# CS 170
# Efficient Algorithms and Intractable Problems

## Lecture 2:
### Divide and Conquer I, Asymptotics

Nika Haghtalab and John Wright

EECS, UC Berkeley

# Announcements

**Discussion sections start today**! 🥳

- For coordination purposes, sign up to let us know which section you'll go to.
- You can still go to any other section, without the RSVP.
- Feeling you need a slower-paced section? Go to LOST section on Fridays.

Homework 1 will be posted today!

- Due ~~next Monday~~ (labor day!). Due on Tuesday

Homework party:

- This Friday and ~~next Monday~~ (labor day!)

# More announcements

Course Email:

- We said “it will be viewable by 2 instructors and 2 head TAs”

→ Revise this to “Also 2 graduate TAs with Admin role”.

## Other issues

Short break in the middle of the lecture?

Noise in the lecture hall?

Any other issues? Tell us on Ed!

# 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 $O(n^2)$

# Recap of last time

Introductions all aro

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 $O$

## The algorithm

(simplify: assume even $n$)

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

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

$$x \times y = \left(a \times 10^{\frac{n}{2}} + b\right)\left(c \times 10^{\frac{n}{2}} + d\right)$$

$$= \underbrace{(a \times c)}_{\text{P1}} 10^n + (\underbrace{a \times d}_{\text{P2}} + \underbrace{c \times b}_{\text{P3}}) 10^{n/2} + \underbrace{(b \times d)}_{\text{P4}}$$

One $n$-digit multiplication → Four $n/2$-digit multiplications

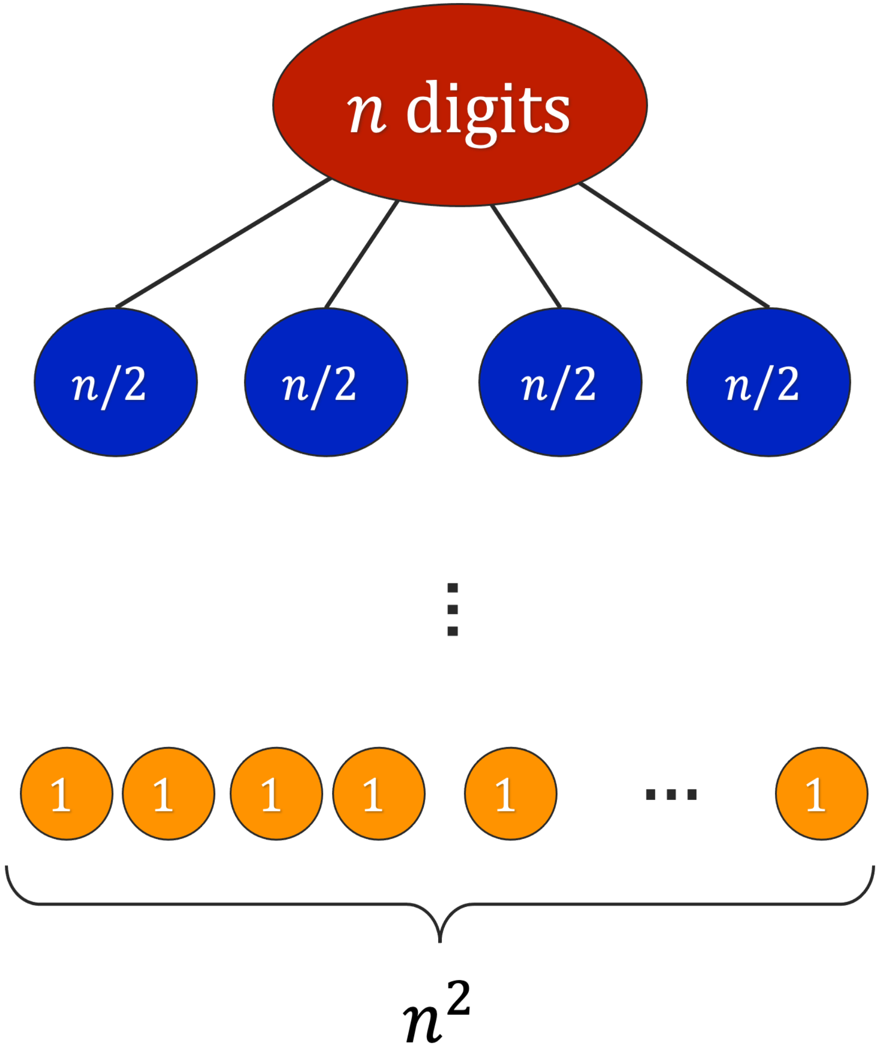
# Recap of last time

Introductions all arc

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 fa

→ Still didn't beat $O$




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

# This lecture

- 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!

$$x \times y = \left(a \times 10^{\frac{n}{2}} + b\right)\left(c \times 10^{\frac{n}{2}} + d\right)$$

$$= \underbrace{(a \times c)}_{\text{P1}} 10^{n} + (\underbrace{a \times d}_{\text{P2}} + \underbrace{c \times b}_{\text{P3}}) 10^{n/2} + \underbrace{(b \times d)}_{\text{P4}}$$

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?

