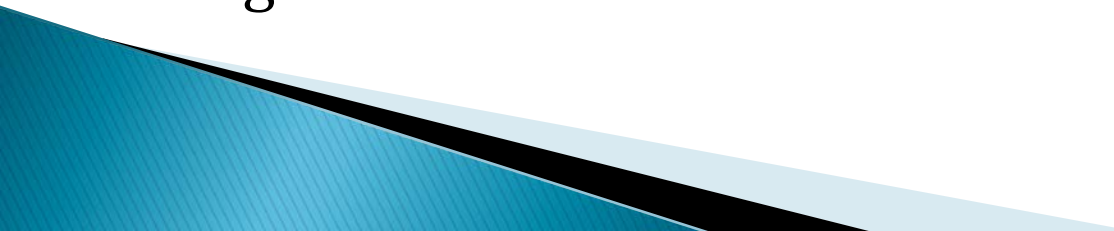


# Funciones de complejidad

# Complejidad temporal de un algoritmo

- ▶ Suele decirse que un algoritmo es bueno si para ejecutarlo se necesita poco tiempo y si requiere poco espacio de memoria. Sin embargo, por tradición, un factor más importante para determinar la eficacia de un algoritmo es el tiempo necesario para ejecutarlo.
  - ▶ En el análisis de algoritmos siempre se escogerá un tipo de operación particular que ocurra en el algoritmo, y se realizará un análisis matemático a fin de determinar el número de operaciones necesarias para completar el algoritmo.
- 

Suponga que para ejecutar un algoritmo se requieren  $(n^3 + n)$  pasos. Se diría a menudo que la complejidad temporal de este algoritmo es del orden de  $n^3$ . Debido a que el término  $n^3$  es de orden superior a  $n$  y a medida que  $n$  se hace más grande, el término  $n$  pierde importancia en comparación con  $n^3$ . A continuación daremos un significado formal y preciso a esta informal afirmación.

### **Definición**

$f(n) = O(g(n))$  si y sólo si existen dos constantes positivas  $c$  y  $n_0$  tales que  $|f(n)| \leq c|g(n)|$  para toda  $n \geq n_0$ .

Por la definición anterior se entiende que, si  $f(n) = O(g(n))$ , entonces  $f(n)$  está acotada en cierto sentido por  $g(n)$  cuando  $n$  es muy grande. Si se afirma que la complejidad temporal de un algoritmo es  $O(g(n))$ , quiere decir que para ejecutar este algoritmo siempre se requiere de menos que  $c$  veces  $|g(n)|$  a medida que  $n$  es suficientemente grande para alguna  $c$ .

- ▶ La importancia del orden de magnitud puede verse al estudiar la siguiente tabla:

Funciones de complejidad temporal.

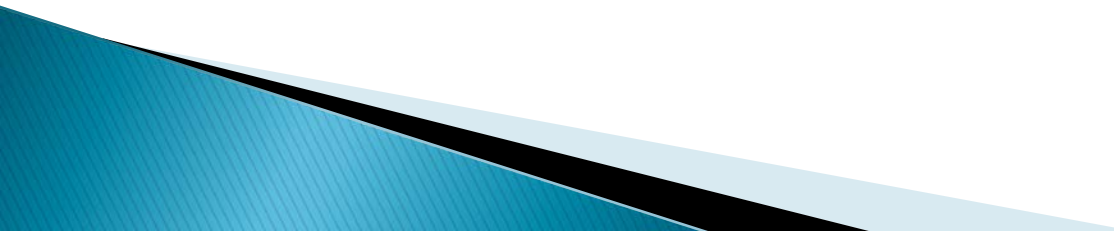
Función de complejidad temporal	Tamaño del problema ( $n$ )			
	10	$10^2$	$10^3$	$10^4$
$\log_2 n$	3.3	6.6	10	13.3
$n$	10	$10^2$	$10^3$	$10^4$
$n \log_2 n$	$0.33 \times 10^2$	$0.7 \times 10^3$	$10^4$	$1.3 \times 10^5$
$n^2$	$10^2$	$10^4$	$10^6$	$10^8$
$2^n$	1 024	$1.3 \times 10^{30}$	$>10^{100}$	$>10^{100}$
$n!$	$3 \times 10^6$	$>10^{100}$	$>10^{100}$	$>10^{100}$

- ▶ En el peor de los casos, una búsqueda secuencial a través de una lista de  $n$  números requiere  $O(n)$  operaciones.
- ▶ Si se tiene una lista ordenada de  $n$  números, es posible usar búsqueda binaria y la complejidad temporal se reduce a  $O(\log_2 n)$ , en el peor de los casos.

