Lecture Notes

This lecture covers three topics. While we can prove that a summation formula is correct by induction, finding such formulas in the first place is still difficult. In the first section, we will give a general approach to this problem. The second topic is evaluation of products. In particular, we'll obtain bounds on the familiar product $n!$. The second topic is asymptotic notation; this is a cryptic shorthand that conveys information about the behavior of a function $f(n)$ for large n.

1 Finding Summation Formulas

The source of the simple formula $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ is still a mystery! Sure, we can prove this statement true by induction, but where did the expression on the right come from? Even more inexplicable is the summation formula for consecutive squares:

$$\begin{aligned}
\sum_{i=1}^{n} i^2 &= \frac{(2n+1)(n+1)n}{6} \\
&= \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \\
&\sim \frac{n^3}{3}
\end{aligned}$$

Here is how we might find the sum-of-squares formula if we forgot it or had never seen it. First, the Integral Method gives a quick estimate of the sum:

$$\int_0^n x^2\ dx \leq \sum_{i=1}^{n} i^2 \leq \int_0^n (x+1)^2\ dx$$

$$\frac{n^3}{3} \leq \sum_{i=1}^{n} i^2 \leq \frac{(n+1)^3}{3} - \frac{1}{3}$$

These upper and lower bounds obtained by the Integral Method show that $\sum_{i=1}^{n} i^2 \sim \frac{n^3}{3}$. To get an exact formula, we then guess the general form of the solution. Where we are uncertain, we can add parameters $a, b, c, \ldots$. For example, we might make the guess:

$$\sum_{i=1}^{n} i^2 = an^3 + bn^2 + cn + d$$

If the guess is correct, then we can determine the parameters a, b, c, and d by plugging in a few values for n. Each such value gives a linear equation in a, b, c, and d. If we plug in enough values, we may get a linear system with a unique solution. Applying this method to our example gives:

$$\begin{aligned}
n = 0 &\quad\rightarrow\quad 0 = d \\
n = 1 &\quad\rightarrow\quad 1 = a + b + c + d \\
n = 2 &\quad\rightarrow\quad 5 = 8a + 4b + 2c + d \\
n = 3 &\quad\rightarrow\quad 14 = 27a + 9b + 3c + d
\end{aligned}$$

Solving this system gives the solution $a = 1/3, b = 1/2, c = 1/6, d = 0$. Therefore, if our initial guess at the form of the solution was correct, then the summation is equal to $n^3/3 + n^2/2 + n/6$. In fact, our initial guess *was* correct, this is the right formula for the sum of squares!

Be careful! After obtaining a formula by this method, always go back and prove it using induction or some other method. This is not merely a check for algebra blunders; if the initial guess at the solution was not of the right form, then the resulting formula will be completely wrong!

2 Computing Products: Estimating $n!$

Sometimes one encounters product notation instead of summation notation. By far the most common product is factorial:

$$n! = 1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n = \prod_{i=1}^{n} i$$

Our goal in this section is to find a good, closed-form estimate of $n!$. The best way to handle a product is to convert it into a sum by taking the logarithm. In the case of factorial, this gives:

$$\begin{aligned}
\ln n! &= \ln(1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n) \\
&= \ln 1 + \ln 2 + \ln 3 + \ldots + \ln(n-1) + \ln n \\
&= \sum_{i=1}^{n} \ln i
\end{aligned}$$

We've not seen a summation containing a logarithm before! Fortunately, one tool that we used in evaluating sums is still applicable: the Integral Method. We can bound the terms of this sum with $\ln x$ and $\ln(x+1)$ as shown in Figure 1. This gives bounds on $\ln n!$ as follows:

$$\begin{aligned}
\int_1^n \ln x \; dx \leq &\quad \textstyle\sum_{i=1}^n \ln i \quad \leq \int_0^n \ln(x+1) \; dx \\
n \ln(\frac{n}{e}) + 1 \leq &\quad \textstyle\sum_{i=1}^n \ln i \quad \leq (n+1)\ln(\frac{n+1}{e}) + 1 \\
(\frac{n}{e})^n e \leq &\quad\quad n! \quad\quad \leq (\frac{n+1}{e})^{n+1} e
\end{aligned}$$

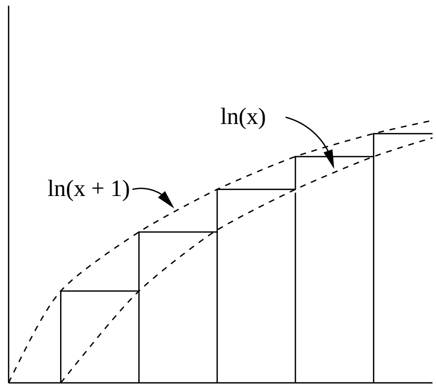

Figure 1: *This figure illustrates the Integral Method for bounding the sum* $\sum_{i=1}^{n} ln\ i$.

