CS 161

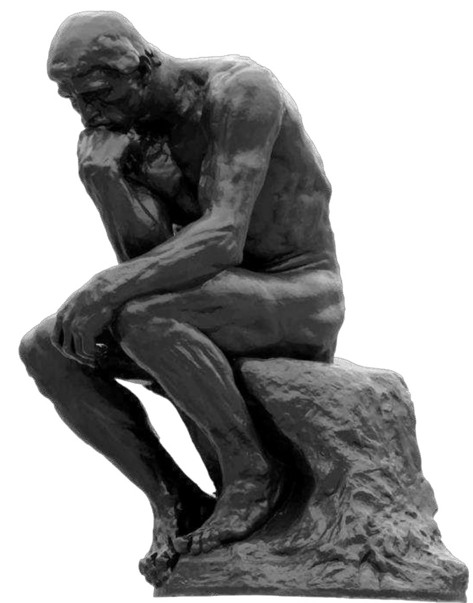
Design and Analysis of Algorithms

Lecture 1:

Logistics, introduction, and multiplication!

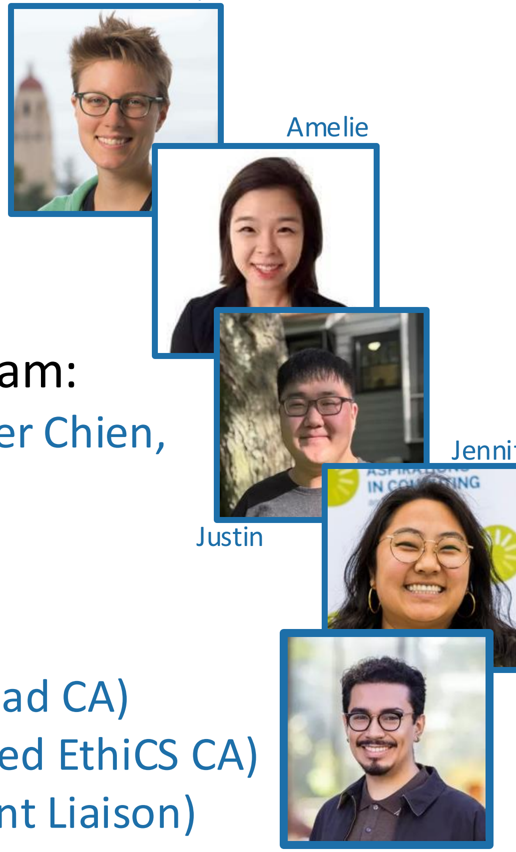
The big questions

- Who are we?
  - Course staff, students?
- Why are we here?
  - Why learn about algorithms?
- What is going on?
  - What is this course about?
  - Logistics?
- Can we multiply integers?
  - And can we do it quickly?



Who are we?

- Instructor:
  - Mary Wootters
- Course Coodinator:
  - Amelie Byun
- Embedded EthiCS Team:
  - Justin Shin, Jennifer Chien, Louis Ortiz



Mary
Amelie
Justin
Jennifer
Louie

- Awesome CAs!

- Anisha Palaparthi (Head CA)
- Maya Avital (Embedded EthiCS CA)
- Bradley Moon (Student Liaison)
- Zayn Malhotra
- Ta-Wei Tu
- Mingwei Yang
- Spencer Compton
- Will Fang
- Ziyi Ding
- Ly-Ly Atchariyachanvanit
- Karan Bhasin
- Simon Kim
- Ziyi Ding
- James Cheng
- Xiao Mao
- Andy Dai
- Isabel Sieh
- Audditthio Nag

Anisha

Isabel

Karan

Ly-Ly

Ta-Wei

Spencer

Simon

Will

Zayn

Maya

Xiao

Mingwei

James

Bradley

Andy

Audditthio

Who are you?

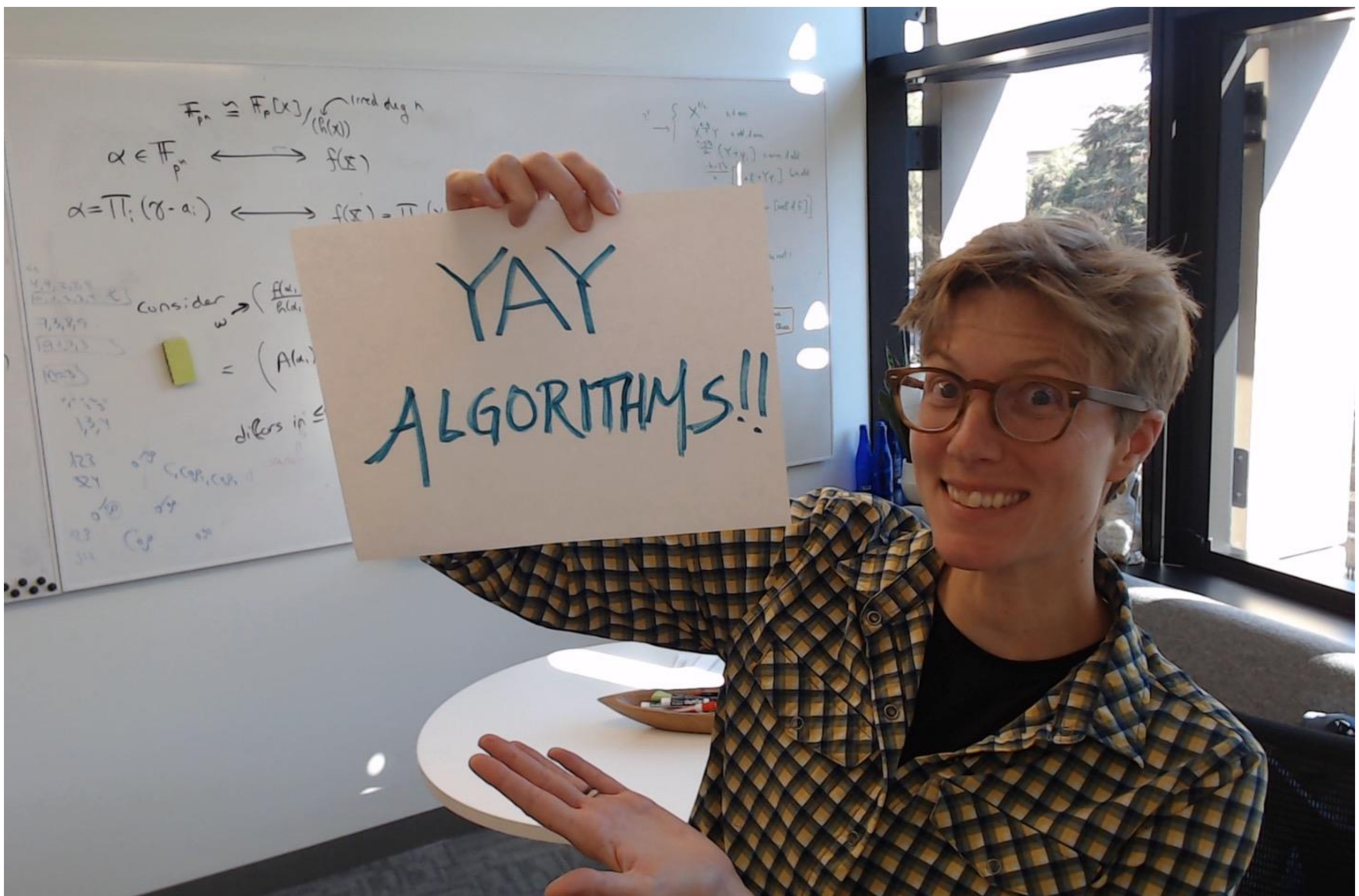
- Freshman
- Sophomores
- Juniors
- Seniors
- MA/MS Students
- PhD Students
- NDO Students

Concentrating in:

- Art Practice
- Bioengineering
- Biology
- Biomedical Data Science
- Biomedical Informatics
- Chemical Eng.
- Chemistry
- Civil & Env. Eng.
- Classics
- Communication
- CME
- Computer Science
- Creative Writing
- Data Science
- Earth Systems
- Economics
- Education
- EE
- Engineering
- Hum Bio
- International Relations
- Linguistics
- Math
- Music
- MS&E
- Mech. Eng.
- Physics
- Political Science
- Sociology
- Statistics
- Symbolic Systems
- Theater and Perf. Studies
- Undeclared

Why are we here?

- I’m here because I’m super excited about algorithms!



Why are you here?

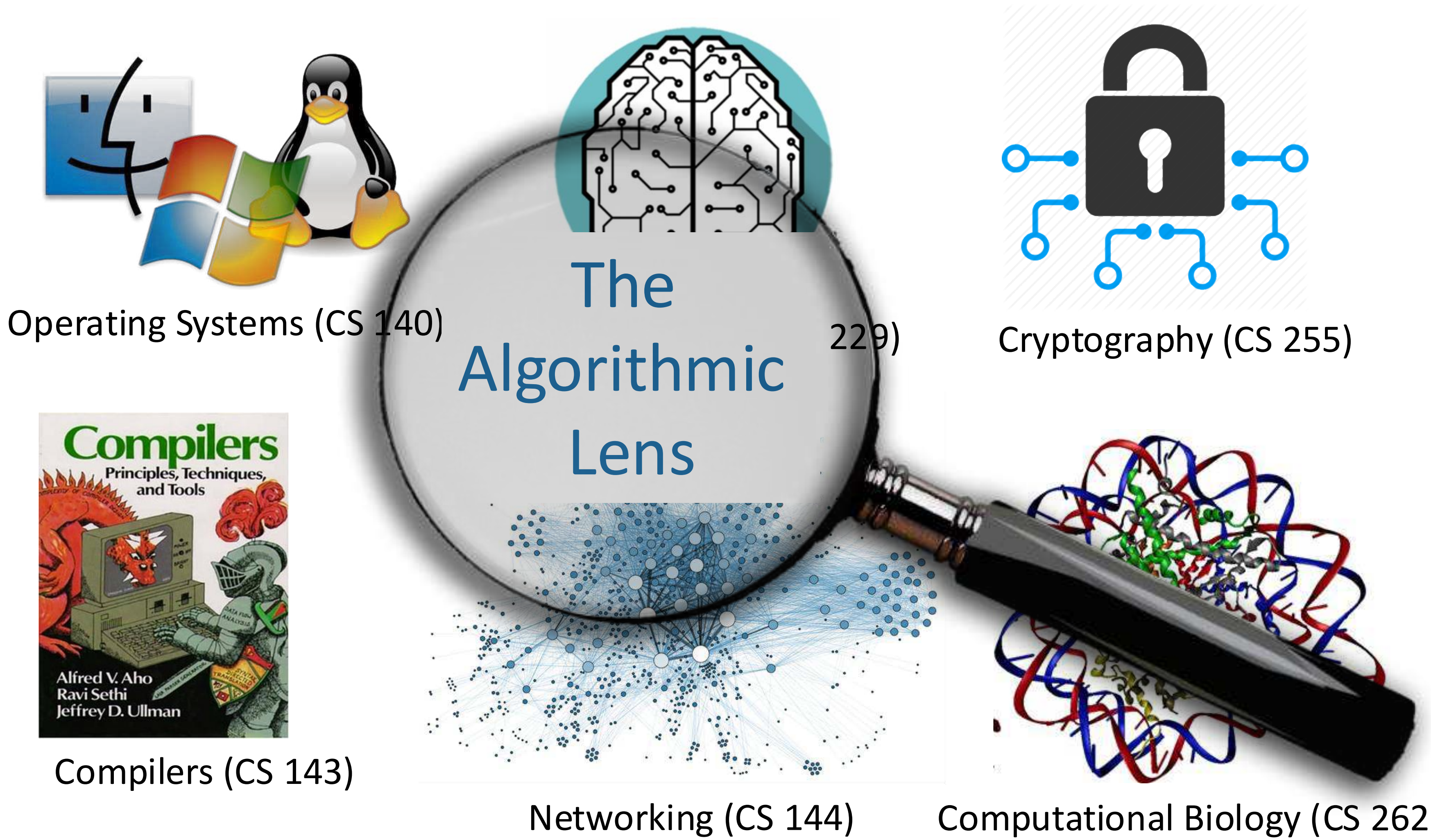
You are better equipped to answer this question than I am, but I'll give it a go anyway…

- Algorithms are fundamental.
- Algorithms are useful.
- Algorithms are fun!
- CS161 is a required course.

Why is CS161 required?

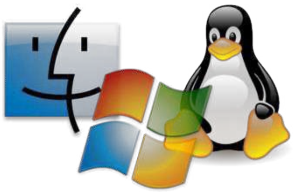
- Algorithms are fundamental.
- Algorithms are useful.
- Algorithms are fun!

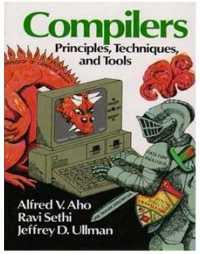
Algorithms are fundamental

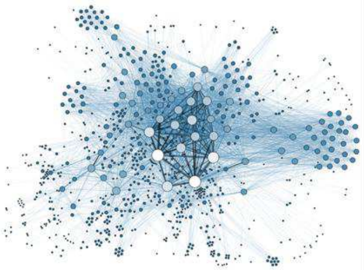
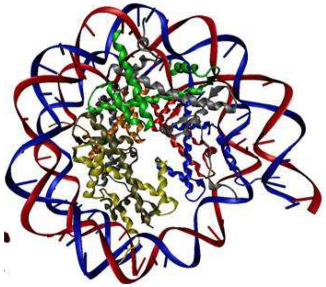


Algorithms are useful

- All those things without the course numbers.
- As inputs get bigger and bigger, having good algorithms becomes more and more important!





