wapedia.

Wiki: Big O notation (1/2)

Search Wikipedia!

big o notation

Search

In mathematics, computer science, and related fields, **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. Big O notation allows its users to simplify functions in order to concentrate on 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 computational complexity theory 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. Big O notation is also used in many other fields to provide similar estimates.

Big O notation is also called **Big Oh notation**, **Landau notation**, **Bachmann-Landau notation**, and **asymptotic notation**. 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.

Contents:

1. 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 times $g(x)$ 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

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

2. 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 term does not depend on x. Omitting this term 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 $x > x_0$:

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

so

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

3. 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.

3. 1. 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^2. 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]

3. 2. Infinitesimal asymptotics

Big O can also be used to describe the error term in an approximation to a mathematical function. 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.

4. 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 runs on the order of *O(n)* when *n* is the number of *digits* of the input number, then it has order *O(log n)* when *n* is the input number itself.

4. 1. Product

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

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

4. 2. Sum

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

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

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

4. 3. Multiplication by a constant

Let *k* be a constant. Then:

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

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

5. 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

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

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

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

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

6. Matters of notation

6. 1. Equals sign

The statement "$f(x)$ is $O(g(x))$" as defined above is usually written as $f(x) = O(g(x))$. This is an abuse of notation; equality of two functions is not asserted, and it cannot be since the property of being $O(g(x))$ is not symmetric:

$O(x) = O(x^2)$ but $O(x^2) \neq O(x)$.

There is also a second reason why that notation is not precise. The symbol $f(x)$ means the value of the function f for the argument x. Hence the symbol of the function is f and not $f(x)$.

For these reasons, it would be more precise to use set notation and write $f \quad O(g)$, thinking of $O(g)$ as the set of all functions dominated by g. However, the use of the equals sign is customary. Knuth pointed out [2] (as de Bruijn [3] did) 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."

6. 2. 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

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

6. 2. 1. 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.

6. 3. 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

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

6. 4. Complex usages

In more complex usage, $O(\dots)$ 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 $O(\dots)$ on the left side, there are *some* functions satisfying each $O(\dots)$ 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.

7. Orders of common functions

Here is a list of classes of functions that are commonly encountered when analyzing algorithms. In each case, *c* is a constant and *n* increases without bound. The slower-growing functions are listed first.

Notation	Name	Example
$O(1)$	constant	Determining if a number is even or odd; using a constant-size lookup table or hash table
$O(\alpha(n))$	inverse	Amortized time per operation using a disjoint set

	Ackermann	
$O(\log^* n)$	iterated logarithmic	The `find` algorithm of Hopcroft and Ullman on a disjoint set
$O(\log \log n)$	log-logarithmic	Amortized time per operation using a bounded priority queue [4]
$O(\log n)$	logarithmic	Finding an item in a sorted array with a binary search
$O\left((\log n)^c\right), c > 1$	polylogarithmic	Deciding if *n* is prime with the AKS primality test
$O\left(n^c\right), 0 < c < 1$	fractional power	Searching in a kd-tree
$O(n)$	linear	Finding an item in an unsorted list; adding two *n*-digit numbers
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, or quasilinear	Performing a Fast Fourier transform; heapsort, quicksort (best case), or merge sort
$O\left(n^2\right)$	quadratic	Multiplying two *n*-digit numbers by a simple algorithm; adding two *n*×*n* matrices; bubble sort (worst case or naive implementation), shell sort, quicksort (worst case), or insertion sort
$O\left(n^3\right)$	cubic	Multiplying two *n*×*n* matrices by simple algorithm; finding the shortest path on a weighted digraph with the Floyd-Warshall algorithm; inverting a (dense) *n*x*n* matrix using LU or Cholesky decomposition
$O\left(n^c\right), c > 1$	polynomial or algebraic	Tree-adjoining grammar parsing; maximum matching for bipartite graphs (grows faster than cubic if and only if $c > 3$)
$L_n[\alpha, c], 0 < \alpha < 1 = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$	L-notation	Factoring a number using the special or general number field sieve
$O\left(c^n\right), c > 1$	exponential or geometric	Finding the (exact) solution to the traveling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute force
$O(n!)$	factorial or combinatorial	Solving the traveling salesman problem via brute-force search; finding the determinant with expansion by minors.
$c_1^{O(c_2^n)}, c_1, c_2 > 1$	double exponential	Deciding the truth of a given statement in Presburger arithmetic

Remarks:

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

Larger growth, such as the single-valued version of the Ackermann function, A(*n*,*n*) is uncommon in practice.

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