Well... Gauss had used this trick too, for complex numbers...

# Karatsuba's Clever Trick

Let us only compute 3 things:

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

**Expressing P2+P3 differently**

$$a \times d + c \times b = (a+b)(c+d) - \mathrm{ac} - \mathrm{bd}$$

**Three subproblems**

$$x \times y = \left(a \times 10^{\frac{n}{2}} + b\right)\left(c \times 10^{\frac{n}{2}} + d\right)$$

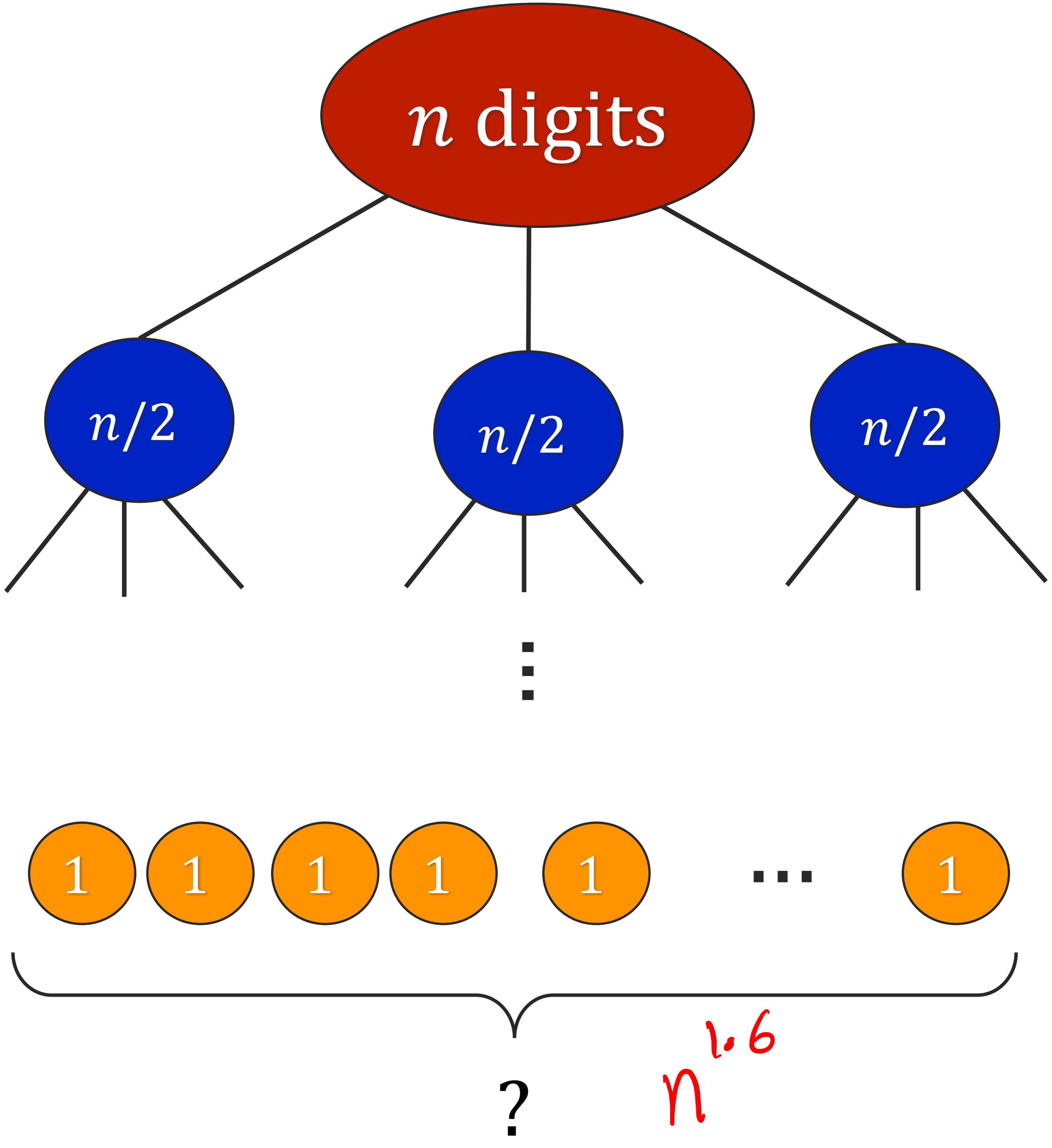
$$= \underbrace{(a \times c)}_{\text{Q1}} 10^n + \underbrace{(a \times d + c \times b)}_{\text{Q3−Q1−Q2}} 10^{n/2} + \underbrace{(b \times d)}_{\text{Q2}}$$

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 |
| ⋮ | ⋮ | ⋮ |
| $t$ | $\boxed{n/2^t}$ | $\boxed{3^t}$ |
| ⋮ | ⋮ | ⋮ |
| $\boxed{\log_2(n)}$ | 1 | $\boxed{3^{\log_2(n)} = n^{\log_2(3)}}$ |

$\approx n^{1.6}$

# 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 $2^k$.

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

$$\underbrace{b_1 \quad b_2 \quad b_3 \quad \cdots}_{b^L} \Big| \underbrace{\cdots b_{n-2} \quad b_{n-1} \quad b_n}_{b^R}$$

$$a \times b = (a^L \times b^L) 2^n + (a^L \times b^R + a^R \times b^L) 2^{n/2} + (a^R \times b^R)$$

$$= \cdots$$

**Practice:** Complete this equation the Karatsuba's way and rederived $O(n^{1.6})$ runtime for multiplying two $n$-bit numbers. Also on Homework 1!

