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 $\Omega$.

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 $\Omega$ 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 $\Omega$ de $n^2$ que utilizar el equivalente matemático más oscuro «$n^2$ está en $O$ del tiempo tardado por el algoritmo».

Gracias a la regla de dualidad, sabemos a partir de la sección anterior que $\sqrt{n} \in \Omega(\log n)$ mientras que $\log n \notin \Omega(\sqrt{n})$, entre otros muchos ejemplos. Hay algo más importante, y es que la regla de dualidad se puede utilizar de la forma evidente para transformar la regla del límite, la regla del máximo y la regla del umbral en reglas acerca de la notación $\Omega$.

A pesar de la fuerte similitud entre las notaciones $O$ y $\Omega$, existe un aspecto en el cual falla su dualidad. Recuérdese que lo que más frecuentemente nos interesa de los algoritmos es la eficiencia en el *caso peor*. Por tanto, cuando decimos que una implementación del algoritmo requiere $t(n)$ microsegundos, queremos decir que $t(n)$ es el tiempo máximo que requiere esa implementación para todos los casos de tamaño $n$. Sea un $f(n)$ tal que $t(n) \in O(f(n))$. Esto significa que existe una constante real positiva $c$ tal que $t(n) \leq cf(n)$ para todo $n$ suficientemente grande. Dado que ningún ejemplar de tamaño $n$ puede tomar más tiempo que el tiempo máximo requerido por los ejemplares de ese tamaño, se sigue que la implementación requiere un tiempo acotado por $cf(n)$ microsegundos para todos los ejemplares suficientemente grandes. Suponiendo que solamente exista un número finito de ejemplares de tamaño, puede haber solamente un número finito de ejemplares, todos ellos de tamaño menor que el umbral, sobre los cuales la implementación requiera un tiempo mayor que $cf(n)$ microsegundos. Suponiendo que $f(n)$ nunca sea cero, todos éstos se pueden resolver utilizando una constante multiplicativa mayor, tal como se hace en la demostración de la regla del umbral.

Por contraste, supongamos que $t(n) \in \Omega(f(n))$. Una vez más, esto significa que existe una constante real positiva $d$ tal que $t(n) \geq df(n)$ para todo $n$ suficientemente grande. Ahora bien, dado que $t(n)$ denota el comportamiento de esa implementación en el caso peor, podemos inferir solamente que, para cada $n$ suficientemente grande, existe al menos *un ejemplar* de tamaño $n$ tal que la implementación requiere al menos $df(n)$ microsegundos para ese ejemplo. Esto no excluye la posibilidad de un comportamiento mucho más rápido sobre otros ejemplares del mismo tamaño. Por tanto, pueden existir infinitos ejemplares en los cuales la implementación requiera menos de $df(n)$ microsegundos. La ordenación por inserción ofrece un ejemplo típico de este comportamiento. Vimos en la Sección 2.4 que se requiere un tiempo cuadrático en el caso peor, pero sin embargo existen infinitos ejemplares en los cuales se ejecuta en un tiempo lineal. Por tanto, tenemos derecho a manifestar que el tiempo de ejecución en el caso peor está tanto en $O(n^2)$ como en $\Omega(n^2)$. Sin embargo, la primera afirmación dice que todo caso suficientemente grande se podrá clasificar en un tiempo cuadrático, mientras que la segunda se limita a decir que al menos un ejemplar de cada tamaño suficientemente grande requiere realmente el tiempo cuadrático: el algoritmo puede ser más rápido en otros ejemplares del mismo tamaño.

Algunos autores definen la notación $\Omega$ en una forma que es diferente de forma sutil pero importante. Dicen que $t(n) \in \Omega(f(n))$ si existe una constante real y positiva $d$ tal que $t(n) \geq df(n)$ para un *número infinito* de valores de $n$, mientras que nosotros requerimos que esta relación sea válida para todos los valores de $n$ salvo un

número finito. Con esta definición, un algoritmo que requiera un tiempo en $\Omega(f(n))$ en el caso peor es tal que existen infinitos casos en los cuales requiere al menos $df(n)$ microsegundos para la constante real positiva adecuada $d$. Esto se corresponde de forma más próxima a nuestra idea intuitiva de lo que debería ser una cota inferior para el rendimiento de un algoritmo. Resulta más natural que lo que *nosotros* queremos decir con «requerir un tiempo en $\Omega$ de $f(n)$». Sin embargo, preferimos nuestra definición porque es más fácil trabajar con ella. En particular, la definición modificada de $\Omega$ no es transitiva y la regla de dualidad falla.

