CS 161
Design and Analysis of Algorithms

Lecture 1:

Logistics, introduction, and multiplication!

The big questions

- Who are we?
  - Professor, TAs, 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
- Awesome TAs:
  - Susanna Maria Baby
  - Jiaxi Chen
  - Aaron Effron
  - Ingerid Fosli
  - Noa Glaser
  - Nick Guo
  - Reyna Hulett
  - Duligur Ibeling
  - Haojun Li
  - Megha Jhunjhunwala
  - Richard Mu
  - Dana Murphy
  - Jayden Navarro
  - Shashwat Silas
  - Maxime Voisin
  - Brian Zhang
  - Anna Zhu


Susanna


Jiaxi


Aaron


Ingerid


Noa

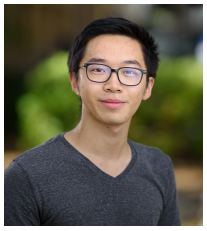

Nick


Reyna


Duligur


Haojun


Megha


Richard
(head TA)

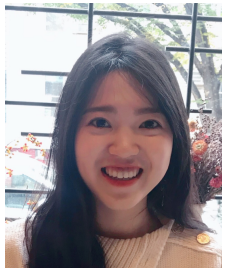

Dana
(head TA)


Jayden


Shashwat


Maxime


Brian


Anna

Who are you?

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

Concentrating in:

- Aero/Astro
- Afro-Amer. Studies
- American Studies
- Art Practice
- Biomedical Informatics
- Chemical Eng.
- Chemistry
- Chinese
- Civil & Env. Eng.
- CME
- Comp. Sci.
- Digital Humanities
- Economics
- EE
- Energy Resources Eng.
- Epidemiology
- German Studies
- Human Biology
- Law
- Math
- Modern Languages
- Music
- MS&E
- Mech. Eng.
- Philosophy
- Philosophy and religious studies
- Physics
- Public Policy
- Science, Tech. and Society
- Statistics
- Spanish
- Symbolic Systems
- 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?

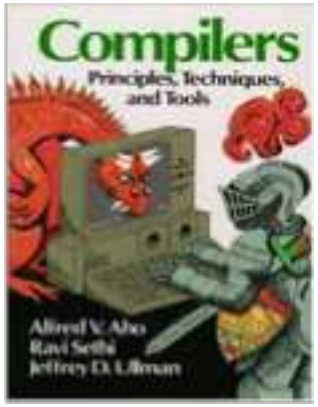
- Algorithms are fundamental.
- Algorithms are useful.
- Algorithms are fun!

Algorithms are fundamental

Operating Systems (CS 140)

229)

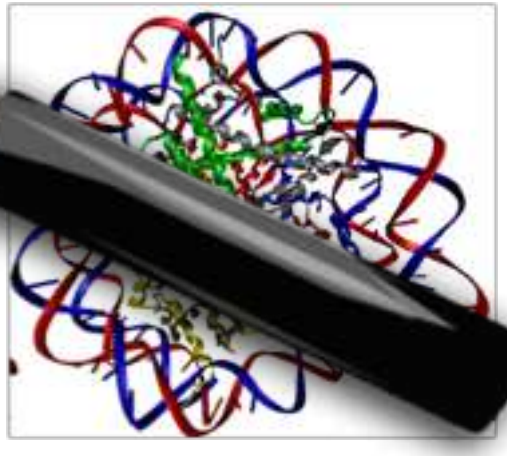
Cryptography (CS 255)



Compilers (CS 143)

The Algorithmic Lens

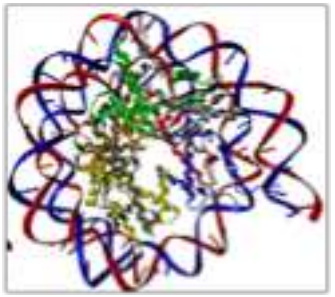
Networking (CS 144)



Computational Biology (CS 262)

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!
- A young field, many exciting research questions!

What's going on?

- Course goals/overview
- Logistics

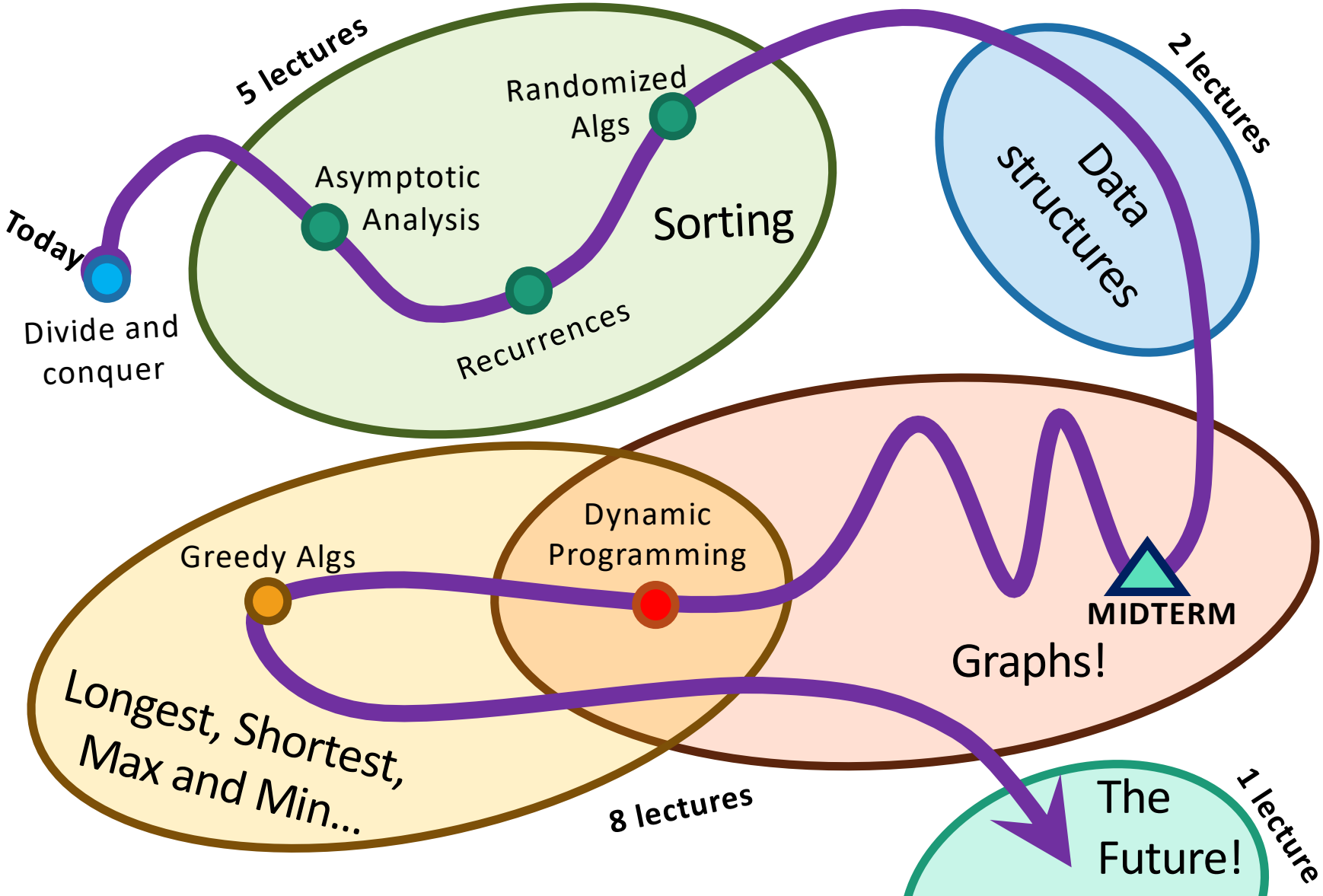
Course goals

- The design and analysis of algorithms
  - These go hand-in-hand

- In this course you will:
  - Learn to think analytically about algorithms
  - Flesh out an "algorithmic toolkit"
  - 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?

