

# Análisis de complejidad de algoritmos II

Esteban Feuerstein<sup>1</sup>

Ricardo Rodríguez<sup>1</sup>    Fernando Schapachnik<sup>1</sup>

<sup>1</sup>Departamento de Computación, FCEyN,  
Universidad de Buenos Aires, Buenos Aires, Argentina

Algoritmos y Estructuras de Datos II,  
segundo cuatrimestre de 2018

## (2) En la última clase...

- Argumentamos a favor de un análisis teórico (en lugar de empírico) como método para comparar algoritmos en base a su tiempo de ejecución.
  - El método era asintótico.
  - Y se basaba en el tamaño de la entrada.
- Acordamos utilizar como marco de referencia una máquina ideal y dimos un cálculo para establecer el costo de ejecutar cada instrucción.
- Como nos interesaba hablar de la complejidad de **clases** de instancias, pasamos a una notación que expresa la complejidad en base al tamaño de la entrada:  $T(n)$ .
- Vimos que debíamos diferenciar al menos los casos promedio, mejor y peor.

### (3) Hoy

- Veremos una forma de aproximar  $T(n)$ : la notación " $O(n)$ ".
- Miraremos más en detalle qué significa *tamaño de la entrada*.
- Aportaremos más elementos para comparar la eficiencia de los algoritmos.

## (4) Análisis asintótico

- Habíamos dicho que íbamos a hacer un análisis asintótico porque cuando los datos a procesar eran pocos, no importaba demasiado cómo procesarlos.
- Sin embargo, a medida que van creciendo, distintos algoritmos pueden diferir en **órdenes de magnitud** en el tiempo que tardan en resolver el mismo problema.
  - ¿Qué es un *orden de magnitud*? Dos valores  $a$  y  $b$  difieren en  $n$  órdenes de magnitud cuando  $a/b \sim 10^n$ .
- Por eso, vamos a analizar cuál es el orden de magnitud, aproximadamente, del tiempo de ejecución de los algoritmos.

## (5) Análisis asintótico (cont.)

¿Por qué es importante que los algoritmos sean asintóticamente eficientes?

n Compl	10	20	30	40	50	60
$n$	0,00001"	0,00002"	0,00003"	0,00004"	0,00005"	0,00006"
$n^2$	0,0001"	0,0004"	0,0009"	0,0016"	0,0025"	0,0036"
$n^3$	0,001"	0,008"	0,027"	0,064"	0,125"	0,216"
$n^5$	0,1"	3,2"	24,3"	1,7'	5,2'	13,0'
$2^n$	0,001"	1,0"	17,9'	12,7 días	35,7 años	366 siglos
$3^n$	0,059"	58,0'	65 años	3855 siglos	$2 \times 10^8$ siglos	$1,3 \times 10^{13}$ siglos

Fuente: Aho, Hopcroft, Ullman

## (6) Análisis asintótico (cont.)

¿Y si le meto un procesador Intel Core ix con  $x \rightarrow \infty$ ?


Máximo tamaño de problema resoluble en una hora.

Compl.	Actual	$\times 100$	$\times 1000$
$n$	$N_1$	$100N_1$	$1000N_1$
$n^2$	$N_2$	$10N_2$	$31,6N_2$
$n^3$	$N_3$	$4,64N_3$	$10N_3$
$n^5$	$N_4$	$2,5N_4$	$3,98N_4$
$2^n$	$N_5$	$N_5 + 6,64$	$N_5 + 9,97$
$3^n$	$N_6$	$N_6 + 4,19$	$N_6 + 6,29$

Fuente: Aho, Hopcroft, Ullman

## (7) Análisis asintótico (cont.)


- **Comportamiento asintótico**: comportamiento para valores de la entrada suficientemente grandes.
- Ya sabíamos que dado un algoritmo podíamos (en teoría) calcular  $T(n)$ .

 El **orden** de  $T(n)$  expresa el comportamiento asintótico.


- Expresiones comunes son: orden logarítmico, lineal, cuadrático, exponencial, etc.

## (8) Cotas

- Vamos a presentar tres cotas para comportamiento asintótico.

 Esto es fundamental. El objetivo del estudio de la complejidad algorítmica es poder establecer estas cotas.

