Lecture 3

Recurrence Relations and how to solve them!

Announcements!

- **Sections and office hours** have been scheduled!
  - See course website for details.
- **HW1** is posted!
  - Due Friday.
  - Follow the Piazza post on HW1 for updates.
- (STILL) sign up for **Piazza!**
  - There's a link on the course website.
  - Course announcements will be posted on Piazza.
- OAE requests and exam conflicts:
  - cs161-win1819-staff@lists.stanford.edu
  - Please send requests by Jan. 21 (one week from today)

Last time….

- Sorting: InsertionSort and MergeSort
- Analyzing correctness of iterative + recursive algs
  - Via “loop invariant” and induction
- Analyzing running time of recursive algorithms
  - By writing out a tree and adding up all the work done.

- How do we measure the runtime of an algorithm?
  - Worst-Case Analysis
  - Big-Oh Notation

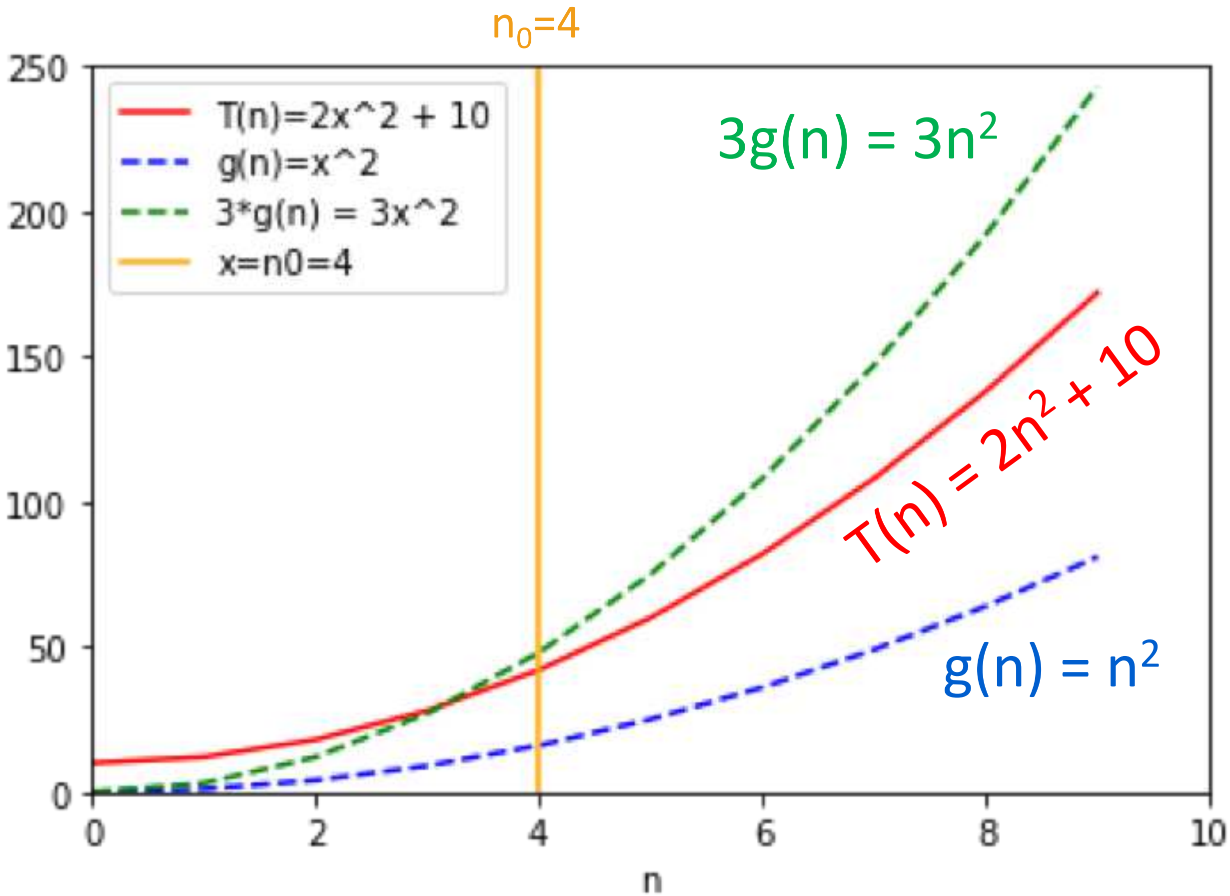
Quick clarification

about proving things about O() notation

SLIDE FROM LAST TIME

Example
$2n^2 + 10 = O(n^2)$

$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



Formally:
- Choose c = 3
- Choose n_0 = 4
- Then:

$$\forall n \geq 4,$$
$$0 \leq 2n^2 + 10 \leq 3 \cdot n^2$$

In order to **formally prove** $2n^2 + 10 = O(n^2)$

- Choose $n_0 = 4$ and $c = 3$.
- Claim: For all $n \geq 4$, we have $0 \leq 2 \cdot n^2 + 10 \leq 3 \cdot n^2$.
- To prove the claim, first notice that for $n \geq 4$,

$$2 \cdot n^2 + 10 \leq 3 \cdot n^2$$
$$\Leftrightarrow$$
$$10 \leq n^2$$
$$\Leftrightarrow$$
$$\sqrt{10} \leq n$$

This is sufficient rigor for your homework!

