Análisis de algoritmos

Isabel Besembel Carrera.

Postgrado en Computación. Universidad de Los Andes.

Edif. Economía. La Hechicera. Mérida, 5010, Venezuela.

ibc@ing.ula.ve

Resumen

Operaciones características y complejidad de cálculo. Notaciones. Funciones, sumatorias y recurrencias. Algoritmos recursivos e iterativos. Evaluación de la eficiencia y notación O.

1 Introducción

La eficiencia de un programa tiene dos ingredientes fundamentales: espacio y tiempo.

- La eficiencia en espacio es una medida de la cantidad de memoria requerida por un programa.
- La eficiencia en tiempo se mide en términos de la cantidad de tiempo de ejecución del programa.

Ambas dependen del tipo de computador y compilador, por lo que no se estudiará aquí la eficiencia de los programas, sino la eficiencia de los algoritmos. Asimismo, este análisis dependerá de si trabajamos con máquinas de un solo procesador o de varios de ellos. Centraremos nuestra atención en los algoritmos para máquinas de un solo procesador que ejecutan una instrucción luego de otra.

2 Operaciones características y complejidad de cálculo

La eficiencia de los algoritmos está basada en una operación característica que el algoritmo repite y que define su *complejidad en Tiempo (T(n))*. T(n) es el número de operaciones características que el algoritmo desarrolla para una entrada N dada. El máximo tiempo de ejecución de un algoritmo para todas las instancias de tamaño N, se denomina la complejidad en tiempo para el peor

caso W(n). Asimismo, la complejidad promedio en tiempo es A(n), donde p_j es la probabilidad de que esta instancia ocurra.

$$W(n) = max_{1 \leq j \leq k} T_j(n) A(n) = \sum_{j=1}^{k} p_j T_j(n) \quad (1)$$

Normalmente se tendrán muchos algoritmos diferentes para resolver un mismo problema, por lo que debe existir un criterio para seleccionar el mejor.

3 Notaciones

El interés principal del análisis de algoritmos radica en saber cómo crece el tiempo de ejecución, cuando el tamaño de la entrada crece. Esto es la *eficiencia asintótica* del algoritmo. La notación asintótica se describe por medio de una función cuyo dominio es los números naturales ($\aleph$). Se consideran las funciones asintóticamente no negativas.

Notación Θ: Límite asintóticamente estrecho. Para una función dada g(n),

Definición 3.1 $\Theta(g(n))$ = { *f(n) / $\exists$ las constantes positivas* c_1, c_2 *y* n_o / $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall\ n \geq n_o$ }

Ejemplo:

$$n^2/2 - 3n = \Theta(n^2)$$

Se denota $f(n) = \Theta(g(n)) \equiv f(n) \in \Theta(g(n))$ y se dice que g(n) es el límite asintóticamente estrecho de f(n). Gráficamente se puede observar en la figura 1.

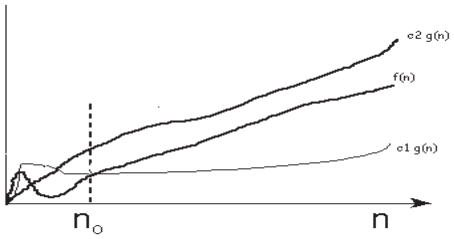

Figura 1: g(n) límite asintóticamente estrecho de f(n).

Notación *O*: Límite asintótico superior

Definición 3.2 *O(g(n))* = { *f(n) / $\exists$ las constantes positivas c y* n_o / $0 \leq f(n) \leq c\ g(n) \forall\ n \geq n_o$ }

Ejemplo: $an^2 + bn + c$ con $a > 0$ tiene $O(n^2)$

El hecho que $f(n) = \Theta(g(n))$ implica f(n) = O(g(n)), por lo que $\Theta(g(n)) \leq O(g(n))$. Se dice que g(n) es el límite asintótico superior de f(n) como se puede observar en la figura 2.

