MÉTHODES DE TRIS

Définition

Un algorithme de tri est, en informatique ou en mathématiques, est un algorithme qui permet d'organiser une collection d'objets selon un ordre déterminé. Les objets à trier font donc partie d'un ensemble muni d'une relation d'ordre. Les ordres les plus utilisés sont l'ordre numérique et l'ordre lexicographique.

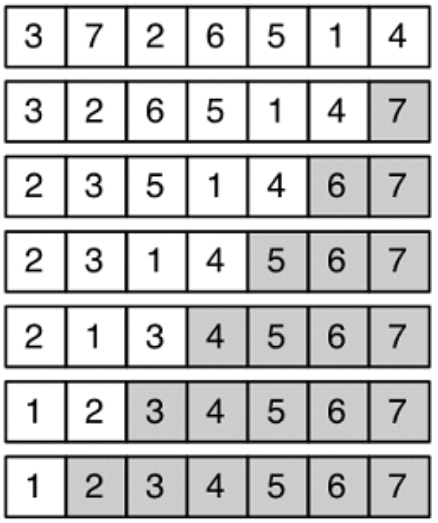
Source : Wikipédia

1 - TRI A BULLES

❖ ***Principe :***

Comparaison 2 à 2 des éléments adjacents et échange s'ils ne sont pas ordonnés. Le programme s'arrête lorsqu'on parcourt la liste sans faire d'échange

Comme les bulles, les plus grands éléments remontent en fin de liste.



❖ ***Programme :***

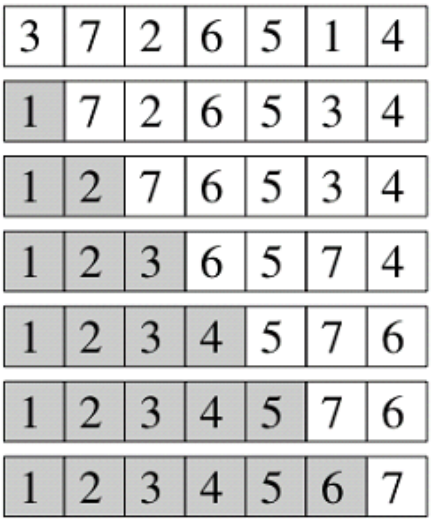
```
def  triBulles (T):
  echange = True
  while echange == True :
     echange = False
     for i in range(0,len(T)-1) :
         if(T[i]> T[i+1]):
             T[i],T[i+1]=T[i+1],T[i]
              echange = True
```

Complexité : $O(N^2)$

2 - TRI PAR SELECTION

❖ ***Principe :***

- Recherche du plus petit élément du tableau et échange avec le premier élément
- Recherche du plus petit élément du tableau entre les positions 2 et n -1 et échange avec le second élément
- ...
- Recherche du plus petit élément entre les positions n-2 et n-1 et échange avec élément en position n-2



❖ ***Programme*:**

```
def triSelection (T):
  for i in range(0,len(T)-1):
         pmin= i
         for j in range (i+1, len(T)):
              if T[j] < T [pmin]: pmin= j
         if pmin!= i:
```

Complexité : $O(N^2)$

```
T[i],T [pmin] =  T[pmin], T[i]
```

3 - TRI PAR INSERTION

❖ ***Principe*** **:**

- La liste étant triée jusqu'à l'élément i-1,
- insérer l'élément i à sa place parmi les i premiers éléments

❖ ***Programme*****:**

```
def tri_Insertion(T) :
    n=len(T)
    for i in range(1,n):
        v = T[i]
        j = i
        while j>0 and T[j-1]> x :
            T[j]= T[j-1]
            j = j-1
        T[j]= v
```

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

Complexité : $O(N^2)$

4- TRI RAPIDE (QUICK SORT)

Le tri rapide (en anglais quick sort) est un algorithme de tri inventé par C.A.R. Hoare en 1961 et fondé sur la méthode de conception diviser pour régner.

Son principe consiste à séparer l'ensemble des éléments en deux parties.