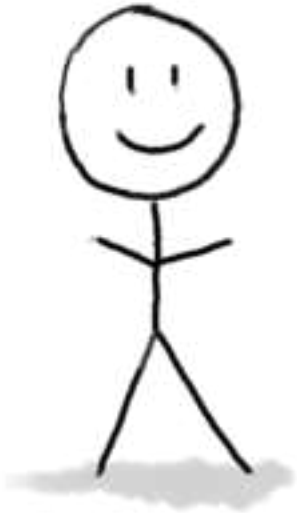
Our internal monologue…

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

Plucky the Pedantic Penguin

Detail-oriented
Precise
Rigorous

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!

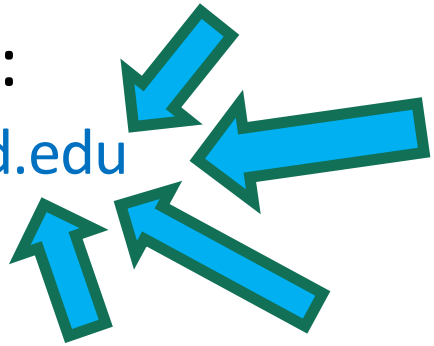
Lucky the Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

Both sides are necessary!

Course elements and resources

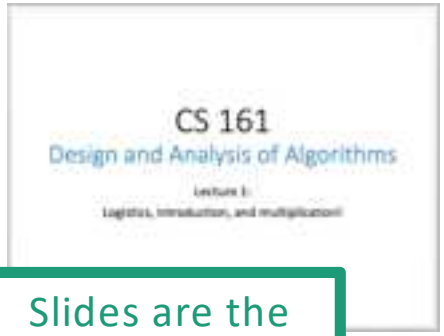
- Course website:
  - cs161.stanford.edu



- Lectures
- Textbook
- IPython Notebooks
- Homework
- Exams
- Office hours, recitation sections, and Piazza

Lectures

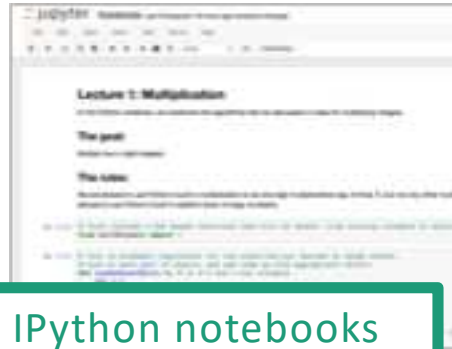
- Right here (NVIDIA Auditorium), M/W, 10:30-11:50!
- Resources available:
  - Slides, Videos, Book, IPython notebooks



Slides are the slides from lecture.

Videos from lecture are available!

Textbook has mathy details that slides may omit



IPython notebooks have implementation details that slides may omit.

- Goal of lectures:
  - Hit the most important points of the week's material
    - Sometimes high-level overview
    - Sometimes detailed examples

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)

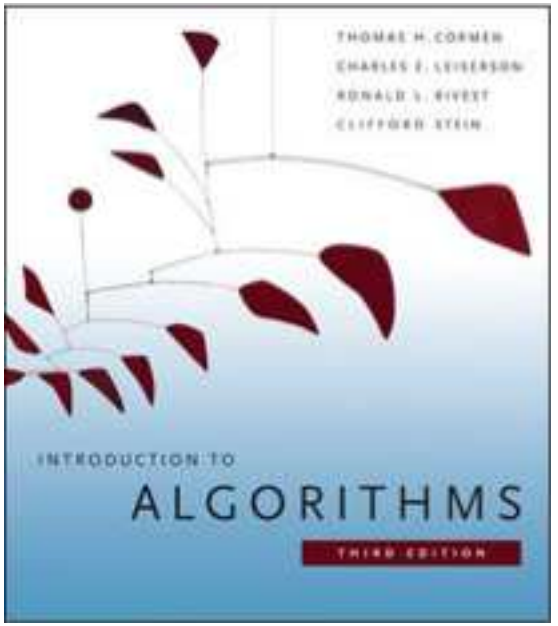
Siggi the Studious Stork
(recommended exercises)

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.**

Textbook

- **CLRS**:
  - Introduction to Algorithms, by Cormen, Leiserson, Rivest, and Stein.

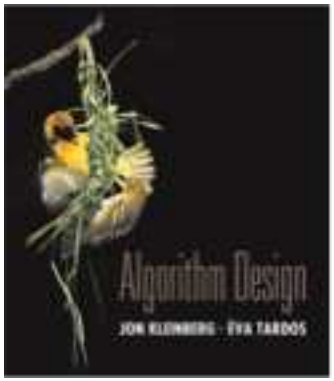


- Available ***FOR FREE ONLINE*** via

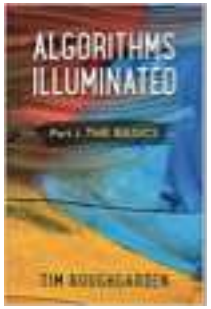
STANFORD UNIVERSITY LIBRARIES

  - Link on course website
- Hard copies on reserve at Terman Eng. library.

We will also sometimes refer to "Algorithm Design" by Kleinberg and Tardos



"Algorithms Illuminated" by Tim Roughgarden is also a great reference.



IPython Notebooks

- Lectures and homework will occasionally use IPython notebooks.
  - I will write Python code, you will read/modify it.
- However, you will need to learn ***some*** Python.
  - For next lecture, the ***pre-lecture exercise*** is to get started with Jupyter Notebooks and with Python.
  - See course website for details.

- The goal is to make the algorithms (and their runtimes) more tangible.
- It is not the goal to become a super Python programmer.
  - (Although if that happens that's cool).

Homework!

Weekly assignments in two parts:

1. Exercises:
   - Check-your-understanding and computations
   - Should be pretty straightforward
   - We recommend you do these on your own

2. Problems:
   - Proofs and algorithm design
   - Not straightforward
   - You may collaborate with your classmates
     (WITHIN REASON: See website for collaboration policy!)

How to get the most out of homework

- Do the exercises on your own.
- Try the problems on your own **before** talking to a classmate.

- If you get help from a TA at office hours:
  - **Try the problem first**.
  - Ask: **"I was trying this approach and I got stuck here."**
  - After you've figured it out, **write up your solution from scratch**, without the notes you took during office hours.

Promise

- At least one of the exam questions (on both the midterm and the final) will be very similar to a homework problem.

Exams

- There will be a **midterm** and a **final**.
  - **MIDTERM:** 2/20, in class
  - **FINAL:** 3/21, 8:30-11:30am

- We will release practice exams and hold review sessions before each.

