CS 561, Lecture 3

Jared Saia
University of New Mexico

Today's Outline

"Listen and Understand! That terminator is out there. It can't be bargained with, it can't be reasoned with! It doesn't feel pity, remorse, or fear. And it absolutely will not stop, ever, until you are dead!" - The Terminator

- Guess and Check with Inequalities
- Solving Recurrences using Recursion Trees
- Solving Recurrences using the Masters Method
- Solving Recurrences using Annihilators

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 $f(n) \leq 2^n$

Inequalities (II)

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:

$$\begin{aligned} f(n) &= f(n-1) + f(n-2) && (1) \\ &\leq 2^{n-1} + 2^{n-2} && (2) \\ &< 2 * 2^{n-1} && (3) \\ &= 2^n && (4) \end{aligned}$$

Recursion-tree method

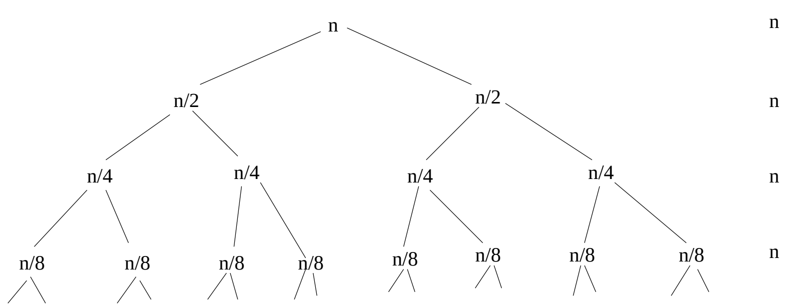
- Each node represents the cost of a single subproblem in a recursive call
- First, we sum the costs of the nodes in each level of the tree
- Then, we sum the costs of all of the levels

Recursion-tree method

- Can use 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

Example 1

- Consider the recurrence for the running time of Mergesort: $T(n) = 2T(n/2) + n$, $T(1) = O(1)$

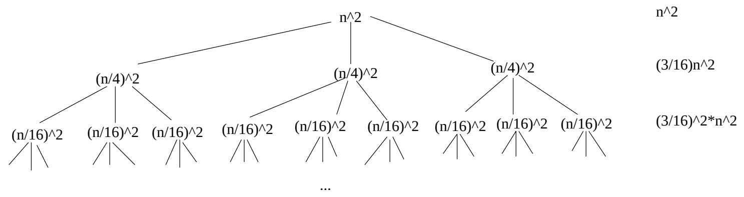


Example 1

- 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.



Example 2

- 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

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n} (3/16)^i n^2 && (5)\\
&< n^2 \sum_{i=0}^{\infty} (3/16)^i && (6)\\
&= \frac{1}{1-(3/16)} n^2 && (7)\\
&= O(n^2) && (8)
\end{aligned}
$$

Master Theorem

- Divide and conquer algorithms often give us running-time recurrences of the form

$$T(n) = a\,T(n/b) + f(n) \qquad (9)$$

- 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.

Master Theorem

- 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 $T(n) = a\,T(n/b) + f(n)$
- 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 $a^i f(n/b^i)$

Details

- 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

Recursion Tree

- 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 $a^{\log_b n} = n^{\log_b a}$

$$a^{\log_b n} = n^{\log_b a} \quad (10)$$
$$a^{\log_b n} = a^{\log_a n * \log_b a} \quad (11)$$
$$\log_b n = \log_a n * \log_b a \quad (12)$$

- 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 $a\,f(n/b) \le K f(n)$ for some constant $K < 1$, then $T(n) = \Theta(f(n))$.
- If $a\,f(n/b) \ge K\,f(n)$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$.
- If $a\,f(n/b) = f(n)$, 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 $a\,f(n/b) = 3n/4$ is smaller than $f(n) = n$ by a factor of 4/3, so $T(n) = \Theta(n)$

