Chapitre 3
Les algorithmes de tris rapides

Programmation en Python–2ème année MP3–

E-mail
mlahby@gmail.com

28 octobre 2014

Plan

1. Les algorithmes de tris classiques
   - Tri par sélection
   - Tri par bulle
   - Tri par insertion
2. Les algorithmes de tris rapides
   - Tri rapide
   - Tri fusion
   - Démonstration mathématique
3. Comparaison de complexité de différentes méthodes de tris

Tri par sélection

La fonction TriSelection(T,n)

```
def TriSelection(T,n) :
  for i in range(n-1) :
    Posmin=i
    for j in range(i+1,n) :
      if T[j]<T[Posmin] :
        Posmin=j
    #Permutation
    T[i],T[Posmin]=T[Posmin],T[i]
```

Complexité

la complexité est $O(n^2)$

- **Principe**
  Le tri par sélection consiste à chercher la plus petite valeur de la liste, puis de la mettre à la dernière place en l'échangeant avec la première valeur. On réitère alors le procédé sur la liste restreinte aux nombres restants et cela jusqu'à obtenir la liste triée.

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

Tri par bulle

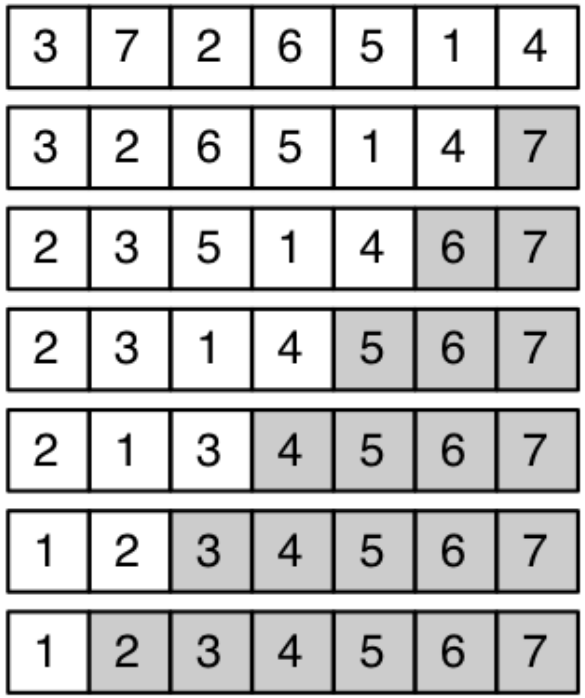
La fonction TriBulle(T,n)

```
def TriBulle(T,n) :
  inversion=True :
  while inversion :
    inversion=False
    for i in range(n-1) :
      if T[i]>T[i+1] :
        #Permutation
        c=T[i]
        T[i]=T[i+1]
        T[i+1]=c
        inversion=False
```

Complexité

la complexité est $O(n^2)$

- **Principe**
  Le principe du tri par bulle consiste à comparer deux à deux les éléments e1 et e2 consécutifs d'un tableau et d'effecteur une permutation si e1 $>$ e2. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation.



Tri par insertion

La fonction TriInsertion(T,n)

```
def TriInsertion(T,n) :
  for i in range(n) :
    j=i-1
    while (j>=0 and T[j]>T[j+1]) :
      #Permutation
      T[j],T[j+1]=T[j+1],T[j]
      j=j-1
```

Complexité

la complexité est $O(n^2)$

- **Principe**
  Le tri par insertion consiste à insérer le premier élément à trier au premier emplacement d'une nouvelle liste, puis on insère chaque nouvel élément à sa bonne place dans la liste dèjà triée. On réitère alors le procédé jusqu'à avoir replacé tous les éléments de la liste initiale. (jeu de cartes ou un paquet de copies).

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

Principe de tri rapide

Définition

Le tri rapide (en anglais quicksort) est un algorithme de tri inventé par C.A.R. Hoare en 19611 et fondé sur la méthode de conception **diviser pour régner**. Il est généralement utilisé sur des tableaux, mais peut aussi être adapté aux autres types de données.