En este libro, utilizamos la notación $\Omega$ sobre todo para proporcionar cotas inferiores acerca del tiempo de ejecución (u otros recursos) de los algoritmos. Sin embargo, esta notación suele utilizarse para proporcionar cotas inferiores acerca de la dificultad intrínseca de resolver ciertos problemas. Por ejemplo, veremos en la Sección 12.2.1 que *cualquier* algoritmo que ordene con éxito $n$ elementos debe requerir un tiempo en $\Omega(n \log n)$ siempre y cuando la única operación efectuada sobre los elementos que hay que ordenar consista en compararlos por parejas para determinar si son iguales, y, en caso contrario, cuál de ellos es el mayor; como resultado, diremos que el problema consistente en ordenar mediante comparaciones tiene una *complejidad* de tiempo de ejecución que está en $\Omega(n \log n)$. En general, resulta mucho más difícil determinar la complejidad de un problema que determinar una cota inferior sobre el tiempo de ejecución de un cierto algoritmo que lo resuelva. Hablaremos más sobre este tema en el Capítulo 12.

**La notación Theta.** Cuando analizamos el comportamiento de un algoritmo, nos sentimos especialmente felices si su tiempo de ejecución está acotado tanto por encima como por debajo mediante múltiplos reales positivos posiblemente distintos de una misma función. Por esta razón, presentamos la notación $\Theta$. Diremos que $t(n)$ está en Theta de $f(n)$, o lo que es lo mismo que $t(n)$ está en el *orden exacto* de $f(n)$, y lo denotamos $t(n) \in \Theta(f(n))$, si $t(n)$ pertenece tanto a $O(f(n))$ como a $\Omega(f(n))$. La definición formal de $\Theta$ es:

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n)).$$

Esto es equivalente a decir que $\Theta(f(n))$ es

$$\{t: \mathbb{N} \to \mathbb{R}^{\geq 0} \mid (\exists c,d \in \mathbb{R}^+)(\forall\ n \in \mathbb{N})\ [df(n) \leq t(n) \leq cf(n)]\}$$

La regla del umbral y la regla del máximo, que formulábamos en el contexto de la notación $O$, es aplicable *mutatis mutandis* a la notación $\Theta$. Curiosamente, la regla del límite para la notación $\Theta$ se reformula en la forma siguiente. Considérense las funciones arbitrarias $f$ y $g$: $\mathbb{N} \to \mathbb{R}^{\geq 0}$.

1. Si $\lim \frac{f(n)}{g(n)} \in \mathbb{R}^+$ entonces $f(n) \in \Theta(g(n))$

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

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

Como ejercicio de manipulación de la notación asintótica, demostraremos ahora un hecho útil:

$$\sum_{i=1}^{n} i^k \in \Theta(n^{k+1})$$

para cualquier entero $k \geq 0$, en donde la suma del lado izquierdo se considera como función de $n$. Por supuesto, éste hecho se sigue de forma inmediata de la proposición 1.7.16, pero resulta instructivo demostrarlo directamente.

La dirección «$O$» es fácil de probar. Para esto, obsérvese simplemente que $i^k \leq n^k$ siempre que $1 \leq i \leq n$. Por tanto, $\sum_{i=1}^{n} i^k \leq \sum_{i=1}^{n} n^k = n^{k+1}$ para todo $n \geq 1$, lo cual demuestra que $\sum_{i=1}^{n} i^k \in O(n^{k+1})$ utilizando 1 como constante multiplicativa.