Para  $n = 10^4$ , la búsqueda secuencial puede requerir  $10^4$  operaciones, mientras que la búsqueda binaria sólo requiere 14 operaciones.

Aunque las funciones de complejidad temporal como  $n^2$ ,  $n^3$ , etc., pueden no ser deseables, siguen siendo tolerables en comparación con una función del tipo  $2^n$ . Por ejemplo, cuando  $n = 10^4$ , entonces  $n^2 = 10^8$ , pero  $2^n > 10^{100}$ . El número  $10^{100}$  es tan grande que no importa cuán rápida sea una computadora, no es capaz de resolver este problema. Cualquier algoritmo con complejidad temporal  $O(p(n))$ , donde  $p(n)$  es una función polinomial, es un algoritmo polinomial. Por otra parte, los algoritmos cuyas complejidades temporales no pueden acotarse con una función polinomial son algoritmos exponenciales.

Hay una gran diferencia entre algoritmos polinomiales y algoritmos exponenciales. Lamentablemente, existe una gran clase de algoritmos exponenciales y no parece haber alguna esperanza de que puedan sustituirse por algoritmos polinomiales.



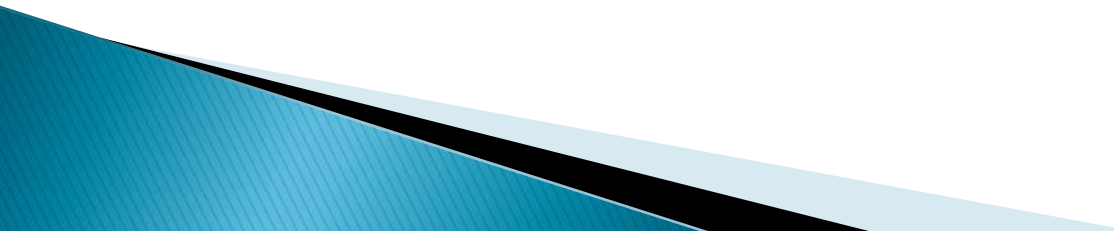
## **ANÁLISIS DEL MEJOR CASO, PROMEDIO Y PEOR DE LOS ALGORITMOS**

Para cualquier algoritmo, se está interesado en su comportamiento en tres situaciones: el mejor caso, el caso promedio y el peor caso. Por lo general el análisis del mejor caso es el más fácil, el análisis del peor caso es el segundo más fácil y el más difícil es el análisis del caso promedio.

## **Funciones de crecimiento**

- ▶ El orden de crecimiento del tiempo de ejecución de un algoritmo da una caracterización simple de la eficiencia del algoritmo y también nos permite comparar el rendimiento relativo de algoritmos alternativos.



- ▶ Cuando miramos los tamaños de entrada lo suficientemente grandes como para hacer solo el orden de crecimiento del tiempo de ejecución relevante, estamos estudiando la eficiencia asintótica de los algoritmos.
  - ▶ Es decir, nos preocupa cómo aumenta el tiempo de ejecución de un algoritmo con el tamaño de la entrada en el límite, ya que el tamaño de la entrada aumenta sin límite.
  - ▶ Por lo general, un algoritmo que sea asintóticamente más eficiente será la mejor opción para todas las entradas excepto las muy pequeñas.
- 

# ¿Cómo medir la dificultad de un problema?

- ▶ Esta pregunta puede responderse de una forma muy intuitiva. Si un problema puede resolverse con un algoritmo con baja complejidad temporal, entonces el problema es sencillo; en caso contrario, se trata de un problema difícil. Esta definición intuitiva conduce al concepto de cota inferior de un problema.

