Lecture 4

Recurrence Relations

Tim LaRock
larock.t@northeastern.edu
bit.ly/cs3000syllabus

May 7, 2020

Business

- Homework 1 due Monday night 11:59PM Boston time. Use Piazza to ask questions!
- Office hours are in Piazza
  - Pinned post describing all of them
  - Zoom links under Staff $\rightarrow$ Resources.
  - Please email the person whose office hours you plan to attend beforehand! No need for a lot of detail, just when you are joining and what you want to talk about.

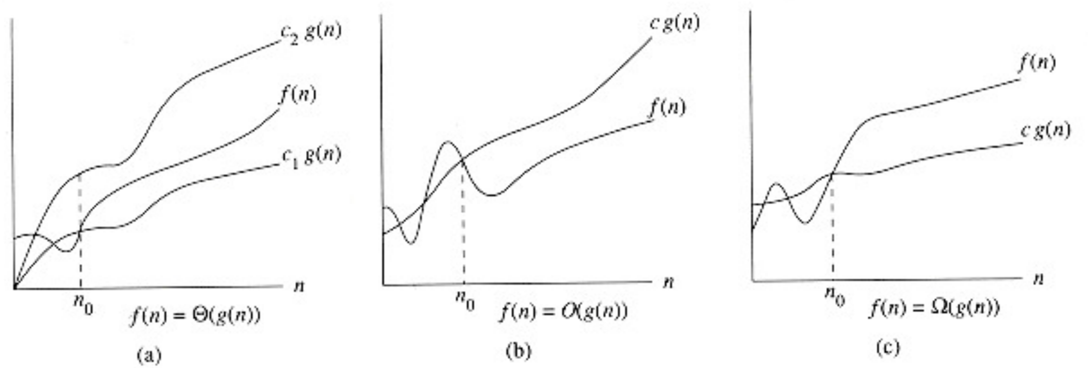
Today

- Clean up asymptotic analysis from last time
- Proving Recurrences
- Recursion Trees

Summary from last time

- Asymptotic analysis is a powerful and flexible framework for reasoning about functional growth
- Capital symbols O and Ω represent *tight* upper and lower bounds
  - Tight bounds "follow" or "scale proportionately with" the bounding function
  - Loose bounds are "left behind" because the bounding function grows more quickly
- We will be almost always interested in the big-O, worst-case runtime of an algorithm

Asymptotic notation, in 3 plots



- $f(n)$ is the function we want to bound
- $g(n)$ is the function we are using to bound $f(n)$
- $c \in \mathbb{R}$ plays slightly different roles for upper/lower:
  - Upper: Push $g(n)$ above $f(n)$
  - Lower: Pull $g(n)$ as close as possible to $f(n)$

CLR Textbook Figure 2.1

Okay, back to recursion...

Recall:

- We studied MergeSort, a procedure for recursively sorting a list of numbers
- We proved the correctness of MergeSort using two inductive proofs
  - First we proved the subroutine Merge was correct, then we showed that if Merge is correct MergeSort must also be correct.
- We wrote down a *recurrence relation* that describes the run time of MergeSort

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

- We claimed that $T(n) = O(n \log n)$. Today we will learn how to show this is true.

Recursion Trees

- Divide and Conquer algorithms often have recurrences of the form
$$T(n) = rT(\frac{n}{c}) + f(n)$$
for some constants r, c and some function $f(n)$

Recursion Trees

- Divide and Conquer algorithms often have recurrences of the form
$$T(n) = rT(\frac{n}{c}) + f(n)$$
for some constants r, c and some function $f(n)$
- Question: What are r, c, and $f(n)$ for MergeSort?

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

Recursion Trees

- Divide and Conquer algorithms often have recurrences of the form
$$T(n) = rT(\frac{n}{c}) + f(n)$$
- We can solve recurrences using *Recursion Trees*

