CSE100: Design and Analysis of Algorithms
Lecture 05 – Recurrences, Asymtotics

Feb 1st 2022

Solving Recurrences and Master Theorem (cont.)

Think of $T(n)$ as a runtime: positive and increasing in n

Asymptotic Bounds (review)

- Let $T(n)$, $g(n)$ be functions of positive integers.
- We say "$T(n)$ is $O\big(g(n)\big)$" if $T(n)$ grows no faster than $g(n)$ as n gets large. Formally,

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

- We say "$T(n)$ is $\Omega\big(g(n)\big)$" if $T(n)$ grows at least as fast as $g(n)$ as n gets large. Formally,

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

- We say "$T(n)$ is $\Theta(g(n))$" iff both: $T(n) = O\big(g(n)\big)$ and $T(n) = \Omega\big(g(n)\big)$

An Example: Formally prove $2n^2 + 10 = O(n^2)$ (review)

- 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 a midterm problem

- 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.

Another Example (review)

- For any $k \geq 1, n^k$ is NOT $O\left(n^{k-1}\right)$.
- Proof:
  - Suppose that it were. Then there is some c, n_0 so that
    $$n^k \leq c \cdot n^{k-1} \text{ for all } n \geq n_0$$
  - Aka, $n \leq c$ for all $n \geq n_0$
  - But that's not true! What about $n = n_0 + c + 1$?!
  - We have a contradiction! It *can't* be that $n^k = O\left(n^{k-1}\right)$.

Recap: Asymptotic Notation

This is my
happy face!

- This makes both Plucky and Lucky happy.
  - **Plucky the Pedantic Penguin** is happy because there is a precise definition.
  - **Lucky the Lackadaisical Lemur** is happy because we don't have to pay close attention to all those pesky constant factors like "11".
- But we should always be careful not to abuse it.
- In the course, (almost) every algorithm we see will be actually practical, without needing to take $n \geq n_0 = 2^{10000000}$.

Questions about asymptotic notation?

Today

- How do we measure the runtime of an algorithm?
  - Worst-case analysis
  - Asymptotic Analysis
- 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.

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 $n \leq 1$:
    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

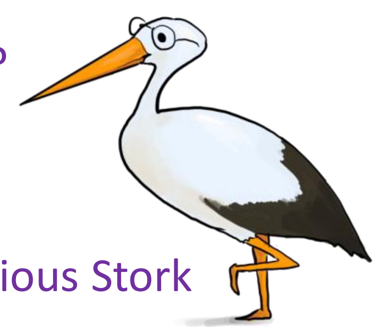
Plucky the Pedantic Penguin

- 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 $T(1) = O(1)$?



