

CS 170

Efficient Algorithms and Intractable Problems

Lecture 2: Divide and Conquer I, Asymptotics

Nika Haghtalab and John Wright

EECS, UC Berkeley

Announcements



- Office hours starts Tuesday (after class). Check the calendar.
- Discussions start this week, first one is on Wednesday
 - Leetcode discussion section. Check the calendar
- Homework 1 is out, Due on Saturday.
 - Homeworks are graded for feedback, not credit.
- Class quizzes will be drawn from homeworks and discussions. So do the homework!

Recap of last time

Introductions all around!

Our motivating questions about algorithms:

- Does it work?
- Is it fast?
- Can I do better?

Technical content:

- Arithmetic and Big Oh notation
- Intro to Divide and Conquer
- First attempt at fast multiplication
→ Still didn't beat $O(n^2)$
- We saw Karatsuba's trick

Recap of last time

Introductions all are

Our motivating ques

- Does it work?
- Is it fast?
- Can I do better

Technical content:

- Arithmetic and Big
 - Intro to Divide and C
 - First attempt at fas
- Still didn't beat O

(simplify: assume even n)

The algorithm

Break up the multiplication of two integers with n digits into multiplication of integers with $n/2$ digits:

$$[x_1 x_2 \cdots x_n] = \overbrace{[x_1, x_2, \cdots, x_{n/2}]}^a \times 10^{\frac{n}{2}} + \overbrace{[x_{n/2+1} x_{n/2+2} \cdots x_n]}^b$$

$$\begin{aligned} x \times y &= \left(a \times 10^{\frac{n}{2}} + b \right) \left(c \times 10^{\frac{n}{2}} + d \right) \\ &= \underbrace{(a \times c)}_{P1} 10^n + \underbrace{(a \times d)}_{P2} + \underbrace{(c \times b)}_{P3} 10^{n/2} + \underbrace{(b \times d)}_{P4} \end{aligned}$$

One n -digit multiplication



Four $n/2$ -digit multiplications

Recap of last time

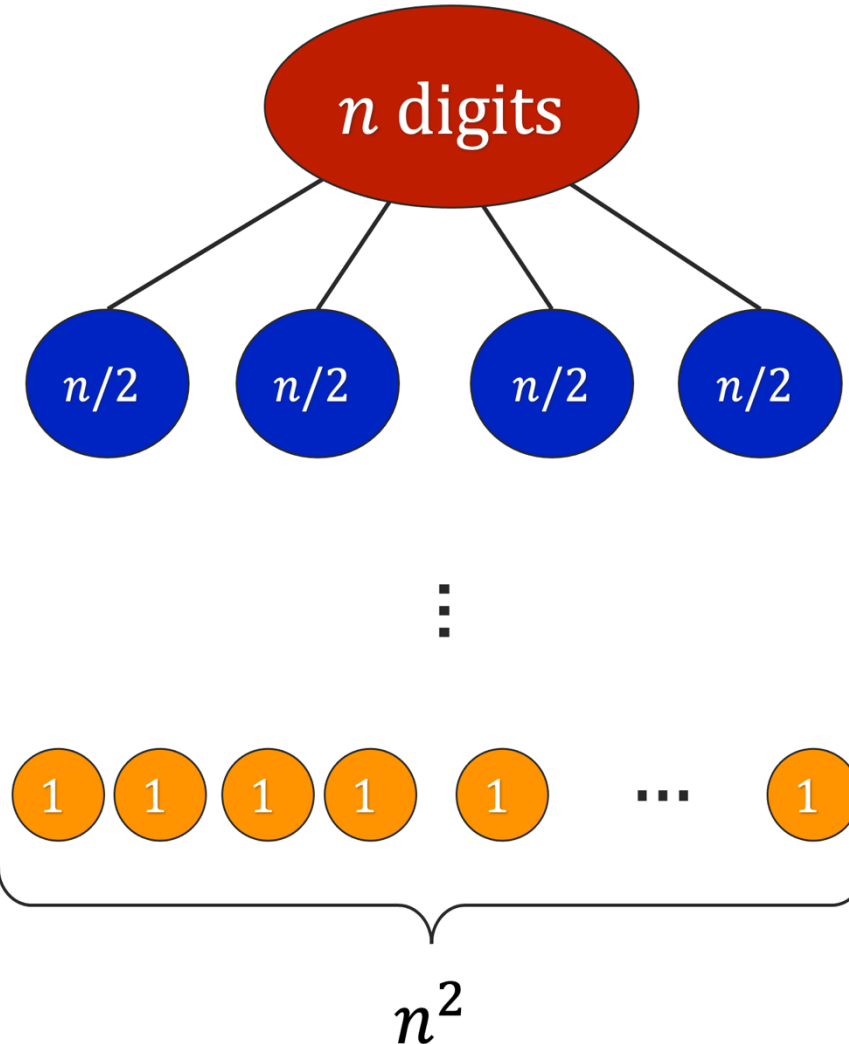
Introductions all are

Our motivating ques

- Does it work?
- Is it fast?
- Can I do better

Technical content:

- Arithmetic and Big
 - Intro to Divide an
 - First attempt at fas
- Still didn't beat O



Layer	# of digits	# problems
0	n	1
1	$n/2$	4
$\vdots$	$\vdots$	$\vdots$
t	$\frac{n}{2^t}$	4^t
$\vdots$	$\vdots$	$\vdots$
$\log_2(n)$	1	$4^{\log_2 n} = n^2$

This lecture

- More details on Karatsuba's algorithm with $O(n^{1.6})$
→ Using divide and conquer, but this time better!
- Reviewing $O(\cdot)$ and $\Omega(\cdot)$ notation formally.
- Recurrence relations and a useful theorem for solving them!

Karatsuba's Idea

Divide and Conquer indeed can lead to a faster algorithm!

$$\begin{aligned}x \times y &= (a \times 10^{\frac{n}{2}} + b)(c \times 10^{\frac{n}{2}} + d) \\&= \underbrace{(a \times c)}_{P1} 10^n + \underbrace{(a \times d)}_{P2} + \underbrace{(c \times b)}_{P3} 10^{n/2} + \underbrace{(b \times d)}_{P4}\end{aligned}$$

The issue is that we are creating 4 sub-problems.
What if we could create fewer subproblems?



Main idea: Could we write $P2+P3$ using what we compute in $P1$ and $P4$, and at most one other $n/2$ -digit multiplication?

