CS 362, Pre Lecture 2

Jared Saia
University of New Mexico
Today’s Outline

• L’Hopital’s Rule
• Log Facts
• Recurrence Relations
For any functions $f(n)$ and $g(n)$ which approach infinity and are differentiable, L’Hopital tells us that:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f'(n)}{g'(n)}$$
Example

- Q: Which grows faster ln\(n\) or \(\sqrt{n}\)?
- Let \(f(n) = \ln n\) and \(g(n) = \sqrt{n}\)
- Then \(f'(n) = 1/n\) and \(g'(n) = (1/2)n^{-1/2}\)
- So we have:

\[
\lim_{n \to \infty} \frac{\ln n}{\sqrt{n}} = \lim_{n \to \infty} \frac{1/n}{(1/2)n^{-1/2}} = \lim_{n \to \infty} \frac{2}{n^{1/2}} = 0
\]

- Thus \(\sqrt{n}\) grows faster than \(\ln n\) and so \(\ln n = o(\sqrt{n})\)
Formal Proof

- $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ formally means that for all positive $c$, there exists $n_0$ such that $\frac{f(n)}{g(n)} < c$ for all $n \geq n_0$
- Cross multiplying: for all positive $c$, there exists $n_0$ such that $f(n) < cg(n)$ for all $n \geq n_0$
- Hence, $f(n) = o(g(n))$ (by definition of $o()$)
- So showing the limit of the ratio is 0 proves the $o()$ relationship.
A digression on logs

*It rolls down stairs alone or in pairs,*  
*and over your neighbor’s dog,*  
*it’s great for a snack or to put on your back,*  
*it’s log, log, log!*  
- “The Log Song” from the Ren and Stimpy Show

- The log function shows up very frequently in algorithm analysis
- As computer scientists, when we use log, we’ll mean $\log_2$  
  (i.e. if no base is given, assume base 2)
Definition

- $\log_x y$ is by definition the value $z$ such that $x^z = y$
- $x^{\log_x y} = y$ by definition
Examples

- \( \log 1 = 0 \)
- \( \log 2 = 1 \)
- \( \log 32 = 5 \)
- \( \log 2^k = k \)

Note: \( \log n \) is way, way smaller than \( n \) for large values of \( n \)
Examples

• \( \log_3 9 = 2 \)
• \( \log_5 125 = 3 \)
• \( \log_4 16 = 2 \)
• \( \log_{24} 24^{100} = 100 \)
Facts about exponents

Recall that:

- \((xy)^z = x^{yz}\)
- \(x^yx^z = xy^z\)

From these, we can derive some facts about logs.
Facts about logs

To prove both equations, raise both sides to the power of 2, and use facts about exponents

• Fact 1: \( \log(xy) = \log x + \log y \)
• Fact 2: \( \log a^c = c \log a \)

Memorize these two facts
Incredibly useful fact about logs

- Fact 3: $\log_c a = \log a / \log c$

To prove this, consider the equation $a = c^{\log_c a}$, take $\log_2$ of both sides, and use Fact 2. **Memorize this fact**
Log facts to memorize

- Fact 1: $\log(xy) = \log x + \log y$
- Fact 2: $\log a^c = c \log a$
- Fact 3: $\log_c a = \log a / \log c$

These facts are sufficient for all your logarithm needs. (You just need to figure out how to use them)
Logs and $O$ notation

- Note that $\log_8 n = \log n / \log 8$.
- Note that $\log_{600} n^{200} = 200 \times \log n / \log 600$.
- Note that $\log_{100000} 30 \times n^2 = 2 \times \log n / \log 100000 + \log 30 / \log 100000$.
- Thus, $\log_8 n$, $\log_{600} n^{600}$, and $\log_{100000} 30 \times n^2$ are all $O(\log n)$.
- In general, for any constants $k_1$ and $k_2$, $\log_{k_1} n^{k_2} = k_2 \log n / \log k_1$, which is just $O(\log n)$.
Take Away

- All log functions of form $k_1 \log_{k_2} k_3 * n^{k_4}$ for constants $k_1, k_2, k_3$ and $k_4$ are $O(\log n)$
- For this reason, we don’t really “care” about the base of log functions not in exponents when we do asymptotic notation
- Thus, binary search, ternary search and k-ary search all take $O(\log n)$ time
Important Note