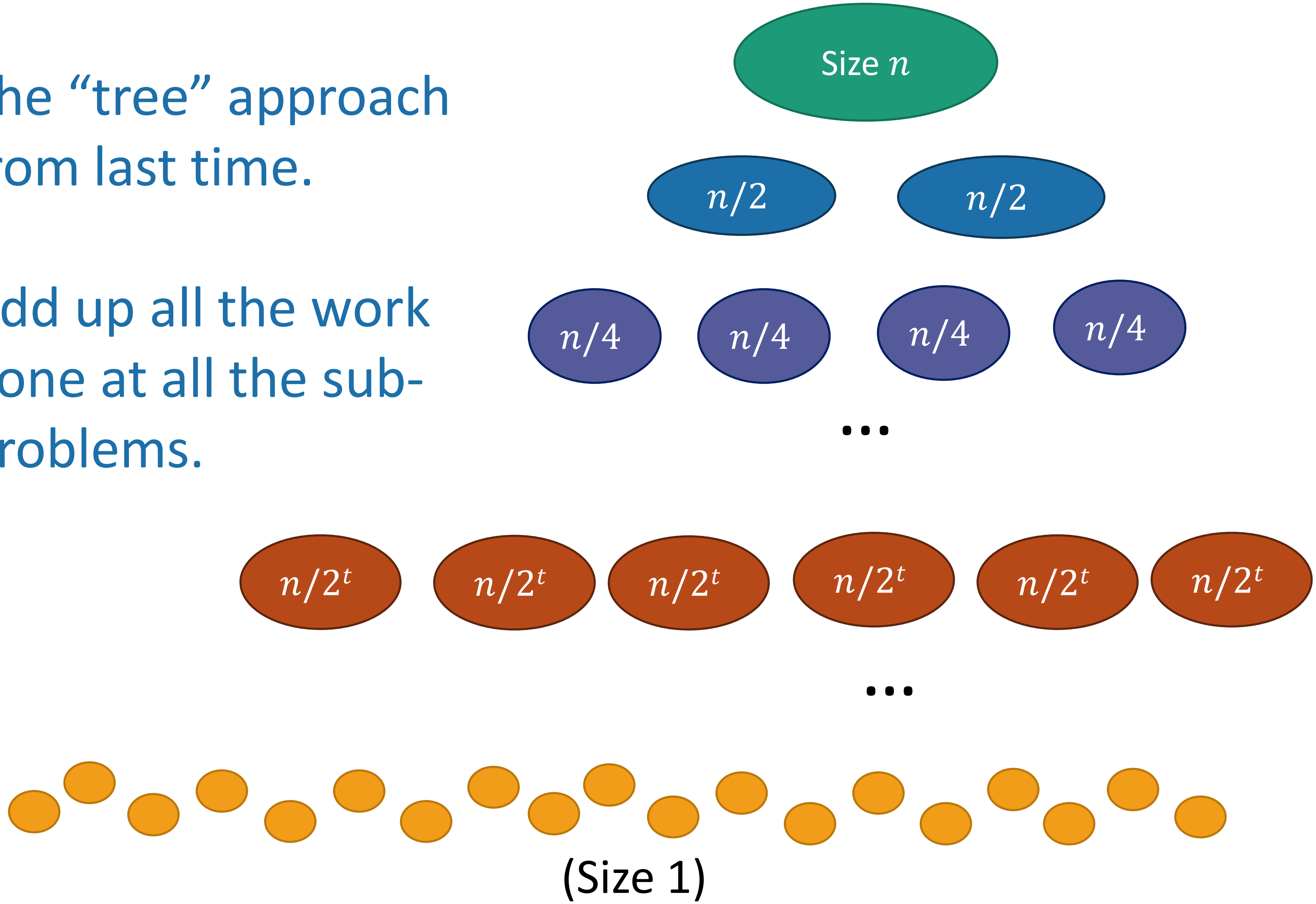
Siggi the Studious Stork

Some excersices

- Let's take a look at these examples (when n is a power of 2):
  1. $T(n) = T\left(\frac{n}{2}\right) + n,$ $\quad T(1) = 1$
  2. $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n,$ $\quad T(1) = 1$
  3. $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n,$ $\quad T(1) = 1$

One approach for all of these

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

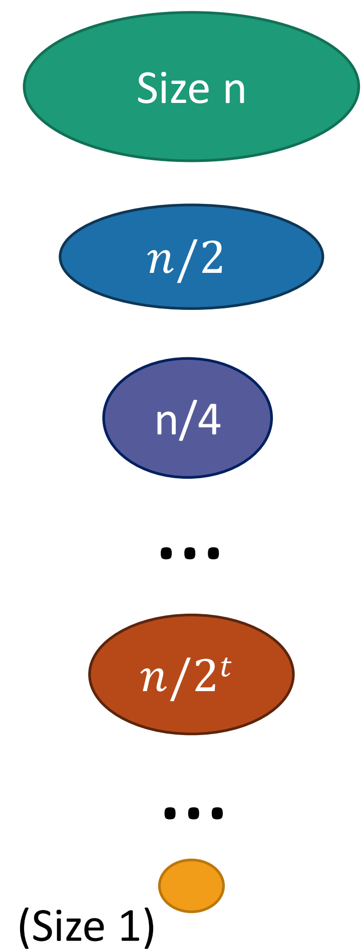


Solutions to exercise (1)

