CS 315 - Day 11, Recurrence Trees

Let's have a quick review of Asymptotics, and consider them in a practical frame:

Q: When is our first exam?

A: At least one day away.

This is a true statement, but not really useful.

$\frac{1}{3}n^2$ is $O(n^3)$ is a true statement, because $0 \leq \frac{1}{3}n^2 \leq cn^3$, but this statement is not as useful as $\frac{1}{3}n^2$ is $O(n^2)$ or even $\theta(n^2)$.

By the way, because O is an "at least" kind of relationship, $n^3 \neq O(n^2)$, because $0 \leq n^3 \not\leq n^2$.

Continuing, directly from the definitions we get: $n = O(n^2)$ and $n^3 = \Omega(n^2)$

Since $n^2 = O(n^2)$ and $n^2 = \Omega(n^2)$, then $n^2 = \Theta(n^2)$ is a ***tight*** and the most correct definition.

Just remember, outside this class, it's almost universal that when you see $O(\,)$ it's meant as the "best" $O(\,)$, which is just our $\Theta(\,)$.

We can recursively decompose a large problem into one or more smaller problems, using solutions to the (easier) subproblems to solve the original problem.

We can describe this decomposition into smaller problems using a recurrence relation. The recurrence relation describes precisely the performance of the algorithm, and we solve (actually usually guess) the recurrence relation to find the running time.

First, let's review both direct and explicit definitions of sequences.

Consider $1, 2, 4, 8, \cdots$

The ***direct***/***explicit*** definition is based on the power of 2, and ***don't forget to quantify***.

$$a_n = 2^n, \forall\, n \geq 0$$

A ***recursive*** definition has the next term depending on a previous term, and ***don't forget to quantify***.

$a_1 = 1$

$a_n = 2a_{n-1}, \forall\, n \geq 2$

A recurrence relation is reminiscent of a recursive sequence definition, and we will practice taking a given a recurrence relation and:

1. Setting up a recursion tree.
2. ***Guessing*** a solution based on the recursion tree.
3. And then ***just sometimes*** proving the solution is correct (via induction).

There is a HW problem in the text where they ask to prove that the solution to the recurrence:

$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$ was $T(n) = O(n \lg n)$.

But how do we find that solution? We will do the proof in class, but first we will just focus on a process to ***guess a solution*** like this.

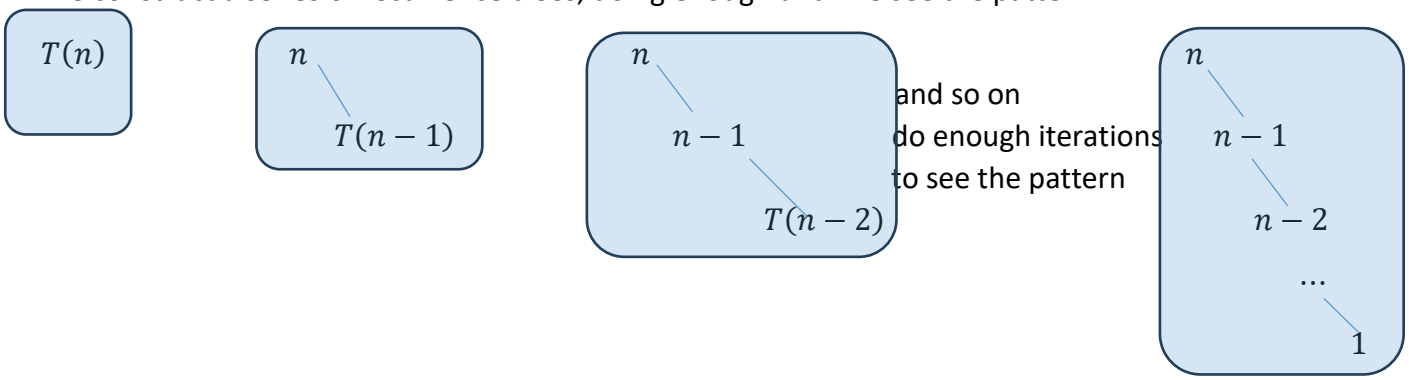
And we won't tell the whole story today, it gets very involved. We will disregard boundary conditions (assume n is an integer and/or a power of 2 if needed), any floors $\lfloor\ \rfloor$, and ceilings $\lceil\ \rceil$.