- This last thing is true for any $n \geq 4$, since

$$\sqrt{10} \approx 3.16 \leq 4.$$

- We also have $0 \leq 2 \cdot n^2 + 10$ for all n, since $n^2 \geq 0$ is always positive.

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's call this running time T(n).
  - when the input has length n.
- We know that T(n) = O(nlog(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 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(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(nlog(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, T(1) = O(1), so sometimes we'll just omit it.

Why does T(1) = O(1)?

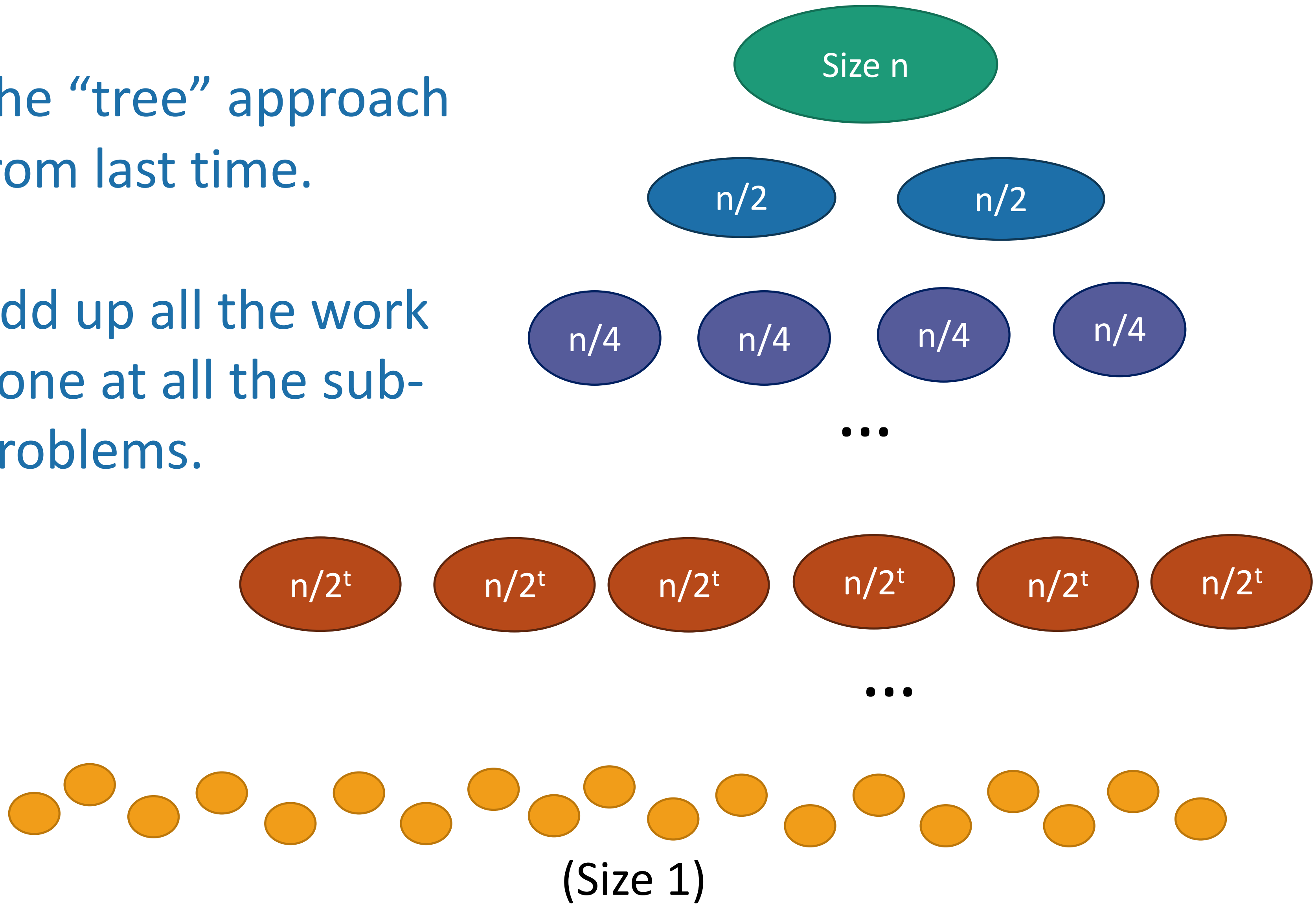
Siggi the Studious Stork

On your pre-lecture exercise

- You played around with these examples (when n is a power of 2):
  1. $T(n) = T\left(\frac{n}{2}\right) + n,\qquad T(1) = 1$
  2. $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n,\qquad T(1) = 1$
  3. $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n,\qquad T(1) = 1$

