CS 362, Lecture 2

Jared Saia
University of New Mexico
Today’s Outline

- L’Hopital’s Rule
- Log Facts
- Recurrence Relation Review
- Recursion Tree Method
- Master Method
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 \( \ln n = O(\sqrt{n}) \).
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:

• \((xy)^z = xyz\)
• \(xy^z = xy+z\)

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: $\log_c a = \frac{\log a}{\log c}$

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)
• Note that \( \log_8 n = \log n / \log 8 \).
• Note that \( \log_{600} n^{200} = 200 \cdot \log n / \log 600 \).
• Note that \( \log_{100000} 30 \cdot n^2 = 2 \cdot \log n / \log 100000 + \log 30 / \log 100000 \).
• Thus, \( \log_8 n \), \( \log_{600} n^{600} \), and \( \log_{100000} 30 \cdot n^2 \) are all \( \mathcal{O}(\log n) \).
• 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 \( \mathcal{O}(\log n) \).
• All log functions of form \( k_1 \log_{k_2} k_3 \cdot n^{k_4} \) 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 the log function 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 \( k_1 \log^{k_2}_{k_3} k_4 \ast n^{k_5} \) 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:

- $\log 10n^2$
- $\log^2 n^4$
- $2^{\log_4 n}$
- $\log \log \sqrt{n}$
Recurrences and Inequalities

- Often easier to prove that a recurrence is no more than some quantity than to prove that it equals something.
- Consider: \( f(n) = f(n-1) + f(n-2), \ f(1) = f(2) = 1 \)
- “Guess” that \( f(n) \leq 2^n \)
Goal: Prove by induction that for \( f(n) = f(n - 1) + f(n - 2) \), \( f(1) = f(2) = 1 \), \( f(n) \leq 2^n \)

- Base case: \( f(1) = 1 \leq 2^1 \), \( f(2) = 1 \leq 2^2 \)
- Inductive hypothesis: for all \( j < n \), \( f(j) \leq 2^j \)
- Inductive step:

\[
\begin{align*}
  f(n) & = f(n - 1) + f(n - 2) \\
  & \leq 2^{n-1} + 2^{n-2} \\
  & < 2 \times 2^{n-1} \\
  & = 2^n
\end{align*}
\]
Recursion-tree method

- Each node represents the cost of a single subproblem in a recursive call
- First, we sum the costs of the nodes in each level of the tree
- Then, we sum the costs of all of the levels
Recursion-tree method

- Used to get a good guess which is then refined and verified using substitution method
- Best method (usually) for recurrences where a term like $T(n/c)$ appears on the right hand side of the equality
• Consider the recurrence for the running time of Mergesort:
  \[ T(n) = 2T(n/2) + n, \quad T(1) = O(1) \]
Example 1

- We can see that each level of the tree sums to $n$
- Further the depth of the tree is $\log n$ ($n/2^d = 1$ implies that $d = \log n$).
- Thus there are $\log n + 1$ levels each of which sums to $n$
- Hence $T(n) = \Theta(n \log n)$
Example 2

- Let’s solve the recurrence $T(n) = 3T(n/4) + n^2$
- Note: For simplicity, from now on, we’ll assume that $T(i) = \Theta(1)$ for all small constants $i$. This will save us from writing the base cases each time.
We can see that the $i$-th level of the tree sums to $(3/16)^i n^2$.

Further the depth of the tree is $\log_4 n$ ($n/4^d = 1$ implies that $d = \log_4 n$)

So we can see that $T(n) = \sum_{i=0}^{\log_4 n} (3/16)^i n^2$
Solution

\[ T(n) = \log_4 n \sum_{i=0}^{\log_4 n} (3/16)^i n^2 \]  \hspace{1cm} (8)

\[ < n^2 \sum_{i=0}^{\infty} (3/16)^i \]  \hspace{1cm} (9)

\[ = \frac{1}{1 - (3/16)} n^2 \]  \hspace{1cm} (10)

\[ = O(n^2) \]  \hspace{1cm} (11)
• Divide and conquer algorithms often give us running-time recurrences of the form

\[ T(n) = aT(n/b) + f(n) \]  \hspace{2cm} (12)

• Where \( a \) and \( b \) are constants and \( f(n) \) is some other function.
• The so-called “Master Method” gives us a general method for solving such recurrences when \( f(n) \) is a simple polynomial.
- Unfortunately, the Master Theorem doesn’t work for all functions $f(n)$
- Further many useful recurrences don’t look like $T(n)$
- However, the theorem allows for very fast solution of recurrences when it applies
Master Theorem

- Master Theorem is just a special case of the use of recursion trees
- Consider equation $T(n) = aT(n/b) + f(n)$
- We start by drawing a recursion tree
The Recursion Tree