Para demostrar la dirección «$\Omega$», obsérvese que $i^k \geq (n/2)^k$ siempre que $i \geq \lceil n/2 \rceil$ y que el número de enteros que hay entre $\lceil n/2 \rceil$ y $n$, ambos inclusive, es mayor que $n/2$. Por tanto, siempre que $n \geq 1$ (lo cual implica que $\lceil n/2 \rceil \geq 1$):

$$\sum_{i=1}^{n} i^k \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} i^k \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \left(\frac{n}{2}\right)^k \geq \frac{n}{2} \times \left(\frac{n}{2}\right)^k = \frac{n^{k+1}}{2^{k+1}}$$

Esto demuestra que $\sum_{i=1}^{n} i^k \in \Omega(n^{k+1})$ utilizando $1/2^{k+1}$ como constante multiplicativa. Se obtiene una constante más ajustada en el problema 3.23.

## 3.4 NOTACIÓN ASINTÓTICA CONDICIONAL

Hay muchos algoritmos que resultan más fáciles de analizar si en un principio limitamos nuestra atención a aquellos casos cuyo tamaño satisfaga una cierta condición, tal como ser una potencia de 2. Considérese por ejemplo el algoritmo «divide y vencerás» para multiplicar enteros grandes que veíamos en la Sección 1.2. Sea $n$ el tamaño de los enteros que hay que multiplicar (suponíamos que eran de un mismo tamaño). El algoritmo se ejecuta directamente si $n = 1$, lo cual requiere $a$ microsegundos para una constante apropiada $a$. Si $n > 1$, el algoritmo se ejecuta recursivamente multiplicando cuatro parejas de enteros de tamaño $\lceil n/2 \rceil$ ( o tres parejas en el algoritmo mejorado que estudiaremos en el Capítulo 7). Además, es precisa una cantidad lineal de tiempo para efectuar tareas adicionales. Por sencillez, digamos que el trabajo adicional requiere de $n$ microsegundos para una constante adecuada $b$ (para ser exactos, sería necesario un tiempo entre $b_1 n$ y $b_2 n$ microsegundos para las constantes adecuadas $b_1$ y $b_2$ (hablaremos más acerca de esto en la Sección 4.7.6).

El tiempo que requiere este algoritmo está dado consiguientemente por la función $t: \mathbb{N} \to \mathbb{R}^{\geq 0}$ que se define recursivamente en la forma

$$t(n) = \begin{cases} a & \text{si } n = 1 \\ 4t(\lceil n/2 \rceil) + bn & \text{en caso contrario} \end{cases} \tag{3.2}$$

En la Sección 4.7 estudiaremos técnicas para resolver recurrencias, pero desafortunadamente la ecuación 3.2 no se puede manejar directamente mediante esas técnicas porque la función $\lceil n/2 \rceil$ es dificultosa. Sin embargo, nuestra recurrencia es fácil de resolver siempre y cuando consideremos solamente el caso en el cual $n$ es una potencia de 2: en este caso $\lceil n/2 \rceil = n/2$ y el redondeo por exceso desaparece. Las técnicas de la Sección 4.7 producen

$$t(n) = (a+b)n^2 - bn$$

*siempre y cuando «n»* sea una potencia de 2. Dado que el término de orden inferior «$-bn$» se puede despreciar, se sigue que $t(n)$ es del orden exacto de $n^2$, siempre y cuando, insistimos, $n$ sea una potencia de 2. Esto se denota mediante $t(n) \in \Theta(n^2 \mid n$ es una potencia de 2).

Más generalmente, sean $f, t: \mathbb{N} \to \mathbb{R}^{\geq 0}$ dos funciones de los números naturales en los números reales no negativos, y sea $P: \mathbb{N} \to \{verdadero, falso\}$ una propiedad de los enteros. Decimos que $t(n)$ está en $O(f(n) \mid P(n))$ si $t(n)$ está acotado superiormente por un múltiplo real positivo de $f(n)$ para todo $n$ suficientemente grande tal que $P(n)$ sea válido. Formalmente, $O(f(n) \mid P(n))$ se define en la forma

$$\{t: \mathbb{N} \to \mathbb{R}^{\geq 0} \mid (\exists c \in \mathbb{R}^{+})\ (\overset{\infty}{\forall} n \in \mathbb{N})\ [P(n) \Rightarrow t(n) \leq cf(n)]\}$$

Los conjuntos $\Omega(f(n) \mid P(n))$ y $\Theta(f(n) \mid P(n))$ se definen de forma similar. Abusando de la notación en una forma familiar escribiremos que $t(n) \in O(f(n) \mid P(n))$ incluso si $t(n)$ y $f(n)$ son negativas o no están definidas en un cierto número de puntos arbitrario —quizá infinito— de $n$ en los cuales $P(n)$ no es válida.