One approach for all of these

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

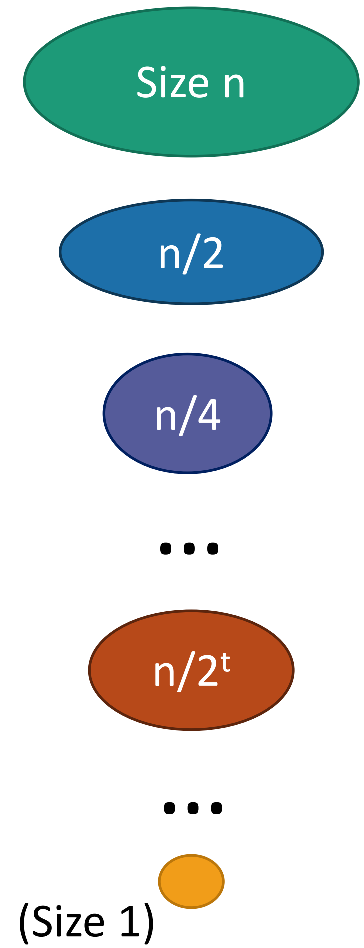


Solutions to pre-lecture exercise (1)

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

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

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



Contribution at this layer:

n

n/2

n/4

$n/2^t$

1

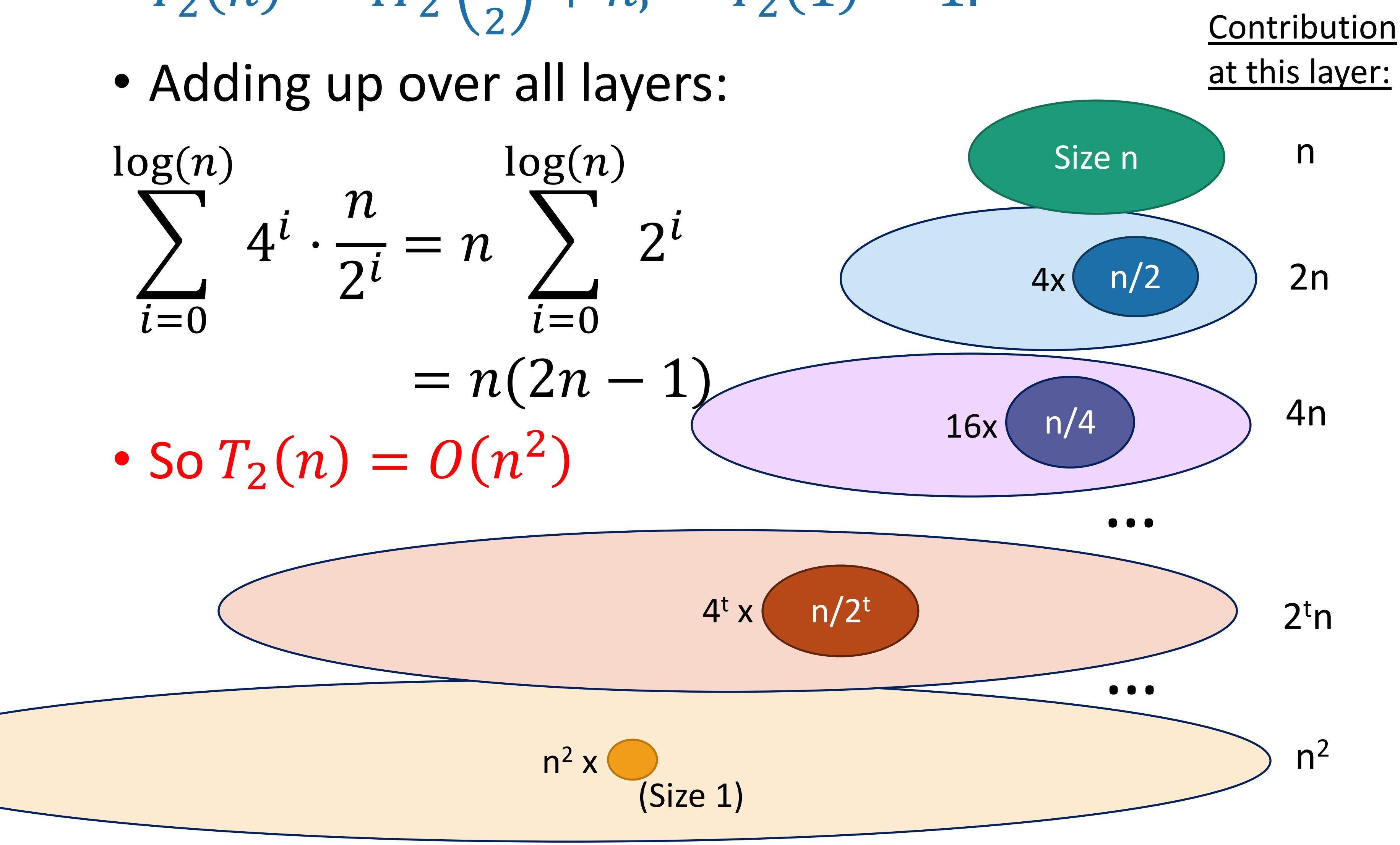
Solutions to pre-lecture exercise (2)

