What is an easy way for finding C and N when proving the Big-Oh of an Algorithm?

Asked 11 years ago Active 8 months ago Viewed 26k times

12

10

I'm starting to learn about Big-Oh notation.

What is an easy way for finding C and N_0 for a given function?

Say, for example:

$(n+1)^5$, or $n^5+5n^4+10n^2+5n+1$

I know the formal definition for Big-Oh is:

> Let f(n) and g(n) be functions mapping nonnegative integers to real numbers. We say that f(n) is O(g(n)) if there is a real constant c > 0 and an integer constant N_0 >= 1 such that f(n) <= cg(n) for every integer N > N_0.

My question is, what is a good, sure-fire method for picking values for c and N_0?

For the given polynomial above $(n+1)^5$, I have to show that it is $O(n^5)$. So, how should I pick my c and N_0 so that I can make the above definition true without guessing?

big-o

edited Sep 2 '09 at 18:30

asked Sep 2 '09 at 18:25

Philosoraptor

5 Answers

Active Oldest Votes

11

You can pick a constant c by adding the coefficients of each term in your polynomial. Since

$| n^5 + 5n^4 + 0n^3 + 10n^2 + 5n^1 + 1n^0 | <= | n^5 + 5n^5 + 0n^5 + 10n^5 + 5n^5 + 1n^5 |$

and you can simplify both sides to get

$| n^5 + 5n^4 + 10n^2 + 5n + 1 | <= | 22n^5 |$

So c = 22, and this will always hold true for any n >= 1.

It's almost always possible to find a lower c by raising N_0, but this method works, and you can do it in your head.

(The absolute value operations around the polynomials are to account for negative coefficients.)

edited May 7 '17 at 3:31

answered Sep 2 '09 at 18:44
Bill the Lizard
358k • 168 • 534 • 830

amazing explanation – csguy Jan 21 at 2:23

3

Usually the proof is done without picking concrete C and N_0. Instead of proving f(n) < C * g(n) you prove that f(n) / g(n) < C.

For example, to prove $n^3 + n$ is $O(n^3)$ you do the following:

$(n^3 + n) / n^3 = 1 + (n / n^3) = 1 + (1 / n^2) < 2$ for any n >= 1. Here you can pick any C >= 2 with $N_0 = 1$.

edited Jan 21 at 5:49

answered Sep 2 '09 at 18:53

2

You can check what the lim abs(f(n)/g(n)) is when n->+infitity and that would give you the constant (g(n) is n^5 in your example, f(n) is (n+1)^5).

Note that the meaning of Big-O for x->+infinity is that if f(x) = O(g(x)), then f(x) "grows no faster than g(x)", so you just need to prove that lim abs(f(x)/g(x)) exists and is less than +infinity.

answered Sep 2 '09 at 18:43
7macaw
51 ● 5

1

It's going to depend greatly on the function you are considering. However, for a given class of functions, you may be able to come up with an algorithm.

For instance, polynomials: if you set C to any value greater than the leading coefficient of the polynomial, then you can solve for N_0.

answered Sep 2 '09 at 18:48
James Rose
235 ● 1 ● 4

0

After you understand the magic there, you should also get that big-O is a **notation**. It means that you *do not have to look for these coefficients in every problem you solve*, once you made sure you understood what's going on behind these letters. You should just operate the symbols according to the *notaion*, according to its rules.

There's no easy generic rule to determine actual values of N and c. You should recall your calculus knowledge to solve it.

The definition to big-O is entangled with [definition of the limit](). It makes c satisfy:

c > lim |f(n)/g(n)|, given n approaches +infinity.

If the sequence is upper-bounded, it always has a limit. If it's not, well, then f is not O(g). After you have picked concrete c, you will

P Shved
83k ● 14 ● 113 ● 160