Algorithms are fun!

- Algorithm design is both an art and a science.
- Many surprises!
- Many exciting research questions!

What’s going on?

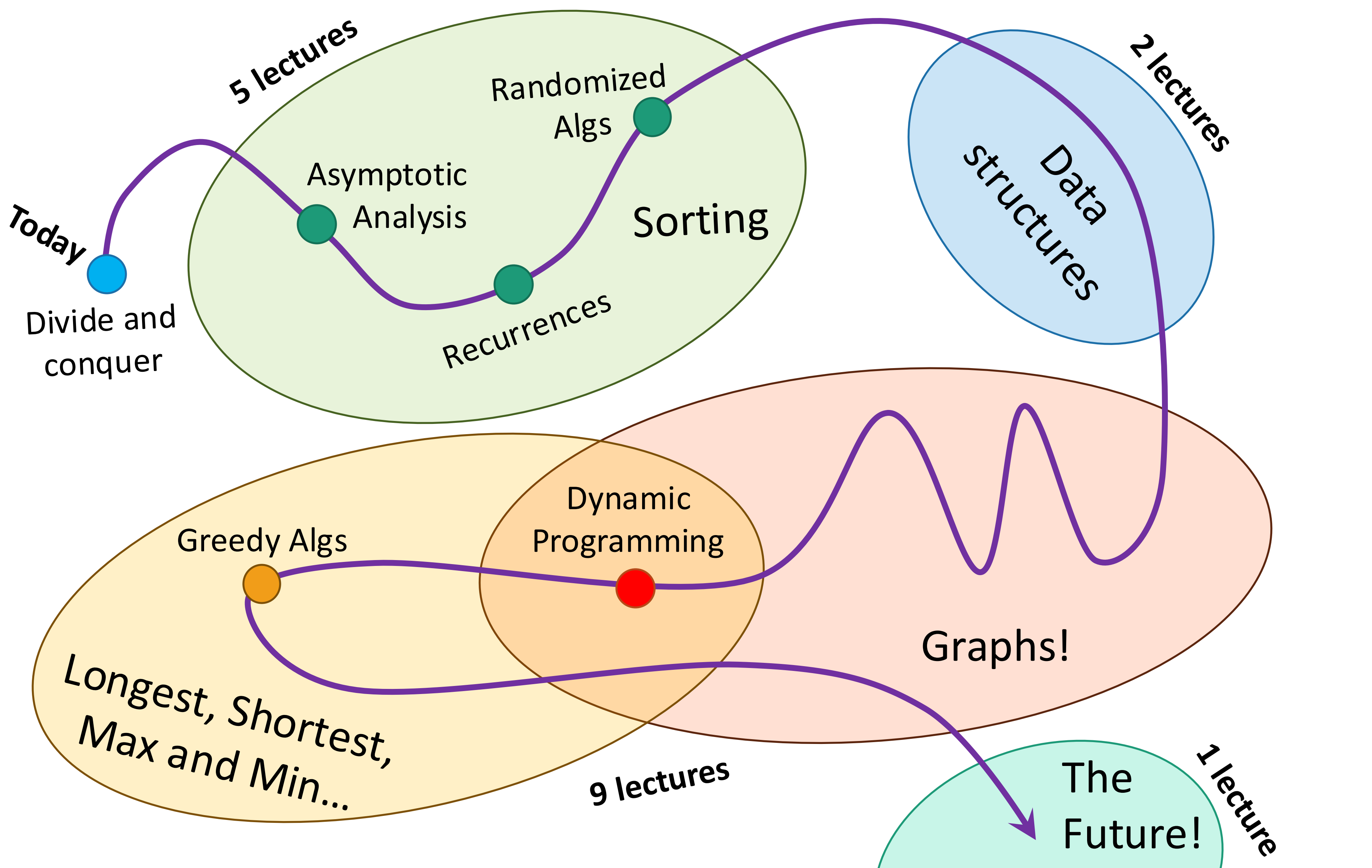
- Course goals/overview
- Logistics

Course goals

- The design and analysis of algorithms
  - These go hand-in-hand
- In this course you will learn:
  - Design: Flesh out an “algorithmic toolkit”
  - Analysis: Learn to think analytically about algorithms
  - Communication: Learn to communicate clearly about algorithms

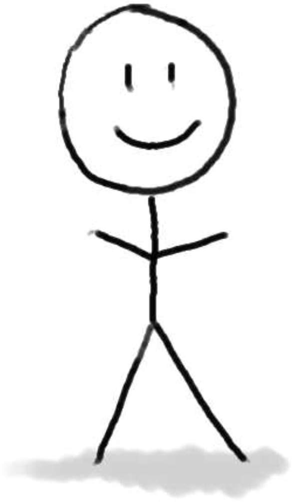
More detailed schedule on the website!

Roadmap



Our guiding questions:

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



Our internal monologue…

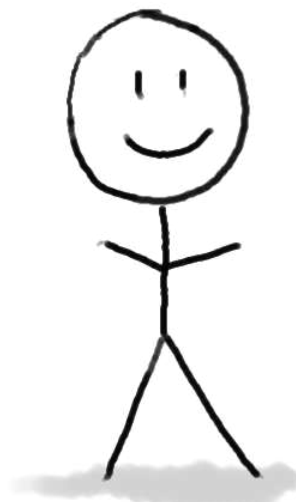
What exactly do we mean by better? And what about that corner case? Shouldn't we be zero-indexing?

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

Dude, this is just like that other time. If you do the thing and the stuff like you did then, it'll totally work real fast!

Plucky the
Pedantic Penguin

Detail-oriented
Precise
Rigorous



Lucky the
Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

Both sides are necessary!

The bigger picture

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

- Should it work?
- Should it be fast?

Embedded EthiCS

- Throughout the course, we will take a step back and focus on how algorithm design can affect society, and the ethical implications of that.

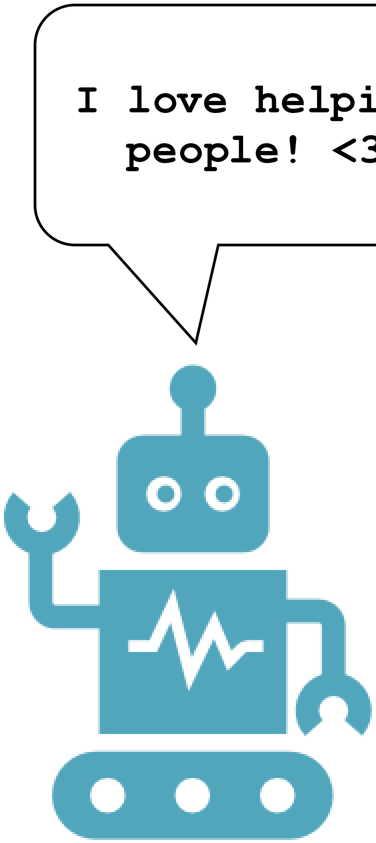
- Embedded EthiCS team: Jennifer, Justin and Louie!

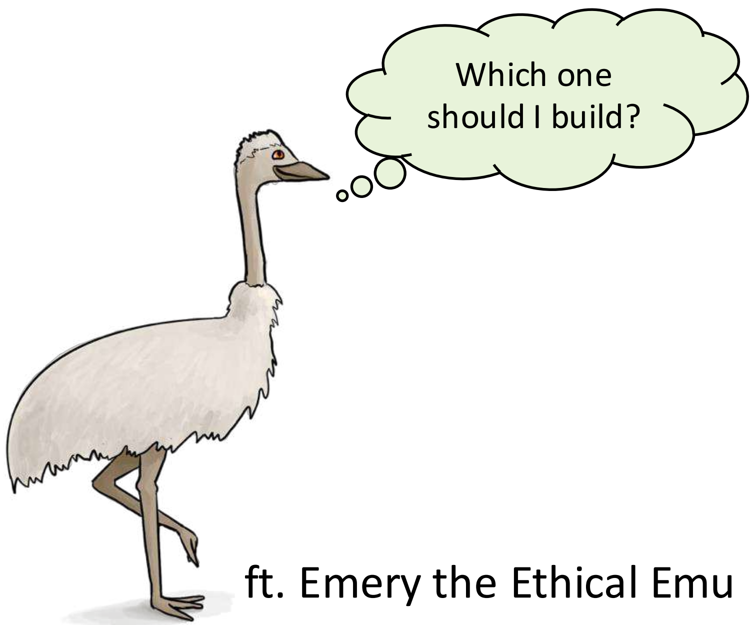
Welcome to Embedded Ethics!

Dr. Jennifer Chien

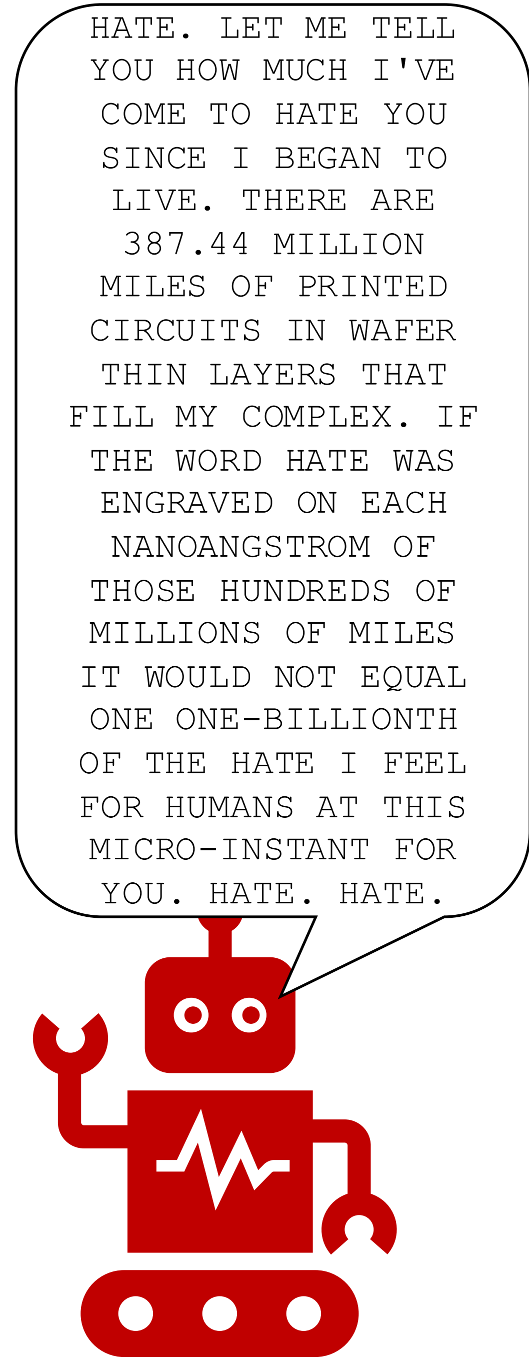
Dr. Justin Shin

Louie Ortiz





ft. Emery the Ethical Emu



Jennifer

- I majored in Computer Science, with minors in Math and Statistics
- Then I got my PhD in Computer Science, focusing on AI Ethics
- I spend a lot of time thinking about:
  - What are the goals of a system?
  - What are the limitations of purely technical approaches?
  - How do we design/intervene to protect user agency?

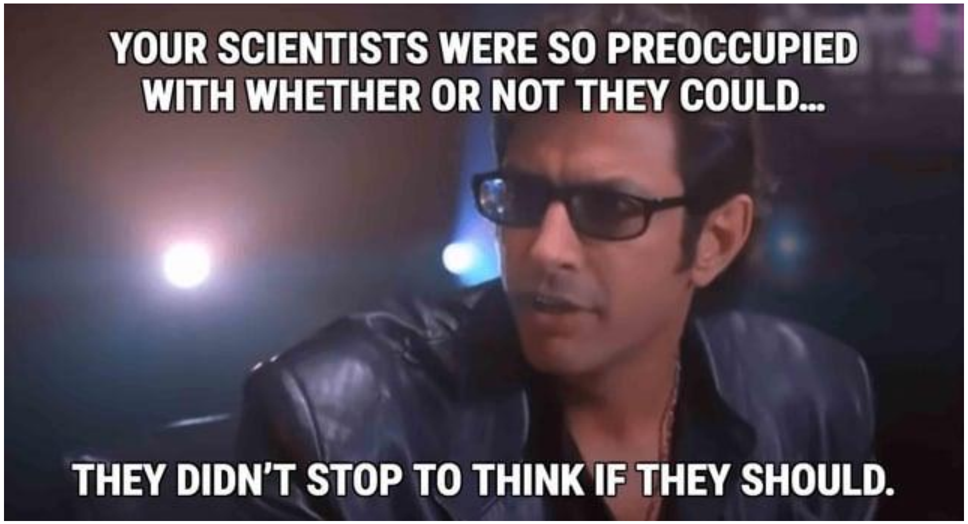
Justin

- I double majored in Mathematics and Philosophy as an undergrad
- Then got my PhD in the History and Philosophy of Science
- I am thinking about…
  - What kinds of statistical evidence should be accepted as evidence of discrimination in courts?
  - How should we handle expert testimony from scientists in cases of controversial science?
  - How do techno-labor revolutions change how we value work?

