Chapter 1

Algorithmes de tri

Al-Khwarizmi (vers 780 – vers 850) Mathématicien persan du IX^{e} siècle. Pour aider ses contemporains, ce sage a écrit un livre regroupant des méthodes claires, à suivre pas à pas, pour résoudre des problèmes mathématiques.

De plus, le titre de ce livre contient l'expression *Al Jabr* (lisez-le à haute voix) ce qui explique que son nom soit associé à l'arrivée de l'algèbre en Europe. Son nom fut traduit par *Algoritmi* en latin .

Source:`https://interstices.info`

Dans ce chapitre nous étudions des algorithmes de tri: comment trier dans un ordre une liste d'éléments comparables ? Nous étudierons leur efficacité ou complexité.

1.1 Spécification pour la construction d'une liste triée

Voici la spécification utilisée dans le chapitre pour le tri de liste.
Entrée : ℓ une liste de longueur n, homogène, d'éléments ordonnables.

Sortie : Aucune

Effet de bord : la liste ℓ est triée.

Les deux algorithmes de tri renvoientt la liste donnée en paramètre triée, on dit que la liste est *triée en place*: la fonction ne retourne rien et travaille par effet de bord. Le principe des tris par *insertion* et par *sélection* est de parcourir le tableau de la gauche vers la droite, en maintenant une partie déjà triée et à sa place définitive:

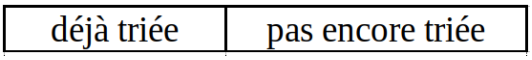

Figure 1.1: Forme de la liste pendant le tri.

1.2 Tri par insertion

Dans une liste déjà triée on déplace l'élément d'indice `i` en le comparant aux précédents que l'on fait remonter jusqu'à ce que $\ell[i]$ soit à la bonne place. C'est l'algorithme le plus naturel pour trier un jeu de carte : on ordonne les deux premières cartes puis on choisit une troisième qu'on *insère* au bon endroit parmi les deux premières. Et on recommence avec une quatrième carte...

Spécification d'un algorithme permettant d'insérer un élément dans une tranche de liste.

Entrée: ℓ liste de longueur n, homogène, d'éléments ordonnables, $i < n$ un indice, tel que ℓ[0:i] est triée.

Sortie: aucune.

Effet de bord: la tranche ℓ[0:i+1] est triée.

Insertion dans une liste triée

1.	aux←ℓ[i]
2.	k←i
3.	**TQ** $k \geqslant 1$ et aux<ℓ[k−1]
4.	ℓ[k]←ℓ[k−1]
5.	k←k−1
6.	**Fin TQ**
7.	ℓ[k]←aux

Tri par insertion

1.**Pour** i variant de 1 à n−1
2. inserer ℓ[i] dans ℓ[0:i+1]
3.**Fin Pour**

1.2.1 Appliquer cet algorithme

Appliquer cet algorithme à la liste R-9-7-D-AS-10-8 en soulignant à chaque tour la tranche de liste triée.

- R-9-7-D-AS-10-8
- **9-R**-7-D-AS-10-8
- **7-9-R**-D-AS-10-8 etc.

1.2.2 Complexité de l'algorithme

Coût de l'insertion dans une liste ℓ[0:i+1]

On compte à nouveau le nombre de comparaisons d'éléments de la liste pour insérer l'élément d'indice i dans la tranche ℓ[0:i+1]. Ce nombre dépend de i et du contenu de la liste. En effet:

- Dans le meilleur des cas, une seule comparaison suffit: lorsque l'élément d'indice i est déjà plus grand que tout ceux de la liste.
- Dans le pire des cas, il faut "redescendre "toute la liste:il y a donc i comparaisons.

Le coût est donc: $1 \leqslant \mathcal{C}_{ins}(i) \leqslant i$.

