

Diviser pour Régner

Table des matières

1 Méthode	1
2 Tris fusion	1
2.1 Présentation	1
2.2 Implémentation en Python	2
2.3 Complexité	2
3 Calcul du minimum d'une liste	4

1) Méthode

Le diviser pour régner est une méthode algorithmique basée sur le principe suivant. On prend un problème (généralement complexe à résoudre), on divise ce problème en une multitude de petits problèmes, l'idée étant que les "petits problèmes" seront plus simples à résoudre que le problème original. Une fois les petits problèmes résolus, on recombine les "petits problèmes résolus" afin d'obtenir la solution du problème de départ.

Le paradigme "diviser pour régner" repose donc sur 3 étapes :

- DIVISER : le problème d'origine est divisé en un certain nombre de sous-problèmes
- RÉGNER : on résout les sous-problèmes (les sous-problèmes sont plus faciles à résoudre que le problème d'origine)
- COMBINER : les solutions des sous-problèmes sont combinées afin d'obtenir la solution du problème d'origine.

Les algorithmes basés sur le paradigme "diviser pour régner" sont très souvent des algorithmes récursifs.

2) Tris fusion

2.1) Présentation

Le tri fusion est un tri récursif basé sur le principe "diviser pour régner". Cet algorithme repose donc sur une fonction auxiliaire, la fonction de fusion, qui prendra en argument deux listes triées et renverra la liste fusionnée.

Voici le principe de la fonction auxiliaire de fusion ordonnée de deux listes L_1 et L_2 de longueurs respectives k et l dans une liste M .

- On initialise i et j à 0. On initialise M à $[]$.
- Tant que $i < k$ et $j < l$, on compare $L_1[i]$ à $L_2[j]$.
 - Si $L_1[i] < L_2[j]$ alors
 - On ajoute $L_1[i]$ à la fin de M .
 - On incrémente i
 - Sinon
 - On ajoute $L_2[j]$ à la fin de M
 - On incrémente j
- On termine la liste M de la façon suivante :

- Si $i = k$ et $j < l$, on ajoute les éléments $L_2[j], \dots, L_2[l - 1]$ dans cet ordre à la fin de la liste M .
- Si $i < k$ et $j = l$, on ajoute les éléments $L_1[i], \dots, L_1[k - 1]$ dans cet ordre à la fin de la liste M .

Exercice 1 :

Suivons l'algorithme précédent pour réaliser à la main la fusion ordonnée des listes $L_1 = [2, 5, 12]$ et $L_2 = [1, 6, 11, 15]$ dans une liste M . Recopiez et complétez le tableau suivant.

i	j	M	$L_1[i]$	$L_2[j]$

Exemple :

Voici dans la FIGURE 1 une représentation de l'algorithme de tri fusion appliqué à la liste $A = [23, 12, 4, 56, 35, 32, 42, 57, 3]$. Étudiez attentivement le schéma ci-dessous afin de mieux comprendre le principe du tri-fusion (identifiez bien les phases "DIVISER" et "COMBINER").

Exercice 2 :

Faire à la main le tri fusion de la liste $[3, 4, 19, 7, 1, 12]$

2.2) Implémentation en Python

On définit d'abord une fonction permettant de fusionner deux listes triées au préalable en une seule liste triée.

Exercice 3 :

Écrire la fonction fusion. On utilisera le [découpage](#) (ou slicing) pour la deuxième partie de l'algorithme.

```
def triFusion(L):
    """tri fusion d'une liste en récursif"""
    if len(L)<2:
        return L
    return fusion(triFusion(L[:len(L)//2]),triFusion(L[len(L)//2:])))
```

Cette fonction renvoie une copie triée de la liste passée en argument, sans la modifier.

Exercice 4 :

Écrire cet algorithme en python sur Spyder (ou sur Thonny) et trier la liste $L=[23,12,4,56,35,57,3,11,6]$

2.3) Complexité

Nous avons vu que le tri par insertion et tri par sélection ont tous les deux une complexité $O(n^2)$. Qu'en est-il pour le tri-fusion ?