La notación asintótica condicional es algo más que una mera comodidad de notación: su interés principal es que se puede eliminar, en general, una vez que ha sido utilizada para facilitar el análisis de un algoritmo. Para verlo, necesitamos unas cuantas definiciones. Una función $f: \mathbb{N} \to \mathbb{R}^{\geq 0}$ es *asintóticamente no decreciente* si existe un umbral entero $n_0$ tal que $f(n) \leq f(n+1)$ para todo $n \geq n_0$. Esto implica por inducción matemática que $f(n) \leq f(m)$ siempre que $m \geq n \geq n_0$. Sea $b \geq 2$ cualquier entero. Se dice que la función $f$ es $b$-uniforme[1], si, además de ser asintóticamente no decreciente, satisface la condición $f(bn) \in O(f(n))$. En otras palabras, tiene que existir una constante $c$ (dependiente de $b$) tal que $f(bn) \leq cf(n)$ para todo $n \geq n_0$. (No hay pérdida de generalidad al utilizar un mismo umbral $n_0$ para los dos propósitos.) Una función se dice *uniforme* o de *ajuste* si es $b$-uniforme para todo entero $b \geq 2$.

La mayoría de las funciones que se suele uno encontrar en el análisis de algoritmos son uniformes (de ajuste, suaves), tal como $\log n$, $n \log n$, $n^2$ o cualquier polinomio cuyo primer coeficiente sea positivo. Sin embargo, las funciones que crecen demasiado deprisa tal como $n^{\lg n}$, $2^n$ o $n!$, no son suaves porque la razón $f(2n)/f(n)$ no está acotada. Por ejemplo:

$$(2n)^{\lg(2n)} = 2n^2\, n^{\lg n}$$

lo cual muestra que $(2n)^{\lg(2n)} \notin O(n^{\lg n})$ porque $2n^2$ no se puede acotar superiormente mediante una constante. Por otra parte, las funciones que están acotadas superiormente mediante algún polinomio, suelen ser suaves siempre que sean asintóticamente no decrecientes; e incluso si no son asintóticamente no decrecientes existen buenas posibilidades de que estén en el orden exacto de alguna otra función que sea suave. Por ejemplo, supongamos que $b(n)$ denota el número de *bits* iguales a uno en la expansión binaria de $n$ (tal como $b(13) = 3$ porque 13 se escribe 1101 en binario) y consideremos $f(n) = b(n) + \lg n$. Es fácil ver que $f(n)$ no es eventualmente no decreciente —y por tanto, que no es uniforme— porque $b(2^k - 1) = k$ mientras que $b(2^k) = 1$ para todo $k$. Sin embargo, $f(n) \in \Theta(\log n)$, que es una función uniforme. (Este ejemplo no es tan artificial como pudiera parecer; véase la Sección 7.8.) En raras ocasiones se encontrarán funciones de crecimiento lento que no estén en el orden exacto de una función suave.

Una propiedad útil de la uniformidad (ajuste o suavización) es que si $f$ es $b$-uniforme para algún entero concreto $b \geq 2$, entonces, de hecho, es uniforme. Para demostrar esto, considérense dos enteros cualesquiera $a$ y $b$ mayores o iguales que 2. Supongamos que $f$ es $b$-uniforme. Debemos demostrar que $f$ es también $a$-uniforme. Sean $c$ y $n_0$ constantes tales que $f(bn) \leq c\, f(n)$ y $f(n) \leq f(n+1)$ para todo $n \geq n_0$. Sea $i = \lceil \log_b a \rceil$. Por definición del logaritmo, $a = b^{\log_b a} \leq b^{\lceil \log_b a \rceil} = b^i$. Considérse cualquier $n \geq 0$. Es fácil mostrar por inducción matemática sobre la $b$-uniformidad de $f$ que $f(b^i n) \leq c^i f(n)$. Pero $f(an) \leq f(b^i n)$ (porque $f$ es asintomáticamente no decreciente y $b^i n \geq an \geq n_0$. Se sigue que $f(an) \leq \hat{c}\, f(n)$ para $\hat{c} = c^i$, y por tanto $f$ es $a$-uniforme.

