Solutions du devoir 1

Exercice 1 **(5 points par question)** Démontrer si les propriétés suivantes sont vraies ou fausses :

a) $f_1(n) = \Theta(g_1(n))$ et $f_2(n) = \Theta(g_2(n)) \Rightarrow f_1(n)f_2(n) = \Theta(g_1(n)g_2(n))$

Il existe $c_1, c_2, c_3, c_4 \in]0; +\infty[$ et $n_1, n_2 \in \mathbb{N}$ tels que :

$$c_1 g_1(n) \leq f_1(n) \leq c_2 g_1(n) \text{ pour tout entier } n \geq n_1$$
$$c_3 g_2(n) \leq f_2(n) \leq c_4 g_2(n) \text{ pour tout entier } n \geq n_2$$

Puisque nos fonctions sont positives (**pas de points en moins si cette justification est absente**), on peut multiplier ces deux inégalités.

Si on pose $N = \max(n_1, n_2)$, $c_5 = c_1 c_3$ et $c_6 = c_2 c_4$, on a alors :

$$c_5 g_1(n) g_2(n) \leq f_1(n) f_2(n) \leq c_6 g_1(n) g_2(n) \text{ pour tout entier } n \geq N$$

et donc $f_1(n)f_2(n) = \Theta(g_1(n)g_2(n))$.

b) $f(n) = O(g(n)) \Rightarrow 3^{f(n)} = O(3^{g(n)})$

Cette propriété est fausse comme le montre le contre-exemple suivant.

On pose $f(n) = 2n$ et $g(n) = n$. Comme $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 2 \in [0; +\infty[$, on a bien

$$f(n) = O(g(n)).$$

Cependant $\lim_{n\to+\infty} \frac{3^{f(n)}}{3^{g(n)}} = \lim_{n\to+\infty} \frac{3^{2n}}{3^n} = \lim_{n\to+\infty} 3^n = +\infty$ et donc

$$3^{f(n)} \neq O(3^{g(n)}).$$

Exercice 2 **(10 points)** L'ordre croissant de complexité des fonctions est le suivant (**1 point en moins pour chaque mauvais classement**, **aucune justification requise même pas les simplifications) :**

$$\lg(n^3) = 3\lg(n), \lg^3(n), \sqrt{n}, \frac{n}{\lg(n)}, 3^{3\log_9(n)} = n\sqrt{n}, \frac{3n^3 + 5n}{n+1} = \Theta(n^2), n^2\lg(n), 2^n$$

<u>Exercice 3</u> **(5 points par question)** Démontrer si les propriétés suivantes sont vraies ou fausses **à l'aide des définitions formelles** :

a) $n! = \Theta\big((n+1)!\big)$

Cette propriété est fausse. En effet, si elle était vraie, il existerait $c_1, c_2 \in \,]0;+\infty[$ et $n_0 \in \mathbb{N}$, tels que

$$c_1(n+1)! \leq n! \leq c_2(n+1)! \text{ pour tout entier } n \geq n_0.$$

Si on divise par $(n+1)!$ on obtient

$$c_1 \leq \frac{1}{n+1} \leq c_2 \text{ pour tout entier } n \geq n_0,$$

ce qui entraine $c_1 = 0$, ce qui n'est pas possible.

b) $3n^3 + 5n^2 + 7 = O(n^3)$

Cette propriété est vraie. On doit trouver $c > 0$ et $n_0 \in \mathbb{N}$ tels que :
$f(n) = 3n^3 + 5n^2 + 7 \leq cn^3$ pour tout entier $n \geq n_0$. On a en fait :
$3n^3 + 5n^2 + 7 \leq 3n^3 + 5n^3 + 7n^3$ pour tout entier $n \geq 1$.
L'affirmation est donc vérifiée pour $c = 15$ et $n_0 = 1$.

c) $\log_3(n) = \Theta\big(\log_4(n)\big)$

Cette propriété est vraie car $\log_3(n) = \frac{1}{\log_4(3)}\log_4(n)$. On a donc

$$\frac{1}{\log_4(3)}\log_4(n) \leq \log_3(n) \leq \frac{1}{\log_4(3)}\log_4(n) \text{ pour tout entier } n \geq 1.$$

d) $n! = \Omega(n^2)$

Cette propriété est vraie car, par exemple, $n! \geq n^2$ pour tout $n \geq 4$.
(pas de preuve nécessaire, mais 2 points en moins si l'inégalité n'est pas vérifiée pour toutes les valeurs de $\boldsymbol{n \geq n_0}$)

<u>Exercice 4</u> **(20 points)** On considère l'équation de récurrence

$$T(n) = 4T\left(\left\lceil\frac{n}{2}\right\rceil\right) + n^3,\ T(1) = 1.$$

On utilise dans toute la suite la règle de l'harmonie (**1 point en moins si cette règle n'est pas nommée au moins une fois, aucune justification requise que cette règle peut bien être appliquée ici**).