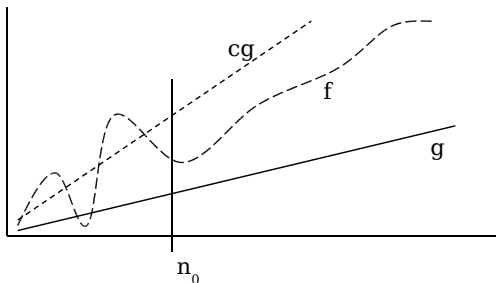
- Tenemos:
  - $O(T(n))$  (O grande), cota superior.
  - $\Omega(T(n))$  (omega), cota inferior.
  - $\Theta(T(n))$  (theta), orden exacto.
- Idea intuitiva: si para  $T(n)$  puedo presentar
  - $O(T(n))$ , “sé” cuánto va a tardar como máximo.
  - $\Omega(T(n))$ , “sé” cuánto va a tardar como mínimo.
  - $\Theta(T(n))$ , “sé” ambas cosas.

 Las comillas están porque lo que tenemos es una aproximación (se verá con claridad cuando presentemos la definición).



## (9) O grande

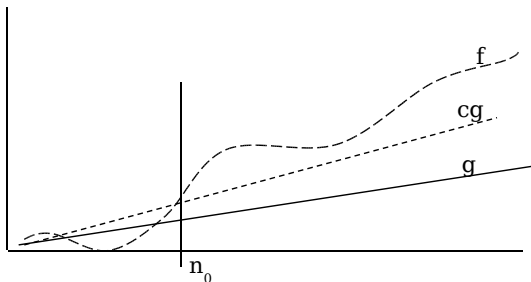
- Definición formal  $O(g) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists n_0 > 0, c > 0 \text{ tales que } (\forall n \geq n_0) f(n) \leq cg(n)\}$ .



- Idea:  $O(g)$  define el conjunto de funciones que, a partir de cierto valor  $n_0$ , tienen a  $g$  multiplicado por un constante  $c$  como **cota superior**.
- Vamos a decir que  $T(n) \in O(g(n))$ . Abusando la notación, también diremos que  $T(n) = O(g(n))$ .
- Por ejemplo, “ $T(n)$  está en  $O(n^2)$ ” o “el algoritmo tiene  $O(\log(n))$ ”.

## (10) $\Omega$

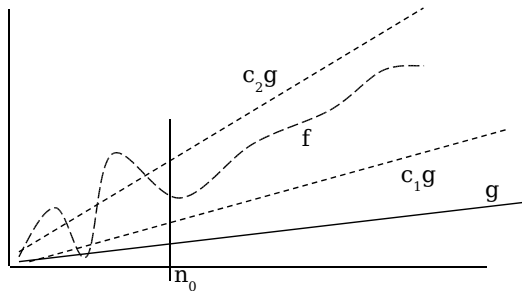
- Definición formal  $\Omega(g) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists n_0 > 0, c > 0 \text{ tales que } (\forall n \geq n_0) f(n) \geq cg(n)\}$ .



- Idea:  $\Omega(g)$  define el conjunto de funciones que, a partir de cierto valor  $n_0$ , tienen a  $g$  multiplicado por un constante  $c$  como **cota inferior**.

# (11) $\Theta$

- Definición formal  $\Theta(g) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists n_0 > 0, c_1 > 0, c_2 > 0 \text{ tales que } (\forall n \geq n_0) \ c_1 g(n) \leq f(n) \leq c_2 g(n)\}$ .



- Idea:  $\Theta(g)$  define el conjunto de funciones que, a partir de cierto valor  $n_0$ , quedan “ensandwichadas” por  $g$  multiplicado respectivamente por dos constantes  $c_1$  y  $c_2$ .
- Además, se cumple que  $\Theta(f) = O(f) \cap \Omega(f)$ .