Karatsuba's Clever Trick

Let us only compute 3 multiplications with $n/2$ digit numbers:

- Q1: $a \times c$
- Q2: $b \times d$
- Q3: $(a + b)(c + d)$

Expressing P2+P3 differently

$$ad + cb = (a + b)(c + d) - ac - bd$$

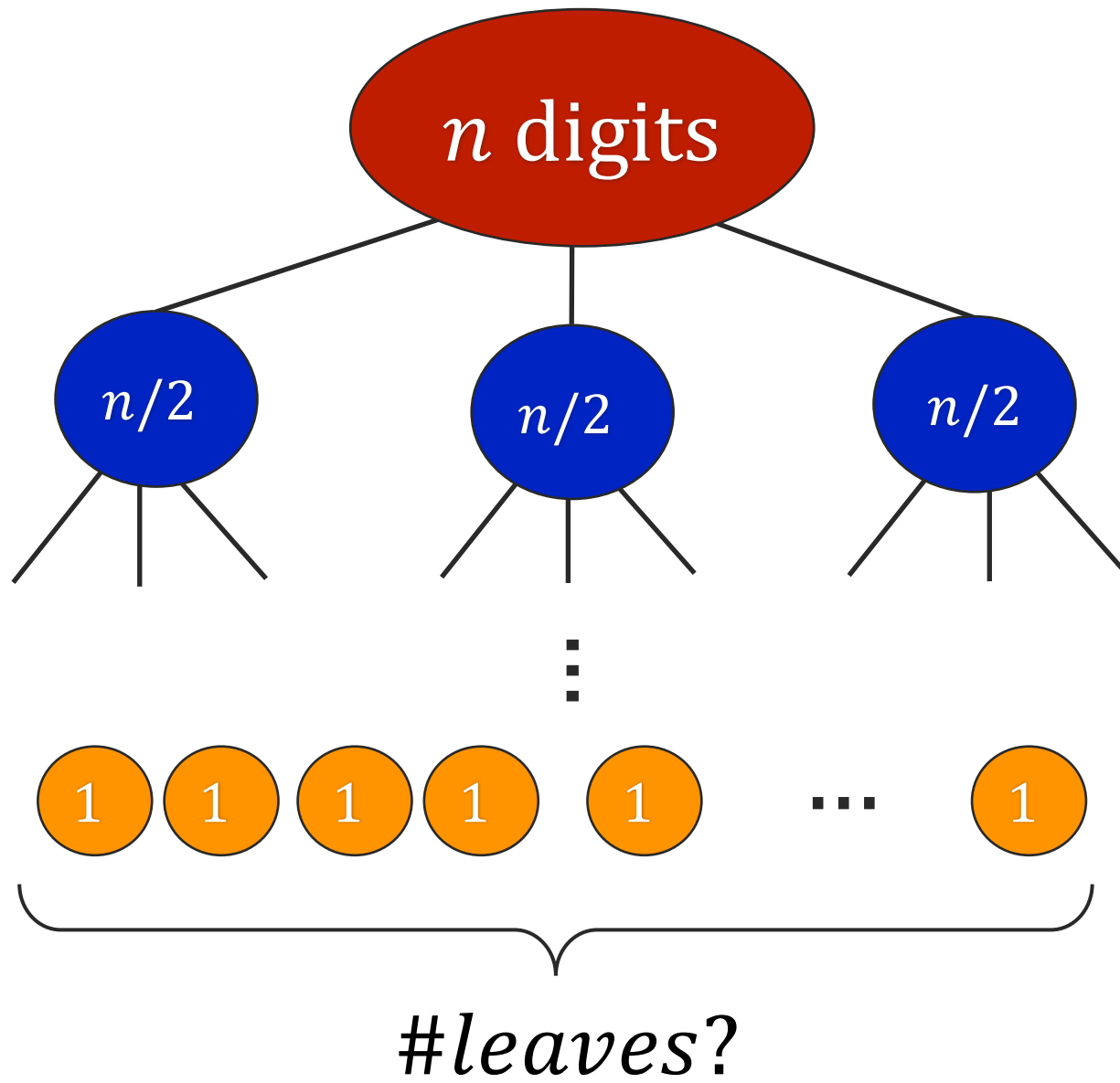
Three subproblems

$$\begin{aligned} x \times y &= (a \times 10^{\frac{n}{2}} + b)(c \times 10^{\frac{n}{2}} + d) \\ &= \underbrace{(a \times c)}_{Q1} 10^n + \underbrace{(a \times d + c \times b)}_{Q3-Q1-Q2} 10^{n/2} + \underbrace{(b \times d)}_{Q2} \end{aligned}$$

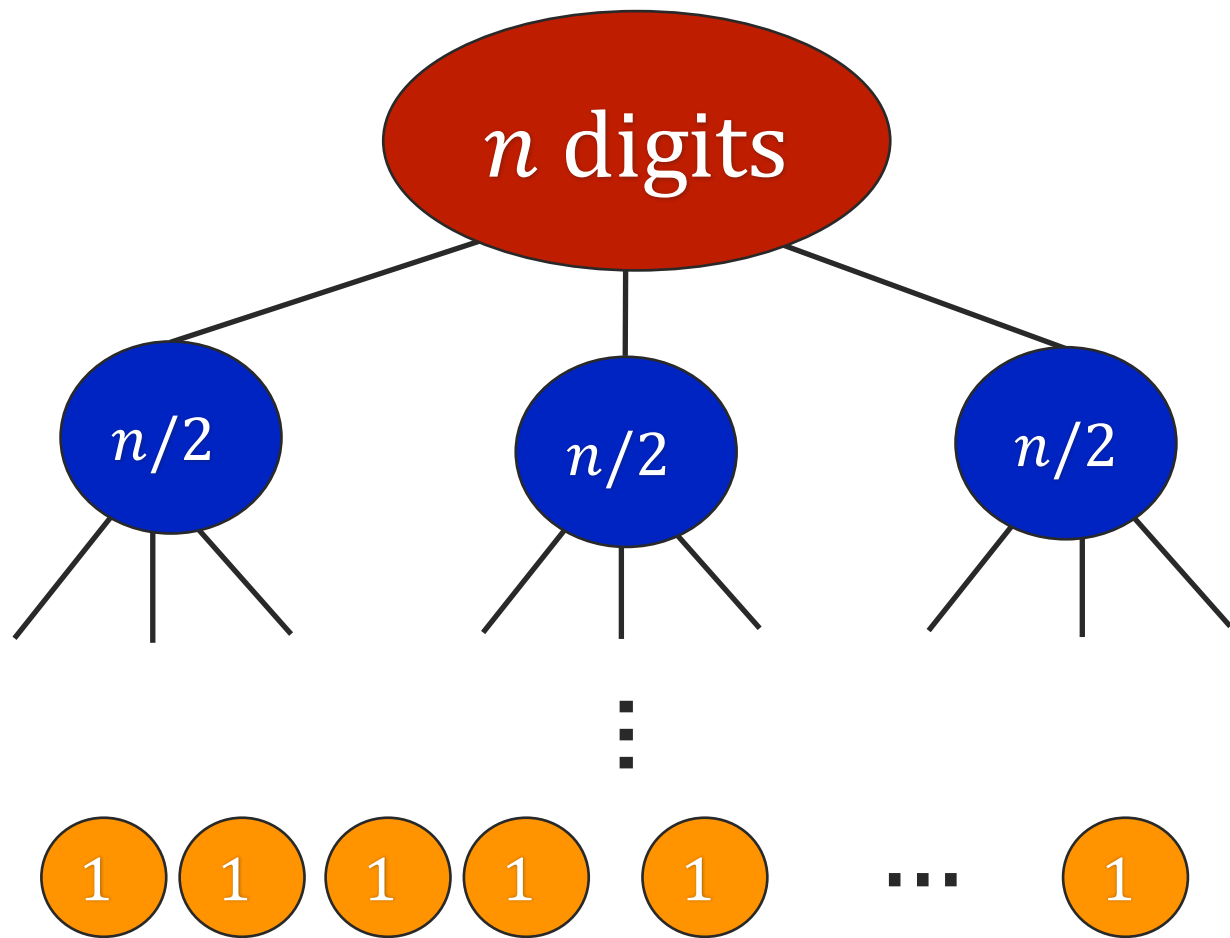
What is the runtime of Karatsuba's algorithm?