Principe de fonctionnement

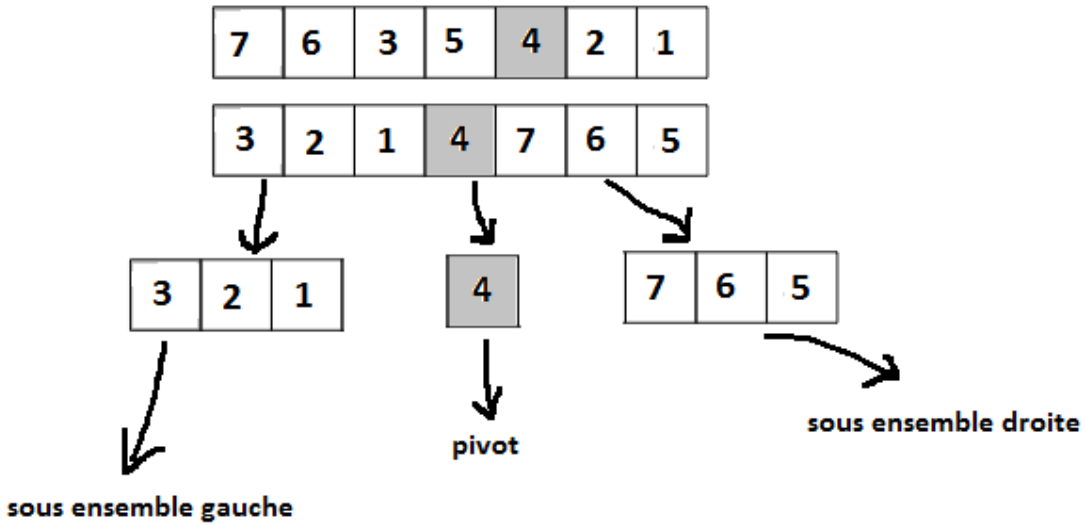
- L'algorithme peut s'effectuer récursivement :
  1. Le premier élément de la liste à trier est appelé "pivot".
  2. On crée un premier sous-enemble constitué des éléments plus petits que le pivot. On les place à gauche du pivot dans l'ordre où ils sont dans la liste à trier.
  3. On crée de la même façon un deuxième sous-enemble constitué des éléments plus grands que le pivot. On les place à droite de celui-ci dans l'ordre où ils apparaissent dans le tableau.
  4. On procède de même, par recursivité, sur les deux sous-ensembles jusqu'à obtenir des sous ensembles d'un seul élément.

Des exemples d'illustration

Exemple 1

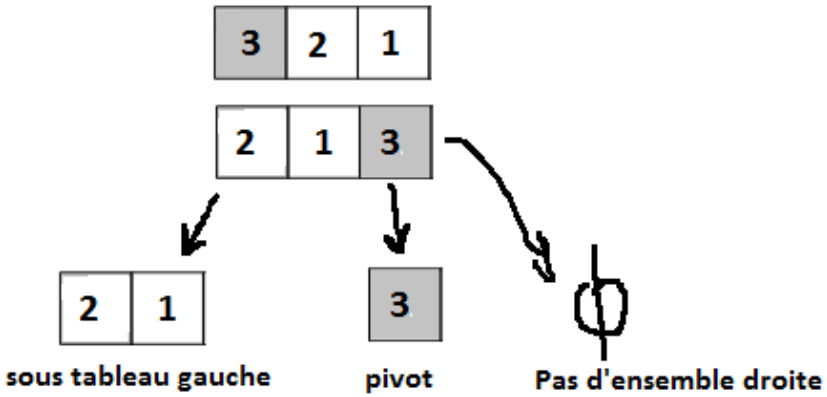
- On va illustrer le tri rapide sur le tableau T=[7,6,3,5,4,2,1].
- On choisit comme **Pivot** une valeur aléotoire du tableau T.

1. **Itération 1 :** Pour trier T[0 :7], on choisit au hasard 4 comme pivot. On place les éléments plus petits que 4, puis 4, puis les autres.