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



More examples

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

- Needlessly 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

The master theorem

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

- 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

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

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), one can show that the Master theorem works fine if you pretend that what you have is:

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

- Read CLRS 4.6.2; and 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)

a = 4
b = 2
d = 1

$a > b^d$

?

- Karatsuba integer multiplication
  - T(n) = 3 T(n/2) + O(n)
  - T(n) = O($n^{\log\_2(3)} \approx n^{1.6}$)

a = 3
b = 2
d = 1

$a > b^d$

?

- MergeSort
  - T(n) = 2T(n/2) + O(n)
  - T(n) = O(nlog(n))

a = 2
b = 2
d = 1

$a = b^d$

?

- That other one
  - T(n) = T(n/2) + O(n)
  - T(n) = O(n)

a = 1
b = 2
d = 1

$a < b^d$

?

Proof of the master theorem

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

Hang on! The hypothesis of the Master Theorem was the the extra work at each level was $O(n^d)$. That's NOT the same as work <= cn^d for some constant c.

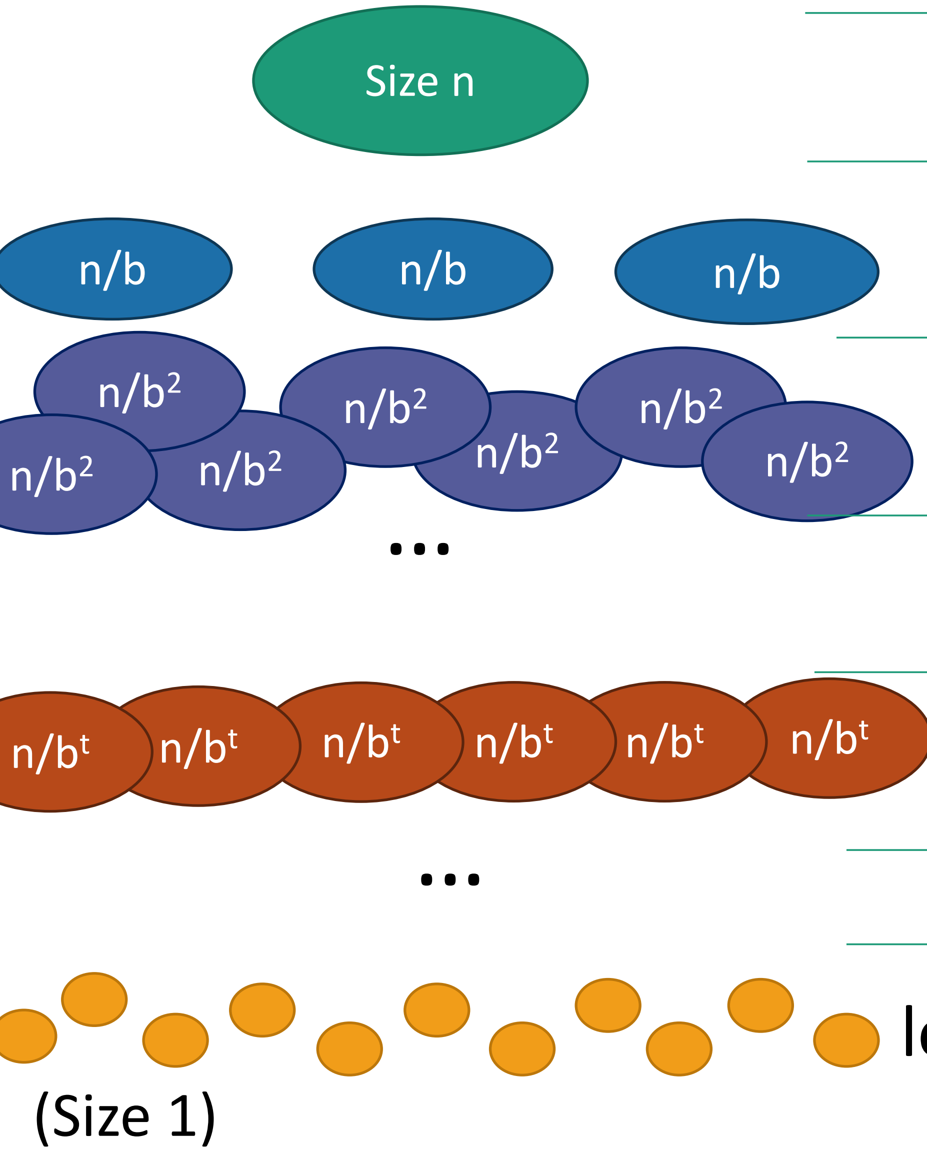
Plucky the Pedantic Penguin

That's true … we'll actually prove a weaker statement that uses this hypothesis instead of the hypothesis that $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. It's a good exercise to make this proof work rigorously with the O() notation.

Siggi the Studious Stork

Recursion tree

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

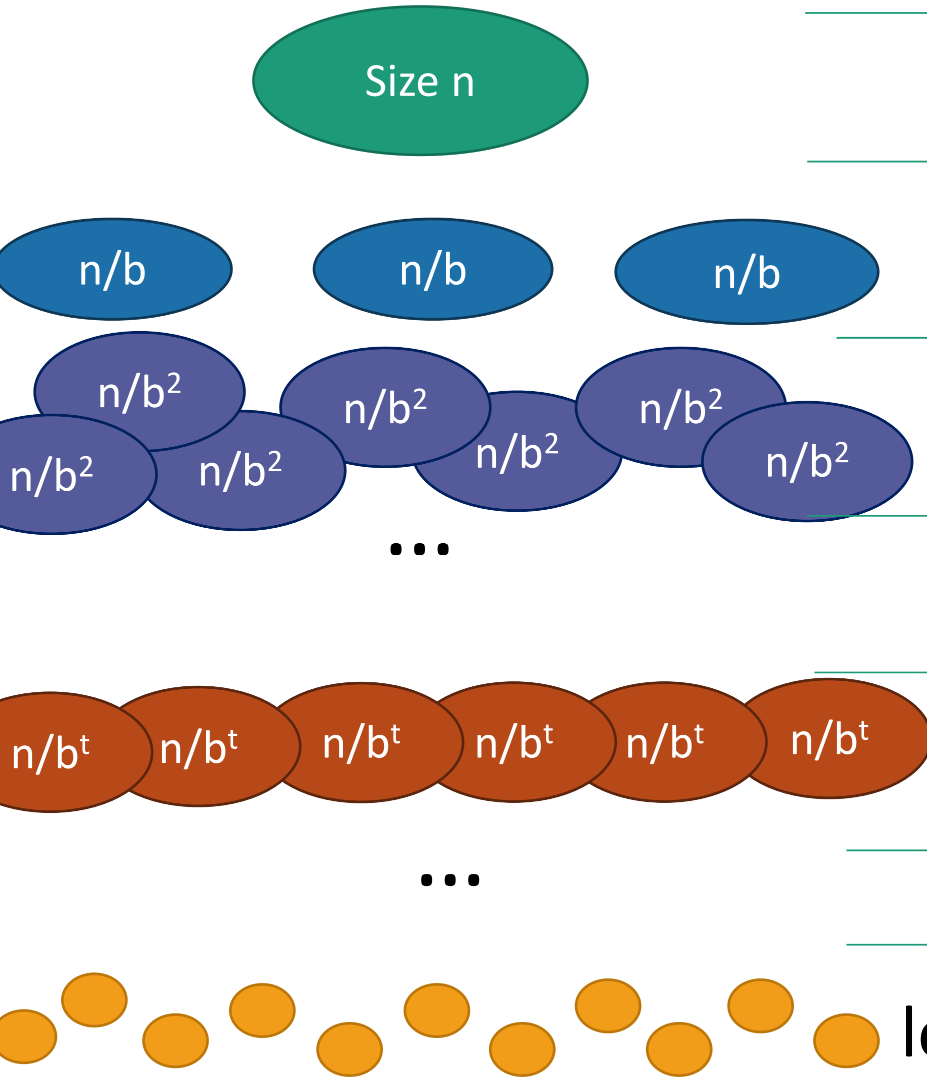


Level	# problems	Size of each problem	Amount of work at this level
0	1	n	
1	a	n/b	
2	a^2	n/b^2	
t	a^t	n/b^t	
…			
$\log_b(n)$	$a^{\log_b(n)}$	1	

Recursion tree

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

Help me fill this in!