Less formally, how many 1-digit multiplications do we do in Karatsuba's algorithm?

Same approach as last lecture,
this time our branching factor is 3 instead of 4



Layer	# of digits	# problems
0	n	1
1	$n/2$	3
$\vdots$	$\vdots$	$\vdots$
t	#digits per node	# nodes
$\vdots$	$\vdots$	$\vdots$
Depth	1	# leaves



Layer	# of digits	# problems
0	n	1
1	$n/2$	3
$\vdots$	$\vdots$	$\vdots$
t		
$\vdots$	$\vdots$	$\vdots$
	1	

Other Algorithms

- **Karatsuba** (1960): $O(n^{1.6})$! **Saw this!**
- **Toom-3/Toom-Cook** (1963): $O(n^{1.465})$



(advanced)

Divide and conquer too! Instead of breaking into three $n/2$ -sized problems, break into five $n/3$ -sized problems.

Hint: Start with 9 subproblems and reduce it to 5 subproblems.

- **Schönhage–Strassen** (1971):
 - Runs in time $O(n \log(n) \log \log(n))$
- **Furer** (2007)
 - Runs in time $n \log(n) \cdot 2^{O(\log^*(n))}$
- **Harvey and van der Hoeven** (2019)
 - Runs in time $O(n \log(n))$

What about binary representation?

We used base 10 so far

→ Counted the # of 1-digit operations, assuming adding/multiplying single digits is easy (memorized our multiplication table!)

What if we use base 2?

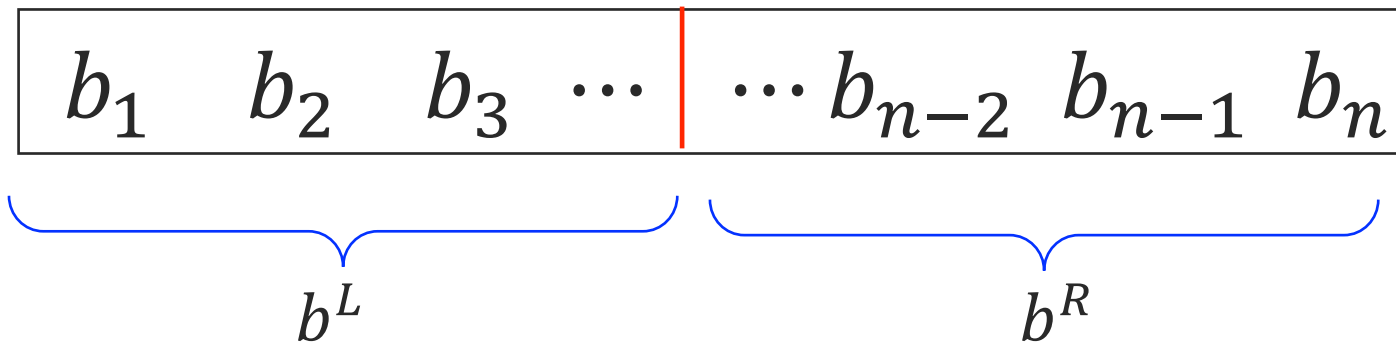
→ We would want to count # of **1-bit** operations.

How do we alter Karatsuba's algorithm for binary numbers?

N-bit integer multiplications

Easy to compute 10^k in base 10. In base 2, it is easy to compute 2^k .

$$[b_1 b_2 \cdots b_n] = [b_1, b_2, \cdots, b_{n/2}] \times 2^{n/2} + [b_{n/2+1} b_{n/2+2} \cdots b_n]$$



$$\begin{aligned} a \times b &= (a^L \times b^L) 2^n + (a^L \times b^R + a^R \times b^L) 2^{n/2} + (a^R \times b^R) \\ &= \dots \end{aligned}$$

Practice: Complete this equation the Karatsuba's way and rederived $O(n^{1.6})$ runtime for multiplying two n -bit numbers. Might see this on HW or Discussion!

Details we skipped

Technically

- We only counted the number of 1-digit problems
- There are other things we do: adding, subtracting, ...
- Shouldn't we account for all of that?

Absolutely!

- We should be more formal, and we will be next!
- In this case, additions/subtractions end up in lower order terms
- Don't affect $O(\cdot)$.

Asymptotic Notations

More Formally

Runtime of Algorithms Asymptotically

Suppose an algorithm with input size n takes

$$T(n) = 5n^2 + 20n \log(n) + 7 \quad \text{ms}$$

$T(n) \in O(n^2)$ also commonly written as $T(n) = O(n^2)$

Why is it a good idea to just say this is $O(n^2)$?

- Constants like 5, 20, 7, depend on the platform and computer.
- Makes it easier to compare the performance of algorithms on large inputs
- Makes algorithm analysis easier
- Sometime clever tricks and representations improve the constants anyway.

Definition of $O(\dots)$

- Let $T(n)$, $g(n)$ be functions of positive integers.
 - Think of $T(n)$ as a runtime: positive and increasing in n .
- We say “ $T(n)$ is $O(g(n))$ ” if and only if
 - for large enough n ,
 - $T(n)$ is *at most* some constant multiple of $g(n)$.

Definition of $O(\dots)$

- Let $T(n)$, $g(n)$ be functions of positive integers.
 - Think of $T(n)$ as a runtime: positive and increasing in n .

- We say “ $T(n)$ is $O(g(n))$ ” if and only if

There exists c and $n_0 > 0$
Such that for all $n \geq n_0$, $T(n) \leq c \cdot g(n)$

- Always give the tightest and simplest $O(\)$ you can.
 - e.g., $5n^2 \in O(n^3)$ and $5n^2 \in O(2n^2 + n)$ too,
 - but give the best bound of $O(n^2)$.

