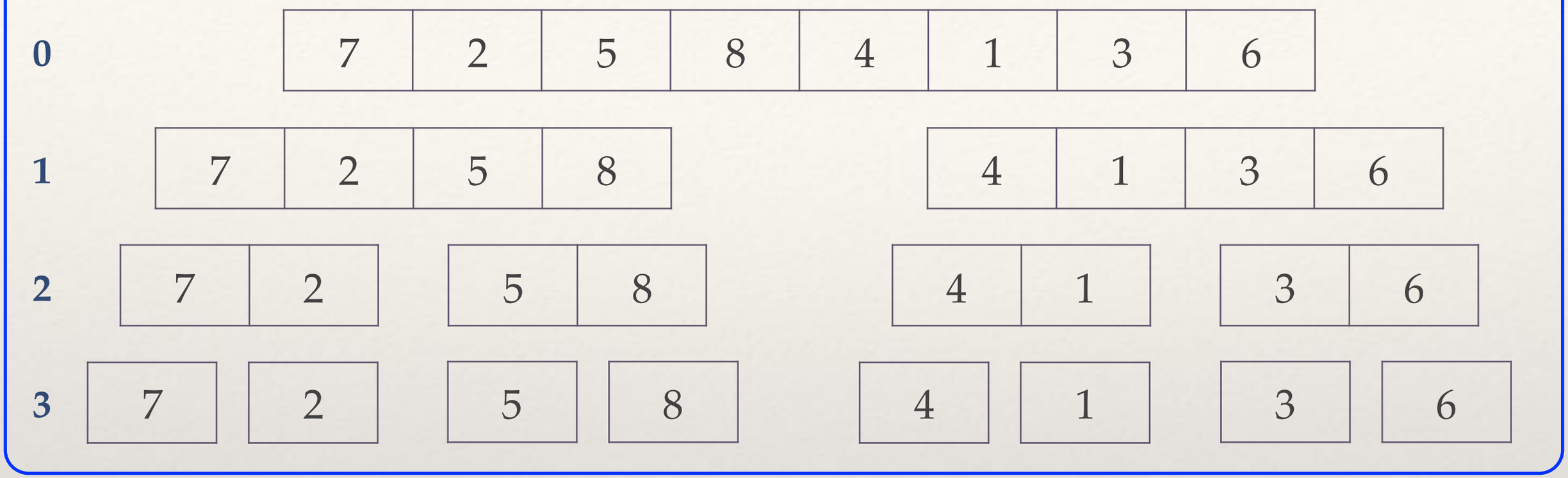
INFORMATIQUE

Tri avancé : MergeSort

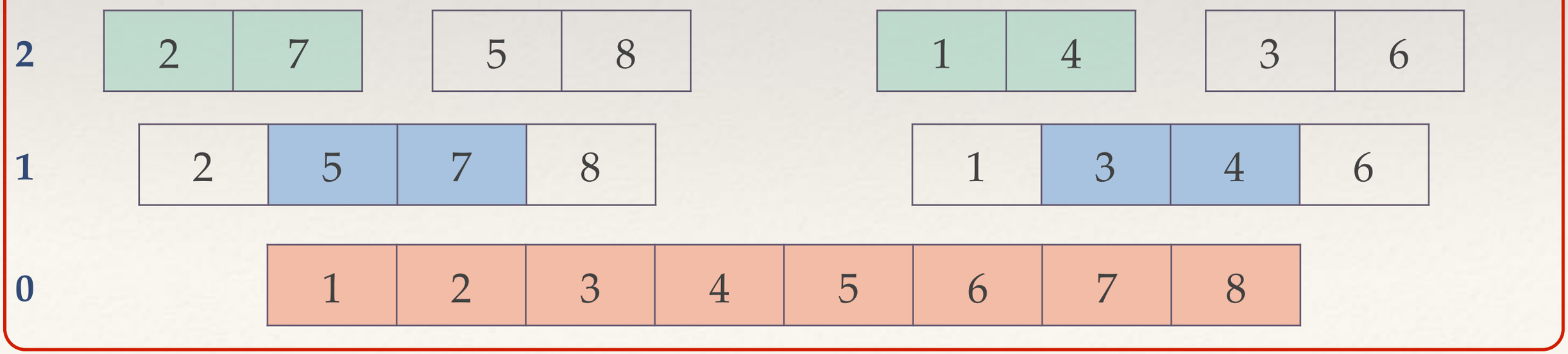
3 -Le Tri Fusion

« merge-sort »

