# Devoir 1

Vous devez remettre vos solutions **sur Moodle avant le mercredi 3 mars 2021 à 23h55 sous la forme d'un unique fichier pdf**. Un retard de 24 heures au maximum sera accepté : pénalité de $\frac{m}{90}$ points, où $m$ est le nombre de minutes de retard. La note 0 sera attribuée au-delà d'un retard de 24 heures. Le nombre total de points pour ce devoir est 160.

Le devoir peut être fait en équipes de 2 étudiant-e-s au maximum. Aucune solution manuscrite ne sera acceptée.

Exercice 1 **(10 points)** Démontrer si les propriétés suivantes sont vraies ou fausses :

a) $f_1(n) = \Theta\big(g_1(n)\big)$ et $f_2(n) = \Theta\big(g_2(n)\big) \Rightarrow f_1(n)f_2(n) = \Theta\big(g_1(n)g_2(n)\big)$
b) $f(n) = O\big(g(n)\big) \Rightarrow 3^{f(n)} = O\big(3^{g(n)}\big)$

Exercice 2 **(10 points)** Classer dans l'ordre croissant de complexité les fonctions suivantes (aucune justification requise) :

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

Exercice 3 **(20 points)** Démontrer si les propriétés suivantes sont vraies ou fausses **à l'aide des définitions formelles** :

a) $n! = \Theta\big((n+1)!\big)$
b) $3n^3 + 5n^2 + 7 = O(n^3)$
c) $\log_3(n) = \Theta\big(\log_4(n)\big)$
d) $n! = \Omega(n^2)$

Exercice 4 **(20 points)** On considère l'équation de récurrence

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

Résoudre cette équation de récurrence :

a) Avec la méthode de l'arbre récursif
b) Avec la méthode de substitution
c) Avec la méthode générale

Exercice 5 **(20 points)** On propose l'algorithme de tri suivant :

**procedure** triBulles($tab$)

- entrée :

$tab$ : tableau indicé de 1 à $n$

- sortie :

$tab$ : tableau indicé de 1 à $n$ trié

**début**

| | | |
|---|---|---|
| 1 | **pour** $i \leftarrow n$ **bas** 2 **faire** | $c_1$ |
| 2 | **pour** $j \leftarrow 1$ **haut** $i - 1$ **faire** | $c_2$ |
| 3 | **si** $tab[j+1] > tab[j]$ **alors** | $c_3$ |
| 4 | $tab[j+1] \leftrightarrow tab[j]$ | $c_4$ |
| 5 | **fin si** | |
| 6 | **fin pour** | |
| 7 | **fin pour** | |

**fin** triBulles

Déterminer la complexité temporelle exacte de cet algorithme en fonction des temps d'exécution $c_1$, $c_2$, $c_3$ et $c_4$ :

a) au pire cas,
b) au meilleur cas.

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.

Exercice 7 **(20 points)** Utiliser, **si possible**, la méthode générale pour résoudre les récurrences suivantes :

a) $T(n) = 3T\left(\left\lfloor\frac{n}{5}\right\rfloor\right) + \log^3(n)$
b) $T(n) = 2T\left(\left\lfloor\frac{n}{8}\right\rfloor\right) + \sqrt{n}\,\lg(n)$
c) $T(n) = 9T\left(\left\lfloor\frac{n}{3}\right\rfloor\right) + n^2$
d) $T(n) = 4T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + n^2\log^5(n)$

Exercice 8 **(20 points)** On considère un tableau $tab$ d'entiers distincts 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.
b) Démontrer la complexité temporelle de votre algorithme avec la méthode générale.
c) Expliquer (informellement) pourquoi votre algorithme est exact.
d) Peut-on trouver un algorithme aussi efficace si les entiers ne sont pas distincts ? Pourquoi ?

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 $\text{Max}[4][5] = 3$.