- $\log^2 n = (\log n)^2$
- $\log^2 n$ is $O(\log^2 n)$, not $O(\log n)$
- This is true since $\log^2 n$ grows asymptotically faster than $\log n$
- All log functions of form $k_1 \log_{k_3}^{k_2} k_4 * n^{k_5}$ for constants $k_1, k_2, k_3, k_4$ and $k_5$ are $O(\log^{k_2} n)$
Simplify and give $O$ notation for the following functions. In the big-O notation, write all logs base 2:

- $\log 10n^2$
- $\log^2 n^4$
- $2^{\log_4 n}$
- $\log \log \sqrt{n}$
Does big-O really matter?

Let \( n = 100000 \) and \( \Delta t = 1\mu s \)

\[
\begin{align*}
\log n & \quad 1.7 \times 10^{-5} \text{ seconds} \\
\sqrt{n} & \quad 3.2 \times 10^{-4} \text{ seconds} \\
n & \quad .1 \text{ seconds} \\
n \log n & \quad 1.2 \text{ seconds} \\
n \sqrt{n} & \quad 31.6 \text{ seconds} \\
n^2 & \quad 2.8 \text{ hours} \\
n^3 & \quad 31.7 \text{ years} \\
2^n & \quad > 1 \text{ century}
\end{align*}
\]

(from Classic Data Structures in C++ by Timothy Budd)
Recurrence Relations

“Oh how should I not lust after eternity and after the nuptial ring of rings, the ring of recurrence” - Friedrich Nietzsche, Thus Spoke Zarathustra

- Getting the run times of recursive algorithms can be challenging
- Consider an algorithm for binary search (next slide)
- Let $T(n)$ be the run time of this algorithm on an array of size $n$
- Then we can write $T(1) = 1$, $T(n) = T(n/2) + 1$
Alg: Binary Search

bool BinarySearch (int arr[], int s, int e, int key){
    if (e-s<=0) return false;
    int mid = (e+s)/2;
    if (key==arr[mid]){ return true; }
    else if (key < arr[mid]){ return BinarySearch (arr,s,mid,key); }
    else{ return BinarySearch (arr,mid,e,key); }
}
Recurrence Relations

- \( T(n) = T(n/2) + 1 \) is an example of a recurrence relation
- A *Recurrence Relation* is any equation for a function \( T \), where \( T \) appears on both the left and right sides of the equation.
- We always want to “solve” these recurrence relation by getting an equation for \( T \), where \( T \) appears on just the left side of the equation
Recurrence Relations

• Whenever we analyze the run time of a recursive algorithm, we will first get a recurrence relation
• To get the actual run time, we need to solve the recurrence relation
Substitution Method

- One way to solve recurrences is the substitution method aka “guess and check”
- What we do is make a good guess for the solution to $T(n)$, and then try to prove this is the solution by induction
Example

• Let’s guess that the solution to \( T(n) = T(n/2) + 1, \ T(1) = 1 \) is \( T(n) = O(\log n) \)

• In other words, \( T(n) \leq c \log n \) for all \( n \geq n_0 \), for some positive constants \( c, n_0 \)

• We can prove that \( T(n) \leq c \log n \) is true by plugging back into the recurrence
Proof

We prove this by induction:

• B.C.: $T(2) = 2 \leq c \log 2$ provided that $c \geq 2$
• I.H.: For all $j < n$, $T(j) \leq c \log(j)$
• I.S.:

$$T(n) = T(n/2) + 1$$
$$\leq (c \log(n/2)) + 1$$
$$= c \log n - \log 2 + 1$$
$$= c \log n - c + 1$$
$$\leq c \log n$$

First step holds by IH. Last step holds for all $n > 0$ if $c \geq 1$. Thus, entire proof holds if $n \geq 2$ and $c \geq 2$. 

