CHAPITRE 5 : DIVISER POUR REGNER ET TRI FUSION

I. « DIVISER POUR REGNER »

1. Principe

Le paradigme de programmation « diviser pour régner » consiste à ramener la résolution d'un problème dépendant d'un entier à la résolution d'un ou plusieurs sous-problèmes dont la taille des entrées passe de à ou une fraction de . Les algorithmes ainsi conçus s'écrivent de manière naturelle de façon récursive.

Cette méthode se décompose en trois phases :

- Diviser : on divise les données initiales en plusieurs sous-parties
- Régner : on résout récursivement chacun des sous-problèmes associés (ou on les résout directement si leur taille est assez petite)
- Combiner : on combine les différents résultats obtenus pour obtenir une solution au problème initial.

2. Exemple

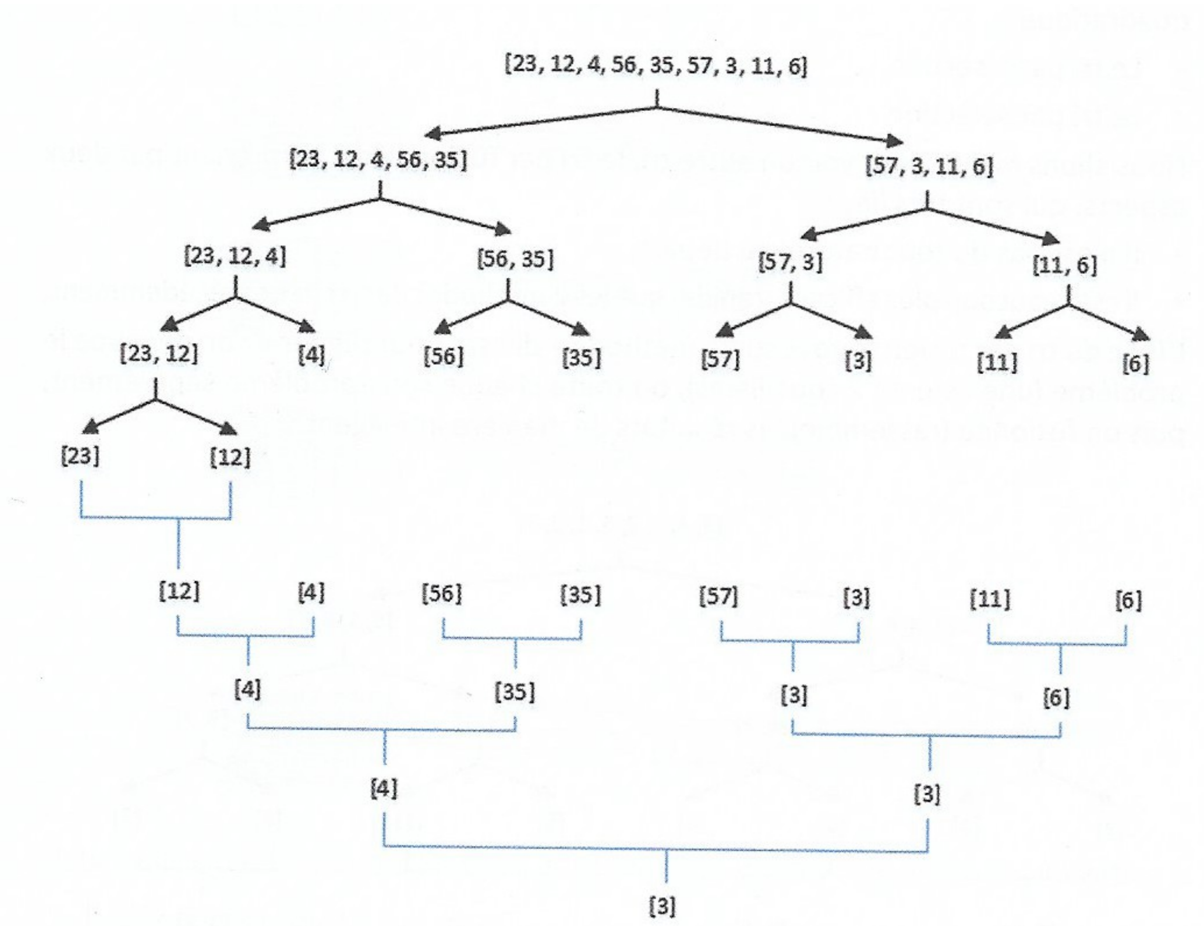
On souhaite déterminer le minimum d'une liste. On va donc découper la liste en deux sous-listes et calculer récursivement le minimum de chaque sous-liste, puis les comparer. Le plus petit des deux sera le minimum de la liste totale.

La condition d'arrêt de la récursivité est d'obtenir une liste à un seul élément, dont le minimum est cet élément.

Les trois étapes sont donc :

- Diviser la liste en deux sous-listes en la « coupant » en deux
- Calculer récursivement le minimum de chaque sous-liste. On arrête la récursion lorsque les listes n'ont plus qu'un seul élément.
- Retourner le plus petit des deux minimums de chacune des sous-listes.