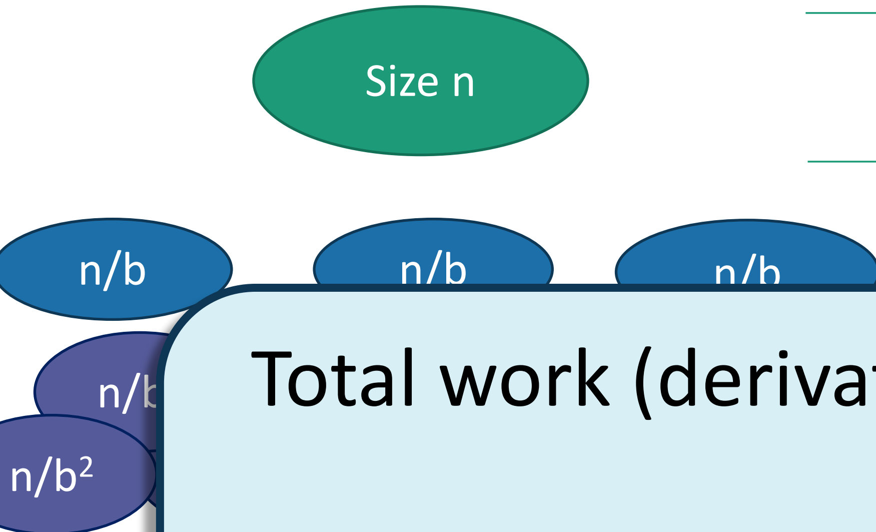
(Size 1)

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

(Let's pretend that the base case isT(1) = c for convenience).

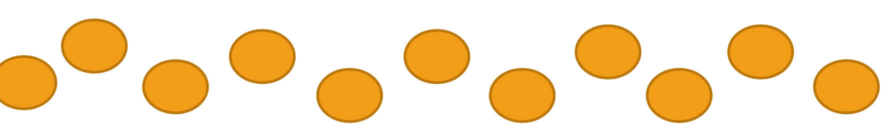
Recursion tree

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



Level	# problems	Size of each problem	Amount of work at this level
0	1	n	$c \cdot n^d$
1	a	n/b	$ac\left(\frac{n}{b}\right)^d$
			$)^d$
			$)^d$
…			
$\log_b(n)$	$a^{\log_b(n)}$	1	$a^{\log_b(n)} c$ (Let's pretend that the base case isT(1) = c for convenience).

…



(Size 1)

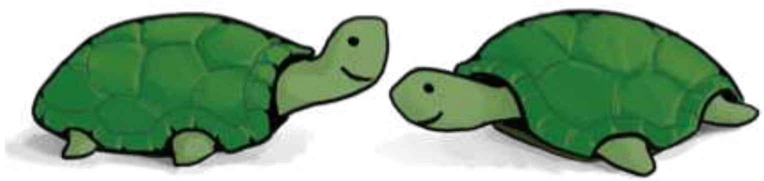
Total work (derivation on board) is at most:

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

Now let's check all the cases

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

Do the first one!



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

Case 1: $a = b^d$

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

$= c \cdot n^d \cdot \sum_{t=0}^{log_b(n)} 1$

$= c \cdot n^d \cdot (\log_b(n) + 1)$

$= c \cdot n^d \cdot \left(\frac{\log(n)}{\log(b)} + 1\right)$

$= \Theta\left(n^d \log(n)\right)$

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

Case 2: $a < b^d$

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

Less than 1!

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:

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

If $0 < x < 1$, this term dominates.

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

If $x > 1$, this term dominates.

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

(Aka, doesn't depend on N).

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

(Aka, $\Theta(x^N)$ if x is constant and N is growing).

Case 2: $a < b^d$

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

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

$= c \cdot n^d \cdot [\text{some constant}]$

$= \Theta\left(n^d\right)$

Case 3: $a > b^d$

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

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

Larger than 1!

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

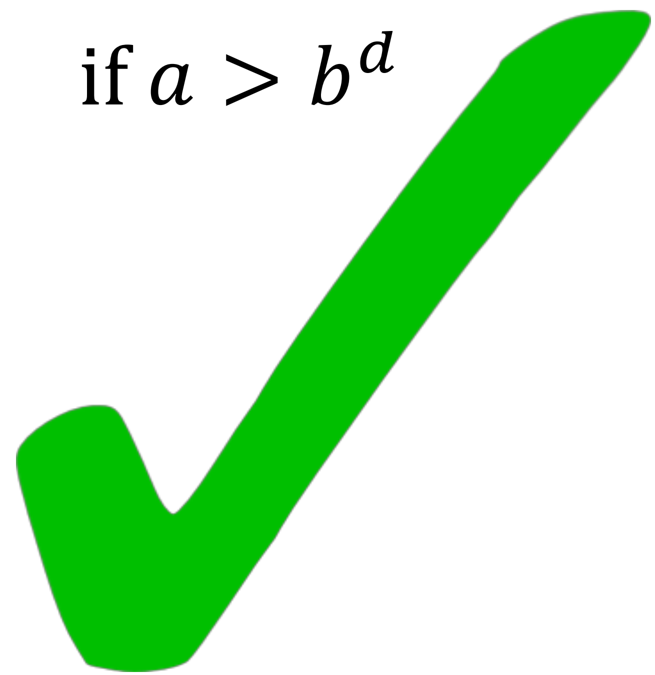
Convice yourself that this step is legit!

