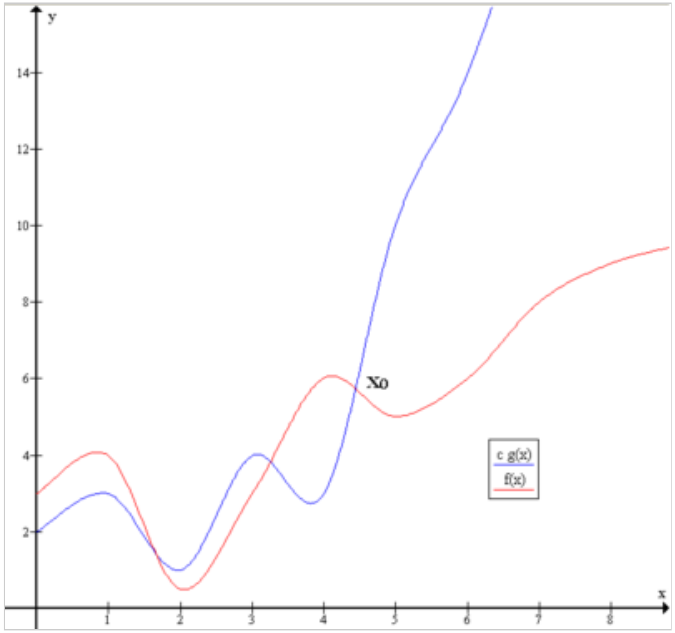
Big O notation

From Wikipedia, the free encyclopedia

In mathematics, **big 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. It is a member of a larger family of notations that is called **Landau notation**, **Bachmann–Landau notation** (after Edmund Landau and Paul Bachmann),[1][2] or **asymptotic notation**. In computer science, big O notation is used to classify algorithms by how they respond (*e.g.,* in their processing time or working space requirements) to changes in input size.[3] In analytic number theory, it is used to estimate the "error committed" while replacing the asymptotic size, or asymptotic mean size, of an arithmetical function, by the value, or mean value, it takes at a large finite argument. A famous example is the problem of estimating the remainder term in the prime number theorem.

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. The letter O is used because the growth rate of a function is also referred to as order of the function. 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.



Example of Big O notation: $f(x) \in O(g(x))$ as there exists $c > 0$ (e.g., $c = 1$) and x_0 (e.g., $x_0 = 5$) such that $f(x) < cg(x)$ whenever $x > x_0$.

Contents

Formal definition

Let f and g 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 there is a positive constant M such that for all sufficiently large values of x, the absolute value of $f(x)$ is at most M multiplied by the absolute value of $g(x)$. 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

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

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

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

Additionally, the notation $O(g(x))$ is also used to denote the set of all functions $f(x)$ that satisfy the relation $f(x)=O(g(x))$. In this case we write

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

Example

In typical usage, the formal definition of O notation is not used directly; rather, the O notation for a function f 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). 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 $x > x_0$:

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

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 considered by some as an abuse of notation.[4]

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. Consider, for example, the exponential series and two expressions of it that are valid when x is small:

$$\begin{aligned} e^x &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots && \text{for all } x \\ &= 1 + x + \frac{x^2}{2} + O(x^3) && \text{as } x \to 0, \\ &= 1 + x + O(x^2) && \text{as } x \to 0. \end{aligned}$$

The second expression (the one with $O(x^3)$) means the absolute-value of the error $e^x - (1 + x + x^2/2)$ is smaller than some constant times $|x^3|$ when x is close enough to 0.

Properties

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

$$f(n) = 9\log n + 5(\log n)^3 + 3n^2 + 2n^3 = O(n^3), \qquad \text{as } n \to \infty.$$

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. Another thing to notice is the sets $O(n^c)$ and $O(c^n)$ are very different. If *c* is greater than one, then the latter grows much faster. A function that grows faster than n^c for any *c* 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 and the function $n^{\log n}$.

We may ignore any powers of *n* inside of the logarithms. The set $O(\log n)$ 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. On the other hand, exponentials with different bases 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 $c^2n^2 = O(n^2)$. 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 variables may also affect the order of the resulting algorithm. For example, if an algorithm's run time is $O(n)$ when measured in terms of the number *n* of *digits* of an input number *x*, then its run time is $O(\log x)$ when measured as a function of the input number *x* itself, because $n = O(\log x)$.