The second line follows from the first by completing the integrations. The third line is obtained by exponentiating.

We now have a closed form estimate of $n!$; namely, $n! \approx (\frac{n}{e})^n$. (We are ignoring stray e and n factors.) A more accurate estimate of $n!$ can be obtained with a more careful calculation:

Fact 1 (Stirling's Formula)

$$\sqrt{2\pi n}(\frac{n}{e})^n e^{1/(12n+1)} \leq n! \leq \sqrt{2\pi n}(\frac{n}{e})^n e^{1/12n}$$

The fact implies that $n! \sim (\frac{n}{e})^n \sqrt{2\pi n}$, since $e^{1/(12n+1)}$ and $e^{1/12n}$ both approach 1 as n grows large. These inequalities can be verified by induction, but the details are nasty. In fact, Stirling's formula is actually given incorrectly in the popular algorithms book used in 6.046.

Though there is no closed form for $n!$, the bounds in Stirling's formula are very tight. For example, if $n = 100$, then Stirling's bounds are:

$$\begin{aligned} 100! &\geq \sqrt{200\pi}(\frac{100}{e})^{100} e^{1/1201} \\ 100! &\leq \sqrt{200\pi}(\frac{100}{e})^{100} e^{1/1200} \end{aligned}$$

The only difference between the upper bound and the lower bound is in the final term. In particular $e^{1/1201} \approx 1.00083299$ and $e^{1/1200} \approx 1.00083368$. As result, the upper bound is no more than $1 + 10^{-6}$ times the lower bound. This is amazingly tight! Remember Stirling's formula; we will use it often.

3 Asymptotic Notation

Asymptotic notation is a shorthand used to give a quick measure of the behavior of a function $f(n)$ as n grows large.

The $\sim$ notation used in the last lecture is an example of asymptotic notation. Recall that $f(n) \sim g(n)$ means $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to 1$. For example, we used this notation to describe the growth of the Harmonic numbers:

$$\begin{aligned} H_n &= \sum_{i=1}^{n} \frac{1}{i} \\ &= \ln n + \gamma + \frac{1}{2n} + \frac{1}{12n^2} + \frac{\epsilon(n)}{120n^4} \\ &\sim \ln n \end{aligned}$$

In other words, $\lim_{n\to\infty} \frac{H_n}{\ln n} = 1$, since the remaining terms become irrelevant as n grows large. Similarly, we saw in the preceding section that $n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$.

There are other forms of asymptotic notation besides $\sim$, however. In particular, there are 5 other symbols to learn:

- O – "Oh" or "Big Oh"
- o – "Little Oh"
- Θ – "Theta"
- ω – "Little Omega"
- Ω – "Omega"

Learning notation can be a bit irritating, but these symbols will be used heavily in 6.046, 6.045, and other courses. Furthermore, it will be on the quiz.

3.1 Big Oh

Big Oh is the most frequently used asymptotic notation. It is used to give an upper bound on the growth of a function, such as the running time of an algorithm.

Definition Given functions $f, g : \mathcal{R} \mapsto \mathcal{R}$, we say that $f(x) = O(g(x))$ if there exists a constant $c > 0$ and an x_0 such that for all $x \geq x_0$, $|f(x)| \leq cg(x)$.

This definition is rather complicated, but the idea is simple; the statement $f(x) = O(g(x))$ means that f grows no faster than g. More precisely, f is smaller than some constant times g for all sufficiently large values of x. In a sense, $f(x) = O(g(x))$means $f(x) \leq g(x)$, except that we're willing to ignore a constant factor (c) and to allow exceptions for small x (i.e., $x < x_0$).

These are many ways of expressing the idea $f(x) = O(g(x))$. Sometimes people will write $f(x) \leq O(g(x))$, $f(x)$ is $O(g(x))$, $f(x) \in O(g(x))$, or $f(x)$ "is on the order of" $g(x)$. These all mean the same thing.

We are ready for some examples. First, is $x = O(x^2)$? It seems that the answer should be "yes", because x grows no faster than x^2. The following theorem proves this intuition correct.

Theorem 3.1 If $f(x) = x$ and $g(x) = x^2$, then $f(x) = O(g(x))$.