Again, our goal is to understand how to set up a recurrence tree and use it to ***guess*** a solution.

Consider a recursive program which eliminates one item at a time (think of a recursive search of an unbalanced binary search tree):

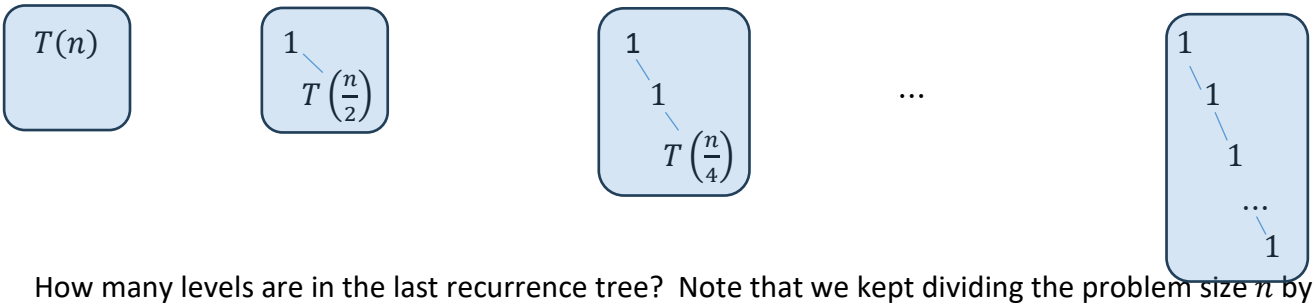
$$T(n) = \begin{cases} 1 & , \quad n = 1 \\ T(n-1) + n & , \quad n \geq 2 \end{cases}$$

We construct a series of recurrence trees, doing enough until we see the pattern.



There are n levels, and total work of: $1 + 2 + \cdots (n-2) + (n-1) + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \theta(n^2)$.

Next, guess a solution for the recurrence representing binary search, $T(n) = T\left(\frac{n}{2}\right) + \theta(1)$



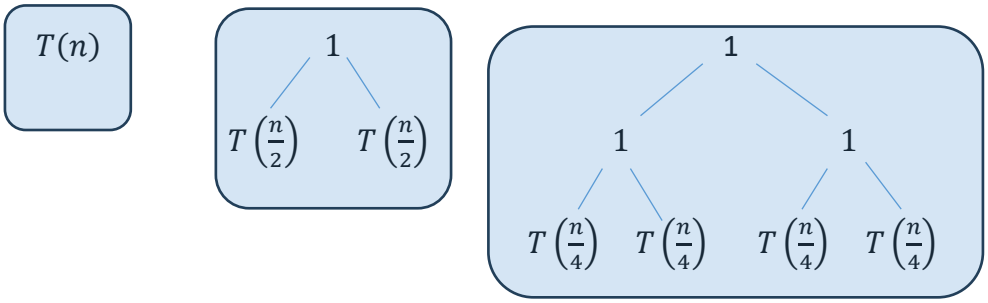
How many levels are in the last recurrence tree? Note that we kept dividing the problem size n by 2. So we can solve for how many of those divides we have until we only have a problem size of 1. With h = height:

