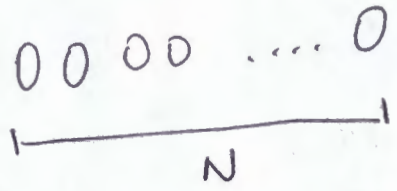
Recurrence lecture.

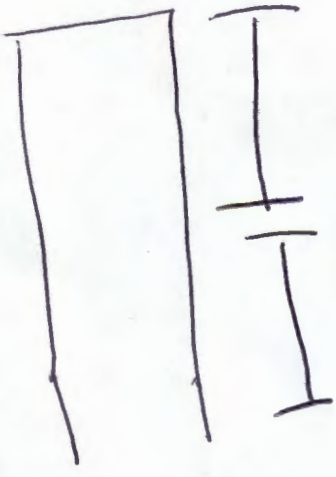
Divide and conquer Idea.

The ~~glass~~ solid ^ golden egg drop. You have n solid ^ golden egg and n story building.
You are tasked with finding the highest story from which you can drop the egg without it breaking. (As payment you get to keep all ~~the~~ eggs ~~that you don't~~ ^ ~~break~~. that you don't drop)

7.3.

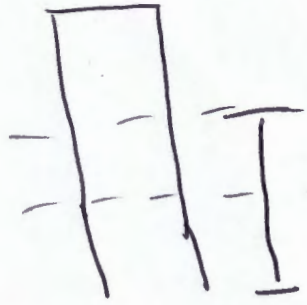


Tallest build
Burj Khalifa
160



floor is higher.
If it does not break

go half way between mid, to floor. If dropped



if does not break
egg is in this range.

* Dived make you problem into ~~some~~ smaller problems

* conquer solve small ~~s~~ sub problem

Consider another example of guessing a number between 1 & 100.

you can ask if it is greater than k.

~~first~~

Let's write a formula for the time it take to finish the egg drop task.

Time it search the second half.

$$T(n) = \begin{cases} T(n/2) + 1 & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$

Time to do n drops

one unit of time/step.

To be more flexible and account for time travelling up and down the stairs. let move

n is an integer going to be rounded

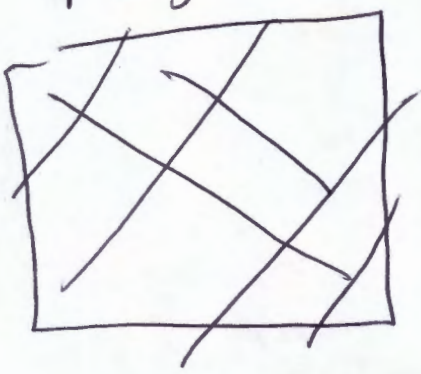
$$T(n) \leq \begin{cases} T(\lceil n/2 \rceil) + c_1 & \text{if } n \geq 2 \\ c_2 & \end{cases}$$

c_1 & c_2 are constants

longest amount of time it takes to walk up to middle floor.

Lets consider a divide an conquere algorithm that we have seen before.

Merge sort code



List size of 1

$T(n/2)$

$T(n/2)$

$1 \times n = n$

~~Merge Sort (A, lo, h~~

Total time for merge sort.

$$T_n = \begin{cases} T(n/2) + T(n/2) + n & \text{if } n \geq 1 \\ 1 & \text{if } n = 1 \end{cases}$$

~~clear~~ cleaner version of this

$$T_n = \begin{cases} 2\,T(n/2) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

① The previous equation is ~~called a~~ ~~reeg~~ commonly referred to as a recurrence relation.

② One way of solving these recurrences is using a recurrence tree.

Consider the following recurrence (ignoring the base case)

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

We can interpret this as solving 2 problem of size $n/2$ and doing n units of additional work.