# 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 $O(.)$.

# 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(…)

- 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 $O\left(g(n)\right)$" if and only if

for large enough n,

$T(n)$ is at most some constant multiple of $g(n)$.

# Definition of O(...)

- 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 $O\big(g(n)\big)$" if and only if

There exists $c$ and $n_0 > 0$

Such that for all $n \geq n_0, T(n) \leq c \cdot g(n)$

# Example

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

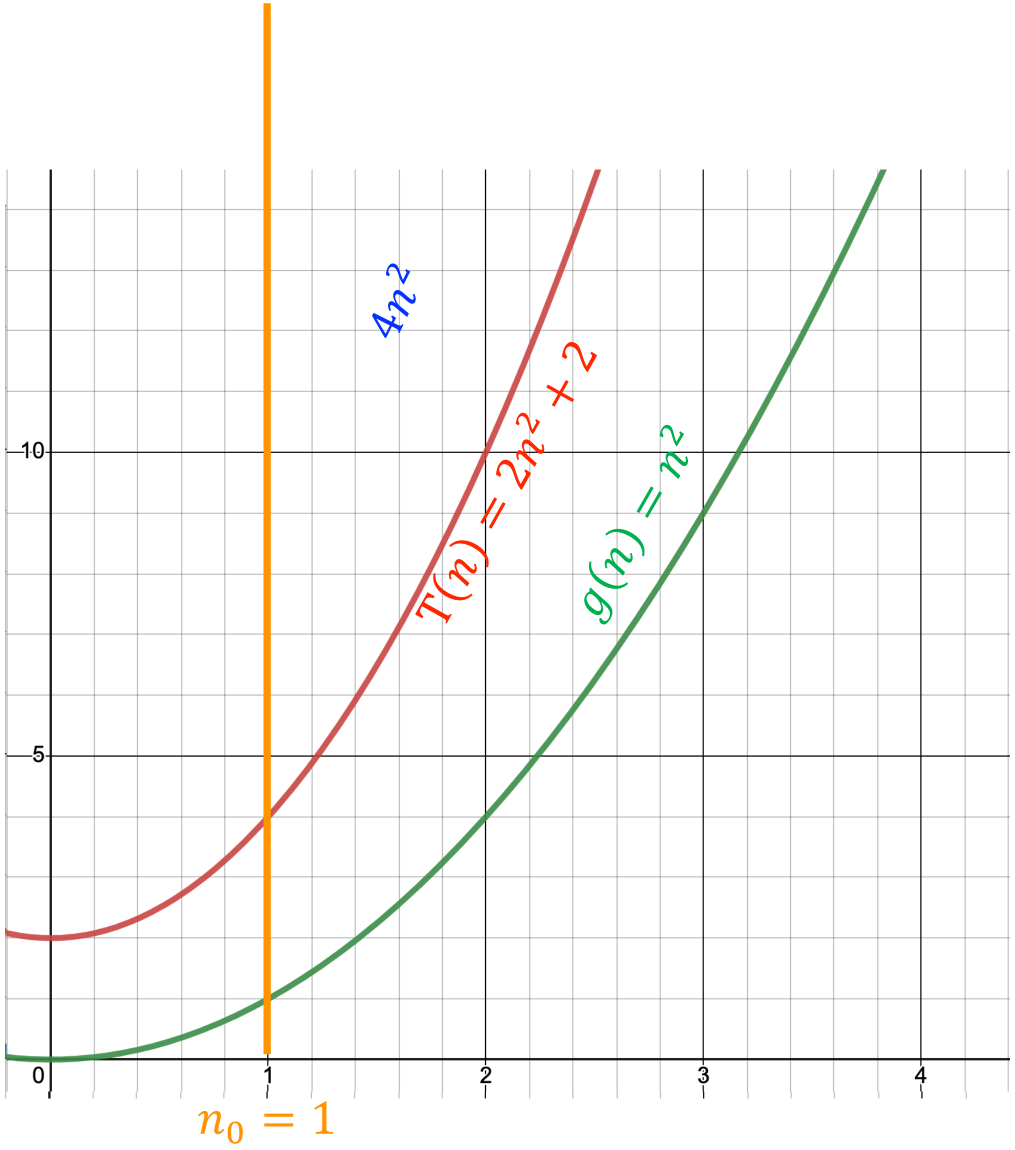
Even though $\mathrm{T}(n)$ is larger than $n^2$ always, we can find $c = 4$ and $n_0 = 1$, such that all $n > n_0$

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

How do you prove the above inequality?

