CS 305
Design and Analysis of Algorithms

09 / 20 / 2021

Instructor: Michael Eckmann

Today's Topics

- Questions / Comments?
- Reading:
  - 93-96 (Master Theorem, to be started today)
- Logarithms
- Master Method of solving recurrences
  - Prove case 1

Logarithms

- Let's show some properties of logarithms on the board.

Logarithms

- Let's go back and look at the a-ary tree and recall there are a^($\log_b n$) --- that is a to the power $\log_b n$
- Any other way to specify a^($\log_b n$) ?

Logarithms

- Let's go back and look at the a-ary tree and recall there are a^($\log_b$n) --- that is a to the power $\log_b$n
- Any other way to specify a^($\log_b$n) ?
- n^($\log_b$a)
- better because it is n to some constant power

Master Method for Recurrences

- The Master Method can be used to solve recurrences of the form: $T(n) = a * T(n/b) + f(n)$
- There are 3 cases to consider each based on a relationship of time spent at leaves vs. at the root.
- As an example, and to help us shortly, let's now count the number of nodes in a perfect tree, say a 3-ary tree.

Master Method for Recurrences

- Recurrences of the form: $T(n) = a * T(n/b) + f(n)$

- Case 1: if time spent at root is "at most little bit smaller" than time spent at all the leaves, then the running time of algorithm is dominated by the time spent at the leaves
- Case 2: if time spent at root is big theta (same as) of time spent at all the leaves then it is lgn * time spent at leaves
- Case 3: if time spent at root is "at least little bit larger" than time spent at all the leaves, then the running time of algorithm is dominated by the time spent at the root

Master Method for Recurrences

- Let's write T(n) from that a-ary recursion tree as time spent at leaves plus time spent at internal nodes
- Let's prove case 1 of master theorem which will tell us the running time when time spent at root is "at most little bit smaller" than time spent at all the leaves