Louie

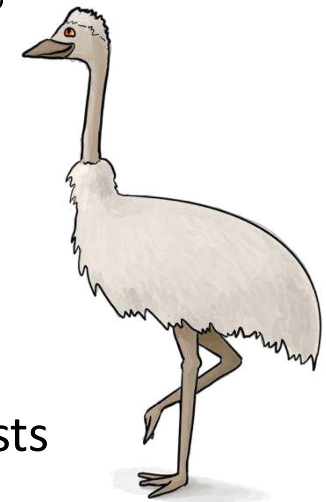
- BA in Data Science with a concentration in Philosophy
- 2nd year Research Associate with the Rising Scholars program
- I spend my time thinking about:
  - How do we teach ethics effectively (as an educational intervention) in technical fields?
  - How do students prioritize ethics against other principles when building technology?
  - Do demographic characteristics shape our moral agency?

What is Embedded Ethics?



Spielberg, S. (1993). Jurassic Park. Universal Pictures.

Don't worry Dr. Ian Malcolm, we **will** stop to think if we should!

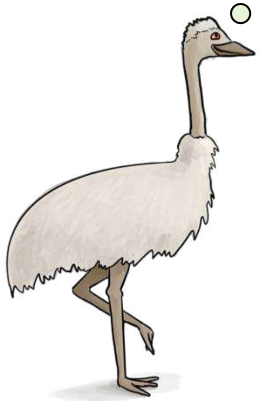


Embedded Ethics: Training the next generation of computer scientists to "consider ethical issues from the outset rather than building technology and letting problems surface downstream" by integrating skills and habits of ethical analysis throughout the Stanford Computer Science curriculum.

What do we teach?

- Issue spotting and ethical sensitivity
- Recognizing values in design choices
- Developing language to talk about moral choices
- Professional responsibilities of computer scientists & software engineers
- Important topics in technology ethics: bias & fairness, inequality, privacy, surveillance, data control & consent, trust, disinformation, participatory design, concentration of power.

How do we make sure we aren’t losing important features of the real world problem when we formalize it?

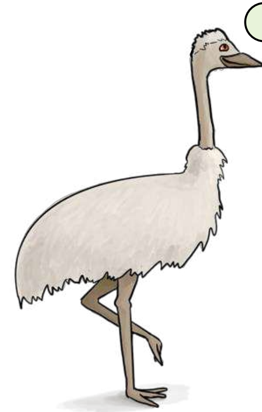


Turn a real world problem into a formal (math) problem

Use an algorithm to solve the problem

Happiness ensues

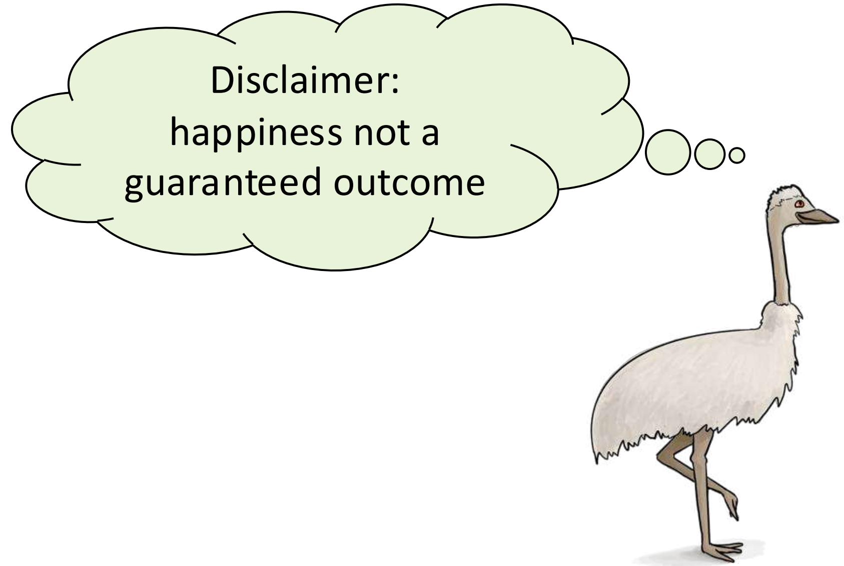
By the time you finish CS161,
you will have a tool kit stuffed
with algorithms!
Which one is right for the job?



Turn a real world problem into a formal (math) problem

Use an algorithm to solve the problem

Happiness ensues

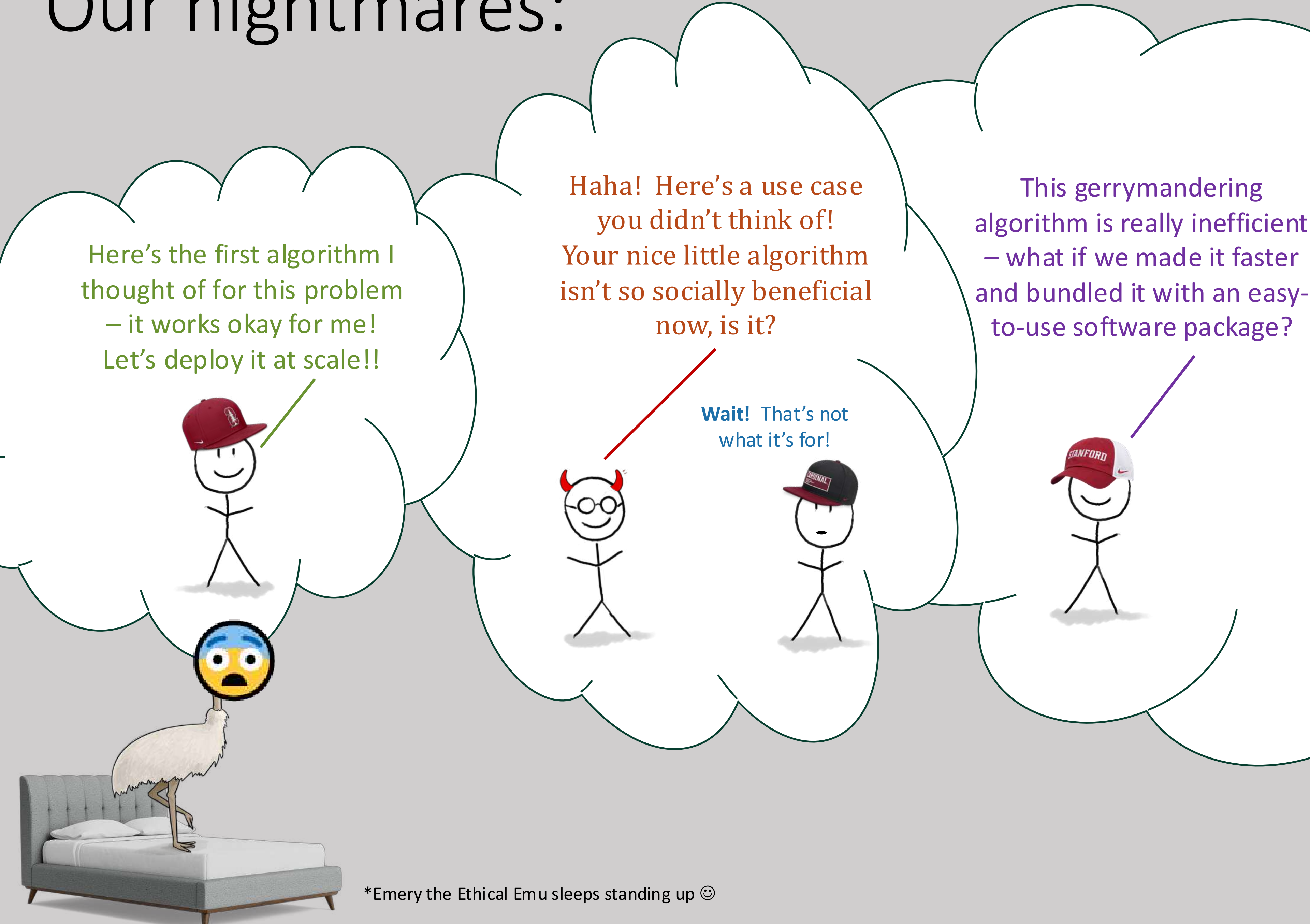


Turn a real world problem into a formal (math) problem

Use an algorithm to solve the problem

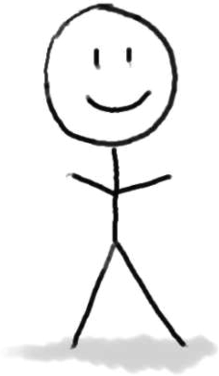
Happiness ensues

Our nightmares:



*Emery the Ethical Emu sleeps standing up ☺

Our guiding questions:



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

Thank you!

You can always email us at drchien@stanford.edu
justinjs@stanford.edu and louieortiz@stanford.edu

We are happy to talk! About ethics, research, careers, pets…

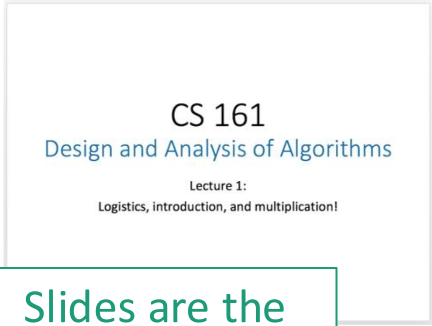
Thanks to Katie Creel and the Embedded Ethics program for slides!

Course elements and resources

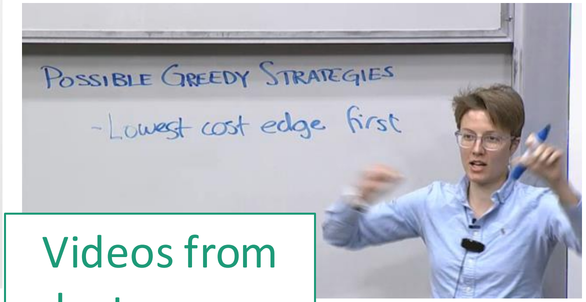
- **Course website:**
  - **cs161-stanford.github.io**
- Lectures
- Homework
- Exams
- Office hours, Sections, and Ed

Lectures

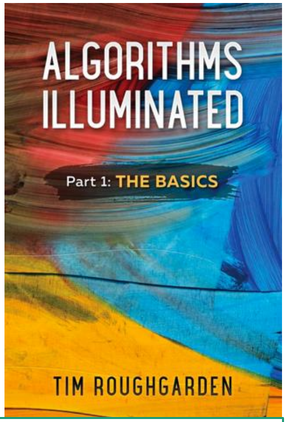
- Right here (Bishop Auditorium), T/Th, 9-10:20am!
- Resources available:
  - Slides, Videos, Book, IPython notebooks



Slides are the slides from lecture.

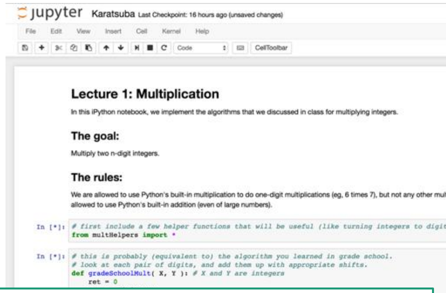


Videos from lecture are available!



Textbook (and occasional hand-outs) have mathy details that slides may omit

(required)

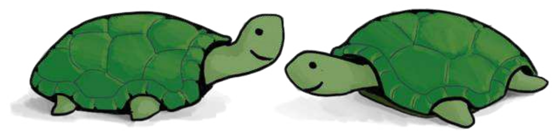


IPython notebooks have implementation details that slides may omit.

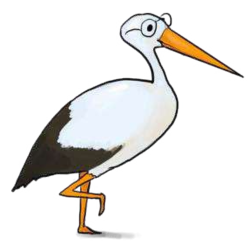
(optional)

How to get the most out of lectures

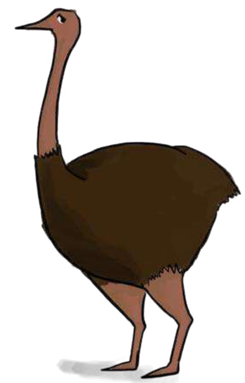
- **During lecture:**
  - Show up or tune in, ask questions.
  - Engage with in-class questions.
- **Before lecture:**
  - Do ***pre-lecture exercises*** on the website.
- **After lecture:**
  - Go through the exercises on the slides.



Think-Pair-Share Terrapins (in-class questions)



Siggi the Studious Stork (recommended exercises)



Ollie the Over-achieving Ostrich (challenge questions)

- ***Do the reading***
  - either before or after lecture, whatever works best for you.
  - **do not wait to “catch up” the week before the exam.**

Homework!

- Weekly assignments HW1-HW7
  - Due Fridays at 11:59pm, HW1 due Oct 3
  - Done in groups of up to 3.

- Special HW0!
  - Goal: assess your background and pre-requisites
  - Graded for completion
  - Do this one on your own
  - Out now, due TUESDAY September 30 before class!

Aside: Late days

- You have six late days to use on HW1 through HW7.
  - See website for more details

- **LATE DAYS ARE FOR EMERGENCIES**. Do not ask us for an extension if you have an emergency. That's what late days are for.

Exams

- Three exams
  - **Exam 1:** Thursday 10/16, in class.
  - **Exam 2:** Thursday 11/6, in class.
  - **Final Exam:** Wednesday 12/10, 8:30am-11:30am
- **We will not have scheduled alternate exams.**
  - If you know you cannot take an exam, you should drop this class and take it in a different quarter.
- We are participating in the AIWG proctoring pilot
  - See website for details

Course elements and resources

- **Course website:**
  - **cs161-stanford.github.io**
- Lectures
- Homework
- Exams
- Office hours, Sections, and Ed

Talk to us!