1 - Descente recursive

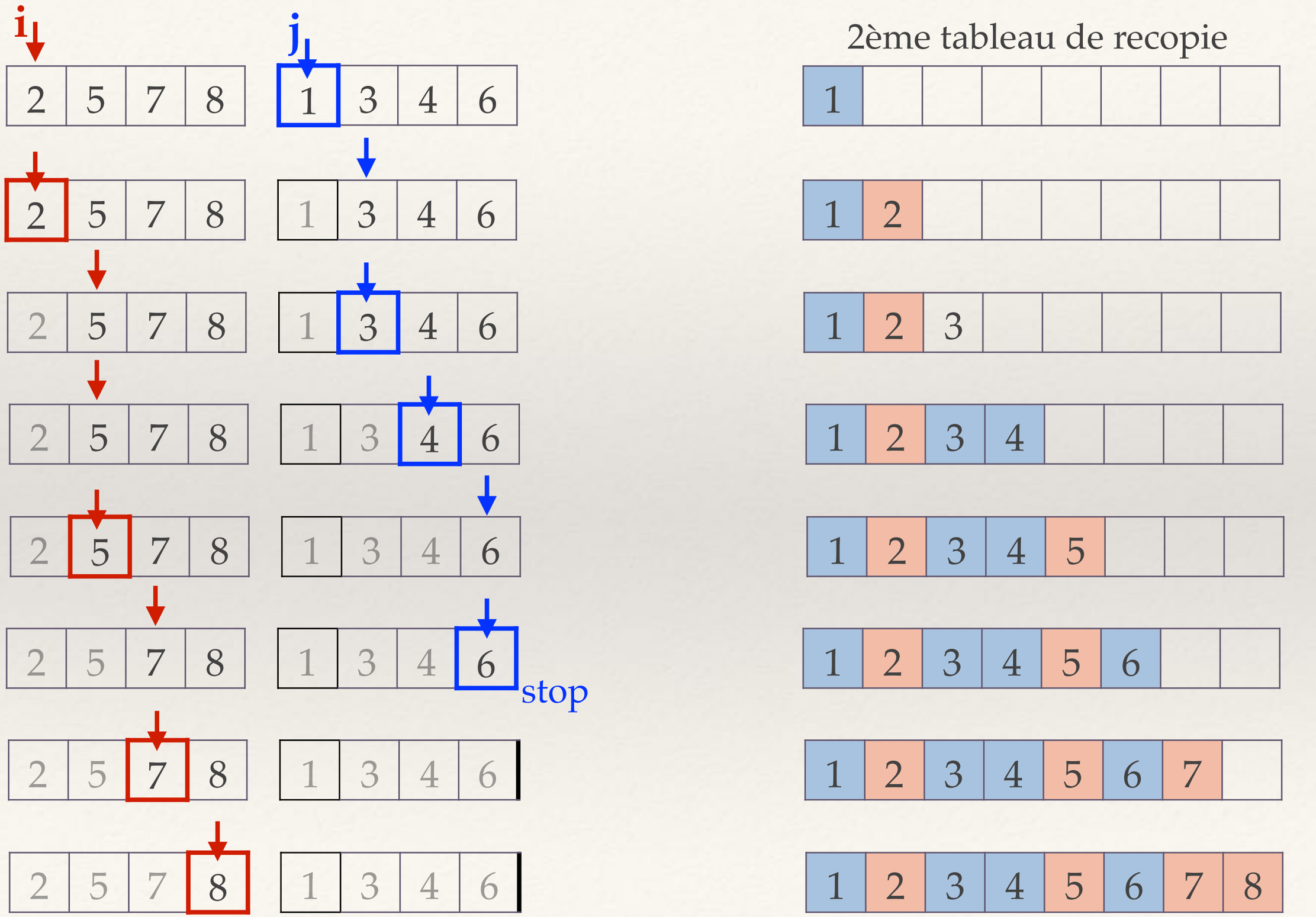


2 - Remontée de Pile : opération de Fusion



Opération de fusion

On réalise un balayage de gauche à droite, à l'aide de 2 curseurs d'indices i et j :



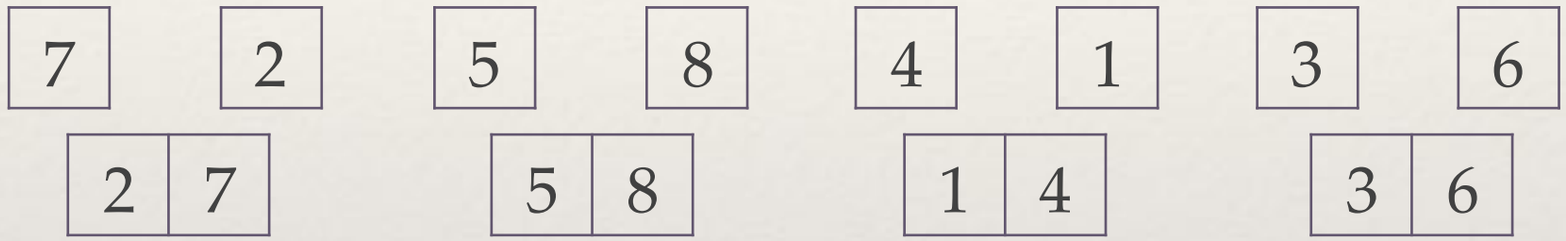
A chaque étape :

Boucle N-fois

- On compare les valeurs en i et en j.
- On sélectionne le plus petit curseur encore disponible
  => On place la valeur associée dans la copie du tableau.
- On avance le curseur sélectionné d'un cran vers la droite.

Notez qu'à chaque niveau de récursion, le coût temporel est le même !

Exemple :



On fusionne **N/2 fois 2 tableaux**, puis N/4 fois 4 **tableaux**, etc => **soit N comparaisons**

Cette opération représente un **Coût linéaire du point de vue temporel**, mais son point faible est la nécessité d'un second tableau pour recopier les données triées :

En effet du point de vue spatial les premières recopies ne posent pas de problème car elles sont faites une à une [Coût de taille 2, puis 4 puis 8 etc]. Le problème vient de la dernière étape de fusion. **Soit un coût spatiale de 2N** car il faut tout recopier d'un coup.

Rq : Un algorithme IN-PLACE sans le second tableau existe mais est peu utilisé en raison de sa difficulté.

Codage

```
def triFusion(tab):

    N=len(tab);      milieu=N//2;

    if (N<2):                    #Gestion de la terminaison
        return tab

    left = triFusion(tab[:milieu]);          right = triFusion(tab[milieu:])

    i = j = 0;    R = []          #on reconstruit le tableau=> Not IN PLACE
    while( len(R) < N ):
        if ( i == len(left)):
            R+=right[j:];      break      #Curseur de gauche bloqué
        if ( j == len(right)):
            R += left[i:];      break      #Curseur de droite bloqué

        if ( right[j] < left[i] ):              #cas standard : on prend le plus petit
            R += [right[j]];      j+=1      #et on avance le curseur.
        else:
            R += [left[i]];         i+=1

return R
```

Complexité spatiale et temporelle

L'ensemble des opérations à réaliser est bien défini :

1 - il y aura h = Log(N) étages de récursion.