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

We'll do this step on the board!

Now let's check all the cases

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



Even more generally, for T(n) = aT(n/b) + f(n)...

Theorem 3.2 (Master Theorem). *Let $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ be a recurrence where $a \geq 1$, $b > 1$. Then,*

- *If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, $T(n) = \Theta\left(n^{\log_b a}\right)$.*
- *If $f(n) = \Theta\left(n^{\log_b a}\right)$, $T(n) = \Theta\left(n^{\log_b a} \log n\right)$.*
- *If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

[From CLRS]

Figure out how to adapt the proof we gave to prove this more general version!

Ollie the Over-Achieving Ostrich

Understanding the Master Theorem

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

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

- 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!

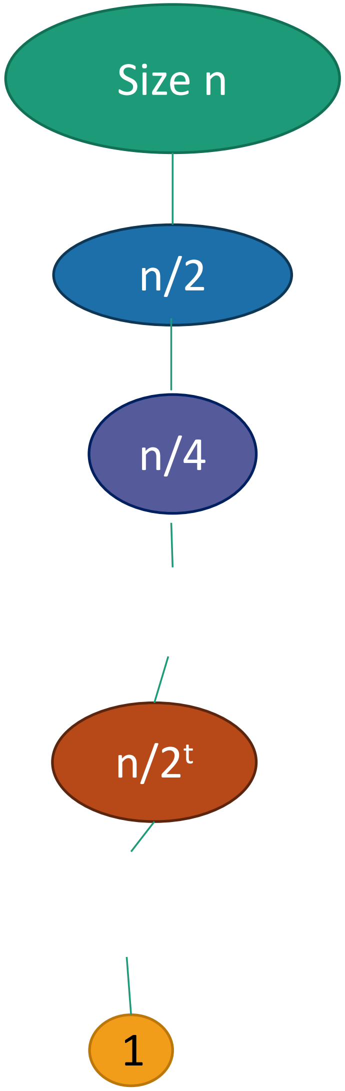
The problems lower in the tree are smaller!
The most work is at the top of the tree!

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



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

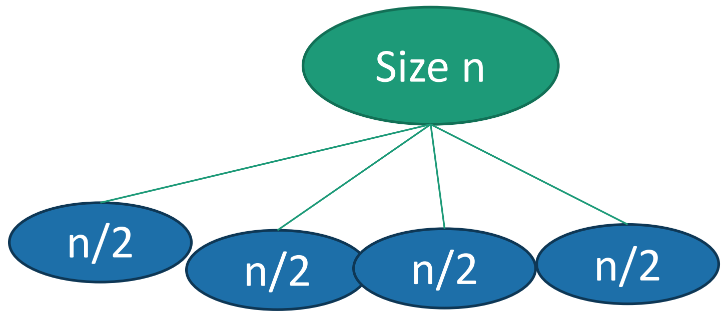
- T(n) = O(work at top) = O(n)

WINNER

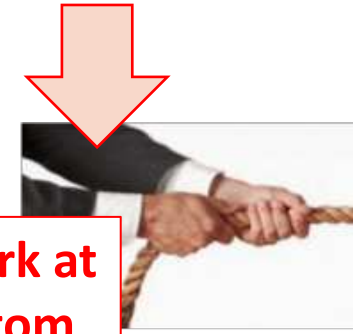
Most work at the top of the tree!

Third example: bushy tree

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



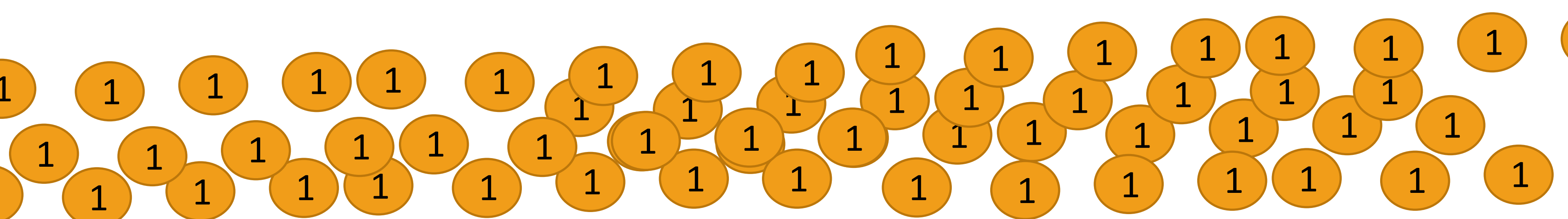
WINNER



Most work at the bottom of the tree!

