

Lecture 3

Recurrence Relations and how to solve them!

Announcements!

- **HW1** is due Wednesday at 1pm!
- OAE letters and exam conflicts:
 - cs161-spr2223-staff@lists.stanford.edu
 - Contact us about alternate exam requests by **WEDNESDAY 4/12 AT THE LATEST.**
 - **OTHERWISE WE MAY NOT BE ABLE TO ACCOMMODATE YOUR REQUEST!!!!**
- We will *not* be doing an educational study in this course.
 - The feedback was overwhelmingly against it.
 - The rest of the HWs will be just like HW1: small groups of your choosing (or individually if you prefer).

More announcements!

- We will be doing **high-resolution feedback!**
 - Each week, a random sample of you will receive a request for feedback – it shouldn't take more than a few minutes.
 - Responses are anonymous.
 - We will use this feedback to improve the course!
 - **Please keep an eye out for these emails!**
- Note that you can also give us anonymous feedback at any time via the form on the website; or non-anonymous feedback at any time on Ed.

Yet More Announcements!

- We will have two **extra review sessions!** 😊
- Induction and Logarithms Properties with Saumya!
 - Thursday, 8:30pm
- Big-O and Summations with Ricky!
 - Friday, 12:30pm
- Both will be recorded!
- Stay tuned for an Ed post about it (or check it out if it's already up).

Ed Heroes!

As of Sunday

Top askers

Name	Questions
Matthew B H L	13
Ellie T	7
Thanawan A	7
Monica D H	5
Daniel G	4
Claire M. M	4

Nahome G
H

Yasmine A

Kevin G

Katie L

Top hearted

Name	Hearts
Rishu Garg STAFF	60
Shubham Anand Jain STAFF	41
Bharath Namboothiry STAFF	30
Monica Di H	28
Mary Wootters STAFF	20
Yasmine A	16
Zach Wi	15
Anna M.	13
Stephan Sharkov STAFF	13
Riya K	12

Top endorsed

Name	Endorsements
Zach W	4
Monica Diane H	3
Anna M	2
Parthav S	1
Nahome G H	1
Victoria H	1
Benson Z	1
Yujung "Jenny" K	1
Deniz K	1

Last time....

- Sorting: InsertionSort and MergeSort
- What does it mean to work and be fast?
 - Worst-Case Analysis
 - Big-Oh Notation
- Analyzing correctness of iterative + recursive algs
 - Induction!
- Analyzing running time of recursive algorithms
 - By writing out a tree and adding up all the work done.

Today

- Recurrence Relations!



- How do we calculate the runtime a recursive algorithm?

- The Master Method

- A useful theorem so we don't have to answer this question from scratch each time.

- The Substitution Method

- A different way to solve recurrence relations, more general than the Master Method.

Running time of MergeSort

- Let $T(n)$ be the running time of MergeSort on a length n array.
- We know that $T(n) = O(n \log(n))$.
- We also know that $T(n)$ satisfies:

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

```
MERGESORT(A):  
  n = length(A)  
  if n ≤ 1:  
    return A  
  L = MERGESORT(A[:n/2])  
  R = MERGESORT(A[n/2:])  
  return MERGE(L,R)
```

Running time of MergeSort

- Let $T(n)$ be the running time of MergeSort on a length n array.
- We know that $T(n) = O(n \log(n))$.
- We also know that $T(n)$ satisfies:

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$$


Last time we showed that the time to run MERGE on a problem of size n is $O(n)$. For concreteness, let's say that it's at most $11n$ operations.

```
MERGESORT(A):  
  n = length(A)  
  if n ≤ 1:  
    return A  
  L = MERGESORT(A[:n/2])  
  R = MERGESORT(A[n/2:])  
  return MERGE(L,R)
```

Recurrence Relations

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ is a **recurrence relation**.
- It gives us a formula for $T(n)$ in terms of $T(\text{less than } n)$
- The challenge:

Given a recurrence relation for $T(n)$, find a closed form expression for $T(n)$.

- For example, $T(n) = O(n \log(n))$

Note that
 $T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$
(with a \leq) is also a recurrence relation! Does it matter for a conclusion like $T(n) = O(n \log(n))$?



Technicalities I

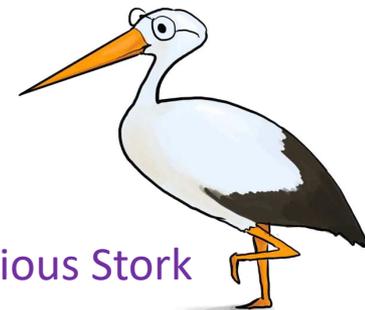
Base Cases



Plucky the
Pedantic Penguin

- Formally, we should always have **base cases** with recurrence relations.
- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ with $T(1) = 1$
is not the same function as
- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ with $T(1) = 1000000000$
- However, no matter what T is, **$T(1)$ is $O(1)$** , so sometimes we'll just omit it.

Why is $T(1) = O(1)$?