Example

Prove that for $T(n) = 2n^2 + 2$, we $T(n) \in O(n^2)$

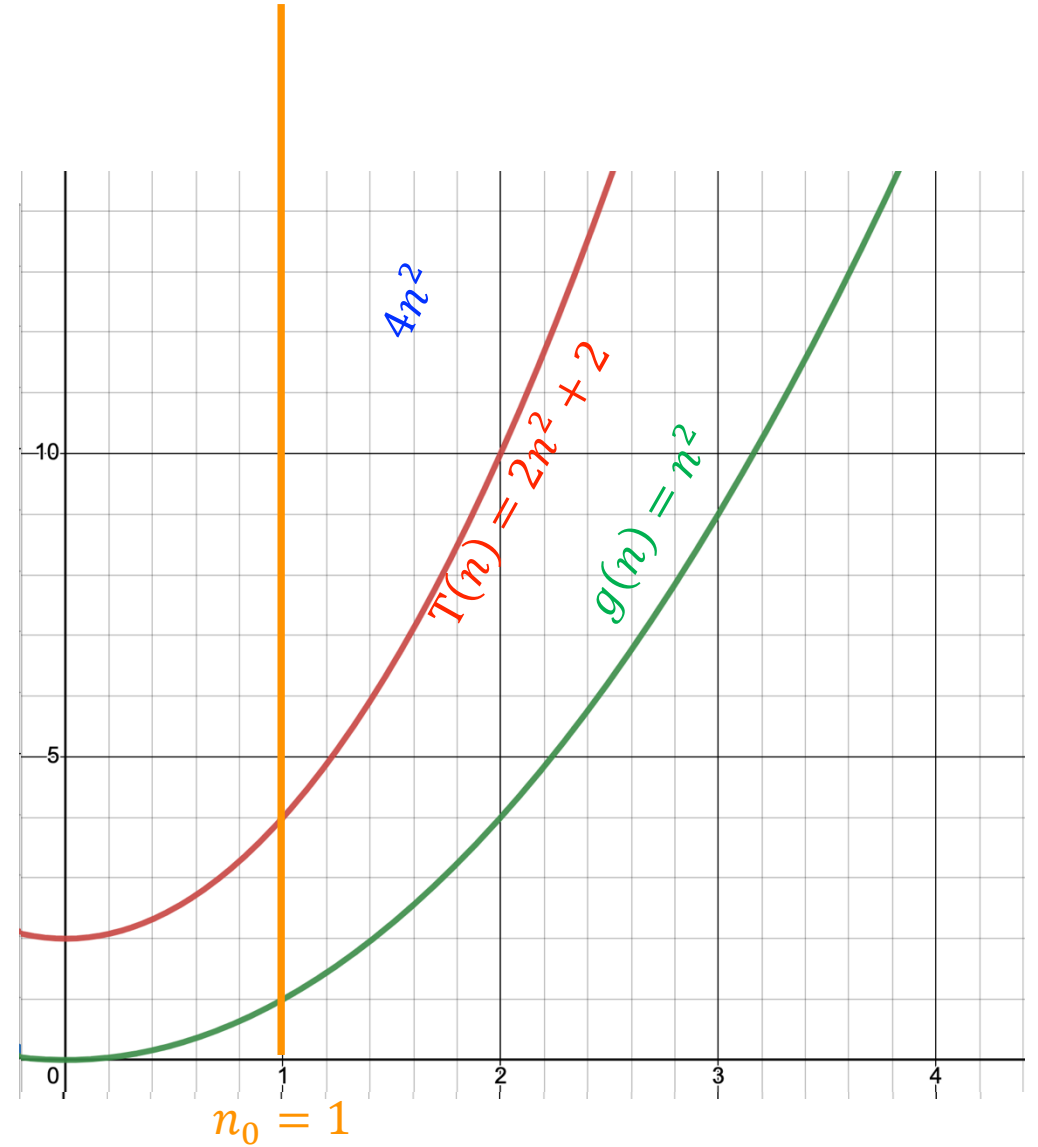
Even though $T(n)$ is larger than n^2 always, we can find $c = 4$ and $n_0 = 1$, such that all $n > n_0$

$$2n^2 + 2 \leq 4n^2$$

How do you prove the above inequality?

Whatever (correct!) math you like!

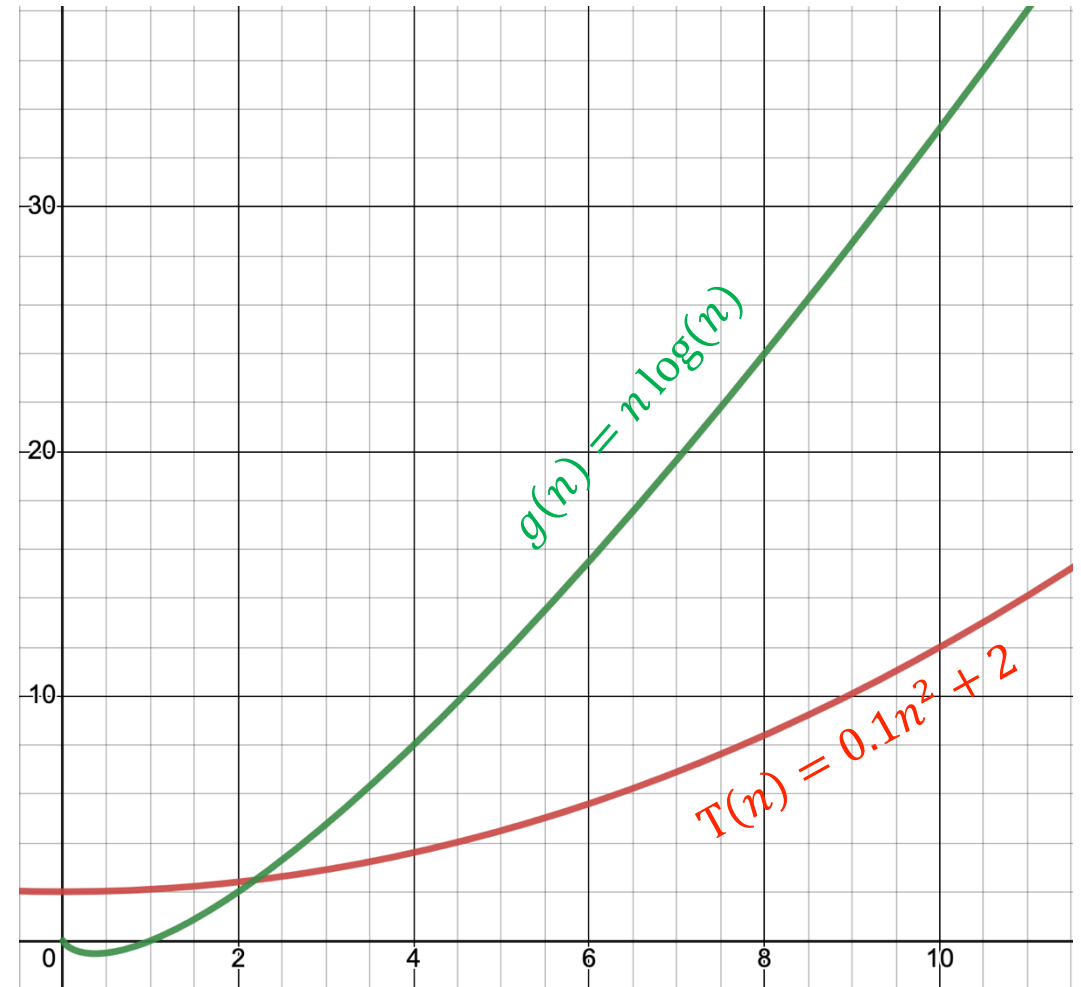
- E.g., equal at n_0 and RHS has larger derivative.



BEWARE! of pictures!

The picture seems to imply that for $T(n) = 0.1n^2 + 2$ we have that $T(n) \in O(n \log(n))$!

What's wrong with this argument and relying on pictures?