a) Avec la méthode de l'arbre récursif (**7 points**)

On pose $n = 2^p$.

$$T(n) = \sum_{i=0}^{p-1} 4^i f\left(\frac{n}{2^i}\right) + 4^p$$

$$T(n) = \sum_{i=0}^{p-1} 4^i f\left(\frac{n}{2^i}\right) + 4^{\log_2(n)}$$

$$T(n) = \sum_{i=0}^{p-1} 4^i \left(\frac{n}{2^i}\right)^3 + n^{\log_2(4)}$$

$$T(n) = n^3 \sum_{i=0}^{p-1} \left(\frac{1}{2}\right)^i + n^2$$

$$T(n) = n^3 \frac{\left(\frac{1}{2}\right)^p - 1}{\frac{1}{2} - 1} + n^2$$

$$T(n) = n^3 \frac{\frac{1}{n} - 1}{\frac{1}{2} - 1} + n^2$$

$$T(n) = 2n^3 - n^2 \in \Theta(n^3)$$

b) Avec la méthode de substitution (**7 points**)

On veut vérifier que $T(n) = \Omega(n^3)$.
On doit donc trouver une constante $c > 0$ telle que $T(n) \geq cn^3$

- **Récurrence** : on suppose $T\left(\left\lceil\frac{n}{2}\right\rceil\right) \geq c\left\lceil\frac{n}{2}\right\rceil^3$. Si on remplace (substitution) dans la relation de récurrence, on obtient :

$$\begin{aligned} T(n) = 4T\left(\left\lceil\frac{n}{2}\right\rceil\right) + n^3 \geq 4c\left\lceil\frac{n}{2}\right\rceil^3 + n^3 \\ \geq \frac{c}{2}n^3 + n^3 \\ \geq \left(\frac{1}{2}c + 1\right)n^3 \\ \geq cn^3, \end{aligned}$$

la dernière égalité étant vraie pour toute constante $c \leq 2$.

- **Initialisation** : Si on prend $c = 1$, la propriété est vérifiée pour $n = 1$.

On montre que $T(n) = O(n^3)$ de la même manière, en utilisant la règle de la l'harmonie.

c) Avec la méthode générale (**6 points**)

$a = 4, b = 2$ et donc $\log_b(a) = 2$.

$f(n) = n^3 = \Omega(n^{2+1})$, donc le cas 3 s'applique avec $\varepsilon = 1 > 0$.

On doit vérifier la deuxième hypothèse :

$af\left(\frac{n}{b}\right) = 4\left(\frac{n}{2}\right)^3 = \frac{1}{2}n^3 \leq cf(n)$ avec $c = \frac{1}{2} < 1$.

<u>Conclusion</u> : $T(n) = \Theta(f(n)) = \Theta(n^3)$.

Exercice 5 **(10 points par question, 1 point en moins pour chaque erreur de calcul)**

a) Le pire cas est le cas où le tableau est trié **en ordre croissant (1 point en moins si inversé ou pas explicité)**.

$$T(\text{triBulles}) = nc_1 + \sum_{i=2}^{n}\big(ic_2 + (i-1)(c_3 + c_4)\big)$$
$$T(\text{triBulles}) = nc_1 + c_2\sum_{i=2}^{n} i + (c_3 + c_4)\sum_{i=2}^{n}(i-1)$$
$$T(\text{triBulles}) = nc_1 + c_2\left(\sum_{i=1}^{n} i\ -1\right) + (c_3 + c_4)\sum_{i=1}^{n-1} i$$
$$T(\text{triBulles}) = nc_1 + c_2\left(\frac{n(n+1)}{2} - 1\right) + (c_3 + c_4)\frac{(n-1)n}{2}$$
$$T(\text{triBulles}) = \frac{c_2 + c_3 + c_4}{2}n^2 + \frac{2c_1 + c_2 - c_3 - c_4}{2}n\ - c_2$$

b) Le meilleur cas est le cas où le tableau est déjà trié **en ordre décroissant (1 point en moins si inversé ou pas explicité)**. Le calcul est le même en remplaçant $c_3 + c_4$ par c_3, et on obtient :

$$T(\text{triBulles}) = \frac{c_2 + c_3}{2}n^2 + \frac{2c_1 + c_2 - c_3}{2}n\ - c_2$$

Exercice 6 **(10 points)** On considère le cas 3 de la méthode générale pour $a = 4$ et $b = 2$. Trouver une fonction $f(n) = \Omega\left(n^{\log_b(a)+\epsilon}\right)$, pour $\epsilon > 0$, telle qu'il n'existe aucune constante $c < 1$, telle que $af\left(\frac{n}{b}\right) \leq cf(n)$ pour n suffisamment grand.