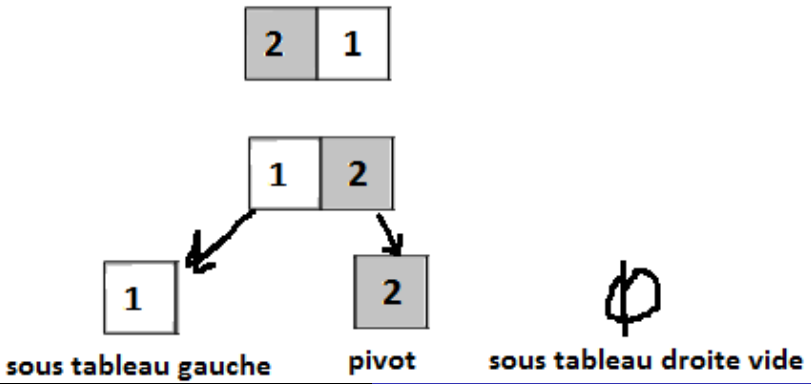


Des exemples d'illustration

2. **Itération 2 :** Pour trier T[0 :3], on choisit 3 comme pivot.
   On place les éléments plus petits que 3, puis 3, puis les autres.

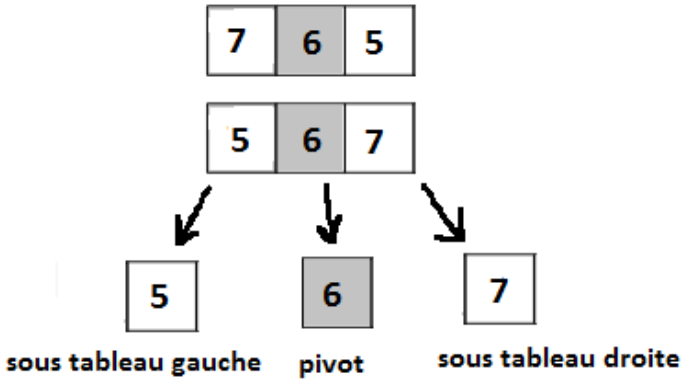


3. **Itération 3 :** Pour trier T[0 :2], on choisit 2 comme pivot..
   On place les éléments plus petits que 2, puis 2, puis les autres.



Des exemples d'illustration

4. **Itération 4 :** Pour trier T[0 :3], on choisit 3 comme pivot.
   On place les éléments plus petits que 3, puis 3, puis les autres.