2 - pour chaque étage le coût de fusion (à la remontée) vaut N.

Soit une complexité temporelle : $\mathbb{C} = O(N.\log(N))$ **dans tous les cas !**

Rq : Si la taille n'est pas une puissance de 2 on en aura h = int(log2(N)) + 1
et on peut raisonner sur un tableau de taille 2**h complété par des valeurs infinies.

Terminaison :

1 - Soit Tn la taille des sous tableaux passés dans les appels récursifs successifs. Tn est une suite décroissante strictement bornée par 0 [Tn+1 ⩽ Tn //2 + 1 < Tn]

La condition de terminaison : Tn < 2 garantit que la pile d'appels récursif a une hauteur finie.

2 - A chaque étage de fusion : le balayage des éléments est borné par N.

Donc **l'algorithme termine**.

Preuve : Soient deux tableaux de taille N1 et N2 triés de tailles positives ou nulles qui sont le résultats de deux appels à tri fusion : Il faut montrer que la fusion de ces deux tableaux supposés triés donne un tableau de taille N1+N2 trié pour tout N1 et N2.

Il y a deux cas au niveaux des feuilles de l'arbre.

N1=0
N2=1

On ne retourne qu'un tableau de taille N2=1 donc trié !

N1=1
N2=1

soient [a] et [b] les tableaux : la fusion renvoie [a,b] ou [b,a] trié !

N1>1
N2>1

On remplit un nouveau tableau vide de taille N1+N2 en choisissant toujours le plus petit entre : le plus petit restant dans L1 et le plus petit restant dans L2

Le nouvel élément introduit est nécessairement :

- **égal ou plus grand que tous ceux déjà introduits.**
- **égal ou plus petit que tous ceux restant à introduire.**

Ainsi par construction de la fusion et puisque les deux tableaux utilisés sont triés (par hypothèse de récurrence) on obtient un tableau de taille N1+N2 trié !

Par récurrence, le résultat de triFusion est toujours un tableau trié !

Stabilité

Le tri fusion est stable : on peut voir et même coder la fusion comme une insertion du tableau de droite dans le tableau de gauche.
Il sera utilisé bien volontiers à chaque fois qu'il y a des tris successifs selon différents critères. Dans ce contexte on peut s'attendre à ce que le tableau soit partiellement trié.

Optimisation

Une optimisation très simple à mettre en oeuvre consiste simplement avant la fusion à vérifier si la plus grande valeur du tableau de gauche est plus petite que la plus petite du tableau de droite : auquel cas il suffit de concaténer les deux tableaux.

Un tableau déja trié sera trié dans un temps linéaire.

Master - Theorem

On considère un problème de taille N que l'on subdivise selon le paradigme diviser pour régner en **a** sous problèmes de taille N/**b**. On résout finalement le problème en assemblant les solutions des sous problèmes pour un coût en O(N^**d**) :

Soit le coût Total à l'étage n : $$T(n) = a\,T(\tfrac{n}{b}) + O(n^d)$$

Le théorème énonce le coût algorithmique total selon trois cas :

1 - Si $a > b^d$ => $T(n) = O(n^{Log_b(a)})$

2 - Si $a = b^d$ => $T(n) = O(n^d\ Log_b(n))$ (Admis)

3 - Si $a < b^d$ => $T(n) = O(n^d)$

Exemples d'applications du théorème

Tri Fusion : a = b = 2 et d = 1

$$T(n) = O(n\ Log(n))$$

Cas 2

Recherche Dichotomique : a = 1, b = 2 et d = 0

$$T(n) = O(Log(n))$$

Min/Max dans un arbre binaire : a = 1, b = 2 et d = 0

Cas 2

Parcours arbre binaire : a = 2, b = 2 et d = 0

$$T(n) = O(n)$$

Cas 1