Algorithmie

Les algorithmes de tri

Définition d’un algorithme de Tri

- Les tableaux permettent de stocker plusieurs éléments de même type
- Lorsque le type de ces éléments possède un ordre total, on peut donc les ranger en ordre croissant ou décroissant
- Trier un tableau c’est donc ranger les éléments d’un tableau en ordre croissant ou décroissant
  Dans ce cours on ne fera que des tris en ordre croissant
- Il existe plusieurs méthodes de tri qui se différencient par leur complexité d’exécution et leur complexité de compréhension pour le programmeur
    - Tri par sélection
    - Tri fusion
    - Tri par insertion
    - Tri rapide

Propriétés d’un algorithme de tri

- **Tri en place :** s'il n'utilise qu'un nombre très limité de variables et qu’il modifie directement la structure qu’il est en train de trier. Ce caractère peut être très important si on ne dispose pas de beaucoup de mémoire.
- **Tri stable :** Un tri est dit stable s'il préserve l’ordonnancement initial des éléments que l'ordre considère comme égaux.
- **Tri interne et externe :** Un tri interne s'effectue entièrement en mémoire centrale tandis qu'un tri externe utilise des fichiers sur une mémoire de masse pour trier des volumes trop importants pour pouvoir tenir en mémoire centrale.
- **Progressif :** Lorsque le trie fourni au fur et à mesure de son exécution, les éléments dans leur ordre final (offre la possibilité d’une exécution parallèle).

Tri par insertion

- Principe :

Le tri par insertion est aussi appelé tri du joueur de cartes ;
son principe est simple :
-> Pour i *de* 1 à $n - 1$, insérer le $i + 1^{ème}$ élément $T[i]$ dans $T[0 \ldots i - 1]$ (qui est déjà trié !).

Tri par insertion

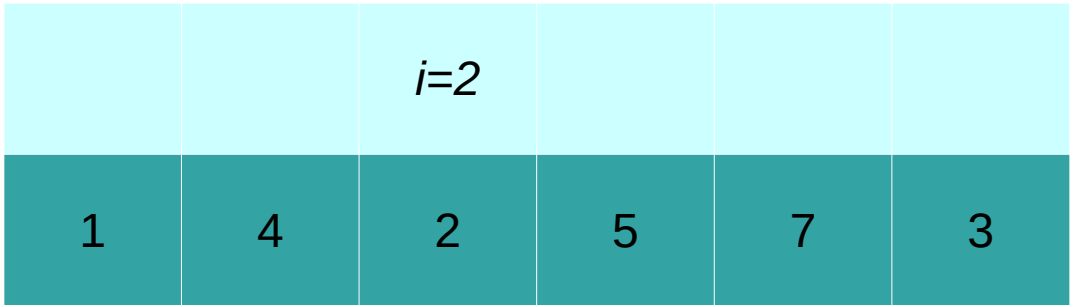
Algorithme :

```
tri_insertion( tableau[ ] ){
    taille = tableau.length
    pour i de 1 à taille {
        index = tableau[i]
        j = i-1
        tantque j >=0 et tableau[j] > index {
            tableau[j+1] = tableau[j]
            j--
        }
        tableau[j+1] = index
    }
}
```

