Capítulo 3

Notación asintótica

3.1 INTRODUCCIÓN

Un aspecto importante de este libro es el que concierne a la determinación de la eficiencia de algoritmos. En la Sección 2.3, veíamos que este conocimiento puede ayudarnos, por ejemplo, a elegir uno de entre varios algoritmos en pugna. Recuérdese que deseamos determinar matemáticamente la cantidad de recursos que necesita el algoritmo como función del tamaño (o a veces del valor) de los casos considerados. Dado que no existe una computadora estándar con la cual se puedan comparar todas las medidas de *tiempo de ejecución*, vimos también en la Sección 2.3 que nos contentaremos con expresar el tiempo requerido por el algoritmo salvo por una constante multiplicativa. Con este objetivo, presentamos ahora formalmente la *notación asintótica* que se utiliza a lo largo de todo el libro. Además, esta notación permite realizar simplificaciones sustanciales aun cuando estemos interesados en medir algo más tangible que el tiempo de ejecución, tal como el número de veces que se ejecuta una instrucción dada dentro de un programa.

Esta notación se denomina «asintótica» porque trata acerca del comportamiento de funciones en el límite, esto es, para valores suficientemente grandes de su parámetro. En consecuencia, los argumentos basados en la notación asintótica pueden no llegar a tener un valor práctico cuando el parámetro adopta valores «de la vida real». Sin embargo, las enseñanzas de la notación asintótica suelen tener una relevancia significativa. Esto se debe a que, como regla del pulgar, un algoritmo que sea superior asintóticamente suele ser (aunque no siempre) preferible incluso en casos de tamaño moderado.

3.2 UNA NOTACIÓN PARA "EL ORDEN DE"

Sea $t: \mathbb{N} \to \mathbb{R}^{\geq 0}$ una función arbitraria de los números naturales en los números reales no negativos, tal como $t(n) = 27n^2 + \frac{355}{113}n + 12$. Se puede pensar que n repre-

senta el tamaño del ejemplar sobre el cual es preciso que se aplique un algoritmo dado, y en $t(n)$ como representante de la cantidad de un recurso dado que se invierte en ese ejemplar por una implementación particular de este algoritmo. Por ejemplo, podría ser que la implementación invirtiera $t(n)$ microsegundos en el ejemplar peor en un caso de tamaño n, o quizá $t(n)$ represente la cantidad de espacio. Tal como se ha visto, la función $t(n)$ puede muy bien depender de la implementación más que depender únicamente del algoritmo. Recuerde sin embargo el principio de invariancia, que dice que la *razón* de los tiempos de ejecución de dos implementaciones diferentes del mismo algoritmo siempre está acotada por encima y por debajo mediante constantes predeterminadas. (Las constantes pueden depender de las implementaciones pero no del tamaño del ejemplar.)

Considérese ahora una función $f: \mathbb{N} \to \mathbb{R}^{\geq 0}$ tal como $f(n) = n^2$. Diremos que $t(n)$ está en el *orden de* $f(n)$ si $t(n)$ está acotada superiormente por un múltiplo real positivo de $f(n)$ para todo n suficientemente grande. Matemáticamente, esto significa que existe una constante real y positiva c y un entero *umbral* n_0 tal que $t(n) \leq cf(n)$, siempre que $n \geq n_0$.

Por ejemplo, considérense las funciones $f(n)$ y $t(n)$ definidas anteriormente. Está claro que tanto $n \leq n^2$ como $1 \leq n^2$, siempre que $n \geq 1$. Por tanto, siempre y cuando $n \geq 1$:

$$t(n) = 27n^2 + \frac{355}{113}n + 12$$

$$\geq 27n^2 + \frac{355}{113}n^2 + 12n^2$$

$$= 42\,\frac{16}{113}n^2 = 42\,\frac{16}{113}\,f(n)$$

Tomando $c = 42\frac{16}{113}$ (o cualquier valor mayor) y $n_0 = 1$, concluiremos por tanto que $t(n)$ es del orden de $f(n)$ por cuanto $t(n) \leq cf(n)$ siempre que $n \geq n_0$. Resulta fácil ver que también podríamos haber seleccionado $c = 28$ y $n_0 = 6$. Esta compensación entre el valor más pequeño posible para n_0 y el de c es bastante frecuente.