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

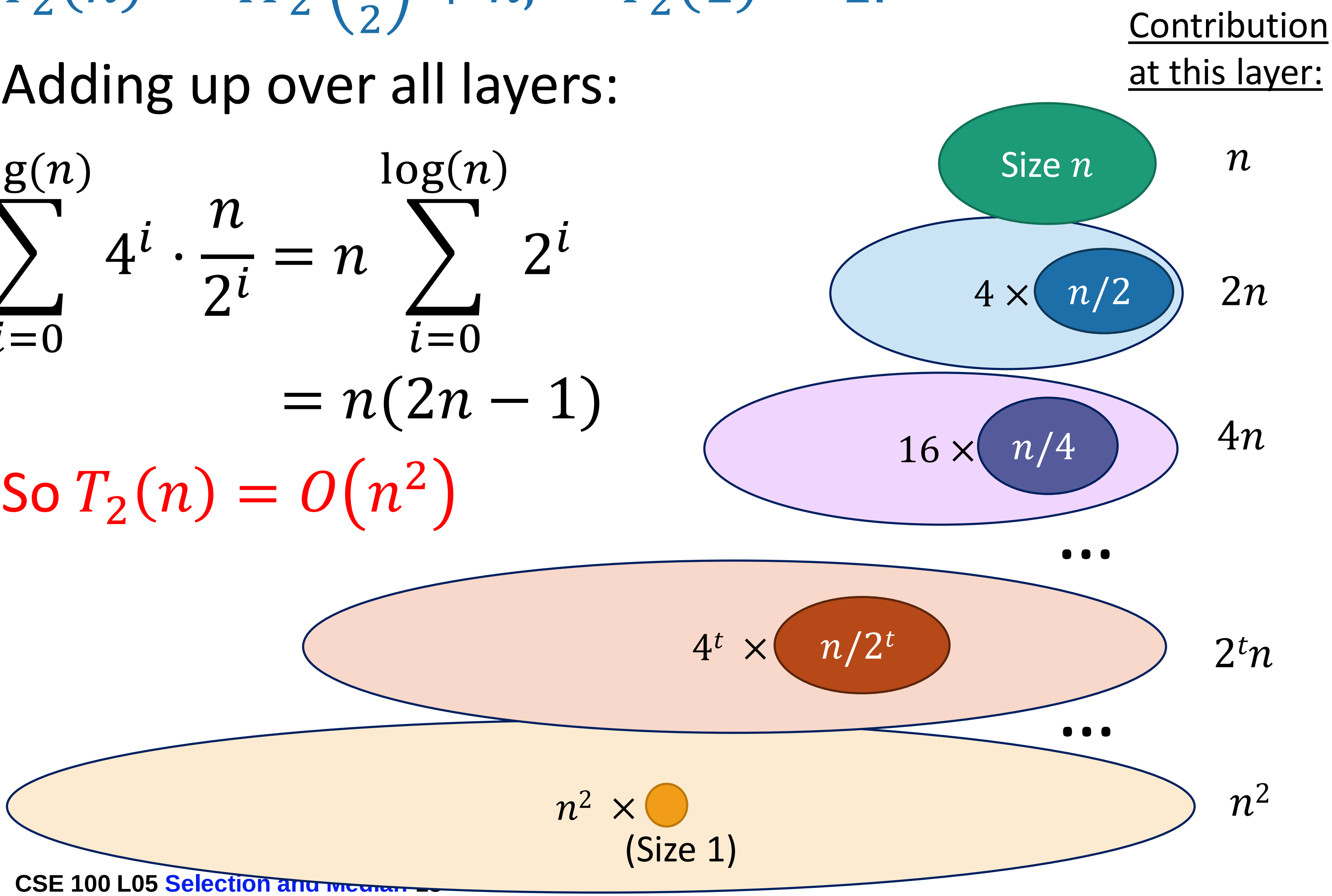
Solutions to exercise (2)

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

This is similar to T_2 from the example exercises.

