Lycée Vaugelas MPSI 2021 - 2022
TD d'informatique n° 13 du 13-14/12/2021 **Tri fusion**

• Le **tri fusion** ou **tri dichotomique** (en anglais *merge sort* ou *mergesort*) est, avec la recherche dichotomique, l'un des exemples le plus illustratifs du paradigme *diviser pour régner*. On attribue sa découverte à John von Neumann en 1945. L'algorithme de tri s'énonce de manière inductive (récursivité descendante ou top-down) :

— partager les éléments à trier en deux moitiés égales,
— trier les deux moitiés (ou ne rien faire si leur taille est inférieure à 2)
— fusionner les deux moitiés triées.

Constatons qu'**il est possible de fusionner deux tableaux triés en temps linéaire** et que **le nombre d'appels récursifs est logarithmique** en la taille du tableau pour comprendre pourquoi cet algorithme est aussi efficace. On obtient une complexité temporelle en $\Theta(n \ln n)$ où n est la taille du tableau à trier, **complexité optimale pour un tri par comparaison.** Ces comparaisons ne sont réalisées que dans la phase de fusion.

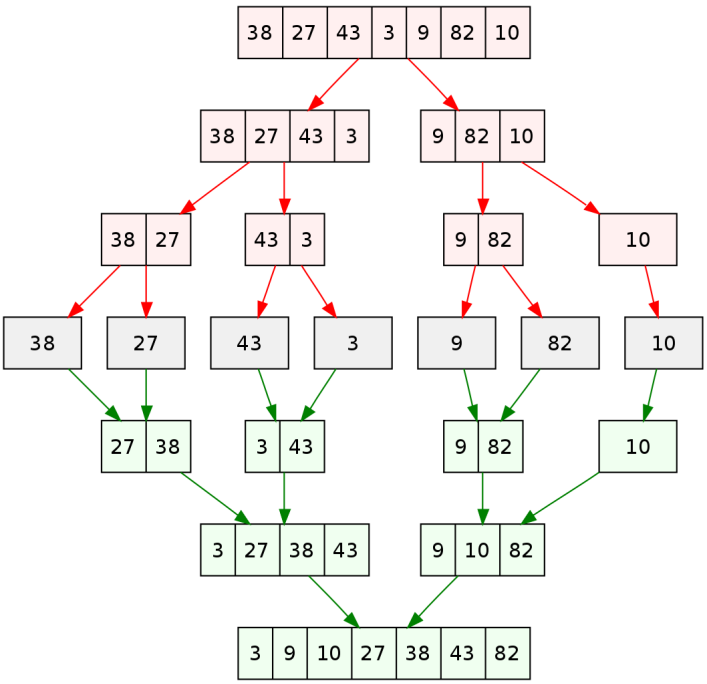
• Une présentation impérative (bottom-up) de l'algorithme est tout aussi simple. Elle consiste à détailler la remontée de l'algorithme récursif :

— décomposer le tableau de taille n en n tableaux de taille 1
— fusionner répétitivement deux tableaux adjacents jusqu'à ce qu'il n'en reste qu'un.

• Le tri fusion possède la qualité de pouvoir être réalisé **en place**, on utilise néanmoins un tableau auxiliaire pour simplifier l'implémentation.

• Le tri fusion possède la qualité d'être **facilement parallélisable** : comme souvent avec la stratégie diviser pour régner, on peut confier le traitement des sous-problèmes à des unités d'exécution distinctes.

• Le tri fusion est particulièrement **efficace pour trier des listes chaînées**, pour lesquelles on ne dispose pas d'accès aléatoire (ce qui signifie que l'on ne peut pas accéder en temps constant à un élément quelconque, contrairement aux tableaux). C'est ainsi que procède le noyau Linux pour trier des listes chaînées. Python, Java et le système d'exploitation mobile Android de Google utilisent un mélange de tri fusion et de tri par insertion, comme l'algorithme *timsort*. L'idée est de déléguer à l'algorithme de tri par insertion le tri des tableaux de petite taille (inférieure à 8 par exemple). Image de https://en.wikipedia.org/wiki/Merge_sort :



Exercice 1 (Tri fusion d'un tableau)

La portion de tableau à trier est délimitée par deux indices g et d. Le partage en deux parties est réalisé avec l'indice $m = \left\lfloor \frac{g+d}{2} \right\rfloor$. Une fonction Python de tri fusion se présente sous la forme suivante, elle trie le tableau `t` en place :

```
def tri_fusion(t):
    tmp = [None] * len(t)
    def fusion(g, m, d):
        tmp[g : m + 1] = t[g : m + 1]
        ... suite à écrire à la question 3
    def tri(g, d):
        if g < d:
            m = (g + d) // 2
            tri(g, m) ; tri(m + 1, d)
            fusion(g, m, d)
    tri(0, len(t) - 1)
```

1. Expliquer le principe et la structure de la fonction `tri_fusion`, en détaillant le rôle de chaque ligne. La récursivité est-elle terminale ?
2. Écrire l'algorithme de fusion en pseudo-code.
3. Compléter la fonction `fusion` afin d'assurer le bon fonctionnement du tri fusion. Les valeurs triées sont écrites dans le tableau `t`.
4. Le tri fusion est-il stable ?
5. Montrer la correction de votre fonction `fusion` à l'aide d'un invariant de boucle.
6. Dans le cas où $n = 2^k$ avec $k \in \mathbb{N}$, résoudre la récurrence

$$T(n) = 2\,T\left(\frac{n}{2}\right) + cn \tag{1}$$

   où c est une constante réelle. En déduire l'ordre de grandeur asymptotique de $T(n)$ pour n quelconque, en encadrant n entre deux puissances de 2.
7. Estimer le nombre de comparaisons effectuées par le tri.
8. Estimer le nombre d'écritures dans un tableau réalisées par le tri.
9. Conclure sur la complexité temporelle du tri fusion.
10. Estimer la complexité spatiale du tri fusion.
11. Proposer une implémentation impérative du tri fusion.

Remarque 1 : pour le principe on ne déclare qu'un tableau temporaire en début de fonction, pour ne pas réaliser de nouvelle allocation lors de chaque appel. En Python ces allocations ne sont pas couteuses.
Remarque 2 : si l'on cherche la médiane d'un tableau de valeurs numériques, on peut trier le tableau par tri-fusion et retourner la valeur située au milieu du tableau trié. La complexité temporelle du calcul de la médiane est alors en $\Theta(n \ln n)$. Il est possible d'améliorer cette complexité temporelle.