Example

- **Karatsuba's multiplication algorithm:** $\boldsymbol{T(n) = 3T(n/2) + n}$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 3, b = 2, f(n) = n$
- Here $a\,f(n/b) = 3n/2$ is bigger than $f(n) = n$ by a factor of 3/2, so $T(n) = \Theta(n^{\log_2 3})$

Example

- **Mergesort:** $\boldsymbol{T(n) = 2T(n/2) + n}$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 2, b = 2, f(n) = n$
- Here $a\,f(n/b) = f(n)$, 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 $a\,f(n/b) = n/2 \log n/2$ 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: $\boldsymbol{T(n) = 4T(n/2) + n \lg n}$
- 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?

In-Class Exercise

- Consider the recurrence: $\boldsymbol{T(n) = 2T(n/4) + n \lg n}$
- 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.

Intro to Annihilators

- Suppose we are given a sequence of numbers $A = \langle a_0, a_1, a_2, \cdots \rangle$
- This might be a sequence like the Fibonacci numbers
- I.e. $A = \langle a_0, a_1, a_2, \dots) = (T(1), T(2), T(3), \cdots \rangle$

Annihilator Operators

We define three basic operations we can perform on this sequence:

1. Multiply the sequence by a constant: $cA = \langle ca_0, ca_1, ca_2, \cdots \rangle$
2. Shift the sequence to the left: $\mathbf{L}A = \langle a_1, a_2, a_3, \cdots \rangle$
3. Add two sequences: if $A = \langle a_0, a_1, a_2, \cdots \rangle$ and $B = \langle b_0, b_1, b_2, \cdots \rangle$, then $A + B = \langle a_0 + b_0, a_1 + b_1, a_2 + b_2, \cdots \rangle$

Annihilator Description

- We first express our recurrence as a sequence T
- *We use these three operators to "annihilate" T, i.e. make it all 0's*
- Key rule: can't multiply by the constant 0
- We can then determine the solution to the recurrence from the sequence of operations performed to annihilate T

Example

- Consider the recurrence $T(n) = 2T(n-1)$, $T(0) = 1$
- If we solve for the first few terms of this sequence, we can see they are $\langle 2^0, 2^1, 2^2, 2^3, \cdots \rangle$
- Thus this recurrence becomes the sequence:

$$T = \langle 2^0, 2^1, 2^2, 2^3, \cdots \rangle$$

Example (II)

Let's annihilate $T = \langle 2^0, 2^1, 2^2, 2^3, \cdots \rangle$

- Multiplying by a constant $c = 2$ gets:

$$2T = \langle 2 * 2^0, 2 * 2^1, 2 * 2^2, 2 * 2^3, \cdots \rangle = \langle 2^1, 2^2, 2^3, 2^4, \cdots \rangle$$

- Shifting one place to the left gets $\mathbf{L}T = \langle 2^1, 2^2, 2^3, 2^4, \cdots \rangle$
- Adding the sequence $\mathbf{L}T$ and $-2T$ gives:

$$\mathbf{L}T - 2T = \langle 2^1 - 2^1, 2^2 - 2^2, 2^3 - 2^3, \cdots \rangle = \langle 0, 0, 0, \cdots \rangle$$

- The annihilator of T is thus $\mathbf{L} - 2$

Distributive Property

- The distributive property holds for these three operators
- Thus can rewrite $\mathbf{L}T - 2T$ as $(\mathbf{L} - 2)T$
- The operator $(\mathbf{L} - 2)$ annihilates T (makes it the sequence of all 0's)
- Thus $(\mathbf{L} - 2)$ is called the *annihilator* of T

0, the "Forbidden Annihilator"

- Multiplication by 0 will annihilate *any* sequence
- Thus we disallow multiplication by 0 as an operation
- In particular, we disallow $(c - c) = 0$ for any c as an annihilator
- Must always have at least one **L** operator in any annihilator!

