MPSI - ITC - TD 13 - Tri-Fusion

Introduction, principe

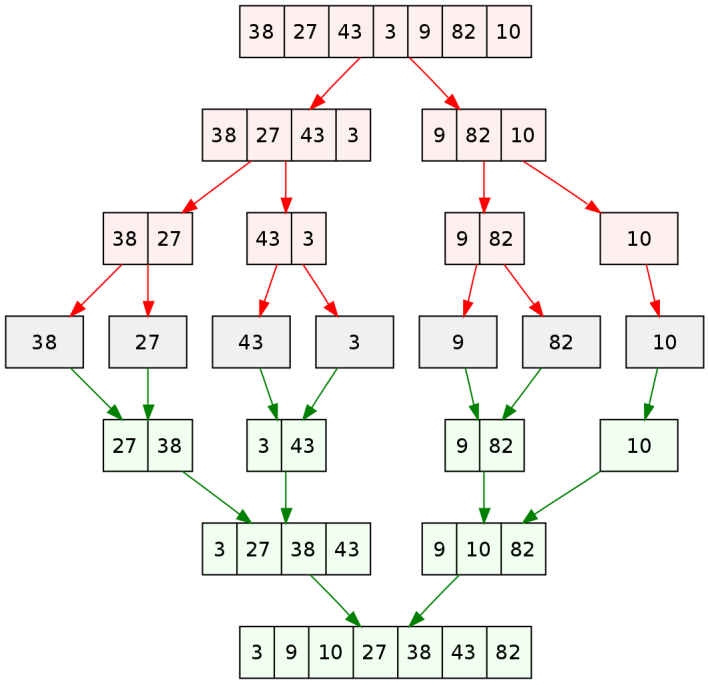
Le **tri fusion** (*mergesort* en anglais) est un tri **dichotomique récursif**, utilisant le principe *diviser pour régner*. On attribue sa découverte à John von Neumann en 1945.

Le principe est le suivant :
- on partage le tableau en deux moitiés de taille égale (à une unité près)
- on trie les deux moitiés (on fait en réalité des appels récursifs, jusqu'à avoir un tableau de taille 1 qui est trié)
- on fusionne les deux moitiés triées à chaque fois

Admis : La meilleure complexité que peut avoir un tri par comparaison est $\Theta(n\ln(n))$.

Remarques sur le tri fusion :
- La complexité du tri fusion est en $\Theta(n\ln(n))$, elle est donc optimale ! (C'est le premier algorithme de tri par comparaison non-quadratique qu'on voit en ITC... mais ce ne sera pas le dernier ;))
- Ce tri est également connu sous le nom de *tri par partition-fusion*.
- Ce tri peut facilement être parallélisé : on peut confier le traitement des sous-problèmes à des unités d'exécution distinctes. (La parallélisation de calculs est largement hors-programme, je vous le dis pour votre culture générale).

Un exemple d'exécution du tri fusion :



Exercice 1 (Le tri fusion)

1. Coder et tester la fonction `fusion(l1, l2)` vue en cours qui prend en entrée deux listes triées l1 et l2 et renvoie une permutation triée de l1 + l2.
2. Coder et tester la fonction tri_fusion vue en cours.
3. En supposant que la fonction fusion est correcte et termine, prouver que la fonction tri_fusion est correcte et termine.
4. Ce tri est il stable ? Si ce n'est pas le cas, rendez-le stable.
5. Vérifiez expérimentalement la stabilité de votre tri en codant une variante tri_fusion_paires qui prend en entrée une liste de paires (prenom, nombre) (comme pendant le cour sur les tris) et qui les trie selon leur première composante. Vous aurez besoin de la fonction auxiliaire fusion_paires correspondante. Sur quelle entrée pouvez-vous tester tri_fusion_paires pour vérifier qu'il est stable ? Quelle sortie devez vous alors avoir ?
6. Majorez le nombre de comparaisons effectuées par la fonction fusion et en déduire que le nombre de comparaisons effectuées pendant un appel à tri_fusion pour une liste de taille n est majoré par la fonction $C(n)$ définie par

$$\begin{cases} C(0) = C(1) = 0 \\ \forall n \geq 2, C(n) = C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C\left(\left\lceil \frac{n}{2} \right\rceil\right) + n \end{cases}$$

7. On pose $\forall k \in \mathbb{N}, u_k = \dfrac{C(2^k)}{2^k}$. Que dire de la suite (u_k) ?
8. En supposant que la fonction C est croissante, montrer qu'elle est dans $\mathcal{O}(n \ln(n))$.
9. Prouver la correction de la fonction fusion à l'aide d'une propriété invariante.
10. Voici une variante *funky* du tri fusion, à regarder et à comprendre :

```
def tri_fusion(l):
    n = len(l)
    if n > 1 :
        i = 0
        j = n - 1
        m = n // 2
        z = tri_fusion(l[:m]) + reversed(tri_fusion(l[m:]))
        for k in range(n):
            if z[i] <= z[j] :
                l[k] = z[i]
                i = i + 1
            else :
                l[k] = z[j]
                j = j - 1
    return l
```