Big O notation

From Wikipedia, the free encyclopedia

In mathematics, computer science, and related fields, **big-O notation** (also known as **big Oh notation**, **big Omicron notation**, **Landau notation**, **Bachmann–Landau notation**, and **asymptotic notation**) (along with the closely related *big-Omega notation*, *big-Theta notation*, and *little o notation*) describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions. Big O notation characterizes functions according to their growth rates: different functions with the same growth rate may be represented using the same O notation.

Although developed as a part of pure mathematics, this notation is now frequently also used in the analysis of algorithms to describe an algorithm's usage of computational resources: the worst case or average case running time or memory usage of an algorithm is often expressed as a function of the length of its input using big O notation. This allows algorithm designers to predict the behavior of their algorithms and to determine which of multiple algorithms to use, in a way that is independent of computer architecture or clock rate. Because big O notation discards multiplicative constants on the running time, and ignores efficiency for low input sizes, it does not always reveal the fastest algorithm in practice or for practically-sized data sets, but the approach is still very effective for comparing the scalability of various algorithms as input sizes become large.

A description of a function in terms of big O notation usually only provides an upper bound on the growth rate of the function. Associated with big O notation are several related notations, using the symbols o, Ω, ω, and Θ, to describe other kinds of bounds on asymptotic growth rates. Big O notation is also used in many other fields to provide similar estimates.

Contents

Formal definition

Let $f(x)$ and $g(x)$ be two functions defined on some subset of the real numbers. One writes

$$f(x) = O(g(x)) \text{ as } x \to \infty$$

if and only if, for sufficiently large values of x, $f(x)$ is at most a constant multiplied by $g(x)$ in absolute value. That is, $f(x) = O(g(x))$ if and only if there exists a positive real number M and a real number x_0 such that

$$|f(x)| \leq M|g(x)| \text{ for all } x > x_0.$$

In many contexts, the assumption that we are interested in the growth rate as the variable x goes to infinity is left unstated, and one writes more simply that $f(x) = O(g(x))$.

The notation can also be used to describe the behavior of f near some real number a (often, $a = 0$): we say

$$f(x) = O(g(x)) \text{ as } x \to a$$

if and only if there exist positive numbers δ and M such that

$$|f(x)| \leq M|g(x)| \text{ for } |x - a| < \delta.$$

If $g(x)$ is non-zero for values of x sufficiently close to a, both of these definitions can be unified using the limit superior:

$$f(x) = O(g(x)) \text{ as } x \to a$$

if and only if

$$\limsup_{x \to a} \left| \frac{f(x)}{g(x)} \right| < \infty.$$

Example

In typical usage, the formal definition of O notation is not used directly; rather, the O notation for a function $f(x)$ is derived by the following simplification rules:

- If $f(x)$ is a sum of several terms, the one with the largest growth rate is kept, and all others omitted.
- If $f(x)$ is a product of several factors, any constants (terms in the product that do not depend on x) are omitted.

For example, let $f(x) = 6x^4 - 2x^3 + 5$, and suppose we wish to simplify this function, using O notation, to describe its growth rate as x approaches infinity. This function is the sum of three terms: $6x^4$, $-2x^3$, and 5. Of these three terms, the one with the highest growth rate is the one with the largest exponent as a function of x, namely $6x^4$. Now one may apply the second rule: $6x^4$ is a product of 6 and x^4 in which the first factor does not depend on x. Omitting this factor results in the simplified form x^4. Thus, we say that $f(x)$ is a big-oh of (x^4) or mathematically we can write $f(x) = O(x^4)$.