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

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

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

The master theorem

- A formula that solves recurrences when all of the *sub-problems are the same size*.
  - We'll see an example later 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 $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

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

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

- 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

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

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

- Needlessly recursive integer mult.
  - $T(n) = 4T(n/2) + O(n)$
  - $T(n) = O(n^2)$

$a = 4$
$b = 2$
$d = 1$

$a > b^d$ ✓

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

$a = 3$
$b = 2$
$d = 1$

$a > b^d$ ✓

- 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 that the extra work at each level was $O(n^d)$. That’s NOT the same as work $\leq cn^d$ for some constant c.

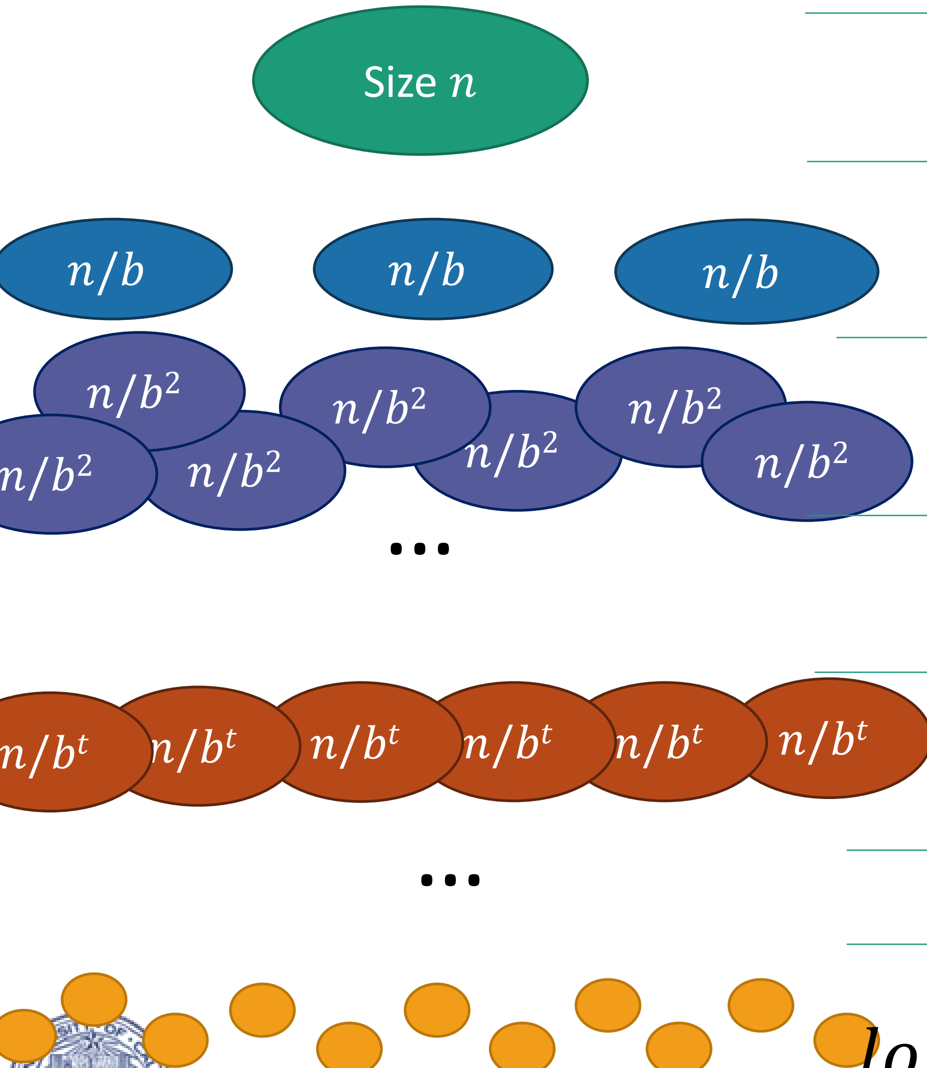
Plucky the Pedantic Penguin

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

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

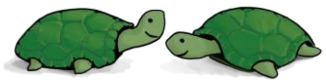


(Size 1)