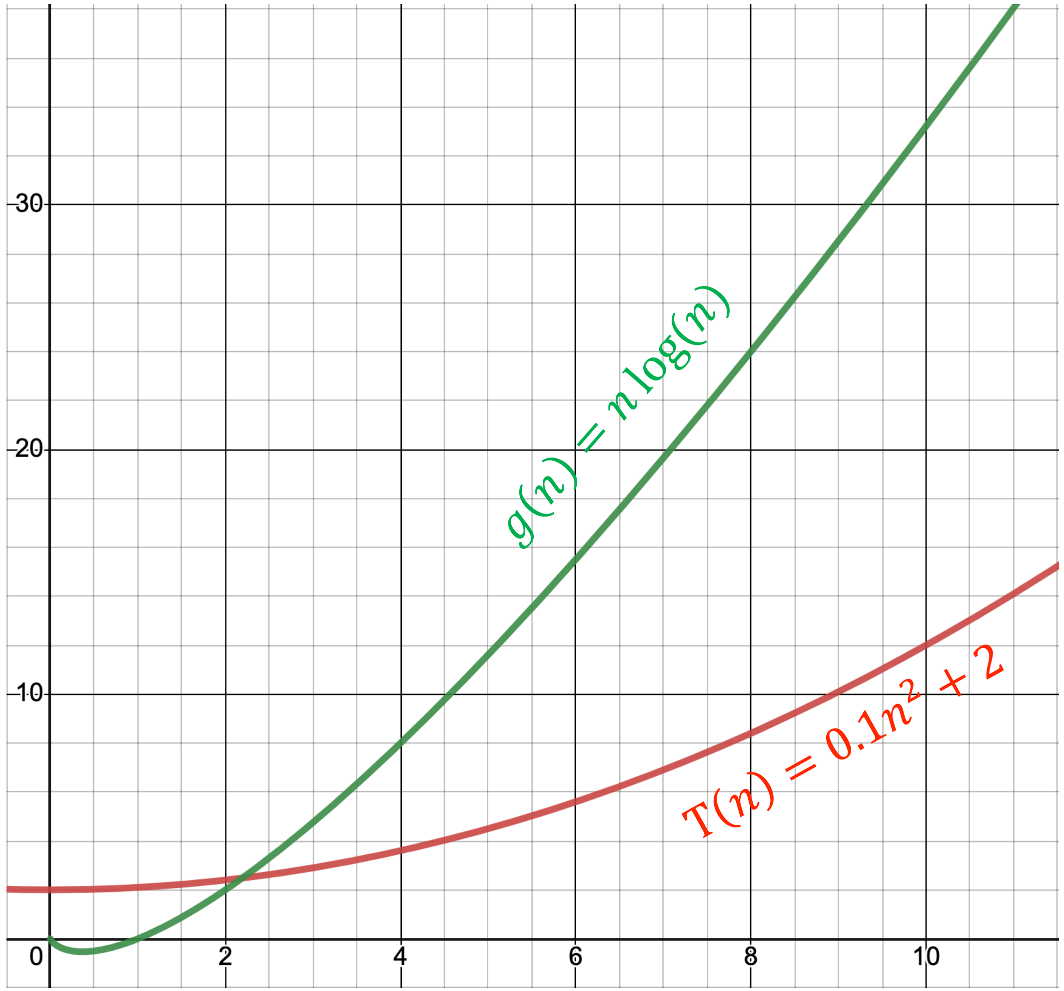
Whatever (correct!) math you like!

