + n - 3$ opérations. Quand $n \to \infty$ on a donc que $2n^2 \gg n-3$. La complexité en temps, dans ce cas est de la forme $\boldsymbol{kn^2}$, On dit **que le coût en temps est quadratique** et est noté **O(n²)**.

2.4. Notion d'invariant de boucle

La propriété *P(i)* est appelée invariant de boucle : c'est une propriété qui est vraie avant et après chaque itération. S'il existe un invariant de boucle, alors l'algorithme est correct.

Les invariants de boucle servent, entre autres, à prouver la validité d'un algorithme comportant une ou plusieurs boucles FOR ou WHILE.

Un invariant de boucle est une proposition qui :

- Est Vraie avant d'entrer dans la boucle (**initialisation**)
- Reste Vraie après une itération (**conservation**)
- Donne le résultat attendu en fin de boucle (**terminaison**)

Dans l'algorithme du tri par insertion, on choisit comme invariant de boucle la proposition suivante :

H : « La liste L[0 :i+1] est triée par ordre croissant à l'issue de l'itération i »

Il nous faut maintenant vérifier les 3 étapes : initialisation, conservation et terminaison.

Initialisation :
Avant d'entrer dans la boucle, i=0, la liste L n'est constituée que du premier élément L[0], elle est donc forcément triée. H est donc Vraie

Conservation :
On se place à l'itération i. La liste est donc L[0 :i+1] = [L[0], L[1],…, L[i]] et est supposée triée si H est Vraie. On effectue un nouveau tour de boucle, on se place donc à l'itération i+1.

```
X ← L[i+1]
j ← i+1
TANT QUE j > 0 et L[j-1] > x :
        L[j] ← L[j-1]
        j ← j-1
L[j] ← x
```

Alors la liste L[0 :i+2] = [L[0], L[1],…, L[i], L[i+1]] est bien triée. H reste Vraie.

Terminaison :
En sortie de boucle, i a pris sa dernière valeur (n-1). La boucle permet de d'effectuer le tri au rang (n-1).

```
X ← L[n-1]
j ← n-1
TANT QUE j > 0 et L[j-1] > x :
        L[j] ← L[j-1]
        j ← j-1
L[j] ← x
```

La liste L est donc bien triée par ordre croissant. La fonction a rempli son objectif. L'algorithme est valide.

3. Le tri par sélection (du minimum)

3.1. Le principe

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, etc…

Prenons maintenant comme exemple la suite de nombres suivante : 6, 1, 9, 3. Trions cette suite avec l'algorithme du tri par sélection dans l'ordre croissant :

1er tour :

6, **1**, 9, 3 : le minimum du tableau est 1, on le place donc sur la première case (en l'échangeant avec le 6).

2ème tour :

1, 6, 9, **3** : le deuxième plus petit élément est 3, on le place sur la deuxième case et on l'échange avec le 6.

3ème tour :

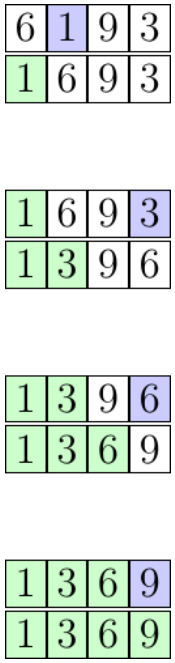
1, 3, 9, **6** : le troisième plus petit élément est 6, on l'échange avec 9 pour le placer sur la troisième case.

4ème tour :

1, 3, 6, **9** : le quatrième plus petit élément du tableau est 9, il est déjà en quatrième position on ne fait rien.

Résultat : 1, 3, 6, 9

Ce tri se décompose réellement en deux étapes distinctes (recherche du minimum et déplacement):



Exemple de tri par sélection

À chaque tour, on cherche le minimum dans l'espace non trié du tableau (le minimum est représenté en bleu, et la partie non triée en blanc), ensuite on déplace cet élément à sa place définitive (représentée en vert). En faisant cela pour chaque élément du tableau, ce dernier se retrouve trié au bout de n tours maximum (n étant la taille du tableau).

3.2. L'algorithme

L'algorithme de tri par sélection sur une liste L de n éléments peut être le suivant :

```
POUR i allant de 0 à n-1 :
        indice_du_min ← i
        POUR j allant de i+1 à n-1 :
                Si L[j] < L[indice_du_min]
                        indice_du_min← j
        Si indice_du_min != i :
                x ← L[indice_du_min]
                L[indice_du_min] ←L[i]
                L[i] ← x
```

3.3. Complexité

Etudions le pire des cas : Pour chaque itération, i, on a

- Une première affectation
- (n-i-1) affectations car il y a n-i-1 valeurs de j
- 3 affectations.

Au final on a donc (n-i+3) affectation par itération i , donc en utilisant les symboles mathématiques, on a au total :

$$\sum_{i=0}^{n-2}(n-i+3) = \sum_{i=0}^{n-2}(n+3) - \sum_{i=0}^{n-2} i = (n-2)(n+3) - \frac{(n-2)(n-1)}{2} = (n-2)\left[(n+3) - \frac{n-1}{2}\right]$$

$$= (n-2)\left[\frac{2n+6-n+1}{2}\right] = (n-2) * \frac{n+7}{2} = \frac{n^2+7n-2n-14}{2} = \frac{1}{2}n^2 + \frac{5}{2}n - 7$$

Le coût en temps est également quadratique.

3.4. Validité de l'algorithme

En choisissant l'invariant de boucle H : « La liste L[0 :i+1] est triée par ordre croissant à l'issue de l'itération i », on peut montrer de la même manière que le tri par insertion que cet algorithme est valide.