Proof. We must prove that there exists a constant $c > 0$ and an x_0 such that for all $x \geq x_0$, $|x| \leq cx^2$. In particular, we can choose $c = 1$ and $x_0 = 1$. Then for all $x \geq 1$, $|x| \leq 1 \cdot x^2$. ■

As a shorthand, the statement of the preceding theorem could have been written $x = O(x^2)$. In fact, this notation is widely used, but we will see that it introduces ambiguities that can be very difficult to sort out.

How about the reverse statement, $x^2 = O(x)$? This is not true; x^2 grows faster than x asymptotically.

Theorem 3.2 $x^2 \neq O(x)$

Proof. The proof is by contradiction. Assume for the purpose of contradiction that $x^2 = O(x)$. Then there exists a constant $c > 0$ and an x_0 such that for all $x \geq x_0$, $|x^2| \leq cx$. This implies that for all $x \geq x_0$, $x \leq c$. However, this is false for $x \geq \max(c + 1, x_0)$. ■

The statement $x^2 = O(x)$ is false because x grows too slowly. We might hope, therefore, that $x^2 = O(100x)$ is true because $100x$ grows much more quickly that x. This statement is also false, however. The proof is the same as before. The factor of 100 does not matter; eventually, the quadratic function x^2 will grow much faster and get much bigger than the linear function $100x$.

Theorem 3.3 $100x^2 = O(x^2)$

Proof. Choose $c = 100$ and $x_0 = 1$. Then the theorem holds, since for all $x \geq 1$, $|100x^2| \leq 100x^2$ ■

Theorem 3.4 $x^2 + 100x + 10 = O(x^2)$

Proof. Choose $c = 111$ and $x_0 = 1$. Then the theorem holds, since for all $x \geq 1$, $|x^2 + 100x + 10| \leq 111x^2$. ■

We can generalize the preceding theorem to an arbitrary quadratic polynomial.

Theorem 3.5 $a_1x^2 + a_2x + a_3 = O(x^2)$

Proof. Choose $c = |a_1| + |a_2| + |a_3|$ and $x_0 = 1$. Then the theorem holds, since for all $x \geq 1$, $|a_1x^2 + a_2x + a_3| \leq (|a_1| + |a_2| + |a_3|)x^2$. ■

Intuitively, exponential functions grow faster than polynomials, even very high degree polynomials.

Theorem 3.6 $x^{10} = O(e^x)$

Proof. Choose $c = 1$ and $x_0 = e^5$. It can be proven using calculus that $10 \ln x \leq x$ for all $x \geq e^5$. Exponentiating gives that for all $x \geq e^5$, we have $e^{10 \ln x} \leq e^x$ or equivalently $x^{10} \leq e^x$. ∎

Big Oh notation is especially useful when describing the running time of an algorithm. For example, suppose that an algorithm for multiplying $n \times n$ matrices runs in cubic time. This fact can be expressed concisely by saying that the running time is $T(n) = O(n^3)$. This allows the speed of an algorithm to be discussed without reference to constant factors or lower-order terms that might be machine specific.

Just knowing that the running time of an algorithm is $O(n^3)$ is useful, because if n doubles we can predict that the running time will increase by a factor of about 8 for large n. In this way, Big Oh notation preserves information about the scalability of an algorithm or system. Scalability is, of course, a big issue in the design of algorithms and systems.

3.2 Pitfalls with Big Oh

There is a long list of ways to make mistakes with Big Oh notation. This section presents some of the ways that Big Oh notation can lead to ruin and despair.

The Exponential Fiasco

Sometimes relationships involving Big Oh are not so obvious. For example, one might guess that $4^x = O(2^x)$ since 4 is only a constant factor larger than 2. This is reasoning is incorrect, however; actually 4^x grows much faster than 2^x.

Theorem 3.7 $4^x \neq O(2^x)$

Proof. The proof is by contradiction. Assume for the purpose of contradiction that $4^x = O(2^x)$. Then there exists a constant $c > 0$ and an x_0 such that for all $x \geq x_0$, $|4^x| \leq c2^x$. This implies that for all $x \geq x_0$, $2^x \leq c$. But this assertion is false for $x \geq \max(\ln(c+1), x_0)$. ∎

Constant Confusion

Every constant is $O(1)$. For example, $17 = O(1)$. This is true because if we let $f(x) = 17$ and $g(x) = 1$, then there exists a $c > 0$ and an x_0 such that $|f(x)| \leq cg(x)$. In particular, we could choose $c = 17$ and $x_0 = 1$, since $|17| \leq 17 \cdot 1$ for all $x \geq 1$. Since we can prove that any constant is $O(1)$, we can show that for all n, $n = O(1)$.