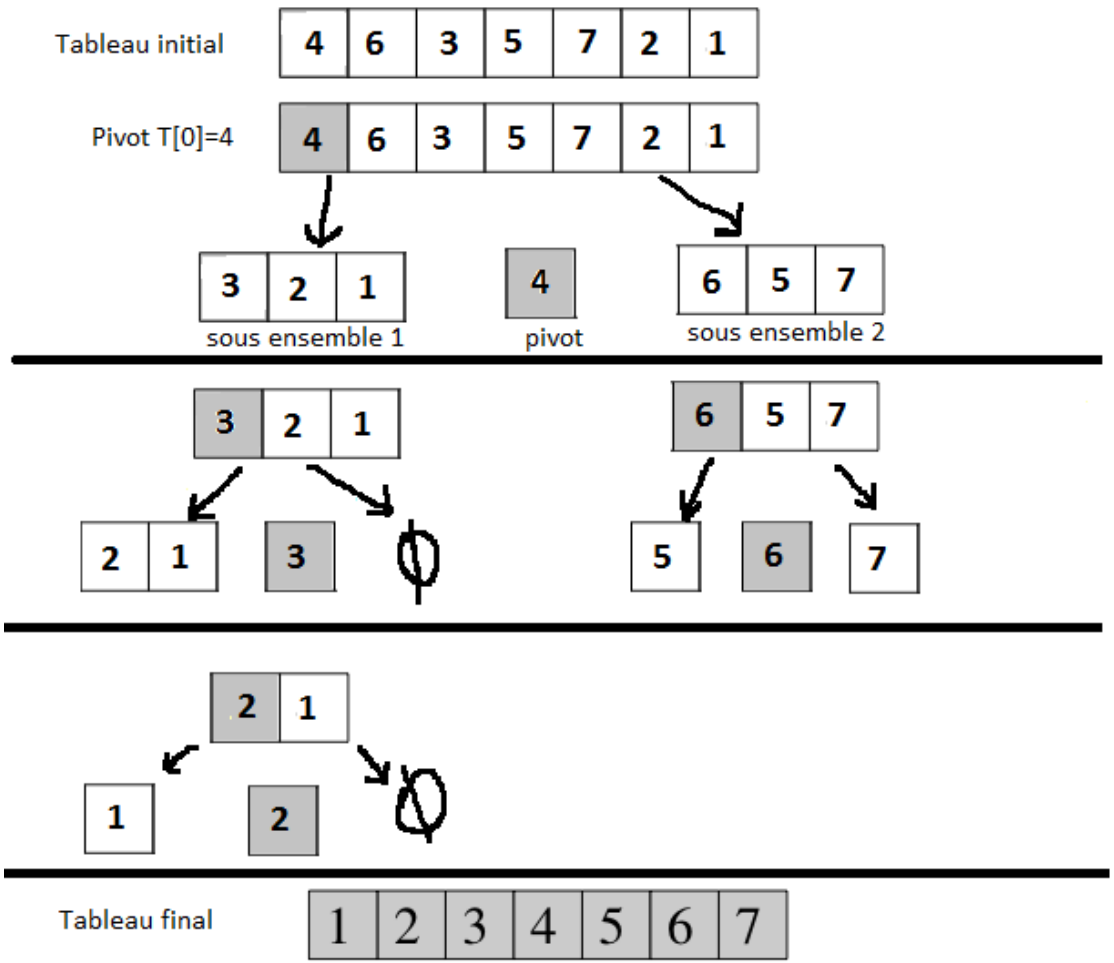


5. **Itération 5 :** Pour trier T[0 :2], on choisit 2 comme pivot..
   On place les éléments plus petits que 2, puis 2, puis les autres.

Des exemples d'illustration

Exemple 2

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



Implémentation de l'algorithme de tri rapide

Tri rapide -Solution 1-

Programme récursif Python

```
def Trirapide(L) :
    if L == [ ] :
        return [ ]
    else :
        n = len(L)-1 #on va balayer la liste L et répartir les valeurs
        L1 = [ ]
        L2 = [ ]
        for k in range(1,n+1) :
            if L[k]<=L[0] :
                L1.append(L[k]) #L1 reçoit les éléments plus petits
            else :
                L2.append(L[k]) #L2 reçoit les éléments plus grands
        L = Trirapide(L1)+[L[0]]+Trirapide(L2)
        return L
```

⇒**Exemple :**

```
>>>T=Trirapide([10,39,21,2,8,6,1])
>>>print(T) vaut [1, 2, 6, 8, 10, 21, 39]
```

Implémentation de l'algorithme de tri rapide

Tri rapide -Solution 2-

- Pour implémenter l'algorithme tri rapide on a besoin de programmer quatre fonctions :
  1. La fonction **Echange(T,i,j)** pour échanger les éléments T[i] et T[j] d'un tableau T
  2. La fonction **Partition(T, premier, dernier)** prend le tableau T et deux indices *premier* et *dernier* en arguments, avec la convention que *premier* est inclus et *dernier* exclu. On suppose qu'il y a au moins un élément dans ce segment,
  3. La fonction **Tri_Rapide_Partition(T, premier, dernier) :** qui permet de trier une liste T en la partitionnant récursivement en sous listes.
  4. La fonction **Tri_Rapide(T)** permettant de trier la liste T en uitlisant l'algorithme tri rapide.