Siggi the Studious Stork

On your pre-lecture exercise

- You played around with these examples (when n is a power of 2):

$$1. \quad T_1(n) = T_1\left(\frac{n}{2}\right) + n, \quad T(1) = 1$$

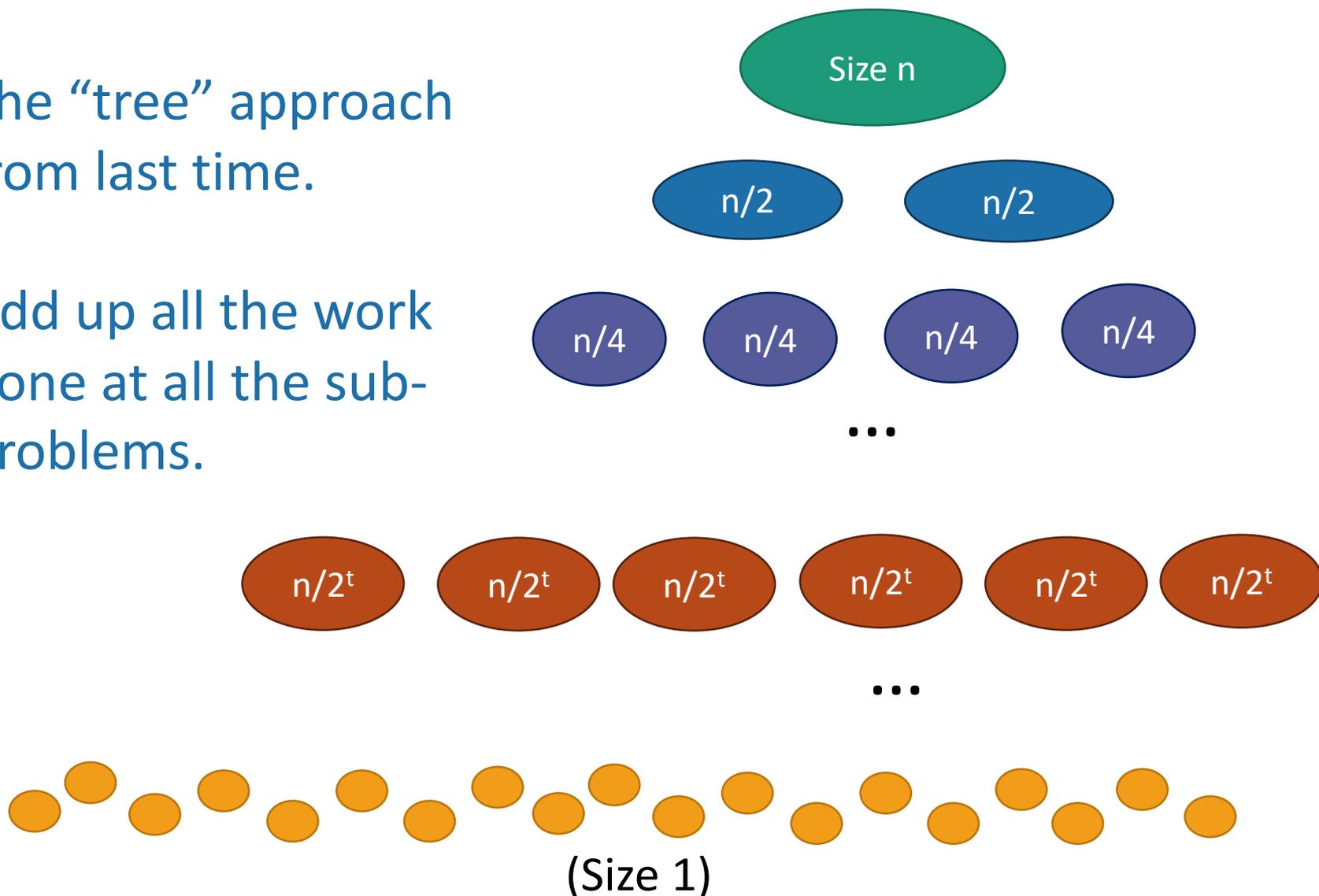
$$2. \quad T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \quad T(1) = 1$$

$$3. \quad T_2(n) = 4 \cdot T_2\left(\frac{n}{2}\right) + n, \quad T(1) = 1$$

The pre-lecture exercise showed you two ways to do this one!

One approach for all of these

- The “tree” approach from last time.
- Add up all the work done at all the sub-problems.

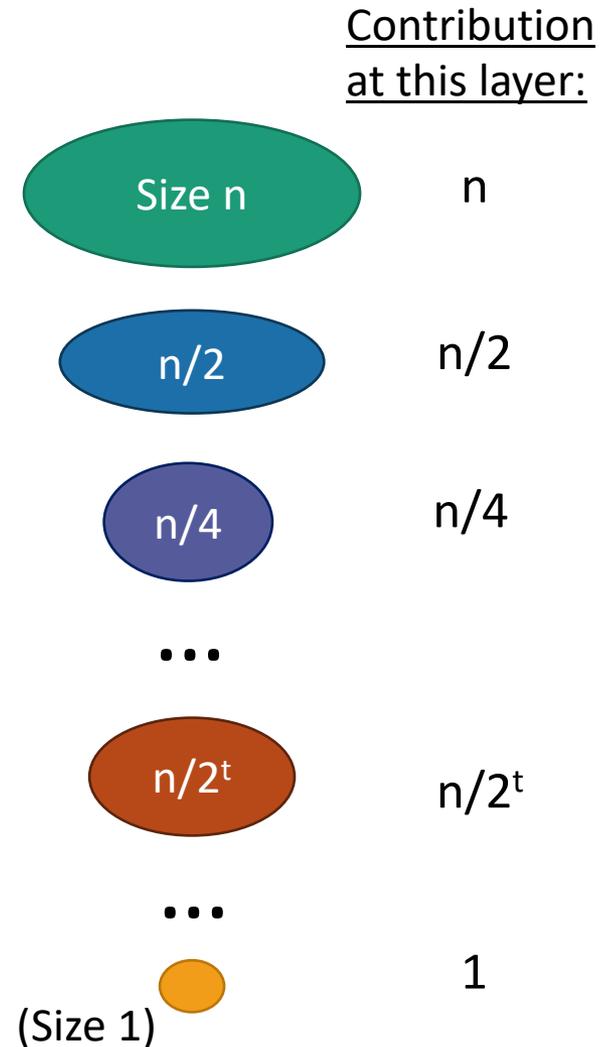


Solutions to pre-lecture exercise (1)

