

IUT	Robert Schuman	
	Institut universitaire de technologie	
	Département informatique	
	Université de Strasbourg	

P4Z : Exprimer la complexité

2019/20

1 Notations O , o , θ

Vos connaissances solides en terme de limites à l'infini vous permettent de savoir que :

$$\lim_{x \rightarrow +\infty} \frac{e^x}{x^2} = +\infty;$$

$$\lim_{x \rightarrow +\infty} \ln(x) - x = -\infty;$$

$$\lim_{x \rightarrow +\infty} \frac{2x^2 - x^3}{3x^3 - x} = -\frac{1}{3}.$$

Se cache derrière la notion de comparaison asymptotique (*i.e.* à l'infini) et de savoir qui converge plus rapidement vers l'infini.

Les mathématiciens ont des notations pour décrire différentes comparaisons asymptotiques et les informaticiens peuvent être amenés à les utiliser si jamais ils leur vient à l'idée de faire des calculs de complexité d'algorithme.

Définition 1.1

Soient f une fonctions réelle pour laquelle la limite en $+\infty$ a un sens.

On définit les ensembles suivants :

- 1) $o(f) = \{g \text{ fonction réelle} \mid \exists x_0 > 0 \text{ t.q. } g(x) = \epsilon(x) \cdot f(x) \text{ avec } \lim_{x \rightarrow +\infty} \epsilon(x) = 0\};$
- 2) $O(f) = \{g \text{ fonction réelle} \mid \exists x_0, k > 0 \text{ t.q. } |g(x)| \leq k \cdot |f(x)|$

<p style="text-align: center;">pour $x \geq x_0$};</p> <p>3) $\Theta(f) = \{g \text{ fonction réelle} \mid \exists x_0, k_1, k_2 > 0 \text{ t.q.}$ $k_1 \cdot f(x) \leq g(x) \leq k_2 \cdot f(x) \text{ pour } x \geq x_0\}$;</p> <p>4) $\Omega(f) = \{g \text{ fonction réelle} \mid f \in O(g)\}$;</p> <p>5) $\omega(f) = \{g \text{ fonction réelle} \mid f \in o(g)\}$.</p>
--

Remarques :

- 1) Il y a une autre définition de $O(f)$. On a donné celle utilisée en théorie des algorithmes (Knuth).
- 2) $g(x) = \epsilon(x) \cdot f(x)$ est équivalent à $\frac{g(x)}{f(x)} = \epsilon(x)$ dès lors que $f(x) \neq 0$.
 $g \in o(f)$ est donc équivalent à $\lim_{x \rightarrow +\infty} \frac{g(x)}{f(x)} = 0$ dans ce cas-là.
- 3) $f \in o(g) \iff g \in \omega(f)$ et $f \in O(g) \iff g \in \Omega(f)$.
- 4) $f = \Theta(g) \iff f \in O(g)$ et $g \in O(f)$.

Notation :

Ne reculant devant aucune ignominie, l'usage a toléré la notation suivante :

$$\ll f = O(g) \gg \text{ ou } \ll f(x) = O(g(x)) \gg \text{ au lieu de } f \in O(g).$$

Et de même pour les autres notations... Que dire devant tant de bassesse...

- 1) Comme $\lim_{x \rightarrow +\infty} \frac{x^n}{e^x} = 0$, on peut dire que $x^n = o(e^x)$ (ou $e^x = \omega(x^n)$);
- 2) Soit $k \in \mathbb{R}^+$. Comme $x^n = o(e^x)$, il existe x_k tel que $\frac{x^n}{e^x} < k \iff x^n = k \cdot e^x$ pour $x > x_k$. Par conséquent, on a aussi $x^n = O(e^x)$.
- 3) En général, on peut dire que si $f = o(g)$ alors $f = O(g)$.

On peut se rappeler des différentes notations de comparaisons en faisant les analogies suivantes :

$f = o(g)$	(croissance asymptotique de) $f < g$
$f = O(g)$	(croissance asymptotique de) $f \leq g$
$f = \Theta(g)$	(croissance asymptotique de) $f = g$
$f = \Omega(g)$	(croissance asymptotique de) $f \geq g$
$f = \omega(g)$	(croissance asymptotique de) $f > g$

2 Lien avec la complexité algorithmique

Pour estimer la performance d'un algorithme, de manière théorique, on ne cherche pas à déterminer exactement le temps de calcul (par exemple) car

cela dépend de beaucoup de paramètres extérieurs à l'algorithme (puissance de la machine, langage utilisé, compilation...) et surtout des données à traiter.

Se donner une mesure de la complexité c'est se donner un comportement asymptotique de la durée du traitement en fonction de la taille des données à traiter.

Dire qu'un algorithme est $\theta(10^n)$ veut dire la chose suivante : lorsque la taille passe d'un ordre $n = 10$ à $n = 12$ le temps de calcul est multiplié par 100 ! C'est mal !

A l'inverse, si l'algorithme est $\theta(\log(n))$, la taille peu doubler et cela ne rajoute qu'un temps constant de calcul à chaque fois : c'est bien !

Plus concrètement, les algorithmes de tris vus précédemment sont entre $O(n \log(n))$ et $O(n^2)$.

Il y a un autre écueil : les données elles-mêmes ont une influence sur l'algorithme. On peut faire exploser le temps de calcul d'un des tris de la feuille précédente en choisissant bien le tableau en entrée. On peut donc être amené à calculer la complexité d'un algorithme en temps moyen ou au cas le plus mauvais.

3 Un résultat pour les algorithmes récursifs

Notons $T(n)$ la complexité d'un algorithme récursif vérifiant :

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^c)$$

avec $a \geq 1$ et $b > 1$.

Alors :

$$T(n) = \begin{cases} O(n^{\log_b(a)}) & \text{si } c < \log_b(a) \\ O(n^c \log(n)) & \text{si } c = \log_b(a) \\ O(n^c) & \text{si } c > \log_b(a) \end{cases}$$

4 Exercices

Exercice 1

Supposons qu'on ait un ordinateur effectuant 1000 opérations significatives par seconde (c'est très peu, vous vous êtes visiblement fait arnaquer, il faut aller au sav !). Si n est la taille des données, remplir le tableau ci-dessous donnant l'ordre de grandeur du temps de calcul suivant la complexité :

n	10	100	1000	10 000	100 000	1 000 000
$\log n$						
n	0,01s	0,1s	1s			
$n \log(n)$						
$n\sqrt{n}$						
n^2						
n^3						
2^n						
$n!$						

Exercice 2

1) Donner la complexité des algorithmes suivants :

Hellos World(n) :

```
pour  $i = 1$  à  $n$  faire
  pour  $j = 1$  à  $n$  faire
    Afficher "Hello World"
  fin pour
fin pour
```

Hellos World2(n) :

```
pour  $i = 1$  à  $n$  faire
  pour  $j = i$  à  $n$  faire
    Afficher "Hello World"
  fin pour
fin pour
```

Hellos World3(n) :

```
si  $n > 1$  alors
  Hellos World3( $n/2$ )
  Hellos World3( $n - n/2$ )
fin si
Afficher "Hello World"
```

2) Quels sont les complexités des algorithmes de recherche d'éléments dans un tableau ?

3) Quels sont les complexités des algorithmes de tris ?