Implémentation de l'algorithme de tri rapide

Tri rapide -Solution 2-

La fonction **Partition(T, premier, dernier)**

```
def partition(T, premier, dernier) :
    pivot = T[premier]
    gauche = premier #On pointe gauche sur l'indice de pivot
    droite = dernier
    flag = 0
    while flag == 0 :
        while T[droite] >= pivot and droite >= gauche :
            droite = droite - 1
        if droite >= gauche :
            Echange(T,gauche,droite)
            pivot=T[droite]
        while gauche <= droite and T[gauche] <= pivot :
            gauche = gauche + 1
        if gauche <= droite :
            Echange(T,gauche,droite)
            pivot=T[gauche]
        if droite < gauche :#condition d'arrêt
            flag = 1
    return droite
```

Implémentation de l'algorithme de tri rapide

Tri rapide -Solution 2-

La fonction **Echange(T,i,j)**

```
def Echange(T,i,j) :
    T[i], T[j] = T[j], T[i]
```

La fonction **Tri_Rapide_Partition(T)**

```
def Tri_Rapide_Partition(T, premier, dernier) :
    if premier < dernier :
        m = partition(T, premier, dernier)
        Tri_Rapide_Partition(T, premier, m-1)
        Tri_Rapide_Partition(T, m+1, dernier)
```

La fonction **Tri_Rapide(T)**

```
def Tri_Rapide(T) :
    "Trie une liste de nombres de manière croissante"
    Tri_Rapide_Partition(T, 0, len(T)-1)
```

Complexité en nombre de comparaisons C(n) -Solution 2-

La fonction partition fait toujours exactement $dernier - premier - 1$ comparaisons. Si la fonction **partition** 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 **partition**, puis d'autres comparaisons par l'intermédiaire des deux appels récursifs à **Tri_Rapide_Partition** (la fonction réécrite avec un seul appel récursif a la même complexité en temps).

- Le pire des cas correspond à K = 0, 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 $C(N) = \frac{N^2}{2}$, 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 :

$$C(N) = N - 1 + 2C(N/2)$$

On déduit que $C(N) = N\ log\ N$, donc la complexité est $O(N\ log\ N)$

Complexité en nombre de affectation A(n) -Solution 2-

En ce qui concerne le nombre d'affectations $A(n)$, on note que la fonction **partition** effectue un appel à **Echange** initial, autant d'appels à **Echange** que d'incrémentations de m, et éventuellement un dernier appel lorsque m != g.

- Le meilleur des cas est atteint lorsque le pivot est toujours à sa place. Il y a alors un seul appel à **Echange**, soit deux affectations. Il est important de noter que ce cas ne correspond pas à la meilleure complexité en termes de comparaisons (qui est alors quadratique).
  Donc $A(N) = 2N$, d'où la complexité est $O(N)$
- Dans le pire des cas, le pivot se retrouve toujours à la position $r - 1$. La fonction **partition** effectue alors $2(dernier - premier)$ affectations. On déduit que $A(N) = N^2$, donc la complexité est $O(N^2)$

Principe de tri fusion

Définition

Le tri fusion (merge sort) est un des premiers algorithmes inventés pour trier un tableau car (selon Donald Knuth) il aurait été proposé par John von Neuman dès 1945 ; il constitue un parfait exemple d'algorithme naturellement récursif qui utilise le concept de la programmation **diviser pour régner**.

Principe de fonctionnement

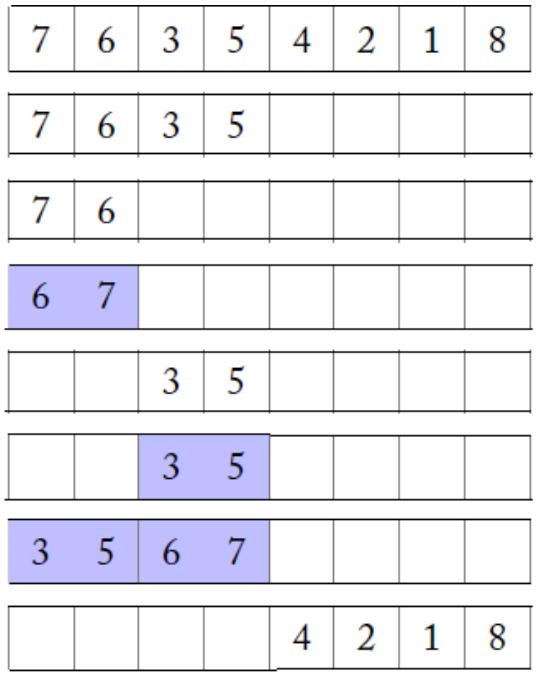
- L'algorithme peut s'effectuer récursivement :
  1. On découpe en deux parties à peu près égales les données à trier.
  2. On trie les données de chaque partie.
  3. On fusionne les deux parties.