Coût de l'agorithme de tri par insertion

C'est le coût de la boucle contenant l'insertion:

$$\mathcal{C}_{tri-inser}(n) = \sum_{1}^{n-1} \mathcal{C}_{ins}(i).$$

Compte-tenu de la remarque faite pour le coût de l'insertion, nous distinguons deux cas:

- Dans le meilleur des cas, il y a une comparaison à faire à chaque tour et:

$$\mathcal{C}_{tri-inser}(n) = \sum_{1}^{n-1} 1 = n-1.$$

- Dans le pire des cas, il faut comparer à tous les éléments qui précèdent et:

$$\mathcal{C}_{tri-inser}(n) = \sum_{1}^{n-1} i = \frac{[1+(n-1)](n-1)}{2} = \frac{n(n-1)}{2} \sim \frac{n^2}{2}.$$

Dans le pire des cas, le coût du tri par insertion est lui aussi quadratique mais dans le meilleur des cas il est linéaire. Le tri par insertion est notamment efficace pour des listes presque triées.

1.2.3 Implémentation à faire en exercice

1.2.4 Correction de l'algorithme

Terminaison

L'algorithme contient deux boucles dont une bornée. Il s'agit de trouver un variant de boucle pour la boucle **TantQue**. k est ce variant de boucle: il décroit vers zéro. La boucle se termine.

Correction partielle

Déterminer l'invariant de boucle.

1.3 Tri par selection

Il fonctionne de façon basique, on *sélectionne* directement le plus petit élément dans la partie droite de la liste (non triée) et on l' échange avec l'élément le plus à gauche de la liste.

Et on répète cet échange:

- On sélectionne le plus petit élément et on l'échange avec le plus à gauche.
- Puis on selectionne le deuxième plus petit élément et on le place en deuxième.
- Et ainsi de suite...

Voici un algorithme permettant de rechercher l'indice d'un élément de valeur minimale dans une tranche de liste.

1.3.1 Spécification pour la sélection du minimum dans une tranche de liste

Entrée : ℓ une liste de longueur n, homogène, d'éléments ordonnables. a et b deux indices $0 \leqslant a < b \leqslant n$

Sortie : indice d'un élément minimal de la liste ℓ.

Algorithme de sélection

```
1.ind_min ← a
2.Pour i variant de a+1 à b−1
3.  Si ℓ[i] < ℓ[ind_min]
4.    ind_min ← i
5.  Fin Si
6.Fin Pour
7.Retourner ind_min
```

Algorithme de tri:

```
1.Pour j variant de 0 à n−1
2.  ind_min←select_min(ℓ,j,n)
3.  echanger ℓ[j] et ℓ[ind_min]
4.Fin Pour
```

1.3.2 Appliquer cet algorithme

Appliquer cet algorithme à la liste R-9-7-D-AS-10-8 en soulignant à chaque tour la tranche de liste triée.

- **7**-9-R-D-AS-10-8
- **7-8**-R-D-AS-10-9 etc.

1.3.3 Complexité

Coût de la selection du minimum dans une liste ℓ de longueur n

Ce coût dépend de la longueur n de la liste. Il y a une comparaison à faire par tour.

$$\mathcal{C}_s(k) = \sum_{i=0}^{n-1} 1 = b - 1 - (a+1) + 1 = b - a - 1 = k - 1$$

Coût de l'algorithme de tri par sélection

Cet algorithme n'a qu'une seule boucle, c'est le coût à chaque étape de la boucle de la sélection de l'indice minimum pour une tranche de longueur $n - i$ (la longueur de la liste où s'effectue la sélection diminue de 1 à chaque tour):

$\mathcal{C}_{tri}(n) = \sum_{i=0}^{n-2} \mathcal{C}_s(n-i) = \sum_{i=0}^{n-2} (n-i) - 1 = \frac{(n-1)[(n-1)+1]}{2} = \frac{n(n-1)}{2} \sim \frac{n^2}{2}$.

Le coût croît donc comme le carré de n: il est quadratique (ou: en n^2)

1.3.4 Appliquer l'algorithme

Appliquer cet algorithme à la liste R-9-7-D-AS-10-8 en soulignant à chaque tour la tranche de liste triée.

- R-9-7-D-AS-10-8
- **7-8**-R-D-AS-10-9
- **7-8-9**-D-AS-10-R

R-9-7-D-AS-10-8

1.3.5 Correction de l'algorithme

Terminaison

L'algorithme est constitué de deux boucles bornées, il se termine donc nécessairement.

