Top-Down Problem Solving

Remington Greko, Tyler Gutowski, and Spencer Hirsch

February 20, 2023

Work with your team to write a report showing your knowledge of the Recursion. Submit the team's report on Canvas. Include a task matrix indicat- ing who did what.

Recursion

There is a cute definition of recursion in the Hacker's Dictionary: Recursion: See Recursion

There is a good description of recursion on Wikipedia, read it. Top-down problem solving requires solving a recurrence relation. There are a similarities between recurrence equations and ordinary differential equations should you desire to explore.

After successful completion of this quest you will understand how to model the time complexity of a recursive algorithm by a recurrence of the form

$$T(n) = aT(n/b) + f(n)$$

together with some initial conditions to get things started. Interpret the recurrence above as saying: To solve a problem with input size n, solve a problem of size n/b (you may need to do this a times) and apply a forcing function f(n) at each step.

There are many ways to solve a recurrence:

1. Guess or Given and Prove
2. Unrolling also called substitution
3. The Master Theorem
4. Generating Functions

Guess or Given and Prove

I like this approach to my third grade teacher Mrs. Beavis asking me to prove x = 2 is a solution to the polynomial equation

1. Show that log(n) (the log base 2 of n) solves the recurrence:

$$T(n) = T(n/2) + 1, T(0) = 1$$

   Mention an algorithm that is described by this recurrence. (Note: you may assume n is a power of 2 so that dividing by 2 never introduces a fraction)

   Solution:

   Assume n = 2^k

$$T(2^k) = T(2^k/2) + 1$$

$$T(2^k) = T(2^{k-1}) + 1$$

$$T(2^k) = (T(2^{k-2}) + 1) + 1$$

$$T(2^k) = T(2^{k-2}) + 2$$

$$T(2^k) = [T(2^{k-3}) + 1] + 2$$

$$T(2^k) = T(2^{k-3}) + 3$$

$$T(2^k) = T(2^{k-k}) + k$$

$$T(2^k) = T(2^0) + k$$

$$T(2^k) = T(1) + k$$

$$T(2^k) = 1 + k$$

$$T(2^k) = log(n) + 1$$

   Therefore,

$$T(n) = log(n)$$

   Binary search is a recursive alogorithm that has a time complexity of (log(n)).

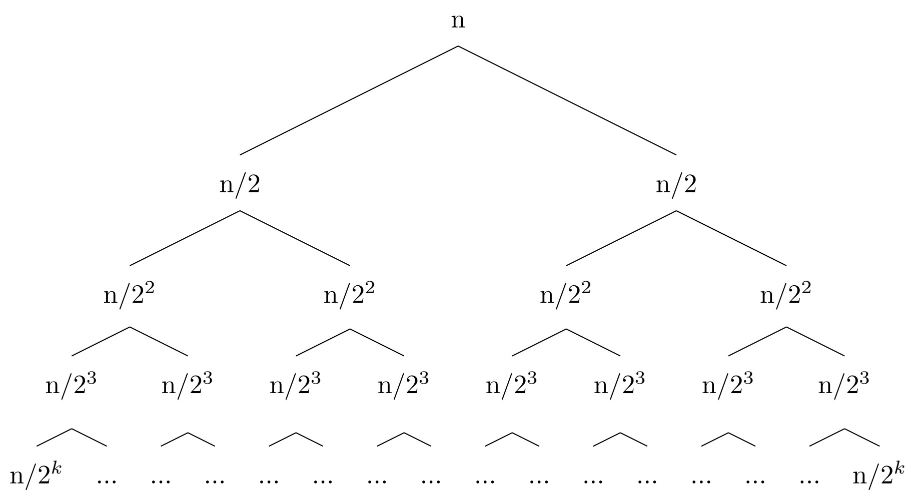
2. Show that nlog(n) solves the recurrence:

$$T(n) = 2T(n/2) + n, T(0) = 1$$

Mention a algorithm that is desribed by this recurrence.

Solution:

I started by using a recursion tree, because it is easy to visualize.



Sum of each row is n. There are k columns.

$$T(n) = nk$$

Assume

$$n/2^k = 1$$

$$n = 2^k$$

$$k = log(n)$$

$$nk = nlog(n)$$

Additionally,

$$T(n) = 2T(n/2) + n$$

$$T(n/2) = 2T(n/2^2) + n/2$$

$$T(n) = 2[T(n/2) + n]$$

$$T(n) = 2[2T(n/2^2) + n/2] + n$$

$$T(n) = 2^k T(n/2^k) + kn$$

$$T(n/2^k) = T(1)$$

$$n/2^k = 1$$

$$n = 2^k$$

$$k = log(n)$$

$$kn = nlog(n)$$

Therefore,

$$T(n) = nlog(n)$$

Mergesort is a recursive algorithm that has a time complexity of O(nlog(n)).

Name	**Section**
Remington Greko	
Tyler Gutowski	O(nlog(n)) Recurrence
Spencer Hirsch	O(log(n)) Recurrence and Example Algorithms