## (12) Características

- Notemos que las tres definen conjuntos de funciones matemáticas, y podrían utilizarse independientemente de la complejidad algorítmica.
- Interpretación:
  - $f \in O(g)$  significa que  $f$  crece, a lo sumo, tanto como  $g$ .
  - $f \in \Omega(g)$  significa que  $f$  crece por lo menos como  $g$ .
  - $f \in \Theta(g)$  significa que  $f$  crece a la misma velocidad que  $g$ .
  - En todos los casos, a partir de cierto momento ( $n_0$ ).
- Veamos ejemplos:
  - $100n^2 + 300n + 10e^{20} \in O(n^2) \subset O(n^3)$
  - $100n^2 + 300n + 10e^{20} \in \Omega(n^2) \subset \Omega(n)$
  - $100n^2 + 300n + 10e^{20} \in \Theta(n^2)$

## (13) Propiedades de $O$

- Toda  $f$  cumple  $f \in O(f)$ .
- $f \in O(g) \Rightarrow O(f) \subseteq O(g)$
- $O(f) = O(g) \iff f \in O(g) \wedge g \in O(f)$
- $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$

⚠  $f \in O(g) \Rightarrow c.f \in O(g)$  (con  $c$  cte.)

⚠ Regla de la suma:

$$f_1 \in O(g) \wedge f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\max(g, h))$$

⚠ Regla del producto:

$$f_1 \in O(g) \wedge f_2 \in O(h) \Rightarrow f_1 \times f_2 \in O(g \times h)$$

⚠ Estas últimas reglas suelen denominarse **álgebra de órdenes**.

- ¿Qué significa eso para el análisis de la complejidad de programas?
- Si  $\lim_{n \rightarrow \infty} f(n)/g(n) \in (0, \infty)$  entonces  $O(f) = O(g)$ .
- Si es 0, entonces  $f \in O(g)$ , es decir,  $O(f) \subset O(g)$  pero  $g \notin O(f)$ .

## (14) Propiedades de $\Omega$

- Toda  $f$  cumple  $f \in \Omega(f)$ .
- $f \in \Omega(g) \Rightarrow \Omega(f) \subseteq \Omega(g)$
- $\Omega(f) = \Omega(g) \iff f \in \Omega(g) \wedge g \in \Omega(f)$
- $f \in \Omega(g) \wedge g \in \Omega(h) \Rightarrow f \in \Omega(h)$
- $f \in \Omega(g) \Rightarrow c.f \in \Omega(g)$  (con  $c$  cte.)
- Regla de la suma:  
 $f_1 \in \Omega(g) \wedge f_2 \in \Omega(h) \Rightarrow f_1 + f_2 \in \Omega(g + h)$
- Regla del producto:  
 $f_1 \in \Omega(g) \wedge f_2 \in \Omega(h) \Rightarrow f_1 \times f_2 \in \Omega(g \times h)$
- Si  $\lim_{n \rightarrow \infty} f(n)/g(n) \in (0, \infty)$  entonces  $\Omega(f) = \Omega(g)$ .
- Si es 0, entonces  $g \in \Omega(f)$ , es decir,  $\Omega(g) \subset \Omega(f)$  pero  $g \notin \Omega(f)$ .

## (15) Propiedades de $\Theta$

- Toda  $f$  cumple  $f \in \Theta(f)$ .
- $f \in \Theta(g) \Rightarrow \Theta(f) \subseteq \Theta(g)$
- $\Theta(f) = \Theta(g) \iff f \in \Theta(g) \wedge g \in \Theta(f)$
- $f \in \Theta(g) \wedge g \in \Theta(h) \Rightarrow f \in \Theta(h)$
- $f \in \Theta(g) \Rightarrow c.f \in \Theta(g)$  (con  $c$  cte.)
- Regla de la suma:  
 $f_1 \in \Theta(g) \wedge f_2 \in \Theta(h) \Rightarrow f_1 + f_2 \in \Theta(\max(g, h))$
- Regla del producto:  
 $f_1 \in \Theta(g) \wedge f_2 \in \Theta(h) \Rightarrow f_1 \times f_2 \in \Theta(g \times h)$
- Si  $\lim_{n \rightarrow \infty} f(n)/g(n) \in (0, \infty)$  entonces  $\Theta(f) = \Theta(g)$ .
- Si es 0, entonces  $\Theta(g) \neq \Theta(f)$ .