- $T_1(n) = T_1\left(\frac{n}{2}\right) + n, \quad T_1(1) = 1.$
- Adding up over all layers:

$$\sum_{i=0}^{\log(n)} \frac{n}{2^i} = 2n - 1$$

- So $T_1(n) = O(n).$



Solutions to pre-lecture exercise (2)

- $T_2(n) = 4T_2\left(\frac{n}{2}\right) + n, \quad T_2(1) = 1.$

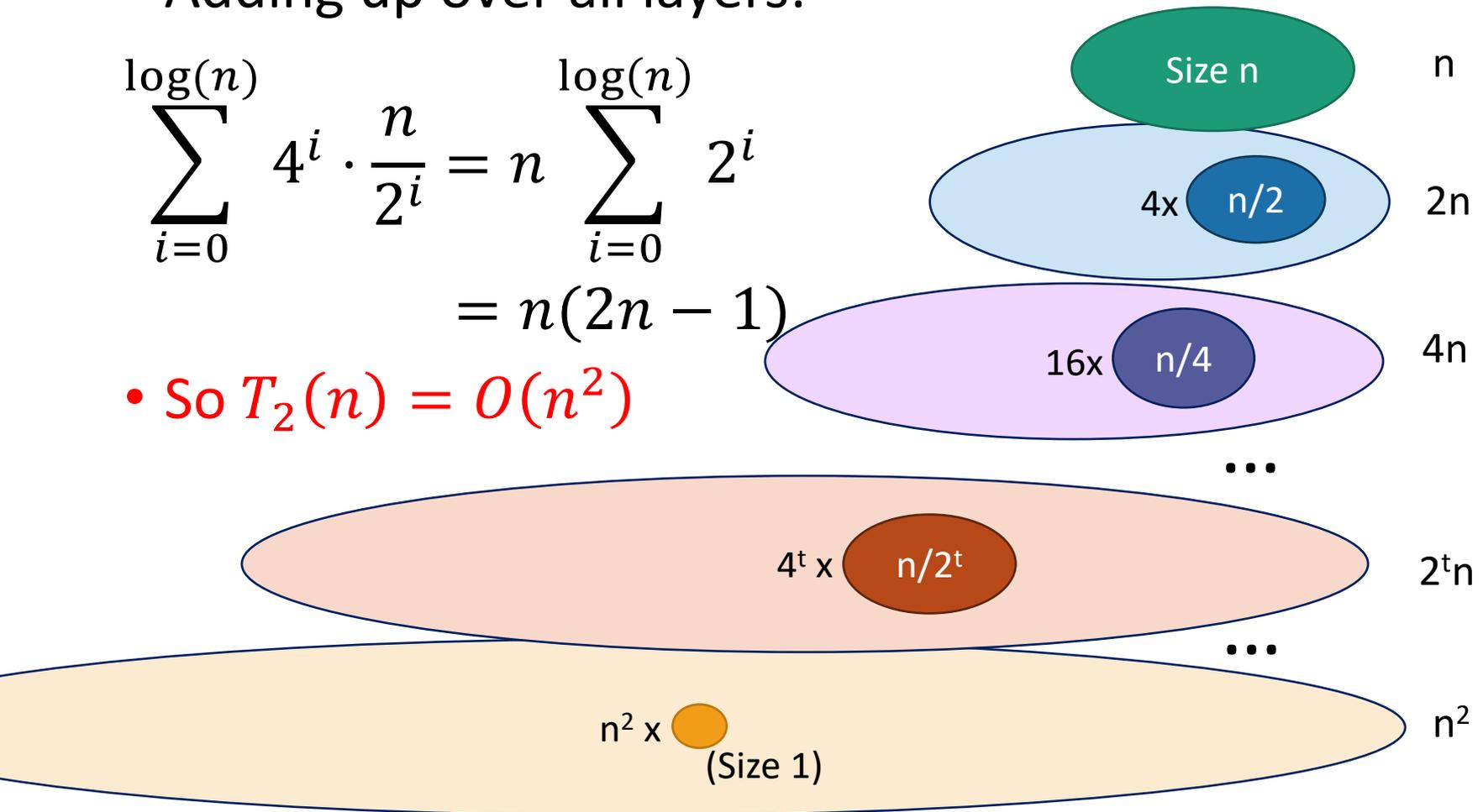
- Adding up over all layers:

$$\sum_{i=0}^{\log(n)} 4^i \cdot \frac{n}{2^i} = n \sum_{i=0}^{\log(n)} 2^i$$

$$= n(2n - 1)$$

- So $T_2(n) = O(n^2)$

Contribution
at this layer:



More examples

$T(n)$ = time to solve a problem of size n .

- Needless recursive integer multiplication

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

- $T(n) = O(n^2)$

This is similar to T_2 from the pre-lecture exercise.