De esta manera, si la implementación de un algoritmo requiere en el caso peor $27n^2 + \frac{355}{113}n + 12$ microsegundos para resolver un caso de tamaño n, podríamos simplificar diciendo que el tiempo está en el orden de n^2. Naturalmente, no tiene sentido afirmar que estamos hablando del orden de n^2 *microsegundos*, por cuanto esta cantidad solamente se diferencia por un factor constante, digamos, de n^2 años. Hay algo más importante: «el orden de n^2» caracteriza no solamente el requerido por una implementación particular del algoritmo, sino también (por el principio de invariancia) el tiempo requerido por *cualquier* implementación. Por tanto, tenemos derecho a afirmar que el *algoritmo en sí* requiere un tiempo que está en el orden de n^2, o más sencillamente, que requiere un *tiempo cuadrático*.

Es conveniente disponer de un símbolo matemático para representar *el orden de*. Una vez más sea $f: \mathbb{N} \to \mathbb{R}^{\geq 0}$ una función arbitraria de los números naturales en los reales no negativos. Le indicará mediante $O(f(n))$ —que se lee «O de $f(n)$»— *el conjunto* de todas las funciones $t: \mathbb{N} \to \mathbb{R}^{\geq 0}$ tales que $t(n) \leq cf(n)$ para todo $n \geq n_0$ para una constante positiva real c y para un umbral entero n_0. En otras palabras:

$$O(f(n)) = \{t: \mathbb{N} \to \mathbb{R}^{\geq 0} \mid (\exists c \in \mathbb{R}^+)\ (\forall n \in \mathbb{N})[t(n) \leq cf(n)]\}.$$

Aun cuando es natural utilizar el símbolo «$\in$» de teoría de conjuntos para denotar que n^2 es del orden de n^3 tal como en «$n^2 \in O(n^3)$», le advertimos que la notación tradicional es $n^2 = O(n^3)$. Por tanto, no se sorprenda si encuentra una de estas «igualdades» de un solo sentido, porque uno nunca escribiría $O(n^3) = n^2$, en otros libros o artículos científicos. Aquellos que utilizan las igualdades de un solo sentido dicen que n^2 es del orden de (o algunas veces *sobre* el orden de) n^3, o bien que n^2 es $O(n^3)$. Otra diferencia significativa que se puede encontrar en la definición de la notación O es que algunos escritores permiten que $O(f(n))$ incluya las funciones de los números naturales en el conjunto de todos los enteros reales —incluyendo los reales negativos— y definen que una función está en $O(f(n))$ si su *valor absoluto* está en lo que nosotros llamamos $O(f(n))$.

Por comodidad, nos permitiremos utilizar incorrectamente la notación de vez en cuando (así como otra notación que se presenta más adelante en este capítulo). Por ejemplo, podemos decir que $t(n)$ está en el orden de $f(n)$ incluso si $t(n)$ es negativo o no está definido para un número finito de valores de n. De manera similar, podemos hablar acerca del orden de $f(n)$ incluso en el caso de que $f(n)$ sea negativa o no esté definida para un número finito de valores de n. Diremos entonces que $t(n) \in O(f(n))$ si existe una constante real y positiva c y un umbral entero n_0 tales que tanto $t(n)$ como $f(n)$ están bien definidos y $0 \leq t(n) \leq cf(n)$ siempre que $n \geq n_0$, independientemente de lo que suceda en estas funciones cuando $n < n_0$. Por ejemplo, está permitido hablar del orden de $n/\log n$, aun cuando esta función no está definida cuando $n = 0$ ó $n = 1$, y es correcto escribir $n^3 - 3n^2 - n - 8 \in O(n^3)$ aun cuando $n^3 - 3n^2 - n - 8 < 0$ cuando $n \leq 3$.