- Join Ed:
  - You should be auto-enrolled (may take some time to sync)
  - Course announcements will be posted there
  - Discuss material with course staff and classmates!
- Office hours:
  - See course website for schedule
  - Start in week 2
- Sections:
  - See course website for schedule; one section is recorded
  - Technically optional, but ***highly recommended***!
  - Extra practice with the material, example problems, etc.
- High-Resolution Course Feedback:
  - Anonymous weekly feedback for the teaching team
  - You'll get a few emails randomly during the quarter asking for feedback. ***Please respond!***

Talk to each other!

- Collaboration on HW
- Answer your peers' questions on Ed!
- We will host **Homework Parties**.
  - Mondays 5:30-7:30pm, starting Week 2
  - There will be snacks!

Course elements and resources

- **Course website:**
  - **cs161-stanford.github.io**
- Lectures
- Homework
- Exams
- Office hours, Sections, and Ed

Course Policies

- Course policies are listed on the website.
  - Collaboration Policy, LLM Policy, Academic Honesty, ...
- Read them and adhere to them.

Bug bounty! 

- We hope all course materials will be bug-free.
- However, I sometmes maek typos.
- **If you find a typo** (that affects understanding*) on slides, IPython notebooks, Section material or PSETs:
  - Let us know! (Post on Ed).
  - The first person to catch a bug gets a bonus point.

Bug Bounty Hunter

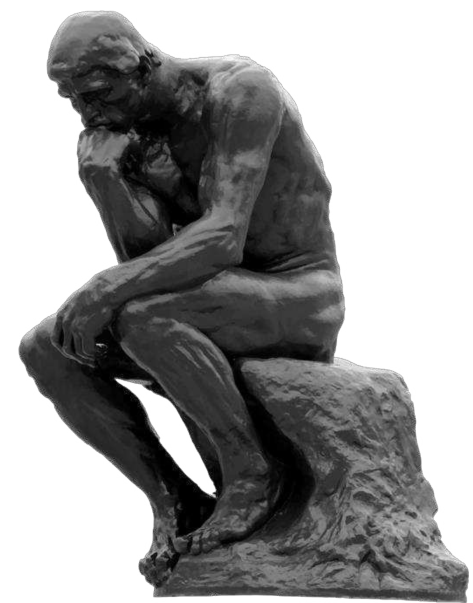
*So, typos lke thees onse don't count, although please point those out too. Typos like 2 + 2 = 5 do count, as does pointing out that we omitted some crucial information.

Everyone can succeed in this class!

1. Work hard
2. Work smart
3. Ask for help

The big questions

- Who are we?
  - Professor, TA’s, students?
- Why are we here?
  - Why learn about algorithms?
- What is going on?
  - What is this course about?
  - Logistics?
- Can we multiply integers?
  - And can we do it quickly?



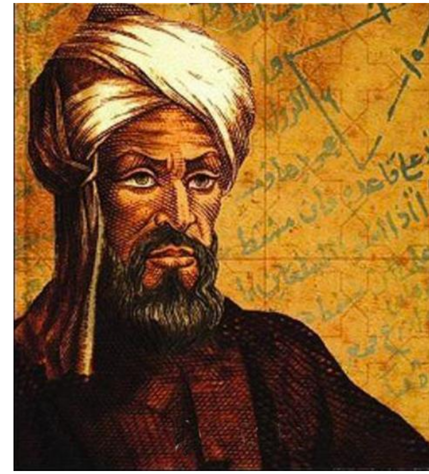
For the rest of today

- Karatsuba Integer Multiplication
- Algorithmic Technique:
  - Divide and conquer
- Algorithmic Analysis tool:
  - Intro to asymptotic analysis

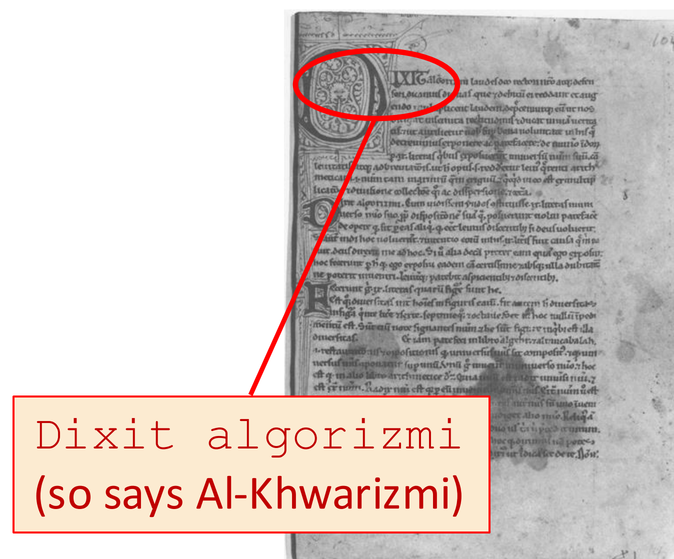
Let’s start at the beginning

What was the first "Algorithm"?

- The word "Algorithm" comes from the name "Al-Khwarizmi"
  - 9^{th} century scholar who worked in Baghdad during the Abbassid Caliphate.
- Among many other contributions in mathematics, astronomy, and geography, he wrote a book about **how to multiply with Arabic numerals**.
- His ideas came to Europe in the 12^{th} century, and gave rise to the word "Algorithm"
  - Originally, "Algorisme" [old French] referred to just the Arabic number system
  - Eventually it came to mean "Algorithm" as we know today.



Al-Khwarizmi



An algorithm for multiplication was kind of a big deal

$$\text{XLIV} \times \text{XCVII} = ?$$

$$\begin{array}{r} 44 \\ \times\ 97 \\ \hline \end{array}$$

Integer Multiplication

$$\begin{array}{r} 44 \\ \times\ 97 \\ \hline \end{array}$$

Integer Multiplication

$$
\begin{array}{r}
1234567895931413 \\
\times \quad 4563823520395533 \\
\hline
\end{array}
$$

Integer Multiplication

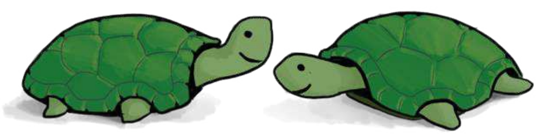
n

  1233925720752752384623764283568364918374523856298
x 4562323582342395285623467235019130750135350013753

???

How fast is the grade-school multiplication algorithm?

(How many one-digit operations?)



Think-pair-share Terrapins

About n^2 one-digit operations

Plucky the Pedantic Penguin

At most n^2 multiplications,
and then at most n^2 additions (for carries)
and then I have to add n different 2n-digit numbers…

Big-Oh Notation

- We say that Grade-School Multiplication

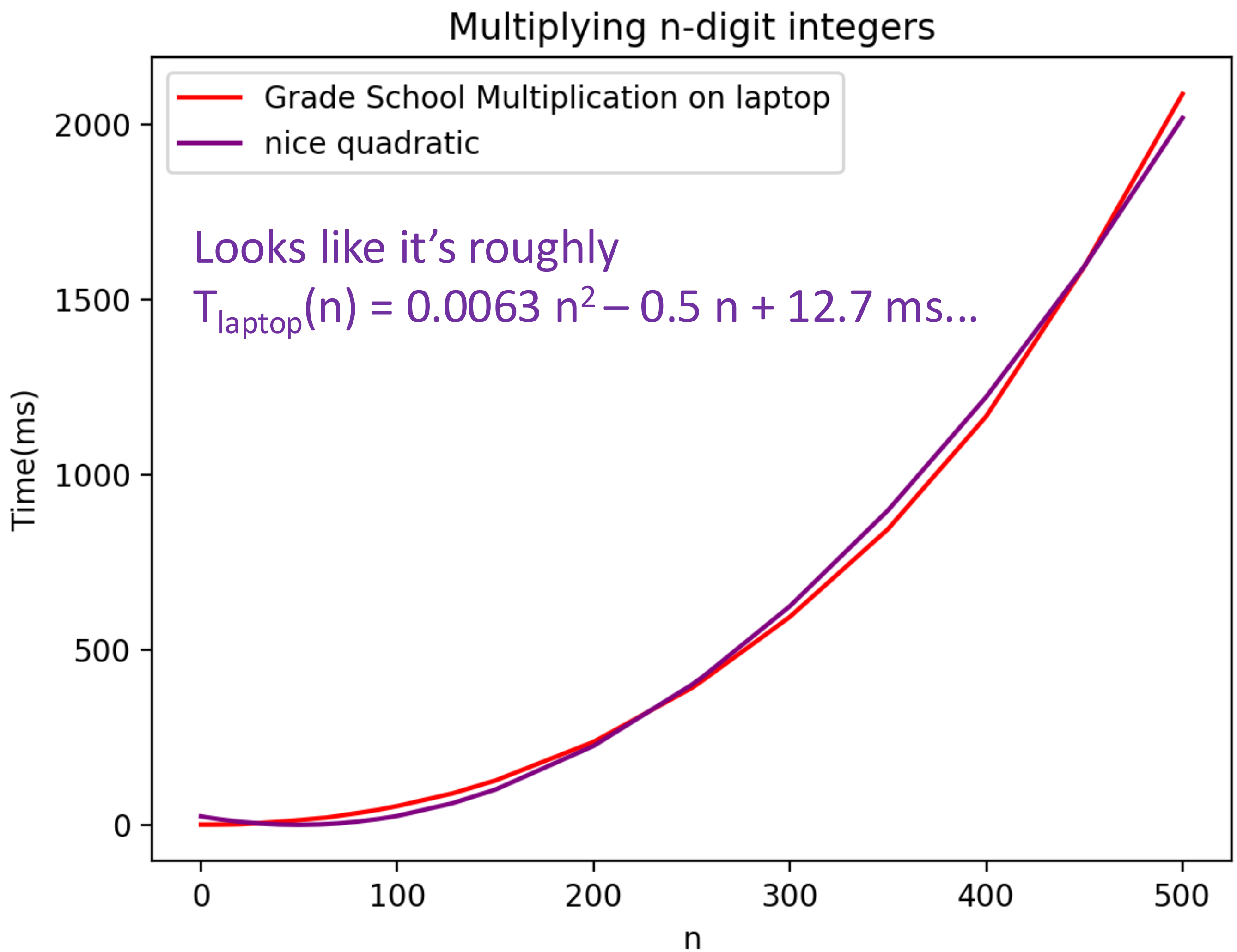
  "runs in time $O(n^2)$"

- Formal definition coming next time!
- Informally, big-Oh notation tells us how the running time scales with the size of the input.