Simarly we can interpret.

$$T(n) = T(n/4) + n^2$$

Solve one problem of size

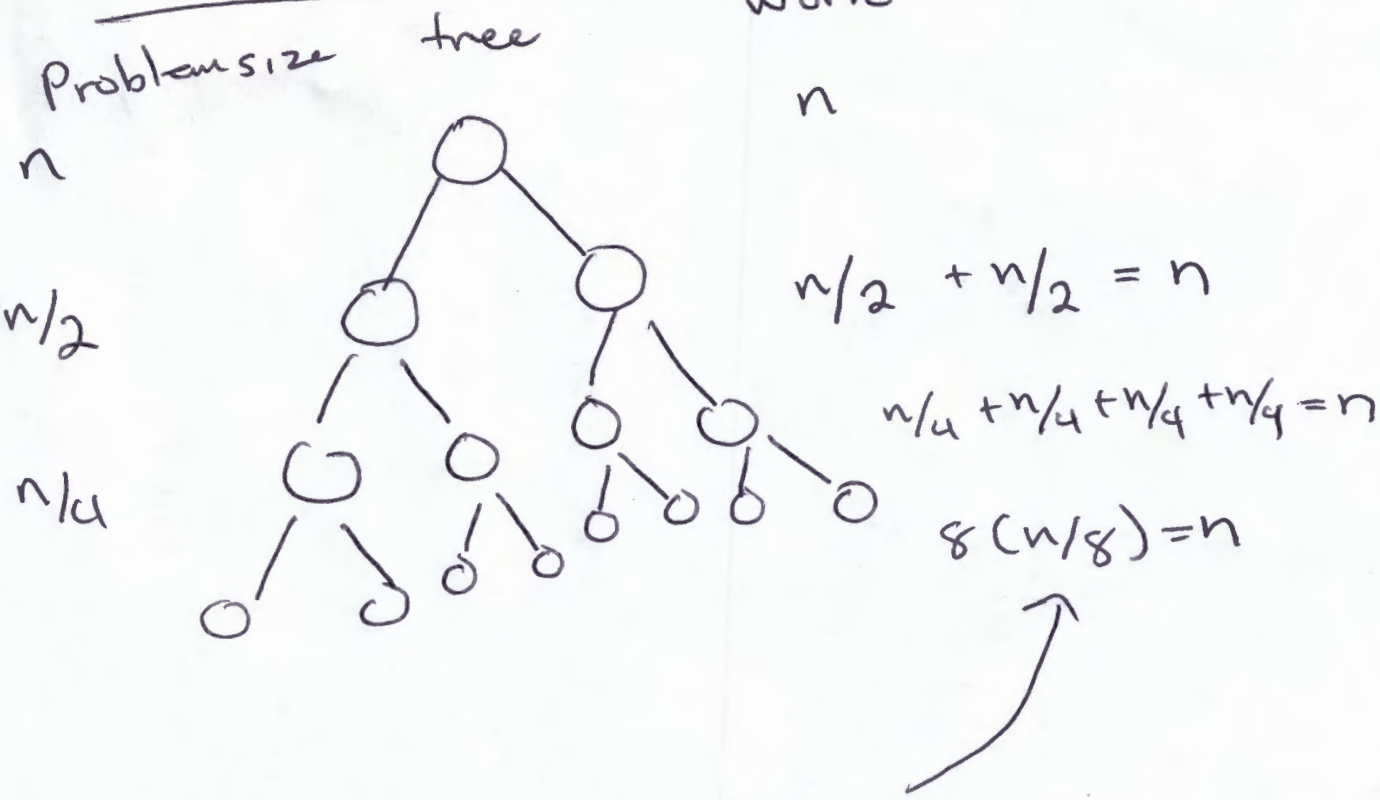
Do n^2 additional units of work.