$$\frac{n}{2^h} = 1$$
$$n = 2^h$$
$$\lg n = \lg 2^h$$
$$\lg n = h$$

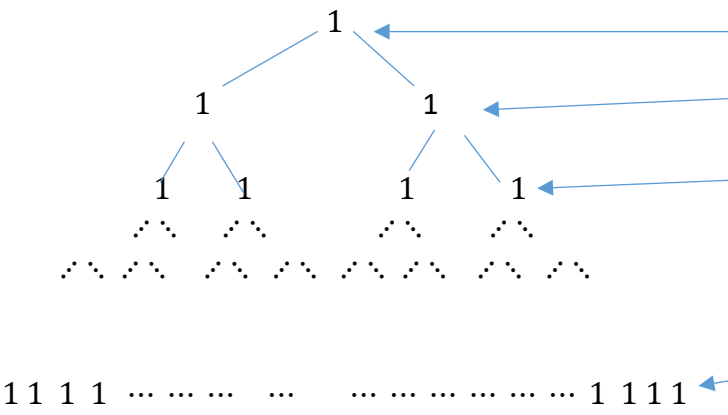
A constant amount of work on each of $\lg n$ levels gives the solution to the recurrence $O(\lg n)$, which is the amount of work for the binary search algorithm.

Next, we will guess a solution to the recurrence: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \theta(1)$

Note that as we are constructing the recurrence trees, there are 2 subproblems, so there will be 2 child branches in the tree.



Which leads to:



Amt of work on each level:

1

$1 + 1 = 2$

$1 + 1 + 1 + 1 = 4$

⋮

These totals look like a Geometric Series…

$1 + 1 + 1 + \cdots + 1 = ?$

So in the last row, there are n 1's, and

$1 + 1 + 1 + \cdots + 1 = n$

Since $n = 2^h$ (which gives $\lg(n) = \lg(2^h) \rightarrow \lg(n) = h$), we have a Geometric Series for the total amount of work:

$$1 + 2 + 4 + \cdots + 2^h$$

$$= \sum_{i=0}^{h} 2^i$$
$$= \frac{2^{h+1} - 1}{2 - 1}$$
$$= 2^{h+1} - 1$$

We can substitute in $\lg(n) = h$, $2^{h+1} - 1 = 2^{\lg n + 1} - 1 = 2^{\lg n} \cdot 2^1 - 1 = 2^1 \cdot 2^{\lg n} - 1 = 2n - 1$

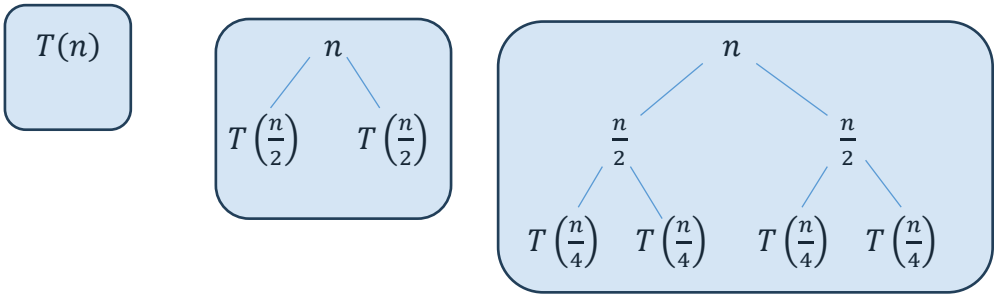
Which is just $\theta(n)$!

It's time to practice, and let's do it in teams:

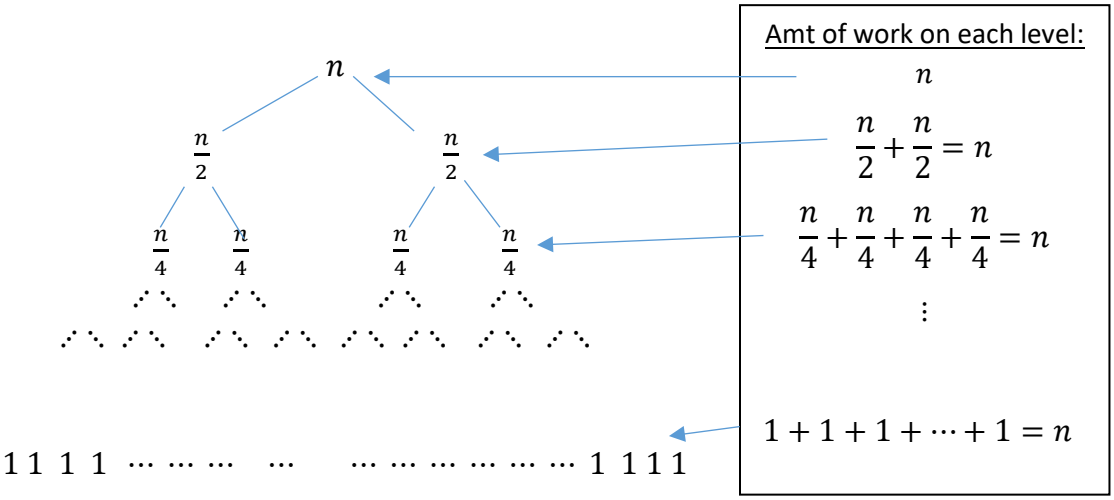
<u>In-class w/***assigned groups of four (4)***, write 0. and 1. on board as ground rule(s):</u>

0. No devices.
1. Intro: Name, POE, Icebreaker.
2. Pick someone on the group who will report out.
3. First, work together to ***guess*** a solution to the recurrence: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$

Now we will guess a solution to the recurrence: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$



Which leads to:



How many levels? Where h = height: $n = 2^h$

$$\lg(n) = \lg(2^h)$$
$$\lg(n) = h$$

So, there is n amount of asymptotic work on each level, and there are $\lg(n)$ levels, so we ***guess*** that the solution to

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$ is $T(n) = n \cdot \lg(n)$

We did the first two steps in those several examples, (1) setting up a recurrence tree, and (2) ***guessing*** a solution. For the most part that will be enough.

But for this most recent example, let's take the third step and ***prove*** that our guess is actually correct.

Use the Principle of Mathematical Induction to prove that when *n* is an exact power of 2, the solution to the recurrence relation (just start with the components of an inductive proof and fill in):

$$T(n) = \begin{cases} 2 & if\ n = 2, \\ 2 \cdot T\left(\frac{n}{2}\right) + n & if\ n = 2^k, for\ k > 1 \end{cases}$$

is $T(n) = n \cdot \lg(n)$

(***hint:*** substitute in $n = 2^k$ before proving)

We first substitute in $n = 2^k$

$$T(2^k) = \begin{cases} 2 & if\ k = 1, \\ 2 \cdot T\left(\frac{2^k}{2}\right) + 2^k & for\ k > 1 \end{cases}$$

And then show $T(2^k) = 2^k \cdot \lg(2^k) = 2^k \cdot k = k \cdot 2^k$

Proof:

Basis Step [Show P(1) is true, that is for *k=1*, $\boldsymbol{T(2^1) = T(2)}$ is true]

$T(2^1) = 2$, ***as given by the recurrence relation,***
and $T(2^1) = 1 \cdot 2^1 = 2$***, by plugging into the formula to prove***

Inductive Step [Assume ***P(k)*** is true , and use it to show ***P(k+1)***]

Assume $T(2^k) = k \cdot 2^k$,

and we will show $T(2^{k+1}) = (k+1) \cdot 2^{k+1}$.

Using the given recurrence, the LHS of *P(k+1)* is:

$T(2^{k+1}) = 2 \cdot T\left(\frac{2^{k+1}}{2}\right) + 2^{k+1} = 2 \cdot T(2^k) + 2^{k+1}$

By substitution from the assumption, $2 \cdot T(2^k) + 2^{k+1} = 2 \cdot k \cdot 2^k + 2^{k+1} = k \cdot 2^{k+1} + 2^{k+1}$

Factoring out a *(k+1)* gives $(k+1) \cdot 2^{k+1}$, which is the RHS of *P(k+1)*.

Q.E.D.