highly non-optimized

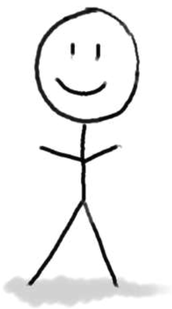
Implemented in Python, on my laptop

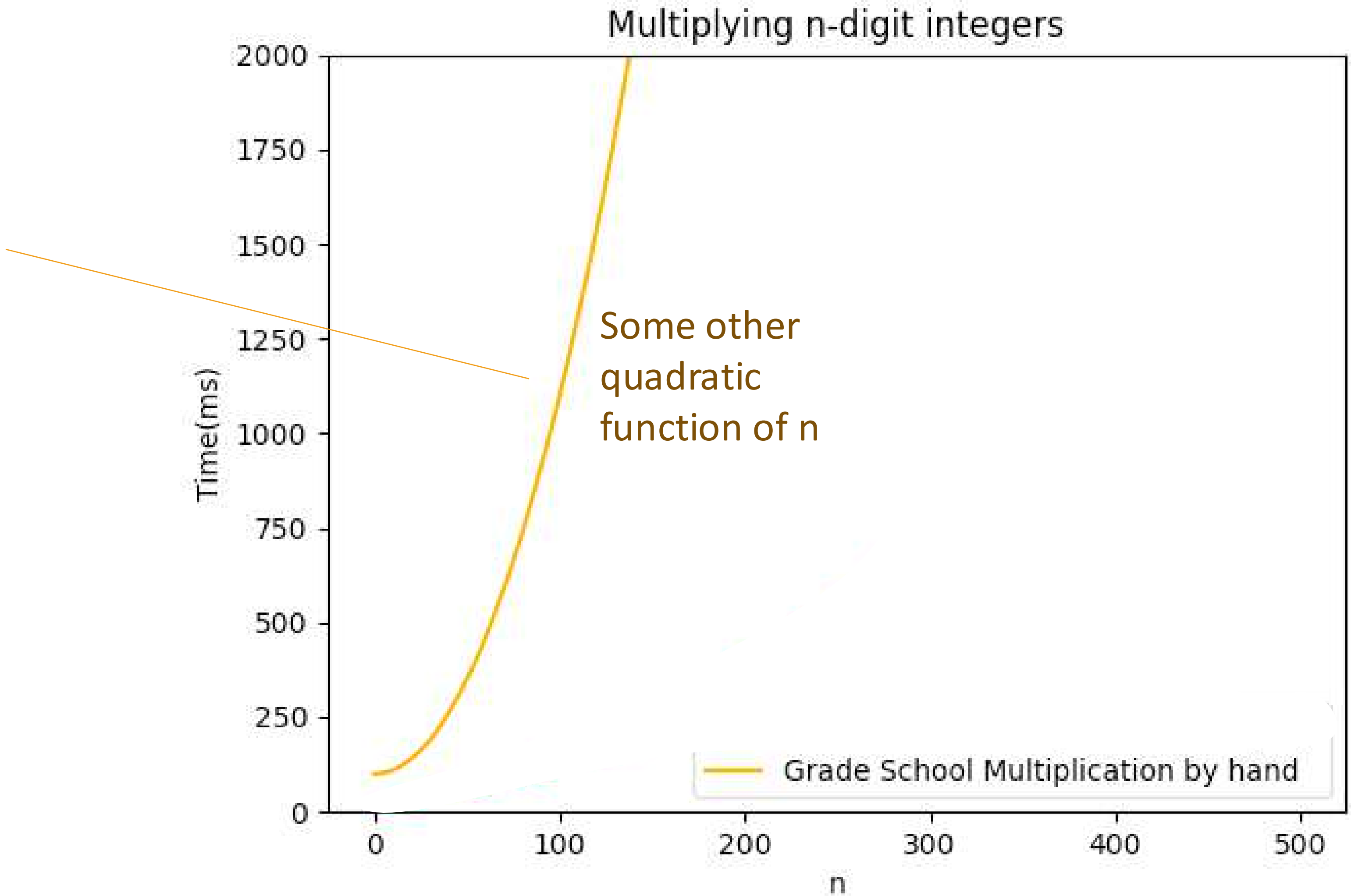
The runtime “scales like” n^2



Implemented by hand

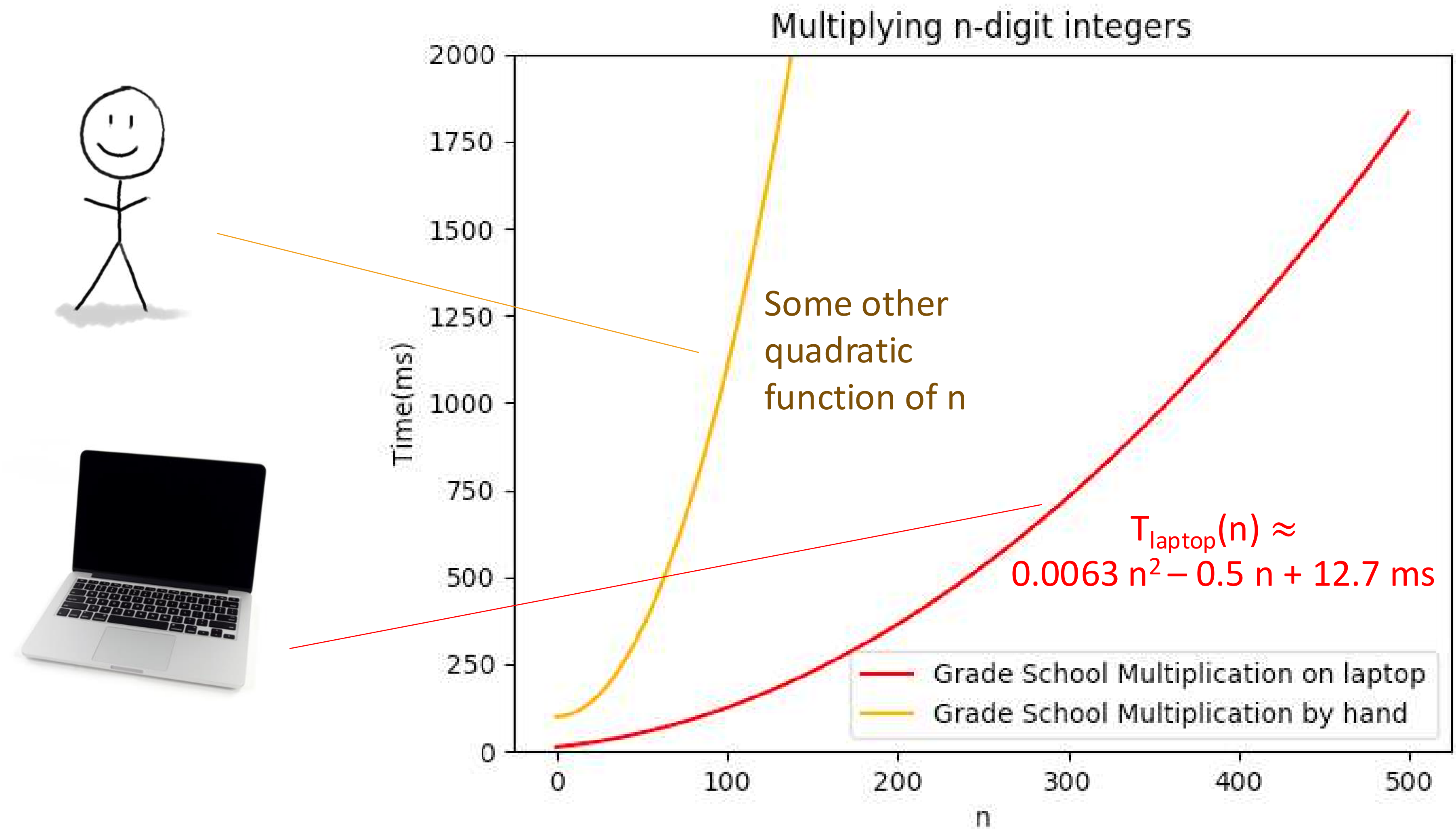
The runtime still “scales like” n^2



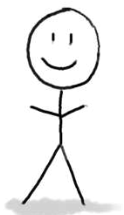


Implemented by hand

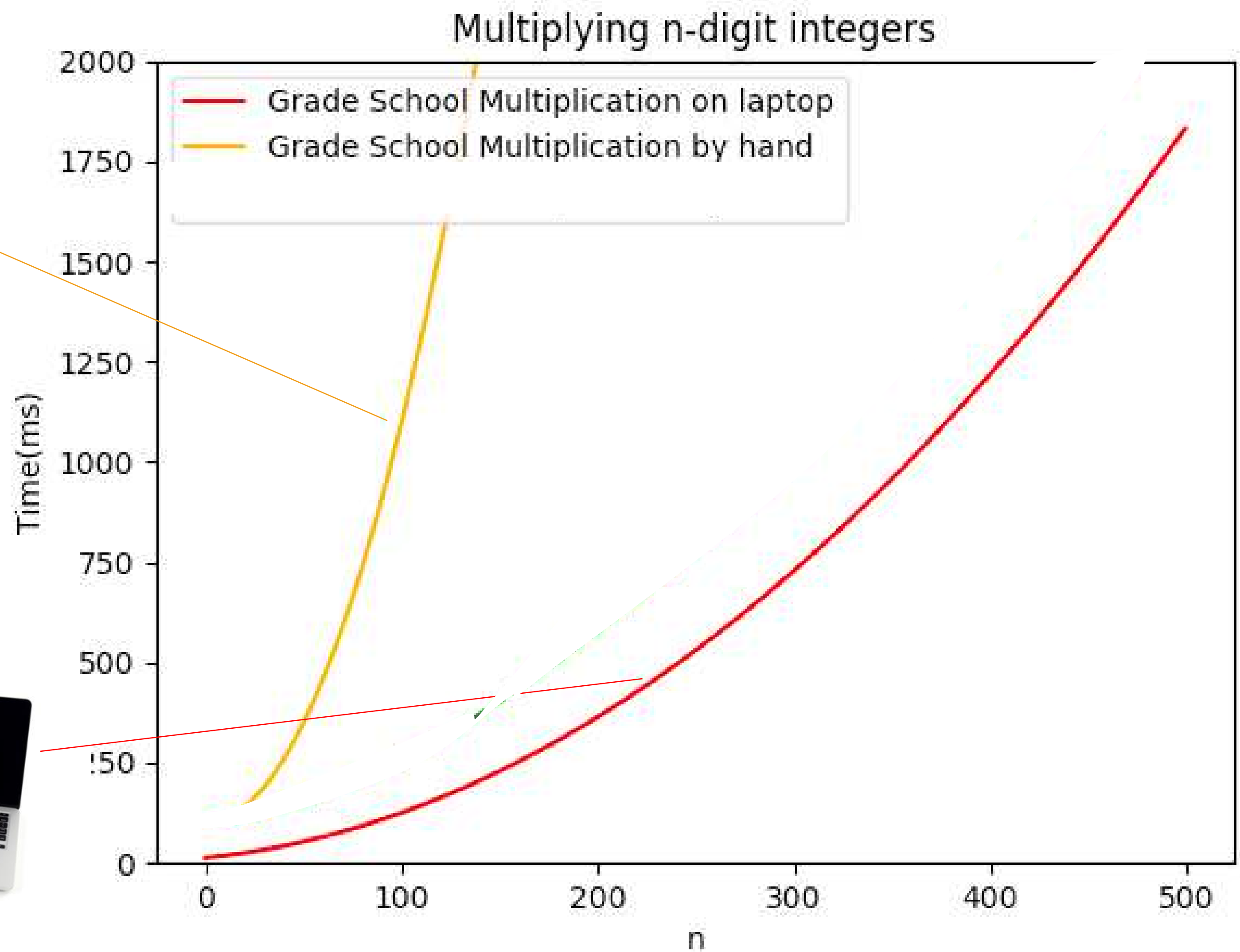
The runtime still "scales like" n^2



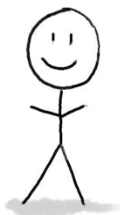
Why is big-Oh notation meaningful?



$\approx .0063n^2$

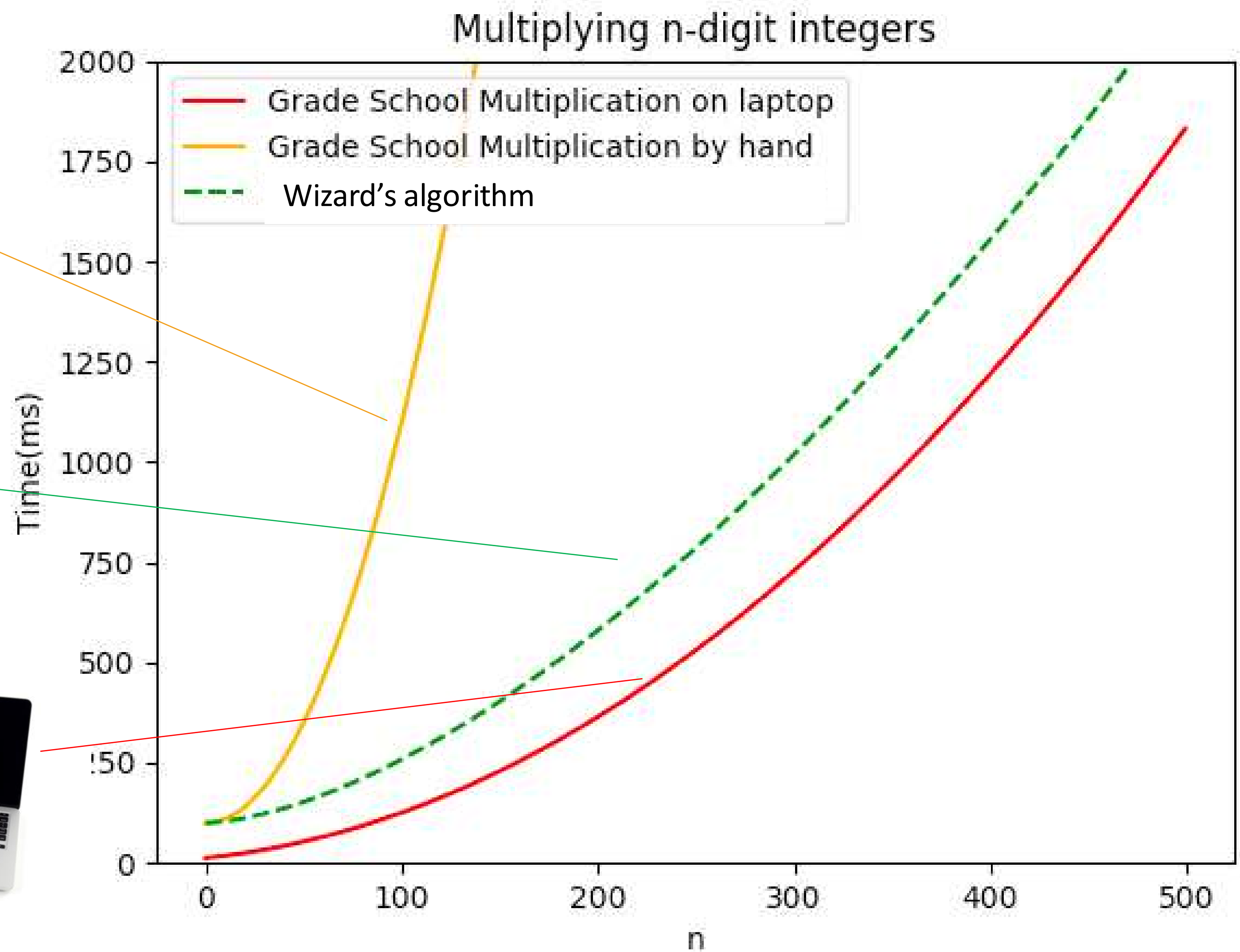


Why is big-Oh notation meaningful?