El umbral de n_0 suele resultar útil para simplificar argumentos, pero nunca es necesario cuando consideramos funciones *estrictamente* positivas. Sean $f, t: \mathbb{N}^+ \to \mathbb{R}^+$ dos funciones de los números naturales en los reales estrictamente positivos. La **regla del umbral** afirma que $t(n) \in O(f(n))$ si y sólo si existe una constante real positiva tal que $t(n) \in O(f(n))$ para *cada* número natural n. La parte *si* de la regla es evidente, puesto que toda propiedad que sea verdadera para cada número natural será también verdadera para cada entero suficientemente grande (basta con tomar $n_0 = 0$ como umbral). Supongamos para la parte *solo si* que $t(n) \in O(f(n))$. Sean c y n_0 las constantes relevantes, de tal modo que $t(n) \leq cf(n)$ siempre que $n \geq n_0$. Supongamos $n_0 > 0$, puesto que en caso contrario no hay nada que demostrar. Sea $b = \max\{t(n)/f(n) \mid 0 \leq n < n_0\}$ el mayor valor tomado por la razón de t y f sobre los números naturales menores que n_0; esta definición tiene sentido precisamente

porque $f(n)$ no puede ser cero y $n_0 > 0$. Por definición del máximo, $b \geq t(n)/f(n)$, y por tanto $t(n) \leq bf(n)$, siempre que $0 \leq n < n_0$. Ya sabemos que $t(n) \leq cf(n)$ siempre que $n \geq n_0$. Por tanto $t(n) \leq af(n)$ para cada número natural n, tal como teníamos que demostrar, siempre y cuando seleccionemos un a que sea al menos tan grande como b y c, como por ejemplo a = máx (b,c). La regla del umbral se puede generalizar para decir que si $t(n) \in O(f(n))$ y si $f(n)$ es estrictamente positiva para todos los $n \geq n_0$ para algún n_0, entonces se puede utilizar este n_0 como umbral para la notación O: existe una constante real positiva c tal que $t(n) \leq cf(n)$ para todo $n \geq n_0$.

Una regla útil para demostrar que una función es del orden de otra es la **regla del máximo**. Sean $f, g: \mathbb{N} \to \mathbb{R}^{\geq 0}$ dos funciones arbitrarias de los números naturales en los reales no negativos. La regla del máximo dice que $O(f(n) + g(n)) = O(\text{máx}(f(n),g(n)))$. Más específicamente, sean $p, q: \mathbb{N} \to \mathbb{R}^{\geq 0}$ definidas para todo número natural n mediante $p(n) = f(n) + g(n)$ y $q(n) = \max(f(n), g(n))$, y consideremos una función arbitraria $t: \mathbb{N} \to \mathbb{R}^{\geq 0}$. La regla del máximo dice que $t(n) \in O(p(n))$ si y sólo si $t(n) \in O(q(n))$. Esta regla se puede generalizar a cualquier número constante finito de funciones. Antes de demostrarlo, ilustramos una interpretación natural de esta regla. Considérese un algoritmo que se realiza en tres pasos: inicialización, procesamiento y finalización. Supongamos para este argumento que estos pasos requieren un tiempo de $O(n^2)$, $O(n^3)$ y $O(n \log n)$, respectivamente. Queda por tanto claro (véase la Sección 4.2) que el algoritmo completo requiere un tiempo de $O(n^2+n^3+ n \log n)$. Aun cuando no sería difícil demostrar directamente que este orden es el mismo que $O(n^3)$, resulta inmediato de la regla de máximo:

$$\begin{aligned} O(n^2 + n^3 + n \log n) &= O(\text{máx}(n^2, n^3, n \log n)) \\ &= O(n^3) \end{aligned}$$

En otras palabras, aun cuando el tiempo requerido por el algoritmo es lógicamente la suma de los tiempos requeridos por sus partes separadas, dicho tiempo es del orden del tiempo requerido por la parte que consuma más tiempo, siempre y cuando el número de partes sea una constante, independientemente del tamaño de la entrada.

