Lecture 21: Big-Theta, Logs
10:00 AM, Oct 23, 2019

Contents

1 More analysis terminology .......................... 1

2 Theorems for proving Big-O class ................. 2

  2.1 Basic Theorems .................................. 2

3 Summary ............................................... 3

Objectives

By the end of this lecture, you will be able to:

- use some new terms — big-theta and big-omega — for describing function growth
- use theorems about properties of Big-O as a toolkit for analyzing procedures

1 More analysis terminology

We know that \( O(n \to n) \) is the set of all functions from \( \mathbb{N} \) to \( \mathbb{N} \), and we can write \( f \in O(g) \) to indicate that the function \( f \) grows no faster than \( g \) (at least up to constant multiples). But while the function \( n \mapsto 1 \) grows no faster than the function \( n \mapsto n \), we have the sense that it’s substantially smaller, not just “off by a factor of 2 or 3,” for instance. That’s confirmed by our knowledge that \( n \mapsto n \notin O(n \mapsto 1) \). Fortunately, just as big-O captures the idea that one function grows slower than another, there’s another notation that means the opposite: we define \( \Omega(f) \) as the set of all functions that eventually dominate \( f \), up to constants. In symbols,

\[
  f \in O(g) \iff g \in \Omega(f).
\]

So because \( n \mapsto 1 \in O(n \mapsto n) \), we can write

\[
  n \mapsto n \in \Omega(n \mapsto 1).
\]

Informally, \( \Omega(f) \) consists of functions that grow at least as fast as \( f \).

By contrast, the functions \( n \mapsto n \) and \( n \mapsto 2n + 1 \) really do seem to be “the same size”; that’s reflected by the fact that \( O(n \mapsto n) = O(n \mapsto 2n + 1) \). There’s a notation for that as well. We write

\[
  f \in \Theta(g)
\]

to mean that \( f \in O(g) \) and \( f \in \Omega(g) \), or, equivalently, that \( f \in O(g) \) and \( g \in O(f) \), or, also equivalently, that \( O(f) = O(g) \). So, \( f \) is eventually less than \( g \) up to constants, and \( g \) is eventually
less that \( f \) up to constants. More informally, \( \Theta(f) \) consists of functions that grow the same way as \( f \).

Most of the analysis we will be doing in this class will be proving our functions lie in Big-O of something, not Big-\( \Omega \) or Big-\( \Theta \). This is because, it is quite hard to accurately say how much time a function will “at least” take to run. Take for example, the contains17? procedure. In the case where 17 is the first item in the list, we are done in constant time! So Big-\( \Omega \) will just be \( \Omega(1) \).

Next consider the length procedure. Our conjecture, though operation counting and plug and chug was:

\[
L(n) = B(n) + A
\]

We can prove that \( L \in O(n \mapsto n) \) by choosing \( M = 1, c = A + B \). It is also true that the function \( n \mapsto n \) is in \( O(L) \), by choosing \( M = 1, c = \frac{1}{B} \). So the runtime of the procedure is not just “at worst linear time”, but is in fact “linear time”! This is true for the length procedure because no matter what the list is, we need to go though every item to find the length. So we can say \( L \in \Theta(n \mapsto n) \).

People often speak of big-O pretty informally, and say things like “Sorting is \( O(n \mapsto n \log n) \)”, when what they really mean is “A reasonably efficient sort algorithm can be written that takes time \( n \mapsto n \log n \), and you can’t find one that runs any faster than that.” In short, they mean that the op-counting function for a decent sorting algorithm is in \( \Theta(n \mapsto n \log n) \). Beware of this usage. It’s sloppy and wrong and misleading to others.

## 2 Theorems for proving Big-O class

We will now prove some useful theorems that will help you find the Big-O category of your conjectures.

### 2.1 Basic Theorems

Firstly, let us prove that any function \( f : \mathbb{N} \mapsto \mathbb{N} \), is in its own Big-O category.

**Theorem 1.** For any function \( f : \mathbb{N} \mapsto \mathbb{N} \), \( f \in O(f) \)

**Proof.** Pick \( M = 1, c = 1 \)

Suppose that \( n > M \). [completely unnecessary in this case]

I claim that \( f(n) \leq cf(n) \) for \( n > M \)

Rewrite: \( f(n) \leq 1f(n) \) for \( n > M \)

Therefore, our claim is true. \( \diamond \)

Next, we prove that a constant function with value 0 is in the Big-O class of any function \( f : \mathbb{N} \mapsto \mathbb{N} \).