- Karatsuba integer multiplication

- $T(n) = 3 T(n/2) + O(n)$

- $T(n) = O(n^{\log_2(3)} \approx n^{1.6})$

- MergeSort

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

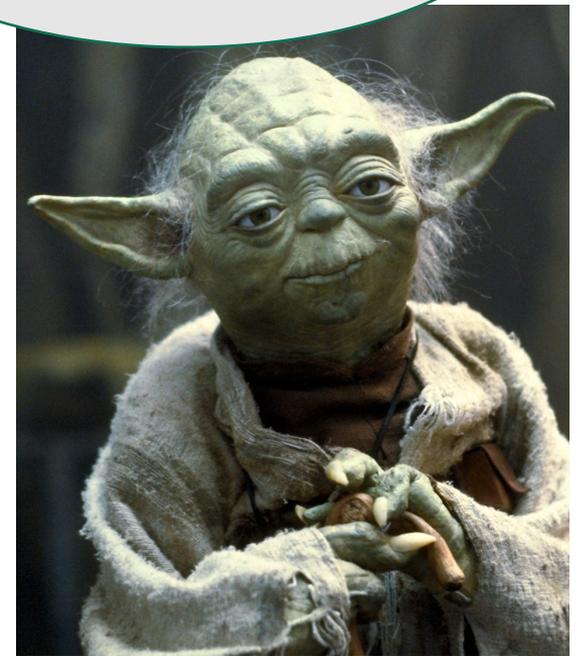
- $T(n) = O(n \log(n))$

What's the pattern?!?!?!?!?

The master theorem

- A formula for many recurrence relations.
 - We'll see an example Wednesday when it won't work.
- Proof: "Generalized" tree method.

A useful
formula it is.
Know why it works
you should.



Jedi master Yoda

We can also take n/b to mean either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$ and the theorem is still true.

The master theorem

- Suppose that $a \geq 1, b > 1$, and d are constants (independent of n).
- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Three parameters:
a : number of subproblems
b : factor by which input size shrinks
d : need to do n^d work to create all the subproblems and combine their solutions.

Many symbols those are....



Technicalities II

Integer division

Plucky the
Pedantic Penguin



- If n is odd, I can't break it up into two problems of size $n/2$.

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)$$

- However (see CLRS, Section 4.6.2 for details), one can show that the Master theorem works fine if you pretend that what you have is:

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

- From now on we'll mostly **ignore floors and ceilings** in recurrence relations.

Examples

(details on board)

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d).$$

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

- Needlessly recursive integer mult.

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

$$\begin{aligned} a &= 4 \\ b &= 2 \\ d &= 1 \end{aligned}$$

$$a > b^d$$



- Karatsuba integer multiplication

- $T(n) = 3 T(n/2) + O(n)$
- $T(n) = O(n^{\log_2(3)} \approx n^{1.6})$

$$\begin{aligned} a &= 3 \\ b &= 2 \\ d &= 1 \end{aligned}$$

$$a > b^d$$



- MergeSort

- $T(n) = 2T(n/2) + O(n)$
- $T(n) = O(n \log(n))$

$$\begin{aligned} a &= 2 \\ b &= 2 \\ d &= 1 \end{aligned}$$

$$a = b^d$$



- That other one

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

$$\begin{aligned} a &= 1 \\ b &= 2 \\ d &= 1 \end{aligned}$$

$$a < b^d$$



Proof of the master theorem

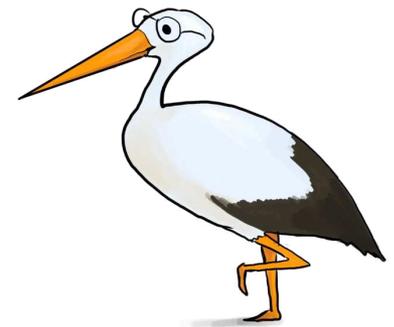
- We'll do the same recursion tree thing we did for MergeSort, but be more careful.
- Suppose that $T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$.

Hang on! The hypothesis of the Master Theorem was that the extra work at each level was $O(n^d)$, but we're writing $\leq cn^d$...



Plucky the
Pedantic Penguin

That's true ... the hypothesis should be that $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. For simplicity, today we are essentially we are assuming the $n_0 = 1$ in the definition of big-Oh. It's a good exercise to verify that the proof works for $n_0 > 1$ too.



Sigi the Studios Stork

Recursion tree

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

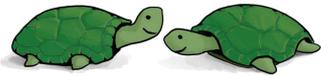


Fill this in! (Say $T(1) = c$)

	Level	# problems	Size of each problem	Amount of work at this level
	0	1	n	
	1	a	n/b	
	2	a^2	n/b^2	
<p>...</p>				
	t	a^t	n/b^t	
<p>...</p>	...			
	$\log_b(n)$	$a^{\log_b(n)}$	1	