One may confirm this calculation using the formal definition: let $f(x) = 6x^4 - 2x^3 + 5$ and $g(x) = x^4$. Applying the formal definition from above, the statement that $f(x) = O(x^4)$ is equivalent to its expansion,

$$|f(x)| \leq M|g(x)|$$

for some suitable choice of x_0 and M and for all $x > x_0$. To prove this, let $x_0 = 1$ and $M = 13$. Then, for all $x > x_0$:

$$\begin{aligned} |6x^4 - 2x^3 + 5| &\leq 6x^4 + |2x^3| + 5 \\ &\leq 6x^4 + 2x^4 + 5x^4 \\ &\leq 13x^4, \\ &\leq 13|x^4| \end{aligned}$$

so

$$|6x^4 - 2x^3 + 5| \leq 13\,|x^4|.$$

Usage

Big O notation has two main areas of application. In mathematics, it is commonly used to describe how closely a finite series approximates a given function, especially in the case of a truncated Taylor series or asymptotic expansion. In computer science, it is useful in the analysis of algorithms. In both applications, the function $g(x)$ appearing within the $O(...)$ is typically chosen to be as simple as possible, omitting constant factors and lower order terms.

There are two formally close, but noticeably different, usages of this notation: infinite asymptotics and infinitesimal asymptotics. This distinction is only in application and not in principle, however—the formal definition for the "big O" is the same for both cases, only with different limits for the function argument.

Infinite asymptotics

Big O notation is useful when analyzing algorithms for efficiency. For example, the time (or the number of steps) it takes to complete a problem

of size n might be found to be $T(n) = 4n^2 - 2n + 2$.

As n grows large, the n^2 term will come to dominate, so that all other terms can be neglected — for instance when $n = 500$, the term $4n^2$ is 1000 times as large as the $2n$ term. Ignoring the latter would have negligible effect on the expression's value for most purposes.

Further, the coefficients become irrelevant if we compare to any other order of expression, such as an expression containing a term n^3 or n^4. Even if $T(n) = 1{,}000{,}000n^2$, if $U(n) = n^3$, the latter will always exceed the former once n grows larger than 1,000,000 ($T(1{,}000{,}000) = 1{,}000{,}000^3 = U(1{,}000{,}000)$). Additionally, the number of steps depends on the details of the machine model on which the algorithm runs, but different types of machines typically vary by only a constant factor in the number of steps needed to execute an algorithm.

So the big O notation captures what remains: we write either

$$T(n) = O(n^2)$$

or

$$T(n) \in O(n^2)$$

and say that the algorithm has *order of* n^2 time complexity.

Note that "=" is not meant to express "is equal to" in its normal mathematical sense, but rather a more colloquial "is", so the second expression is technically accurate (see the "Equals sign" discussion below) while the first is a common abuse of notation.[1]

Note that when used in computer science the domain of the expressions involved is the set of natural numbers and the functions f and g take only on finite values (i.e., $f(100)=\infty$ is not allowed). In such uses, $f(n)=O(g(n))$ implies (and is actually equivalent to) that $|f(n) / g(n)|$ is bounded by a constant no matter how the natural number n is selected. That is, in this case $f(n)=O(g(n))$ implies that there exists some constant $C>0$ such that $|f(n)| \leq C|g(n)|$ holds for all natural numbers n. This implication does not hold in the more general case when the domain of f and g may contain a finite accumulation point.

Infinitesimal asymptotics

Big O can also be used to describe the error term in an approximation to a mathematical function. The most significant terms are written explicitly, and then the least-significant terms are summarized in a single big O term. For example,

$$e^x = 1 + x + \frac{x^2}{2} + O(x^3) \qquad \text{as } x \to 0$$

expresses the fact that the error, the difference $e^x - (1 + x + x^2/2)$, is smaller in absolute value than some constant times $|x^3|$ when x is close enough to 0.

Properties

If a function $f(n)$ can be written as a finite sum of other functions, then the fastest growing one determines the order of $f(n)$. For example

$$f(n) = 9\log n + 5(\log n)^3 + 3n^2 + 2n^3 \in O(n^3).$$

In particular, if a function may be bounded by a polynomial in n, then as n tends to *infinity*, one may disregard *lower-order* terms of the polynomial.

$O(n^c)$ and $O(c^n)$ are very different. The latter grows much, much faster, no matter how big the constant c is (as long as it is greater than one). A function that grows faster than any power of n is called *superpolynomial*. One that grows more slowly than any exponential function of the form c^n is called *subexponential*. An algorithm can require time that is both superpolynomial and subexponential; examples of this include the fastest known algorithms for integer factorization.

