TD n°5 - Corrigé

Exercice 1

On applique l'algorithme, que je ne re-décris pas ici :

1. Représentation exacte : 0|10000100|11000010000000000000000
2. Représentation exacte : 1|10000011|11100010100000000000000
3. Représentation approchée : 0|10000110|00100011100001000001100

Exercice 2

On vous donne les nombres suivants, sur 32 bits. Déterminer ce qu'ils représentent

1. C'est un nombre normalisé.
   Son signe est négatif et l'exposant décalé est 149 donc l'exposant réel est 22.
   La mantisse est $1 + 1/2 + 1/8$
   Le nombre est donc $-1,625 \times 2^{22} = -6815744$.
2. C'est un nombre normalisé.
   Son signe est positif et l'exposant décalé est 117 donc l'exposant réel est -10.
   La mantisse est $1 + 1/8 + 1/64 + 1/128 + 1/2048 \approx 1,1489$.
   Le nombre est donc environ -0.0011224747.
3. C'est NaN, l'exposant décalé est 255 et la mantisse non nulle.
4. C'est un nombre dénormalisé car l'exposant décalé est 0.
   Son signe est négatif et sa mantisse est $1/4 + 1/8$.
   Le nombre est donc $-0.375 \times 2^{-126} \approx -4.408 \times 10^{-39}$.

Exercice 3

Vrai ou faux ?

1. Oui car on s'intéresse aux grandes valeurs de n pour étudier l'efficacité des algorithme.
2. Non, ça dépend de ce qu'on fait à l'intérieur. Si on imbrique une autre boucle, ça peut être un $\Theta(n^2)$ par exemple.
3. Oui. Il suffit d'écrire la fonction si vous n'êtes pas convaincus.
4. C'est vrai par définition de Θ et O.
5. Non, quand l'entier n est stocké en mémoire, il prend $log_2(n)$ bits de mémoire.
6. C'est faux, on a introduit la notion de complexité pour s'abstraire au maximum du langage. La plupart des langages utilisent les mêmes boucles et des structures de données similaires, donc les algorithmes ont asymptotiquement les mêmes complexités.
   Par contre il est vrai que le temps d'exécution d'un programme en Python est parfois jusqu'à 10 fois plus élevé que celui du même programme en C. Ceci est du à des différences techniques de l'implémentation de l'interpréteur Python et du compilateur de C.

Exercice 4

Première suite

On définit $u_n = \dfrac{n(n+1)}{2}$. Cette suite est positive, donc on omettra les valeurs absolues dans la preuve.

D'un côté on a quelque soit $n \in \mathbb{N}$, $\dfrac{1}{2}n^2 \leq n^2/2 + n$ donc on a la moitié de ce qu'on veut. (avec $n_0 = 0$ et $B = 1/2$)

De l'autre on a quelque soit $n \in \mathbb{N}$, $n \leq n^2$, ce qui permet d'affirmer que $u_n = n^2/2 + n/2 \leq n^2$. On a donc montré par définition (avec $n_0 = 0$ et $A = 1$) que $u_n = O(n^2)$.

On peut conclure $u_n = \Theta(n^2)$.

Deuxième suite

On note $v_n = \log_2(n+1) + \ln(2n^2)$. Pour montrer le O, on va la majorer.

On a d'une part pour $n \geq 2$, $\log_2(n+1) = \dfrac{\ln(n+1)}{\ln(2)} \leq \dfrac{\ln(2n)}{\ln(2)}$ puisque $n + 1 \leq 2n$ et le logarithme est croissant.

On a donc pour $n \geq 2$ $\log_2(n+1) \leq \dfrac{\ln(2) + \ln(n)}{\ln(2)}$ puis $\log_2(n+1) \leq \dfrac{2\ln(n)}{\ln(2)}$ par croissance du logarithme.

D'autre part, $\ln(2n^2) = \ln(2) + 2 * \ln(n)$ par propriétés du logarithme.

On peut donc majorer par croissance du logarithme pour $n \geq 2$: $\ln(2n^2) \leq \ln(n) + 2 * \ln(n) = 3 * \ln(n)$.