Uniqueness

- An annihilator annihilates exactly *one* type of sequence
- In general, the annihilator $\mathbf{L} - c$ annihilates any sequence of the form $\langle a_0 c^n \rangle$
- If we find the annihilator, we can find the type of sequence, and thus solve the recurrence
- We will need to use the base case for the recurrence to solve for the constant a_0

Example

If we apply operator $(\mathbf{L}-3)$ to sequence T above, it fails to annihilate T

$$\begin{aligned}(\mathbf{L}-3)T &= \mathbf{L}T+(-3)T \\ &= \langle 2^1, 2^2, 2^3, \cdots\rangle + \langle -3\times 2^0, -3\times 2^1, -3\times 2^2, \cdots\rangle \\ &= \langle (2-3)\times 2^0, (2-3)\times 2^1, (2-3)\times 2^2, \cdots\rangle \\ &= (2-3)T = -T\end{aligned}$$

Example (II)

What does $(\mathbf{L}-c)$ do to other sequences $A=\langle a_0 d^n\rangle$ when $d \neq c$?:

$$\begin{aligned}(\mathbf{L}-c)A &= (\mathbf{L}-c)\langle a_0, a_0 d, a_0 d^2, a_0 d^3, \cdots\rangle \\ &= \mathbf{L}\langle a_0, a_0 d, a_0 d^2, a_0 d^3, \cdots\rangle - c\langle a_0, a_0 d, a_0 d^2, a_0 d^3, \cdots\rangle \\ &= \langle a_0 d, a_0 d^2, a_0 d^3, \cdots\rangle - \langle c a_0, c a_0 d, c a_0 d^2, c a_0 d^3, \cdots\rangle \\ &= \langle a_0 d - c a_0, a_0 d^2 - c a_0 d, a_0 d^3 - c a_0 d^2, \cdots\rangle \\ &= \langle (d-c)a_0, (d-c)a_0 d, (d-c)a_0 d^2, \cdots\rangle \\ &= (d-c)\langle a_0, a_0 d, a_0 d^2, \cdots\rangle \\ &= (d-c)A\end{aligned}$$

Uniqueness

- The last example implies that an annihilator annihilates one type of sequence, but does not annihilate other types of sequences
- Thus Annihilators can help us classify sequences, and thereby solve recurrences

Lookup Table

- The annihilator $\mathbf{L}-a$ annihilates any sequence of the form $\langle c_1 a^n\rangle$

Example

First calculate the annihilator:

- Recurrence: $T(n) = 4 * T(n-1)$, $T(0) = 2$
- Sequence: $T = \langle 2, 2*4, 2*4^2, 2*4^3, \cdots \rangle$
- Calulate the annihilator:
  - $\mathbf{L}T = \langle 2*4, 2*4^2, 2*4^3, 2*4^4, \cdots \rangle$
  - $4T = \langle 2*4, 2*4^2, 2*4^3, 2*4^4, \cdots \rangle$
  - Thus $\mathbf{L}T - 4T = \langle 0, 0, 0, \cdots \rangle$
  - And so $\mathbf{L} - 4$ is the annihilator

Example (II)

Now use the annihilator to solve the recurrence

- Look up the annihilator in the "Lookup Table"
- It says: "The annihilator $\mathbf{L} - 4$ annihilates any sequence of the form $\langle c_1 4^n \rangle$"
- Thus $T(n) = c_1 4^n$, but what is c_1?
- We know $T(0) = 2$, so $T(0) = c_1 4^0 = 2$ and so $c_1 = 2$
- Thus $T(n) = 2 * 4^n$

In Class Exercise

Consider the recurrence $T(n) = 3 * T(n-1)$, $T(0) = 3$,

- Q1: Calculate $T(0)$,$T(1)$,$T(2)$ and $T(3)$ and write out the sequence T
- Q2: Calculate $\mathbf{L}T$, and use it to compute the annihilator of T
- Q3: Look up this annihilator in the lookup table to get the general solution of the recurrence for $T(n)$
- Q4: Now use the base case $T(0) = 3$ to solve for the constants in the general solution

Todo

- HW 1