CHAPITRE 1 : LES ALGORITHMES DE TRI

I. Le problème du tri :

Un algorithme de tri est un algorithme qui résout un problème de tri :

- **Entrée :** une séquence (liste) de n objets **$(a_0,a_1,\cdots,a_{n-2},a_{n-1})$.** Ces objets peuvent être des entiers, des chaînes de caractères . . . mais il faut qu'on puisse les ordonner par une relation d'ordre
- **Sortie :** un réarrangement **$(a'_0,a'_1,\cdots,a'_{n-2},a'_{n-1})$** de la séquence d'entrée telle que : $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-2} \leq a'_{n-1}$. Ce réarrangement peut être effectué sur place par modification du même conteneur (liste) de la séquence initiale ou par création d'un nouveau conteneur pour la séquence ordonnée.

II. Importance en informatique :

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

✍ **Dans ce chapitre, pour simplifier, on se cantonnera au tri de listes de nombres.**

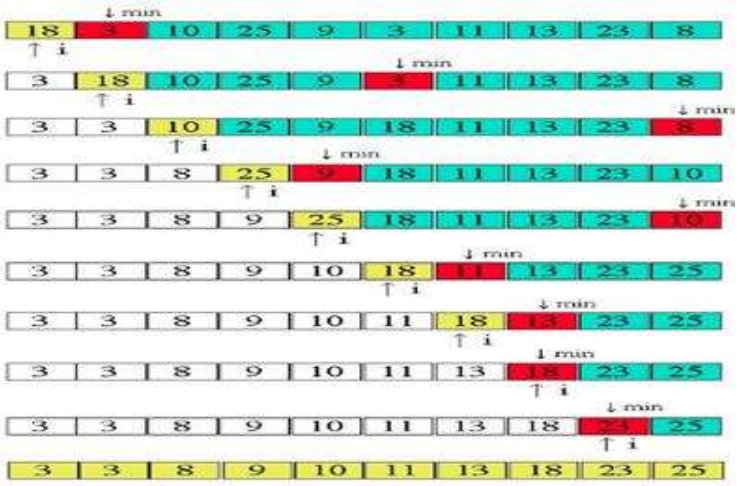
III. Le tri par sélection :

III-1. Algorithme :

Le principe du tri par sélection d'une liste L = [L[0], L[1]….., L[len(L)-1]]:

- Pour chaque entier i (0 ≤ i ≤ len(L)-2) :
  - ➢ parcourir les éléments L[i], L[i + 1], . . ., L[len(L)-1]], retenir l'indice k du plus petit.
  - ➢ placer au rang i le plus petit élément d'indice k (en échangeant L[i] et L[k]).

Exemple :



III-2. Code Python

```
def tri_selection( L ):
    for i in range(len(L)-1):
        k=i
        for j in range(i+1,len(L)):
            if L[j]<L[k]:
                k=j
        L[i],L[k]=L[k],L[i]
```

III-3. Temps d'exécution :

Pour tester le temps d'exécution, on a utilisé un PC Core 2 duo 2 Ghz - 2 Go de RAM avec Python 3.2. Les nombres à trier sont des entiers aléatoires compris entre -100 et 100. Tous les temps donnés sont en secondes :

Taille de la liste	Temps d'exécution
N=0	0.0
N=5000	1.7009999752044678
N=10000	7.036999940872192
N=15000	15.96999979019165
N=20000	27.799999952316284
N=25000	43.67999982833862
N=30000	60.85599994659424

IV. Le tri par Bulle (ou par échange) :

IV-1. Algorithme :

La technique d'échange consiste à comparer les éléments de la liste, 2 par 2, et à les permuter (ou échanger) s'ils ne sont pas dans le bon ordre, jusqu'à ce que la liste soit complètement triée.
Le tri Bulle effectue des comparaisons entre voisins.