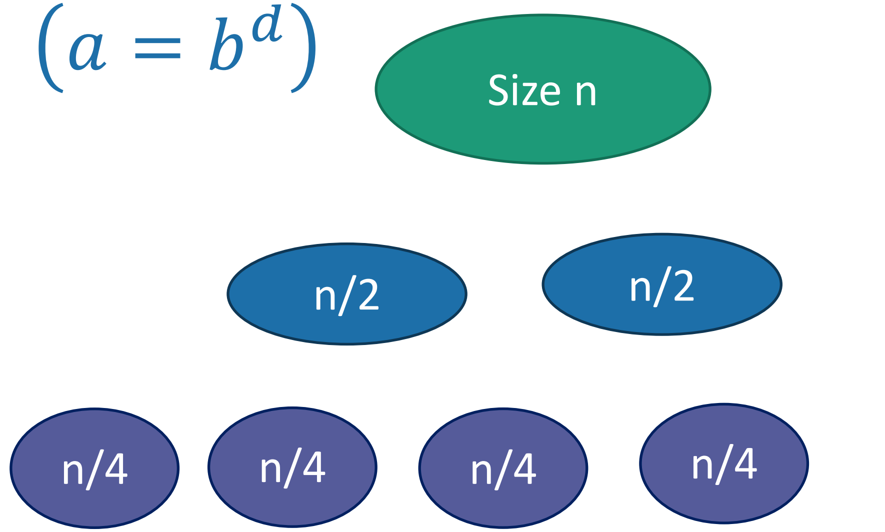
- There are a HUGE number of leaves, and the total work is dominated by the time to do work at these leaves.

- T(n) = O(work at bottom) = O($4^{\text{depth of tree}}$) = O(n^2)



Second example: just right

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



- The branching **just** balances out the amount of work.
- The same amount of work is done at every level.
- T(n) = (number of levels) * (work per level)
- = log(n) * O(n) = O(nlog(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:

  $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$, with $T(0) = 0, T(1) = 1$.

- The Master Method says $T(n) = O(n \log(n))$.
- We will prove this via the Substitution Method.

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

Step 1: Guess the answer

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

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

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

Simplify

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

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

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

Simplify

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

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

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

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

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.

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

Step 2: Prove the guess is correct.

- Inductive Hyp. (n): $T(j) = j(\log(j) + 1)$ for all $1 \le j \le n$.
- Base Case (n=1): $T(1) = 1 = 1 \cdot (\log(1) + 1)$
- Inductive Step:
  - Assume Inductive Hyp. for n=k-1:
    - Suppose that $T(j) = j(\log(j) + 1)$ for all $1 \le j \le k - 1$.
  - $T(k) = 2 \cdot T\left(\frac{k}{2}\right) + k$ 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 $n \ge 1, T(n) = n(\log(n) + 1)$

We just replaced the "n" in the statement of the inductive hypothesis with an "k-1" to get the I.H. for k-1.

We're being sloppy here about floors and ceilings…what would you need to do to be less sloppy?

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?

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: $T(j) \leq C \cdot j \log(j)$ for $2 \leq j \leq n$
- 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-1.
- $T(k) = 2T\left(\frac{k}{2}\right) + 32k$
- $\leq 2C\frac{k}{2}\log\left(\frac{k}{2}\right) + 32k$
- $= k(C \cdot \log(k) + 32 \; - C)$
- $\leq k(C \cdot \log(k))$ 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: $T(j) \leq C \cdot j \log(j)$ for $2 \leq j \leq n$
- 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: $T(n) \leq 32 \cdot n \log(n)$

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

Step 3: Profit.

- ***Theorem:*** $T(n) = O(n\log(n))$
- ***Proof:***
  - Inductive Hypothesis: $T(j) \leq 32 \cdot j\log(j)$ for $2 \leq j \leq n$
  - Base case: $T(2) = 2 \leq 32 \cdot 2\log(2)$ is true.
  - Inductive step:
    - Assume Inductive Hyp. for n=k-1.
    - $T(k) = 2T\left(\frac{k}{2}\right) + 32k$ 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 $n \geq 2$.

This slide added after lecture

Aside:

The form of the inductive hypothesis

- In the previous examples, we had an inductive hypothesis of the form:

$$T(j) \leq 32 \cdot j \log(j) \text{ for } 2 \leq j \leq n$$

  - The reason it was written like that is because that's what it should be if I'm doing "standard" induction. That is, the inductive step is: assuming that the inductive hypothesis holds for k-1, show that it holds for k.
- However, if one uses strong induction, it's fine to use an inductive hypothesis of the form:

$$T(n) \leq 32 \cdot n \log(n)$$

  - In this case, the inductive step would be: assuming that the inductive hypothesis holds for all $2 \leq j < k$, show that it holds for k.
- Both ways are totally fine.

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!