Correction partielle

A faire.

1.3.6 Exercices

Exercice 1

1. Implémentez l'algorithme *insertion* en Python.
2. Ecrivez alors l'algorithme de tri par *insertion*. (Eventuellement en utilisant une fonction `insertion(a, b)` où a et b sont les deux bornes de la tranche de liste).

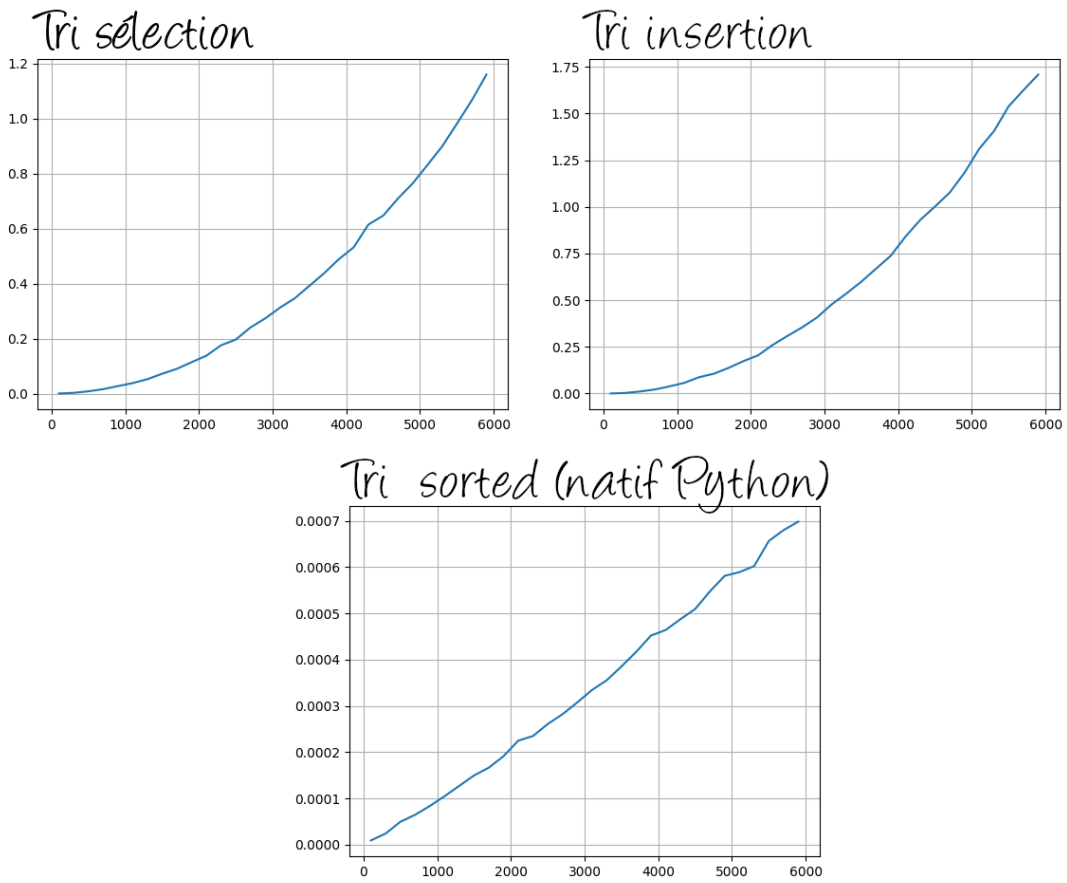

Figure 1.2: Comparaison de trois algorithmes de tri.

Exercice 2

1. Implémentez l'algorithme de *sélection* en python.
2. Ecrivez alors l'algorithme de tri par *sélection*. (Eventuellement en utilisant une fonction `selection(a, b)` où a et b sont les deux bornes de la tranche de liste).

Exercice 3

1. Avec une couleur d'un jeu de cartes, faire fonctionner ces deux algorithmes.
2. A l'aide de la fonction *timeit()*. Mesurer le temps d'execution pou chacun de ces tris et le tri *sort* de Python pour des listes de 100 éléments genérées alatoirement. Tracer un graphique et commenter.