- À la première étape, L[0] et L[1] sont comparés et éventuellement permutés.
  Ensuite, l'algorithme compare L[1] et L[2], et ainsi de suite jusque L[n-2] et L[n-1].
  Etant parti de L[0] jusqu'à L[n-2], on peut voir qu'après cette première étape, L[n-1] contiendra le maximum des valeurs de L, la sous-liste de L d'indices 0..(n-2) contenant les autres valeurs déjà légèrement réorganisées.
- À la 2ème étape, on recommence donc les comparaisons/échanges pour les valeurs entre L[0] et L[n-2], ce qui aura pour effet de mettre en place dans L[n-2], le second maximum et de réorganiser à nouveau légèrement les autres valeurs dans la sous-liste de L d'indices 0..(n-3).
- À la ième étape, on devra donc réorganiser la sous-liste de L d'indices 0..(n-i)

Exemple :

3	7	2	6	5	1	4
3	2	6	5	1	4	7
2	3	5	1	4	6	7
2	3	1	4	5	6	7
2	1	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7

IV-2. Code Python

```
def tri_bulle( L ):
    n = len(L)
    for j in range(n-1,0,-1):
        for i in range(j):
            if L[i] > L[i+1]:
            L[i],L[i+1]=L[i+1],L[i]
```

IV-3. Temps d'exécution :

Pour tester le temps d'exécution, on a utilisé un PC Core 2 duo 2 Ghz - 2 Go de RAM avec Python 3.2. Les nombres à trier sont des entiers aléatoires compris entre -100 et 100. Tous les temps donnés sont en secondes :

Taille de la liste	Temps d'exécution
N=0	0.0
N=5000	2.43399977684021
N=10000	8.674000024795532
N=15000	19.796000003814697
N=20000	35.66200017929077
N=25000	55.33300018310547
N=30000	79.20099997520447

V. Le tri par insertion:

V-1. Algorithme :

La technique consiste à considérer chaque élément à trier, un par un et à l'insérer en bonne place relative dans la partie des éléments déjà triés aux étapes précédentes.
À chaque étape, l'insertion d'un élément est effectuée.

- Au départ, L[0] est l'élément de référence.
- À l'étape 1, l'élément L[1] est placé correctement par rapport à la partie déjà triée L[:1], c'est-à-dire uniquement L[0].
- À la ième étape, L[i] est donc inséré dans la sous liste L[:i].

Exemple :

3	7	2	6	5	1	4
3	7	2	6	5	1	4
2	3	7	6	5	1	4
2	3	6	7	5	1	4
2	3	5	6	7	1	4
1	2	3	5	6	7	4
1	2	3	4	5	6	7

V-2. Code Python

```
def tri_insertion( L ):
    n = len(L)
    for i in range( 1,n):
        x=L[i]
        for j in range(i):
            if L[j] > x:
            L[i:i+1]=[]
            L[j:j]=[x]
            break
```

V-3. Temps d'exécution :

Pour tester le temps d'exécution, on a utilisé un PC Core 2 duo 2 Ghz - 2 Go de RAM avec Python 3.2. Les nombres à trier sont des entiers aléatoires compris entre -100 et 100. Tous les temps donnés sont en secondes :

Taille de la liste	Temps d'exécution
N=0	0.0
N=5000	1.373000144958496
N=10000	5.241999864578247
N=15000	11.871999979019165
N=20000	20.918999910354614
N=25000	33.305999994277954
N=30000	46.76900005340576

VI. Les Tris récursives :

Pourquoi un autre algorithme de tri ? Parce que les algorithmes de tri par sélection, par insertion ou par bulles sont lents lorsque le nombre de données à trier est grand.

✍ L'algorithme de tri rapide ou par fusion, est beaucoup plus rapide pour les listes de grande taille.

✍L'algorithme de tri rapide ou par fusion se programme naturellement de façon **récursive**

✍ L'algorithme de tri rapide ou par fusion est basé sur le principe **diviser pour régner**.