Des exemples d'illustration

Exemple 1

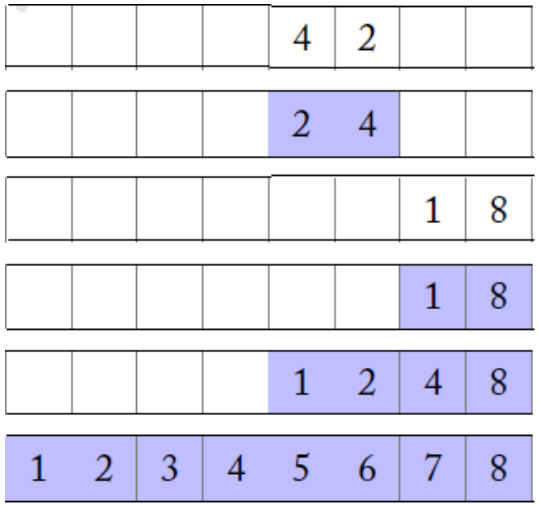
- On va illustrer le tri fusion sur le tableau T=[7,6,3,5,4,2,1,8].

- Pour trier T[0 :8], on trie T[0 :4] et T[4 :8].
- Pour trier T[0 :4], on trie T[0 :2] et T[2 :4].
- Pour trier T[0 :2], on trie T[0 :1] et T[1 :2].
- On fusionne T[0 :1] et T[1 :2].
- Pour trier T[2 :4], on trie T[2 :3] et T[3 :4].
- On fusionne T[0 :1] et T[1 :2]
- On fusionne T[0 :2] et T[2 :4].
- Pour trier T[4 :8], on trie T[4 :6] et T[6 :8].



Des exemples d'illustration

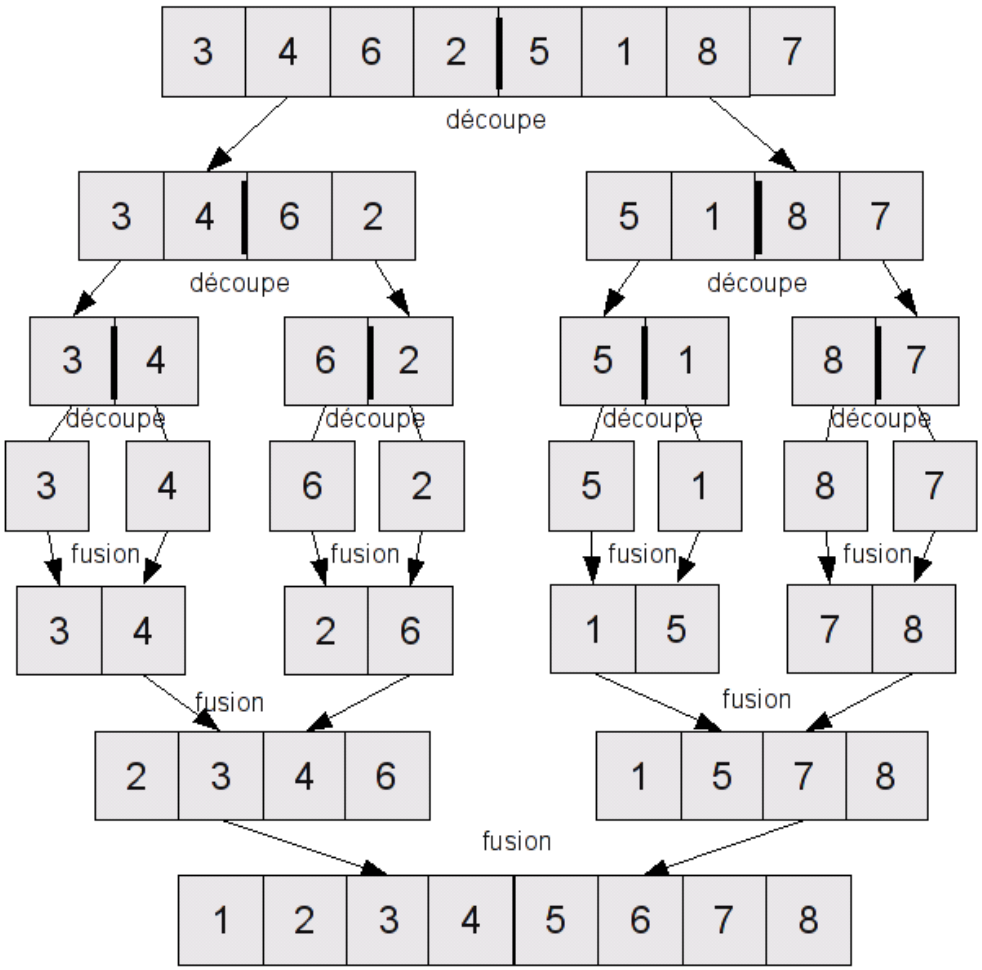
- Pour trier T[4 :6], on trie T[4 :5] et T[5 :6].
- On fusionne T[4 :5] et T[5 :6]
- Pour trier T[6 :8], on trie T[6 :7] et T[7 :8].
- On fusionne T[6 :7] et T[7 :8].
- On fusionne T[4 :6] et T[6 :8].
- On fusionne T[0 :4] et T[4 :8].