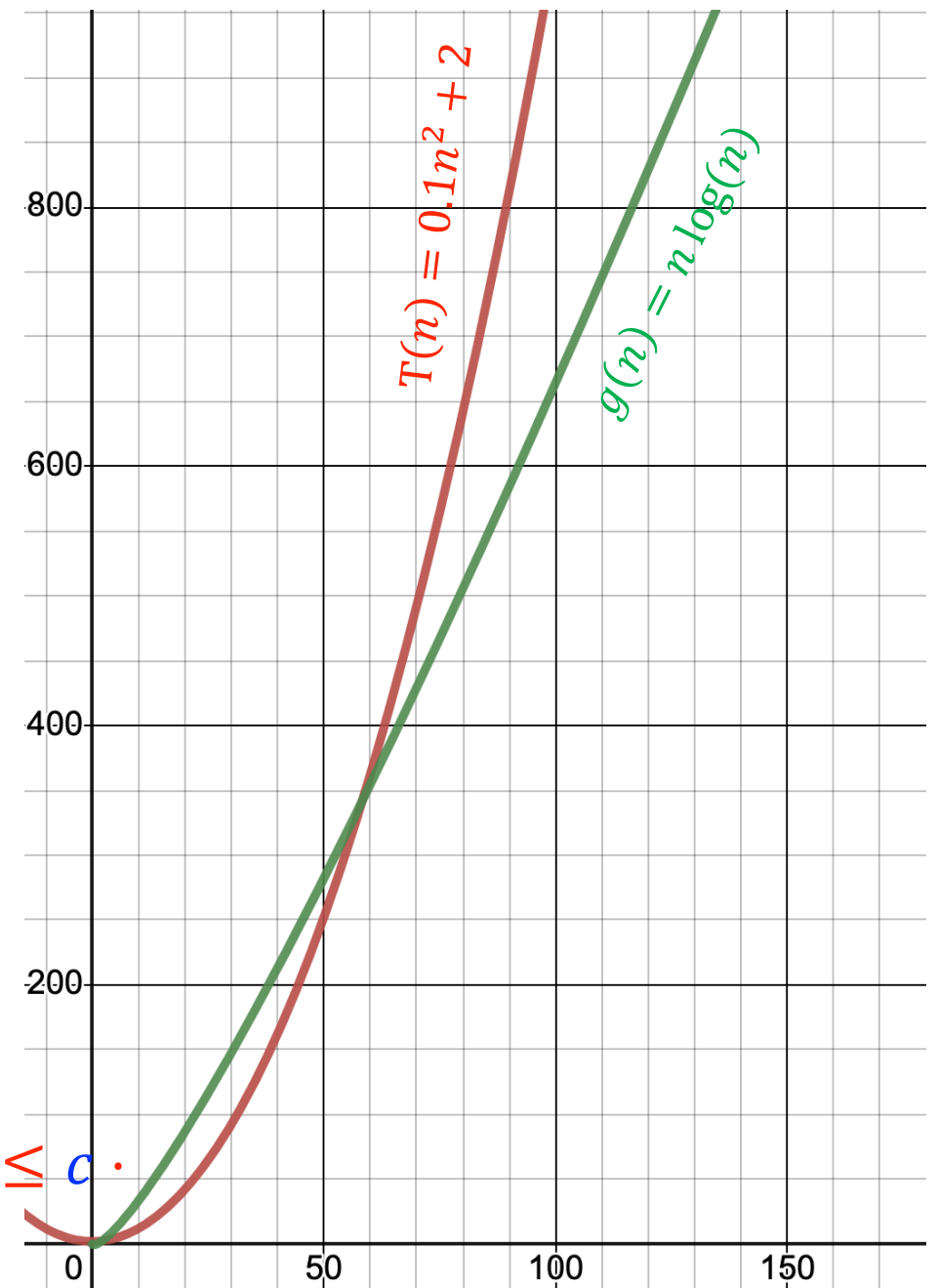


BEWARE! of pictures!

The picture seems to imply that for $T(n) = 0.1n^2 + 2$ we have that $T(n) \in O(n \log(n))$!

What's wrong with this argument and relying on pictures?

That is why you should come up with c and n_0 and mathematically prove that for all $n \geq n_0$, $T(n) \leq c \cdot g(n)$.



How to prove $0.1n^2 \notin O(n)$!

- Proof by contradiction:
- Suppose that $0.1n^2 \in O(n)$.
- Then there is some positive c and n_0 so that:

$$\forall n \geq n_0, \quad 0.1n^2 \leq c n$$

- Divide both sides by n :

$$\forall n \geq n_0, \quad 0.1n \leq c$$

- That's not correct. Let $n = n_0 + 10c$
 - Then $n \geq n_0$, but $0.1n > c$.
- Contradiction!

Recap of Proof Techniques

To prove $T(n) \in O(g(n))$:

- You have to come up with c and n_0 so that the definition is satisfied.

To prove $T(n) \notin O(g(n))$

- You have to rule out **all possible** c and n_0 .
- One approach is to use **proof by contradiction**:
 - Suppose there exists a c and an n_0 so that the definition *is* satisfied.
 - Derive a contradiction,
→ e.g., by finding **large enough** n (as a function of c and n_0), for which the definition is not satisfied.

$\Omega(\dots)$ means lower bound

- Let $T(n)$, $g(n)$ be functions of positive integers.
 - Think of $T(n)$ as a runtime: positive and increasing in n .
- We say “ $T(n) \in \Omega(g(n))$ ” if and only if
 - There exists c and $n_0 > 0$
 - Such that for all $n \geq n_0$, $c \cdot g(n) \leq T(n)$


Switched these
compared to $O()$!!