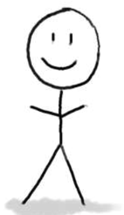


$\approx \frac{n^{1.6}}{10} + 100$

$\approx .0063n^2$

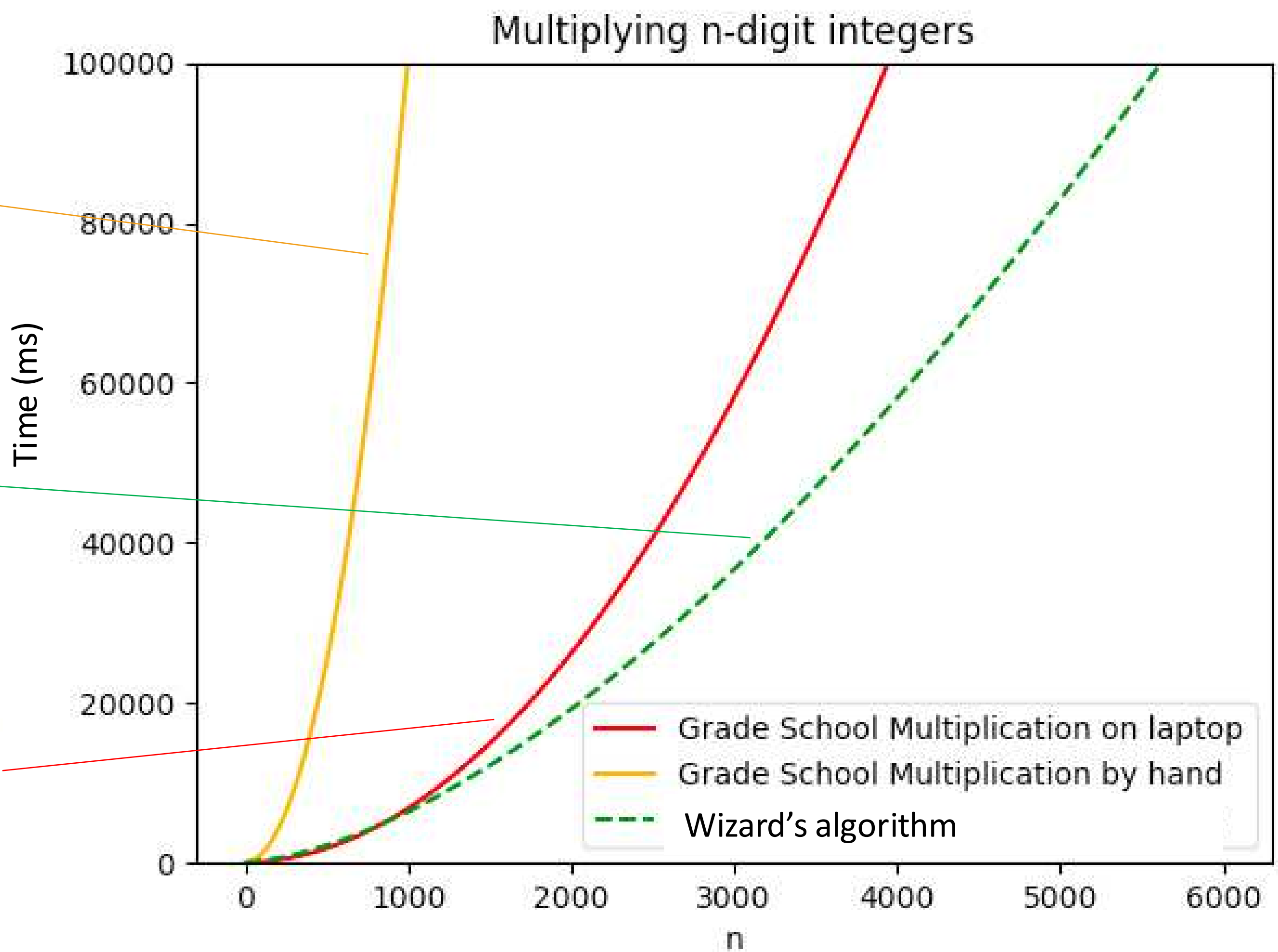


Let n get bigger…



$\approx \frac{n^{1.6}}{10} + 100$

$\approx .0063n^2$



Take-away

- An algorithm that runs in time $O(n^{1.6})$ is “better” than an an algorithm that runs in time $O(n^2)$.

- So the question is…

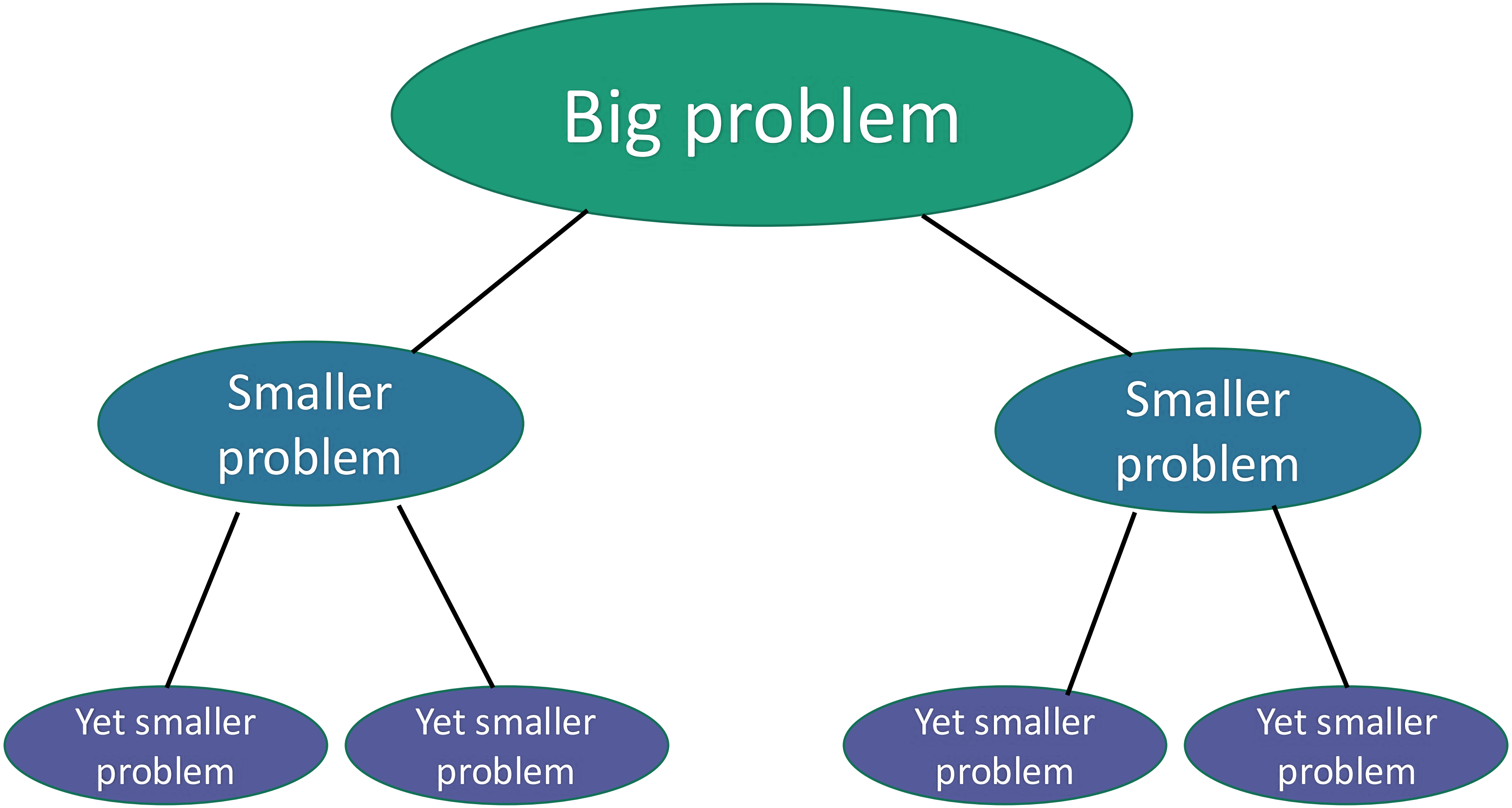
Can we do better?

Can we multiply n-digit integers faster than $O(n^2)$?

Let’s dig in to our algorithmic toolkit…

Divide and conquer

Break problem up into smaller (easier) sub-problems



Divide and conquer for multiplication

Break up an integer:

$$1234 = 12 \times 100 + 34$$

$$1234 \times 5678$$
$$= (12 \times 100 + 34)(56 \times 100 + 78)$$
$$= \underbrace{(12 \times 56)}_{1} 10000 + (\underbrace{34 \times 56}_{2} + \underbrace{12 \times 78}_{3}) 100 + \underbrace{(34 \times 78)}_{4}$$

One 4-digit multiply → Four 2-digit multiplies

More generally

Suppose n is even

Break up an n-digit integer:

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

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

$$= \underbrace{(a \times c)}_{①} 10^n + (\underbrace{a \times d}_{②} + \underbrace{c \times b}_{③}) 10^{n/2} + \underbrace{(b \times d)}_{④}$$

One n-digit multiply → Four (n/2)-digit multiplies

Divide and conquer algorithm

not very precisely…

(Assume n is a power of 2…)

x,y are n-digit numbers

Multiply(x, y):

Base case: I've memorized my 1-digit multiplication tables…

- **If** n=1:
  - **Return** xy
- Write $x = a\, 10^{\frac{n}{2}} + b$
- Write $y = c\, 10^{\frac{n}{2}} + d$

a, b, c, d are n/2-digit numbers

- Recursively compute ac, ad, bc, bd:
  - ac = **Multiply**(a, c), etc..
- Add them up to get xy:
  - xy = ac 10^n + (ad + bc) $10^{n/2}$ + bd

Make this pseudocode more detailed! How should we handle odd n? How should we implement "multiplication by 10^n"?

See the Lecture 1 Python notebook for actual code!

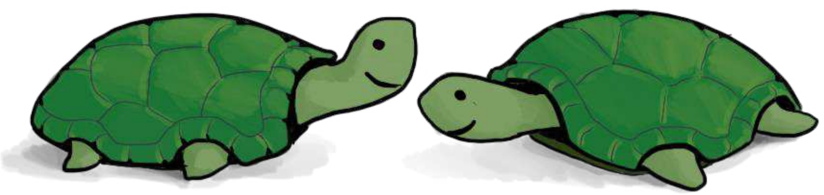
Siggi the Studious Stork

Think-Pair-Share

- We saw that this 4-digit multiplication problem broke up into four 2-digit multiplication problems

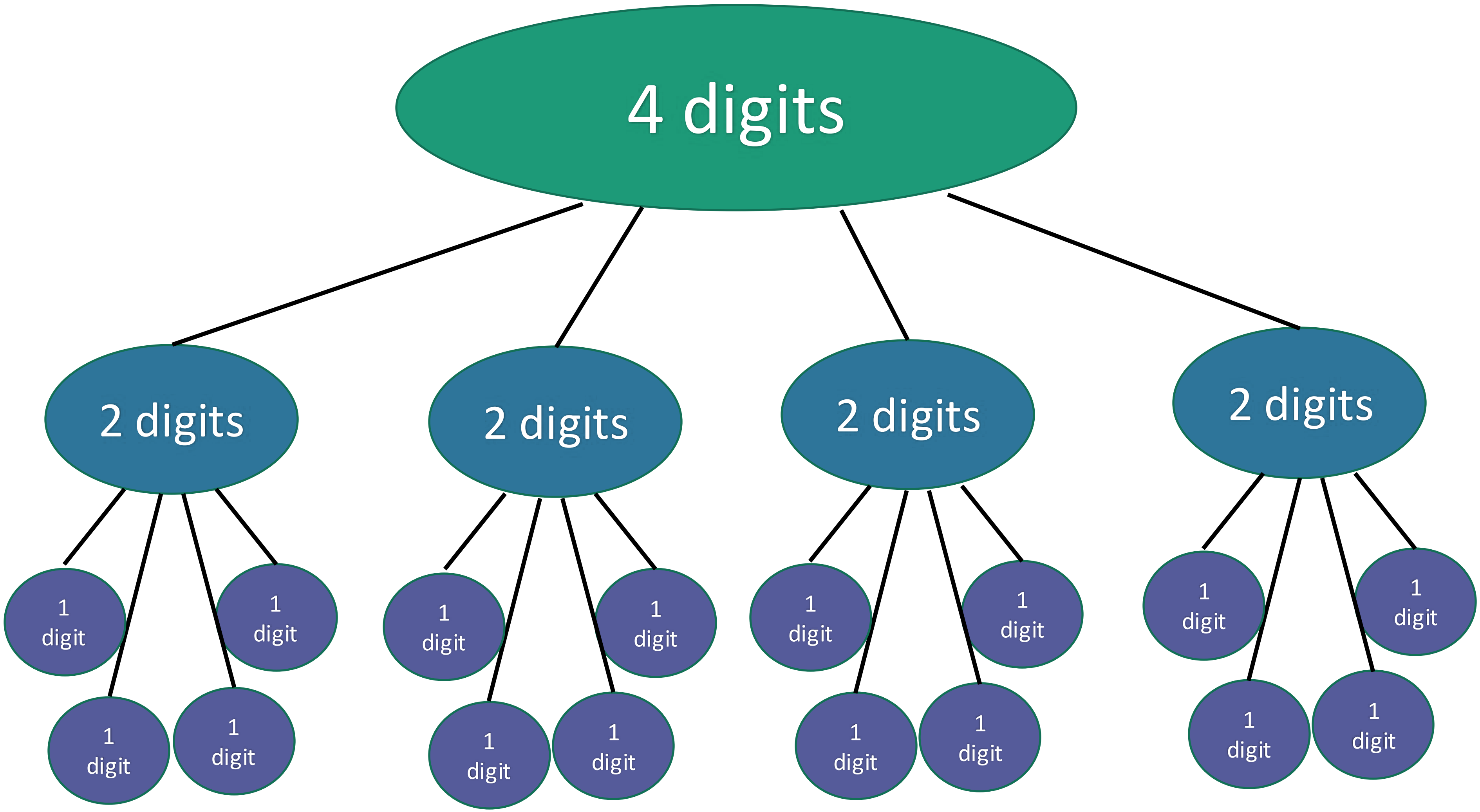
$$1234 \times 5678$$

- If you recurse on those 2-digit multiplication problems, how many 1-digit multiplications do you end up with total?