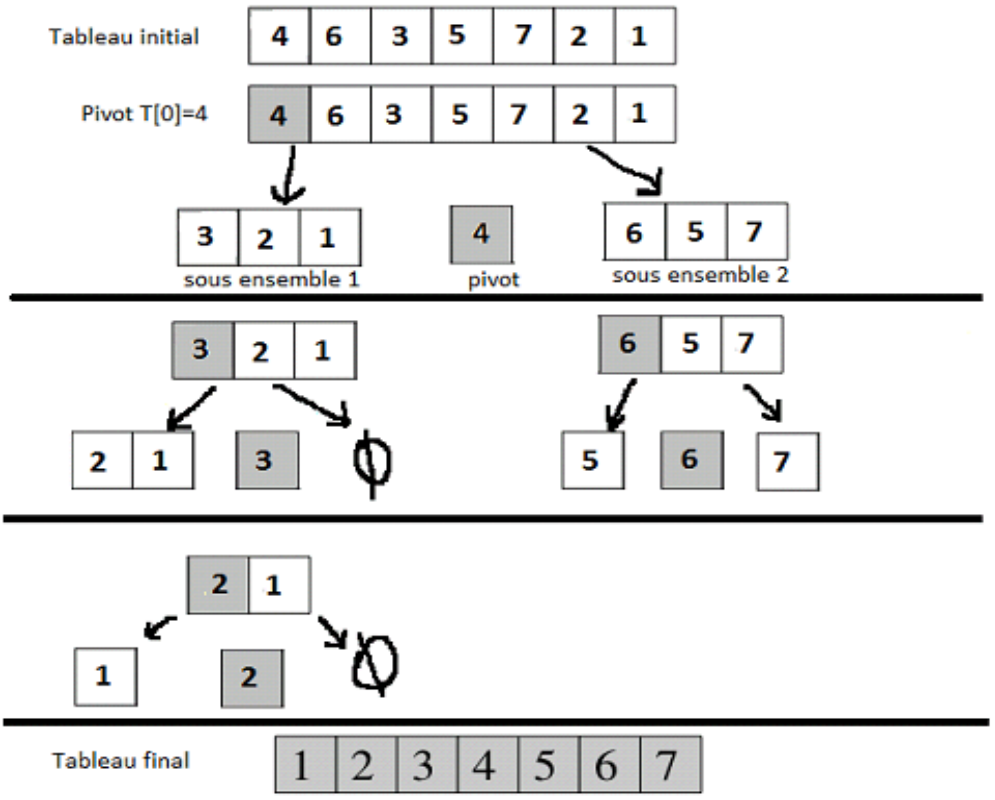
❖ ***Principe*** **:**

L'algorithme peut s'effectuer récursivement:

- une valeur **pivot** est choisie au hasard. (peut aussi être le premier ou le dernier élément de la liste à trier).
- On crée un premier sous-ensemble constitué des éléments plus petits que le pivot. On les place à gauche du pivot.
- On crée de la même façon un deuxième sous-ensemble constitué des éléments plus grands que le pivot. On les place à droite de celui-ci.
- On procède de même, par récursivité, sur les deux sous-ensembles jusqu'à obtenir des sous ensembles d'un seul élément.

❖ ***Exemple 1 :***

On va illustrer le tri rapide sur le tableau **T=[4,6,3,5,7,2,1] (Pivot=T[0]).**



❖ ***Exemple 2 :***

- Soit la liste : **L = [4, 23, 3, 42, 2, 14, 45, 18, 38, 16]**
- Prenons comme pivot la dernière valeur : **pivot = 16**
- Nous obtenons donc :

  L1 = [4, 14, 3, 2]

  L2 = [23, 45, 18, 38, 42]
- A cette étape voici l'arrangement de L :
  L = L1 + **[pivot]** + L2 = [4, 14, 3, 2, **16**, 23, 45, 18, 38, 42]

- En appliquant la même démarche au deux sous-listes : L1 (pivot=2) et L2 (pivot=42)

  [**4, 14, 3, <u>2</u>, 16, 23, 45, 18, 38, <u>42</u>**]
- nous obtenons :

L11=[ ] liste vide	L21=[**23, 38, 18**]
L12=[**3, 4, 14**]	L22=[**45**]
L1=L11 + pivot + L12 = (**<u>2</u>,3, 4, 14**)	L2=L21 + pivot + L22 = (**23, 38, 18, <u>42</u>, 45**)

❖ ***Programme*: (Solution 1)**

```
def Trirapide(L):
    if L==[]: return[]
    #on va balayer la liste L et repartir les valeurs
    n=len(L)
    L1=[]
    L2=[]
    pivot=L[0] #ici on a pris le premier élément comme pivot.
    for k in range(1,n):
         if L[k]<= pivot:
              L1.append(L[k]) #L1 reçoit les éléments plus petits
         else:
              L2.append(L[k]) #L2 reçoit les éléments plus grands
    L = Trirapide(L1) + [pivot] + Trirapide(L2)
    return L
```