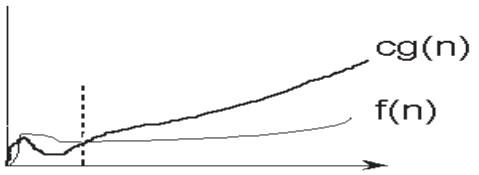

Figura 2: g(n) límite asintótico superior de f(n) .

Notación Ω: Límite asintótico inferior

Definición 3.3 $\Omega(g(n)) = \{ f(n) / \exists$ *las constantes positivas* c *y* $n_o / 0 \leq cg(n) \leq f(n) \; \forall \; n \geq n_o \}$

Ejemplo: $an^2 + bn + c$ con $a > 0$ tiene $O(n^2) = \Omega(n^2) = \Theta(n^2)$. Se dice que g(n) es el límite asintótico inferior de f(n) como se puede observar en la figura 3.

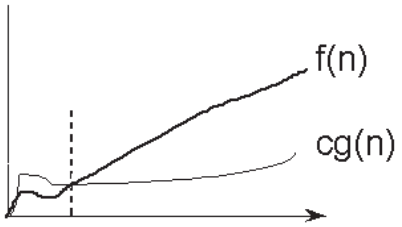

Figura 3: g(n) límite asintótico inferior de f(n) .

Teorema 3.4 *Para cualquier función f(n) y g(n), f(n) =* Θ*(g(n)) si y solo si f(n) = O(g(n)) y f(n) =* Ω*(g(n))*

Notación o: Límite superior no asintóticamente estrecho

Definición 3.5 *o(g(n)) = { f(n) / para cualquier constante* $c > 0$, $\exists$ *una constante* $n_o > 0$ / $0 \leq f(n) < cg(n) \forall\ n \geq n_o$ }

Ejemplo: $2n = o(n^2)$ pero $2n^2 \neq o(n^2)$.

Notación ω: Límite inferior no asintóticamente estrecho

Definición 3.6 $\omega(g(n))$ = { *f(n) / para cualquier constante* $c > 0$, $\exists$ *una constante* $n_o > 0$ / $0 \leq cg(n) < f(n) \forall\ n \geq n_o$ }

Ejemplo: $n^2/2 = \omega(n)$ pero $n^2/2 \neq \omega(n^2)$

3.1 Comparación de las funciones

Para f(n) y g(n) asintóticamente positivas se tiene:

1. Transitividad
   - Si f(n) = Θ(g(n)) y g(n) = Θ(h(n)) entonces f(n) = Θ(h(n))
   - Si f(n) = O(g(n)) y g(n) = O(h(n)) entonces f(n) = O(h(n))
   - Si f(n) = Ω(g(n)) y g(n) = Ω(h(n)) entonces f(n) = Ω(h(n))
   - Si f(n) = o(g(n)) y g(n) = o(h(n)) entonces f(n) = o(h(n))
   - Si f(n) = ω(g(n)) y g(n) = ω(h(n)) entonces f(n) = ω(h(n))
2. Reflexibidad
   - f(n) = Θ(g(n))
   - f(n) = O(g(n))
   - f(n) = Ω(g(n))
3. Simetría $f(n) = \Theta(g(n)) \Longleftrightarrow g(n) = \Theta(f(n))$
4. Simetría transpuesta
   - $f(n) = O(g(n)) \Longleftrightarrow g(n) = \Theta(g(n))$
   - $f(n) = o(g(n)) \Longleftrightarrow g(n) = \omega(g(n))$

Analogía entre la comparación asintótica de dos funciones f y g y la comparación entre dos números reales a y b:

- $f(n) = O(g(n)) \approx a \leq b$
- $f(n) = \Omega(g(n)) \approx a \geq b$
- $f(n) = \Theta(g(n)) \approx a = b$
- $f(n) = o(g(n)) \approx a < b$

- $f(n) = \omega(g(n)) \approx a > b$

Para dos números reales cualesquiera una de las siguientes comparaciones es cierta: $a < b$, $a = b$ ó $a > b$. No todas las funciones son comparables asintóticamente. Ejemplo: f(n) = n y g(n) = $n^{1+sen(n)}$, no pueden ser comparadas usando la notación asintótica pues el valor del exponente 1 + sen(n) oscila entre 0 y 2 tomando todos los valores en ese rango.