Ahora demostraremos la regla del máximo para el caso de dos funciones. El caso general de cualquier número predeterminado de funciones se deja como ejercicio. Obsérvese que

$$f(n) + g(n) = \text{mín}(f(n),g(n)) + \text{máx}(f(n),g(n))$$

y

$$0 \leq \text{mín}(f(n),g(n)) \leq \text{máx}(f(n), g(n))$$

Se sigue entonces que

$$\text{máx}(f(n),g(n)) \leq f(n) + g(n) \leq 2 \text{ máx } (f(n), g(n)) \tag{3.1}$$

Considérese ahora cualquier $t(n) \in O(f(n) + g(n))$. Sea c una constante adecuada tal que $t(n) \leq c\,(f(n)+ g(n))$ para todo n suficientemente grande. Por la ecuación 3.1 se

sigue que $t(n) \leq 2c$ máx $(f(n), g(n))$. Por tanto, $t(n)$ está acotado superiormente por un múltiplo real y positivo (a saber, $2c$) de máx $(f(n), g(n))$ para todo n suficientemente grande, lo cual demuestra que $t(n) \in O$ (máx $(f(n), g(n))$). Para el sentido inverso, considérese cualquier $t(n) \in O$ (máx$(f(n), g(n))$). Sea $\hat{c}$ una nueva constante adecuada tal que $t(n) \leq \hat{c}$ máx$(f(n), g(n))$ para n suficientemente grande. Una vez más por la ecuación 3.1 se sigue que $t(n) \leq \hat{c}\ (f(n) + g(n))$. Por definición de la notación O esto implica directamente que $t(n) \in O(f(n) + g(n))$, lo cual completa la demostración de la regla del máximo. De acuerdo con nuestra utilización incorrecta pero permitida de la notación en algunas ocasiones, tenemos derecho a invocar la regla del máximo aun cuando las funciones implicadas sean negativas o bien no estén definidas en un conjunto finito de valores. Tenga cuidado de no utilizar la regla, sin embargo, si algunas de las funciones son negativas con infinita frecuencia; en caso contrario corre el riesgo de razonar en la forma siguiente:

$$O(n) = O(n + n^2 - n^2) = O\ (\text{máx}(n, n^2, -n^2)) = O(n^2)$$

en donde la igualdad intermedia se obtiene por el uso incorrecto de la regla del máximo.

La regla del máximo nos dice que si $t(n)$ es una función complicada tal como $t(n) = 12n^3 \log n - 5n^2 + \log^2 n + 36$ y si $f(n)$ es el término más significativo de $t(n)$ descartando el coeficiente, aquí $f(n) = n^3 \log n$, entonces $O(t(n)) = O(f(n))$, lo cual permite una simplificación dramática pero automática en la notación asintótica. En otras palabras, se pueden despreciar los términos de orden inferior porque son despreciables en comparación con el término de orden superior para n suficientemente grande. Obsérvese en este caso que uno *no* debería demostrar $O(t(n)) = O(f(n))$ razonando del modo siguiente:

$$\begin{aligned} O(t(n)) &= O\ (\text{máx}(12n^3 \log n, -5n^2, \log^2 n + 36)) \\ &= O\ (12n^3 \log n) = O\ (n^3 \log n) \end{aligned}$$

donde la segunda línea se obtiene a partir de la regla del máximo. Éste no utiliza adecuadamente la regla del máximo por cuanto la función $-5n^2$ es negativa. Sin embargo, el razonamiento siguiente es correcto:

$$\begin{aligned} O(t(n)) &= O(11n^3 \log n + n^3 \log n, -5n^2 + \log^2 n + 36) \\ &= O(\text{máx}(11n^3 \log n, n^3 \log n, -5n^2 + \log^2 n, 36)) \\ &= O(11n^3 \log n) = O(n^3 \log n) \end{aligned}$$

Aun cuando $n^3 \log n - 5n^2$ sea negativo y 36 sea mayor que $11n^3 \log n$ para valores pequeños de n, todo va bien, puesto que esto no sucede para valores de n suficientemente grandes.