$O(\log n)$ is exactly the same as $O(\log(n^c))$. The logarithms differ only by a constant factor (since $\log(n^c) = c\log n$) and thus the big O notation ignores that. Similarly, logs with different constant bases are equivalent. Exponentials with different bases, on the other hand, are not of the same order. For example, 2^n and 3^n are **not** of the same order.

Changing units may or may not affect the order of the resulting algorithm. Changing units is equivalent to multiplying the appropriate variable by a constant wherever it appears. For example, if an algorithm runs in the order of n^2, replacing n by cn means the algorithm runs in the order of c^2n^2, and the big O notation ignores the constant c^2. This can be written as $c^2n^2 \in O(n^2)$. If, however, an algorithm runs in the order of 2^n, replacing n with cn gives $2^{cn} = (2^c)^n$. This is not equivalent to 2^n in general.

Changing of variable may affect the order of the resulting algorithm. For example, if an algorithm's running time is $O(n)$ when measured in

terms of the number n of *digits* of an input number x, then its running time is $O(\log x)$ when measured as a function of the input number x itself, because $n = \Theta(\log x)$.

Product

$$f_1 \in O(g_1) \text{ and } f_2 \in O(g_2) \Rightarrow f_1 f_2 \in O(g_1 g_2)$$

$$f \cdot O(g) \subset O(fg)$$

Sum

$$f_1 \in O(g_1) \text{ and } f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(|g_1| + |g_2|)$$

This implies $f_1 \in O(g) \text{ and } f_2 \in O(g) \Rightarrow f_1 + f_2 \in O(g)$, which means that $O(g)$ is a convex cone.

If f and g are positive functions, $f + O(g) \in O(f + g)$

Multiplication by a constant

Let k be a constant. Then:

$O(kg) = O(g)$ if k is nonzero.

$f \in O(g) \Rightarrow kf \in O(g).$

Multiple variables

Big O (and little o, and Ω…) can also be used with multiple variables.

To define Big O formally for multiple variables, suppose $f(\vec{x})$ and $g(\vec{x})$ are two functions defined on some subset of $\mathbb{R}^n$. We say

$$f(\vec{x}) \text{ is } O(g(\vec{x})) \text{ as } \vec{x} \to \infty$$

if and only if

$$\exists C \, \exists M > 0 \text{ such that } |f(\vec{x})| \leq C|g(\vec{x})| \text{ for all } \vec{x} \text{ with } x_i > M \text{ for all } i.$$

For example, the statement

$$f(n, m) = n^2 + m^3 + \mathrm{O}(n + m) \text{ as } n, m \to \infty$$

asserts that there exist constants C and M such that

$$\forall n, m > M : |g(n, m)| \leq C(n + m),$$

where $g(n,m)$ is defined by

$$f(n, m) = n^2 + m^3 + g(n, m).$$

Note that this definition allows all of the coordinates of $\vec{x}$ to increase to infinity. In particular, the statement

$$f(n, m) = \mathrm{O}(n^m) \text{ as } n, m \to \infty$$

(i.e., $\exists C \, \exists M \, \forall n \, \forall m \ldots$) is quite different from

$$\forall m : f(n, m) = \mathrm{O}(n^m) \text{ as } n \to \infty$$

(i.e., $\forall m \, \exists C \, \exists M \, \forall n \ldots$).

Matters of notation

Equals sign

The statement "$f(x)$ is $O(g(x))$" as defined above is usually written as $f(x) = O(g(x))$. Some consider this to be an abuse of notation, since the use of the equals sign could be misleading as it suggests a symmetry that this statement does not have. As de Bruijn says, $O(x) = O(x^2)$ is true but $O(x^2) = O(x)$ is not.[2] Knuth describes such statements as "one-way equalities", since if the sides could be reversed, "we could deduce

ridiculous things like $n = n^2$ from the identities $n = O(n^2)$ and $n^2 = O(n^2)$."[3]

For these reasons, it would be more precise to use set notation and write $f(x) \in O(g(x))$, thinking of $O(g(x))$ as the class of all functions $h(x)$ such that $|h(x)| \leq C|g(x)|$ for some constant C.[3] However, the use of the equals sign is customary. Knuth pointed out that "mathematicians customarily use the = sign as they use the word 'is' in English: Aristotle is a man, but a man isn't necessarily Aristotle."[4]

