Exercices : solutions de la série #1

Solution à l'exercice 1.

a. $an^2 + bn + c \in O(n^2)$

Puisque $a, b, c \geq 0$, alors pour $n \geq 1$,

$$\begin{aligned} an^2 + bn + c &\leq an^2 + bn^2 + cn^2 \\ &= (a + b + c)n^2 \end{aligned}$$

Donc $N = 1$ et $c = (a + b + c)$ pour la définition de O.

b. $(1 + 2 + \ldots + n) \in O(n^2)$

Pour $n \geq 1$,

$$\begin{aligned} (1 + 2 + \ldots + n) &= n(n + 1)/2 \\ &= n^2/2 + n/2 \\ &\leq n^2/2 + n^2/2 \\ &= n^2 \end{aligned}$$

Donc, $N = 1$ et $c = 1$ dans la définition de O .

c. $(1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n}) \in \Omega(1)$

Pour $n \geq 1$,

$$\begin{aligned} (1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n}) &= \frac{1 - \frac{1}{2}^{n+1}}{1 - \frac{1}{2}} \\ &= \frac{1 - \frac{1}{2}^{n+1}}{\frac{1}{2}} \\ &= 2 - (\frac{1}{2})^n \\ &\geq 1 \end{aligned}$$

Donc, $N = 1$ et $c = 1$ dans la définition de Ω.

Solution à l'exercice 2.

a. $\lg\ (n \times 2^n) \in O(n)$

En appliquant les équivalences suivantes, puis les propriétés sur les sommes et les produits (ou les propriétés 6 et 7 du manuel) :

$$\begin{aligned} \lg\ (n \times 2^n) &= \lg\ n + \lg 2^n \\ &= \lg\ n + n \times \lg\ 2 \\ &\in O(\lg\ n) + O(n) \times O(1) \\ &= O(n) \end{aligned}$$

b. $n \times \lg\ (2^{n^2})$

$$\begin{aligned} n \times \lg(2^{n^2}) &= n \times n^2 \times lg\ 2 \\ &\in O(n^3) \times O(1) \\ &= O(n^3) \end{aligned}$$

c. $\lg\ (n+4)^2$

Pour $n \geq 4$,

$$\begin{aligned}
\lg\ (n+4)^2 &= 2 \times \lg\ (n+4) \\
&\leq 2 \times \lg\ (n+n) \\
&= 2 \times \lg\ (2n) \\
&= 2 \times (\lg\ 2 + \lg\ n) \\
&\in O(1) \times (O(1) + O(\lg\ n)) \\
&\in O(1) \times O(\lg\ n) \\
&= O(lg\ n)
\end{aligned}$$

d. $2^{1000} + 2n^2$

$$\begin{aligned}
2^{1000} + 2n^2 &\in O(1) + O(1) \times O(n^2) \\
&= O(n^2)
\end{aligned}$$

e. $2^{(3-n)}$

Pour $n \geq 3$,

$$\begin{aligned}
2^{(3-n)} &\leq 1 \\
&\in O(1)
\end{aligned}$$

Solution à l'exercice 3.

On peut arriver au résultat désiré de deux façons différentes :

a. On choisit une opération *barométrique*. Cette opération doit être telle que le nombre total des autres opérations sera dominé asymptotiquement par le nombre de ces opérations barométriques. Ici, une telle opération barométrique serait la comparaison "`ai == aj`" : on peut voir que toute exécution de celle-ci dominera l'exécution de n'importe quelle autre, c'est-à-dire, que cette opération sera exécutée plus souvent que n'importe quelle autre.

   Notons tout d'abord qu'en termes de cette opération, il s'agit ici d'une analyse de tous les cas, puisque le nombre de comparaisons ne dépendra pas des données reçues. On peut donc produire directement un estimé Θ.

   Puisque la boucle externe comporte exactement n itérations et que chaque exécution de la boucle interne comporte à son tour exactement n itérations, on en conclut donc que l'algorithme entraine l'exécution de n^2 opérations élémentaires. On peut donc en conclure que l'algorithme est $\Theta(n^2)$.

b. Une autre façon d'arriver à un résultat équivalent est de déterminer l'ordre de grandeur du nombre total d'opérations pour chaque partie de l'algorithme, puis de simplifier en utilisant les propriétés de la somme et du produit de fonctions de complexité. Notons que même avec cette méthode, il s'agit quand même d'une analyse de tous les cas possibles, puisque l'exécution d'une instruction `SI` sera toujours $\Theta(1)$, que la condition soit vraie ou non (exécuter une branche `ALORS` en $\Theta(1)$ vs. ne pas exécuter cette branche aussi en $\Theta(1)$).

On obtient alors quelque chose qui ressemble à ce qui suit (pas facile à illustrer avec LaTeX, avec `Th` dénotant Θ `;(` :

```
DEBUT
  nbOccurencesMode <- 0                                          Th(1)
  POUR i <- 1 A n FAIRE                                                |
    nbOccurences <- 0                                            Th(1) |
    POUR j <- 1 A n FAIRE                                        |     n itérations
      SI ai == aj ALORS                              |           Th(n) |
        nbOccurences <- nbOcccurences + 1            Th(1)       |     |
      FIN                                            |           |     |
    FIN
    SI nbOccurences > nbOccurencesMode ALORS         |
      nbOccurencesMode <- nbOccurences               Th(1)
      mode <- ai                                     |
    FIN                                              |
  FIN
FIN
                                                                 ----------
                                                                 Th(n)
```

En mots :

- L'instruction `SI` de la boucle interne est $\Theta(1)$. Cette instruction est exécutée exactement n fois, donc $\Theta(n)$ pour la boucle interne au complet.
- Dans la boucle externe, l'instruction d'affectation initiale est $\Theta(1)$, l'instruction `SI` à la fin de la boucle externe est $\Theta(1)$, alors que la boucle interne est $\Theta(n)$. Le corps de la boucle externe est donc $\Theta(n)$. Cette boucle s'exécute n fois. Au total, pour la boucle externe, on a donc $\Theta(n^2)$ opérations.
- La première instruction d'affectation est $\Theta(1)$. La boucle externe est $\Theta(n^2)$. Au total, on a donc $\Theta(n^2)$ opérations.