Product

$$f_1 = O(g_1) \text{ and } f_2 = O(g_2) \Rightarrow f_1 f_2 = O(g_1 g_2)$$
$$f \cdot O(g) = O(fg)$$

Sum

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

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

If *f* and *g* are positive functions, $f + O(g) = O(f + g)$

Multiplication by a constant

Let *k* be a constant. Then:
$O(kg) = O(g)$ if *k* is nonzero.
$f \in O(g) \Rightarrow kf = O(g).$

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[5]

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

Equivalently, the condition that $x_i \geq M$ for some i can be replaced with the condition that $\|\vec{x}\| \geq M$, where $\|\vec{x}\|$ denotes the Chebyshev distance. For example, the statement

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

asserts that there exist constants *C* and *M* such that

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

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

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

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

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

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

This is not the only generalization of big O to multivariate functions, and in practice, there is some inconsistency in the choice of definition.[6]

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.[7] 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 $n^2 = O(n^2)$."[8] 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.[8] 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."[9]

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

$$g(x) - h(x) = O(f(x)).$$

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 $55n^3 + 2n + 10$ steps before it terminates. Thus the overall time complexity of the algorithm can be expressed as $T(n) = 55n^3 + O(n^2)$. Here the terms $2n+10$ are subsumed within the faster-growing $O(n^2)$. Again, this usage disregards some of the formal meaning of the "=" symbol, 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:

$$\begin{aligned} f(m) &= O(m^n), \\ g(n) &= O(m^n). \end{aligned}$$

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

$$g(x) = O(f(x)).$$

rather than the less explicit

$$g = O(f),$$

Multiple usages

In more complicated usage, *O*(...) 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*(...) on the left side, there are *some* functions satisfying each *O*(...) 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. In this use the "=" is a formal symbol that unlike the usual use of "=" is not a symmetric relation. Thus for example $n^{O(1)} = O(e^n)$ does not imply the false statement $O(e^n) = n^{O(1)}$

Orders of common functions

Further information: Time complexity § Table of common time complexities

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.

Notation	Name	Example
$O(1)$	constant	Determining if a binary number is even or odd; Calculating $(-1)^n$; Using a constant-size lookup table
$O(\log\log n)$	double logarithmic	Number of comparisons spent finding an item using interpolation search in a sorted array of uniformly distributed values
$O(\log n)$	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
$O((\log n)^c),\ c > 1$	polylogarithmic	Matrix chain ordering can be solved in polylogarithmic time on a Parallel Random Access Machine.
$O(n^c),\ 0 < c < 1$	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
$O(n\log^* n)$	n log-star n	Performing triangulation of a simple polygon using Seidel's algorithm, or the union–find algorithm. Note that $\log^*(n) = \begin{cases} 0, & \text{if } n \le 1 \\ 1 + \log^*(\log n), & \text{if } n > 1 \end{cases}$
$O(n\log n) = O(\log n!)$	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 travelling 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; generating all unrestricted permutations of a poset; finding the determinant with expansion by minors; enumerating all partitions of a set

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 $O(n^{c+\varepsilon})$ 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

"Little o" redirects here. For the baseball player, see Omar Vizquel.

The informal assertion "*f*(*x*) is little-o of *g*(*x*)" is formally written *f*(*x*) = *o*(*g*(*x*)), or in set notation *f*(*x*) ∈ *o*(*g*(*x*)). Intuitively, it means that *g*(*x*) grows much faster than *f*(*x*), or similarly, that 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, *f*(*n*) = *o*(*g*(*n*)) (or *f*(*n*) ∈ *o*(*g*(*n*))) as $n \to \infty$ means that for every positive constant ε there exists a constant *N* such that

$$|f(n)| \le \epsilon |g(n)| \qquad \text{for all } n \ge N\ .$$[8]

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 ε, however small.[4] In this way little-o notation makes a stronger statement than the corresponding big-O notation: every function that is little-o of *g* is also big-O of *g*, but not every function that is big-O *g* is also little-o of *g* (for instance *g* itself is not, unless it is identically zero near ∞).