Other arithmetic operators

Big O notation can also be used in conjunction with other arithmetic operators in more complicated equations. For example, $h(x) + O(f(x))$ denotes the collection of functions having the growth of $h(x)$ plus a part whose growth is limited to that of $f(x)$. Thus,

$$g(x) = h(x) + O(f(x))$$

expresses the same as

$$g(x) - h(x) \in O(f(x)).$$

Example

Suppose an algorithm is being developed to operate on a set of n elements. Its developers are interested in finding a function $T(n)$ that will express how long the algorithm will take to run (in some arbitrary measurement of time) in terms of the number of elements in the input set. The algorithm works by first calling a subroutine to sort the elements in the set and then perform its own operations. The sort has a known time complexity of $O(n^2)$, and after the subroutine runs the algorithm must take an additional $55n^3 + 2n + 10$ time before it terminates. Thus the overall time complexity of the algorithm can be expressed as

$$T(n) = O(n^2) + 55n^3 + 2n + 10.$$

This can perhaps be most easily read by replacing $O(n^2)$ with "some function that grows asymptotically slower than n^2 ". Again, this usage disregards some of the formal meaning of the "=" and "+" symbols, but it does allow one to use the big O notation as a kind of convenient placeholder.

Declaration of variables

Another feature of the notation, although less exceptional, is that function arguments may need to be inferred from the context when several variables are involved. The following two right-hand side big O notations have dramatically different meanings:

$$f(m) = O(m^n),$$
$$g(n) = O(m^n).$$

The first case states that $f(m)$ exhibits polynomial growth, while the second, assuming $m > 1$, states that $g(n)$ exhibits exponential growth.

To avoid confusion, some authors use the notation

$$g \in O(f),$$

rather than the less explicit

$$g(x) \in O(f(x)).$$

Complex usages

In more complex usage, $O(\ldots)$ can appear in different places in an equation, even several times on each side. For example, the following are true for $n \to \infty$

$$(n+1)^2 = n^2 + O(n)$$
$$(n + O(n^{1/2}))(n + O(\log n))^2 = n^3 + O(n^{5/2})$$
$$n^{O(1)} = O(e^n).$$

The meaning of such statements is as follows: for *any* functions which satisfy each $O(\ldots)$ on the left side, there are *some* functions satisfying each $O(\ldots)$ on the right side, such that substituting all these functions into the equation makes the two sides equal. For example, the third equation above means: "For any function $f(n) = O(1)$, there is some function $g(n) = O(e^n)$ such that $n^{f(n)} = g(n)$." In terms of the "set notation" above, the meaning is that the class of functions represented by the left side is a subset of the class of functions represented by the right side.

Orders of common functions

Here is a list of classes of functions that are commonly encountered when analyzing the running time of an algorithm. In each case, c is a constant and n increases without bound. The slower-growing functions are generally listed first.

See table of common time complexities for a more comprehensive list.

Notation	Name	Example
$O(1)$	constant	Determining if a number is even or odd; using a constant-size lookup table or hash table
$O(\log n)$	logarithmic	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap.
$O(n^c),\ 0 < c < 1$	fractional power	Searching in a kd-tree
$O(n)$	linear	Finding an item in an unsorted list or a malformed tree (worst case) or in an unsorted array; Adding two n-bit integers by ripple carry.
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, or quasilinear	Performing a Fast Fourier transform; heapsort, quicksort (best and average case), or merge sort
$O(n^2)$	quadratic	Multiplying two n-digit numbers by a simple algorithm; bubble sort (worst case or naive implementation), shell sort, quicksort (worst case), selection sort or insertion sort
$O(n^c),\ c > 1$	polynomial or algebraic	Tree-adjoining grammar parsing; maximum matching for bipartite graphs
$L_n[\alpha, c],\ 0 < \alpha < 1 = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$	L-notation or sub-exponential	Factoring a number using the quadratic sieve or number field sieve
$O(c^n),\ c > 1$	exponential	Finding the (exact) solution to the traveling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute-force search
$O(n!)$	factorial	Solving the traveling salesman problem via brute-force search; finding the determinant with expansion by minors.