Si on prend pour exemple la liste , on peut représenter les étapes par l'arbre suivant :



On peut utiliser la fonction dont l'algorithme est donné ci-dessous pour déterminer le minimum d'une liste avec la méthode du « diviser pour régner » :

```
Fonction Minimum(L, d, f) :
  Si d == f :
        Retourner L[d]
  Sinon :
        m = (d+f) // 2
        x = Minimum(L, d, m)
        y = Minimum(L,m+1, f)
        Si x < y
           Retourner x
        Sinon
           Retourner y
```

Expliquer le rôle des différentes variables utilisées, et le fonctionnement global de cet algorithme, puis le traduire en langage Python et le tester avec la liste précédente.

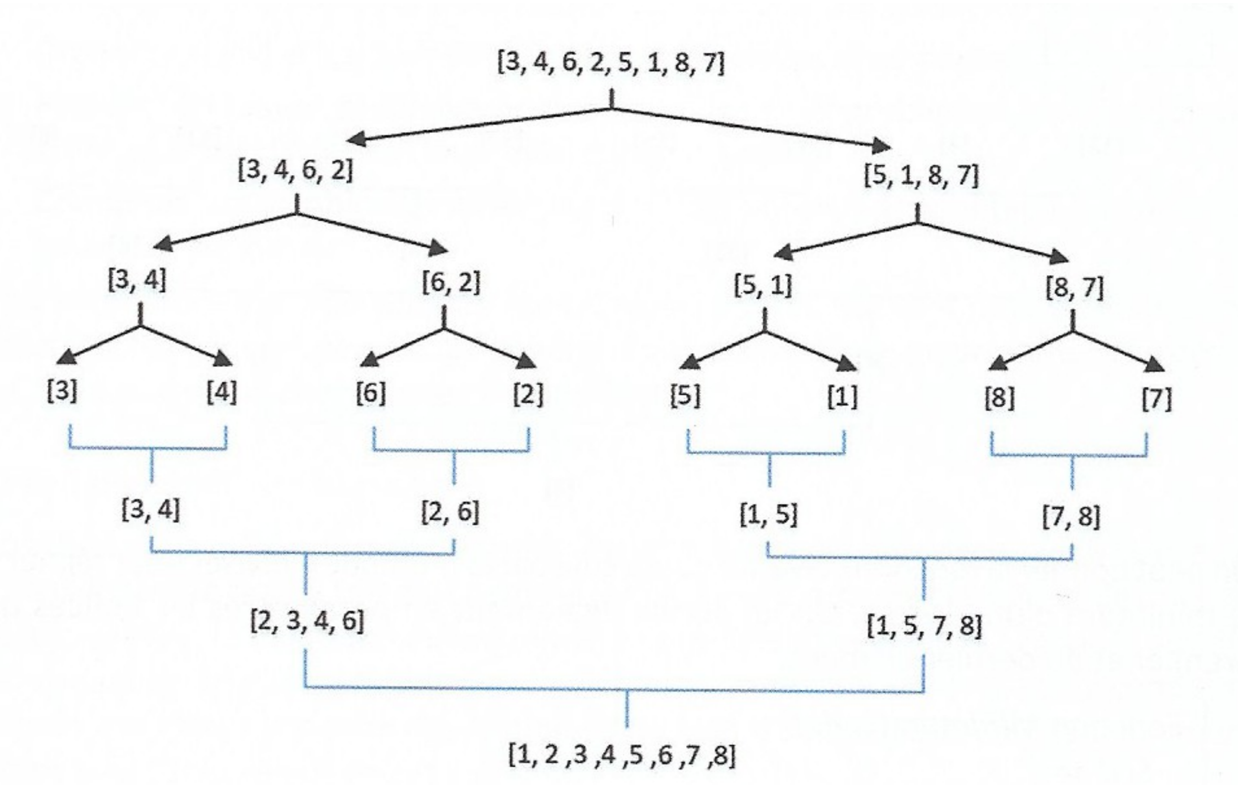
Exercices d'application

II. TRI FUSION

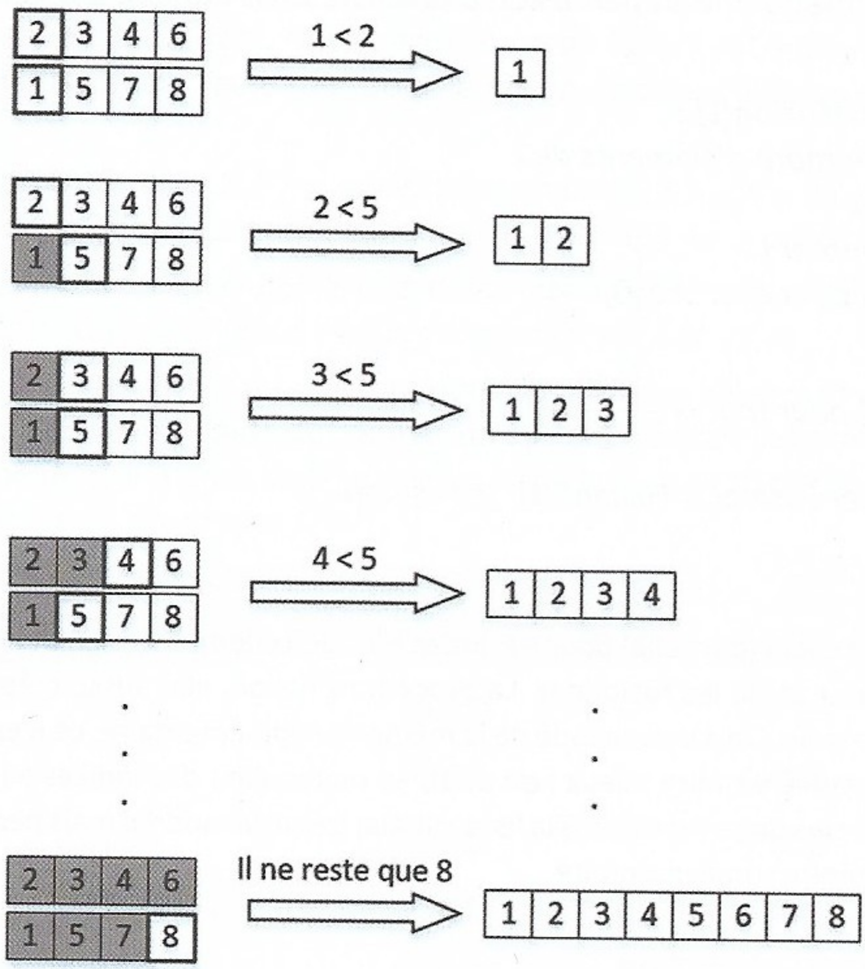
On a déjà vu en première deux méthodes de tri, dont le coût est quadratique : le tri par insertion et le tri par sélection.

Une autre méthode de tri, beaucoup plus efficace, est le tri par fusion.

L'idée du tri fusion repose sur la méthode « diviser pour régner » : on découpe la liste à trier en deux sous-listes, on traite chaque problème séparément, puis on rassemble (on fusionne) les résultats. Ce principe est illustré ci-dessous avec la liste :



La fusion (partie en bleu) est efficace, car les deux listes obtenues sont triées. Il suffit donc de les parcourir dans l'ordre : le plus petit élément de la liste triée est le plus petit des deux premiers éléments des sous-listes. On l'ajoute donc à la liste triée, et on le retire de la sous-liste correspondante, et on recommence (récursion) :



La fusion de deux listes et peut s’effectuer à l’aide de l’algorithme ci-dessous :

```
Fonction Fusion(L1, L2) :
   Si L1 est vide :
        Retourner L2
   Si L2 est vide :
        Retourner L1
