Definitions of Big-O, etc. (CSC 285, Spring 2015)

Lane A. Hemaspaandra

2015/1/31 (last revised 2015/1/31/115pm)

1 Definitions

These definitions are modeled on those of the book by Greene and Knuth (with some editing and changes). Here, we speak only of functions \( f \) and \( g \) whose codomain is the set of nonnegative integers.

Intuitively, saying \( f(n) = O(g(n)) \) means that, if one is allowed to exclude some prefix of the values of \( n \), for all other (natural-number) values of \( n \), \( f(n) \) is smaller than \( g(n) \), give-or-take a (uniform) multiplicative constant. Now let’s see how one turns that vague statement into a real definition.

**Big-O** We say that \( f(n) = O(g(n)) \) (sometimes phrased “\( f(n) = O(g(n)) \) as \( n \to \infty \)” but that added part these days is almost always taken as implicitly there) if there exist integers \( n_0 \) and \( k \) such that \( f(n) \leq kg(n) \) for all \( n \geq n_0 \).

**Big-Omega** We say that \( f(n) = \Omega(g(n)) \) (sometimes phrased “\( f(n) = \Omega(g(n)) \) as \( n \to \infty \)” but that added part these days is almost always taken as implicitly there) if there exist integers \( n_0 \) and \( k \) such that \( f(n) \geq kg(n) \) for all \( n \geq n_0 \).

(Side note: There exist other definitions that differ from this not just syntactically but even semantically, sometimes extremely much so. When reading a paper, always be alert as to how it is using things, and if it defines a notion, do make sure to note if its definition differs from what you were expected.)

**Big-Theta** If \( f(n) = O(g(n)) \) and \( f(n) = \Omega(g(n)) \), we say that \( f(n) = \Theta(g(n)) \).

**Little-O** (which we might not see or use at all, but in case we do...) We say that \( f(n) = o(g(n)) \) (sometimes phrased “\( f(n) = o(g(n)) \) as \( n \to \infty \)” but that added part these days is almost always taken as implicitly there) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \).
2 Examples

Here are a few examples.

True: \[ n = O(n^2). \]
True: \[ n = O(2^n). \]
True: \[ 2015n = \Theta(n). \]
True: \[ n = O(n \log n). \]
True: \[ n + 1/(n + 1) = \Theta(17n + 3454321). \]
True: \[ n^2 = \Omega(n). \]
True: \[ n + \log n = \Theta(n). \]
True: \[ n = o(n^2). \]
False: \[ 2015n = o(n). \]

The kind of detailed proving of Big-O examples that you can find in books or on the Internet (e.g., although if you try to cut and paste these address be careful about the ~ as it won’t cut and paste right, http://pages.pacificcoast.net/~cazelais/222/big-o.pdf and http://web.eecs.utk.edu/~booth/311-01/notes/bigOex.html) generally isn’t of interest to us in this course, since for the cases we’ll hit, usually things are clear and intuitive.

3 How We’ll Encounter This Notation

For us, we’ll most typically encounter big-O notation regarding the running time of programs. For a particular program, consider the function \( f(n) \) such that \( f(n) \) is the largest number of steps that the program runs for any input whose “size” (typically: number of bits or number of characters) is \( n \) (so we’re taking a maximum, over all the different inputs of size \( n \)). This is called the “run-time” or the “running time” of that program. Then the type of claim one often sees is, “The running time of Algorithm A is \( O(n^2) \).”

However, big-O can be used in other contexts. For example, it is true that the number of binary strings of length at most \( n \) is \( O(2^n) \), although it would be true and more precise simply to state (as can be easily proved by induction) that the number of binary strings of length at most \( n \) is exactly \( O(2^n + 1 - 1) \).

As a coda, which happens not to be about big-O notation: If there exists a polynomial \( p(n) \) such that “the running time of (deterministic) Algorithm A is, for each \( n \), less than or equal to \( p(n) \),” then we say that A is a polynomial-time algorithm. And (assuming A is being viewed as a language-accepting machine) the language (i.e., set) that A is accepting is said to belong to the complexity class \( P \), the class of languages that can be accepted in (deterministic) polynomial time.