Recursion tree

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$



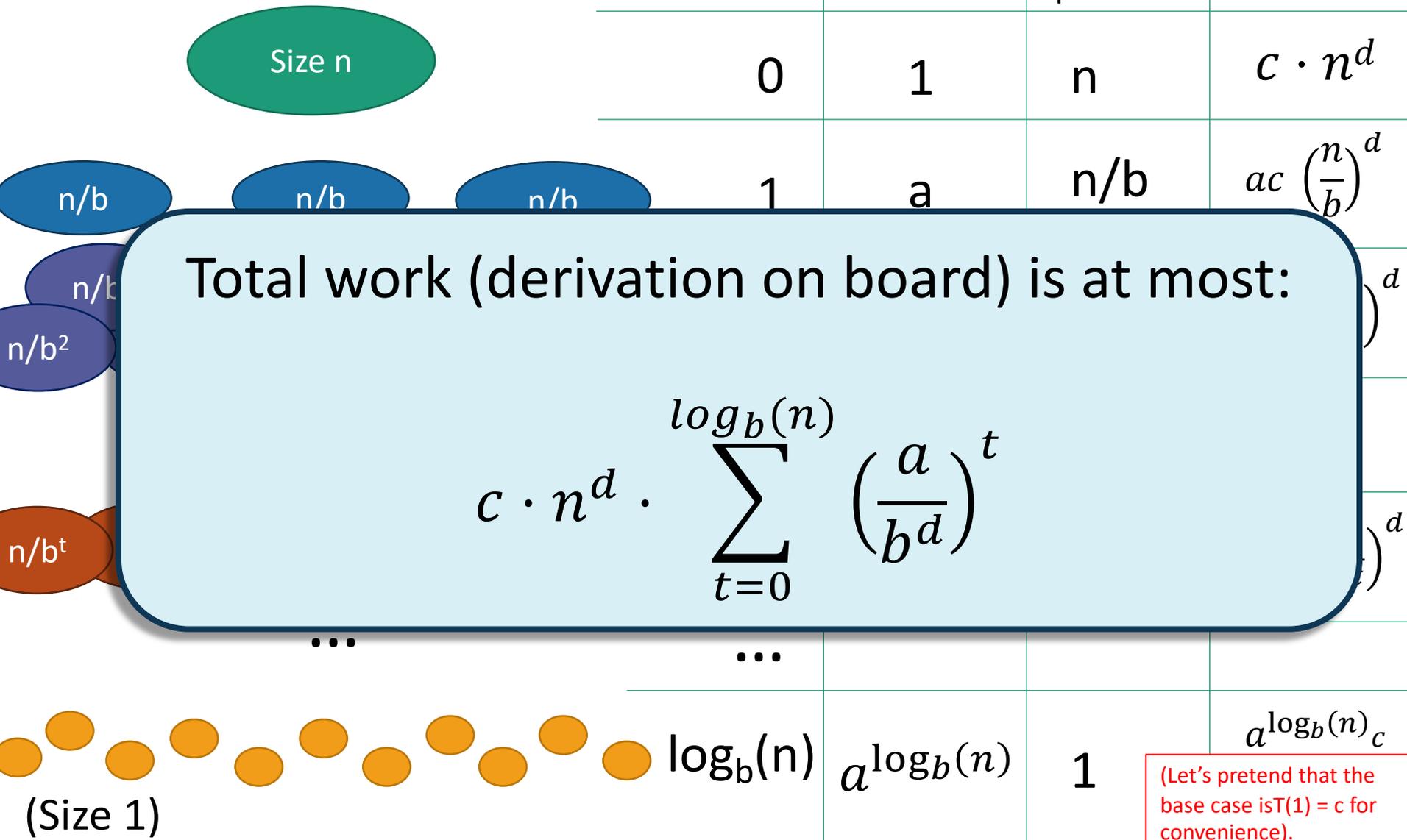
Fill this in! (Say $T(1) = c$)

	Level	# problems	Size of each problem	Amount of work at this level
	0	1	n	$c \cdot n^d$
	1	a	n/b	$a c \left(\frac{n}{b}\right)^d$
	2	a^2	n/b^2	$a^2 c \left(\frac{n}{b^2}\right)^d$
<p>...</p>				
	t	a^t	n/b^t	$a^t c \left(\frac{n}{b^t}\right)^d$
<p>...</p>	...			
<p>(Size 1)</p>	$\log_b(n)$	$a^{\log_b(n)}$	1	$a^{\log_b(n)} c$

(Let's pretend that the base case is $T(1) = c$ for convenience).

Recursion tree

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$



Total work (derivation on board) is at most:

$$c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t$$

(Let's pretend that the base case is $T(1) = c$ for convenience).

Now let's check all the cases

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Case 1: $a = b^d$

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

- $T(n) \leq c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t$ ← Equal to 1!
= $c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} 1$
= $c \cdot n^d \cdot (\log_b(n) + 1)$
= $c \cdot n^d \cdot \left(\frac{\log(n)}{\log(b)} + 1\right)$
= $\Theta(n^d \log(n))$

Case 2: $a < b^d$

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

• $T(n) \leq c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t$ ← Less than 1!

*Assume that x is constant and N is growing.

Aside: Geometric sums

- What is $\sum_{t=0}^N x^t$?
- You may remember that $\sum_{t=0}^N x^t = \frac{x^{N+1}-1}{x-1}$ for $x \neq 1$.
- Morally:

$$x^0 + x^1 + x^2 + x^3 + \dots + x^N$$

If $0 < x < 1$, this term dominates, and the whole sum is $\Theta(1)$

(If $x = 1$, all terms the same)

If $x > 1$, this term dominates, and the whole sum is $\Theta(x^N)$.

$$1 \leq \frac{1 - x^{N+1}}{1 - x} \leq \frac{1}{1 - x}$$

$$x^N \leq \frac{x^{N+1} - 1}{x - 1} \leq x^N \cdot \left(\frac{x}{x - 1} \right)$$

Case 2: $a < b^d$

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

- $T(n) \leq c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t$ ← Less than 1!
= $c \cdot n^d \cdot [\text{some constant}]$
= $\Theta(n^d)$