24
Recurrences and Induction

Recurrences and Induction are closely related:

- To *find* a solution to $f(n)$, solve a recurrence
- To *prove* that a solution for $f(n)$ is correct, use induction

*For both recurrences and induction, we always solve a big problem by reducing it to smaller problems!*
Some Examples

- The next several problems can be attacked by induction/recurrences
- For each problem, we’ll need to reduce it to smaller problems
- Question: How can we reduce each problem to a smaller subproblem?
Sum Problem

• $f(n)$ is the sum of the integers $1, \ldots, n$
Tree Problem

- $f(n)$ is the maximum number of leaf nodes in a binary tree of height $n$

Recall:

- In a binary tree, each node has at most two children
- A $leaf$ node is a node with no children
- The height of a tree is the length of the longest path from the root to a leaf node.
Binary Search Problem

- $f(n)$ is the maximum number of queries that need to be made for binary search on a sorted array of size $n$. 
Dominoes Problem

- \( f(n) \) is the number of ways to tile a 2 by \( n \) rectangle with dominoes (a domino is a 2 by 1 rectangle)
Simpler Subproblems

- Sum Problem: What is the sum of all numbers between 1 and $n - 1$ (i.e. $f(n - 1)$)?
- Tree Problem: What is the maximum number of leaf nodes in a binary tree of height $n - 1$? (i.e. $f(n - 1)$)
- Binary Search Problem: What is the maximum number of queries that need to be made for binary search on a sorted array of size $n/2$? (i.e. $f(n/2)$)
- Dominoes problem: What is the number of ways to tile a 2 by $n - 1$ rectangle with dominoes? What is the number of ways to tile a 2 by $n - 2$ rectangle with dominoes? (i.e. $f(n - 1), f(n - 2)$)
Recurrences

- Sum Problem: $f(n) = f(n - 1) + n, \ f(1) = 1$
- Tree Problem: $f(n) = 2 \times f(n - 1), \ f(0) = 1$
- Binary Search Problem: $f(n) = f(n/2) + 1, \ f(2) = 1$
- Dominoes problem: $f(n) = f(n - 1) + f(n - 2), \ f(1) = 1, \ f(2) = 2$
Guesses

- Sum Problem: $f(n) = (n + 1)n/2$
- Tree Problem: $f(n) = 2^n$
- Binary Search Problem: $f(n) = \log n$
- Dominoes problem: $f(n) = \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left( \frac{1-\sqrt{5}}{2} \right)^n$
Inductive Proofs

“Trying is the first step to failure” - Homer Simpson

- Now that we’ve made these guesses, we can try using induction to prove they’re correct (the substitution method)
- We’ll give inductive proofs that these guesses are correct for the first three problems
Sum Problem

- Want to show that $f(n) = (n + 1)n/2$.
- Prove by induction on $n$
- Base case: $f(1) = 2 \times 1/2 = 1$
- Inductive hypothesis: for all $j < n$, $f(j) = (j + 1)j/2$
- Inductive step:

$$f(n) = f(n - 1) + n$$
$$= n(n - 1)/2 + n$$
$$= (n + 1)n/2$$

Where the first step holds by IH.
Tree Problem

- Want to show that $f(n) = 2^n$.
- Prove by induction on $n$
- Base case: $f(0) = 2^0 = 1$
- Inductive hypothesis: for all $j < n$, $f(j) = 2^j$
- Inductive step:

$$f(n) = 2 \cdot f(n-1)$$
$$= 2 \cdot (2^{n-1})$$
$$= 2^n$$

Where the first step holds by IH.
Binary Search Problem

- Want to show that $f(n) = \log n$. (assume $n$ is a power of 2)
- Prove by induction on $n$
- Base case: $f(2) = \log 2 = 1$
- Inductive hypothesis: for all $j < n$, $f(j) = \log j$
- Inductive step:

$$f(n) = f(n/2) + 1$$
$$= \log n/2 + 1$$
$$= \log n - \log 2 + 1$$
$$= \log n$$

Where the first step holds by IH.
Exercise

- Consider the recurrence $f(n) = 2f(n/2) + 1$, $f(1) = 1$
- Guess that $f(n) \leq cn - 1$:
- Q1: Show the base case - for what values of $c$ does it hold?
- Q2: What is the inductive hypothesis?
- Q3: Show the inductive step.
Reading

- Chapter 3 and 4, and Appendices in the text