Outils logiques et algorithmiques – TD 3 – Correction

Exercice 1

1. $C(0) = 0$ et $C(n+1) = 2C(n) + 1$. D'où : $C(n) = 2^n - 1$ (preuve simple par récurrence sur n, ou remarquer que $C(n+1) + 1 = 2(C(n) + 1)$, c'est-à-dire que $(C(n) + 1)_{n \in \mathbb{N}}$ est une suite géométrique).
2. Nombre d'opérations nécessaires : $2^{64} - 1 > 16 \times 10^{18}$. D'où plus de 16×10^9 secondes, qui font plus de 185 000 jours et plus de 500 ans.

Exercice 2

1. Méthodes traditionnelle : n^2 multiplications de chiffres (chaque chiffre de a est multiplié avec chaque chiffre de b).
2. Équations récursives :

$$\left\{ \begin{array}{rcl} C(1) & = & 1 \\ C(n) & = & 3C(n-1) \end{array} \right.$$

   Suite géométrique : $C(n) = 3^n C(1) = 3^n$.

3. Équations récursives :

$$\left\{ \begin{array}{rcl} C(1) & = & 1 \\ C(n) & = & 4C(\frac{n}{2}) \end{array} \right.$$

   On démontre que $C(2^x) = (2^x)^2$ par récurrence sur x.
   - Cas de base : $C(2^0) = 1 = (2^0)^2$.
   - Hérédité. Soit x tel que $C(2^x) = (2^x)^2$. Alors $C(2^{x+1}) = 4C(2^x) = 4(2^x)^2 = (2 \times 2^x)^2 = (2^{x+1})^2$.

   La complexité est donc comparable à la version naïve.

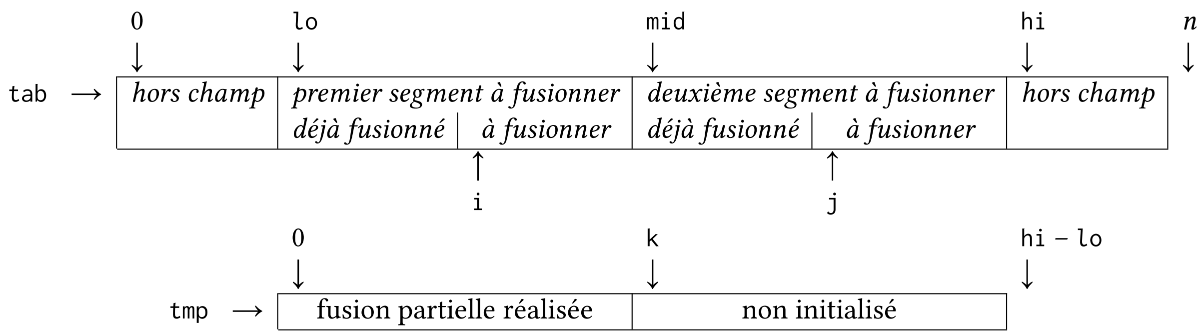
4. Équations récursives :

$$\left\{ \begin{array}{rcl} C(1) & = & 1 \\ C(n) & = & 3C(\frac{n}{2}) \end{array} \right.$$

   Valable à condition de ne bien calculer qu'une seule fois les deux produits $a_1 b_1$ et $a_2 b_2$ qui servent deux fois chacun. Par récurrence, $C(2^x) = 3^x$. Autrement dit, si n est un puissance de 2, on a $C(n) = 3^{\log_2(n)}$. Fin du calcul : $C(n) = 3^{\log_2(n)} = (2^{\log_2(3)})^{\log_2(n)} = 2^{\log_2(3) \times \log_2(n)} = (2^{\log_2(n)})^{\log_2(3)} = n^{\log_2(3)}$. On a $\log_2(3) < 1.6$, donc c'est mieux que la multiplication naïve.

Exercice 3

1. Schéma général.



   Invariants sur les plages de valeurs des variables :
   - `i` est un indice du segment `tab`$[lo, mid[$, ou est égal à `mid`, c'est-à-dire `i` $\in$ [`lo`, `mid`]
   - `j` est un indice du segment `tab`$[mid, hi[$, ou est égal à `hi`, c'est-à-dire `j` $\in$ [`mid`, `hi`]

— `k = i + j`.