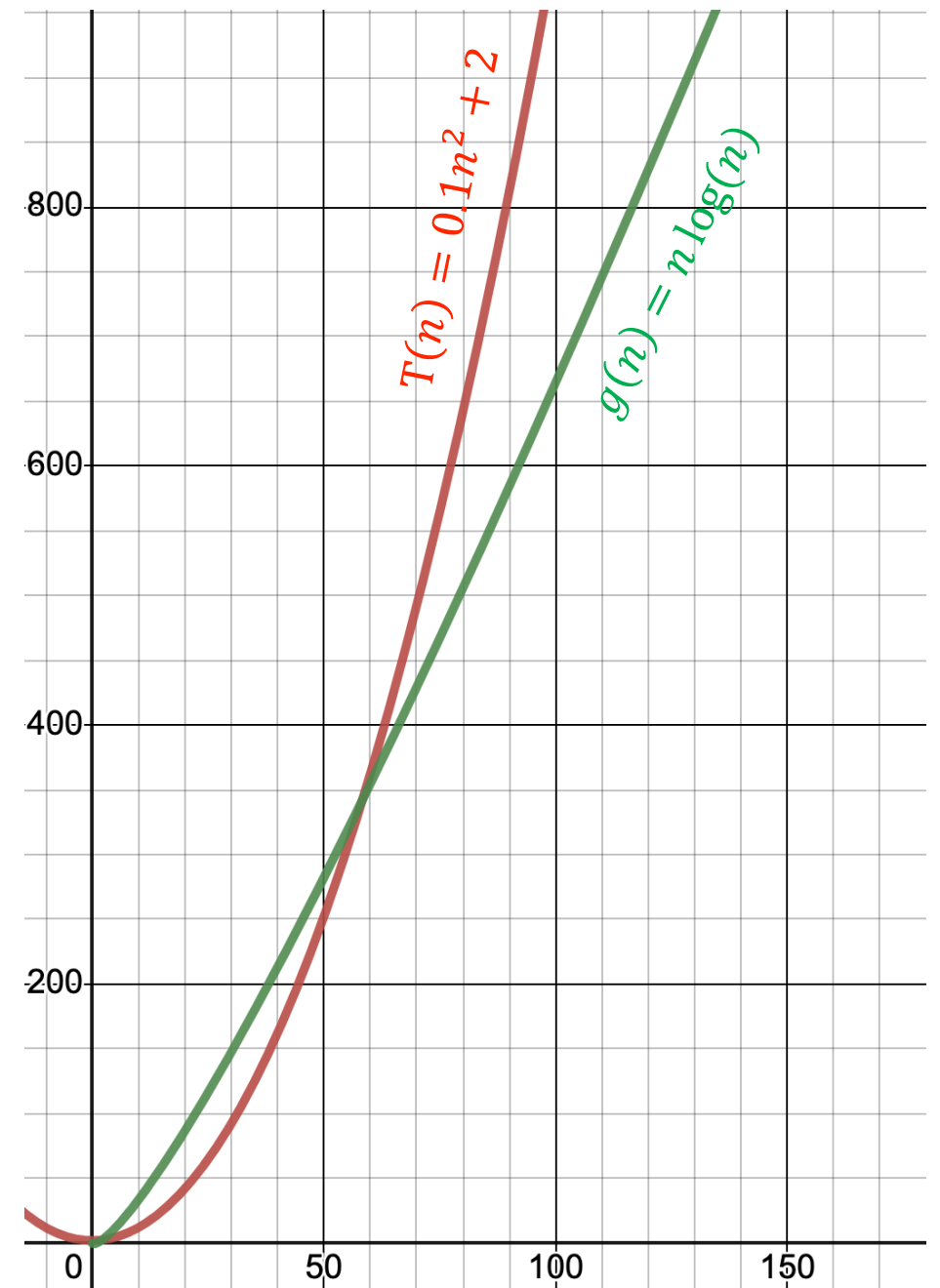
Example

Indeed, $0.1n^2 + 2 \in \Omega(n \log(n))$!



Prove this formally:

Find constants c and $n_0 > 0$, such that
for all $n \geq n_0$, $c \, n \log(n) \leq 0.1n^2 + 2$.



$\Theta(\dots)$ means both!

We say “ $T(n)$ is $\Theta(g(n))$ ” iff both:

$$T(n) = O(g(n))$$

and

$$T(n) = \Omega(g(n))$$

Example: Asymptotics of the geometric series

Take any constant r and function $T(n) = 1 + r + r^2 + \dots + r^n$

Show that $T(n) = \begin{cases} \Theta(r^n) & \text{if } r > 1 \\ \Theta(1) & \text{if } r < 1 \\ \Theta(n) & \text{if } r = 1 \end{cases}$



Prove formally
at home!

Proof Idea: Recall sum of a geometric series that for $r \neq 1$:

$$1 + r + r^2 + \dots + r^n = \frac{r^{n+1} - 1}{r - 1}$$

Intuition:

- For $r > 1$, this is approximately $\frac{r^{n+1}}{r} = r^n$.
- For $r < 1$, $\frac{r^{n+1}-1}{r-1} \approx \frac{1}{1-r}$

Prove formally at home (also EX 0.2 of the book).

Revisiting Karatsuba's Alg runtime, more formally

What is the runtime of Karatsuba's Alg?

At each layer, we have 3 problems
→ Each problem of size $\frac{n}{2}$.

Karatsuba's Alg in 1 layer

$$\begin{aligned} Q1 &= a \times c & Q2 &= b \times d & Q3 &= (a + b)(c + d) \\ x \times y &= Q1 \times 10^n + (Q3 - Q1 - Q2)10^{n/2} + Q2 \end{aligned}$$

We have to do a bunch of other operations

- Finding a, b, c, d by shifting n -digit arrays.
- $n/2$ -digit additions $a + b, c + d$
- n -digit additions $Q3 - Q1 - Q2$
- $2n$ -digit additions $Q1 \times 10^n + (Q3 - Q1 - Q2)10^{n/2} + Q2$

$O(n)$

More precisely $\leq 20n$

Runtime :

$$T(n) = \underbrace{3 T\left(\frac{n}{2}\right)}_{\text{Recursive calls to the next layer}} + \underbrace{20n}_{\text{Work in this layer}}$$

Recurrence Relations

Recurrence relations give a formula for $T(n)$, i.e., the runtime on size n problems in terms of $T(k)$ where $k < n$.

Recursive calls to
the next layer Work in this layer

$$T(n) = \underbrace{3 T\left(\frac{n}{2}\right)}_{\text{Recursive calls to the next layer}} + \underbrace{20n}_{\text{Work in this layer}} \text{ is a **recurrence relation**.}$$

$T(1) = O(1)$ Base case (e.g., $T(1) = 5$ or 500)

Main question:

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

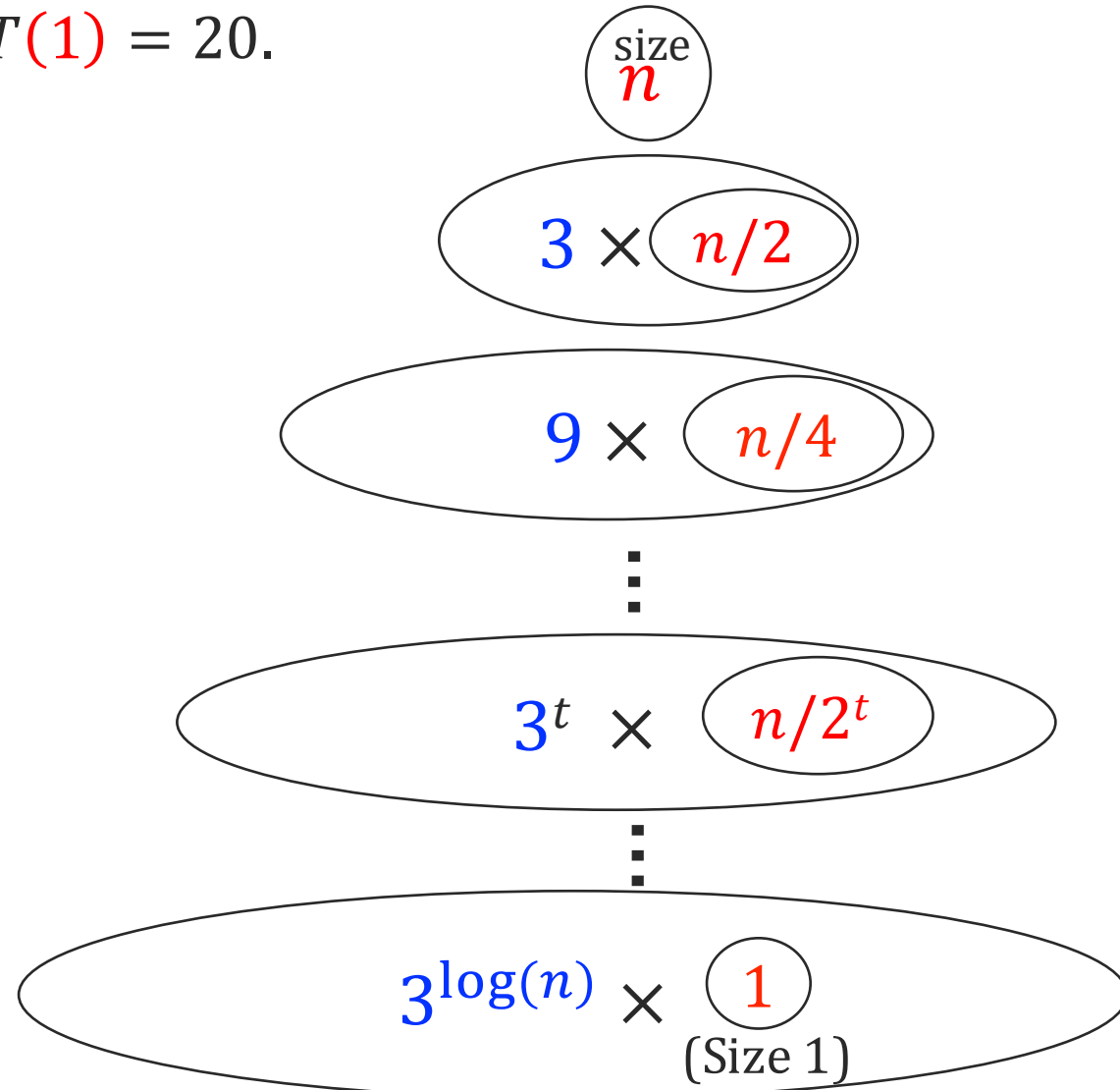
For example, we hope that $T(n) = O(n^{1.6})$ for the above recurrence!