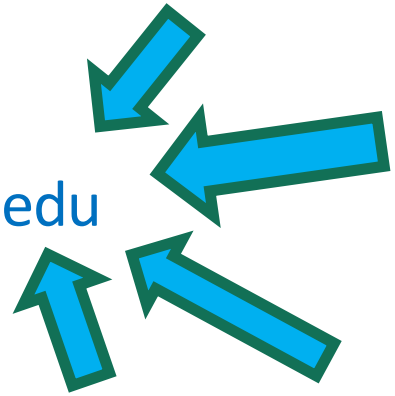
- If you have a conflict with these exams, email cs161-win1819-staff@lists.stanford.edu ASAP!!!!!

Talk to us!

- Sign up for Piazza:
  - See course website for link
  - Course announcements will be posted there
  - Discuss material with TAs and your classmates
- Office hours:
  - See course website for schedule
- Recitation sections:
  - See course website for schedule
  - Optional, but highly recommended!
  - Extra practice with the material, example problems, etc.

Course elements and resources

- Course website:
  - cs161.stanford.edu



- Lectures
- Textbook
- IPython Notebooks
- Homework
- Exams
- Office hours, recitation sections, and Piazza

Bug bounty!

- We hope all PSETs and slides will be bug-free.
- Howover, I sometmes maek typos.
- **If you find a typo** (that affects understanding*) on slides, IPython notebooks, Section material or PSETs:
  - Let us know! (Email marykw or post on Piazza).
  - 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.

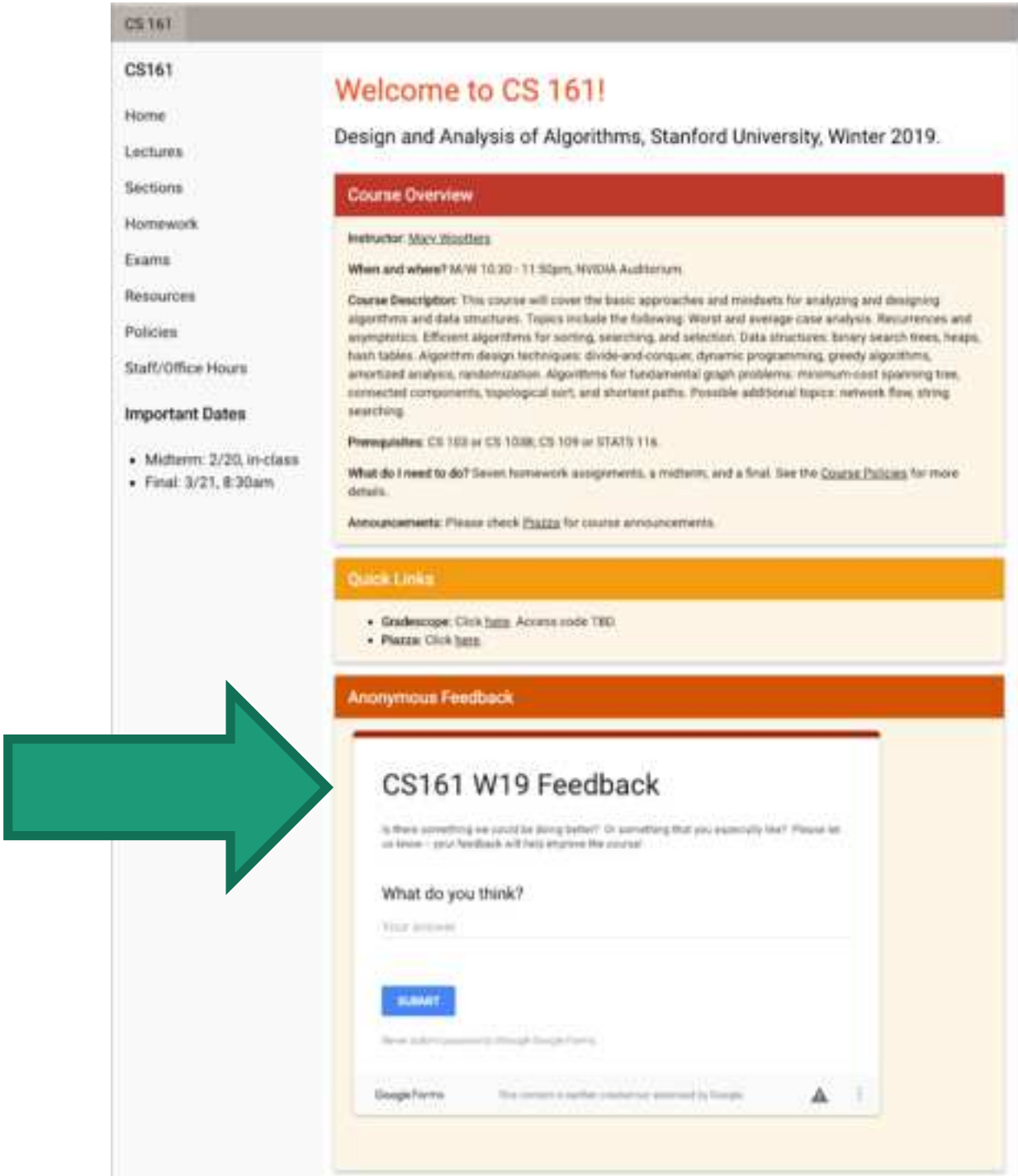
For SCPD Students

- There will be office hours held online, evenings and weekends.
- One of the recitation sections will be recorded.
- See the website for more details!

Stanford | Center for Professional Development

Feedback!

- There is an anonymous Google form on the course website.
- Please help us improve the course!
- Give feedback early and often!



A note on course policies

- Course policies are listed on the website.
- Read them and adhere to them.
- That's all I'm going to say about course policies.

Everyone can succeed in this class!

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

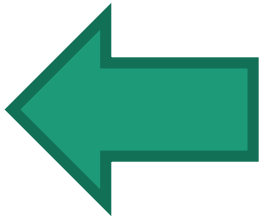
The big questions

- 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?

Course goals

- Think analytically about algorithms
- Flesh out an "algorithmic toolkit"
- Learn to communicate clearly about algorithms

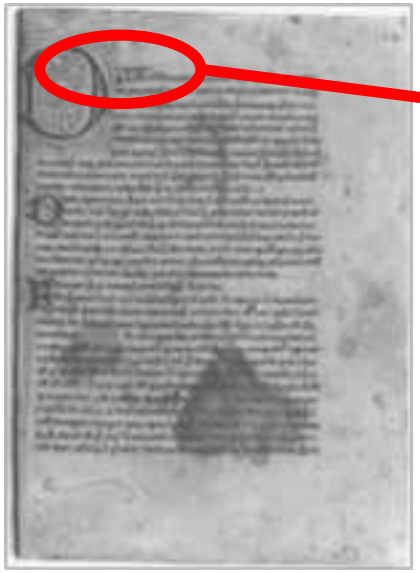
Today's goals

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

Let’s start at the beginning

Etymology of "Algorithm"

- Al-Khwarizmi was a 9th-century scholar, born in present-day Uzbekistan, who studied and 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 12th century.



Dixit algorizmi
(so says Al-Khwarizmi)

- Originally, "Algorisme" [old French] referred to just the Arabic number system, but eventually it came to mean "Algorithm" as we know today.

This was kind of a big deal

XLIV × XCVII = ?

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

Integer Multiplication

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

Integer Multiplication

$$
\begin{array}{rr}
 & 12345678959314 13 \\
\times & 4563823520395533 \\
\hline
\end{array}
$$

Integer Multiplication

n

1233925720752752384623764283568364918374523856298

x 456232358234239528562346723501913075013535001375 3

???

How fast is the grade-school multiplication algorithm?