4 Funciones y notaciones comunes

Monotonía: Una función f(n) es *monótona creciente* si m $\leq$ n implica f(m) $\leq$ f(n).

Es *monótona decreciente* si m $\leq$ n implica f(m)$\geq$ f(n).

Una función f(n) es *estrictamente creciente* si m $<$ n implica f(m) $<$ f(n) y *estrictamente decreciente* si m $<$ n implica f(m) $>$ f(n).

Pisos y techos: Para cualquier número real x, el *piso* de x $\lfloor x \rfloor$, es el número entero más grande menor o igual a x, y el *techo* de x $\lceil x \rceil$ es el número entero más pequeño mayor o igual a x.

$\forall$ número real x, $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$. Para cualquier entero n, $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$.

Para cualquier entero n y los enteros $a \neq 0$ y $b \neq 0$, $\lceil \lceil n/a \rceil / b \rceil = \lceil n/ab \rceil$ y $\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor$. Ambas funciones son monótonas crecientes.

Polinomios: Dado un entero positivo d, un polinomio en n de grado d es una función $p(n) = \sum_{i=0}^{d} a_i n^i$ donde $a_0, a_1, \ldots, a_d$ son coeficientes del polinomio y $a_d \neq 0$. Un polinomio es asintóticamente positivo si $a_d > 0$.

Para un polinomio asintóticamente positivo de grado d $p(n) = \Theta(n^d)$.

La función n^d es monótona creciente si d $\geq$ 0, y monótona decreciente si d $\leq$ 0.

La función f(n) está limitada por un polinomio si $f(n) = n^{O(1)}$, que es equivalente a $f(n) = O(n^k)$ para alguna constante k.

Exponenciales: Para todo real a $\neq$ 0, m y n se tienen las siguientes identidades:

$a^0 = 1$	$a^1 = a$	$a^{-1} = 1/a$
$(a^m)^n = a^{mn}$	$(a^m)^n = (a^n)^m$	$a^m a^n = a^{m+n}$

para todo n y a $\geq$ 1, la función a^n es monótona decreciente en n.

Para toda constante real a y b tal que a $>$ 1, $\lim_{n\to\infty} \frac{n^b}{a^n}$ por lo que $n^b = o(a^n)$, así cualquier función exponencial positiva crece más rápido que cualquier polinomio.

Siendo e = 2.71828..., la base de la función logaritmo natural, se tiene que para todo real x,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

Para todo real x, se tiene que $e^x \geq 1+x$ y cuando x = 0 también es igual. Cuando $|x| \leq 1$, se tiene la aproximación $1 + x \leq e^x \leq 1 + x + x^2$. Cuando x $\to$ 0, la aproximación de e^x por 1 + x es buena, $e^x = 1+x+\Theta(x^2)$. $\forall x, \lim_{n\to\infty}(1+\frac{x}{n}^n = e^x$.

Logaritmos: Se asume que $\lg n + k = (\lg n) + k$. Para n > 0 y b > 1, la función $\log_b n$ es estrictamente creciente. Para todo real a > 0, b> 0, c > 0 y n, se tiene que:

$\lg n = \log_2 n$	$\ln n = \log_e n$	$\lg^k n = (\lg n)^k$
$\lg \lg n = \lg(\lg n)$	$a = b^{\log_b a}$	$\log_c(ab) = \log_c a + \log_c b$
$\log_b a^n = n \log_c a$	$\log_b a = \frac{\log_c a}{\log_c b}$	$\log_b \frac{1}{a} = -\log_b a$
$\log_b a = \frac{1}{\log_a b}$	$a^{\log_b n} = n^{\log_b a}$	

Para $\ln(1 + x)$ cuando $|x| < 1$,

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$

Para $x > -1$,

$$\frac{x}{1 + x} \leq \ln(1 + x) \leq x$$

cumpliéndose la igualdad para x=0.