Para describir la cota inferior se usará la notación  $\Omega$ , que se define como sigue:

## Definición

La cota inferior de un problema es la complejidad temporal mínima requerida por cualquier algoritmo que pueda aplicarse para resolverlo.

## Definición

$f(n) = \Omega(g(n))$  si y sólo si existen constantes positivas  $c$  y  $n_0$  tales que para toda  $n > n_0$ ,  $|f(n)| \geq c |g(n)|$ .

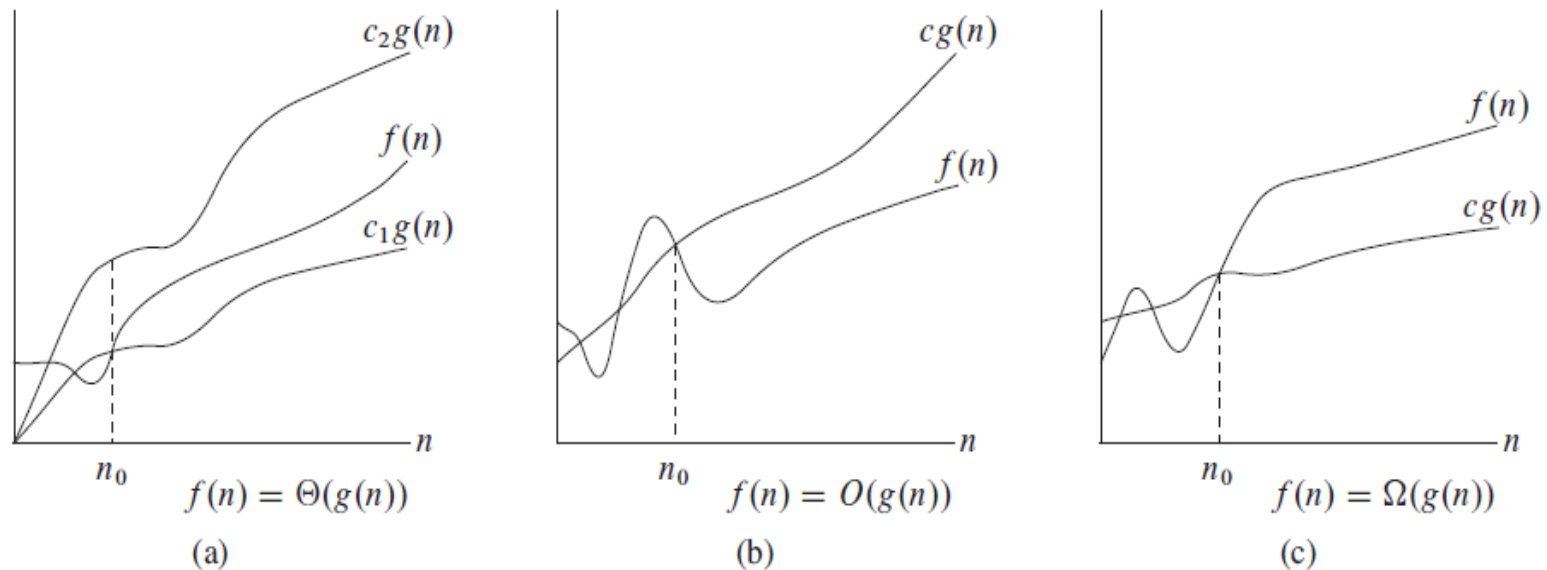


# Cota superior $O$ grande, Cota inferior ( $\Omega$ ) y Cota ajustada asintótica ( $\Theta$ )

- ▶ Así como  $O(g(n))$  representa el conjunto de las funciones que acotan superiormente a la función de complejidad de un determinado algoritmo.
- ▶  $\Omega(g(n))$  representa al conjunto de las funciones que la acotan inferiormente.
- ▶  $\Theta(g(n))$  representa al conjunto de las funciones que crecen a la misma velocidad.