(How many one-digit operations?)

Think-pair-share Terrapins

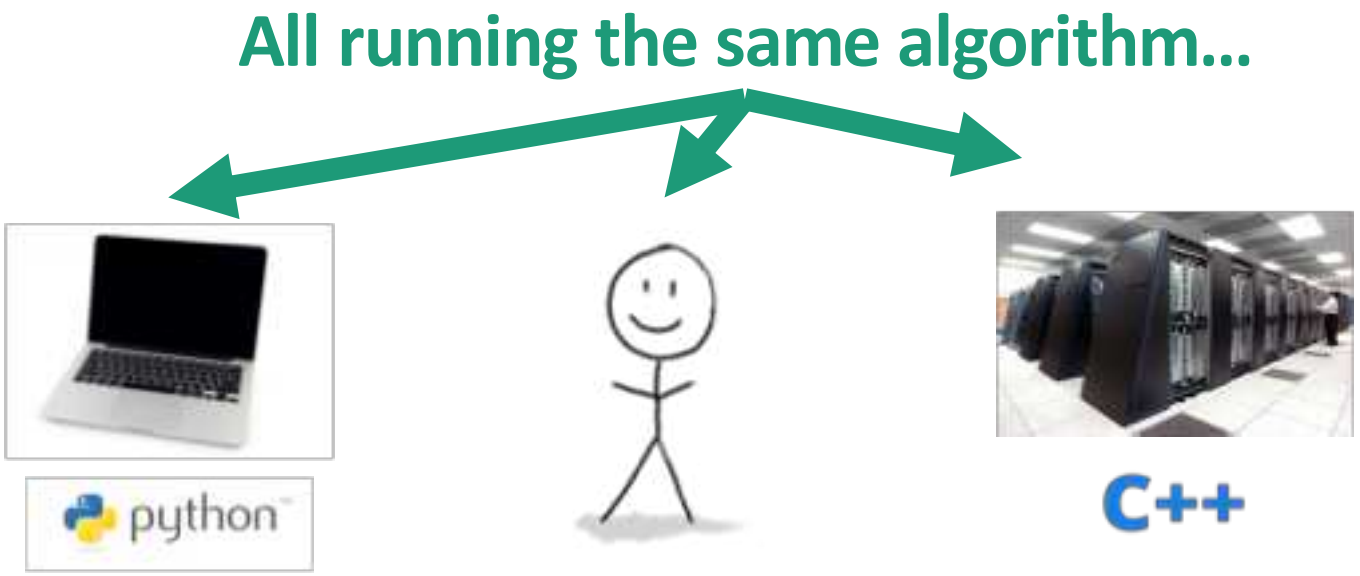
About n^2 one-digit operations

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…

Does that answer the question?

- What does it mean for an algorithm to be “fast”?

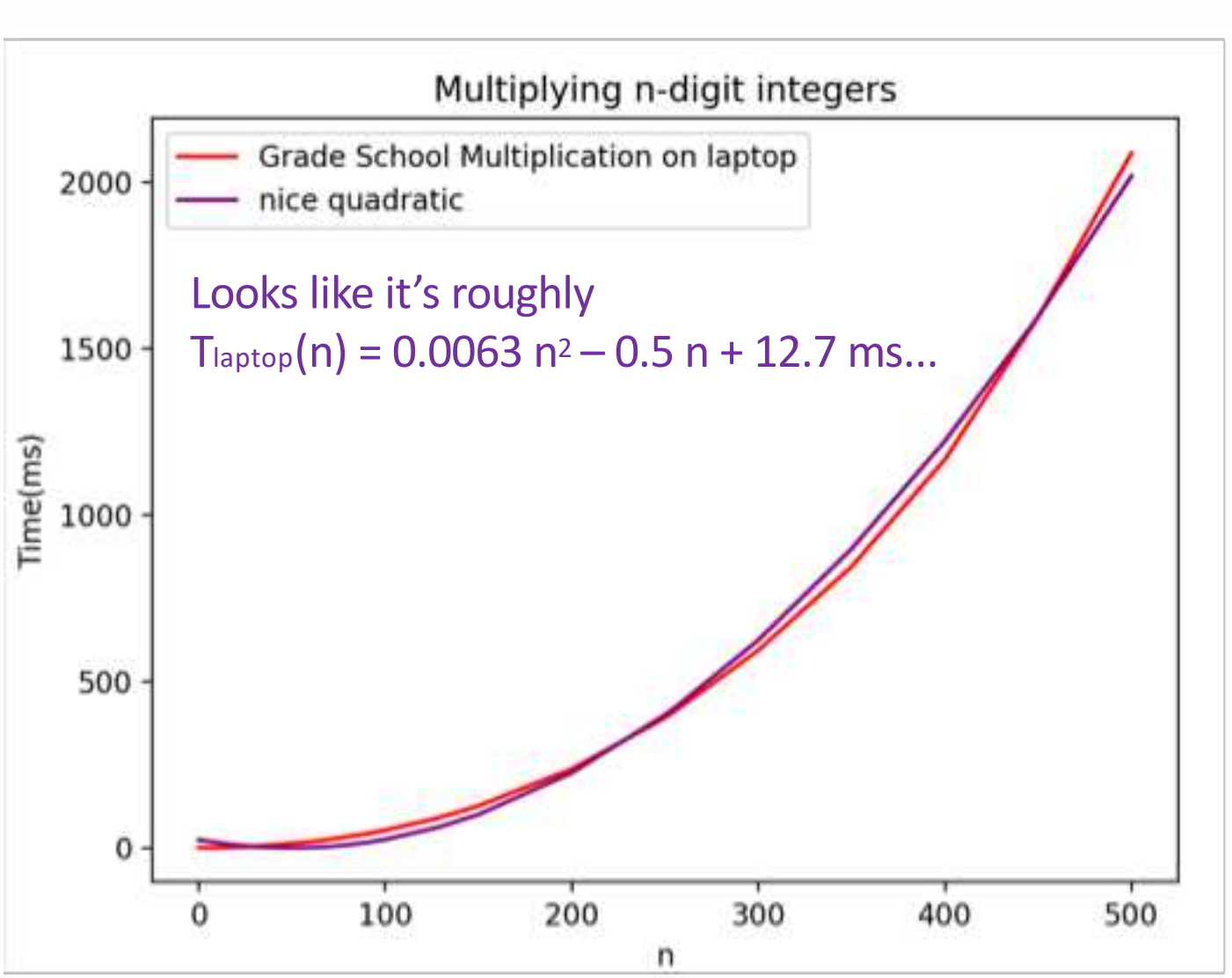


- When we say “About n^2 one-digit operations”, we are measuring how the runtime scales with the size of the input.