Las funciones uniformes son interesantes como consecuencia de la **regla de la uniformidad** (ajuste o suavidad). Sea $f$: $\mathbb{N}^+ \to \mathbb{R}^{\geq 0}$ una función suave y sea $t$: $\mathbb{N}^+ \to \mathbb{N}^+$ una función asintóticamente no decreciente. Considérese cualquier entero $b \geq 2$. La regla de uniformidad afirma que $t(n) \in \Theta(f(n))$ siempre que $t(n) \in \Theta(f(n) \mid n$ es una potencia de $b)$. La regla es aplicable igualmente a la notación $O$ y a la notación $\Omega$. Antes de demostrar la regla, la ilustraremos con el ejemplo utilizado al principio de esta sección.

Ya vimos que es fácil obtener la fórmula condicional asintótica

$$t(n) \in \Theta(n^2 \mid n \text{ es una potencia de } 2) \tag{3.3}$$

a partir de la ecuación 3.2, mientras que resulta más difícil efectuar el análisis de $t(n)$ cuando $n$ no es una potencia de 2. La regla de la uniformidad nos permite in-

ferir directamente de la ecuación 3.3 que $t(n) \in \Theta(n^2)$ siempre y cuando especifiquemos que $n^2$ es una función suave y que $t(n)$ es asintóticamente no decreciente. La primera condición es inmediata, por cuanto $n^2$ es obviamente no decreciente y $(2n)^2 = 4n^2$. La segunda se demuestra fácilmente por inducción matemática sobre la ecuación 3.2; véase el problema 3.28. Por tanto, el uso de la notación condicional asintótica como paso intermedio da lugar al resultado final consistente en que $t(n) \in \Theta(n^2)$ incondicionalmente.

Demostraremos ahora la regla de uniformidad. Sea $f(n)$ una función uniforme y sea $t(n)$ una función eventualmente no decreciente tal que $t(n) \in \Theta(f(n) \mid n$ es una potencia de $b$) para algún entero $b \geq 2$. Sea $n_0$ el mayor de los umbrales implicado por las condiciones anteriores: $f(m) \leq f(m+1)$, $t(m) \leq t(m+1)$ y $f(bm) \leq cf(m)$ siempre que $m \geq n_0$ y $df(m) \leq t(m) \leq af(m)$ siempre que $m \geq n_0$ sea una potencia de $b$, para constantes adecuadas $a$,$c$ y $d$. Para todo entero positivo $n$, sea $\underline{n}$ la mayor potencia de $b$ que no es mayor que $n$ (formalmente, $\underline{n} = b^{\lfloor \log_b n \rfloor}$) y sea $\overline{n} = b\underline{n}$. Por definición, $n/b < \underline{n} \leq n < \overline{n}$ y $\overline{n}$ es una potencia de $b$. Considérese cualquier $n \geq \max(1, bn_0)$.

$$t(n) \leq t(\overline{n}) \leq af(\overline{n}) = af(b\underline{n}) \leq acf(\underline{n}) \leq acf(n)$$

Esta ecuación emplea sucesivamente los hechos consistentes en que $t$ es eventualmente no decreciente (y $\overline{n} \geq n \geq n_0$), $t(m)$ es del orden de $f(m)$ cuando $m$ es una potencia de $b$ (y $\overline{n}$ es una potencia de $b$ mayor o igual que $n_0$), $\overline{n} = b\underline{n}$, $f$ es $b$-uniforme (y $\underline{n} > n/b \geq n_0$), y $f$ es asintóticamente no decreciente ($n \geq \underline{n} > n_0$). Esto demuestra que $t(n) \leq acf(n)$ para *todos* los valores de $n \geq \max(1, bn_0)$, y por tanto $t(n) \in O(f(n))$. La demostración de que $t(n) \in \Omega(f(n))$ es parecida.

## 3.5 NOTACIÓN ASINTÓTICA CON VARIOS PARÁMETROS

Puede suceder, cuando se analiza un algoritmo, que su tiempo de ejecución dependa simultáneamente de más de un parámetro del ejemplar en cuestión. Esta situación es típica de ciertos algoritmos para problemas de grafos, por ejemplo, en los cuales el tiempo depende tanto del número de nodos como del número de aristas. En tales casos la noción «tamaño del ejemplar» que se ha utilizado hasta el momento, puede perder gran parte de su significado. Por esta razón, se generaliza la notación asintótica de forma natural para funciones de varias variables.