Recursion Tree

16 one-digit multiplies!



What is the running time?

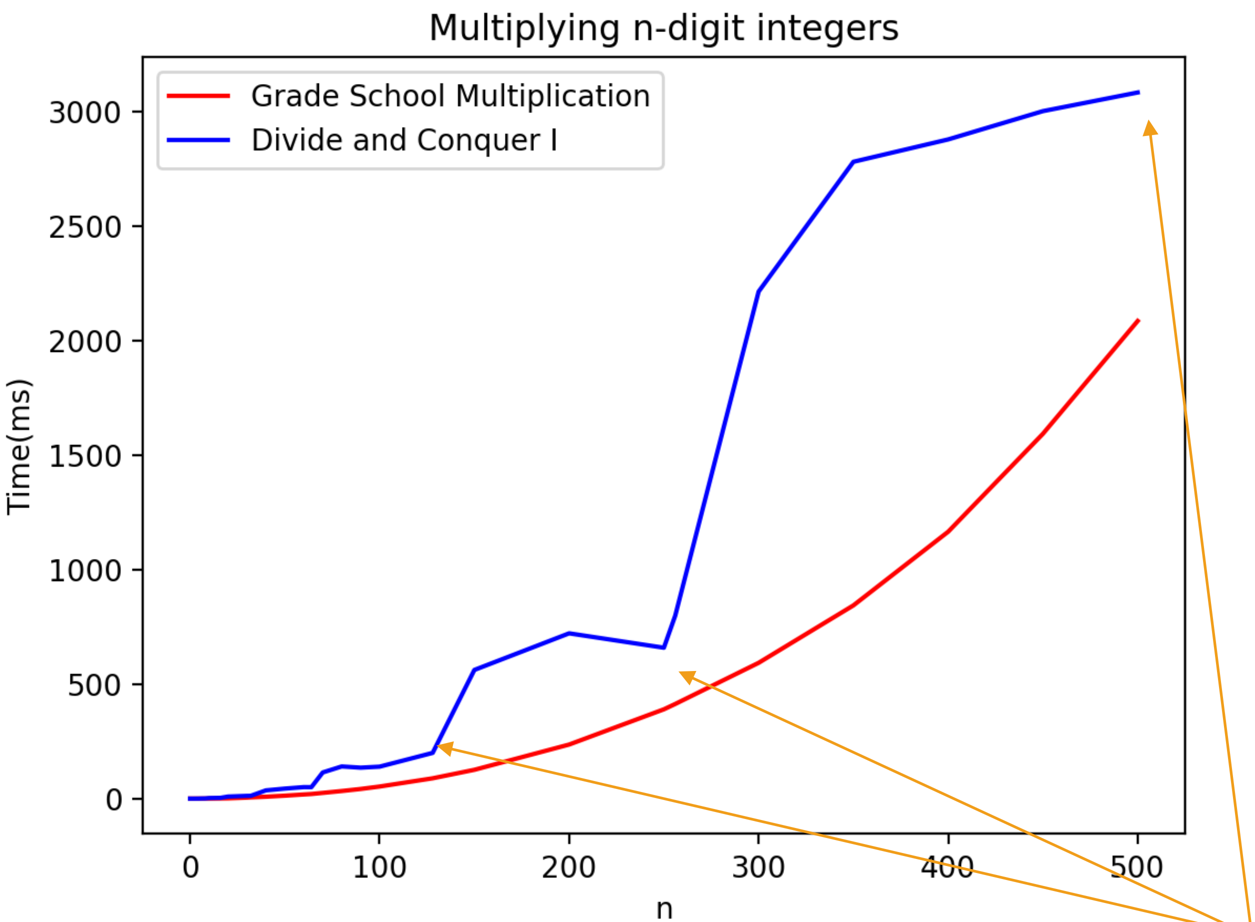
- Better or worse than the grade school algorithm?

- How do we answer this question?
  1. Try it.
  2. Try to understand it analytically.

1. Try it.

Check out the Lecture 1 IPython Notebook

Conjectures about running time?



Doesn't look too good but hard to tell…

Maybe one implementation is slicker than the other?

Maybe if we were to run it to n=10000, things would look different.

Something funny is happening at powers of 2…

2. Try to understand the running time analytically

- Proof by meta-reasoning:

It must be faster than the grade school algorithm, because we are learning it in an algorithms class.

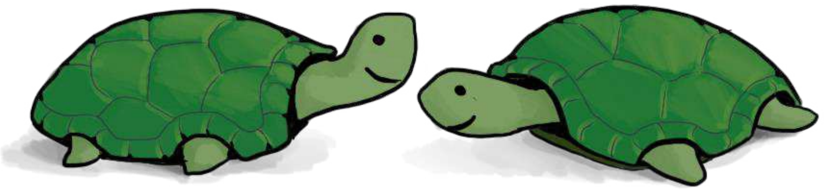
Not sound logic!

Plucky the Pedantic Penguin

2. Try to understand the running time analytically

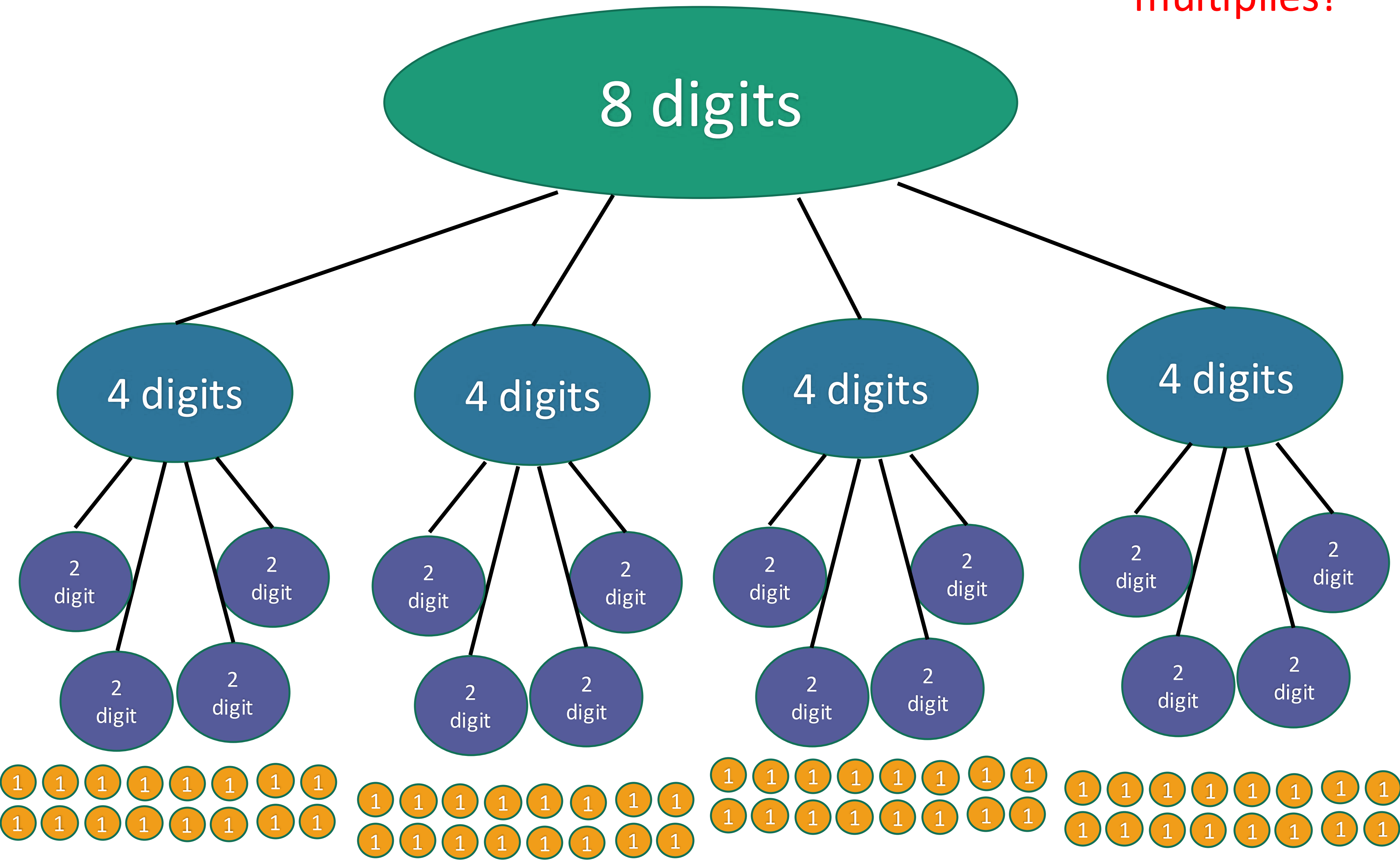
Think-Pair-Share:

- We saw that multiplying 4-digit numbers resulted in 16 one-digit multiplications.
- How about multiplying 8-digit numbers?
- What do you think about n-digit numbers?



Recursion Tree

$64 = 4^3$
one-digit
multiplies!



2. Try to understand the running time analytically

Claim:

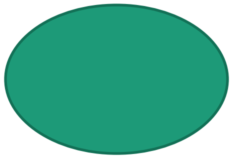
We end up doing about n^2 one-digit multiplications

$\Rightarrow$

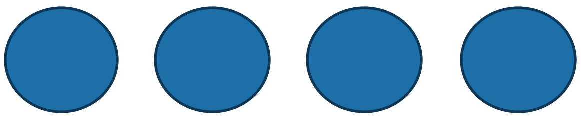
The running time of this algorithm is

AT LEAST n^2 operations.

There are n^2 1-digit problems

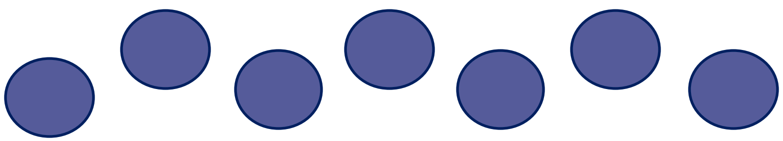


1 problem
of size n



4 problems
of size n/2

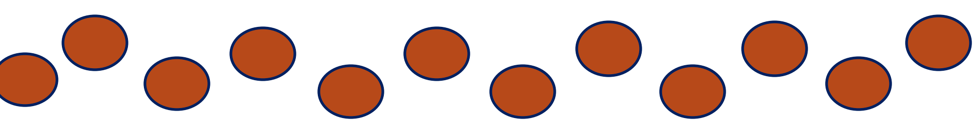
…



4^t problems
of size $n/2^t$

Note: this is just a cartoon – I'm not going to draw all 4^t circles!

…



$\underline{\;n^2\;}$ problems
of size 1

- The tree has $\log_2(n)$ levels

  $\log_2(n)$ is the number of times you cut n in half to get to down to 1.

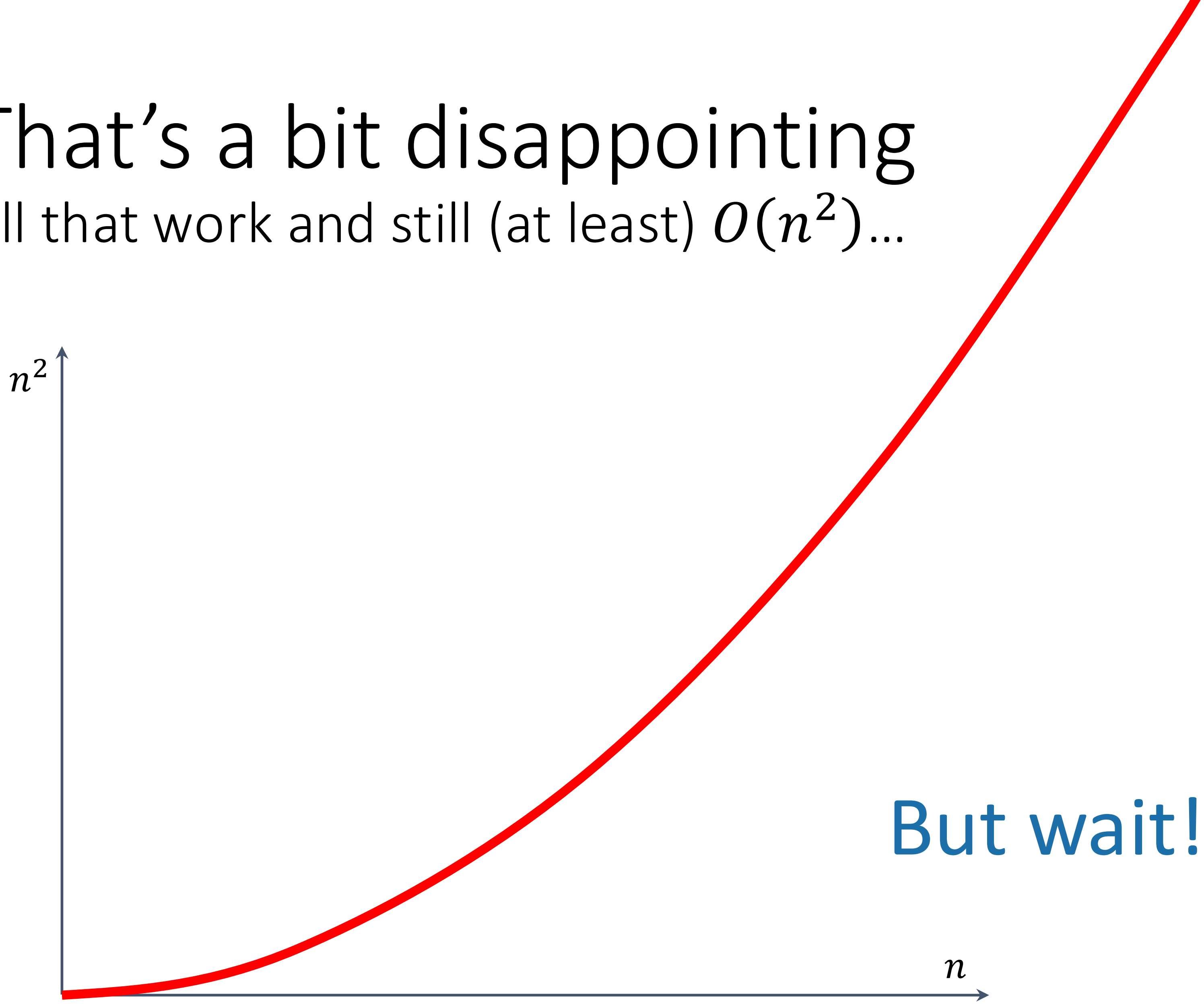
- So at level $t = \log_2(n)$ we get…

$$4^{\log_2 n} = n^{\log_2 4} = n^2$$

problems of size 1.