Now here is the key question: Is $n = O(1)$? Didn't we just answer this? Not quite. This question is ambiguous; there are two possible interpretations of the notation, and these give rise to two different answers!

The core problem is that n can refer to either of two functions. On the one hand, n might refer to the constant function that takes on the value n everywhere. That is, by n we mean the function f where for all x, $f(x) = n$. In this case, $n = O(1)$ as we showed above.

On the other hand, n might refer to the function that grows linearly. In this case, by n we mean the function f where for all n, $f(n) = n$. Now $n \neq O(1)$.

The reason that this confusion is particularly associated with $O(1)$ and not with $O(\ln n)$ or $O(x^2)$ is that $O(1)$ contains no reference to a variable.

If we ask, "Is $n = O(\ln n)$?", then it is fairly clear that n refers to the function that grows linearly, and the answer is no. On the other hand if we ask, "Is $n = O(\ln x)$?", then probably n refers to the constant function that takes on value n everywhere, and the answer is yes. However, if we ask, "Is $n = O(1)$?", then we do not know what function n represents, and we can not say for certain.

We can construct a false theorem that exploits this ambiguity.

False Theorem 3.8

$$\sum_{i=1}^{n} i = O(n)$$

Recall that in reality $\sum_{i=1}^{n} i = n(n+1)/2 \neq O(n)$.

Proof. Define $f(n) = \sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$. Since we have shown that every constant i is $O(1)$, $f(n) = O(1) + O(1) + \ldots + O(1) = O(n)$. ∎

The error stems from confusion over what is meant in the statement $i = O(1)$. It is true that $i = 1, 2, 3$, and even the first 100 values of i are $O(1)$. But not all values are $O(1)$; for example, the last value of i in the summation is equal to n is therefore not $O(1)$. To avoid this error, just do not use $i = O(1)$ if there is any chance i can grow with a parameter.

Induction Destruction

Let's try to prove the same false theorem another way.

False Theorem 3.9

$$\sum_{i=1}^{n} i = O(n)$$

Proof. The proof is by induction on n. Let $P(n)$ be the predicate that $f(n) = O(n)$ where $f(n) = \sum_{i=1}^{n} i$. In the base case, $P(1)$ is true because $f(1) = 1 = O(1)$. In the inductive step, assume that $f(n) = O(n)$ to prove that $f(n+1)$ is $O(n)$. Observe that $f(n+1) = f(n) + (n+1) = O(n) + O(n) = O(n)$. ∎

There is no error in the proof; the problem is that the proof does not imply the theorem! We have proven that for all $n \geq 1$, $\sum_{i=1}^{n} i = O(n)$. More precisely, this says that for each $n \geq 1$, the constant represented by the summation $\sum_{i=1}^{n} i$ is $O(n)$. This is true; in fact, every constant is $O(1)$. What we wanted to prove was that the function $f(n) = \sum_{i=1}^{n} i$ is $O(n)$. The notation obscures this essential distinction.

In general, never mix induction and Big Oh notation. Instead, here is a correct approach to proving an asymptotic upper bound by induction. Let $f(n) = \sum_{i=1}^{n} i$. To prove that $f(n) = O(n)$, we have to show that there exists a constant $c > 0$ and an n_0 such that for all $n \geq n_0$, $f(n) \leq cn$. Suppose we choose and try to prove the false theorem by induction on n.

Proof (attempt). The proof is by induction. Let $P(n)$ be the predicate that $f(n) \leq cn$. In the base case, $P(1)$ is true because $f(1) \leq c$. *(Let's assume that we chose c large enough so that this holds.)* In the inductive step, assume that $f(n) < cn$ to prove that $f(n+1) \leq c(n+1)$. By the definition of $f(n)$, we have $f(n+1) = f(n) + n + 1$. By induction, this is at most $cn + n + 1 = (c+1)n + 1$.

Now we are stuck; to complete the inductive step, we must prove that $(c+1)n + 1 \leq c(n+1)$. However, this is simply not true, regardless of how large a value c is chosen. If n becomes very large, then the left side of the inequality is about $(c+1)n$, and the right side is about cn. Therefore, the inequality does not hold and the proof does not go through.

Lower Bound Blunder

Sometimes people incorrectly use Big Oh in the context of a lower bound. For example, they might say, "The running time is at least $O(n^2)$." We have defined a meaning for $f(n) = O(g(n))$ and assigned the same meaning to $f(n) \leq O(g(n))$, but $f(n) \geq O(g(n))$ is meaningless. In the next section we will introduce asymptotic notation to describe lower bounds.