- The root contains the value $f(n)$
- It has $a$ children, each of which contains the value $f(n/b)$
- Each of these nodes has $a$ children, containing the value $f(n/b^2)$
- In general, level $i$ contains $a^i$ nodes with values $f(n/b^i)$
- Hence the sum of the nodes at the $i$-th level is $a^i f(n/b^i)$
• The tree stops when we get to the base case for the recurrence
• We’ll assume \( T(1) = f(1) = \Theta(1) \) is the base case
• Thus the depth of the tree is \( \log_b n \) and there are \( \log_b n + 1 \) levels
• Let $T(n)$ be the sum of all values stored in all levels of the tree:

$$T(n) = f(n) + a f(n/b) + a^2 f(n/b^2) + \cdots + a^i f(n/b^i) + \cdots + a^L f(n/b^L)$$

• Where $L = \log_b n$ is the depth of the tree
• Since $f(1) = \Theta(1)$, the last term of this summation is $\Theta(a^L) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$
A “Log Fact” Aside

- It’s not hard to see that \( a^{\log_b n} = n^{\log_b a} \)

\[
\begin{align*}
  a^{\log_b n} & = n^{\log_b a} \quad (13) \\
  a^{\log_b n} & = a^{\log_a n \cdot \log_b a} \quad (14) \\
  \log_b n & = \log_a n \cdot \log_b a \quad (15)
\end{align*}
\]

- We get to the last eqn by taking \( \log_a \) of both sides
- The last eqn is true by our third basic log fact
Master Theorem

- We can now state the Master Theorem
- We will state it in a way slightly different from the book
- Note: The Master Method is just a “short cut” for the recursion tree method. It is less powerful than recursion trees.
Master Method

The recurrence \( T(n) = aT(n/b) + f(n) \) can be solved as follows:

- If \( \frac{a f(n/b)}{f(n)} \leq K \) for some constant \( K < 1 \), then \( T(n) = \Theta(f(n)) \).
- If \( \frac{a f(n/b)}{f(n)} \geq K \) for some constant \( K > 1 \), then \( T(n) = \Theta(n \log_b a) \).
- If \( \frac{a f(n/b)}{f(n)} = 1 \), then \( T(n) = \Theta(f(n) \log_b n) \).
Proof

• If $f(n)$ is a constant factor larger than $a f(n/b)$, then the sum is a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$.

• If $f(n)$ is a constant factor smaller than $a f(n/b)$, then the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $\Theta(n^{\log_b a})$.

• Finally, if $a f(n/b) = f(n)$, then each of the $L + 1$ terms in the summation is equal to $f(n)$.
Example

- $T(n) = T(3n/4) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 1, b = 4/3, f(n) = n$
- Here $af(n/b) = 3n/4$ is smaller than $f(n) = n$ by a factor of $4/3$, so $T(n) = \Theta(n)$
• Karatsuba’s multiplication algorithm: \( T(n) = 3T(n/2) + n \)
• If we write this as \( T(n) = aT(n/b) + f(n) \), then \( a = 3, b = 2, f(n) = n \)
• Here \( af(n/b) = 3n/2 \) is bigger than \( f(n) = n \) by a factor of \( 3/2 \), so \( T(n) = \Theta(n^\log_2 3) \)
Example

- **Mergesort:** $T(n) = 2T(n/2) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 2, b = 2, f(n) = n$
- Here $af(n/b) = f(n)$, so $T(n) = \Theta(n \log n)$
Example

- \( T(n) = T(n/2) + n \log n \)
- If we write this as \( T(n) = aT(n/b) + f(n) \), then \( a = 1, b = 2, f(n) = n \log n \)
- Here \( af(n/b) = n/2 \log n/2 \) is smaller than \( f(n) = n \log n \) by a constant factor, so \( T(n) = \Theta(n \log n) \)
In-Class Exercise

• Consider the recurrence: \( T(n) = 4T(n/2) + n \log n \)
• Q: What is \( f(n) \) and \( af(n/b) \)?
• Q: Which of the three cases does the recurrence fall under (when \( n \) is large)?
• Q: What is the solution to this recurrence?
In-Class Exercise

- Consider the recurrence: $T(n) = 2T(n/4) + n \lg n$
- Q: What is $f(n)$ and $a f(n/b)$?
- Q: Which of the three cases does the recurrence fall under (when $n$ is large)?
- Q: What is the solution to this recurrence?
Take Away

- Recursion tree and Master method are good tools for solving many recurrences
- However these methods are limited (they can’t help us get guesses for recurrences like $f(n) = f(n - 1) + f(n - 2)$)
- For info on how to solve these other more difficult recurrences, review the notes on annihilators on the class web page.
Todo

- Read Chapter 3 and 4 in the text
- Work on Homework 1