(au moins 6 points sur 10 dès qu'il y a eu une « tentative pertinente », même si tout n'est pas rigoureux ou ne « fonctionne pas totalement »)

$a = 4, b = 2$ et donc $\log_b(a) = 2$.

On pose $f(n) = \begin{cases} n^3 & \text{si } n \text{ est pair} \\ n^3 2^n & \text{si } n \text{ est impair} \end{cases}$

$f(n) = \Omega(n^{2+1})$, donc cas 3 avec $\varepsilon = 1 > 0$.

Soit n un entier pair tel que $\frac{n}{2}$ est impair, on a alors :

$$af\left(\frac{n}{b}\right) = 4\left(\frac{n}{2}\right)^3 2^{\frac{n}{2}} = \frac{1}{2}n^3\sqrt{2}^n \geq f(n) = n^3,$$

La « deuxième hypothèse » ne peut donc être vérifiée.

<u>Exercice 7</u> **(5 points par question, 1 point en moins au a) ET au b) si ε n'est pas clairement identifié).** Utiliser, **si possible**, la méthode générale pour résoudre les récurrences suivantes :

a) $T(n) = 3T\left(\left\lceil \frac{n}{5} \right\rceil\right) + \log^3(n)$

$a = 3,\ b = 5, \log_5(3) \approx 0{,}682.$
$f(n) = \log^3(n) = O(\sqrt{n})$ par exemple, le cas 1 s'applique donc en prenant
$\varepsilon = \log_5(3) - \frac{1}{2} > 0$, et on obtient ainsi $T(n) = \Theta\left(n^{\log_5(3)}\right)$.

b) $T(n) = 2T\left(\left\lceil \frac{n}{8} \right\rceil\right) + \sqrt{n}\lg(n)$

$a = 2,\ b = 8, \log_8(2) = \frac{1}{3}.$
$f(n) = \sqrt{n}\lg(n) = \Omega(\sqrt{n})$ par exemple, le cas 3 s'applique donc en prenant
$\varepsilon = \frac{1}{2} - \frac{1}{3} > 0$, et on obtient ainsi $T(n) = \Theta\left(\sqrt{n}\lg(n)\right)$.

On doit vérifier la deuxième hypothèse (**1 point seulement en moins si oublié ou mal rédigé**) :

$af\left(\frac{n}{b}\right) = 2\sqrt{\frac{n}{8}}\lg\left(\frac{n}{8}\right) = \frac{\sqrt{n}}{\sqrt{2}}(\lg(n) - \lg(8)) \leq \frac{1}{\sqrt{2}}\sqrt{n}\lg(n)$, donc $c = \frac{1}{\sqrt{2}} < 1$.

c) $T(n) = 9T\left(\left\lceil \frac{n}{3} \right\rceil\right) + n^2$

$a = 9,\ b = 3, \log_3(9) = 2.$
$f(n) = n^2 = \Theta(n^2)$, donc le cas 2 s'applique, et on obtient ainsi
$T(n) = \Theta\left(n^{\log_b(a)}\lg(n)\right) = \Theta(n^2\lg(n))$.

d) $T(n) = 4T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n^2\log^5(n)$

$a = 4,\ b = 2, \log_2(4) = 2.$ $f(n)$ croît **plus vite** que $n^{\log_b(a)}$ mais pas « polynomialement » plus vite. On ne peut pas appliquer la méthode générale.

<u>Exercice 8</u> **(20 points)** On considère un tableau <u>trié</u> tab d'entiers <u>distincts</u> indicé de 1 à n.

a) Trouver un algorithme, dont la complexité temporelle est en $O(\lg(n))$ au pire cas, et qui trouve un indice $1 \leq i \leq n$, tel que $tab[i] = i$, à condition qu'un tel indice existe. **(10 points, même si les lignes 9 et 10 manquent mais que $<$ est remplacé par $\leq$ à la ligne 11)**

fonction rechercheT(tab, min, max) **retourne** entier

- entrées :
    - tab : tableau **trié** d'entiers distincts indicé de 1 à n
    - $min \leq max$: 2 entiers entre 1 et n
- sortie :
    - s : un entier tel que $tab[s] = s$ ou -1 si un tel indice n'existe pas

début
1 **si** $min = max$ **faire**
2 **si** $tab[min] = min$ **faire**
3 $s \longleftarrow min$
4 **sinon faire**
5 $s \longleftarrow -1$
6 **fin si**
7 **sinon faire**
8 $m \leftarrow \left\lfloor \frac{min+max}{2} \right\rfloor$
9 **si** $m = tab[m]$ **faire**
10 $s \longleftarrow m$
11 **sinon si** $m < tab[m]$ **faire**
12 $s \longleftarrow$ rechercheT(tab, min, m)
13 **sinon faire**
14 $s \longleftarrow$ rechercheT$(tab, m+1, max)$
15 **fin si**
16 **fin si**
17 **retourner** s
fin rechercheT

b) Démontrer la complexité temporelle de votre algorithme avec la méthode générale. **(5 points)**

Au pire cas, on obtient la relation de récurrence $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c$.