**Theorem 2.** For any function \( f : \mathbb{N} \mapsto \mathbb{N} \) and \( Z : \mathbb{N} \mapsto \mathbb{N} : n \mapsto 0 \), \( Z \in O(f) \)

**Proof.** Pick \( M = 1, c = 1 \)

Suppose that \( n > M \). [again, unnecessary in this case]

I claim that \( Z(n) \leq cf(n) \) for \( n > M \)

Rewrite: \( 0 \leq 1f(n) \) for \( n > M \)

Therefore, our claim is true. \( \diamond \)
We prove that for a function \( f : \mathbb{N} \to \mathbb{N} \), \( 2f \) is in the Big-O class of the function \( f \).

**Theorem 3.** For any function \( f : \mathbb{N} \to \mathbb{N} \), \( 2f \in O(f) \)

**Proof.** Pick \( M = 1, c = 2 \)
Suppose that \( n > M \). [still unnecessary in this case]
I claim that \( (2f)(n) \leq cf(n) \) for \( n > M \)
Rewrite: \( (2f)(n) = 2f(n) \leq 2f(n) \) for \( n > M \)
Therefore, our claim is true. \( \diamond \)

Next, we prove that for a function \( f : \mathbb{N} \to \mathbb{N} \), any positive multiple of \( f \) is in the Big-O class of the function \( f \).

**Theorem 4.** For any function \( f : \mathbb{N} \to \mathbb{N} \), and a number \( a > 0 \), \( af \in O(f) \)

**Proof.** Pick \( M = 1, c = a \)
Suppose that \( n > M \). [still unnecessary in this case]
I claim that \( (af)(n) \leq cf(n) \) for \( n > M \)
Rewrite: \( (af)(n) = af(n) \leq af(n) \) for \( n > M \)
Therefore, our claim is true. \( \diamond \)

For our next theorem, we want to prove that if a function \( f \) is in the Big-O class of a function \( g \), which is in turn in the Big-O class of a function \( h \), then \( f \) is also in the Big-O class of function \( h \).

**Theorem 5.** If \( f \in O(g), g \in O(h) \), then \( f \in O(h) \)

Let us first try confirming this for a specific case with concrete values.
Suppose that (1), for \( n > 100 \), we know that \( f(n) \leq 22g(n) \).
And (2), for \( n > 33 \), we know that \( g(n) \leq 4h(n) \).
Then, from (2), we know that for \( n > 100 \), \( g(n) \leq 4h(n) \).
What can we say (for \( n > 100 \)), about the relationship of \( f(n) \) to \( h(n) \)?
\[
\begin{align*}
    f(n) & \leq 22g(n) \\
    g(n) & \leq 4h(n) \\
    22g(n) & \leq 4 \times 22h(n) \\
    f(n) & \leq 4 \times 22h(n)
\end{align*}
\]
And we have proved that our claim holds true for these values! Is it possible to generalize this proof to work for any values of \( n \)? If you observe carefully, our proof doesn’t really require our values to be these exact numbers and will also work in the general case!

**Proof.** For \( n > M_1 \), we have \( f(n) \leq c_1g(n) \).
For \( n > M_2 \), we know that \( g(n) \leq c_2h(n) \).
Then, for \( n > \max(M_1, M_2) \), we have \( f(n) \leq c_1c_2h(n) \).
Hence, \( f \in O(h) \).
To summarize, if \([M_1, c_1]\) and \([M_2, c_2]\) show that \( f \in O(g) \) and \( g \in O(h) \), then \([\max(M_1, M_2), c_1c_2]\) shows that \( f \in O(h) \). \( \diamond \)

Trying out a proof with concrete values as first can be a very useful technique in approaching these proofs. Often, some kind of pattern will emerge which will help you write the general case.
3 Summary

Ideas

- Big-Theta and Big-Omega can also be used describe function growth, and are often confused with Big-O.
- The depth of a tree containing $n$ nodes is approximately $\log(n)$.

Skills

- Have a toolkit of theorems to use when analyzing your own programs and determining what Big-O class they belong to.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci170/feedback](http://cs.brown.edu/courses/csci170/feedback)