

seance_12nov

November 12, 2025

```
[7]: def fusion(t1: list, t2: list) -> list:
      """Fusionne deux listes triées en une seule liste"""
      i, j = 0, 0
      resultat = []
      while i < len(t1) and j < len(t2):
          if t1[i] <= t2[j]:
              resultat.append(t1[i])
              i += 1
          else:
              resultat.append(t2[j])
              j += 1
      # Ajouter les éléments restants
      resultat.extend(t1[i:])
      resultat.extend(t2[j:])
      return resultat
```

```
[9]: l1=[10,15,20]
      l2=[1,2,3,4]
      fusion(l1,l2)
```

```
[9]: [1, 2, 3, 4, 10, 15, 20]
```

Terminaison

A chaque étape dans la boucle interne la quantité $i + j$ augmente de une unité.

Donc la grandeur $|t_1| + |t_2| - (i + j)$ est entière strictement décroissante et positive.

On a identifié un *variant* de boucle : la fonction termine (au bout d'un moment on sort de la boucle).

Complexité

Il y a au plus $|t_1| + |t_2|$ passages dans la boucle. Chaque passage est en $O(1)$. Donc complexité totale de la boucle en $O(|t_1| + |t_2|)$

Les lignes 13 et 14 (**extend**) ont pour complexité le nombre d'élément ajoutés. Cette phase est $O(|t_1| + |t_2|)$.

Complexité totale de la fonction en $O(|t_1| + |t_2|)$

```
[3]: def tri_fusion(t: list) -> list:
      """Trie une liste : méthode du tri fusion."""
      if len(t) <= 1:
          return t
      milieu = len(t) // 2
      gauche = tri_fusion(t[:milieu])
      droite = tri_fusion(t[milieu:])
      return fusion(gauche, droite)
```

```
[4]: tri_fusion([10,1,8,98,35,10,0,56])
```

```
[4]: [0, 1, 8, 10, 10, 35, 56, 98]
```

Terminaison

Le cas d'arrêt (si la lg est ≤ 1) termine car on retourne la liste.

Les appels récursifs à `tri_fusion` se font avec des listes strictement plus courtes que la liste initiale (laquelle est non vide, sinon on est dans le cas d'arrêt). On a donc identifié une quantité entière positive strictement décroissante (il s'agit d'un *variant de récursion*).

La fonction termine pour tout appel.

On note $c(n)$ la partie entière par excès du nombre n : $\lceil n \rceil$

On note $f(n)$ la partie entière par défaut du nombre n : $\lfloor n \rfloor$

Si la liste à trier est de taille n . Alors la complexité $C(n)$ de l'appel `tri_fusion(t)` respecte la récurrence

$$C(n) = C(c(\frac{n}{2})) + C(f(\frac{n}{2})) + O(n)$$

Où $O(n)$ désigne le coût de la découpe en deux moitiés et de la fusion des moitiés triées.

On simplifie la relation en

$$C(n) = C(c(\frac{n}{2})) + C(f(\frac{n}{2})) + n$$

Supposons $n = 2^k$. Alors $c(\frac{n}{2}) = 2^{k-1} = f(\frac{n}{2})$

La relation devient

$$C(2^k) = C(2^{k-1}) + C(2^{k-1}) + 2^k = 2C(2^{k-1}) + 2^k$$

Donc

$$C(2^k) = 2(2C(2^{k-2}) + 2^{k-1}) + 2^k = 2^2C(2^{k-2}) + 2 \times 2^k$$

$$C(2^k) = 2^2(2C(2^{k-3}) + 2^{k-2}) + 2^k = 2^3C(2^{k-3}) + 3 \times 2^k$$

Par récurrence on obtient

$$C(2^k) = 2^k C(2^{k-k}) + k \times 2^k = 2^k C(1) + k 2^k$$

avec $C(1)$ en $O(1)$

Si n est une puissance de deux la complexité est majorée et aussi minorée par un multiple de $n \log_2 n$

Si n n'est pas une puissance de 2. On encadre n entre deux puissances de 2 consécutives.

$$2^{\lfloor \log_2 n \rfloor} \leq n = 2^{\log_2(n)} < 2^{\lfloor \log_2 n \rfloor + 1}$$

La relation précédente s'applique aux deux bornes. On a une minoration par une constante fois $\lfloor \log_2 n \rfloor 2^{\lfloor \log_2 n \rfloor}$ et une majoration par une constante fois $(\lfloor \log_2 n \rfloor + 1) 2^{\lfloor \log_2 n \rfloor + 1}$

Or, ces deux quantités sont équivalentes à $n \log_2 n$ et on en déduit que la complexité du tri fusion pour n'importe quelle liste est *pseudo-linéaire*.

Rq : (puisque la complexité est croissante en la tailles)

```
[ ]: def recherche_naive(t,x):  
    for i in range(len(t)):  
        if t[i] == x:  
            return i  
    return -1
```

Complexité.

Si x est le premier élément de t alors la complexité est un $O(1)$ (meilleur cas).

Le pire cas se produit si $x \notin t$: il faut alors parcourir toute la liste et la complexité est linéaire en la longueur de la liste. $O(n)$ si n est la longueur de la liste.

```
[ ]:
```