highly non-optimized

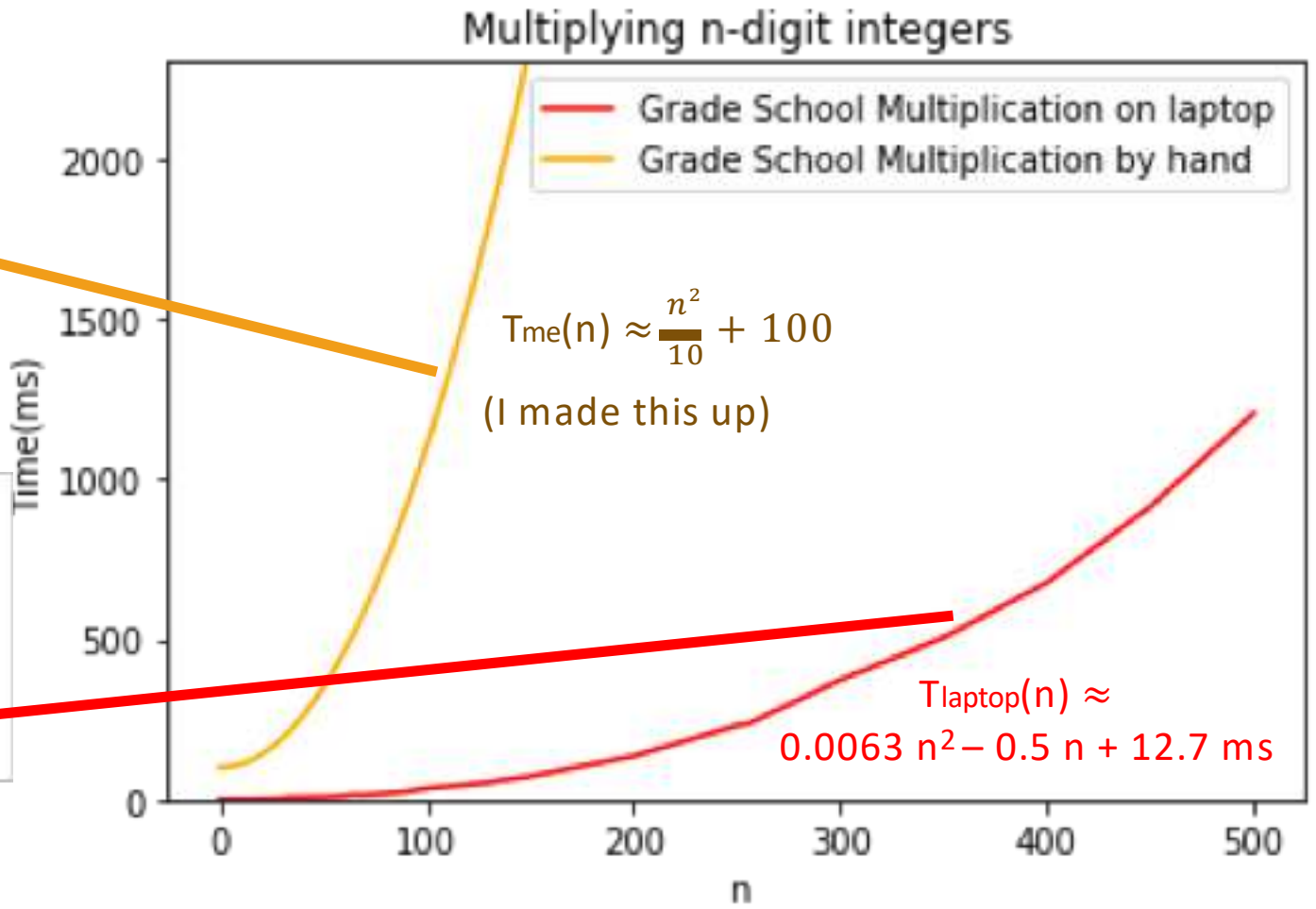
Implemented in Python, on my laptop

The runtime "scales like" n^2



Implemented by hand

The runtime still "scales like" n^2



python

Super Informally…

Asymptotic Analysis

- We will say Grade School Multiplication "runs in time $O(n^2)$"
- Formalizes what it means to "scale like n^2"
- We will see a formal definition on Wednesday.
- Informally, definition-by-example:

Number of milliseconds on an input of size n	Asymptotic Running Time
$\frac{1}{10} \cdot n^2 + 100$	$O(n^2)$
$0.063 \cdot n^2 - .5\,n + 12.7$	$O(n^2)$
$100 \cdot n^2 - 10^{10000}\sqrt{n}$	$O(n^2)$
$\frac{1}{10} n^{1.6} + 100$	$O(n^{1.6})$

(Only pay attention to the largest power of n that appears.)

We say this algorithm is "asymptotically faster" than the others.

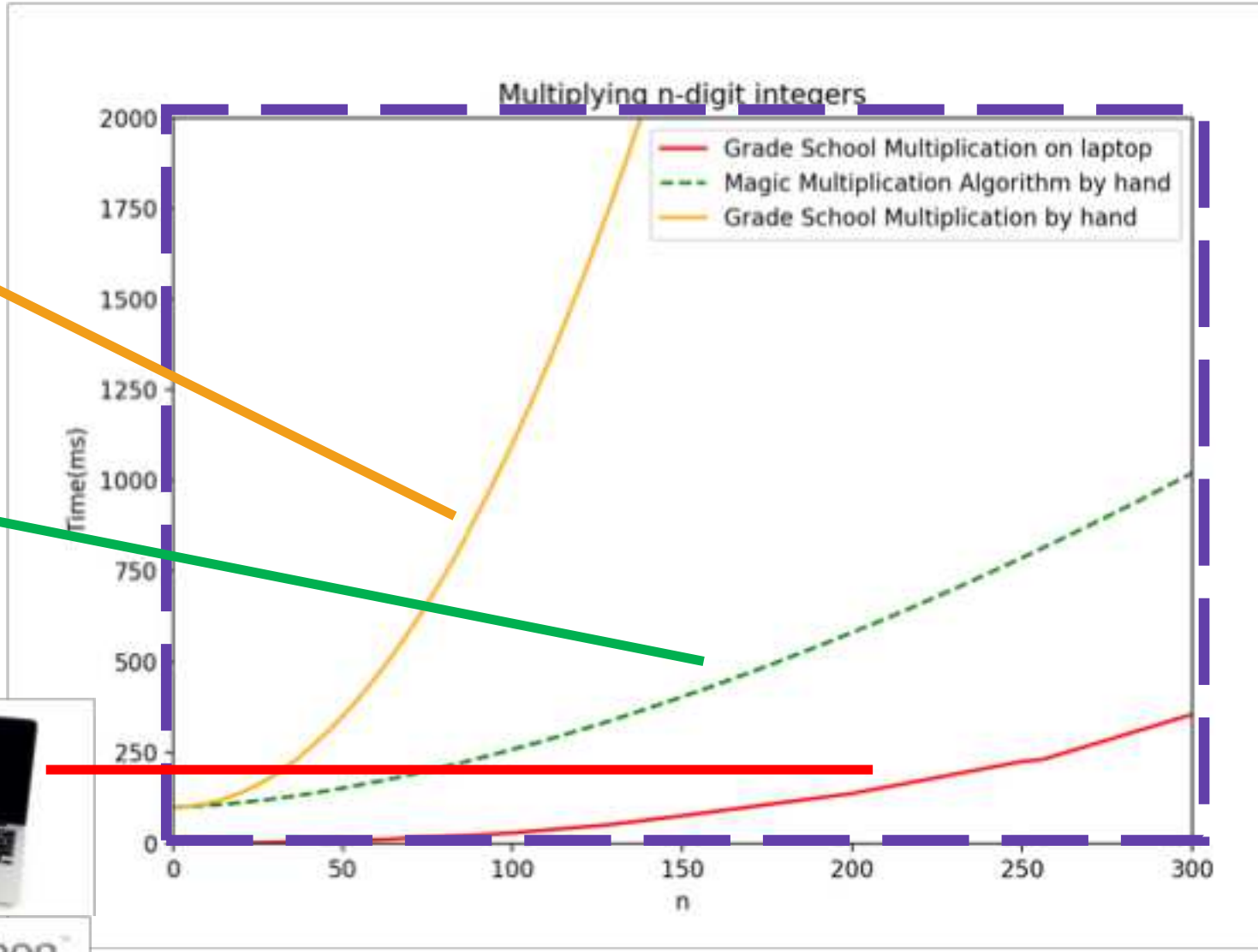
Why is asymptotic analysis meaningful?