Solve Karatuba's Alg Recurrence Relation

$$T(n) = 3T\left(\frac{n}{2}\right) + 20n, \quad T(1) = 20.$$

Abstraction of the tree

Total contribution
in this layer



Solving Recurrence Relations Generally

The tree method, as we just did

- Keep track of the number and size of problems in each step
- Account for total amount of computation done in each layer.
- Sum over all the computation done in the layers.

The Master Theorem

The tree method, as we just did

- Keep track of the number and size of problems in each step
- Account for total amount of computation done in each layer.
- Sum over all the computation done in the layers.

The Master Theorem

Suppose that $a \geq 1$, $b > 1$, and $d \geq 0$ are constants (independent of n).

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

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

More on the Master Theorem

- Can it be used to solve any recurrence relation?

→ Nope! But it is a useful tool in many cases.

→ So, make sure you are also comfortable with the tree method.

- Don't we need a base case?

→ Yes!

→ Take $T(1) = O(1)$, the exact constant in this case doesn't affect the $O(\cdot)$.

- What if n/b is not an integer?

→ The Master Theorem is also correct with $T(n) = a \cdot T\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$.

→ We will mostly **ignore floors and ceilings** in recurrence relations.

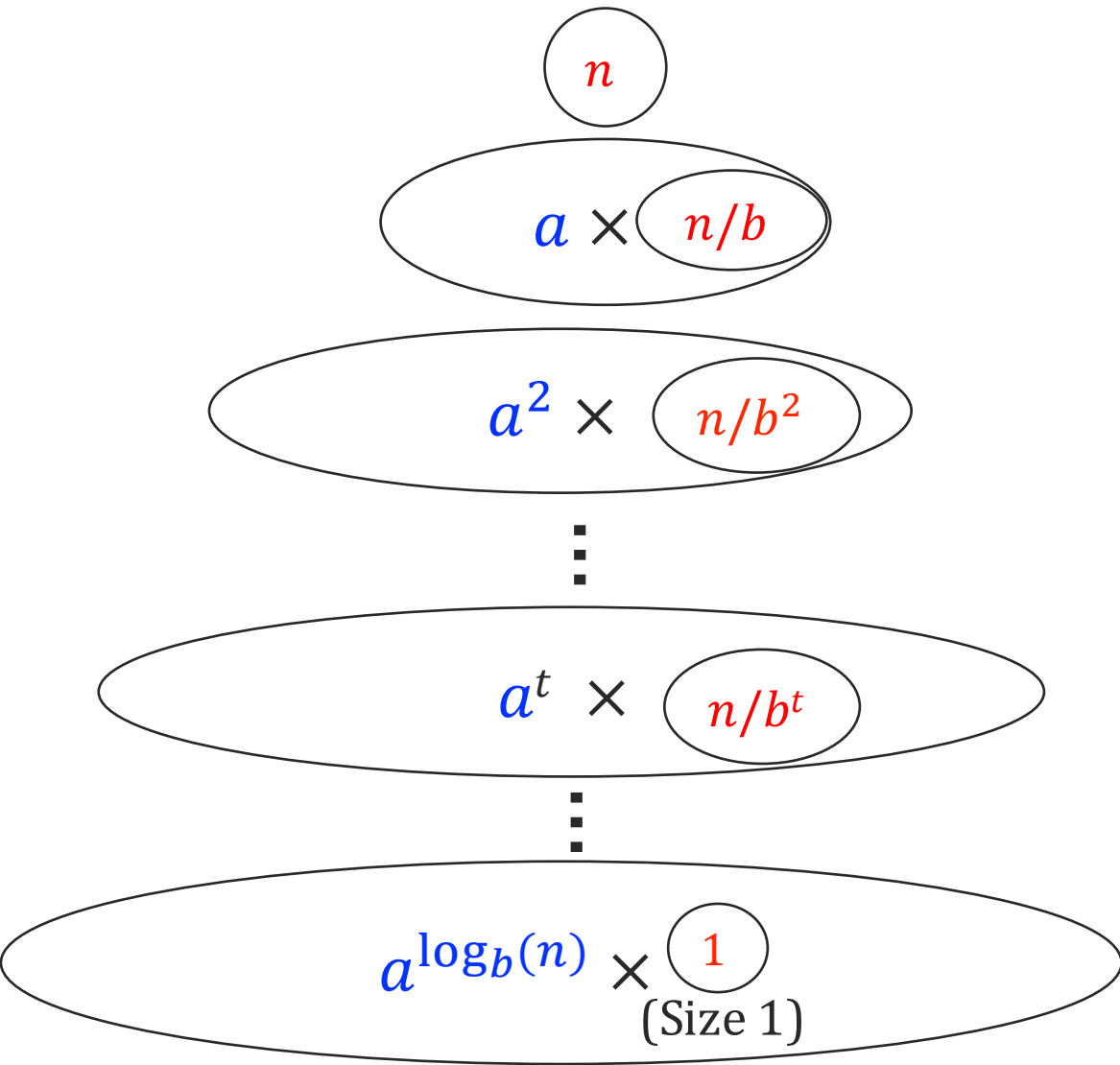
Overview of the proof of Master Theorem

- See Section 2.2 of the book for a complete proof.

For the proof, suppose that $T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$.

