1 Tri à bulles

Le *tri à bulles* d'un tableau consiste à le parcourir en échangeant toute paire d'éléments consécutifs présentant une inversion et à recommencer jusqu'à ce que le tableau soit trié.

Exercice 1 *Montrer qu'après k parcours du tableau, les k plus grands éléments du tableau sont à la bonne place.*

Exercice 2 *Écrire de manière détaillée cet algorithme (en pseudo code ou en C). On portera une attention toute particulière aux bornes des boucles mises en jeu.*

Exercice 3 *Quelle est la complexité de cet algorithme dans le cas le pire et en moyenne, en terme de nombre de comparaisons entre éléments du tableau ?*

2 Tri fusion

Le *tri par fusion*, consiste à "couper" le tableau à trier en deux sous-tableaux de même taille, puis à appeler **récursivement** le tri sur les deux sous-tableaux ainsi obtenus, que l'on fusionne une fois triés à l'aide d'une fonction de complexité linéaire. En voici la version en pseudo-code où A est le tableau et (p, r) les indices entres lesquels on souhaite le trier :

Algorithme 1 TRI-FUSION(A, p, r)
1 **si** $p < r$ **alors**
2 $\quad q \leftarrow \lfloor (p + r)/2 \rfloor$
3 $\quad$ TRI-FUSION(A, p, q)
4 $\quad$ TRI-FUSION$(A, q + 1, r)$
5 $\quad$ FUSIONNER(A, p, q, r)

Exercice 4 *Comment réaliser en temps linéaire (en $r - p$) la fusion des deux tableaux triés situés entre p et q et entre $q + 1$ et r ?*

Exercice 5 *Calculer la complexité en terme de nombre de comparaisons pour n de la forme particulière $n = 2^k$.*

Exercice 6 *Généraliser le résultat précédent au cas d'un entier n quelconque.*

3 Exponentiation binaire

Le problème de l'exponentiation consiste, étant donné deux entiers a et b à calculer a^b. Afin de réaliser efficacement ce calcul, l'idée des algorithmes dits *d'exponentiation binaire* est de décomposer b en base 2 :

$$b = \sum_{i=0}^{k} b_i \times 2^i \quad \text{avec} \quad b_i \in \{0, 1\}$$

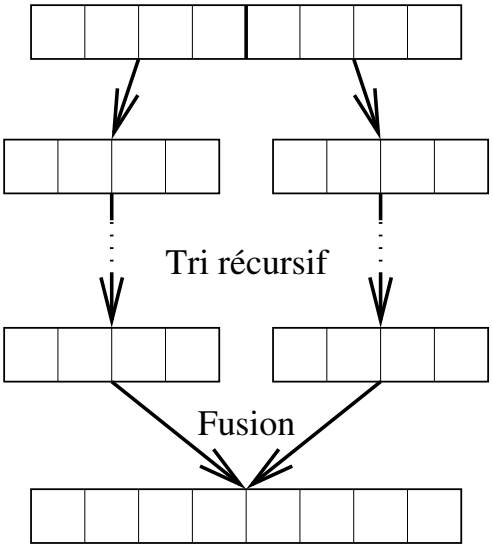

FIGURE 1 – Représentation graphique du tri fusion

On peut ensuite remarquer que

$$a^b = a^{\left(\sum_{i=0}^{k} b_i \times 2^i\right)} = \prod_{i=0}^{k} a^{b_i \times 2^i} = \prod_{i=0}^{k} \left(a^{(2^i)}\right)^{b_i}$$

Afin de calculer a^b, il suffit donc d'être capable de calculer les $a^{(2^i)}$ qui peuvent s'obtenir par élévation successive au carré et d'en multiplier ensuite certains entre eux (ceux pour lesquels b_i vaut 1).

Exercice 7 *Quelle est la complexité de l'algorithme naïf d'exponentiation réalisant l'opération $a \times a \times ... \times a$.*

Exercice 8 *Écrire un algorithme capable de décomposer un entier en base 2.*

Exercice 9 *En déduire une fonction capable de calculer rapidement a^b. Quelle est sa complexité temporelle et spatiale, dans le cas le pire et en moyenne.*

Exercice 10 *Modifier la fonction précédente de manière à ce que sa complexité en mémoire soit constante (i.e. indépendante de la taille de l'exposant b).*

Application à la cryptographie :

Le système de chiffrement RSA, très utilisé aujourd'hui, repose sur l'opération d'exponentiation modulaire puisque, à partir d'un message M représenté par un entier, on obtient son chiffré en calculant

$$C = M^e \bmod n$$

où mod$n$ désigne la réduction modulaire "modulo n".

Exercice 11

1. *Comment adapter l'algorithme précédant pour intégrer les réductions modulaires ?*
2. *Les nombres manipulés (n, m et e) sont typiquement longs de 1024 bits ; combien d'opérations sont nécessaires pour calculer le chiffré d'un message ?*
3. *On peut dans certains cas se contenter d'exposants plus petits. Un choix classique est $e = 65537$; pourquoi ce nombre plutôt qu'un autre nombre de 16 bits ?*