Diviser Pour Régner :

Diviser pour régner (divide and conquer) est une technique algorithmique consistant à abandonner par itération ou récursivité une branche du traitement afin de diminuer l'effort à faire pour obtenir la solution à un problème.

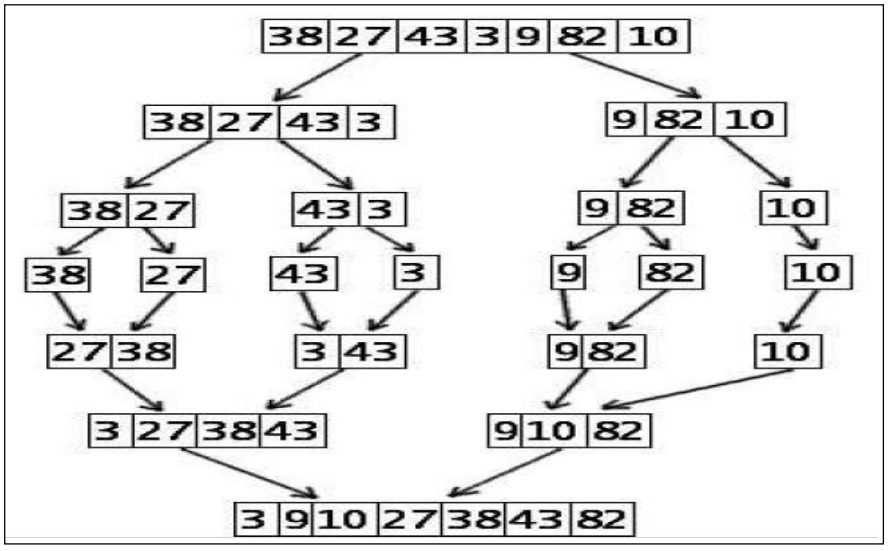
Un exemple simple est la recherche dichotomique, qui consiste à diviser un ensemble de données ordonnés en deux, dont l'un des sous ensembles est abandonné au profit de l'autre.

VI-1. Le tri par fusion (Merge Sort) :

a. Algorithme :

- Partant d'une liste d'éléments à trier, on commence par diviser cette liste en deux sous-listes (presque de même taille)
- on tri les deux sous listes
- puis on fusionne les sous listes triées.
- Une liste à un élément est triviale à trier et forme la condition d'arrêt de la procédure récursive.

Exemple :



b. Code Python

- Version itérative pour la fonction **Fusion**

```
def fusion(L1,L2) :
        L=[]
        i,j=0,0
        while i<len(L1)  and  j<len(L2):
                if L1[i]<L2[j]:
                        L+=[L1[i]]
                        i=i+1
                else:
                        L+=[L2[j]]
                        j=j+1
        return L+L1[i : ]+L2[j : ]

def tri_fusion(L):
        if len(L) <=1: return L
        return fusion(tri_fusion(L[:len(L)//2]),tri_fusion(L[len(L)//2:]))
```

- Version récursive pour la fonction **Fusion**

```
def fusion(L1,L2) :
        if L1==[] :return L2
        if L2==[] :return L1
        if L1[0]<L2[0] :
                return [L1[0]]+fusion(L1[1 :],L2)
        else :
                return [L2[0]]+fusion(L1,L2[1 :])

def tri_fusion(L):
        if len(L)==1: return L
        return fusion(tri_fusion(L[:len(L)//2]),tri_fusion(L[len(L)//2:]))
```

c. Temps d'exécution :

Pour tester le temps d'exécution, on a utilisé un PC Core 2 duo 2 Ghz - 2 Go de RAM avec Python 3.2. Les nombres à trier sont des entiers aléatoires compris entre -100 et 100. Tous les temps donnés sont en secondes :

Taille de la liste	Temps d'exécution
N=0	0.0
N=5000	0.07800006866455078
N=10000	0.14100003242492676
N=15000	0.23399996757507324
N=20000	0.312000036239624
N=25000	0.40599989891052246
N=30000	0.49900007247924805