$O(g(n))$	Cota superior asintótica	Cualquiera sea la función $g(n)$ , crecerá más de prisa o, a lo sumo, igual que $f(n)$ .
$\Omega(g(n))$	Cota inferior asintótica	Cualquiera sea la función $g(n)$ , crecerá más lento o, a lo sumo, tan rápido como $f(n)$ .
$\Theta(g(n))$	Cota ajustada asintótica	Cualquiera sea la función $g(n)$ , crecerá a la misma velocidad que $f(n)$ .

# Cota superior O grande, Cota inferior ( $\Omega$ ) y Cota ajustada asintótica ( $\Theta$ )



**Figure 3.1** Graphic examples of the  $\Theta$ ,  $O$ , and  $\Omega$  notations. In each part, the value of  $n_0$  shown is the minimum possible value; any greater value would also work. (a)  $\Theta$ -notation bounds a function to within constant factors. We write  $f(n) = \Theta(g(n))$  if there exist positive constants  $n_0$ ,  $c_1$ , and  $c_2$  such that at and to the right of  $n_0$ , the value of  $f(n)$  always lies between  $c_1g(n)$  and  $c_2g(n)$  inclusive. (b)  $O$ -notation gives an upper bound for a function to within a constant factor. We write  $f(n) = O(g(n))$  if there are positive constants  $n_0$  and  $c$  such that at and to the right of  $n_0$ , the value of  $f(n)$  always lies on or below  $cg(n)$ . (c)  $\Omega$ -notation gives a lower bound for a function to within a constant factor. We write  $f(n) = \Omega(g(n))$  if there are positive constants  $n_0$  and  $c$  such that at and to the right of  $n_0$ , the value of  $f(n)$  always lies on or above  $cg(n)$ .

# Cota ajustada asintótica ( $\Theta$ )

- ▶ Para una función dada  $g(n)$ , denotamos por  $\Theta(g(n))$  el conjunto de funciones:

$\Theta(g(n)) = \{ f(n) : \text{existen constantes positivas}$

$c_1, c_2$  y  $n_0$  tales que:

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ para toda } n \geq n_0 \}$$

# Cota superior asintótica ( $O$ )

- ▶ La notación  $O$  provee un límite asintótico superior.
- ▶ Para una función dada  $g(n)$ , denotamos por  **$O(g(n))$**  que se pronuncia (big  $O$  de  $g$  de  $n$ ) o a veces solo ( $O$  de  $g$  de  $n$ ) el conjunto de funciones:

$O(g(n)) = \{ f(n) : \text{existen constantes positivas}$

$c$  y  $n_0$  tales que:

$0 \leq f(n) \leq c g(n)$  para toda  $n \geq n_0 \}$

# Cota inferior asintótica ( $\Omega$ )

- ▶ La notación  $\Omega$  provee un límite asintótico inferior.
- ▶ Para una función dada  $g(n)$ , denotamos por  **$\Omega(g(n))$**  que se pronuncia (big-Omega de g de n) o a veces solo (Omega de g de n) al conjunto de funciones:

$\Omega(g(n)) = \{ f(n) : \text{existen constantes positivas}$

$c$  y  $n_0$  tales que:

$0 \leq c g(n) \leq f(n)$  para toda  $n \geq n_0$  }

# Notación $o$

- ▶ Usamos la notación  $o$  para denotar un límite superior que no es asintóticamente ajustado.
- ▶ Definimos formalmente  $o(g(n))$  que se puede leer como ("little-oh de g de n") como el conjunto:

$$o(g(n)) = \{ f(n) : \text{para cualquier constante positiva } c > 0 \text{ si existe una constante } n_0 > 0 \text{ tal que: } 0 \leq f(n) < c g(n) \text{ para toda } n \geq n_0 \}$$



# Notación $\omega$

- ▶ Usamos la notación  $\omega$  para denotar un límite inferior que no es asintóticamente ajustado.
- ▶ Definimos formalmente  $\omega(g(n))$  que se puede leer como ("little-omega de g de n") como el conjunto:

$$\omega(g(n)) = \{ f(n) : \text{para cualquier constante positiva } c > 0 \text{ si existe una constante } n_0 > 0 \text{ tal que: } 0 \leq c g(n) < f(n) \text{ para toda } n \geq n_0 \}$$