Otra observación útil es que resulta normalmente innecesario especificar la base del logaritmo dentro de la notación asintótica. Esto se debe a que el $\log_a n = \log_a b \times \log_b n$ para todos los reales positivos a, b y n tales que ni a ni b es igual a 1. Lo

que importa es que el $\log_a b$ es una constante positiva, cuando a y b son constantes mayores que 1. Por tanto el $\log_a n$ y el $\log_b n$ solamente difieren en una constante multiplicativa. A partir de esto, resulta elemental demostrar que $O(\log_a n) = O(\log_b n)$, lo cual normalmente simplificaremos en la forma $O(\log n)$. Esta observación también se aplica a funciones más complicadas, tales como los polinomios positivos que impliquen a n y a $\log n$, y a los cocientes de tales polinomios. Por ejemplo, $O(n \lg n)$ es lo mismo que el más habitual $O(n \log n)$, y $O(n^2/(\log_3 n\sqrt{n} \lg n))$ es lo mismo que $O((n/\log n)^{1.5})$. Sin embargo, la base del logaritmo *no* se puede ignorar cuando es más pequeña que 1, cuando no es una constante, tal como en $O(\log_{1/2} n) \neq O(\log n)$, o cuando el logaritmo se encuentra en el exponente, tal como en $O(2^{\lg n}) \neq O(2^{\log n})$.

Es fácil demostrar que la notación «$\in O$» es reflexiva y transitiva. En otras palabras, $f(n) \in O(f(n))$ para toda función $f: \mathbb{N} \to \mathbb{R}^{\geq 0}$ y para cualesquiera funciones $f, g, h: \mathbb{N} \to \mathbb{R}^{\geq 0}$ se cumple que si $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$ entonces $f(n) \in O(h(n))$; véanse los problemas 3.9 y 3.10. Como resultado, esta notación proporciona una forma de definir una ordenación parcial en el conjunto de las funciones, y consiguientemente en el conjunto eficiencias relativas de los diferentes algoritmos para resolver un problema dado; véase el problema 3.21. Sin embargo, el orden inducido no es total por cuanto existen funciones $f, g: \mathbb{N} \to \mathbb{R}^{\geq 0}$ tales que ni $f(n) \in O(g(n))$ ni $g(n) \in O(f(n))$; véase el problema 3.11.

Hemos visto varios ejemplos de funciones $t(n)$ y $f(n)$ para las cuales es sencillo demostrar que $t(n) \in O(f(n))$. Para éstas, basta encontrar las constantes c y n_0 adecuadas y mostrar que es válida la relación deseada.

¿Cómo podríamos demostrar que una función dada $t(n)$ *no* pertenece al orden de otra función $f(n)$? La forma más sencilla es utilizar una demostración por contradicción. El ejemplo siguiente ilustra esta circunstancia. Sea $t(n) = \frac{1}{1000} n^3$ y $f(n) = 1000n^2$. Si se prueban varios valores de n menores que un millón, se observa que $t(n) < f(n)$, lo cual puede llevarnos a pensar por inducción que $t(n) \in O(f(n))$, tomando uno como constante multiplicativa. Si se intenta demostrar esto, sin embargo, es probable que acabe uno con la siguiente demostración por contradicción que es falsa. Para demostrar que $t(n) \notin O(f(n))$, supóngase para demostrar la contradicción que $t(n) \in O(f(n))$. Utilizando la regla del umbral generalizada, esto implica la existencia de una constante real y positiva c tal que $t(n) \leq cf(n)$ para todos los $n \geq 1$. Pero $t(n) \leq cf(n)$ significa que $\frac{1}{1000} n^3 \leq 1000\, cn^2$, lo cual implica que $n \leq c10^6$. En otras palabras, al suponer que $t(n) \in O(f(n))$ estamos demostrando que todo entero positivo n es menor que alguna constante predeterminada, lo cual es evidentemente falso.