The statement $f(n) = O(n!)$ is sometimes weakened to $f(n) = O\left(n^n\right)$ to derive simpler formulas for asymptotic complexity.

For any $k > 0$ and $c > 0$, $O(n^c(\log n)^k)$ is a subset of $O\left(n^{c+\varepsilon}\right)$ for any $\varepsilon > 0$, so may be considered as a polynomial with some bigger order.

Related asymptotic notations

Big O is the most commonly used asymptotic notation for comparing functions, although in many cases Big O may be replaced with Big Theta Θ for asymptotically tighter bounds. Here, we define some related notations in terms of Big O, progressing up to the family of Bachmann–Landau notations to which Big O notation belongs.

Little-o notation

The relation $f(x) \in o(g(x))$ is read as "$f(x)$ is little-o of $g(x)$". Intuitively, it means that $g(x)$ grows much faster than $f(x)$, or similarly, the growth of $f(x)$ is nothing compared to that of $g(x)$. It assumes that f and g are both functions of one variable. Formally, it states

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0.$$

Or alternatively:

Let $f(x)$ and $g(x)$ be two functions defined on some subset of the real numbers. One writes

$$f(x) = o(g(x)) \text{ as } x \to \infty$$

if and only if, for each positive constant M, $f(x)$ is at most M multiplied by $g(x)$ in absolute value, for each x, which is large enough. That is, $f(x) = o(g(x))$ if and only if, for every $M > 0$, there exists a constant x_0, such that

$$|f(x)| \leq\ M|g(x)| \text{ for all } x > x_0.$$

Note the difference between the earlier formal definition for the big-O notation, and the present definition of little-o: while the former has to be true for *at least one* constant M the latter must hold for *every* positive constant.[1] In this way little-o notation puts a stronger restriction compared to big-O, in that every function that is little-o is also big-O, but not every function that is big-O is also little-o.

For example,

- $2x \in o(x^2)$
- $2x^2 \notin o(x^2)$
- $1/x \in o(1)$

Little-o notation is common in mathematics but rarer in computer science. In computer science the variable (and function value) is most often a natural number. In mathematics, the variable and function values are often real numbers. The following properties can be useful:

- $o(f) + o(f) \subseteq o(f)$
- $o(f)o(g) \subseteq o(fg)$
- $o(o(f)) \subseteq o(f)$
- $o(f) \subset O(f)$ (and thus the above properties apply with most combinations of o and O).

As with big O notation, the statement "$f(x)$ is $o(g(x))$" is usually written as $f(x) = o(g(x))$, which is a slight abuse of notation.

Family of Bachmann–Landau notations

Notation	Name	Intuition	As $n \to \infty$, eventually...	Definition
$f(n) \in O(g(n))$	Big Omicron; Big O; Big Oh	f is bounded above by g (up to constant factor) asymptotically	$\|f(n)\| \leq g(n) \cdot k$ for some k	$\exists k > 0, n_0 \; \forall n > n_0 \; \|f(n)\| \leq \|g(n) \cdot k\|$ or $\exists k > 0, n_0 \; \forall n > n_0 \; f(n) \leq g(n) \cdot k$
$f(n) \in \Omega(g(n))$ (Note that, since the beginning of the 20th century, papers in number theory have been increasingly and widely using this notation in the weaker sense that f = o(g) is false)	Big Omega	f is bounded below by g (up to constant factor) asymptotically	$\|f(n)\| \geq g(n) \cdot k$ for some k	$\exists k > 0, n_0 \; \forall n > n_0 \; g(n) \cdot k \leq \|f(n)\|$
$f(n) \in \Theta(g(n))$	Big Theta	f is bounded both above and below by g asymptotically	$\|g(n)\| \cdot k_1 \leq \|f(n)\| \leq \|g(n)\| \cdot k_2$ for some k_1, k_2	$\exists k_1, k_2 > 0, n_0 \; \forall n > n_0$ $\|g(n)\| \cdot k_1 \leq \|f(n)\| \leq \|g(n)\| \cdot k_2$
$f(n) \in o(g(n))$	Small Omicron; Small O; Small Oh	f is dominated by g asymptotically	$\|f(n)\| \leq \|g(n)\| \cdot \varepsilon$ for every ε	$\forall \varepsilon > 0 \; \exists n_0 \; \forall n > n_0 \; \|f(n)\| \leq \|g(n) \cdot \varepsilon\|$
$f(n) \in \omega(g(n))$	Small Omega	f dominates g asymptotically	$\|f(n)\| \geq \|g(n)\| \cdot k$ for every k	$\forall k > 0 \; \exists n_0 \; \forall n > n_0 \; \|g(n) \cdot k\| \leq \|f(n)\|$
$f(n) \sim g(n)$	on the order of; "twiddles"	f is equal to g asymptotically	$f(n)/g(n) \to 1$	$\forall \varepsilon > 0 \; \exists n_0 \; \forall n > n_0 \; \|\frac{f(n)}{g(n)} - 1\| < \varepsilon$