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



# BEWARE! of pictures!

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



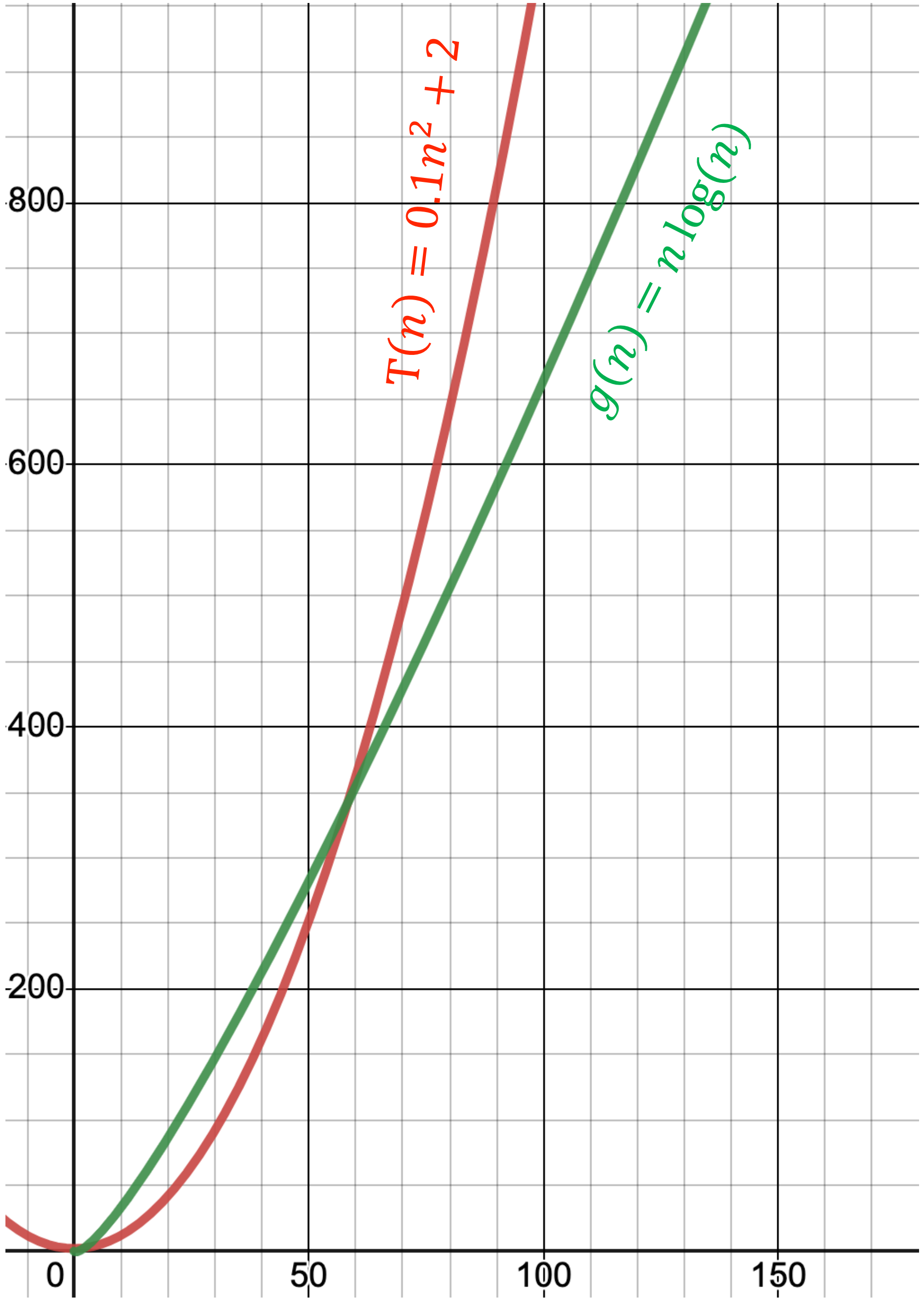

What's wrong with this argument and relying on pictures?

# BEWARE! of pictures!

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

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 $\overset{0.1}{}n^2 \in O(n)$.
- Then there is <u>some positive $c$ and $n_0$</u> so that:

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

- Divide both sides by $n$:

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

- That's not correct. Let $n = n_0 + 10$ c
  - 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.

# $\Omega(\ldots)$ 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 $n \geq n_0$, $c \cdot g(n) \leq T(n)$

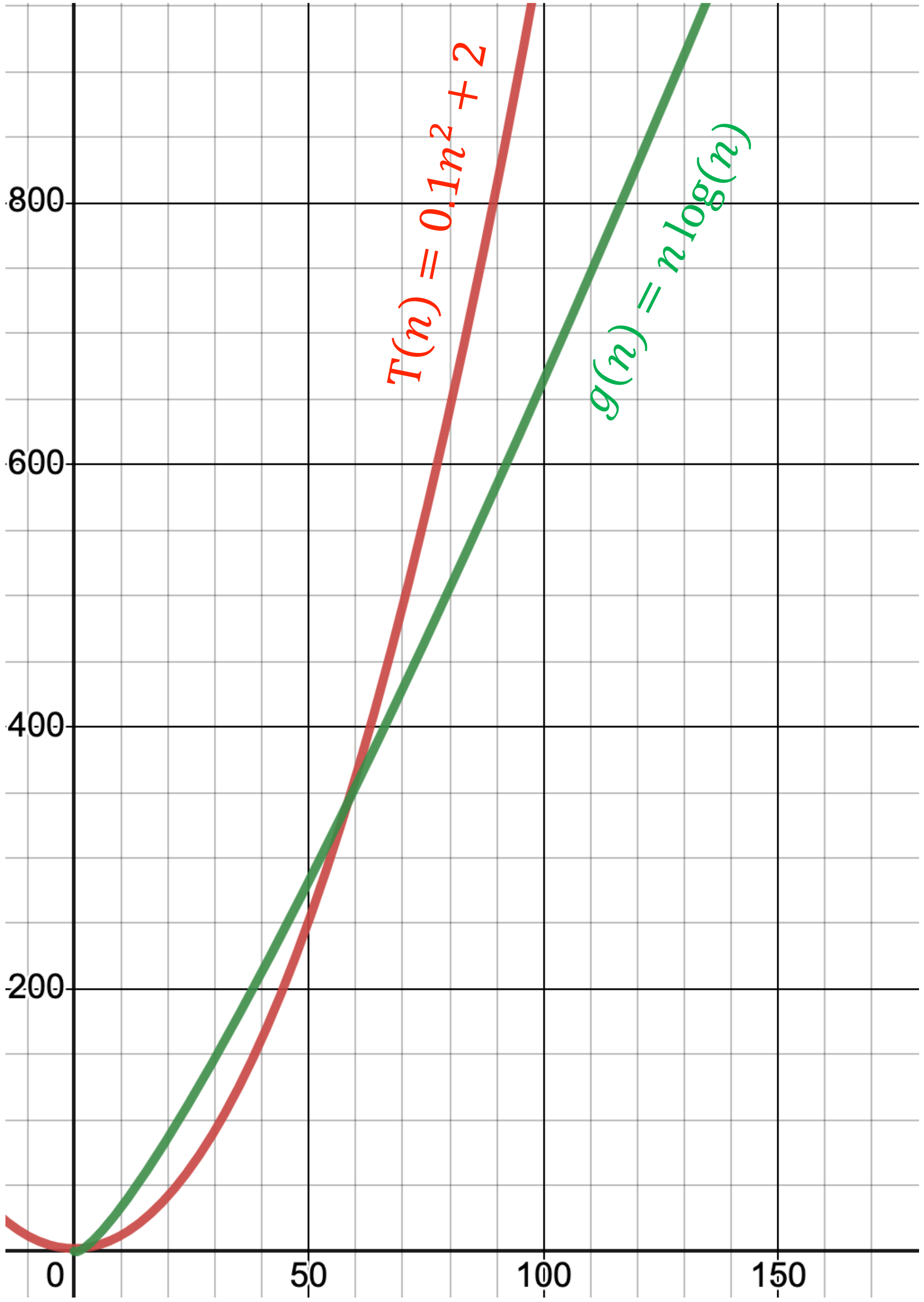
Switched these!!

