TD3 : Ordres de grandeur et complexités simples.

Amar AHMANE.
MP2I

1 – Classement

Étant donné les conditions données sur les classes, on peut en former 7 avc les fonctions données. Tout est résumé dans ce joli petit tableau :

Θ_1	Θ_2	Θ_3	Θ_4	Θ_5	Θ_6	Θ_7
$n \mapsto \log n$	$n \mapsto \sqrt{n}$	$n \mapsto n^2 + \sqrt{n}$	$n \mapsto n^2\sqrt{n}$	$n \mapsto 2n^2 + n^3$	$n \mapsto 2^n$	$n \mapsto 3^n$
$n \mapsto \log n^2$	–	$n \mapsto 3n^2 - 5n$	–	–	$n \mapsto 2^{n+2}$	–
$n \mapsto \log 2n$	–	–	–	–	–	–

2 – Min et Max

Soient $f, g \in (\mathbb{R}_+^*)^{\mathbb{N}}$.

Question 1 Montrons que $f = \Theta(g) \Leftrightarrow g = \Theta(f)$.

$\boxed{\Rightarrow}$ Supposons que $f = \Theta(g)$. Il existe alors $c, d \in \mathbb{R}_+^*$ et $n_0 \in \mathbb{N}$ tels que

$$\forall n \in \mathbb{N}, \quad n \geq n_0 \Longrightarrow dg(n) \leq f(n) \leq cg(n)$$

Soit $n \in \mathbb{N}$. Supposons que $n \geq n_0$. D'une part

$$f(n) \leq cg(n) \Leftrightarrow \frac{1}{c} f(n) \leq g(n)$$

D'autre part

$$dg(n) \leq f(n) \Leftrightarrow g(n) \leq \frac{1}{d} f(n)$$

Finalement,

$$\frac{1}{c} f(n) \leq g(n) \leq \frac{1}{d} f(n)$$

En posant $(c', d') = \left(\frac{1}{d}, \frac{1}{c}\right)$, on a

$$\boxed{d' f(n) \leq g(n) \leq c' f(n)}$$

$\boxed{\Leftarrow}$ Le problème est symétrique.

Question 2 Montrons que $\max(f, g) = \Theta(f + g)$.
Soit $n \in \mathbb{N}$. On a, puisque on a toujours soit $\max(f, g) = f$ soit $\max(f, g) = g$, et comme f et g sont à valeurs dans $\mathbb{R}_+^*$:

$$\max(f, g)(n) \leq f(n) + g(n)$$

Mais aussi, comme

$$f(n) \leq \max(f, g)(n) \quad \text{et} g(n) \leq \max(f, g)(n)$$

Il en découle que

$$f(n)+g(n) \leq 2\max(f,g) \Leftrightarrow \frac{f(n)+g(n)}{2} \leq \max(f,g)(n)$$

Donc, finalement,

$$\frac{f(n)+g(n)}{2} \leq \max(f,g)(n) \leq f(n)+g(n)$$

Question 3 Tout ce que l'on peut dire de $\min(f,g)$ est que

$$\min(f,g) = \mathcal{O}(f) \text{ (ou } \mathcal{O}(g) \text{ ou } \mathcal{O}(f+g)\text{)}$$

3 – Boucles

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        for(int k = 0; k < n; k++){
            // cout constant
        }
    }
}
```

En supposant que l'on fasse p opérations de coût constant après être entré dans la troisième boucle, on fera ainsi le nombre d'opérations suivant :

$$\begin{aligned}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} np \\ &= \sum_{i=0}^{n-1} n \times np \\ &= n \times n^2 = n^3 p\end{aligned}$$

Ainsi, en notant $T_1(n)$ le nombre d'opérations total de coût constant effectuées à la fin du programme, et T_1 la fonction qui à n associe ce nombre on a que $T_1 = \mathcal{O}(n^3)$.

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < i; j++){
    }
}
```

Pour ce second programme, on supposera encore que l'on est en train d'effectuer p opérations de coût constant après entre entré dans la seconde boucle. On fera ainsi le nombre d'opérations suivant :

$$\begin{aligned}\sum_{i=0}^{n-1}\sum_{j=0}^{i-1} p &= \sum_{i}^{n-1} ip \\ &= p\frac{n(n-1)}{2}\end{aligned}$$