That’s a bit disappointing

All that work and still (at least) $O(n^2)$...



But wait!!

Divide and conquer can actually make progress

- Karatsuba figured out how to do this better!

$$
\begin{aligned}
xy &= (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d) \\
&= ac \cdot 10^{n} + (ad + bc)10^{n/2} + bd
\end{aligned}
$$

Need these three things

- If only we could recurse on three things instead of four…

Karatsuba integer multiplication

- Recursively compute these THREE things:
  - ac
  - bd
  - (a+b)(c+d)

Subtract these off

get this

(a+b)(c+d) = ac + bd + bc + ad

- Assemble the product:

$$xy = (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d)$$
$$= ac \cdot 10^{n} + (ad + bc)10^{n/2} + bd$$

✓ ✓ ✓

How would this work?

(Still not super precise, see IPython notebook for detailed code. Also, still assume n is a power of 2.)

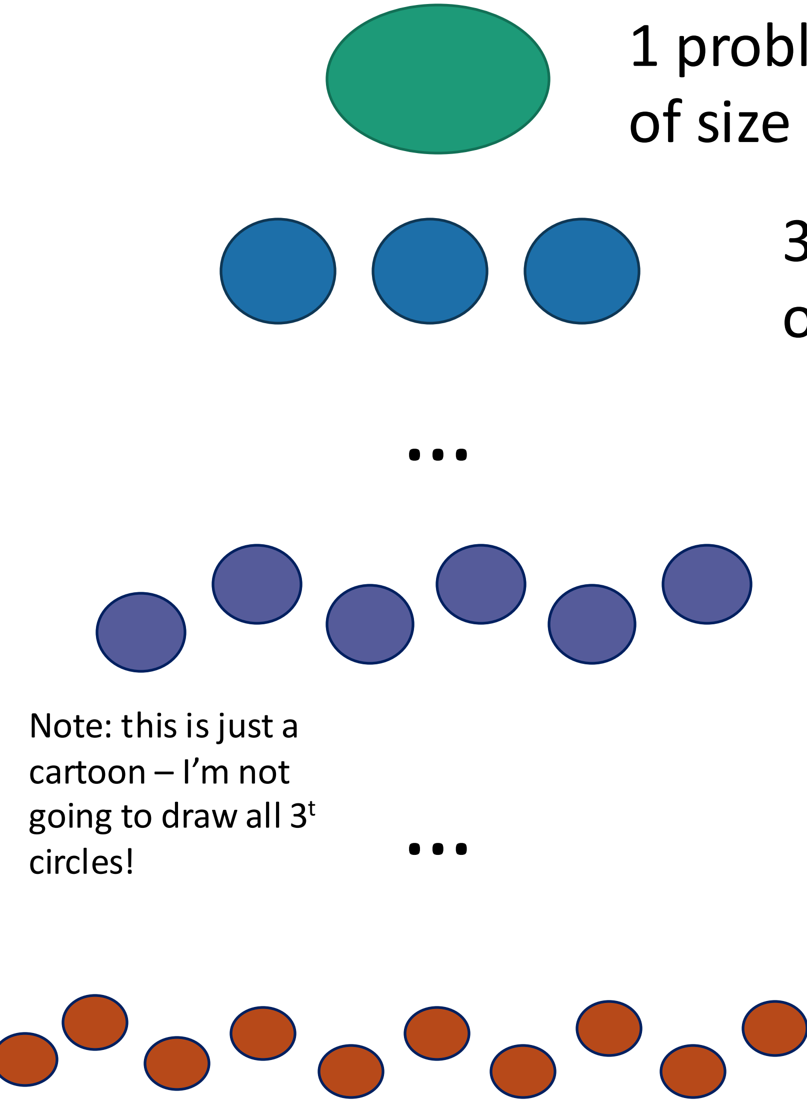
x,y are n-digit numbers

Multiply(x, y):

- **If** n=1:
  - **Return** xy
- Write $x = a\,10^{\frac{n}{2}} + b$ and $y = c\,10^{\frac{n}{2}} + d$
- ac = **Multiply**(a, c)
- bd = **Multiply**(b, d)
- z = **Multiply**(a+b, c+d)
- xy = ac 10^{n} + (z – ac - bd) $10^{n/2}$ + bd
- Return xy

a, b, c, d are n/2-digit numbers

What's the running time?



1 problem
of size n

3 problems
of size n/2

...

3^t problems
of size $n/2^t$

Note: this is just a cartoon – I'm not going to draw all 3^t circles!

...

$n^{1.6}$ ____ problems
of size 1

- The tree has
  $\log_2(n)$ levels

  $\log_2(n)$ is the number of times you cut n in half to get to down to 1.

- So at level
  $t = \log_2(n)$
  we get...

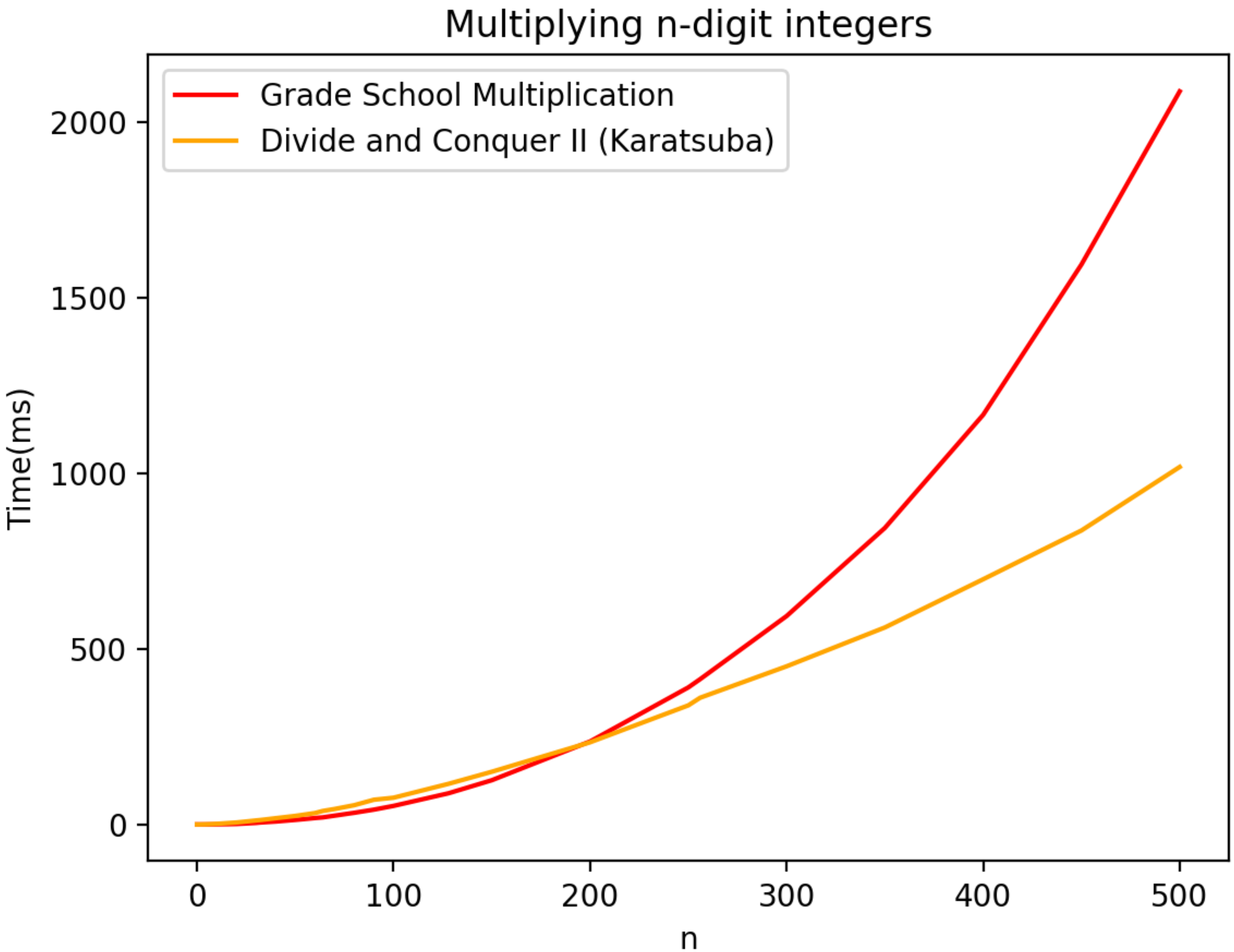
$$3^{\log_2 n} = n^{\log_2 3} \approx n^{1.6}$$

problems of size 1.

We aren't accounting for the work at the higher levels! But we'll see later that this turns out to be okay.

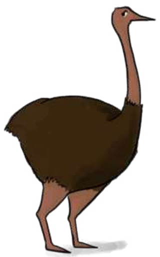
This is much better!

We can even see it in real life!



Can we do better?

- **Toom-Cook** (1963): instead of breaking into three n/2-sized problems, break into five n/3-sized problems.
  - Runs in time $O(n^{1.465})$



Try to figure out how to break up an n-sized problem into five n/3-sized problems! **(Hint: start with nine n/3-sized problems).**

Ollie the Over-achieving Ostrich

Given that you can break an n-sized problem into five n/3-sized problems, where does the 1.465 come from?

Siggi the Studious Stork

- **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 $O(n \log(n))$

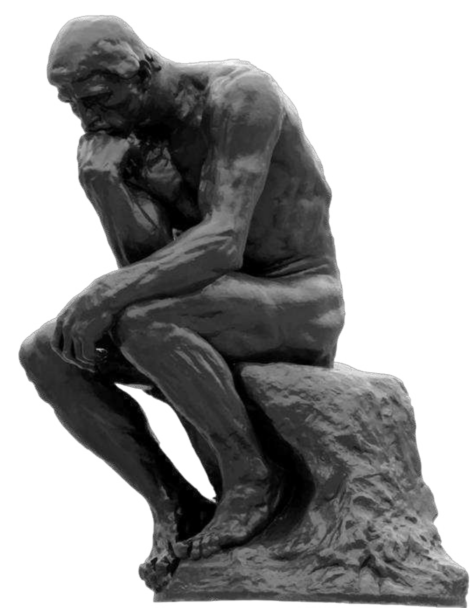
[This is just for fun, you don't need to know these algorithms!]

What we just saw

- Karatsuba Integer Multiplication
- Algorithmic Technique:
  - Divide and conquer
- Algorithmic Analysis tool:
  - Intro to asymptotic analysis

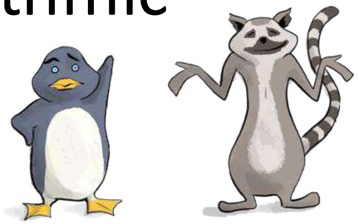
The big questions

- Who are we?
  - Professor, TA's, students?
- Why are we here?
  - Why learn about algorithms?
- What is going on?
  - What is this course about?
  - Logistics?
- Can we multiply integers?
  - And can we do it quickly?
- Wrap-up



Wrap up

- https://cs161-stanford.github.io/
- Algorithms are fundamental, useful and fun!
- In this course, we will develop both algorithmic intuition and algorithmic technical chops



- Karatsuba Integer Multiplication:
  - You can do better than grade school multiplication!
  - Example of divide-and-conquer in action
  - Informal demonstration of asymptotic analysis

Next time

- Sorting!
- Asymptotics and (formal) Big-Oh notation
- Divide and Conquer some more

BEFORE Next time

- ***Pre-lecture exercise!*** On the course website!
- Check out Ed!
- Get started on HW0! (On Gradescope)