Recursion Trees

- Divide and Conquer algorithms often have recurrences of the form

$$T(n) = rT(\frac{n}{c}) + f(n)$$

$L = \log_c n$

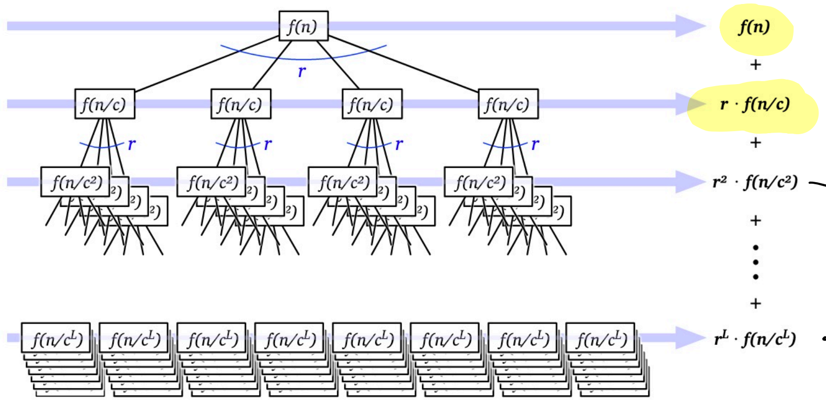
$r = 4$

- We can solve recurrences using *Recursion Trees*

L is the max depth of the tree

$f(n)$

$f(\frac{n}{c^i})$



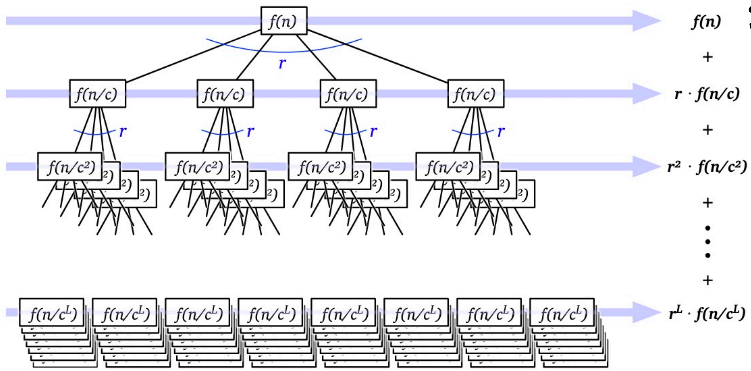
$f(\frac{n}{c})$

$f(\frac{n}{c^2})$

$f(\frac{n}{c^L})$

Figure 1.9. A recursion tree for the recurrence $T(n) = r\,T(n/c) + f(n)$

Recursion Trees



$f(\frac{n}{c^i})$ $f(\frac{n}{c^0})$

$i=1$ $f(\frac{n}{c})$

$i=2$ $f(\frac{n}{c^2})$

Figure 1.9. A recursion tree for the recurrence $T(n) = r\,T(n/c) + f(n)$

- We can solve a recurrence by summing the values in the nodes

Mergesort

$$T(n) = \sum_{i=0}^{L} r^i f(\frac{n}{\cancel{b^i}\ c^i}) = \sum_{i=0}^{\log_2 n} 2^i \frac{n}{2^i} = \sum_{i=0}^{\log_2 n} n$$

where $L = \log_c n$ represents the depth of the tree

Recursion Tree for MergeSort

Recall our simplified MergeSort recurrence

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

- What will be the value of the root node in the recursion tree?
- How many nodes will be at the 2nd level?

Recursion Tree for MergeSort

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

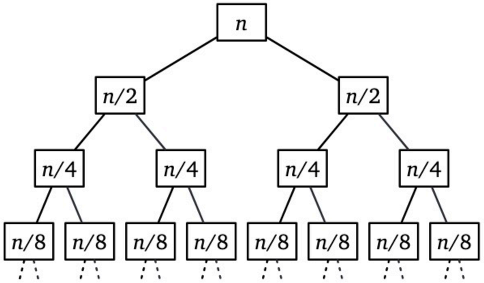


Figure 1.10. The recursion tree for mergesort

Recursion Tree for MergeSort

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

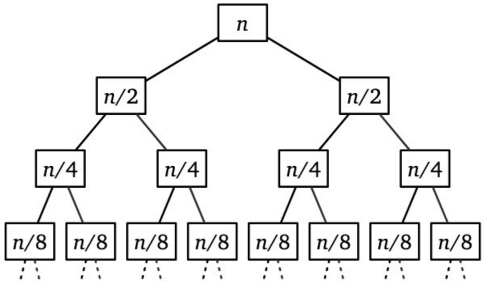


Figure 1.10. The recursion tree for mergesort

Question: Can you see a straightforward way to get the sum of the nodes in this tree? Hint: Think level-by-level

Recursion Tree for MergeSort

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

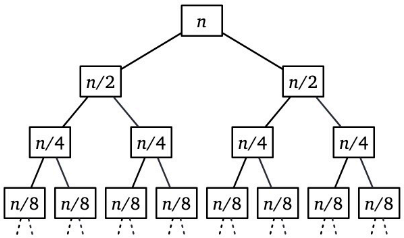


Figure 1.10. The recursion tree for mergesort

There are 2^i nodes at level i, each with value $\frac{n}{2^i}$, so every level sums to n. So we get

$$T(n) = \sum_{i=0}^{L} n$$

Recursion Tree for MergeSort

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \longrightarrow O(n \log_2 n)$$

$$T(n) = \sum_{i=0}^{L} n$$

$L = \log_2 n$

$L = \log_c n$

$= \log_2 n$

$T(n) = O(n \log_2 n)$

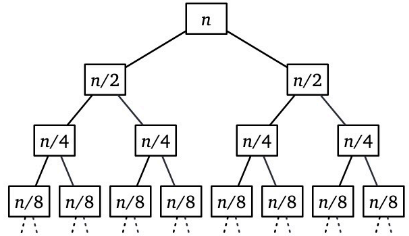


Figure 1.10. The recursion tree for mergesort

I argue we are done now - why?

Recursion Tree for MergeSort

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

$rT(\frac{n}{c})$

$$T(n) = \sum_{i=0}^{L} n = O(nL) = O(n \log n)$$

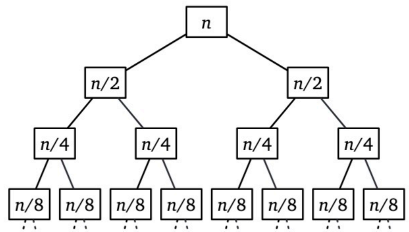


Figure 1.10. The recursion tree for mergesort

2^i

So we find confirm that MergeSort runs in $O(n \log n)$

Wrap up

- Please work on the homework and ask questions on Piazza!
- No suggested reading for next class at this point. I may announce some tomorrow.

Final Thoughts