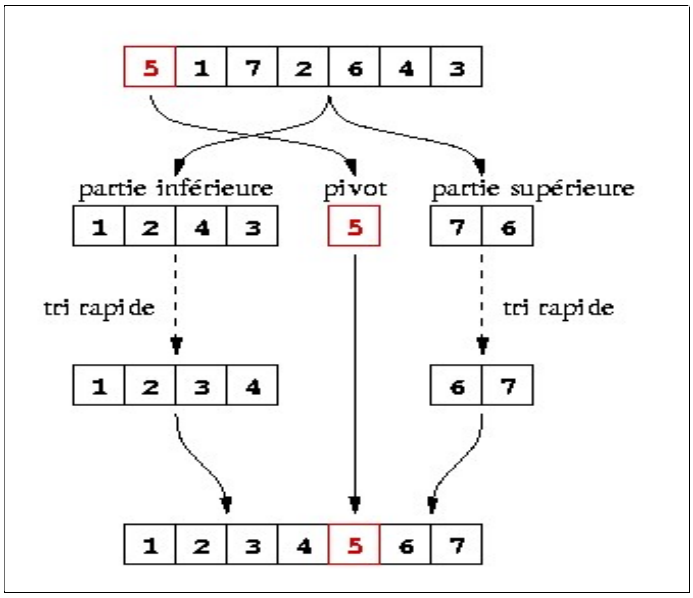
VI-2. Le tri rapide (Quicksort):

d. Algorithme :

La méthode consiste à placer un élément de la liste (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite.
Cette opération s'appelle **le partitionnement**.
Pour chacune des sous-listes, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.
Concrètement, pour partitionner une sous-liste :

- Choisir un élément du tableau appelé pivot (premier , le dernier ou aléatoire).
- Ordonner les éléments du tableau par rapport au pivot.
- Appeler récursivement le tri sur les sous-tableaux à gauche et à droite du pivot.

Exemple :



e. Code Python

```
def tri_rapide(L):
        if len(L)<=1: return L
        Linf=[]
        Lsup=[]
        Lpivot=[]
        p=L[0]    #le choix du pivot
        for i in range(len(L)):
                if L[i]>p:
                        Lsup+=[L[i]]
                elif L[i]<p:
                        Linf+=[L[i]]
                else:
                        Lpivot+=[L[i]]
        return tri_rapide(Linf)+Lpivot+tri_rapide(Lsup)
```

f. Temps d'exécution :

Pour tester le temps d'exécution, on a utilisé un PC Core 2 duo 2 Ghz - 2 Go de RAM avec Python 3.2. Les nombres à trier sont des entiers aléatoires compris entre -100 et 100. Tous les temps donnés sont en secondes :

Taille de la liste	Temps d'exécution
N=0	0.0
N=5000	0.016000032424926758
N=10000	0.04699993133544922
N=15000	0.06299996376037598
N=20000	0.09299993515014648
N=25000	0.12400007247924805
N=30000	0.1400001049041748

VII. Comparaison entre les tris :

Taille	Temps d'exécution				
	Sélection	Bulle	Insertion	Fusion	Rapide
N=0	0.0	0.0	0.0	0.0	0.0
N=5000	1.7009999752044678	2.43399977684021	1.373000144958496	0.07800006866455078	0.016000032424926758
N=10000	7.036999940872192	8.674000024795532	5.241999864578247	0.14100003242492676	0.04699993133544922
N=15000	15.96999979019165	19.796000003814697	11.871999979019165	0.23399996757507324	0.06299996376037598
N=20000	27.799999952316284	35.66200017929077	20.918999910354614	0.312000036239624	0.09299993515014648
N=25000	43.67999982833862	55.33300018310547	33.305999994277954	0.40599989891052246	0.12400007247924805
N=30000	60.85599994659424	79.20099997520447	46.76900005340576	0.49900007247924805	0.1400001049041748