Similarly we can interpret this recurrence

$$T(n) = 3T(n-1) + n$$

Solve 3 problem of size $n-1$ and do n units of additional work.

let draw four levels of a recursion tree



New page

$T(n) = 2T(n/2) + n \quad \cdots (1)$

Substitute.

for $n/2$

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

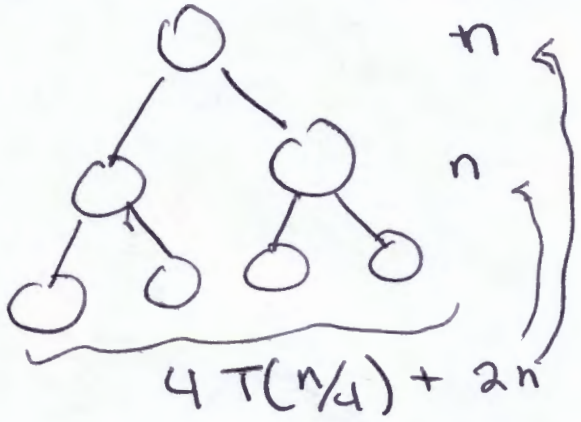
$T(n/2) = 2T(n/4) + n/2 \quad \cdots (2)$

Substitute equat 2 into 1

$T(n) = 2(2T(n/4) + n/2) + n$

$4T(n/4) + n + n$

$2n$



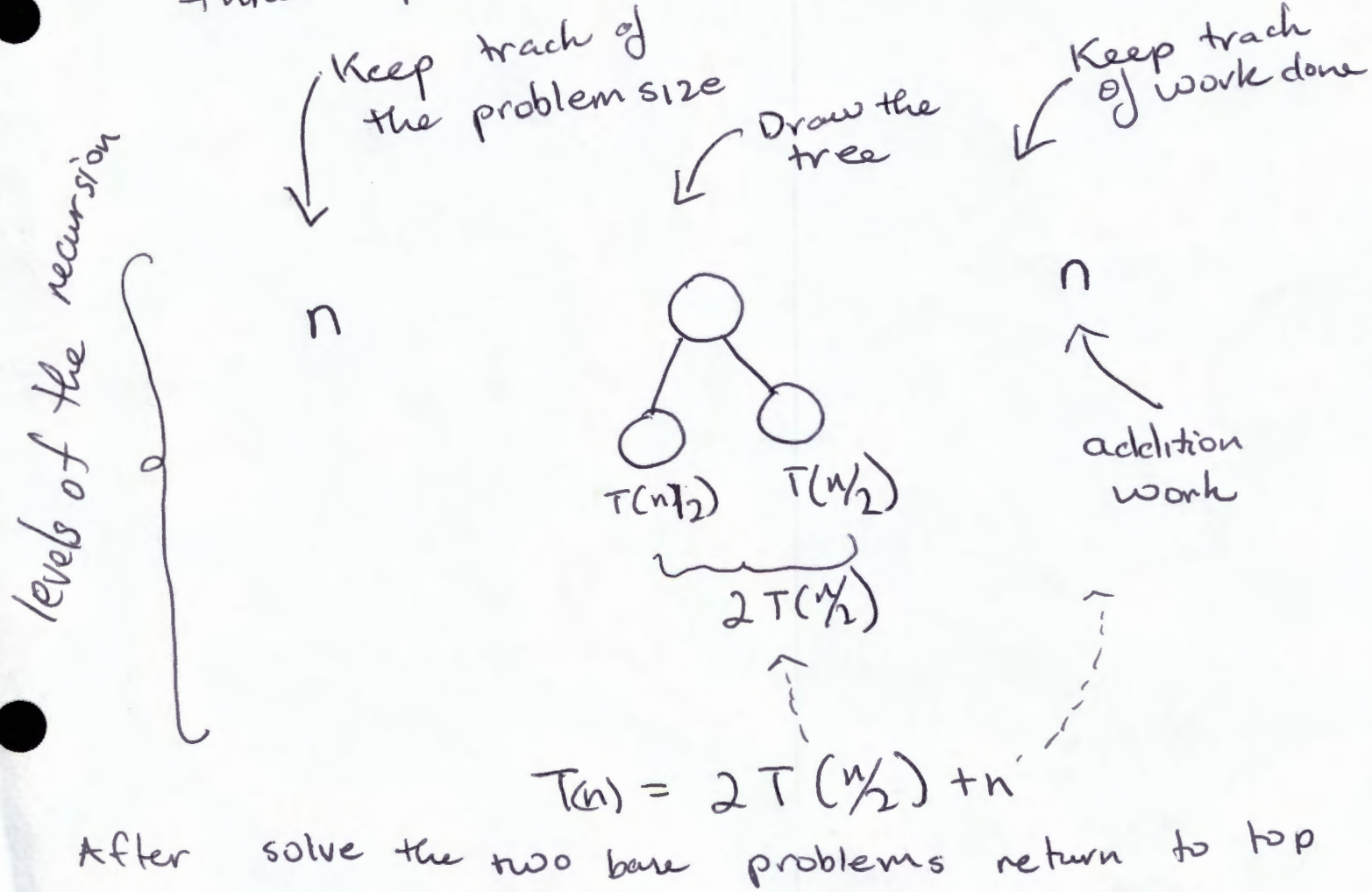
One way of solving these recurrences is using a recurrence / recursion trees.

