

We have seen several algorithms in the course, and loosely characterized the time it takes to run them in terms of the “size”  $n$  of the input. Let’s now tighten up our analysis. We will study the behavior of algorithms by comparing the number of operations required, which is typically a complicated function of  $n$  of the input, against some simpler function of  $n$ . This simpler function describes either an upper bound, in a certain technical sense to be specified below.

## Big O (an upper bound for asymptotic behavior)

Let  $t(n)$  be a well-defined sequence of integers. This sequence  $t(n)$  represents the “time” or number of steps it takes an algorithm to run as a function of some variable  $n$  which itself represents the “size” of the input. We will consider  $n \geq 0$  and  $t(n)$  to both be positive integers.

Let  $g(n)$  be another well defined sequence of integers. We commonly consider  $g(n)$  to be one of the following:

$$1, \log n, n, n \log n, n^2, n^3, 2^n, \dots$$

**Definition:** The sequence  $t(n)$  is  $O(g(n))$  if there exists two positive numbers  $n_0$  and  $c$  such that, for all  $n > n_0$ ,

$$t(n) \leq c g(n).$$

We say  $t(n)$  is “big O of  $g(n)$ ”.

The condition  $n \geq n_0$  allows us to ignore how  $t(n)$  compares with  $g(n)$  when  $n$  is small. In this sense, our analysis is *asymptotic*.

### Example 1

The function  $t(n) = 5 + 7n$  is  $O(n)$ . To prove this, we write:

$$\begin{aligned} t(n) &= 5 + 7n \\ &\leq 5n + 7n, \quad n \geq 1 \\ &= 12n \end{aligned}$$

and so  $n_0 = 1$  and  $c = 12$  satisfies the definition. An alternative proof is:

$$\begin{aligned} t(n) &= 5 + 7n \\ &\leq n + 7n, \quad n \geq 5 \\ &= 8n \end{aligned}$$

and so  $n_0 = 5$  and  $c = 8$  also satisfies the definition.

A few notes: first, if you can show  $t(n)$  is  $O(g(n))$  using constants  $c, n_0$ , then you can always increase  $c$  and/or  $n_0$  and be sure that these constants will satisfy the definition also. So, don’t think of the  $c$  and  $n_0$  as being unique – there are infinitely many upper bounds!

Similarly, for the above example,  $t(n)$  is also  $O(n^2)$  since  $t(n) \leq 12n \leq 12n^2$  for  $n \geq 1$ . Admittedly, this is a less interesting upper bound, and we typically we are interested in tighter upper bounds, i.e.  $O(n)$  is a tighter bound than  $O(n^2)$ . But the point is the definition still holds.

Second, many students write proofs that they think are correct, but in fact the “proofs” are nonsense. For example, the following “proof” is nonsense:

$$\begin{aligned} 5 + 7n &< cn \\ 5n + 7n &< cn, \quad n \geq 1 \\ 12n &< cn \end{aligned}$$

So  $c > 12$  and  $n_0 = 1$ . The reason this is incorrect is that (a) the first statement basically assumes what you are trying to prove, and (b) there is no clear connection between the statements. What implies what? Are the statements equivalent, etc? *Such proofs will get grades of 0. This is not the “big O” you want.*

Third, What would it mean to say  $t(n)$  is  $O(1)$ , i.e.  $g(n) = 1$ ? Applying the definition, it would mean that there exists a  $c$  and  $n_0$  such that  $t(n) < c$  for all  $n \geq n_0$ . That is,  $t(n)$  is bounded by some constant. Why would this be used? Sometimes when we analyze an algorithm’s performance, we consider different parts of the algorithm separately. Some of those parts (such as assigning values to variables outside of any loops or any recursive calls) might be done once. Such parts take  $O(1)$  time.

Finally, some students try to do big O proofs by using ideas from Calculus and taking “limits”. This is technically correct, provided you are careful and you know what you are doing. However, unless you have taken MATH 242 (Real Analysis), you probably *don’t* know how to do this correctly. *For COMP 250, I am not allowing you to use limits in your big O proofs.*

### Example 2 (see a different example in slides)

The function  $t(n) = 17 - 46n + 8n^2$  is  $O(n^2)$ . To prove this, we want to show there exists positive  $c$  and  $n_0$  such that, for all  $n \geq n_0$ ,

$$17 - 46n + 8n^2 \leq cn^2 .$$

$$\begin{aligned} t(n) &= 17 - 46n + 8n^2 \\ &\leq 17 + 8n^2, \quad n > 0 \\ &\leq 17n^2 + 8n^2, \quad n \geq 1 \\ &= 25n^2 \end{aligned}$$

and so  $n_0 = 1$  and  $c = 25$  does the job.

Alternatively,

$$\begin{aligned} t(n) &= 17 - 46n + 8n^2 \\ &\leq 17 + 8n^2, \quad n > 0 \\ &\leq n^2 + 8n^2, \quad n \geq 5 \\ &= 9n^2 \end{aligned}$$

and so  $c = 9$  and  $n_0 = 5$  does the job.