Invariant sur la forme des tableaux. Le segment $[0, \texttt{k}[$ de `tmp` contient une permutation des segments $[\texttt{lo}, \texttt{i}[$ et $[\texttt{mid}, \texttt{j}[$ de `tab`. En outre, toutes les valeurs de $\texttt{tmp}[0, \texttt{k}[$ sont inférieures ou égales à toutes les valeurs des segments $\texttt{tab}[i, mid[$ et $\texttt{tab}[j, hi[$.

$$\begin{cases} \exists \sigma \in \mathfrak{S}_{\texttt{k}}, \forall x \in [0, x[, \ (\sigma(x) < \texttt{i} - \texttt{lo} \wedge \texttt{tmp}[x] = \texttt{tab}[\texttt{lo} + \sigma(x)]) \ \vee \ (\sigma(x) \geq \texttt{i} - \texttt{lo} \wedge \texttt{tmp}[x] = \texttt{tab}[\texttt{mid} + \sigma(x) - (\texttt{i} - \\ \forall x \in [0, k[, y \in [\texttt{i}, \texttt{mid}[, \ \texttt{tmp}[x] \leq \texttt{tab}[y] \\ \forall x \in [0, k[, y \in [\texttt{j}, \texttt{hi}[, \ \texttt{tmp}[x] \leq \texttt{tab}[y] \end{cases}$$

2. Arrêt quand $\texttt{i} = \texttt{mid}$ ET $\texttt{j} = \texttt{hi}$. À chaque tour, l'un exactement de `i` ou `j` est incrémenté de 1. Au total $(\texttt{mid} - \texttt{lo}) + (\texttt{hi} - \texttt{mid}) = \texttt{hi} - \texttt{lo} = n$ tours de boucle. On a 4 accès par tour tant que ni `i` ni `j` n'a atteint sa limite, puis 2 accès par tour une fois que l'un des deux l'a atteinte. Il faut attendre au minimum $n_0 = \min(\texttt{mid} - \texttt{lo}, \texttt{hi} - \texttt{mid})$ pour que l'un des deux compteurs atteigne sa limite. On a donc un nombre total compris entre $4n_0 + 2(n - n_0) = 2(n + n_0)$ et $4n$ pour la boucle `while`. Si `mid` est précisément au milieu de `lo` et `hi`, on obtient $n_0 = \frac{n}{2}$, d'où un nombre d'accès compris entre $3n$ et $4n$. Dans tous les cas, on ajoute à la fin $2n$ accès (n lectures et n écritures) pour recopier le contenu de `tmp` dans $\texttt{tab}[\texttt{lo}, \texttt{hi}[$, d'où total entre $5n$ et $6n$.
3. Les tableaux de taille 0 ou 1 ne nécessitent aucun accès. Au-delà, on ajoute les coûts des deux appels récursifs au coût de la fusion.

$$\begin{cases} C(0) &=& 0 \\ C(1) &=& 0 \\ C(n) &=& C(\lfloor \frac{n}{2} \rfloor) + C(\lceil \frac{n}{2} \rceil) + C_{\text{merge}}(n) \qquad \text{si } n > 2 \end{cases}$$

Si on s'intéresse au pire cas, où $C_{\text{merge}}(n)$ a toujours la valeur maximale, la dernière équation se réécrit

$$C(n) \quad = \quad C(\lfloor \tfrac{n}{2} \rfloor) + C(\lceil \tfrac{n}{2} \rceil) + 6n$$

4. Lorsque n est une puissance de 2, les équations sont simplifiées.

$$\begin{cases} C(2^0) &=& 0 \\ C(2^k) &=& 2 \times C(2^{k-1}) + 6 \times 2^k \qquad \text{si } k > 1 \end{cases}$$

Résolution : en divisant les deux côtés de la deuxième équation par 2^k on obtient

$$\frac{C(2^k)}{2^k} = \frac{2 \times C(2^{k-1})}{2^k} + \frac{6 \times 2^k}{2^k} = \frac{C(2^{k-1})}{2^{k-1}} + 6 = \ldots = \frac{C(2^0)}{2^0} + 6 \times k = 6 \times k$$

Ainsi

$$C(2^k) \quad = \quad 6 \times k \times 2^k$$

Si $n = 2^k$ on a donc

$$C(n) \quad = \quad 6n log(2)$$

Exercice 4

1. Équations récursives :

$$\begin{cases} C(1) = C(2) &=& 1 \\ C(n) &=& 3C(\frac{2n}{3}) + 1 \qquad \text{si } n > 2 \end{cases}$$

Application de *master theorem* avec $a = 3$, $b = 1,5$, et $f(n) = 1$: on est dans le premier cas et $C(n) = \Theta(n^{\log_{1,5}(3)}) \approx \Theta(n^{2,7})$.
2. C'est le pire qu'on ait vu !

Exercice 5

1. `r - l` $= n$ accès.
2. Équations récursives :

$$\begin{cases} C(0) = C(1) & = & 0 \\ C(n) & = & 2C(\frac{n}{2}) + n \qquad \text{si } n > 1 \end{cases}$$

   Master theorem avec $a = 2$, $b = 2$ et $f(n) = n$. On est dans le cas 2 : complexité $\Theta(n\log(n))$.
3. En notant k le nombre d'éléments différents on obtient l'ordre de grandeur $\Theta(n \times k) = \mathcal{O}(n^2)$ pour l'algorithme naïf. L'approche naïve peut rester compétitive si le nombre d'éléments différents est faible.

Exercice 6

1. Dans tous les cas : $\Theta(k \times n)$.
2. Meilleurs cas : $\Theta(n)$. Pire cas : $\Theta(n^2)$.
3. En moyenne : $\Theta(n)$.