Sea $f: \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{\geq 0}$ una función de parejas de números naturales en los reales no negativos, tal como $f(m, n) = m \log n$. Sea $t: \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{\geq 0}$ otra función de éstas. Diremos que $t(m, n)$ es del orden de $f(m, n)$, lo cual se denota $t(m, n) \in O(f(m, n))$ si $t(m, n)$ está acotada superiormente por un múltiplo positivo de $f(m, n)$ siempre que tanto $m$ como $n$ sean suficientemente grandes. Formalmente, $O(f(m, n))$ se define en la forma

$$\{t: \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{\geq 0} \mid (\exists c \in \mathbb{R}^+)\ (\forall\ m, n \in \mathbb{N})\ [t(m, n) \leq cf(m, n)]\}$$

(No hay necesidad de utilizar dos umbrales distintos para $m$ y $n$ en $\forall\, m, n \in$ .) La generalización a más de dos parámetros, de la notación asintótica condicional y de las notaciones $\Theta$ y $\Omega$, se hace de forma similar.

La notación asintótica con varios parámetros es parecida a lo que hemos visto hasta el momento, salvo por una diferencia esencial: la regla del umbral ya no es válida. De hecho, el umbral es indispensable en algunas ocasiones. La razón es que aun cuando nunca hay más de un número finito de números no negativos que sean menores que cualquier umbral dado, existe en general un número infinito de parejas $\langle m, n \rangle$ de números no negativos tales que o bien $m$ o bien $n$ está por debajo del umbral; véase el problema 3.32. Por esta razón, $O(f(m, n))$ puede tener sentido aunque $f(m, n)$ sea negativa o no esté definida para un conjunto infinito de puntos, siempre y cuando todos estos puntos se puedan descartar mediante una selección adecuada del umbral.

## 3.6 OPERACIONES SOBRE NOTACIÓN ASINTÓTICA

Para simplificar algunos cálculos, podemos manipular la notación asintótica empleando operadores aritméticos. Por ejemplo, $O(f(m)) + O(g(n))$ representa el conjunto de operaciones obtenidas sumando punto a punto toda función de $O(f(n))$ a cualquier función de $O(g(n))$. Intuitivamente este conjunto representa el orden del tiempo requerido por un algoritmo compuesto por una primera fase que requiere un tiempo del orden de $f(n)$ seguido por una segunda fase que requiera un tiempo del orden de $g(n)$. Hablando con propiedad, las constantes ocultas que multiplican a $f(n)$ y $g(n)$ pueden muy bien ser distintas, pero esto no tiene importancia porque es sencillo demostrar que $O(f(n)) + O(g(n))$ es idéntico a $O(f(n) + g(n))$. Por la regla del máximo, sabemos que esto también es lo mismo que $O(\max(f(n), g(n)))$ y, si se prefiere, $\max(O(f(n)), O(g(n)))$.

Más formalmente si **op** es un operador binario y si $X$ e $Y$ son conjuntos de funciones de en , en particular conjuntos descritos mediante la notación asintótica, entonces «$X$ **op** $Y$» denota el conjunto de funciones que se pueden obtener seleccionando una función de $X$ y una función de $Y$, y «**op**-erando» uno con otro punto a punto. Siguiendo el espíritu de la notación asintótica, solamente exigimos que la función resultante sea el valor **op**erado correcto más allá de un cierto umbral. Formalmente, $X$ **op** $Y$ denota

$$\{t: \to \mid (\exists f \in X)\ (\exists g \in Y)\ (\forall\, n \in )\ [t(n) = f(n)\ \mathbf{op}\ g(n)]\}$$

Si $g$ es una función de en , extenderemos la notación escribiendo $g$ **op** $X$ para denotar $\{g\}$ **op** $X$, el conjunto de funciones que se pueden obtener **op**erando la función $g$ con una función de $X$. Además, si $a \in$ utilizamos $a$ **op** $X$ para denotar $cst_a$ **op** $X$, donde $cst_a$ denota la función constante $cst_a(n) = a$ para todo entero $n$. En otras palabras, $a$ **op** $X$ denota el conjunto de funciones que se pueden obtener **op**erando $a$ con el valor de una función de $X$. También denotaremos la notación

simétrica $X$ **op** $g$ y $X$ **op** $a$, y toda esta teoría de operaciones sobre conjuntos se extiende de forma evidente a los operadores que no sean binarios y a las funciones.