# 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 $n \geq n_0$, $c\, n \log(n) \leq 0.1n^2 + 2$.



# $\Theta(\ldots)$ means both!

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

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

and

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

# Example: Asymptotics of the geometric series

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

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 $r \neq 1$:

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

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?

**Karatsuba's Alg in 1 layer**

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

$x \times y = \text{Q1} \times 10^n + (\text{Q3} - \text{Q1} - \text{Q2})10^{n/2} + \text{Q2}$

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

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 $\text{Q3} - \text{Q1} - \text{Q2}$
- $2n$-digit additions $\text{Q1} \times 10^n + (\text{Q3} - \text{Q1} - \text{Q2})10^{n/2} + \text{Q2}$
- ....

$O(n)$
More precisely $\leq 20n$

Runtime : $T(n) = 3\,T\left(\frac{n}{2}\right) + 20n$

# Recurrence Relations

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

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

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

$T(n) = \sum_{t=0}^{\log_2(n)} 3^t \cdot 20 \cdot \left(\frac{n}{2^t}\right)$

$= 20n \sum \left(\frac{3}{2}\right)^t$

$= 20n \cdot \Theta\left(\left(\frac{3}{2}\right)^{\log_2(n)}\right)$

$= 20n \cdot n^{\log_2(3/2)}$

$= 20n \cdot n^{\log_2(3) - \cancel{\log_2(2)}\ 1}$

$20\, n^{1 + \log_2(3) - 1} = n^{\log_2(3)}$

Abstraction of the tree

| Abstraction of the tree | Total contribution in this layer |
|---|---|
| size $n$ | $20 \cdot n$ |
| $3 \times n/2$ | $3 \times 20 \times \left(\frac{n}{2}\right)$ |
| $9 \times n/4$ | $9 \times 20 \left(\frac{n}{4}\right)$ |
| ⋮ | |
| $3^t \times n/2^t$ | $\boxed{3^t} \times \boxed{20 \left(\frac{n}{2^t}\right)}$ |
| ⋮ | |
| $3^{\log(n)} \times 1$ (Size 1) | $3^{\log(n)} \times 20$ ~ $T(1)$ |

# Solve Karatuba's Alg Recurrence Relation

a # subproblem

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

b

$$\sum_{i=0}^{\log(n)} 20n\left(\frac{3}{2}\right)^i = 20n \sum_{i=0}^{\log(n)} \left(\frac{3}{2}\right)^i$$

$$= O\left(n\left(\frac{3}{2}\right)^{\log(n)}\right)$$

$$= O\left(n \times n^{\log 3 - \log 2}\right)$$

$$= O\left(n^{\log(3)}\right) = O(n^{1.6})$$

| Abstraction of the tree | Total contribution in this layer |
|---|---|
| size $n$ | $20n$ |
| $3\times$ $n/2$ | $3\times 20\left(\frac{n}{2}\right)$ |
| $9\times$ $n/4$ | $9\times 20\left(\frac{n}{4}\right)$ |
| $\vdots$ | |
| $3^t \times$ $n/2^t$ | $3^t\times 20\left(\frac{n}{2^t}\right)$ |
| $\vdots$ | |
| $3^{\log(n)}\times$ $1$ (Size 1) | $3^{\log(n)}\times 20(1)$ |