## (16) Familias interesantes

- $O(1)$ : complejidad constante.
- $O(\log(n))$ : complejidad logarítmica.
- Recordemos:  $\log_b(n) = \log_2(n)/\log_2(b)$ . I.e.,  $\log_b(n) = c \cdot \log_2(n)$ .
- $O(n)$ : complejidad lineal (y cerquita:  $O(n \log(n))$ ).
- $O(n^2)$ : complejidad cuadrática.
- $O(n^3)$ : complejidad cúbica.
- $O(n^k)$ ,  $k \geq 1$ : complejidad polinomial.
- $O(2^n)$ : complejidad exponencial.
- Recordemos que  $1 \prec \log(n) \prec n^k (k \geq 1) \prec k^n \prec n! \prec n^n$  (donde  $f \prec g$  significa que la función  $g$  crece más rápido que  $f$ ).




## (17) ¿Qué pasó con $T(n)$ ?


- ¿Cómo se relaciona todo esto con la complejidad de un algoritmo?
- Primero, recordemos lo que habíamos dicho.

## (18) Cotas

- Vamos a presentar tres cotas para comportamiento asintótico.

 Esto es fundamental. El objetivo del estudio de la complejidad algorítmica es poder establecer estas cotas.

- Tenemos:
  - $O(T(n))$  (O grande), cota superior.
  - $\Omega(T(n))$  (omega), cota inferior.
  - $\Theta(T(n))$  (theta), orden exacto.
- Idea intuitiva: si para  $T(n)$  puedo presentar
  - $O(T(n))$ , “sé” cuánto va a tardar como máximo.
  - $\Omega(T(n))$ , “sé” cuánto va a tardar como mínimo.
  - $\Theta(T(n))$ , “sé” ambas cosas.

 Las comillas están porque lo que tenemos es una aproximación (se verá con claridad cuando presentemos la definición).

## (19) ¿Qué pasó con $T(n)$ ? (cont.)

- Si para un algoritmo sabemos que  $T_{peor} \in O(g)$ , podemos asegurar que para entradas de tamaño creciente, en **todos los casos** el tiempo será a lo sumo proporcional a  $g$ .
- Si lo que sabemos es  $T_{prom} \in O(g)$ , entonces, para entradas de tamaño creciente, **en promedio**, el tiempo será a lo sumo proporcional a  $g$ .
- ¿Y si lo que sabemos es que  $T_{peor} \in \Omega(g)$ ? Entonces, para entradas de tamaño creciente, en **el peor caso**, el tiempo va a ser proporcional a  $g$ .
- Se suele decir “el problema  $X$  está en  $\Omega(g)$ ”, lo que significa que cualquier algoritmo que lo resuelva cumple con  $T_{peor} \in \Omega(g)$ .

## (20) Tamaño de la entrada

- ¿Cuál es la complejidad de multiplicar dos enteros?
- Depende de cuál sea la medida del tamaño de la entrada.
- Podría considerarse que todos los enteros tienen tamaño  $O(1)$ , pero eso no sería útil para comparar este tipo de algoritmos.
- En este caso, conviene pensar que la medida es el logaritmo del número.
- Si por el contrario estuviésemos analizando algoritmos que ordenan arreglos de enteros, lo que importa no son los enteros en sí, sino cuántos tengamos.
- Entonces, para ese problema, la medida va a decir que todos los enteros *miden* lo mismo.


## (21) Algunas observaciones

- La utilización de las cotas asintóticas para comparar funciones de tiempo de ejecución se basa en la hipótesis de que son suficientes para decidir el mejor algoritmo, prescindiendo de las constantes de proporcionalidad. Sin embargo, esta hipótesis puede no ser cierta cuando el tamaño de la entrada es pequeño, o cuando las constantes involucradas son demasiado grandes, etc. Por ejemplo, nadie implementa el algoritmo de Strassen de multiplicación de enteros.
- Obtener buenas cotas inferiores es usualmente difícil, aunque en general existe una cota inferior trivial para cualquier algoritmo: al menos hay que leer los datos y luego escribirlos, de forma que ésa sería una primera cota inferior.


## (22) Conceptos importantes de la clase de hoy

- Importancia del análisis asintótico.

- Cotas:  $O$ ,  $\Omega$ ,  $\Theta$ .

 Propiedades de las cotas: álgebra de órdenes.

- Relación entre las cotas y  $T(n)$ .

 Tamaño de la entrada.

## (23) Tarea

- Leer el tema de la bibliografía.
- Empezar con la práctica.
- ⚠ Aplicar las propiedades de  $O$  más las reglas para  $t(i)$  para calcular la complejidad de algunos algoritmos.
- Venir a la práctica con dudas.