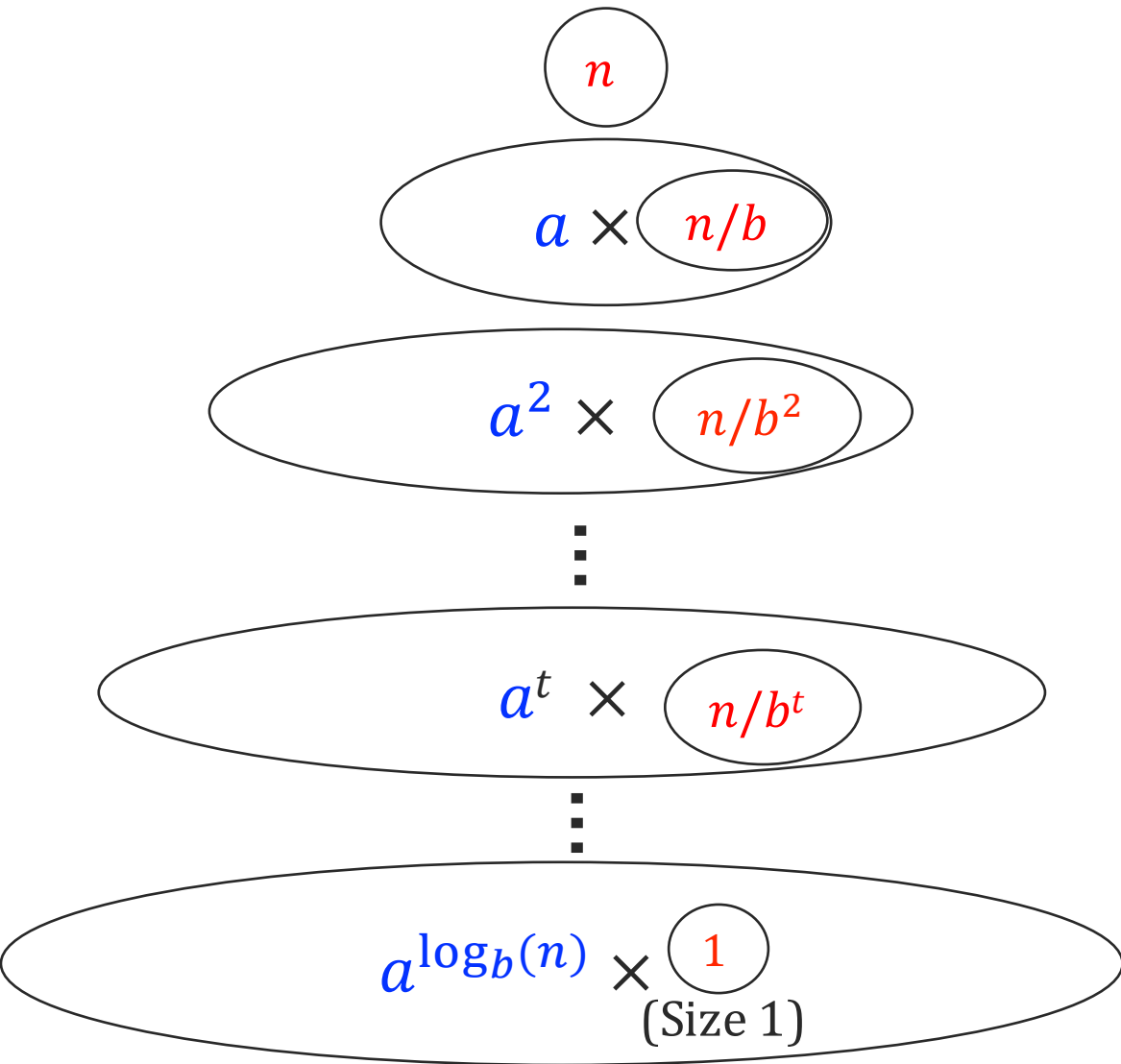
- For formal recursive arguments, we always substitute a constant.
 - Precise relationship between each layer's parameter and the amount of work.
 - Let's assume $T(1) = c$, too. For convenience!
 - Just do the tree method!

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



Layer	Problem size	# problems	Work @ this layer
0	n	1	
1	n/b	a	
$\vdots$	$\vdots$	$\vdots$	
t	n/b^t	a^t	
$\vdots$	$\vdots$	$\vdots$	
$\log_b(n)$	1	$a^{\log_b(n)}$	

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



Layer	Problem size	# problems	Work @ this layer
0	n	1	$c \cdot n^d$
1	n/b	a	$a \cdot c \cdot \left(\frac{n}{b}\right)^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
t	n/b^t	a^t	$a^t \cdot c \cdot \left(\frac{n}{b^t}\right)^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\log_b(n)$	1	$a^{\log_b(n)}$	$a^{\log_b(n)} \cdot c$

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

Total computation on all layers:

$$cn^d \cdot \underbrace{\sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t}_{\text{Looks so familiar ...}}$$

Looks so familiar ...

Layer	Problem size	# problems	Work @ this layer
0	n	1	$c \cdot n^d$
1	n/b	a	$a \cdot c \cdot \left(\frac{n}{b}\right)^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
t	n/b^t	a^t	$a^t \cdot c \cdot \left(\frac{n}{b^t}\right)^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\log_b(n)$	1	$a^{\log_b(n)}$	$a^{\log_b(n)} \cdot c$

Proof of the Master Theorem

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

Total computation on all layers:

$$cn^d \cdot \underbrace{\sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t}_{\text{Geometric series}}$$

$$1 + r + r^2 + \dots + r^n = \begin{cases} \Theta(r^n) & \text{if } r > 1 \\ \Theta(1) & \text{if } r < 1 \\ \Theta(n) & \text{if } r = 1 \end{cases}$$

The Master Theorem

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

Total computation on all layers:

$$cn^d \cdot \underbrace{\sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t}_{\text{Geometric series}}$$

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

$n^d \cdot n^{\log_b(\frac{a}{b^d})} = n^{d+\log_b(a)-\log_b(b^d)}$
 $= n^{d+\log_b(a)-d}$
 $= n^{\log_b(a)}$

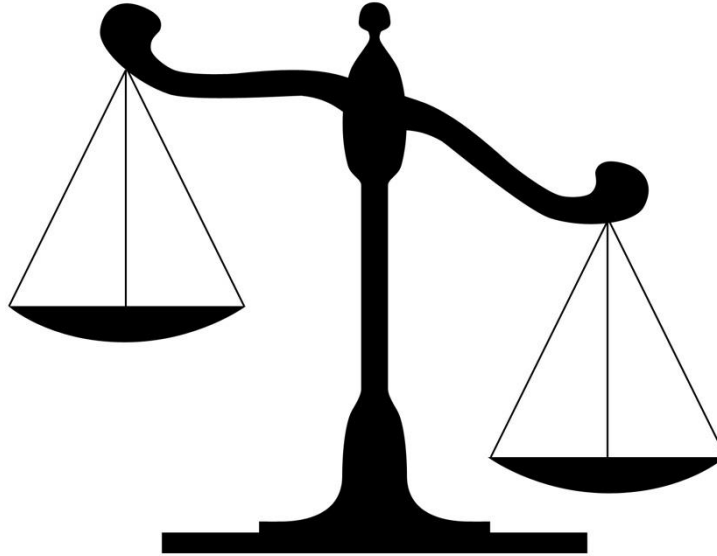
$$1 + r + r^2 + \dots + r^n = \begin{cases} \Theta(r^n) & \text{if } r > 1 \\ \Theta(1) & \text{if } r < 1 \\ \Theta(n) & \text{if } r = 1 \end{cases}$$

Master Theorem's Interpretation

$$a \quad \text{vs.} \quad b^d$$

Wide tree
 $a > b^d$

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



Tall and narrow
 $a < b^d$

Problem size shrinks fast,
so **most work is at the
top of the tree!**

$$a = b^d$$

Branching perfectly balances
total amount of work per layer.
All layers contribute equally.

Wrap up

Karatsuba Integer Multiplication:

You can do better than grade school multiplication!

Example of divide-and-conquer in action

Runtime analysis, informal and formal.

Asymptotics, recurrence relations, and Master theorem

Tree method is intuitive and fun!

Master theorem is useful!

Next time

- More divide and conquer
- Matrix multiplications
- Median selection