If $g(x)$ is nonzero, or at least becomes nonzero beyond a certain point, the relation $f(x) = o(g(x))$ is equivalent to

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

For example,

- $2x = o(x^2)$
- $2x^2 \neq o(x^2)$
- $1/x = o(1)$

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 (expressed in the more recent, set-theoretical notation) can be useful:

- $c \cdot o(f) = o(f)$ for $c \neq 0$
- $o(f)o(g) \subseteq o(fg)$
- $o(o(f)) \subseteq o(f)$
- $o(f) \subset O(f)$ (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 some consider an abuse of notation.

Big Omega notation

There are two very widespread and incompatible definitions of the statement

$$f(x) = \Omega(g(x)) \ (x \to a),$$

where *a* is some real number, ∞, or −∞, where *f* and *g* are real functions defined in a neighbourhood of *a*, and where *g* is positive in this neighbourhood.

The first one (chronologically) is used in analytic number theory, and the other one in computational complexity theory. When the two subjects meet, this situation is bound to generate confusion.

The Hardy–Littlewood definition

In 1914 G.H. Hardy and J.E. Littlewood introduced the new symbol Ω,[10] which is defined as follows:

$$f(x) = \Omega(g(x)) \ (x \to \infty) \ \Leftrightarrow \ \limsup_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| > 0.$$

Thus $f(x) = \Omega(g(x))$ is the negation of $f(x) = o(g(x))$.

In 1918 the same authors introduced the two new symbols Ω_R and Ω_L,[11] thus defined:

$$f(x) = \Omega_R(g(x)) \ (x \to \infty) \ \Leftrightarrow \ \limsup_{x\to\infty} \frac{f(x)}{g(x)} > 0;$$

$$f(x) = \Omega_L(g(x)) \ (x \to \infty) \ \Leftrightarrow \ \liminf_{x\to\infty} \frac{f(x)}{g(x)} < 0.$$

Hence $f(x) = \Omega_R(g(x))$ is the negation of $f(x) < o(g(x))$, and $f(x) = \Omega_L(g(x))$ the negation of $f(x) > o(g(x))$.

Contrary to a later assertion of D.E. Knuth,[12] Edmund Landau did use these three symbols, with the same meanings, in 1924.[13]

These Hardy-Littlewood symbols are prototypes, which after Landau were never used again exactly thus.

Ω_R became Ω_+, and Ω_L became Ω_-.

These three symbols $\Omega, \Omega_+, \Omega_-$, as well as $f(x) = \Omega_\pm(g(x))$ (meaning that $f(x) = \Omega_+(g(x))$ and $f(x) = \Omega_-(g(x))$ are both satisfied), are now currently used in analytic number theory.[14]

Simple examples

We have

$$\sin x = \Omega(1) \ (x \to \infty),$$

and more precisely

$$\sin x = \Omega_\pm(1) \ (x \to \infty).$$

We have

$$\sin x + 1 = \Omega(1) \ (x \to \infty),$$

and more precisely

$$\sin x + 1 = \Omega_+(1) \ (x \to \infty);$$

however

$$\sin x + 1 \neq \Omega_-(1) \ (x \to \infty).$$

The Knuth definition

In 1976 Donald Knuth published a paper to justify his use of the Ω-symbol to describe a stronger property. Knuth wrote: "For all the applications I have seen so far in computer science, a stronger requirement […] is much more appropriate". He defined

$$f(x) = \Omega(g(x)) \Leftrightarrow g(x) = O(f(x))$$

with the comment: "Although I have changed Hardy and Littlewood's definition of Ω, I feel justified in doing so because their definition is by no means in wide use, and because there are other ways to say what they want to say in the comparatively rare cases when their definition applies".[12] However, the Hardy–Littlewood definition had been used for at least 25 years.[15]