Finalement pour $n \geq 2$, $u_n \leq (3 + \frac{2}{\ln(2)}) \ln(n)$, d'où $u_n = O(\ln(n))(n \to +\infty)$ (on utilise la définiton avec $n_0 = 2$ et $A = 3 + \frac{2}{\ln(2)}$)

Exercice 5

Pour chacune des fonctions ci-dessous, donner le nombre d'additions effectuées en fonction des valeurs des arguments.

1. Première fonction : la boucle fait $min(n, m)$ itérations puisque i et j augmentent de 1 en 1 et que la boucle s'arête dès que soit i dépasse n, soit j dépasse m. Le contenu de la boucle est 2 additions.
   Ainsi au total on fait $2min(n, m)$ additions.
2. Deuxième fonction : la différence est que cette fois on a un OU. La boucle va donc faire $max(n, m)$ itérations, et ne s'arrêtera que lorsque $i > n$ et $j > m$. Chaque itération fait 2 additions.
   Au total on a $2max(n, m)$ additions.
3. Troisième fonction : la boucle fait n itérations.
   À chaque itération on a une addition obligatoire et une autre dans le if qui s'effectue seulement quand $j \leq m$.
   Au début j vaut 1 donc le programme va dans le if et augmente j de 1. Ceci est fait tant que j n'atteint pas $m + 1$. Comme j augmente de 1 en 1 durant cette phase, on peut dire que le programme rentre dans le if $min(n, m)$ fois.
   La complexité est donc de $n + min(n, m)$ additions.
4. Quatrième fonction : La boucle fait toujours n itérations.
   À chaque itération, le programme rentre soit dans le if soit dans le else, fait donc une seule addition.
   Au total on fait donc n additions.

Exercice 6

Pour les suites u_n suivantes, trouver une suite v_n la plus simple possible telle que $u_n = \Theta(v_n)$.

1. $u_n = 2^{35} + 5n^8 + 2076n^7$
   2^{35} est une constante donc un $O(n^8)$.
   $5n^8$ est un $O(n^8)$.
   $2076n^7$ est un $O(n^8)$ (comparaison entre polynomes).
   Donc $u_n = O(n^8)$. De plus pour tout n, $u_n \geq n^8$. Donc $u_n = \Theta(n^8)$.
2. $u_n = 1024n + 2048\sqrt{5n}$
   On a $1024n = O(n)$.
   Montrons $\sqrt{n} = O(n)$. Pour $n \neq 0$ les suites ne s'annulent pas, donc on peut étudier $\sqrt{n}/n = 1/\sqrt{n} \leq 1$. Le quotient des deux suites est majoré, donc $\sqrt{n} = O(n)$.
   Ainsi $u_n = O(n)$. De plus pour tout n, $u_n \geq n$. Donc $u_n = \Theta(n)$.
3. $n^2/3 + 10^{100}n$
   C'est un $\Theta(n^2)$. la justification est la même qu'à la question 1.
4. $4\ln(5n) + \sqrt{3n}$
   On sait que $\forall x > 0, \ln(x) < x$.
   Donc $4\ln(5n) = 8\ln(5\sqrt{n}) < 8 * 5\sqrt{n}$.
   On conclut que pour tout $n > 0$, $u_n \leq (8 * 5 + \sqrt{3})\sqrt{n}$, donc $u_n = O(\sqrt{n})(n \to \infty)$.
   De plus pour tout n $u_n \geq \sqrt{n}$ car $\sqrt{3} \geq 1$.
   Donc $u_n = \Theta(\sqrt{n})(n \to \infty)$.
5. $u_n = 2n \sum_{i=1}^{2n}(3i + 1) + \sum_{p=1}^{n} 2^p$

   On développe un peu : $u_n = 6n \sum_{i=1}^{2n} i + 3n * 2n + 2^{n+1} - 1 = 6n2n(2n + 1)/2 + 3n * 2n + 2^{n+1} - 1$

   Le premier terme est un $\Theta(n^3)$, le deuxième un $\Theta(n^2)$, le troisième est un $\Theta(2^n)$ et le dernier un $\Theta(1)$.
   Le troisième terme domine tous les autres, c'est à dire que tous les termes sont en fait un $O(2^n)$. Il est donc immédiat que $u_n = O(2^n)$.
   De plus $u_n \geq 2^{n+1}$ pour tout n, donc $u_n = \Theta(2^n)$