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

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

# 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(.).

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

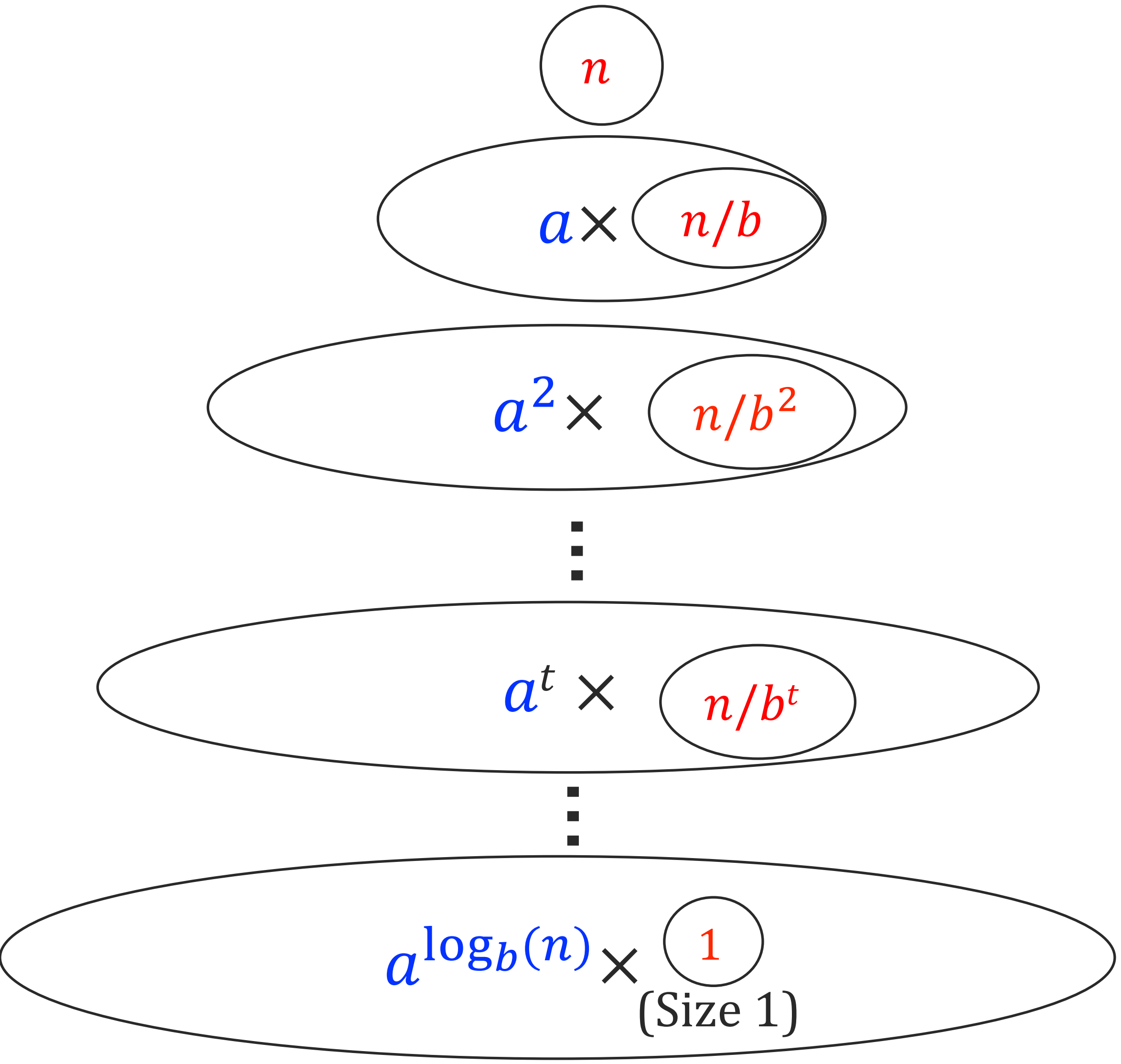
- For formal recursive arguments, we always substitute a constant.

→ Precise relationship between each layer's parameter and the amount of work.

→ Let's assume $T(1) = c$, too. For convenience!