Let's assume n is a power of 2

this recurrence

$$T(n) = 2T(n/2) + n \quad *$$

We can represent this with a recursion tree
The recursion tree diagram has three parts



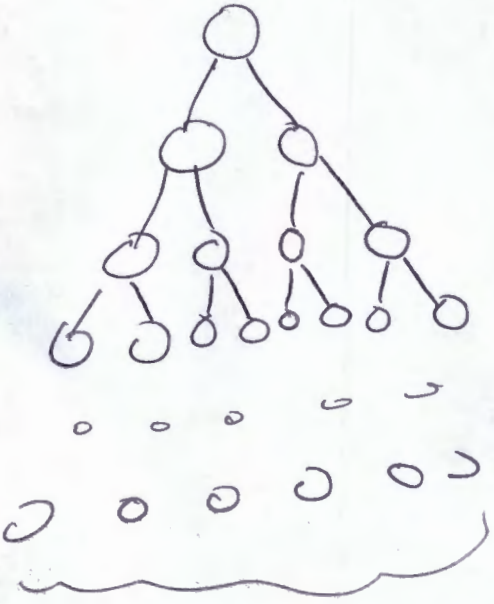
After solve the two base problems return to top

~~Now we can do~~

We have one additional problem that we need to solve.....

How deep will the tree get.

	problem size	work
0	$n/2^0$	n
1	$n/2^1$	$n/2+n/2=n$
2	$n/2^2$	$n/4+n/4+n/4+n/4=n$
3	$n/2^3$	$8(n/8)$
$n/2^i=1$	1	$n \times 1 = n$



this is the base case

$2^i[n/(2^i)] = n$

↑ work at each level

$n/2^i = 1$

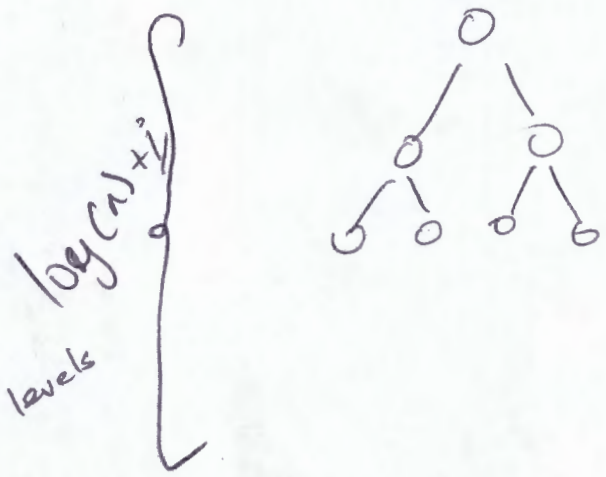
$n = 2^i$

$\log_2(n) = i$

But remember i started for 0

{0 1 2} size of 3

total level $i+1 = \log_2(n)+1$



n amount of work at each level

Solution

$$T(n) = n(\log_2(n)+1)$$
$$= n\log_2(n)+n$$
$$T(n) \approx n\log_2(n)$$
$$T(n) = O(n\log_2(n))$$