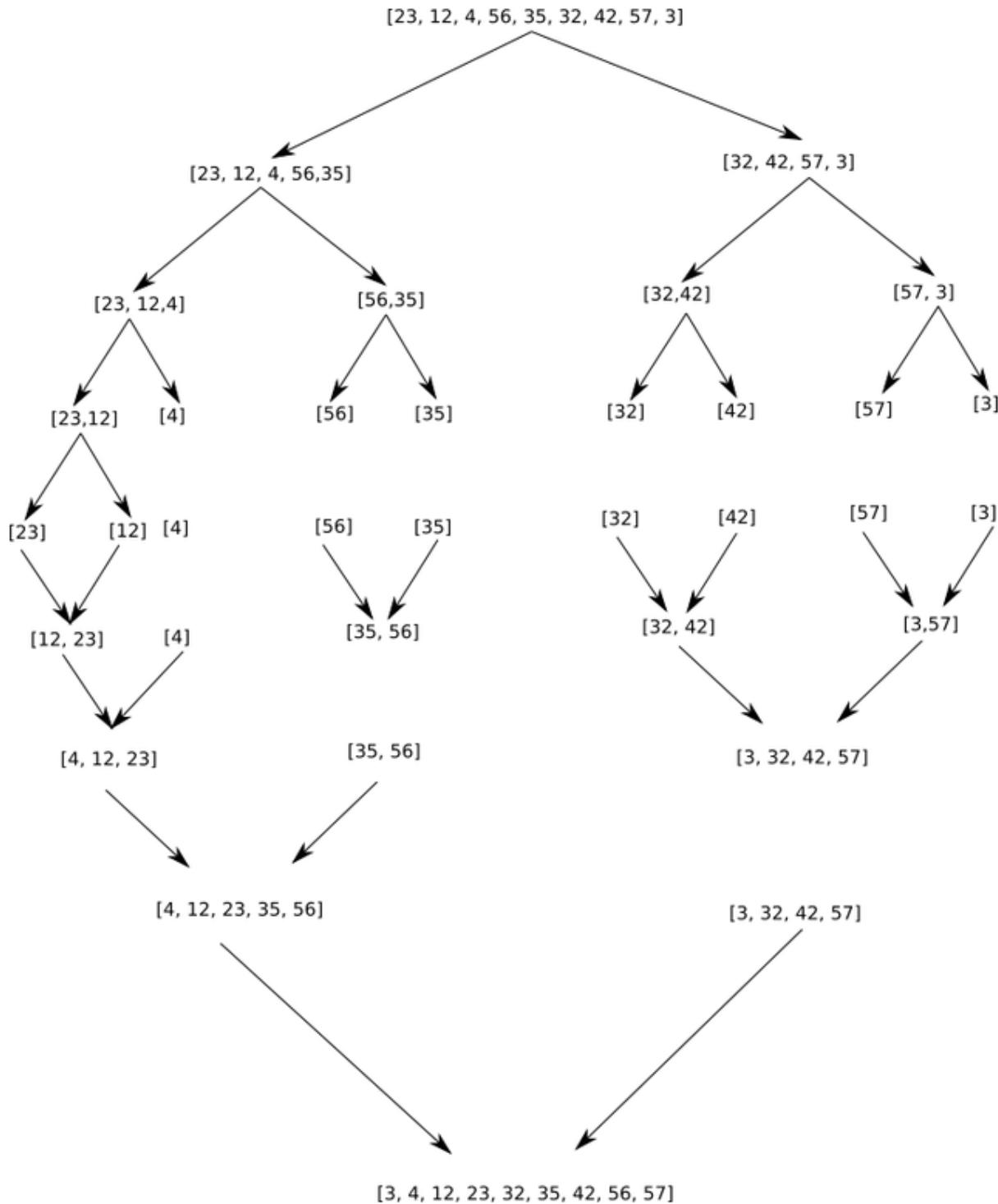
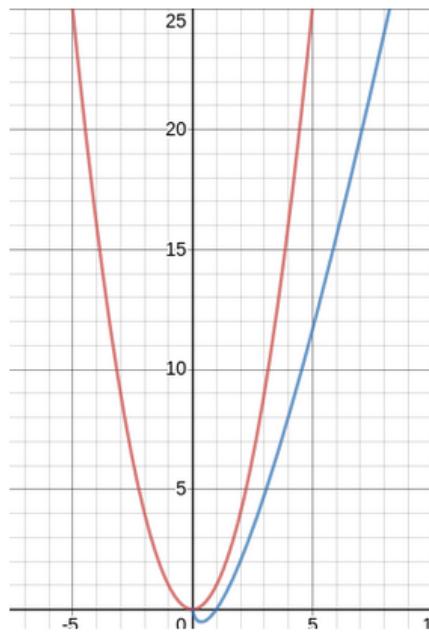


FIGURE 1 – Exemple de représentation de l'algorithme de tri fusion

Le calcul rigoureux de la complexité de cet algorithme sort du cadre de ce cours. Mais, en remarquant que la première phase (DIVISER) consiste à "couper" les tableaux en deux plusieurs fois de suite, intuitivement, on peut dire qu'un logarithme base 2 doit intervenir. La deuxième phase consiste à faire des comparaisons entre les premiers éléments de chaque tableau à fusionner, on peut donc supposer que pour un tableau de n éléments, on aura n comparaisons. En combinant ces 2 constatations on peut donc dire que la complexité du tri-fusion est en $O(n \cdot \log(n))$ (encore une fois la "démonstration" proposée ici n'a rien de rigoureux).