$\approx \frac{n^2}{10} + 100$

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

$\approx .0063n^2$

python

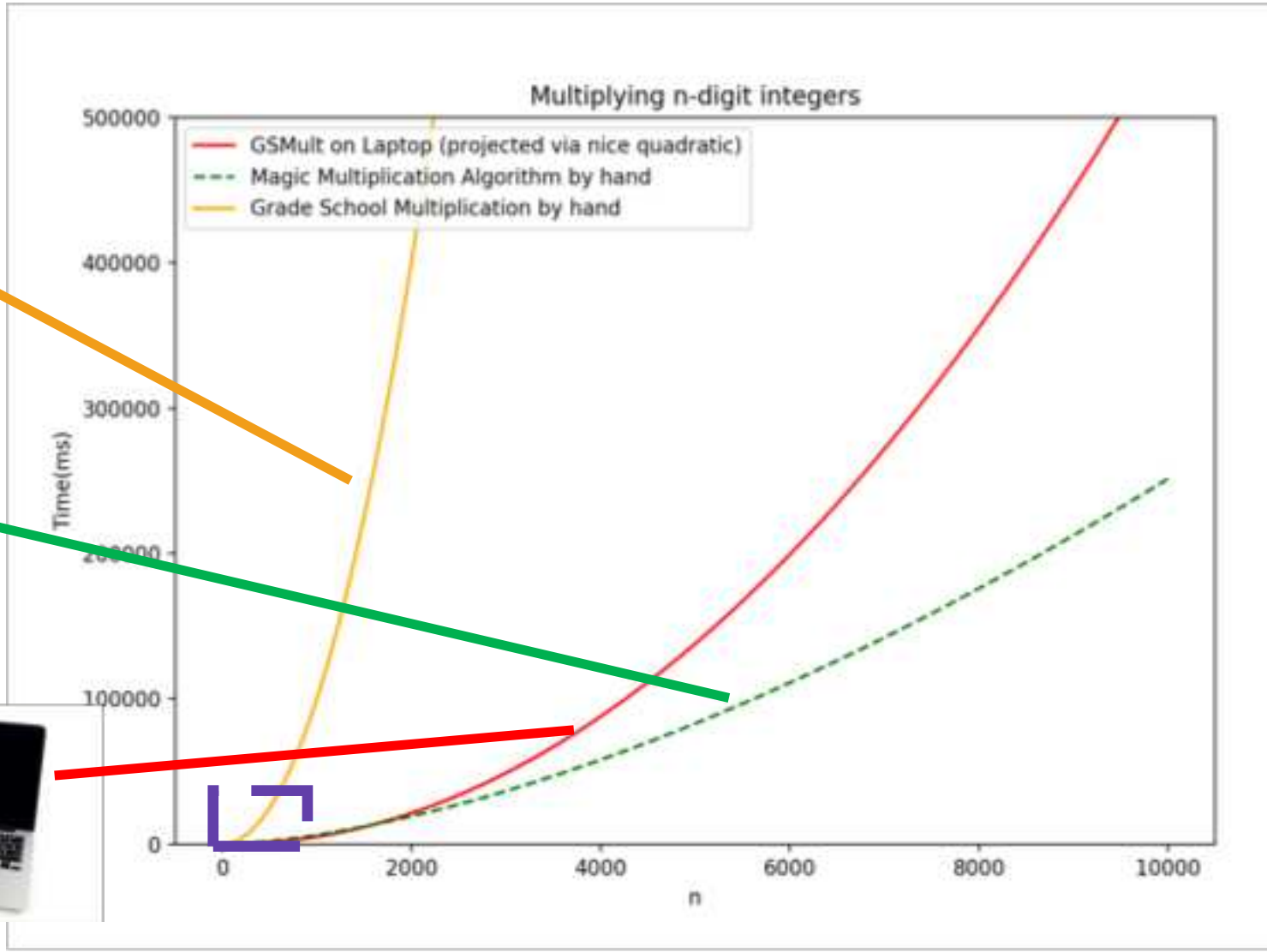


Let n get bigger…

$\approx \frac{n^2}{10} + 100$

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

$\approx .0063n^2$



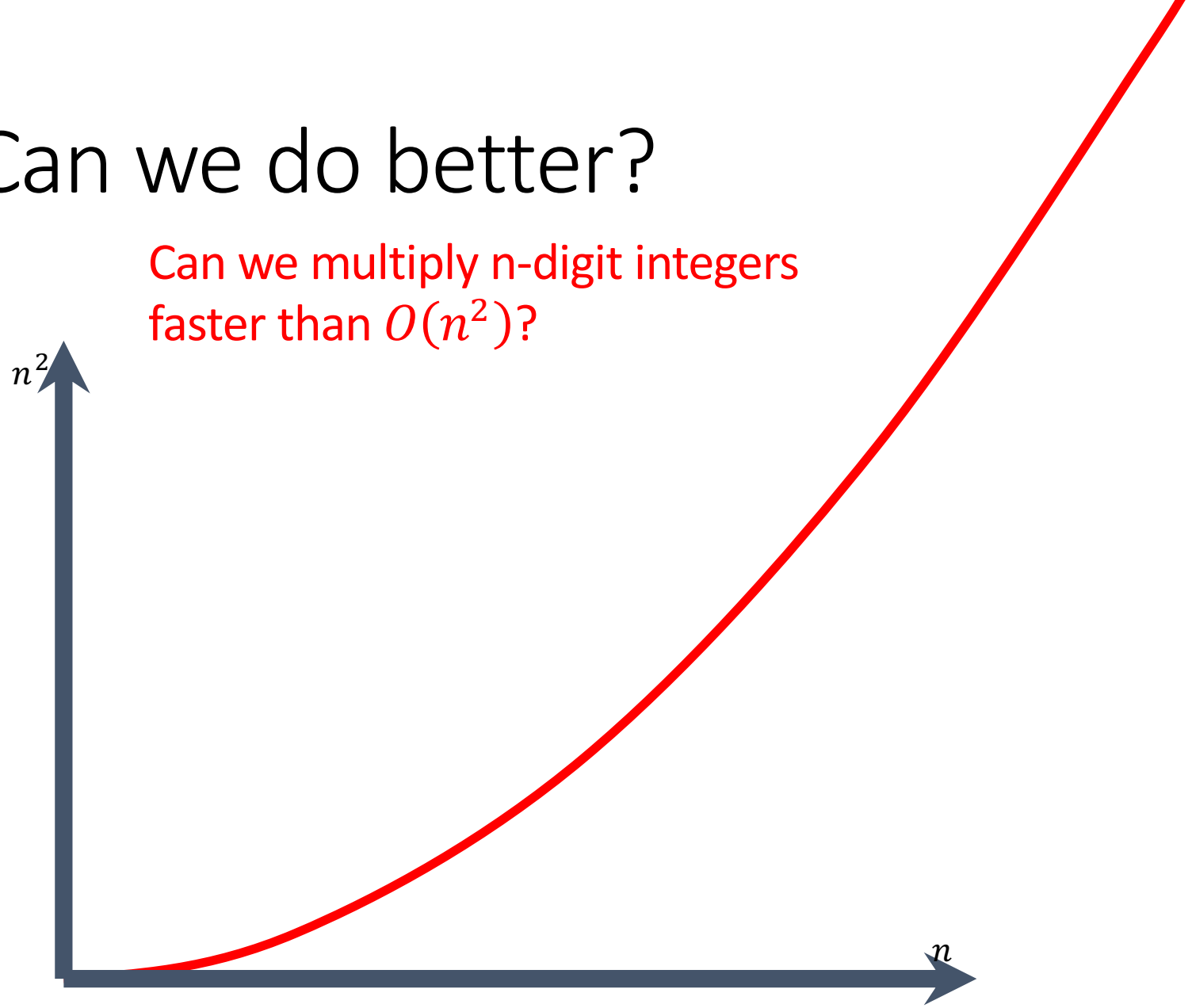
Asymptotic analysis is meaningful

- For large enough input sizes, the “asymptotically faster” will be faster than the “asymptotically slower” one, no matter what your computational platform.