There for the runtime of merge sort is bounded above by $O(n\log_2(n))$

More general if we look at the following relation

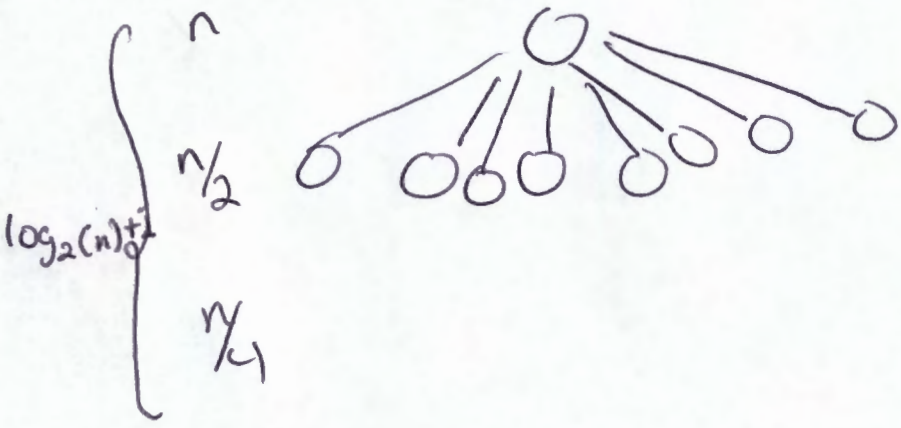
$$T(n) = n(\log_2(n)+a)$$
$$= an\log_2(n) + an$$

↑ account for variation of each node

Stop & think

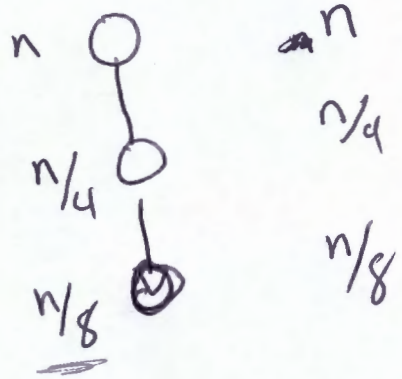
Qu12 Draw the recursion trees for.