❖ ***Programme: (Solution 2)***

Pour implémenter l'algorithme tri rapide on a besoin de programmer trois fonctions:

1. La fonction : **`partitionner(T, premier, dernier)`** prend le tableau **T** et deux indices **premier** et **dernier** en arguments, (avec premier et dernier inclus). On suppose qu'il y a au moins un élément dans ce segment,
2. La fonction : **`tri_Rapide_Partition(T,premier,dernier):`** qui permet de trier le tableau **T** en le partitionnant récursivement en sous-tableaux.
3. La fonction : **`triRapide(T)`** permettant de trier le tableau T en utilisant la fonction tri_Rapide_Partition.

1-

```
def partitionner(T, premier, dernier):
    pivot = T[dernier]     #on prend le dernier élément comme pivot
    j = premier
    for i in range(premier, dernier) :
         if T[i]<=p :
              T[i],T[j]=T[j],T[i]
              j=j+1
    T[dernier],T[j]=T[j],T[dernier]
    return j        #position définitive du pivot
```

2-

```
def tri_Rapide_Partition (T, premier, dernier):
    if premier < dernier :

       p=partitionner(T, premier, dernier)  #position d'un pivot

       #Trier le sous-Tableau gauche
       tri_Rapide_Partition(T,premier,p-1)

       #Trier le sous-Tableau droit
        tri_Rapide_Partition(T, p+1, dernier)
```

3-

```
def triRapide(T):
   tri_Rapide_Partition( T, 0, len(T) -1 )
```

Appel.

```
>>> L = [ 4, 23, 3, 42, 2, 14, 4,45, 18, 38, 16 ]
>>> triRapide(L)
>>> L
[2, 3, 4, 4, 14, 16, 18, 23, 38, 42, 45]
```

❖ ***Calcul de complexité : (Solution 2)***

- La fonction ***partitionner*** fait toujours exactement **dernier - premier- 1** comparaisons.
- Si La fonction ***partitionner*** détermine un segment de longueur **K** et un autre de longueur **N-1-k**, la fonction ***tri_ Rapide_Partition*** va donc effectuer ***N- 1*** comparaisons par l'intermédiaire de partitionner, puis d'autres comparaisons par l'intermédiaire des deux appels récursifs à la fonction ***tri_Rapide_Partition***

- **Le pire des cas** correspond à **K=0** (c'est à dire quand une des parties est vide et l'autre contient **n-1**), ce qui donne, en notant **C(N)** la complexité du tri d'un tableau de longueur **N**, l'équation de récurrence suivante:

$$C(N)= N-1+ C(N-1)$$

Donc $\mathbf{C(N)= N^2/2}$

d'où la complexité est $O(N^2)$

- **Le meilleur des cas** correspond à un segment coupé en deux moitiés égales, c'est-à-dire **K = N/2**. L'équation de récurrence devient alors:

$$\mathbf{C(N)= N-1+2.C(N/2)}$$

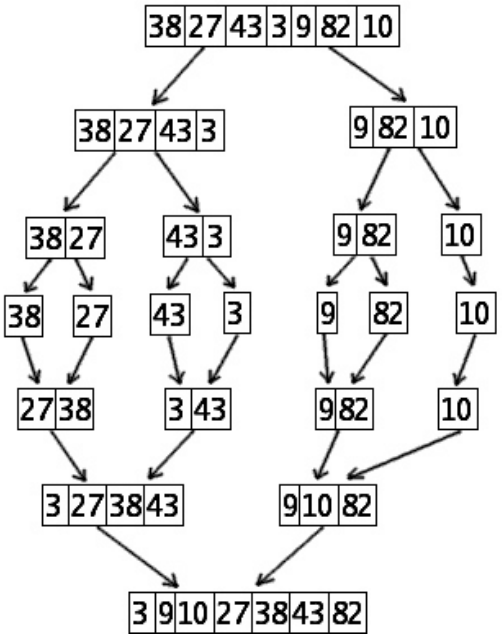
On déduit que $C(N)= N.log(N)$

donc la complexité est $O(N.log(N))$

V- TRI PAR FUSION

❖ ***Principe*** **:**

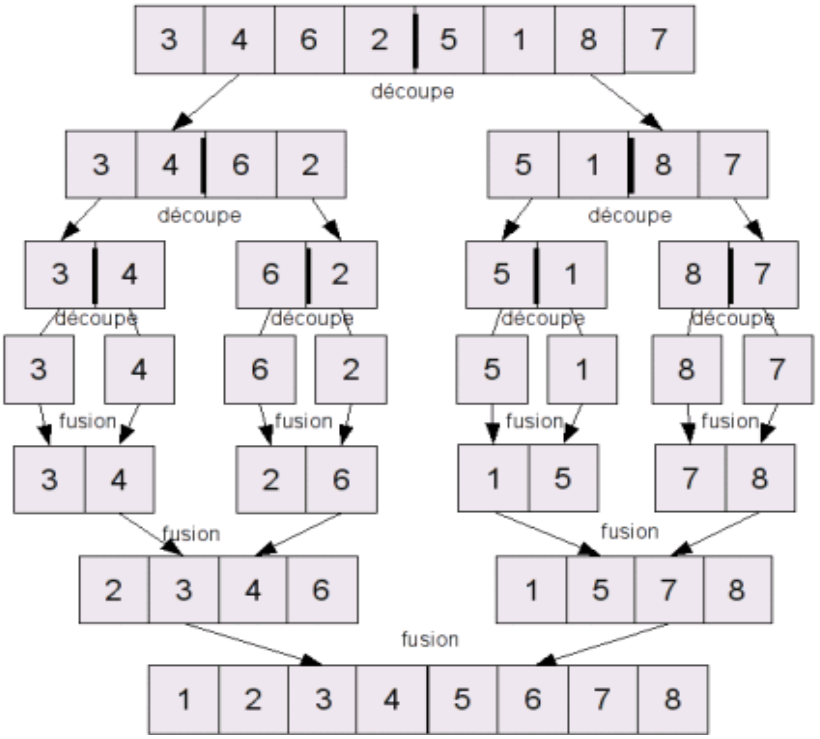
Le tri par **fusion** est un algorithme de tri basé sur la technique algorithmique ***diviser pour régner***. L'opération principale de l'algorithme est la fusion, qui consiste à réunir deux listes triées en une seule.



Le principe de cet algorithme tend à adopter une formulation récursive :

- On découpe les données à trier en deux parties plus ou moins égales
- On trie les 2 sous-parties ainsi déterminées
- On fusionne les deux sous-parties pour retrouver les données de départ

❖ ***Exemple :***



❖ ***Implémentation :***

L'implémentation de cet algorithme repose essentiellement en 2 fonctions :

- **fusion :** Permet de fusionner deux listes triées de telle sorte que la liste résultante la soit aussi.
- **triFusion :** Une fonction récursive qui assure le découpage de la liste et l'appel de la fonction de fusion.

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

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

```
>>> L = [ 4, 23, 3, 42, 2, 14, 4,45, 18, 38, 16 ]
>>> triFusion(L)
>>> L
[2, 3, 4, 4, 14, 16, 18, 23, 38, 42, 45]
```

❖ ***Calcul de complexité (tri par fusion):***

Si on note :

- **C(N)** : le nombre total de comparaisons effectuées par la fonction **triFusion**
- **f (N)** : le nombre total de comparaisons effectuées par la fonction **Fusion**
- Pour trier un tableau de longueur **N**, on a l'équation de récurrence suivante:

  C(N)=2.C(N/2)+ f (N)

  Les deux appels récursifs se font sur deux segments de même longueur **N/2**
- **Dans le meilleur des cas**, la fonction **fusion** n'examine que les éléments de l'un des deux segments car ils sont tous plus petits que ceux de l'autre segment.

  Dans ce cas **f (N)= N/2**,

  donc **C(N)= (½).N.log(N)**

  d'où la complexité est : ***O(N.log(N))***
- **Dans le pire des cas**, tous les éléments sont examinés par la fonction **fusion**

  Dans ce cas **f (N)= N-1,**

  donc **C(N)= N.log(N).** D'où la complexité est :

  O(N.log(N))

❖ ***Comparaison*** **DE COMPLEXITÉ DE DIFFÉRENTES MÉTHODES DE TRIS**

Algorithme de tri	meilleur cas	moyenne	pire cas
Insertion	N	$N^2/4$	$N^2/2$
Sélection	N	$N^2/4$	$N^2/2$
Bulle	N	$N^2/4$	$N^2/2$
Rapide	$N\ log(N)$	$2N\ log(N)$	$N^2/2$
Fusion	$\frac{1}{2}\ N\ log(N)$	$N\ log(N)$	$N\ log(N)$