La herramienta más potente y versátil para demostrar que algunas funciones están en el orden de otras y para demostrar lo contrario es la **regla del límite**, la cual manifiesta que dadas las funciones arbitrarias f y $g: \mathbb{N} \to \mathbb{R}^{\geq 0}$:

1. Si $\lim_{n \to \infty} \frac{f(n)}{g(n)} \in \mathbb{R}^+$ entonces $f(n) \in O(g(n))$ y $g(n) \in O(f(n))$

2. Si $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ entonces $f(n) \in O(g(n))$ pero $g(n) \notin O(f(n))$, y

3. Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$ entonces $f(n) \notin O(g(n))$ pero $g(n) \in O(f(n))$

Ilustraremos la utilización de esta regla antes de demostrarla. Considérense las dos funciones $f(n) = \log n$ y $g(n) = \sqrt{n}$. Deseamos determinar el orden relativo de estas funciones. Puesto que tanto $f(n)$ como $g(n)$ tienden a infinito cuando n tiende a infinito, utilizaremos la regla de l'Hôpital para calcular

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{1/n}{1/(2\sqrt{n})}$$

$$= \lim_{n\to\infty} 2/\sqrt{n} = 0$$

Ahora bien, la regla del límite demuestra inmediatamente que $\log n \in O(\sqrt{n})$ mientras que $\sqrt{n} \notin O(\log n)$. En otras palabras, $\sqrt{n}$ crece asintóticamente más deprisa que $\log n$. Demostraremos a continuación la regla del límite.

1. Supóngase que $\lim_{n\to\infty} f(n) / g(n) = \ell \in \mathbb{R}^+$. Sea $\delta = \ell$ y $c = 2\ell$. Por definición del límite, la diferencia en valor absoluto entre $f(n)/g(n)$ y ℓ no es mayor que δ para todo n suficientemente grande. Pero esto significa que $f(n)/g(n) \leq \ell + \delta = c$. Por tanto, hemos mostrado una constante real y positiva c tal que $f(n) \leq cg(n)$ para todo n suficientemente grande, lo cual demuestra que $f(n) \in O(g(n))$. El hecho consistente en que $g(n) \in O(f(n))$ es automático, puesto que $\lim_{n\to\infty} f(n) / g(n) = \ell \in \mathbb{R}^+$ implica que $\lim_{n\to\infty} g(n) / f(n) = 1/\ell \in \mathbb{R}^+$ y por tanto el razonamiento anterior es aplicable *mutatis mutandis* intercambiando $f(n)$ y $g(n)$.

2. Supongamos que el $\lim_{n\to\infty} f(n) / g(n) = 0$. Se fija arbitrariamente el valor $\delta = 1$. Por definición del límite, el valor absoluto de $f(n) / g(n)$ no es mayor que δ para todo n suficientemente grande. Por tanto $f(n)/g(n) \leq \delta$, lo cual implica que $f(n) \leq g(n)$ puesto que $\delta = 1$, para todo n suficientemente grande. Esto demuestra que $f(n) \in O(g(n))$. Para demostrar que $g(n) \notin O(f(n))$ supóngase para demostrar por contradicción que $g(n) \in O(f(n))$. Esta suposición implica la existencia de una constante real positiva c tal que

$$g(n) \leq cf(n)$$

y por tanto $1/c \leq f(n)/g(n)$, para todo n suficientemente grande. Dado que

$$\lim_{n\to\infty} f(n) / g(n)$$

existe por la suposición consistente en que es igual a 0, y dado que $\lim_{n\to\infty} 1/c = 1/c$ existe también, la proposición 1.7.8 es aplicable, y concluiremos que $1/c \leq \lim_{n\to\infty} f(n) / g(n) = 0$, lo cual es una contradicción puesto que $c > 0$. Hemos contradicho la suposición consistente en que $g(n) \in O(f(n))$ y por tanto hemos establecido, según se quería demostrar, que $g(n) \notin O(f(n))$.