Des exemples d'illustration

Exemple 2

- On va illustrer le tri fusion sur le tableau T=[3,4,6,2,5,1,8,7].



Implémentation de l'algorithme de tri fusion

Tri fusion -Solution 1-

- Pour implémenter l'algorithme tri fusion on a besoin de programmer deux fonctions :
  1. Une fonction récursive **Fusion(T1,T2)** permettant de fusionner deux listes triées T1 et T2.
  2. Une fonction récursive **Tri_Fusion(T)** permettant de trier la liste T en uitlisant l'algorithme tri fusion.

Programme récursif **Fusion(T1,T2)**

```
def Fusion(T1,T2) :
    if T1==[] :
        return T2
    if T2==[] :
        return T1
    if T1[0]<T2[0] :
        return [T1[0]]+Fusion(T1[1 :],T2)
    else :
        return [T2[0]]+Fusion(T1,T2[1 :])
```

Implémentation de l'algorithme de tri fusion

Tri fusion -Solution 1-

Programme récursif **Tri_Fusion(T)**

```
def Tri_Fusion(T) :
    if len(T)<=1 :
        return T
    T1=[T[i] for i in range(len(T)//2)]#Pour générer la liste T1=T[0 :len(T)//2]
    T2=[T[i] for i in range(len(T)//2,len(T))]
    return Fusion(Tri_Fusion(T1),Tri_Fusion(T2))
```

⇒**Exemple :**

```
>>>T=[10,39,21,2,8,6,1]
>>>T=Tri_Fusion(T)
>>>print(T)
[1, 2, 6, 8, 10, 21, 39]
```

Implémentation de l'algorithme de tri fusion

Tri fusion -Solution 2-

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

1. Une fonction itérative **Fusion(T1,T2,g,m,d)** qui prend en arguments deux tableaux, T1 et T2, et les trois indices g, m et d. Les portions a1[g..m[et a1[m..d[sont supposées triées. Cette fonction permet d'effectuer la fusion de ces deux portions.
2. La fonction récursive **Tri_Fusion_Rec(g, d)** prend en arguments les indices g et d délimitant la portion à trier. Si le segment contient au plus un élément, c'est-à-dire si $g >= d - 1$, il n'y a rien à faire. Sinon, on partage l'intervalle en deux moitiés égales : on calcule l'élément médian m, puis on trie récursivement a[g..m[et a[m..d[.
3. Une fonction **Tri_Fusion(T)** permettant de trier la liste T en uitlisant l'algorithme tri fusion.