   Si L1[0] < L2[0] :
        Retourner [L1[0] + Fusion(L1[1 :], L2)]
   Sinon :
        Retourner [L2[0] + Fusion(L1, L2[1:])]
```

Expliquer le fonctionnement de cet algorithme et l’implémenter en Python.

La fonction permettant le tri par fusion d'une liste est donnée par l'algorithme suivant :

Fonction ***TriFusion(L)*** *:*
nb = le nombre d'éléments de L
Si nb <=1 :
Retourner L
L1 = L[x] pour tout $x \in \left[0, \frac{nb}{2}\right[$
L2 = L[x] pour tout $x \in \left[\frac{nb}{2}, nb\right[$
Retourner Fusion(TriFusion(L1), TriFusion(L2))

Expliquer le fonctionnement de cet algorithme, l'implémenter en Python, puis le tester avec la liste précédente.

Complexité

On part d'une liste de éléments (on va supposer que est une puissance de 2). On le coupe en deux, ce qui donne deux listes de éléments, puis 4 listes de éléments …

Le découpage s'arrête lorsqu'on a des listes de taille 1.

On appelle le nombre de découpes nécessaires pour un tableau de taille .

On a et (si on double la taille du tableau, il faut une découpe de plus).

On reconnait la fonction logarithme de base 2, donc la phase de découpage nécessite opérations.

A chaque étape de fusion, on parcourt les deux demi-listes une seule fois, donc le coût est linéaire .

Il y a donc étapes à opérations chacune, ce qui fait opérations. C'est beaucoup moins que pour les tris vus en première qui sont en .

Voici, à titre d'exemple, les temps d'exécution sur une même machine pour un tri par sélection, et pour un tri par fusion :

Nombre d'éléments dans la liste (n)	Tri par sélection	Tri par fusion
100	0.006s	0.006s
1 000	0.069s	0.010s
10 000	2.162s	0.165s
20 000	7.526s	0.326s
40 000	28.682s	0.541s

<u>Exercices d'application</u>

TP : suite de Fibonacci, programmation dynamique et mémoïsation.