Una función es polilogarítmicamente limitada si $f(n) = \lg^{O(1)} n$. Para cualquier constante $a > 0$, $\lg^b n = o(n^a)$, por lo que cualquier función polinomial positiva crece más rápidamente que cualquier función polilogarítmica.

Factoriales: Para $n \geq 0$, n!=1x2x3x...n. Un límite superior débil es $n! \leq n^n$, ya que cada término es el producto de a lo sumo n.

Usando la aproximación de Stirling se prueba:

$$n! = o(n^n)$$

$$n! = \Omega(2^n)$$

$$\lg(n!) = \Theta(n \lg n)$$

$$\sqrt{2n\pi}\frac{n}{e}^n \leq n! \leq \sqrt{2n\pi}\frac{n}{e}^{n+\frac{1}{12n}}$$

Números de Fibonacci: $F_0 = 0, F_1 = 1, F_i = F_{i-1} + F_{i+1}$ para $i \geq 2$.

Se prueba por inducción que

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$

donde ϕ es la sección dorada igual a $\frac{1+\sqrt{5}}{2}$ y $\hat{\phi} = \frac{1-\sqrt{5}}{2}$.

Los números de Fibonacci crecen exponencialmente.

Iteración de la función logarítmica: La notación $\lg^* n$ denotará la función logarítmica iterada, la cual se define como: $\lg^* n = min\{i \geq 0 : \lg^{(i)} n \leq 1\}$ donde $\lg^{(i)} n$ denota la función logaritmo aplicada i veces y definida como:

$$\lg^{(i)} n = \begin{cases} n & \text{si i=0} \\ \lg(\lg^{(i-1)} n) & \text{si } i > 0 \text{ y } \lg^{(i-1)} n > 0 \\ nodefinida & \text{si } i > 0 \text{ y } \lg^{(i-1)} n \leq 0 \text{ ó } \lg^{(i-1)} n \text{ es indefinida} \end{cases} \tag{2}$$

Esta función crece muy lentamente, ejemplo:

$$\lg^* 2 = 1, \lg^* 4 = 2, \lg^* 16 = 3 \lg^* 65536 = 4, \lg^* 2^{65536} = 5$$

5 Sumatorias

Dada una secuencia $a_1, a_2, \ldots$ de números, la sumatoria finita $a_1 + a_2 + \ldots + a_n$ se escribe como: $\Sigma_{i=1}^{n} a_i$. Si n=0, el valor de la sumatoria se define como 0. Si n no es entero, se asume un límite superior de $\lfloor n \rfloor$. Asimismo, si la suma comienza con k=x y x no es entero se asume como valor inicial de la suma $\lfloor x \rfloor$.

Dada una secuencia $a_1, a_2, \ldots$ de números, la sumatoria infinita $a_1 + a_2 + \cdots$ se escribe como: $\Sigma_{k=1}^{\infty} a_k$ que es interpretado como: $\lim_{n \to \infty} \sum_{k=1}^{n} a_k$. Si el límite no existe, la serie *diverge*, de lo contrario *converge*. Una serie que *converge absolutamente* es una serie $\sum_{k=1}^{\infty} a_k$, para la cual $\sum_{k=1}^{\infty} |a_k|$ también converge.

Linealidad: Para cualquier real c y unas secuencias finitas $a_1, a_2, \ldots$ y $b_1, b_2, \ldots$ la propiedad

$$\sum_{k=1}^{n} (ca_k + b_k) = c \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$$

la mantienen las series convergentes infinitas. La propiedad lineal se aplica a la notación asintótica, por ejemplo:

$$\sum_{k=1}^{n} \Theta(f(k)) = \Theta(\sum_{k=1}^{n} f(k))$$

donde la notación Θ de la parte izquierda se aplica a la variable k y la de la derecha a n.