- So the question is…

Can we do better?

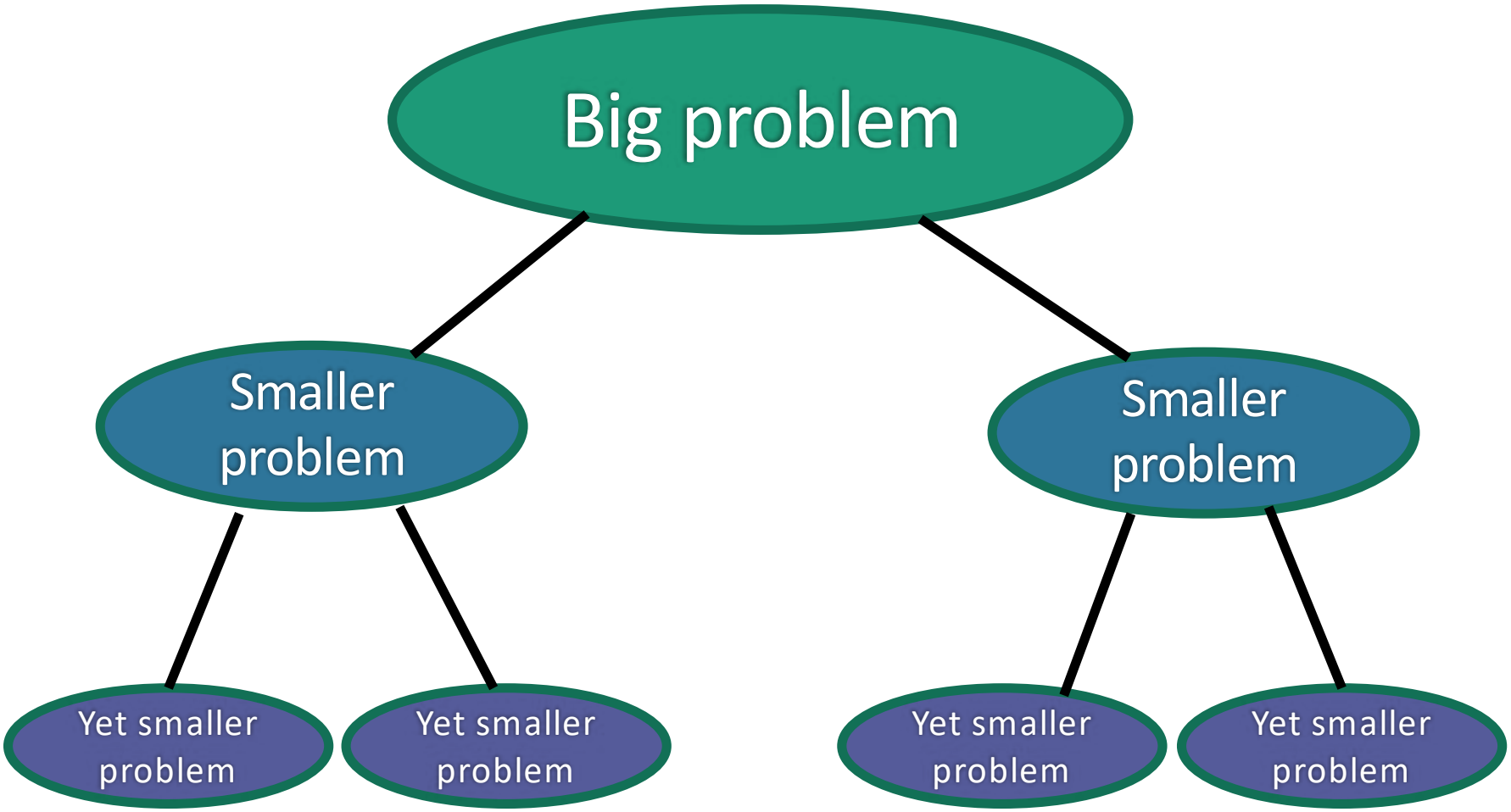
Can we multiply n-digit integers faster than $O(n^2)$?



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:

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

$$1234 \times 5678$$

$$= (12 \times 100 + 34) (56 \times 100 + 78)$$

$$= (12 \times 56) 10000 + (34 \times 56 + 12 \times 78) 100 + (34 \times 78)$$

1 2 3 4

One 4-digit multiply → Four 2-digit multiplies

More generally

Suppose n is even

Break up an n-digit integer:

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

$$\begin{aligned} x \times y &= (a \times 10^{n/2} + b)(c \times 10^{n/2} + d) \\ &= \underbrace{(a \times c)10^n}_{1} + \underbrace{(a \times d}_{2} + \underbrace{c \times b)10^{n/2}}_{3} + \underbrace{(b \times d)}_{4} \end{aligned}$$

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):

- **If** n=1:
  - **Return** xy

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

- 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!

