

## not big O

The definition of big O has several “quantifiers” in it, namely “for all” and “there exists”. Those of you who have been exposed to formal logic (predicate logic, in particular) will have studied how such quantifiers work and practiced using them, but for many of you this will be new. For this reason, and also to make sure you understand what the definition of big O means, let’s consider what it means for a function  $g(n)$  *not* to be an asymptotic upper bound of  $t(n)$ , that is, what does it mean to say that statement “ $t(n)$  is  $O(g(n))$ ” is false? or equivalently, what does the statement “ $t(n)$  is *not*  $O(g(n))$ ” mean ?

Saying  $t(n)$  is *not*  $O(g(n))$  means that there do *not* exist two positive constants  $c$  and  $n_0$  such that, for all  $n \geq n_0$ ,  $t(n) \leq cg(n)$ . Logically, this is equivalent to saying that, *for any* positive constants  $c_0$  and  $n_0$ , there exists at least one  $n$  such that  $n > n_0$  and  $t(n) > c_0g(n)$ . Let’s look a few examples of  $t(n)$  and  $g(n)$  for which  $t(n)$  is not  $O(g(n))$ .

**Example:**  $t(n) = 12^n$  is *not*  $O(9^n)$

See the Exercises on big O for the solution.

**Example:**  $t(n) = 3n^2 + 5n + 2$  is *not*  $O(n)$ , *i.e.*  $g(n) = n$ .

To prove formally that  $t(n)$  is not  $O(n)$ , we use a different argument from what we saw last lecture, namely we use a *proof by contradiction*. We hypothesize that  $t(n)$  is indeed  $O(n)$ , and then we show that this hypothesis cannot be correct, *i.e.* it leads to a contradiction.

Suppose  $t(n)$  is  $O(n)$ . Let  $c > 0$  and  $n_0 \geq 0$  be two constants that satisfy the big O definition. To derive a contradiction, we will show that there exists an  $n$  such that  $n \geq n_0$  and  $t(n)$  is *not* less or equal to  $cn$ , that is,

$$3n^2 + 5n + 2 > cn.$$

Dividing both sides by  $n$  gives an equivalent inequality, so we want to show there exists an  $n \geq n_0$  such that

$$3n + 5 + \frac{2}{n} > c.$$

To get a concrete value of the  $n$  that produces a contradiction, note that

$$\begin{aligned} 3n + 5 + \frac{2}{n} &> 3n + 5, \quad \text{if } n > 0 \\ &\geq c, \quad \text{if } n \geq \frac{c-5}{3} \end{aligned}$$

So, any  $n > \max(\frac{c-5}{3}, n_0)$  will contradict the big O definition for the chosen constants  $n_0, c$ . Hence it is possible to find such constants that satisfy the big O definition. Hence  $t(n)$  is not  $O(n)$ .

## Big Omega (asymptotic lower bound)

With big O, we defined an asymptotic upper bound behavior on the performance of an algorithm. There is a similar definition for the lower bound behavior. This lower bound is used to claim that something can only be done so fast. You will make great use of  $\Omega()$  arguments in future Algorithms courses.

We say that  $t(n)$  is  $\Omega(g(n))$  – “big Omega of  $g(n)$ ” – if there exists a positive integer  $n_0$  and a constant  $c > 0$  such that

$$t(n) \geq c g(n)$$

for all  $n > n_0$ . The idea is that  $t(n)$  grows at least as fast as  $g(n)$  times some constant, for sufficiently large  $n$ . Note that the only difference between the definition of  $O()$  and  $\Omega()$  is the  $\leq$  vs.  $\geq$  inequality.

As an example, you will prove in COMP 251 that the  $t(n)$  of *any* sorting algorithm is  $\Omega(n \log n)$ . There are sorting algorithms such as insertion sort that take  $c_2 n^2 + c_1 n + c_0$  operations, and there are sorting algorithms such as mergesort that take only  $cn \log n$  operations. However, as you will see in COMP 251, it is mathematically impossible to have an algorithm that sorts  $n$  arbitrary numbers using fewer than  $cn \log n$  operations.

Here are a few examples. See the big O and  $\Omega$  exercises for more.

**Example:**  $t(n) = \max(20, \frac{1}{2}(n - 30))$  is  $\Omega(n)$

We want to find positive  $n_0, c$  such that for all  $n \geq n_0$ ,  $t(n) \geq cn$ . We are looking for a lower bound, so we try  $c = \frac{1}{4}$ , i.e. some  $c$  less than  $\frac{1}{2}$ . In order for  $\frac{1}{2}(n - 30) \geq \frac{1}{4}n$ , we must have  $n - 30 \geq \frac{1}{2}n$ , or  $n \geq 60$ .

You might think you are done by taking say  $n_0 = 60$ . However, be careful. This may not be high enough, since we haven't yet considered the “max” part of  $t(n)$ . Let's verify that part too. Note that  $t(n) \geq \frac{1}{2}(n - 30)$ . So, if  $\frac{1}{2}(n - 30) \geq \frac{1}{4}n$  for any  $n \geq 60$ , then we can be sure  $t(n) > \frac{1}{4}n$  for any  $n \geq 60$  too, and we are done.

*See the slides for two other examples. These examples can be found in the big O and  $\Omega$  exercises.*