Case 3: $a > b^d$

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

• $T(n) \leq c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t$ ← Larger than 1!

$$= \Theta\left(n^d \left(\frac{a}{b^d}\right)^{\log_b(n)}\right)$$

$$= \Theta(n^{\log_b(a)})$$

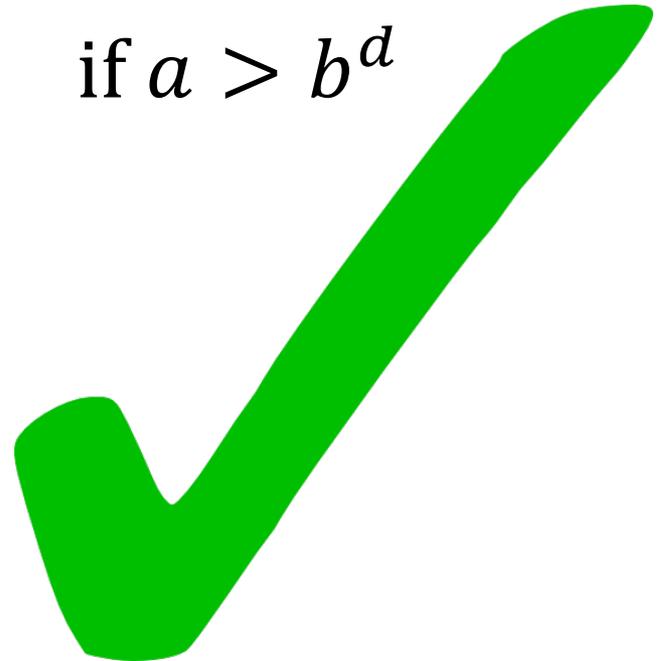
Convince yourself that this step is legit!

We'll do this step on the board!



Now let's check all the cases

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$



Understanding the Master Theorem

- Let $a \geq 1$, $b > 1$, and d be constants.
- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

- What do these three cases mean?

The eternal struggle



**Branching causes the number
of problems to explode!
The most work is at the
bottom of the tree!**

**The problems lower in
the tree are smaller!
The most work is at
the top of the tree!**

Consider our three warm-ups

1. $T(n) = T\left(\frac{n}{2}\right) + n$

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

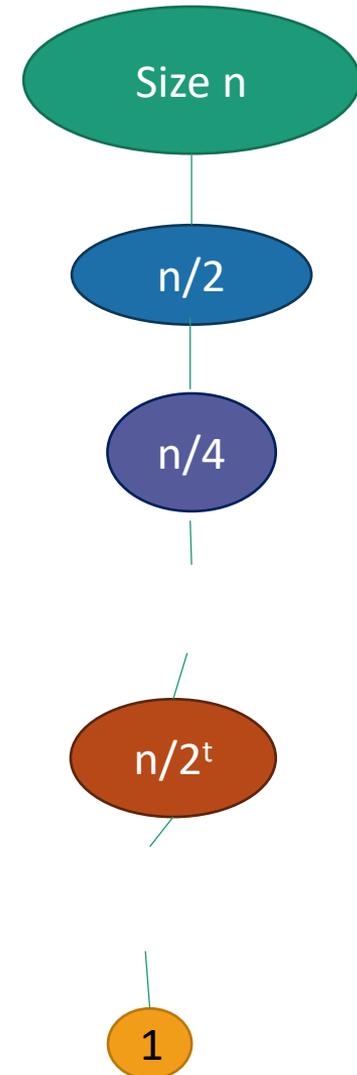
3. $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n$

First example: tall and skinny tree

$$1. T(n) = T\left(\frac{n}{2}\right) + n, \quad (a < b^d)$$

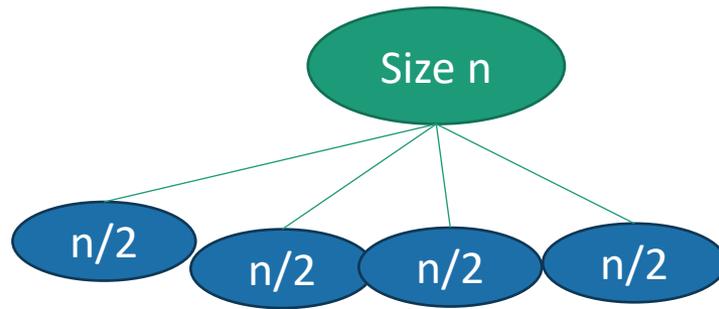
- The amount of work done at the top (the biggest problem) swamps the amount of work done anywhere else.

- $T(n) = O(\text{work at top}) = O(n)$



Third example: bushy tree

$$3. T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n, \quad (a > b^d)$$

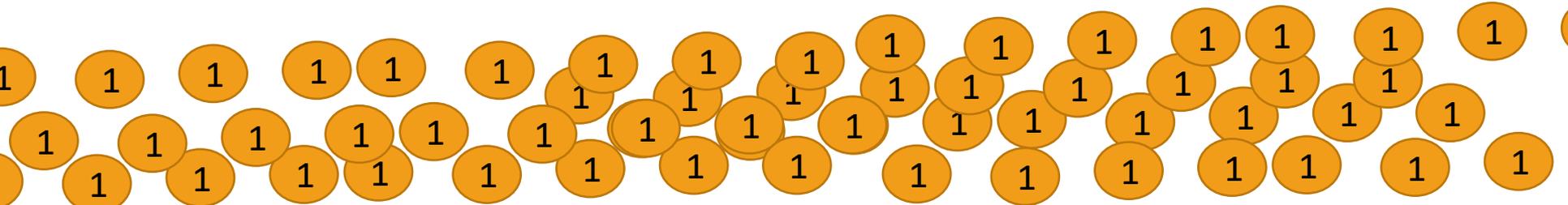


WINNER



Most work at the bottom of the tree!