Family of Bachmann–Landau notations

Notation	Name	Intuition	Informal definition: for sufficiently large n...	Formal Definition
$f(n) = O(g(n))$	Big O; Big Oh; Big Omicron[12]	f is bounded above by g (up to constant factor) asymptotically	$\|f(n)\| \leq k \cdot \|g(n)\|$ for some positive k	$\exists k > 0 \ \exists n_0 \ \forall n > n_0 \ \|f(n)\| \leq k \cdot \|g(n)\|$
$f(n) = \Omega(g(n))$	Big Omega	**Two definitions :** Number theory: f is not dominated by g asymptotically Complexity theory: f is bounded below by g asymptotically	Number theory: $f(n) \geq k \cdot g(n)$ for infinitely many values of n and for some positive k Complexity theory: $f(n) \geq k \cdot g(n)$ for some positive k	Number theory: $\exists k > 0 \ \forall n_0 \ \exists n > n_0 \ f(n) \geq k \cdot g(n)$ Complexity theory: $\exists k > 0 \ \exists n_0 \ \forall n > n_0 \ f(n) \geq k \cdot g(n)$
$f(n) = \Theta(g(n))$	Big Theta	f is bounded both above and below by g asymptotically	$k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$ for some positive k_1, k_2	$\exists k_1 > 0 \ \exists k_2 > 0 \ \exists n_0 \ \forall n > n_0$ $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$
$f(n) = o(g(n))$	Small O; Small Oh	f is dominated by g asymptotically	$\|f(n)\| \leq k \cdot \|g(n)\|$, for every fixed positive number k	$\forall k > 0 \ \exists n_0 \ \forall n > n_0 \ \|f(n)\| \leq k \cdot \|g(n)\|$
$f(n) = \omega(g(n))$	Small Omega	f dominates g asymptotically	$\|f(n)\| \geq k \cdot \|g(n)\|$, for every fixed positive number k	$\forall k > 0 \ \exists n_0 \ \forall n > n_0 \ \|f(n)\| \geq k \cdot \|g(n)\|$
$f(n) \sim g(n)$	On the order of	f is equal to g asymptotically	$f(n)/g(n) \to 1$	$\forall \varepsilon > 0 \ \exists n_0 \ \forall n > n_0 \ \left\|\frac{f(n)}{g(n)} - 1\right\| < \varepsilon$

Aside from the 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 occasionally used in computer science.

Aside from the Big *O* notation, the small *o*, Big Omega Ω and $\sim$ notations are the three most often used in number theory; the small omega ω notation is never used in number theory.

Use in computer science

For more details on this topic, see Analysis of algorithms.

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 tighter bounds (i.e., numbers 2 and 3 below) are usually strongly preferred over looser bounds (i.e., number 1 below).

1. $T(n) = O(n^{100})$
2. $T(n) = O(n^3)$
3. $T(n) = \Theta(n^3)$

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 (number 2 in the lists above) would be used more commonly than the Big Theta notation (bullets number 3 in the lists above). 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 Õ (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 $\varepsilon > 0$).

Also 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 $o(x)$.

History (Bachmann–Landau, Hardy, and Vinogradov notations)

The symbol O 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.[16] The number theorist Edmund Landau adopted it, and was thus inspired to introduce in 1909 the notation o;[17] hence both are now called Landau symbols. These notations were used in applied mathematics during the 1950s for asymptotic analysis.[18] The big O was popularized in computer science by Donald Knuth, who re-introduced the related Omega and Theta notations.[12] Knuth also noted that the Omega notation had been introduced by Hardy and Littlewood[10] under a different meaning "$\neq o$" (i.e. "is not an o of"), and proposed the above definition. Hardy and Littlewood's original definition (which was also used in one paper by Landau[13]) is still used in number theory (where Knuth's definition is never used). In fact, Landau also used in 1924, in the paper just mentioned, the symbols Ω_R ("right") and Ω_L ("left"), which were introduced in 1918 by Hardy and Littlewood,[11] and which were precursors for the modern symbols Ω_+ ("is not smaller than a small o of") and Ω_- ("is not larger than a small o of"). Thus the Omega symbols (with their original meanings) are sometimes also referred to as "Landau symbols".

Also, Landau never used the Big Theta and small omega symbols.

Hardy's symbols were (in terms of the modern O notation)

$$f \preceq g \iff f \in O(g) \text{ and } f \prec g \iff f \in o(g);$$