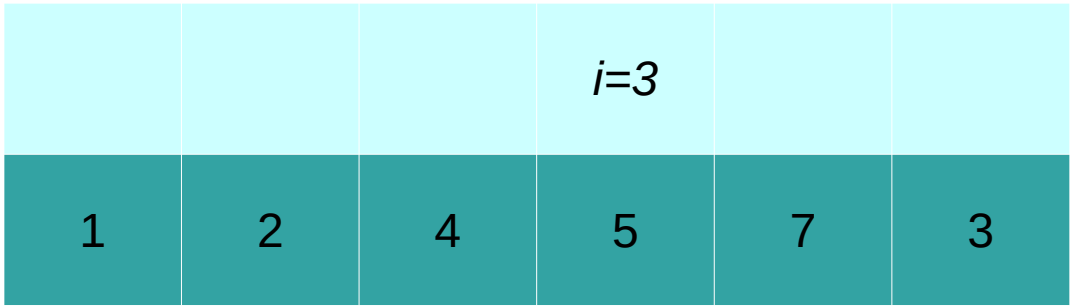
Tri par insertion

	i=1				
1	4	2	5	7	3

Tri par insertion



Tri par insertion



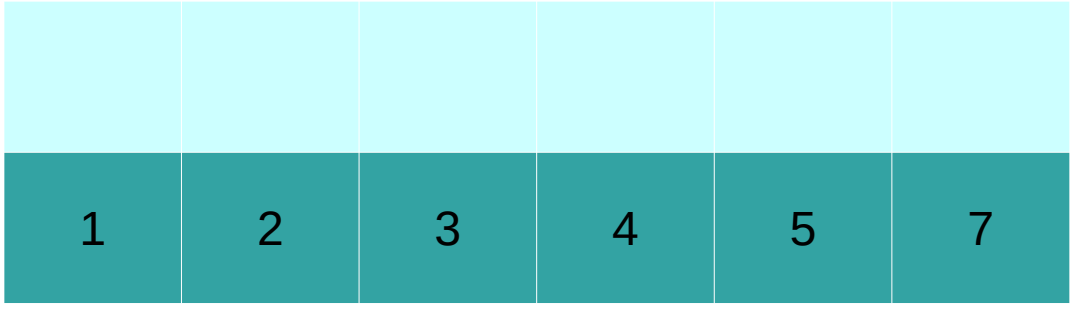
Tri par insertion

				i=4	
1	2	4	5	7	3

Tri par insertion

					i=5
1	2	4	5	7	3

Tri par insertion



Tri par insertion

Voir programme AlgoTri.c
→ Ouvrir avec un logiciel (un IDE quelconque)
→ Ouvrir l'emplacement dans un terminal

On va utiliser GNU Compiler Collection
→ gcc -Wall AlgoTri.c -o Tri
L'option **Wall** (warnings all) active tous les messages d'avertissement du compilateur.
Il est conseillé de toujours utiliser cette option afin de s'assurer que le code source utilisé est parfait.
L'option **-o** permet de choisir le nom attribué à l'exécutable (Tri dans le cas présent).
En l'absence de cette option, la commande de compilation doit être gcc -Wall AlgoTri.c et dans ce cas, en l'absence de précision sur le nom de l'exécutable, le compilateur lui attribue par défaut le nom a.out.

Pour exécuter le fichier essai, on ouvre le Terminal dans le répertoire du programme et on saisit la commande ./Tri

→ ./Tri

Tri par insertion

Complexité

La complexité du tri par insertion est $\theta(n2$) dans le pire cas et en moyenne, et linéaire ($O(n)$) dans le meilleur cas.

Plus précisément :

- Dans le pire cas, atteint lorsque le tableau est trié à l'envers, l'algorithme effectue de l'ordre de $n^2/2$ affectations et comparaisons
- Si les éléments sont distincts et que toutes leurs permutations sont équiprobables (ie avec une distribution uniforme), la complexité en moyenne de l'algorithme est de l'ordre de $n^2/4$ affectations et comparaisons
- Si le tableau est déjà trié, il y a $n - 1$ comparaisons et au plus n affectations

Tri par insertion

Complexité

Dans le cas où nous avons un tableau à trier qui contient n éléments, nous aurons :
$1 + 2 + 3 + \cdots . + n - 3 + n - 2 + n - 1$ décalages.

- Que vaut cette somme $S = 1 + 2 + 3 + \ldots . + n - 3 + n - 2 + n - 1$?
- Somme de suite numérique S=n.(n-1)/2

Écrivant S par rapport à n :
En mettant : $S + S = n + n + n + \ldots . + n + n + n$ (puisque $1 + n - 1 = n$, $2 + n - 2 = n$, $3 + n - 3 = n$, $\ldots . , n - 3 + 3 = n$, $n - 2 + 2 = n$ *et* $n - 1 + 1 = n$)
Soit, $2 * S = n + n + n + \ldots . + n + n + n$
Si vous comptez bien nous avons n-1 fois n, ce que l'on peut écrire :
$2 . S = n(n - 1)$
→ S=n.(n-1)/2