Como ejemplo, consideremos el significado de $n^{O(1)}$. Aquí, «$n$» y «1» denotan la función identidad $Id(n)=n$ y la función $cst_1(n)=1$, respectivamente. Por tanto, una función $t(n)$ pertenece a $n^{O(1)}$ si existe una función $f(n)$ acotada superiormente por una constante $c$ tal que $t(n) = n^{f(n)}$ para todo $n \geq n_0$ para algún $n_0$. En particular esto implica que $t(n) \leq k + n^k$ para todo $n \geq 0$, siempre y cuando $k \geq c$ y $k \geq t(n)$ para todo $n < n_0$. Por tanto, toda función de $n^{O(1)}$ está acotada superiormente por un polinomio. A la inversa, considérese cualquier polinomio $p(n)$ cuyo primer coeficiente sea positivo y cualquier función $t(n)$ tal que $1 \leq t(n) \leq p(n)$ para todo $n$ suficientemente grande. Sea $k$ el grado de $p(n)$. Es fácil demostrar que $p(n) \leq n^{k+1}$ para tanto, $g(n) \in O(1)$, lo cual demuestra que $t(n) \in n^{O(1)}$. La conclusión es que $n^{O(1)}$ reque $1 \leq t(n) \leq n^{k+1}$, se sigue que $0 \leq g(n) \leq k + 1$ para $n$ suficientemente grande. Por tanto, $g(n) \in O(1)$, lo cual demuestra que $t(n) \in n^{o(1)}$. La conclusión es que $n^{o(1)}$ representa el conjunto de todas las funciones acotadas superiormente por algún polinomio, siempre y cuando la función esté acotada inferiormente por la constante uno para todo $n$ suficientemente grande.

## 3.7 PROBLEMAS

**Problema 3.1.** Considérese una implementación del algoritmo que requiera un tiempo que esté acotado superiormente por la improbable función

$$t(n) = 3 \text{ segundos} - 18n \text{ milisegundos} + 27n^2 \text{ microsegundos}$$

para resolver un ejemplar de tamaño $n$. Hallar la función más sencilla posible $f: \mathbb{N} \to \mathbb{R}^{\geq 0}$ tal que el algoritmo requiera un tiempo en el orden de $f(n)$.

**Problema 3.2.** Considérense dos algoritmos $A$ y $B$ que requieren un tiempo en $\Theta(n^2)$ y $\Theta(n^3)$, respectivamente, para resolver el mismo problema. Si los demás recursos, tal como el espacio de almacenamiento y el tiempo de programación no son relevantes, ¿podremos estar seguros de que el algoritmo $A$ siempre será preferible al algoritmo $B$? Justifique su respuesta.

**Problema 3.3.** Considérense dos algoritmos $A$ y $B$ que requieran un tiempo en $O(n^2)$ y en $O(n^3)$, respectivamente. ¿Podría existir una implementación del algoritmo $B$ que fuera más eficiente (en términos de tiempo de ejecución) que una implementación del algoritmo $A$ para *todos* los ejemplares? Justifique su respuesta.

**Problema 3.4.** ¿Qué significa $O(1)$? ¿Qué significa $\Theta(1)$?

**Problema 3.5.** ¿Cuáles de las siguientes afirmaciones son verdaderas? Demuestre sus respuestas.

1. $n^2 \in O(n^3)$
2. $n^2 \in \Omega(n^3)$
3. $2^n \in \Theta(2^{n+1})$
4. $n! \in \Theta((n+1)!)$

**Problema 3.6.** Demuestre que si $f(n) \in O(n)$ entonces $[f(n)]^2 \in O(n^2)$.

**Problema 3.7.** En contraste con el problema 3.6, demostrar que $2^{f(n)} \in O(2^n)$ no se sigue necesariamente de $f(n) \in O(n)$.