L'algorithme est donc $\Theta(n^2)$.

Solution à l'exercice 4.

a. Dans les deux cas, la complexité sera $\Theta(log_{10}\ n) = \Theta(\lg\ n)$.

- Dans le cas de la version itérative, l'analyse est semblable à d'autres analyses que nous avons déjà faites. On peut utiliser deux stratégies :
  (a) Avec une opération barométrique : Ici, la comparaison "`n > 0` serait une opération appropriée, puisqu'elle sera exécutée au moins aussi souvent que n'importe quelle autre instruction. La question clé est alors de déterminer le nombre de fois où la boucle `TANTQUE` sera exécutée. Pour ce genre d'analyse (on verra plus loin un théorème qui justifie une telle façon de procéder), on peut supposer que `n` est d'une forme appropriée facilitant l'analyse. Dans ce cas, puiqu'à chaque itération la valeur de `n` est divisée par 10, supposons donc que $\texttt{n} = 10^k$ pour un certain $k \geq 0$. Dans ce cas, la valeur de `n` évoluera comme suit :

Numéro de l'itération	Valeur de `n` au début de l'itération	Valeur de `n` à la fin de l'itération
1	10^k	10^{k-1}
2	10^{k-1}	10^{k-2}
...	...	...
i	10^{k-i+1}	10^{k-i}
...	...	...
k	10^1	10^0
$k+1$	10^0	0

La boucle `TANTQUE` s'exécutera donc de $\Theta(k)$ fois, c'est-à-dire, $\Theta(log_{10}\ n)$. Le nombre total d'opération barométriques sera donc lui aussi $\Theta(log_{10}\ n)$. Or, l'une des propriétés vues au chapitre 1 nous indique que le logarithme utilisé n'a pas d'importance pour définir l'ordre de complexité. On peut donc plus simplement conclure que l'algorithme dans son ensemble est $\Theta(\lg\ n)$.

(b) En comptant toutes les opérations élémentaires. Le même raisonnement que dans le cas précédent s'applique pour déterminer le nombre d'itérations de la boucle.

```
DEBUT
  FONCTION sc( n: Natural ): Natural
  DEBUT
    somme <- 0                                          Th(1)
    TANTQUE n > 0 FAIRE                                 |
      somme <- somme + (n MOD 10)  Th(1) | Th(1)        | Th(lg n)
      n <- n / 10                  Th(1) |              |
    FIN                                                 |
    RETOURNER somme                                     Th(1)
  FIN
FIN

                                                       ----------
                                                        Th(lg n)
```

- Dans le cas de la version récursive, si on choisit la comparaison `n == 0` comme opération barométrique, on obtient lest équations de récurrence suivantes :
  - $T(n) = T(n/10) + 1$
  - $T(0) = 1$

  En utilisant le théorème général avec $a = 1$, $b = 10$ et $k = 0$, la solution est $\Theta(n^0\ \lg\ n) = \Theta(\lg\ n)$ (puisque $1 = 10^0$).

  On peut aussi obtenir la solution désirée par substitution répétée (en supposant que $n = 10^k$ pour un $k \geq 0$) :

$$\begin{aligned}
T(n) &= T(n/10) + 1 \\
&= [T(n/100) + 1] + 1 = T(n/10^2) + 2 \\
&= [T(n/10^3) + 1] + 2 = T(n/10^3) + 3 \\
&= \ldots \\
&= T(n/10^k) + k = T(1) + k \\
&= [T(1/10) + 1] + k = T(0) + k + 1 \\
&= 1 + k + 1 \\
&= k + 2 \\
&\in \Theta(\lg_{10}\ n) \\
&\in \Theta(\lg\ n)
\end{aligned}$$

b. L'algorithme itératif serait probablement plus rapide. Dans la plupart des langages, un appel de procédure est plus coûteux qu'une simple itération d'une boucle. Notons toutefois que dans certains langages fonctionnels, le compilateur est capable de reconnaitre que certaines formes d'appels récursifs sont en fait des boucles et le compilateur peut effectivement transformer ces appels en boucle, donc sans avoir besoin d'allouer un bloc d'activation de fonction à chaque appel. Ceci peut se faire en reconnaissant l'utilisation d'un patron d'accumulation et en éliminant ensuite les appels récursifs de queue (*tail calls*).

   La fonction transformée serait la suivante :

```
FONCTION sc_r( n: Natural ): Natural
  RETOURNER sc_r_acc(n, 0)
FIN

FONCTION sc_r_acc( n: Natural, acc: Natural ): Natural
DEBUT
  SI n == 0 ALORS
    RETOURNER acc
  SINON
    RETOURNER sc_r_acc( n MOD 10, acc + n / 10 )
FIN
```

   Un appel récursif de `sc_r_acc` est toujours la dernière opération exécutée. Un tel appel peut donc réutiliser le bloc d'activation de la fonction, ce qui en termes de surcoûts d'exécution correspond à exécuter une boucle.