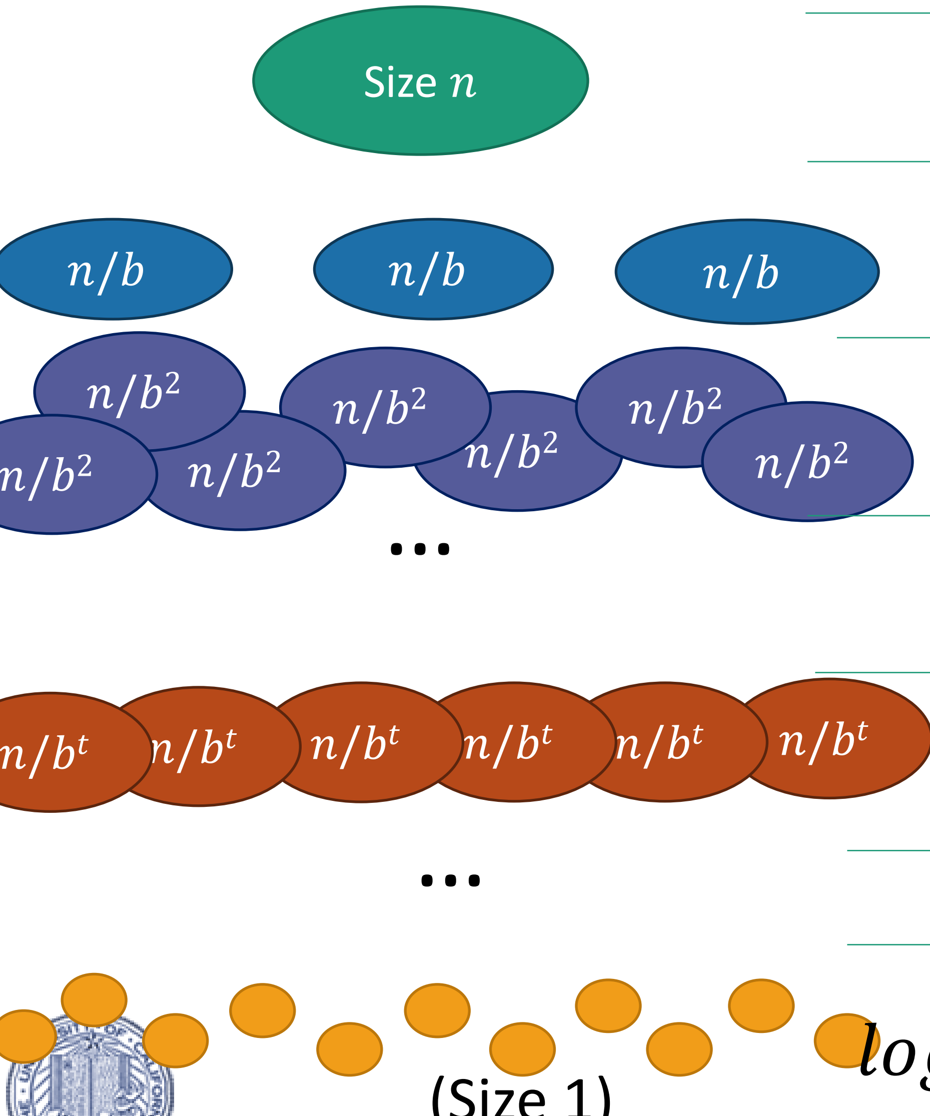
Level	# problems	Size of each problem	Amount of work at this level
0	1	n	
1	a	n/b	
2	a^2	n/b^2	
t	a^t	n/b^t	
…			
$log_b(n)$	$a^{\log_b(n)}$	1	

Recursion tree

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



Help me fill this in!