$8T(n/2) + n^3$



Qu12

$T(n/4) + n$



terminates when

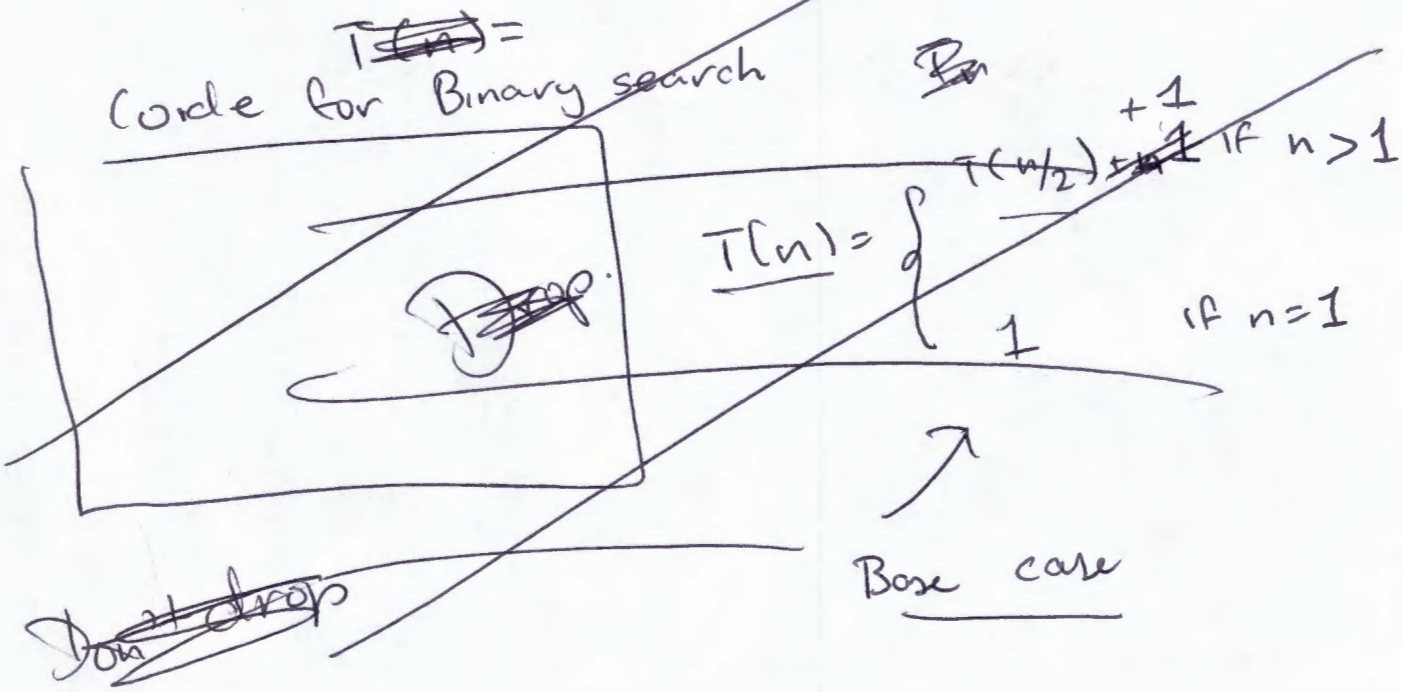
$n/4^i = 1$

$n = 4^i$

$\log_4(n) = i$

~~2 lets consicler 1 more recure~~

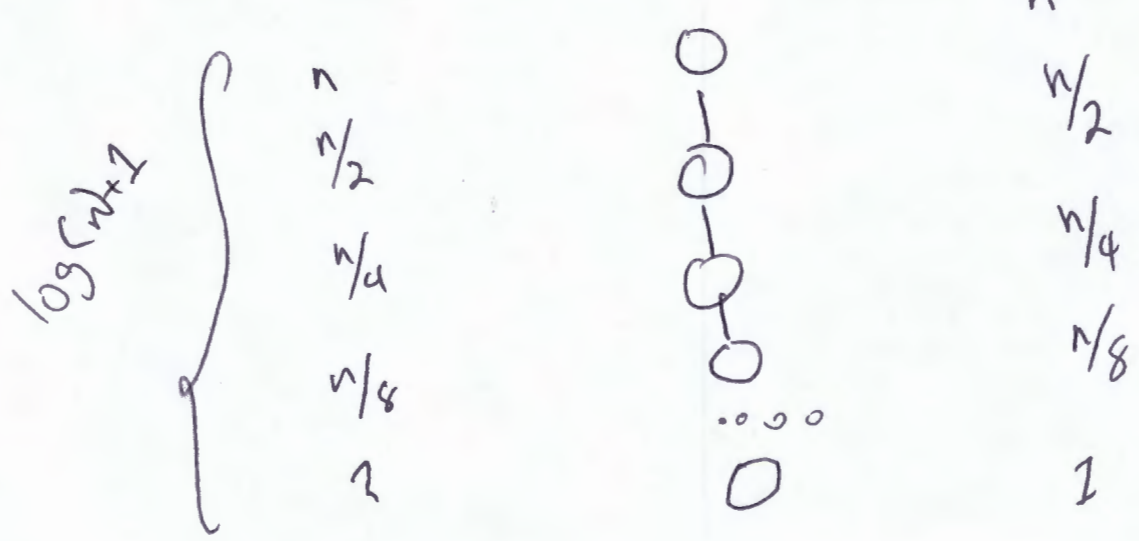
Quiz code to recurence.