- There are a HUGE number of leaves, and the total work is dominated by the time to do work at these leaves.
- $T(n) = O(\text{work at bottom}) = O(4^{\text{depth of tree}}) = O(n^2)$

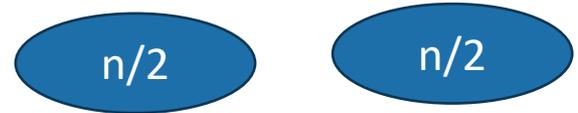


Second example: just right

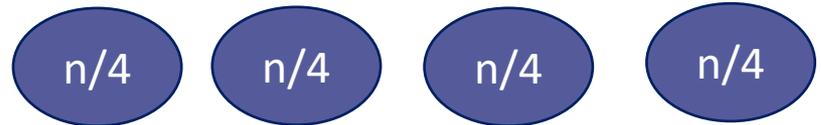
$$2. T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \quad (a = b^d)$$



- The branching **just** balances out the amount of work.



- The same amount of work is done at every level.



- $T(n) = (\text{number of levels}) * (\text{work per level})$
- $= \log(n) * O(n) = O(n \log(n))$



What have we learned?

- The “**Master Method**” makes our lives easier.
- But it’s basically just codifying a calculation we could do from scratch if we wanted to.

The Substitution Method

- Another way to solve recurrence relations.
- More general than the master method.
- **Step 1:** Generate a guess at the correct answer.
- **Step 2:** Try to prove that your guess is correct.
- **(Step 3: Profit.)**

The Substitution Method

first example

- Let's return to:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \text{ with } T(1) = 1.$$

- (assuming n is a power of 2...)
- The Master Method says $T(n) = O(n \log(n))$.
- We will prove this via the Substitution Method.

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \text{ with } T(1) = 1.$$

Step 1: Guess the answer

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$
- $T(n) = 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$
- $T(n) = 4 \cdot T\left(\frac{n}{4}\right) + 2n$
- $T(n) = 4 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$
- $T(n) = 8 \cdot T\left(\frac{n}{8}\right) + 3n$
- ...

Expand $T\left(\frac{n}{2}\right)$

Simplify

Expand $T\left(\frac{n}{4}\right)$

Simplify

You can guess the answer however you want: meta-reasoning, a little bird told you, wishful thinking, etc. One useful way is to try to “unroll” the recursion, like we’re doing here.



Guessing the pattern: $T(n) = 2^j \cdot T\left(\frac{n}{2^j}\right) + j \cdot n$

Plug in $j = \log(n)$, and get

$$T(n) = n \cdot T(1) + \log(n) \cdot n = n(\log(n) + 1)$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \text{ with } T(1) = 1.$$

Step 2: Prove the guess is correct.

- Inductive Hyp.: $T(n) = n(\log(n) + 1)$.
- Base Case (n=1): $T(1) = 1 = 1 \cdot (\log(1) + 1)$
- Inductive Step:
 - Assume Inductive Hyp. for $1 \leq n < k$:
 - Suppose that $T(n) = n(\log(n) + 1)$ for all $1 \leq n < k$.
 - Prove Inductive Hyp. for n=k:
 - $T(k) = 2 \cdot T\left(\frac{k}{2}\right) + k$ by definition
 - $T(k) = 2 \cdot \left(\frac{k}{2} \left(\log\left(\frac{k}{2}\right) + 1\right)\right) + k$ by induction.
 - $T(k) = k(\log(k) + 1)$ by simplifying.
 - So Inductive Hyp. holds for n=k.
- Conclusion: For all $n \geq 1$, $T(n) = n(\log(n) + 1)$

We're assuming the n is a power of 2, so we're not worried about floors and ceilings. What would you need to change if n wasn't a power of 2?



Step 3: Profit

- Pretend like you never did Step 1, and just write down:
- *Theorem:* $T(n) = O(n \log(n))$
- *Proof:* [Whatever you wrote in Step 2]

What have we learned?

- The substitution method is a different way of solving recurrence relations.
- **Step 1:** Guess the answer.
- **Step 2:** Prove your guess is correct.
- **Step 3:** Profit.
- We'll get more practice with the substitution method next lecture!

Another example (if time)

(If not time, that's okay; we'll see these ideas in Lecture 4)

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n$
- $T(2) = 2$

- Step 1: Guess: $O(n \log(n))$ (divine inspiration).
- But I don't have such a precise guess about the form for the $O(n \log(n))$...
 - That is, what's the leading constant?
- Can I still do Step 2?

Aside: What's wrong with this?

This is NOT
CORRECT!!!



Plucky the
Pedantic
Penguin

- **Inductive Hypothesis:** $T(n) = O(n \log(n))$
- Base case: $T(2) = 2 = O(1) = O(2 \log(2))$
- Inductive Step:
 - Suppose that $T(n) = O(n \log(n))$ for $n < k$.
 - Then $T(k) = 2 \cdot T\left(\frac{k}{2}\right) + 32 \cdot k$ by definition
 - So $T(k) = 2 \cdot O\left(\frac{k}{2} \log\left(\frac{k}{2}\right)\right) + 32 \cdot k$ by induction
 - But that's $T(k) = O(k \log(k))$, so the I.H. holds for $n=k$.
- Conclusion:
 - By induction, $T(n) = O(n \log(n))$ for all n .