La comparaison des courbes de la fonction n^2 (en rouge) et $n \cdot \log(n)$ (en bleu) :



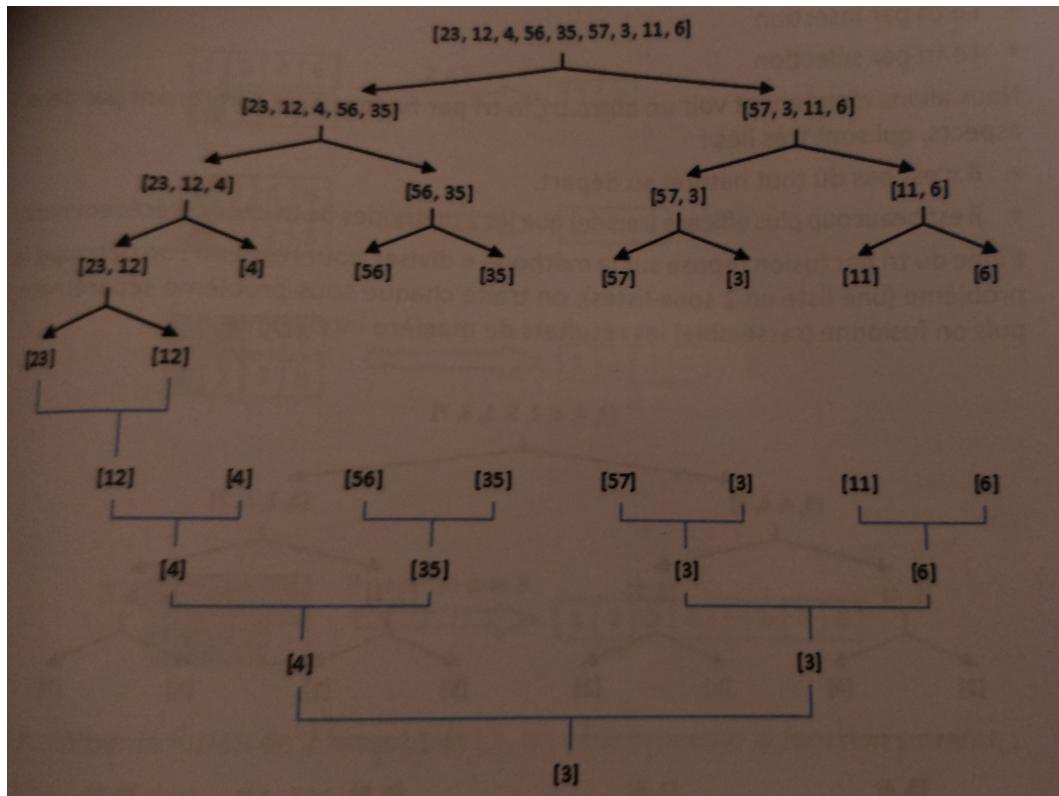
Nous montrons que l'algorithme de tri-fusion est plus "efficace" que l'algorithme de tri par insertion ou que l'algorithme de tri par sélection.

3) Calcul du minimum d'une liste

Voici donc les trois étapes de la résolution de ce problème via la méthode "diviser pour régner" :

- diviser la liste en deux sous-listes en la "coupant" en deux.
- calculer récursivement le minimum de chacune de ces sous-listes. Arrêter la récursion lorsque les listes n'ont plus qu'un seul élément.
- retourner le plus petit des deux minimums précédents.

Prenons par exemple la liste suivante : $L=[23,12,4,56,35,57,3,11,6]$. On peut représenter les étapes de la méthode "diviser pour régner" par l'arbre suivant :



On peut donner la fonction suivante qui calcule par la méthode "diviser pour régner" le minimum d'une liste. On lui donne également en paramètres les indices du premier et du dernier élément.

Exercice 5 : Écrire cet algorithme.