Series aritméticas:

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n = \frac{1}{2}n(n+1) = \Theta(n^2)$$

Series geométricas o exponenciales: Para $x \neq 1$

$$\sum_{k=1}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

cuando $|x| < 1$ se tiene

$$\sum_{k=1}^{n} x^k = \frac{1}{1 - x}$$

Series armónicas: Para n enteros positivos, el n-ésimo armónico es

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$$

Para cualquier secuencia $a_1, a_2, \ldots, a_n$

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = a_n - a_0$$

asimismo se tiene

$$\sum_{k=0}^{n-1} (a_k - a_{k+1} = a_0 - a_n$$

Por ejemplo,

$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} (\frac{1}{k} - \frac{1}{k+1}) = 1 - \frac{1}{n}$$

Productoria: El producto finito $a_1 a_2 \ldots a_n \prod_{k=1}^{n} a_k$ si n=0 el valor del producto es 1. Para convertir de sumatoria a productoria se usa la identidad

$$\lg(\prod_{k=1}^{n} a_k) = \sum_{k=1}^{n} \lg a_k$$

5.1 Métodos para describir los tiempos de corrida

Los métodos más usados para describir el tiempo de corrida de los algoritmos se basan en encontrar una limitación para la suma de una secuencia mediante:

5.1.1 Inducción matemática:

La serie geométrica dada es

$$\sum_{k=0}^{n+1} 3^k = \sum_{k=0}^{n} 3^k + 3^{n+1} \leq c3^n + 3^{n+1} = (\frac{1}{3} + \frac{1}{c})c3^{n+1} \leq c3^{n+1}$$

y está limitada con $O(3^n)$.

5.1.2 Limitando los términos:

Normalmente se escoge el término más grande para limitar a los otros, en general si $a_{max} = max_{1 \leq k \leq n} a_k$ entonces $\sum_{k=0}^{n} a_k \leq n a_{max}$. Este método no es recomendable si la serie puede ser limitada por una serie geométrica, en cuyo caso se prefiere el límite de esta última.

5.1.3 Fisión:

Se expresa la sumatoria en dos o más series separando el rango del índice y encontrando la limitación de cada serie. En general,

$$\sum_{k=0}^{n} a_k = \sum_{k=0}^{k_0-1} a_k + \sum_{k=k_0}^{n} a_k = \Theta(1) + \sum_{k=k_0}^{n} a_k$$

Ejemplo:

$$\sum_{k=1}^{n} k = \sum_{k=1}^{n/2} k + \sum_{k=\frac{n}{2}+1}^{n} k \geq \sum_{k=1}^{n/2} 0 + \sum_{k=\frac{n}{2}+1}^{n} (\frac{n}{2}) \geq (\frac{n}{2})^2 = \Omega(n^2)$$

6 Recurrencias

Cuando un algoritmo contiene llamadas a él mismo (recursivo), su tiempo de ejecución se expresa normalmente con una recurrencia. Una *recurrencia* es una ecuación o desigualdad que describe una función en términos de sus valores para sus entradas más pequeñas. Para encontrar los límites asintóticos (Θ, O) se presentan tres métodos: por substitución, por iteración y el método maestro.

6.1 Método por substitución

Se enuncia una forma de solución y se prueba por inducción que dicha solución asociada a una constante es buena. Este método se aplica cuando se puede enunciar la solución posible de la recurrencia.

Ejemplo: Encontrar el límite superior de la recurrencia T(n)=2T($\lfloor \frac{n}{2} \rfloor$)+n. Suponemos que la solución es T(n)=O(n lg n). Se probará que T(n) $\leq cn \lg n$ para una constante apropiada c>0. Se inicia asumiendo que el límite se da para