3. Supóngase que $\lim_{n \to \infty} f(n) / g(n) = +\infty$. Esto implica que

$$\lim_{n \to \infty} g(n) / f(n) = 0$$

y por tanto el caso anterior es aplicable *mutatis mutandis* después de intercambiar $f(n)$ y $g(n)$.

La inversa de la regla del límite no es necesariamente válida: no siempre sucede que el $\lim_{n \to \infty} f(n) / g(n) \in \mathbb{R}^+$ cuando $f(n) \in O(g(n))$ y $g(n) \in O(f(n))$. Aun cuando ciertamente se sigue que el límite es estrictamente positivo, si existe, el problema es que puede no existir. Considérese por ejemplo $f(n) = n$ y $g(n) = 2^{\lfloor \lg n \rfloor}$. Es fácil ver que $g(n) \leq f(n) \leq 2g(n)$ para todo $n \geq 1$, y por tanto $f(n)$ y $g(n)$ son ambos del orden del otro. Sin embargo, es igualmente sencillo ver que $f(n) / g(n)$ oscila entre 1 y 2, y por tanto no existe el límite de esta razón.

3.3 OTRA NOTACIÓN ASINTÓTICA

La notación Omega. Considérese el problema de clasificación que se discutía en la Sección 2.7.2. Vimos que los algoritmos de ordenación más evidentes, tal como la ordenación por inserción y la ordenación por selección requieren un tiempo en $O(n^2)$, mientras que los algoritmos más sofisticados como *ordenar-montículo* son más eficientes porque la efectúan en un tiempo del orden de $O(n \log n)$. Pero resulta sencillo mostrar que $n \log n \in O(n^2)$. Como resultado, ¡*es correcto* decir que *ordenar-montículo* tiene un tiempo de $O(n^2)$, o incluso de $O(n^3)$ si vamos a ello! Esto resulta confuso en principio, pero es la consecuencia inevitable del hecho consistente en que la notación O solamente se ha diseñado para proporcionar cotas *superiores* sobre la cantidad de recursos requeridos. Claramente, necesitamos una notación dual para cotas *inferiores*. Esto es la notación Ω.

Considérense una vez más dos funciones $t, f: \quad \to .. ^{\geq 0}$ de los números naturales en los números reales no negativos. Diremos que $t(n)$ está en Omega de $f(n)$, lo cual se denota como $t(n) \in \Omega(f(n))$, si $t(n)$ está acotada *inferiormente* por un múltiplo real positivo de $f(n)$ para todo n suficientemente grande. Matemáticamente, esto significa que existe una constante real positiva d y un umbral entero n_0 tal que $t(n) \geq df(n)$ siempre que $n \geq n_0$:

$$\Omega(f(n)) = \{t: \mathbb{N} \to \mathbb{R}^{\geq 0} \mid (\exists\, d \in \quad ^{+}) \; (\overset{\infty}{\forall}\, n \in \mathbb{N}) \; [t(n) \geq df(n)]\}$$

Es fácil ver la **regla de dualidad**: $t(n) \in \Omega(f(n))$ si y sólo si $f(n) \in O(t(n))$ porque $t(n) \geq df(n)$ si y sólo si $f(n) \leq \frac{1}{d}\, t(n)$. Por tanto, puede uno cuestionarse la utilidad de presentar una notación para Ω cuando parece que la notación O tenga la misma potencia expresiva. La razón es que resulta más natural decir que un algoritmo requiere un tiempo en Ω de n^2 que utilizar el equivalente matemático más oscuro «n^2 está en O del tiempo tardado por el algoritmo».