Bachmann–Landau notation was designed around several mnemonics, as shown in the *As $n \to \infty$, eventually...* column above and in the bullets below. To conceptually access these mnemonics, "omicron" can be read "o-*micro*n" and "omega" can be read "o-*mega*". Also, the lower-case versus capitalization of the Greek letters in Bachmann–Landau notation is mnemonic.

- The *o-**micro**n mnemonic*: The o-*micro*n reading of $f(n) \in O(g(n))$ and of $f(n) \in o(g(n))$ can be thought of as "O-*smaller than*" and "o-*smaller than*", respectively. This *micro*/smaller mnemonic refers to: for sufficiently large input parameter(s), f grows at a rate that may henceforth be **less** than cg regarding $g \in O(f)$ or $g \in o(f)$.
- The *o-**mega** mnemonic*: The o-*mega* reading of $f(n) \in \Omega(g(n))$ and of $f(n) \in \omega(g(n))$ can be thought of as "O-*larger than*". This *mega*/larger mnemonic refers to: for sufficiently large input parameter(s), f grows at a rate that may henceforth be **greater** than cg regarding $g \in \Omega(f)$ or $g \in \omega(f)$.
- The ***upper**-case mnemonic*: This mnemonic reminds us when to use the upper-case Greek letters in $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$: for sufficiently large input parameter(s), f grows at a rate that may henceforth be **equal** to cg regarding $g \in O(f)$.
- The ***lower**-case mnemonic*: This mnemonic reminds us when to use the lower-case Greek letters in $f(n) \in o(g(n))$ and $f(n) \in \omega(g(n))$: for sufficiently large input parameter(s), f grows at a rate that is henceforth **inequal** to cg regarding $g \in O(f)$.

Aside from Big O notation, the Big Theta Θ and Big Omega Ω notations are the two most often used in computer science; the Small Omega ω notation is rarely used in computer science.

Informally, especially in computer science, the Big O notation often is permitted to be somewhat abused to describe an asymptotic tight bound where using Big Theta Θ notation might be more factually appropriate in a given context. For example, when considering a function $T(n) = 73n^3 + 22n^2 + 58$, all of the following are generally acceptable, but tightnesses of bound (i.e., bullets 2 and 3 below) are usually strongly preferred over laxness of bound (i.e., bullet 1 below).

1. $T(n) = O(n^{100})$, which is identical to $T(n) \in O(n^{100})$
2. $T(n) = O(n^3)$, which is identical to $T(n) \in O(n^3)$
3. $T(n) = \Theta(n^3)$, which is identical to $T(n) \in \Theta(n^3)$.

The equivalent English statements are respectively:

1. $T(n)$ grows asymptotically no faster than n^{100}
2. $T(n)$ grows asymptotically no faster than n^3
3. $T(n)$ grows asymptotically as fast as n^3.