$\lfloor \frac{n}{2} \rfloor$. Substituyendo,

$$T(n) \leq 2(c\lfloor \frac{n}{2} \rfloor \lg(\lfloor \frac{n}{2} \rfloor)) + n$$

$$T(n) \leq cn \lg(\frac{n}{2}) + n$$

$$T(n) = cn \lg n - cn \lg 2 + n$$

$$T(n) = cn \lg n - cn + n$$

$$T(n) \leq cn \lg n$$

se cumple para $c \geq 1$

La inducción matemática requiere que se pruebe la solución encontrada para condiciones límites. Si se asume T(1) = 1, no se puede escoger c grande, ya que T(1) $\leq$ c 1 lg 1 = 0. Esta dificultad se resuelve tomando n = 2 o n = 3. Así, T(2)=4 y T(3)=5 y la prueba inductiva T(n) $\leq cn \lg n$ para $c \geq 2$ es suficiente. Se pueden usar cambios de variables para simplificar, ejemplo: $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$. Se hace $m = \lg n$ y $T(2^m) = 2T(2^{m/2}) + m$ se hace $S(m) = T(2^m)$ y se obtiene S(m) = 2 S(m/2) + m, que es parecida a la anterior, por lo cual $T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$

6.2 Método iterativo

Se expande la recurrencia y se expresa como una sumatoria de términos que depende sólo de n y de condiciones iniciales. Ejemplo:

$$T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + n$$

$$T(n) = n + 3(\lfloor \frac{n}{4} \rfloor + 3T(\lfloor \frac{n}{16} \rfloor))$$

$$T(n) = n + 3(\lfloor \frac{n}{4} \rfloor + 3(\lfloor \frac{n}{16} \rfloor + 3T(\lfloor \frac{n}{64} \rfloor)))$$

$$T(n) = n + 3\lfloor \frac{n}{4} \rfloor + 9\lfloor \frac{n}{16} \rfloor + 27T(\lfloor \frac{n}{64} \rfloor)$$

$$T(n) = n + 3\frac{n}{4} + 9\frac{n}{16} + 27\frac{n}{64} + \cdots + 3^{\log_4 n}\Theta(1)$$

$$T(n) \leq n\sum_{i=0}^{\infty}(\frac{3}{4})^i + \Theta(n^{\log_4 3})$$

T(n) = 4n + o(n) = O(n)

6.3 Método maestro

Se usa para resolver recurrencias del tipo T(n) = a T(n/b) + f(n), donde f(n) es una función asintóticamente positiva y las constantes $a \geq 1$, $b > 1$. Ella expresa que el problema se divide en a subproblemas de tamaño n/b. Los a subproblemas se resuelven recursivamente en tiempo T(n/b). El costo de dividir el problema y combinar sus soluciones es descrito por f(n). Normalmente en estos casos no se consideran los techos y pisos.

Teorema 6.1 Teorema maestro*: Sean $a \geq 1$, $b > 1$ constantes, f(n) una funcin y T(n) una recurrencia para números enteros no negativos, T(n) = a T(n/b) + f(n), donde n/b puede ser $\lfloor n/b \rfloor$ o $\lceil n/b \rceil$ entonces T(n) está limitado asintóticamente segn:*

1. *Si $f(n) = O(n^{\log_b a - e})$ para alguna constante $e > 0$, entonces $T(n) = \Theta(n^{\log_b a})$*
2. *Si $f(n) = \Theta(n^{\log_b a})$, entonces $T(n) = \Theta(n^{\log_b a} \lg n)$*
3. *Si $f(n) = \Omega(n^{\log_b a + e})$ para alguna constante $e > 0$, y si $af(n/b) \leq cf(n)$ para alguna constante $c < 1$ y n suficientemente grande, entonces $T(n) = \Theta(f(n))$.*

7 Evaluación de la eficiencia

La mejor técnica para diferenciar la eficiencia de los algoritmos es el estudio de los órdenes de complejidad. El orden de complejidad se expresa generalmente en términos de la cantidad de datos procesados por el programa, denominada N. Este N puede ser el tamaño dado o estimado.

Ejemplo: Un algoritmo que procese un vector V(N) tendrá un orden de complejidad de N, ya que si N crece, en esa misma proporción crece el orden de complejidad de él.