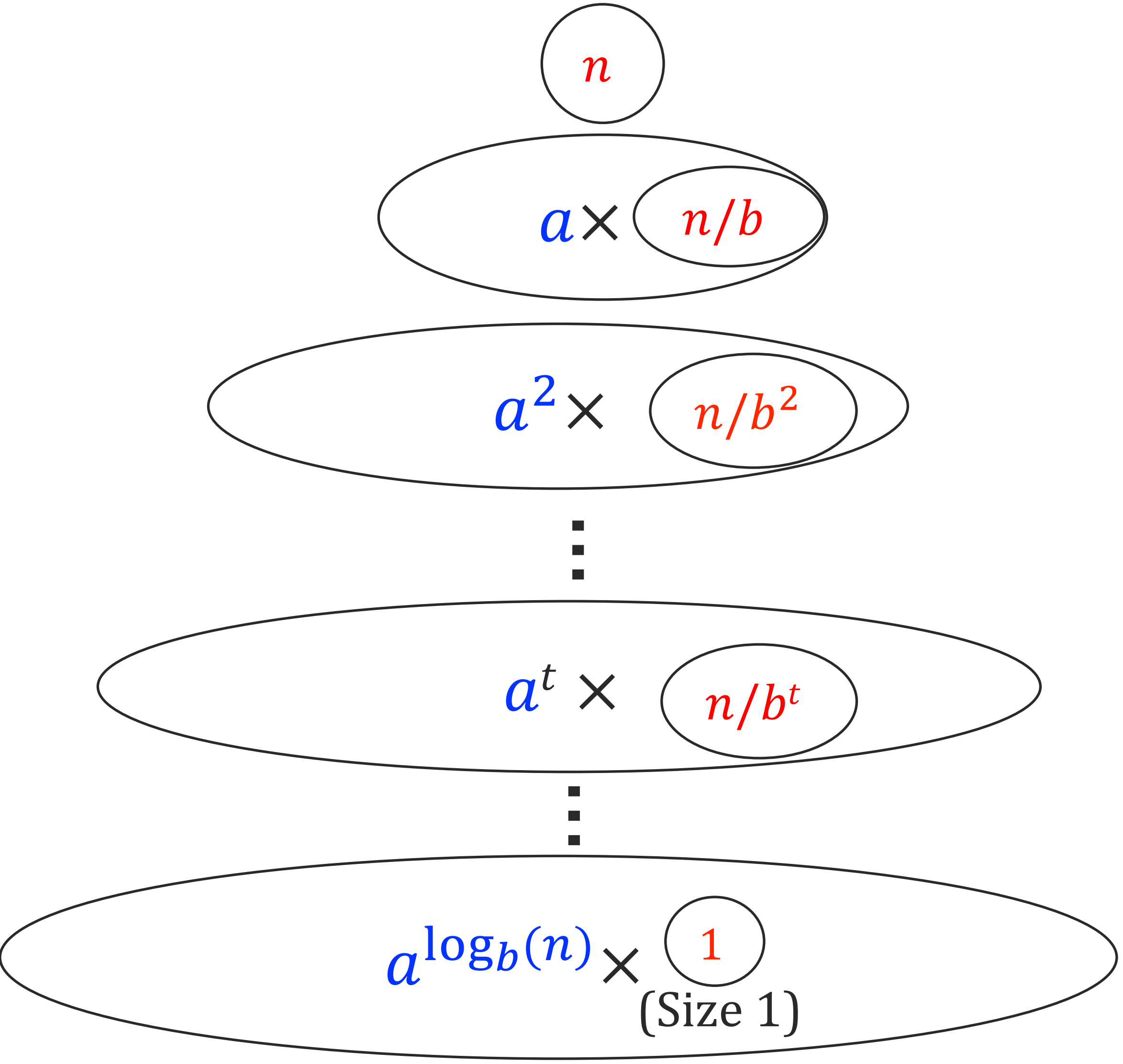
→ Just do the tree method!

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




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

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

| 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:

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

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

# The Master Theorem

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

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

Total computation on all layers:

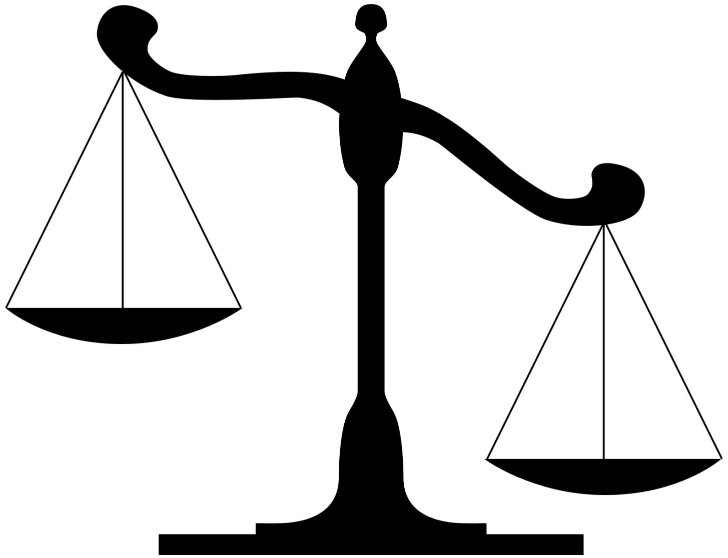
$$cn^d \cdot \underbrace{\sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t}_{\text{Geometric series}} = \begin{cases} \Theta\left(n^d \left(\frac{a}{b^d}\right)^{\log_b(n)}\right) & \text{if } a > b^d \\ \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log(n)) & \text{if } r = 1 \end{cases}$$

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

* In lecture we used $\Theta$ here. These should be $O(\cdot)$ only.

# Master Theorem's Interpretation

$a$ vs. $b^d$



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