Figure out
what's
wrong
here!!!



Siggi the Studios Stork

Red flag: We can use the same reasoning to prove that $T(n) = O(n)$, which is not true!

This slide skipped in class

Aside: What's wrong with this?

- Inductive Hypothesis: $T(n) = O(n)$
- Base case: $T(2) = 2 = O(1) = O(2)$
- Inductive Step:
 - Suppose that $T(n) = O(n)$ for $n < k$.
 - Then $T(k) = 2 \cdot T\left(\frac{k}{2}\right) + 32 \cdot k$ by definition
 - So $T(k) = 2 \cdot O\left(\frac{k}{2}\right) + 32 \cdot k$ by induction
 - But that's $T(k) = O(k)$, so the I.H. holds for $n=k$.
- Conclusion:
 - By induction, $T(n) = O(n)$ for all n .

This is NOT
CORRECT!!!



Plucky the
Pedantic
Penguin

Red flag: We can use the same reasoning to prove that $T(n) = O(n)$, which is not true!

This slide skipped in class

Aside: What's wrong with this?

This is NOT
CORRECT!!!



Plucky the
Pedantic
Penguin

- Inductive Hypothesis: $T(n) = O(n)$
- Base case: $T(2) = 2 = O(1) = O(2)$
- Inductive Step:
 - Suppose that $T(n) = O(n)$ for $n < k$.
 - Then $T(k) = 2 \cdot T\left(\frac{k}{2}\right) + 32 \cdot k$ by definition
 - So $T(k) = 2 \cdot O\left(\frac{k}{2}\right) + 32 \cdot k$ by induction
 - But that's $T(k) = O(k)$, so the I.H. holds for $n=k$.
- Conclusion:
 - By induction, $T(n) = O(n)$ for all n .

The problem is that this doesn't make any sense. We can't have " $T(n) = O(n)$ for $n < k$," since the def. of big-Oh needs to hold for all n .

Another example (if time)

(If not time, that's okay; we'll see these ideas in Lecture 4)

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n$
- $T(2) = 2$

- Step 1: Guess: $O(n \log(n))$ (divine inspiration).
- But I don't have such a precise guess about the form for the $O(n \log(n))$...
 - That is, what's the leading constant?
- Can I still do Step 2?

Step 2: Prove it, working backwards to figure out the constant

- **Guess:** $T(n) \leq C \cdot n \log(n)$ for some constant C TBD.
- **Inductive Hypothesis (for $n \geq 2$):** $T(n) \leq C \cdot n \log(n)$
- **Base case:** $T(2) = 2 \leq C \cdot 2 \log(2)$ as long as $C \geq 1$
- **Inductive Step:**

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n$$
$$T(2) = 2$$

Inductive step

- Assume that the inductive hypothesis holds for $n < k$.
- $T(k) = 2T\left(\frac{k}{2}\right) + 32k$
- $\leq 2C \frac{k}{2} \log\left(\frac{k}{2}\right) + 32k$
- $= k(C \cdot \log(k) + 32 - C)$
- $\leq k(C \cdot \log(k))$ as long as $C \geq 32$.
- Then the inductive hypothesis holds for $n=k$.

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n$$

$$T(2) = 2$$

Step 2: Prove it, working backwards to figure out the constant

- **Guess:** $T(n) \leq C \cdot n \log(n)$ for some constant C TBD.
- **Inductive Hypothesis (for $n \geq 2$):** $T(n) \leq C \cdot n \log(n)$
- **Base case:** $T(2) = 2 \leq C \cdot 2 \log(2)$ as long as $C \geq 1$
- **Inductive step:** Works as long as $C \geq 32$
 - So choose $C = 32$.
- **Conclusion:** $T(n) \leq 32 \cdot n \log(n)$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n$$
$$T(2) = 2$$

Technically this proof isn't quite correct since we assumed n was a power of 2, but the theorem statement is about all large enough n .



Plucky the Pedantic Penguin

Step 3: Profit.

- **Theorem:** $T(n) = O(n \log(n))$
- **Proof:**
 - **Inductive Hypothesis:** $T(n) \leq 32 \cdot n \log(n)$
 - **Base case:** $T(2) = 2 \leq 32 \cdot 2 \log(2)$ is true.
 - **Inductive step:**
 - Assume Inductive Hyp. for $n < k$.
 - $T(k) = 2T\left(\frac{k}{2}\right) + 32k$ By the def. of $T(k)$
 - $\leq 2 \cdot 32 \cdot \frac{k}{2} \log\left(\frac{k}{2}\right) + 32k$ By induction
 - $= k(32 \cdot \log(k) + 32 - 32)$
 - $= 32 \cdot k \log(k)$
 - This establishes inductive hyp. for $n = k$.
 - **Conclusion:** $T(n) \leq 32 \cdot n \log(n)$ for all $n \geq 2$.
 - By the definition of big-Oh, with $n_0 = 2$ and $c = 32$, this implies that $T(n) = O(n \log(n))$

Why two methods?

- Sometimes the Substitution Method works where the Master Method does not.
- More on this next time!

Next Time

- What happens if the sub-problems are different sizes?
- And when might that happen?

BEFORE Next Time

- Pre-Lecture Exercise 4!
- Hand in HW1!