La cantidad de tiempo de procesamiento de un algoritmo (T(n)), por lo general viene dado en función de N, y puede expresarse en base a los casos

típicos de ese N, caso promedio A(n), o en base a casos extremos no deseables, como el peor de los casos W(n).

El orden de complejidad se define como una función que domina la ecuación que expresa en forma exacta el tiempo de ejecución del programa.

g(x) domina a f(x), si dada una constante C cualquiera C*g(x)≥ f(x) ∀x

g(x) domina asintóticamente a f(x), si g(x) domina a f(x) para los valores muy grandes de x.

$$f(N) = N^2 + 5N + 100$$

$$g(N) = N^2$$

entonces g(N) domina a f(N)

El orden de complejidad se denota con una o mayúscula, O(g(N)) O(N^2), O(N). Un orden O(N) indica que el tiempo de ejecución decrece suavemente en proporción al decrecimiento de N.

Aunque dos algoritmos tengan el mismo orden O(N), ellos pueden tener diferentes tiempos de ejecución para iguales valores de N.

Reglas para determinar el orden de complejidad:

1. O(C*g) = O(g)
2. O(f * g) = O(f) * O(g) O(f/g) = O(f) / O(g)
3. O(f+g) = función dominante entre O(f) y O(g)

Ejemplos: O(2456 * N) = O(N)
O((20 * N) * N) = O(20 * N) * O(N) = O(N^2)

Algunas funciones de dominación más comunes son:

N^b domina a N!	N! domina a b^N
b^N domina a c^N si $b > c$	b^N domina a N^a si $a \geq 0$
N^n domina a N^m si $n \geq m$	N domina a $\log_a N$ si $a \geq 1$
$\log_a N$ domina a $\log_b N$ si $b \geq a \geq 1$	$\log_a N$ domina a 1 si $a \geq 1$

Un algoritmo de O(1) tiene complejidad constante y por ello son los más eficientes y los preferidos. La mayoría de los programas tienen complejidad polinomial O(N^a), donde N es la variable y a es una constante mayor que 1. Ejemplo: O(N), O(N^2), O(N^3), O($\log N$)

La complejidad no polinomial (NP) es aquella que tiene un orden mayor que la polinomial. Ejemplo: La complejidad exponencial O(a^N)

Nombres de las más usadas:

- $\log N$ complejidad logarítmica ($\log_2 N \equiv \lg N$)
- N complejidad lineal
- N^2 complejidad cuadrática
- N^3 complejidad cúbica
- 2^N complejidad exponencial

Comparación entre diferentes complejidades:

N	$\log N$	N $\log N$	N^2	N^3	2^N	3^N
1	0	0	1	1	2	3
2	1	2	4	4	4	9
4	2	8	16	64	16	81
8	3	24	64	512	256	6.561
16	4	64	256	4.096	65.538	43.046.721
32	5	160	1.024	32.768	4.294.967.296	??
64	6	384	4.096	262.144	*	??
128	7	896	16.384	2.097.152	**	??

* el número de instrucciones que puede ejecutar un supercomputador de 1 GFLOP en 500 años

** sería 500 billones de veces la edad del universo (20 billones de años) en nanosegundos.

Los algoritmos sin lazos y sin recursión tienen complejidad constante. La determinación del orden de complejidad de un algoritmo se inicia por los lazos y las recursiones. Los lazos anidados tendrán complejidad polinómica.

Correspondencia entre la estructura de programación y el orden de complejidad.

Estructura	Orden
Secuencial (S)	O(1)
S1 S2	La función dominante entre O(S1) y O(S2)
Si (condición) entonces S1 sino S2 fsi	La función dominante entre O(S1), O(S2) y O(condición), en el peor de los casos
[S1] i = 1, N	O(N * S1)

Ejemplo de la utilización de esta tabla para los casos siguientes:

[m(i, j) = 0 j = 1, N] i = 1, N tiene O(N^2)

a = a + b_i i = 3, 8 tiene O(1)