Implémentation de l'algorithme de tri fusion

Tri fusion -Solution 2-

Programme itératif **Fusion(T1,T2,g,m,d)**

```
def Fusion(T1, T2, g, m, d) :
    i, j = g, m
    for k in range(g, d) :
        if i < m and (j == d or T1[i] <= T1[j]) :
            T2[k] = T1[i]
            i = i+1
        else :
            T2[k] = T1[j]
            j = j+1
```

Implémentation de l'algorithme de tri fusion

Tri fusion -Solution 2-

Le code complet : **Tri_Fusion(T)** et **Tri_Fusion_Rec(g,d)**

```
def Tri_Fusion(T) :
    tmp = T[: ]
    def Tri_Fusion_Rec(g, d) :
        if g >= d - 1 :
            return None
            m = (g+d)//2
        Tri_Fusion_Rec(g, m)
        Tri_Fusion_Rec(m, d)
        tmp[g :d] = T[g :d]
        Fusion(tmp, T, g, m, d)
    Tri_Fusion_Rec(0, len(T))
```

Complexité en nombre de comparaisons C(n) -Solution 2-

Si on note $C(N)$ (resp. $f(N)$) le nombre total de comparaisons effectuées par **Tri_Fusion** (resp. **Fusion**) pour trier un tableau de longueur N, on a l'équation de récurrence suivante :

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

En effet, 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 $C(N) = \frac{1}{2} N \ log \ N$, d'où la complexité est $O(N \ log \ N)$
- Dans le pire des cas, tous les éléments sont examinés par **Fusion** et $f(N) = N - 1$, donc $C(N) = N \ log \ N$. D'où la complexité est $O(N \ log \ N)$

Complexité en nombre de affectation A(n) -Solution 2-

Le nombre d'affectations est le même dans tous les cas : N affectations dans la fonction **Fusion** (chaque élément est copié de T1 vers T2) et N affectations effectuées par la copie de T vers tmp.
On note $A(N)$ le nombre total d'affectations pour trier un pour trier un tableau de N éléments.

- Dans le pire des cas l'équation de récurrence suivante :

$$A(N) = 2A(N/2) + 2N$$

Donc, le total d'affectations est $A(N) = 2Nlog(N)$, d'où la complexité est $O(Nlog(N))$

- Le meilleur des cas correspond à la même chose que le pire des cas.

La complexité de tri fusion : démonstration mathématique

⇒**Problème :**

Soit $C(n)$ le nombre de comparaisons pour l'algorithme tri fusion. On suppose que la taille du tableau initial est une puissance de 2 : $n = 2^i$. Montrer que $C(n) <= Knlog(n)$

⇒**Démontration :**

- La fusion demande à chaque appel au plus $n/2$ comparaisons.
- $C(n) <= 2xC(\frac{n}{2}) + \frac{n}{2}$

$$C(n) <= 2x[2C(\frac{n}{2^2}) + \frac{n}{2^2}] + \frac{n}{2}$$

$$C(n) <= 2^2xC(\frac{n}{2^2}) + 2\frac{n}{2}$$

$$C(n) <= 2^3xC(\frac{n}{2^3}) + 3\frac{n}{2}$$

$$.....$$

$$C(n) <= 2^ixC(\frac{n}{2^i}) + i\frac{n}{2}$$

$$C(n) <= kn + \frac{1}{2}nlog_2(n) <= Knlog(n)$$

Comparaison de complexité de différentes méthodes de tris

⇒Complexité en termes de nombre de comparaisons

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	$\frac{1}{2}\ N\ log(N)$	$N\ log(N)$	$N\ log(N)$

⇒Complexité en termes de nombre d'affectations

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	$2N$	$2N\ log(N)$	N^2
Fusion	$2\ N\ log(N)$	$2\ N\ log(N)$	$2\ N\ log(N)$