TD1 – Tri partition/fusion (merge sort)

Concept : Diviser pour régner; Analyse de coût
Méthode : Décomposition récursive

Auteur: Nicolas Gast

Le tri fusion est un algorithme de tri utilisant le principe de "diviser pour régner", inventé par John von Neumann en 1945. Il effectue au plus $O(n \log n)$ opérations, qui est la complexité en temps dans le pire cas optimale parmi les algorithmes de tri à comparaison. Le tri fusion (ou ses variantes, comme Timsort) est utilisé par des librairies standards de plusieurs langages, comme Python, Java ou Perl.
Dans ce TD, on propose d'étudier la complexité du tri fusion, et d'une variante moins connue : le tri fusion en place.

Exercice 1: Tri fusion "classique"

L'algorithme de tri fusion est un algorithme récursif qui consiste à trier les deux moitiés du tableau séparément puis à fusionner les deux sous-tableaux triés ainsi obtenus. Pour trier un tableau de taille n, il s'opère récursivement de la façon suivante :

— On trie le sous tableau constitué des $\lceil n/2 \rceil$ premiers éléments du tableau.
— On trie le sous tableau constitué des $\lfloor n/2 \rfloor$ derniers éléments du tableau.
— On fusionne les deux tableaux triés en un seul tableau trié.

1. On veut trier le tableau de caractères "AZERTYUIOPQSDFGHJKLM".
   a) Écrire les valeurs des paramètres et les valeurs de retour de tous les appels récursifs à la fonction `tri_fusion`.
   b) Compter le nombre de comparaisons qu'effectue cet algorithme sur cet exemple.
2. Écrire (en pseudo-code) une fonction `fusion(T1, T2)` qui prend en entrée deux tableaux triées par ordre croissant (de taille n_1 et n_2) et rend un nouveau tableau de taille $n_1 + n_2$ contenant les éléments de T1 et T2 triés par ordre croissant.
   a) Quel est la complexité de votre algorithme ?
   b) Si ce n'est pas le cas : améliorer votre algorithme pour qu'il effectue au plus $n_1 + n_2$ opérations.
3. Écrire (en pseudo-code), une fonction récursive `tri_fusion(T)` qui effectue le tri de T par ordre croissant.
4. Soit $C(n)$ le temps que met le tri fusion pour trier un tableau de taille n. Donner une formule de récurrence pour $C(n)$ et la résoudre.

Exercice 2: Tri fusion en place

Au prix d'une complexité plus grande, on peut effectuer le tri fusion sans utiliser de tableau extérieur. Pour cela, on modifie l'opération de fusion. L'idée est la suivante : pour fusionner deux sous tableaux V et W, on découpe chaque sous-tableau en deux parties de même taille (pour simplifier, on suppose pour l'instant que $n = 2^k$). Ces parties sont notées V_1, V_2 et W_1, W_2. On effectue alors les opérations de fusion suivantes (voir Figure 1) :

— fusion(V_1,W_1)
— fusion(V_2,W_2)
— fusion(V_2',W_1')

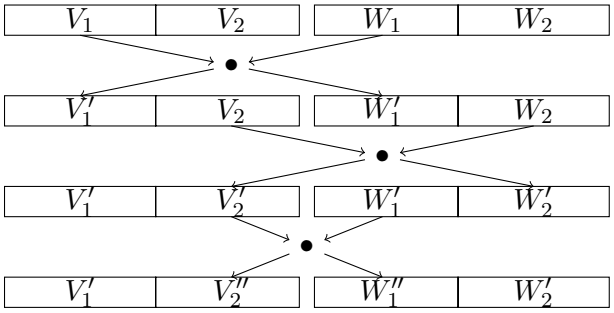

FIGURE 1 – Les opérations de fusion du tri de fusion en place

1. Soit $D(n)$ le nombre de comparaisons qu'effectue l'algorithme de fusion en place pour fusionner deux tableaux de taille $n/2 = 2^{k-1}$ et $n/2 = 2^{k-1}$.
   a) Écrire une formule de récurrence pour $D(n)$.
   b) En déduire la complexité de la fusion en place lorsque $n = 2^k$.
   c) En déduire la complexité du tri en place lorsque $n = 2^k$.
2. Montrer que l'opération de fusion est correcte.
3. Écrire la fonction `fusion_en_place` en pseudo-code.
4. Écrire la fonction `tri_en_place` en pseudo-code.
5. Généraliser l'algorithme et l'étude de la complexité à des tableaux de taille quelconque (différent de 2^k).

Exercice 3: Exercice à rendre (extrait du deuxième Quick 2014-2015)

Pour un problème donné, un algorithme naïf a une complexité en $O(n^3)$. On a aussi trouvé deux solutions de type diviser pour régner :

Div1 Découper le problème de taille n en deux sous-problèmes de taille $n/2$ et les recombiner en temps $O(n^2)$.

Div2 Découper le problème en quatre sous-problèmes de taille $n/3$ et les recombiner en temps $O(n)$.

1. Quelle est la complexité de Div1 ?
2. Quelle est la complexité de Div2 ?
3. Quelle solution choisir : naïf, div1 ou div2 ?

Rappel : L'étude de la complexité d'algorithmes de type *diviser pour régner* implique souvent des formule de récurrences reliant $C(n)$ à $C(n/b)$. Le résultat suivant est alors souvent utile :

Théorème 1 (Master theorem, Theorem 4.1 du livre de Cormen et. al)**.** *Soit $a \geq 1$ et $b > 1$ deux constantes. Soit $f(n)$ une fonction et $C(n)$ définit par la récurrence :*

$$C(n) = aC\left(\frac{n}{b}\right) + f(n).$$

$C(n)$ a les bornes asymptotiques suivantes :

$C(n) = \Theta(n^{\log_b a})$ *si* $f(n) = O(n^{\log_b a - \varepsilon})$ *pour un* $\varepsilon > 0$;
$C(n) = \Theta(n^{\log_b a} \log n)$ *si* $f(n) = \Theta(n^{\log_b a})$
$C(n) = \Theta(f(n))$ *si* $f(n) = \Omega(n^{\log_b a + \varepsilon})$ *pour un* $\varepsilon > 0$ *et si* $f(n) \geq acf(n/b)$ *pour* $c \geq 1$.