So while all three statements are true, progressively more information is contained in each. In some fields, however, the Big O notation (bullets number 2 in the lists above) would be used more commonly than the Big Theta notation (bullets number 3 in the lists above) because functions that grow more slowly are more desirable. For example, if $T(n)$ represents the running time of a newly developed algorithm for input size n, the inventors and users of the algorithm might be more inclined to put an upper asymptotic bound on how long it will take to run without making an explicit statement about the lower asymptotic bound.

Extensions to the Bachmann–Landau notations

Another notation sometimes used in computer science is $\tilde{O}$ (read *soft-O*): $f(n) = \tilde{O}(g(n))$ is shorthand for $f(n) = O(g(n) \log^k g(n))$ for some k. Essentially, it is Big O notation, ignoring logarithmic factors because the growth-rate effects of some other super-logarithmic function indicate a growth-rate explosion for large-sized input parameters that is more important to predicting bad run-time performance than the finer-point effects contributed by the logarithmic-growth factor(s). This notation is often used to obviate the "nitpicking" within growth-rates that are stated as too tightly bounded for the matters at hand (since $\log^k n$ is always $o(n^\varepsilon)$ for any constant k and any $\varepsilon > 0$).

The L notation, defined as

$$L_n[\alpha, c] = O\left(e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}\right),$$

is convenient for functions that are between polynomial and exponential.

Generalizations and related usages

The generalization to functions taking values in any normed vector space is straightforward (replacing absolute values by norms), where f and g need not take their values in the same space. A generalization to functions g taking values in any topological group is also possible.

The "limiting process" $x \to x_o$ can also be generalized by introducing an arbitrary filter base, i.e. to directed nets f and g.

The o notation can be used to define derivatives and differentiability in quite general spaces, and also (asymptotical) equivalence of functions,

$$f \sim g \iff (f - g) \in o(g)$$

which is an equivalence relation and a more restrictive notion than the relationship "f is $\Theta(g)$" from above. (It reduces to $\lim f/g = 1$ if f and g are positive real valued functions.) For example, $2x$ is $\Theta(x)$, but $2x - x$ is not $o(x)$.

Graph theory

It is often useful to bound the running time of graph algorithms. Unlike most other computational problems, for a graph $G = (V, E)$ there are two relevant parameters describing the size of the input: the number $|V|$ of vertices in the graph and the number $|E|$ of edges in the graph. Inside asymptotic notation (and only there), it is common to use the symbols V and E, when someone really means $|V|$ and $|E|$. We adopt this convention here to simplify asymptotic functions and make them easily readable. The symbols V and E are never used inside asymptotic notation with their literal meaning, so this abuse of notation does not risk ambiguity. For example $O(E + V \log V)$ means $O((E, V) \mapsto |E| + |V| \cdot \log |V|)$ for a suitable metric of graphs. Another common convention—referring to the values $|V|$ and $|E|$ by the names n and m, respectively—sidesteps this ambiguity.

History

The notation was first introduced by number theorist Paul Bachmann in 1894, in the second volume of his book *Analytische Zahlentheorie*

("analytic number theory"), the first volume of which (not yet containing big O notation) was published in 1892.[5] The notation was popularized in the work of number theorist Edmund Landau; hence it is sometimes called a Landau symbol. It was popularized in computer science by Donald Knuth, who (re)introduced the related Omega and Theta notations.[6] He also noted that the (then obscure) Omega notation had been introduced by Hardy and Littlewood[7] under a slightly different meaning, and proposed the current definition. Hardy's symbols were (in terms of the modern *O* notation)

$$f \lesssim g \iff f \in O(g) \text{ and } f \ll g \iff f \in o(g);$$

other similar symbols were sometimes used, such as $\preceq$ and $\prec\prec$.

The big-O, standing for "order of", was originally a capital omicron; today the identical-looking Latin capital letter O is used, but never the digit zero.