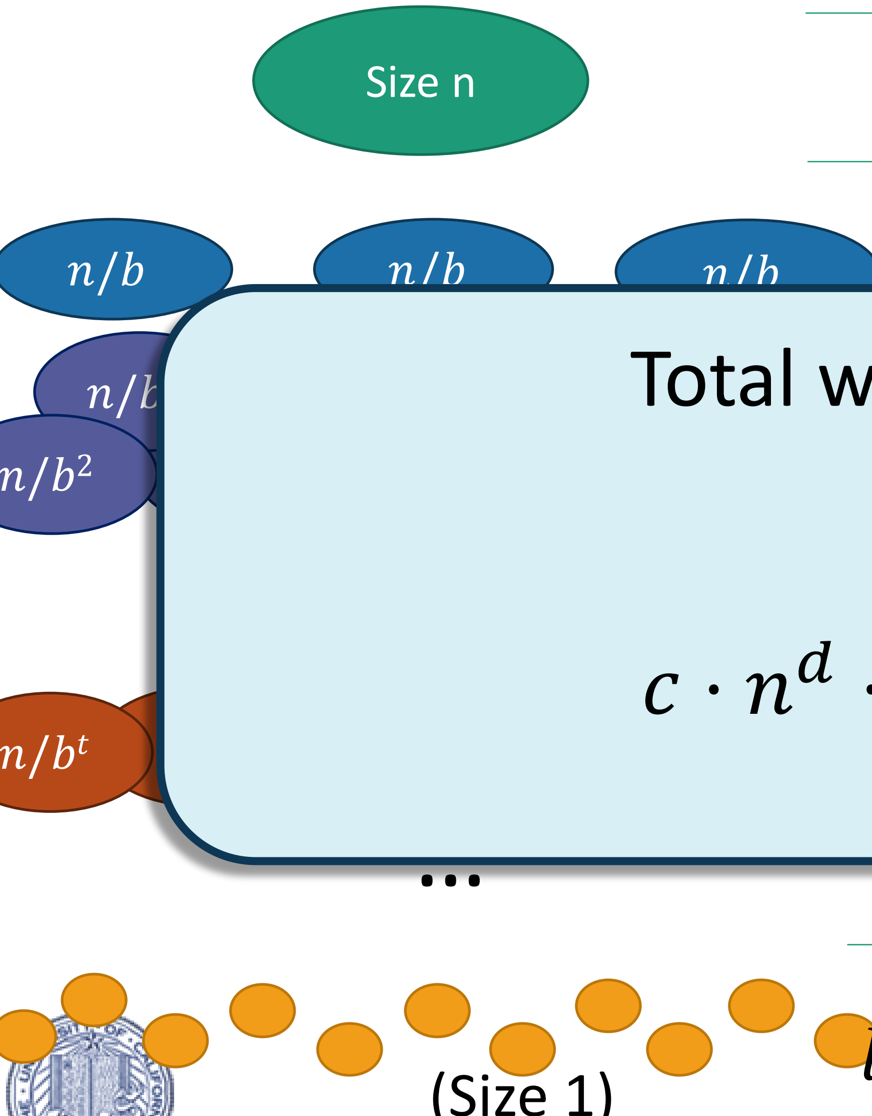


Level	# problems	Size of each problem	Amount of work at this level
0	1	n	$c \cdot n^d$
1	a	n/b	$ac\left(\frac{n}{b}\right)^d$
2	a^2	n/b^2	$a^2c\left(\frac{n}{b^2}\right)^d$
t	a^t	n/b^t	$a^tc\left(\frac{n}{b^t}\right)^d$
...			
$log_b(n)$	$a^{\log_b(n)}$	1	$a^{\log_b(n)}c$

(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	$ac\left(\frac{n}{b}\right)^d$
...			
$log_b(n)$	$a^{\log_b(n)}$	1	$a^{\log_b(n)}c$

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

Total work is at most:

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