Tri par insertion

Exercice

Tri rapide

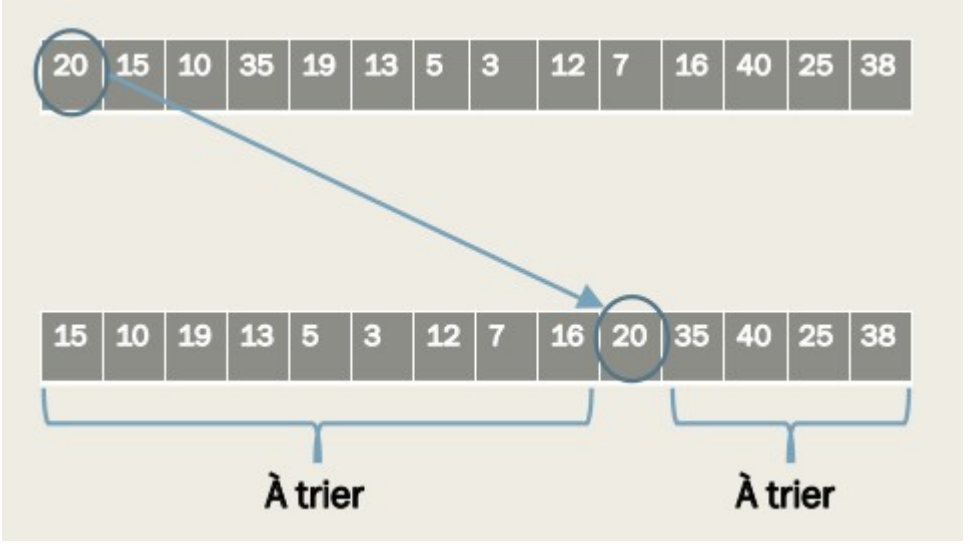
Principe :

Le tri rapide est un tri récursif

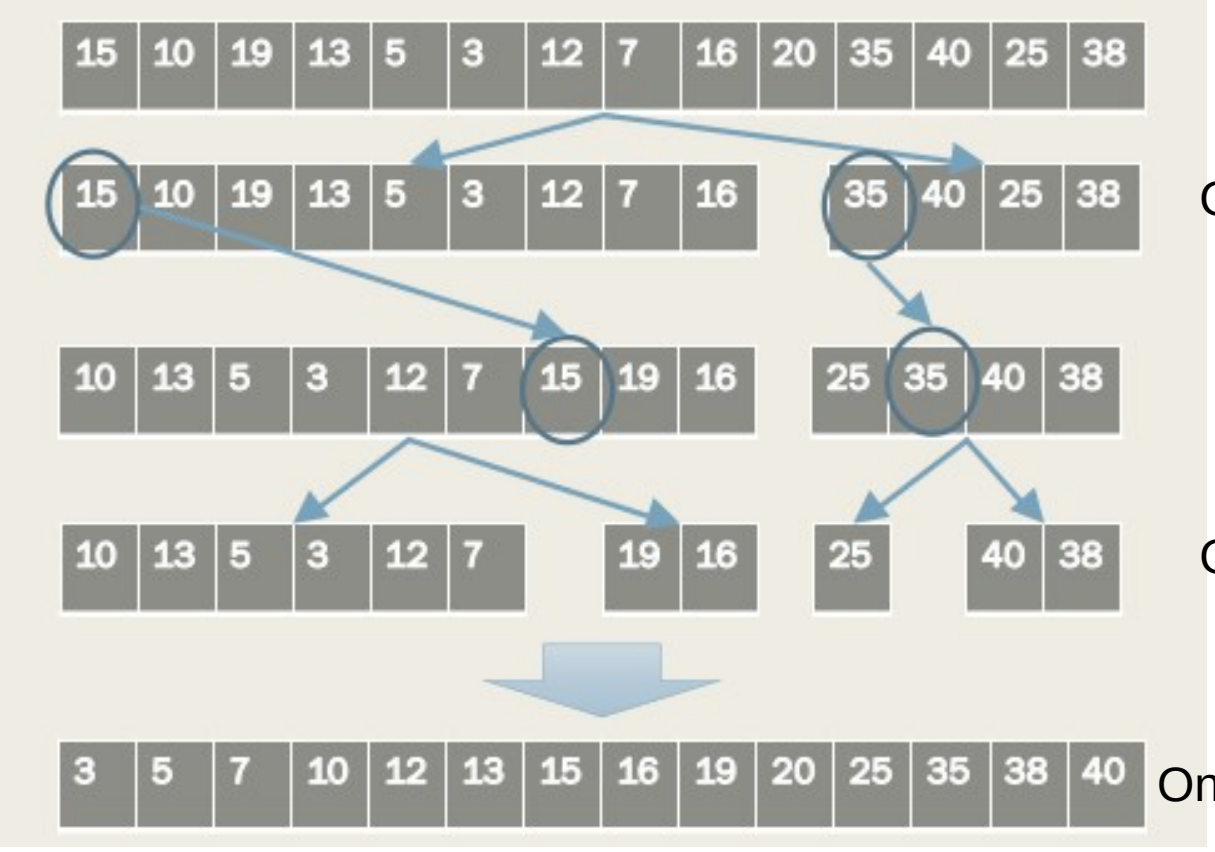
Étant donné un tableau de $T[1, \ldots, n]$:

- Choisir un élément (le “pivot”) p dans T
- Placer les éléments inférieurs à p au début de T
- Placer p à sa place dans T
- Placer les éléments supérieurs à p à la fin de T
- En utilisant un tri rapide, trier la première partie de T puis la seconde

Tri rapide


20 15 10 35 19 13 5 3 12 7 16 40 25 38
15 10 19 13 5 3 12 7 16 20 35 40 25 38
À trier
À trier

Tri rapide



Pivot = 20

On range en 2 tableaux

Pivots = 15 et 35

On range en 4 tableaux

etc

On regroupe les tableaux

Tri rapide

Principe de l’algorithme

Le tri rapide utilise le principe de diviser pour régner, c’est-à-dire que l’on va choisir un élément du tableau (qu’on appelle pivot), puis l’on réorganise le tableau initial en deux sous tableaux :

- L’un contenant les éléments inférieurs au pivot.
- L’autre contenant les éléments supérieurs au pivot.

On continue ce procédé (qu'on appelle partitionnement, c'est-à-dire choisir un pivot et réorganiser le tableau) jusqu’à se retrouver avec un tableau découpé en N sous tableaux (N étant la taille du tableau), qui est donc trié.

Exemple
Prenons 5, 9, 7, 3, 8 comme suite de nombres, et trions la dans l'ordre croissant avec l'algorithme du tri rapide :

Tri rapide

5, 9, 7, 3, 8 -> on choisit le **pivot**, dans notre cas je choisis l'élément du milieu, 7.

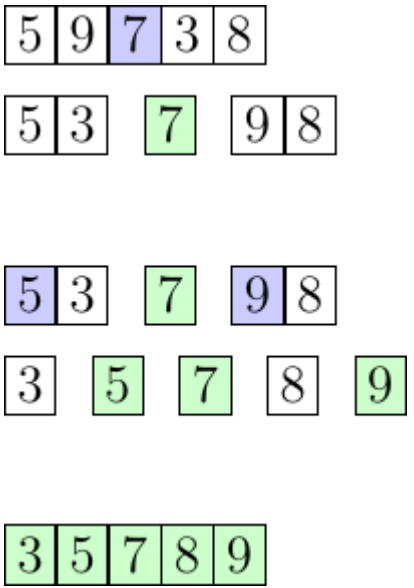
5, 3 | **7** | 9, 8 -> on découpe le tableau en trois parties, une partie avec des éléments inférieurs au pivot (5 et 3), la partie contenant le pivot (7), et une partie avec les éléments supérieurs au pivot (9 et 8). On peut déjà dire qu'on a placé le pivot à sa place définitive dans le tableau, puisque les autres éléments sont soit supérieurs soit inférieurs à lui.

5, 3 | 7 | 9, 8 -> on recommence en choisissant de nouveau un pivot pour chaque sous tableaux créés.

3 | 5 | 7 | 8 | 9 -> dernière étape du partitionnement, désormais aucuns sous tableaux ne contient plus d'un élément, le tri est donc terminé.

3, 5, 7, 8, 9

Tri rapide



On utilise le principe de récursivité pour implémenter notre tri rapide.
Les appels récursifs s'arrêtent quand le sous tableau actuel n'a plus qu'un seul élément, sinon on partitionne notre tableau (choix du pivot et réorganisation), puis on recommence l'opération sur les deux parties du tableau ne contenant pas le pivot (la partie où les éléments lui sont inférieurs, et la partie où ils sont supérieurs).

Tri rapide

Exercice

Tri fusion