See also

- Asymptotic expansion: Approximation of functions generalizing Taylor's formula
- Asymptotically optimal: A phrase frequently used to describe an algorithm that has an upper bound asymptotically within a constant of a lower bound for the problem
- Limit superior and limit inferior: An explanation of some of the limit notation used in this article
- Nachbin's theorem: A precise method of bounding complex analytic functions so that the domain of convergence of integral transforms can be stated
- Big O in probability notation: O_p,o_p
- Computational complexity theory: A sub-field strongly related to this article

Notes

1. ^ **a b** Thomas H. Cormen et al., 2001, Introduction to Algorithms, Second Edition (http://highered.mcgraw-hill.com/sites/0070131511/)
2. **^** N. G. de Bruijn (1958). *Asymptotic Methods in Analysis* (http://books.google.com/?id=_tnwmvHmVwMC&pg=PA5&vq=%22The+trouble+is%22) . Amsterdam: North-Holland. pp. 5–7. ISBN 9780486642215. http://books.google.com/?id=_tnwmvHmVwMC&pg=PA5&vq=%22The+trouble+is%22.
3. ^ **a b** Ronald Graham, Donald Knuth, and Oren Patashnik (1994). *Concrete Mathematics* (http://books.google.com/?id=pntQAAAAMAAJ&dq=editions:ISBN0201558025) (2 ed.). Reading, Massachusetts: Addison-Wesley. p. 446. ISBN 9780201558029. http://books.google.com/?id=pntQAAAAMAAJ&dq=editions:ISBN0201558025.
4. **^** Donald Knuth (June/July 1998). "Teach Calculus with Big O" (http://www.ams.org/notices/199806/commentary.pdf) . *Notices of the American Mathematical Society* **45** (6): 687. http://www.ams.org/notices/199806/commentary.pdf. (Unabridged version (http://www-cs-staff.stanford.edu/~knuth/ocalc.tex))
5. **^** Nicholas J. Higham, *Handbook of writing for the mathematical sciences*, SIAM. ISBN 0-89871-420-6, p. 25
6. **^** Donald Knuth. *Big Omicron and big Omega and big Theta (http://doi.acm.org/10.1145/1008328.1008329)* , ACM SIGACT News, Volume 8, Issue 2, 1976.
7. **^** G. H. Hardy and J. E. Littlewood, *Some problems of Diophantine approximation,* Acta Mathematica 37 (1914), p. 225

Further reading

- Paul Bachmann. *Die Analytische Zahlentheorie. Zahlentheorie*. pt. 2 Leipzig: B. G. Teubner, 1894.
- Edmund Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. 2 vols. Leipzig: B. G. Teubner, 1909.
- G. H. Hardy. *Orders of Infinity: The 'Infinitärcalcül' of Paul du Bois-Reymond*, 1910.
- Marian Slodicka (Slodde vo de maten) & Sandy Van Wontergem. *Mathematical Analysis I*. University of Ghent, 2004.
- Donald Knuth. *The Art of Computer Programming*, Volume 1: *Fundamental Algorithms*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4. Section 1.2.11: Asymptotic Representations, pp. 107–123.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 3.1: Asymptotic notation, pp. 41–50.
- Michael Sipser (1997). *Introduction to the Theory of Computation*. PWS Publishing. ISBN 0-534-94728-X. Pages 226–228 of section 7.1: Measuring complexity.
- Jeremy Avigad, Kevin Donnelly. *Formalizing O notation in Isabelle/HOL (http://www.andrew.cmu.edu/~avigad/Papers/bigo.pdf)*
- Paul E. Black, "big-O notation" (http://www.nist.gov/dads/HTML/bigOnotation.html) , in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 11 March 2005. Retrieved December 16, 2006.
- Paul E. Black, "little-o notation" (http://www.nist.gov/dads/HTML/littleOnotation.html) , in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 17 December 2004. Retrieved December 16, 2006.
- Paul E. Black, "Ω" (http://www.nist.gov/dads/HTML/omegaCapital.html) , in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 17 December 2004. Retrieved December 16, 2006.
- Paul E. Black, "ω" (http://www.nist.gov/dads/HTML/omega.html) , in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 29 November 2004. Retrieved December 16, 2006.
- Paul E. Black, "Θ" (http://www.nist.gov/dads/HTML/theta.html) , in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 17 December 2004. Retrieved December 16, 2006.

External links

- Introduction to Asymptotic Notations (http://www.soe.ucsc.edu/classes/cmps102/Spring04/TantaloAsymp.pdf)