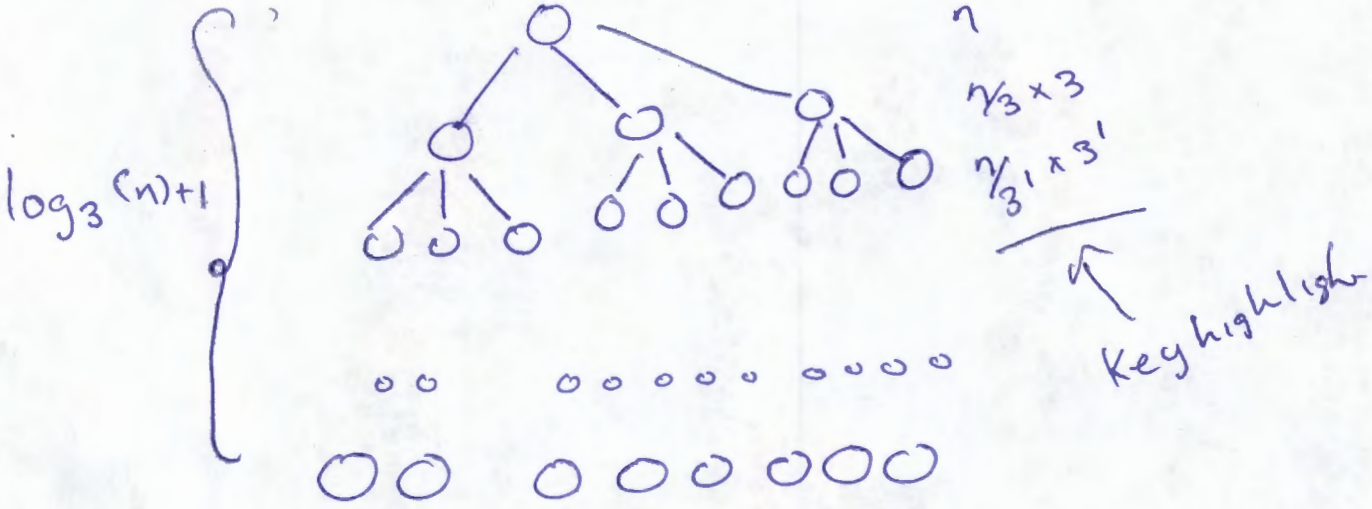
Lets ~~cond~~ consicler anothe recmence

$$T(n) = \begin{cases} T(n/2) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

n/2 n/2



$$T(n) = \begin{cases} 3T(n/3) + n & \text{if } n \geq 3 \\ 1 & \text{if } n < 3 \end{cases}$$

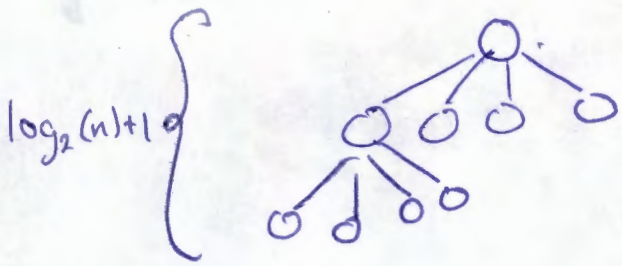


$\log_b(n) = \Theta(\log_2(n))$ $b > 1$

New page

$$T(n) = \begin{cases} 4T(n/2) + n & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$

$$T(n) = \begin{cases} 4T(n/2) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$



$4^{(i)}$ $n/2^{(i)}$ ← amount of work

Number of nodes

Page 1

total number of

Total amount $4^i(n/2^i) = 2^i n$

$$\sum_{i=0}^{\log_2(n)} 2^i n = n \sum_{i=0}^{\log_2 n} 2^i$$

$$T(n) = n \sum_{i=0}^{\log_2 n} 2^i$$

$$= n\left(\frac{1-2^{(\log_2 n)+1}}{1-2}\right)$$

$$= n\left(\frac{1-2n}{-1}\right)$$

$$= 2n^2 - n$$

$$= \cancel{\Theta}\, O(n^2)$$