$a = 1$, $b = 2$, $\log_2(1) = 0$.

$f(n) = c = \Theta(n^0)$, donc le cas 2 s'applique, et on obtient ainsi

$T(n) = \Theta\left(n^{\log_b(a)} \lg(n)\right) = \Theta(\lg(n))$.

c) Expliquer (informellement) pourquoi votre algorithme est exact. **(3 points)**

Comme les éléments du tableau sont des entiers triés (en ordre croissant) et distincts, on a forcément $tab[i+1] \geq tab[i] + 1$. Si $tab[i] > i$, il n'est donc plus possible d'avoir $tab[k] = k$ pour $k > i$ (k ne peut plus « rattraper » $tab[k]$). On peut donc se contenter alors de vérifier pour les indices strictement inférieurs à i.

d) Peut-on trouver un algorithme aussi efficace si les entiers ne sont pas distincts ? Pourquoi ? **(2 points)**

Si les entiers ne sont pas distincts, l'information sur un élément du tableau ne peut pas nous aider à éliminer systématiquement la moitié du tableau. **(Cette explication est suffisante, il n'est pas nécessaire d'avoir donné l'exemple ci-dessous pour avoir 2 points.)**

Par exemple pour les trois tableaux $tab_1 = [-1\ 2\ 2\ 6\ 9\ ]$, $tab_2 = [-1\ 0\ 2\ 4\ 7\ ]$ et $tab_3 = [-1\ 0\ 2\ 6\ 9\ ]$, on a $tab[3] < 3$. Pourtant, on obtient successivement $s_1 = 2 < 3$, $s_2 = 4 > 3$ et $s_3 = -1$.

Exercice 9 **(30 points)** :

a) Donner un algorithme de programmation dynamique pour résoudre le problème suivant :

Entrée : une matrice carrée A de taille $n \times n$ dont les coefficients valent 0 ou 1

Sortie : la largeur maximum k d'un carré de 1 dans A

Vous devez utiliser dans votre algorithme $\text{Max}[i][j]$ la largeur maximum d'un carré de 1 dans A, dont le coin inférieur droit est en position $(i; j)$, où i et j sont deux entiers entre 1 et n.

b) Expliciter le pire cas et le meilleur cas de votre algorithme, et donner un ordre de grandeur de la complexité temporelle de votre algorithme dans chacun de ces deux cas (aucune preuve formelle nécessaire).

Exemple d'entrée et de sortie :

Entrée : $A = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$

Sortie : $k = 3$

On a par exemple ici $\text{Max}[2][2] = 0, \text{Max}[4][2] = 1$, $\text{Max}[2][6] = 2$ et $\text{Max}[4][5] = 3$.

a) **(20 points, 2 points en moins si utilisation de min au lieu des lignes 10 à 16)**

fonction TrouverLongueurCarreMax(A) **retourne** entier

- entrée :
  A : matrice dont les coefficients valent 0 ou 1, et dont les lignes et les colonnes sont numérotées de 1 à n
- sortie :
  k : entier, la largeur maximum d'un carré de 1 dans A

début

```
1     pour i ⟵ 1 haut n faire
2         Max[i][1] ⟵ A[i][1]
3         Max[1][i] ⟵ A[1][i]
4     fin pour
5     pour i ⟵ 2 haut n faire
6         pour j ⟵ 2 haut n faire
7             si A[i][j] = 0 alors
8                 Max[i][j] ⟵ 0
9             sinon
10                temp ⟵ Max[i − 1][j − 1]
11                si Max[i − 1][j] < temp alors
12                    temp ⟵ Max[i − 1][j]
13                fin si
14                si Max[i][j − 1] < temp alors
15                    temp ⟵ Max[i][j − 1]
16                fin si
17                Max[i][j] ⟵ 1 + temp
18            fin si
19        fin pour
20    fin pour
21    res ⟵ 0
22    pour i ⟵ 1 haut n faire
23        pour j ⟵ 1 haut n faire
24            si Max[i][j] > res alors
25                res ⟵ Max[i][j]
26            fin si
27        fin pour
28    fin pour
29    retourner res
```

fin TrouverLongueurCarreMax

b) Le meilleur cas est le cas d'une matrice de 0, et le pire cas le cas d'une matrice de 1. **(4 points, 2 points par cas)**

Dans tous les cas, on calcule une matrice de taille $n \times n$, dont chaque coefficient prend un temps borné par une constante à calculer. La complexité temporelle est donc en $\Theta(n^2)$. **(6 points, 2 points en moins si la justification n'est pas assez claire)**