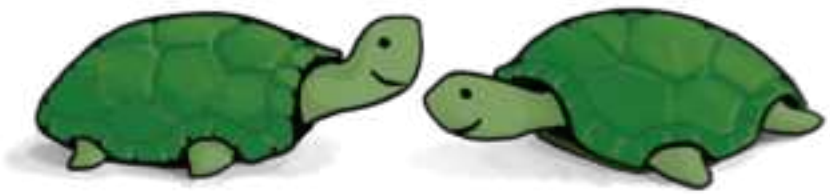
Siggi the Studious Stork

Think-Pair-Share

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

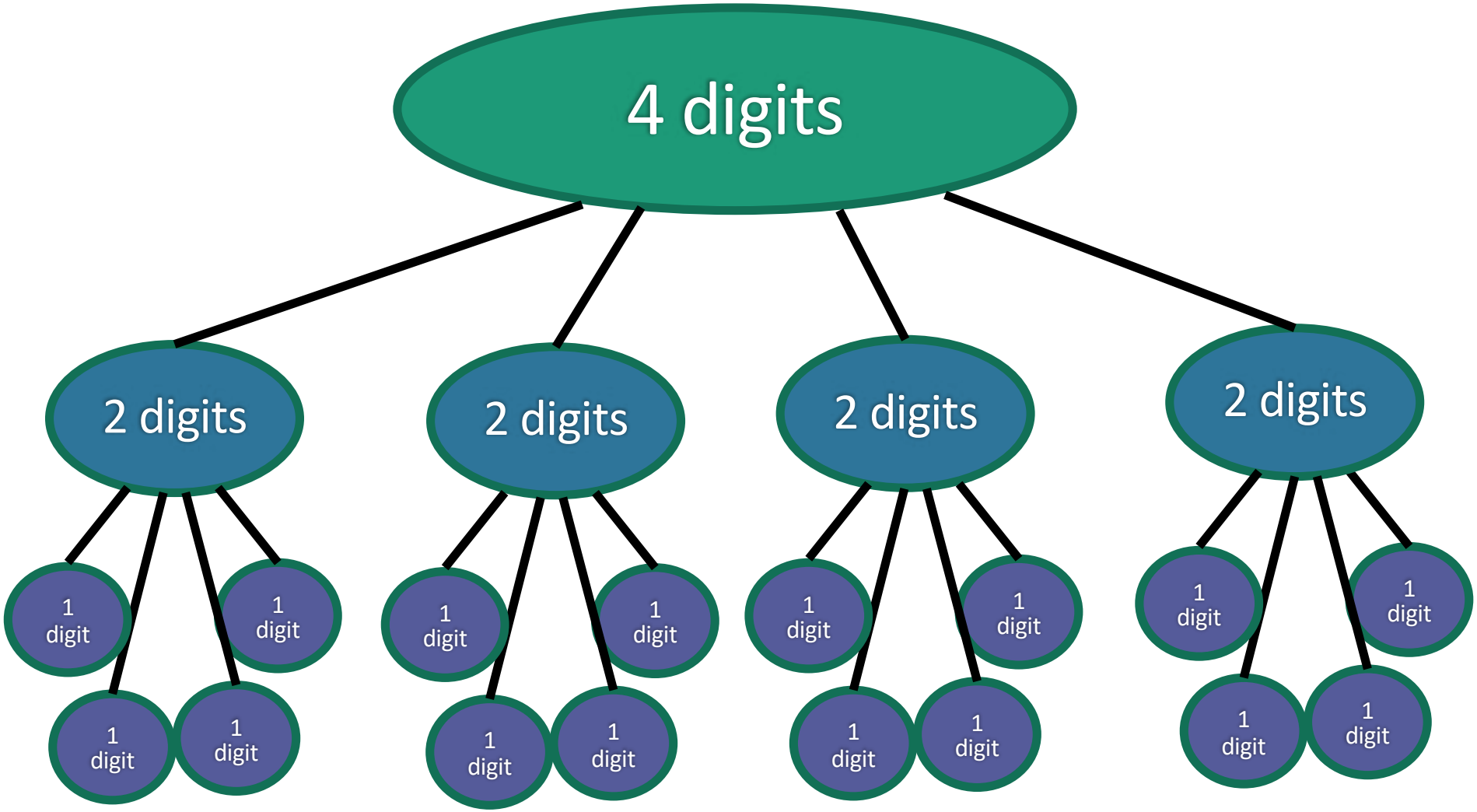
$$1234 \times 5678$$

- 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?

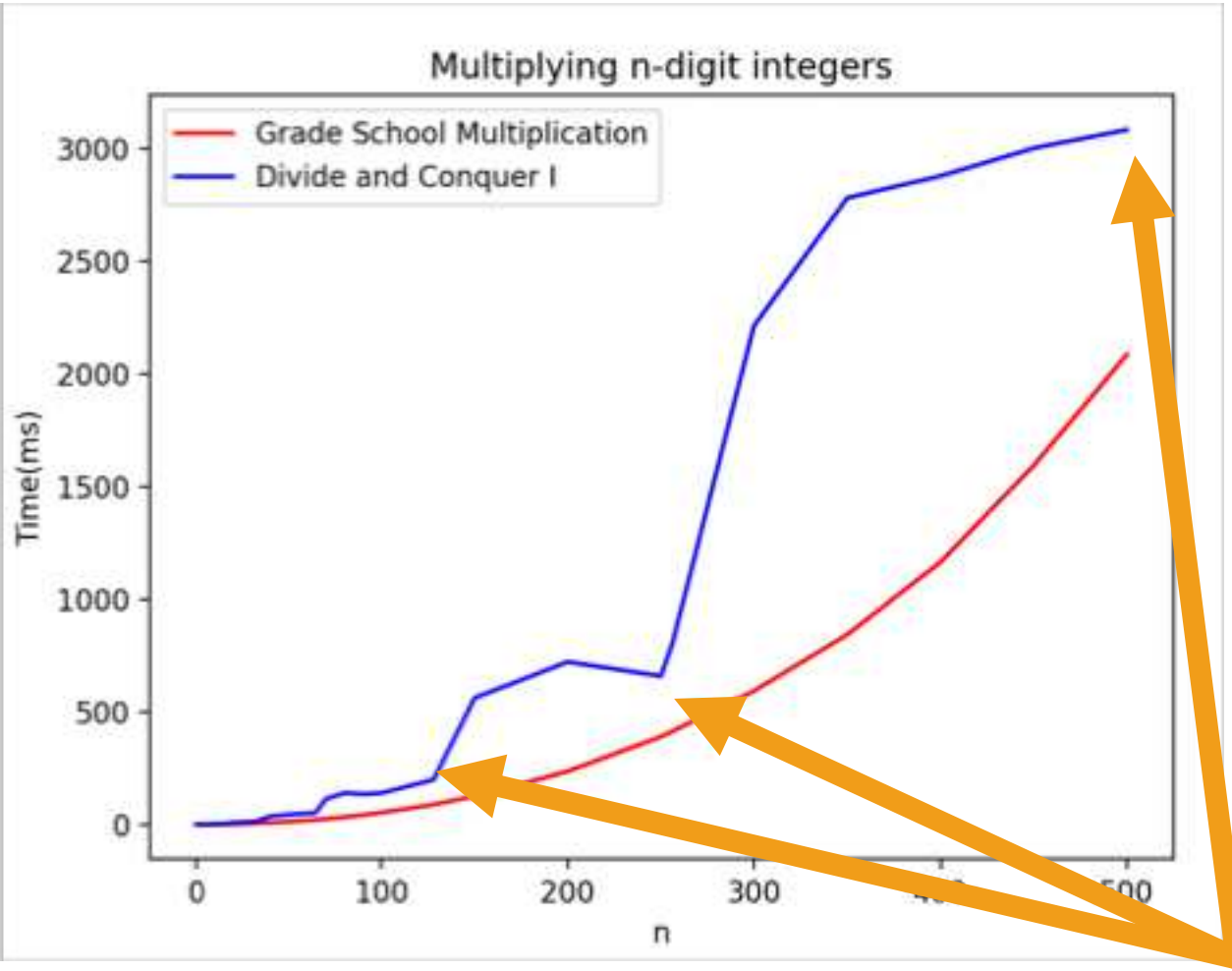
- Better or worse than the grade school algorithm?

- 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…

Concerns with the conclusiveness of this approach?

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.

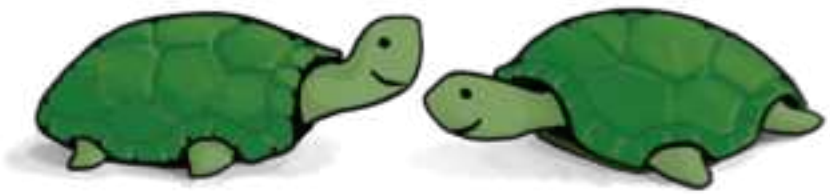
Not sound logic!

Plucky the Pedantic Penguin

2. Try to understand the running time analytically

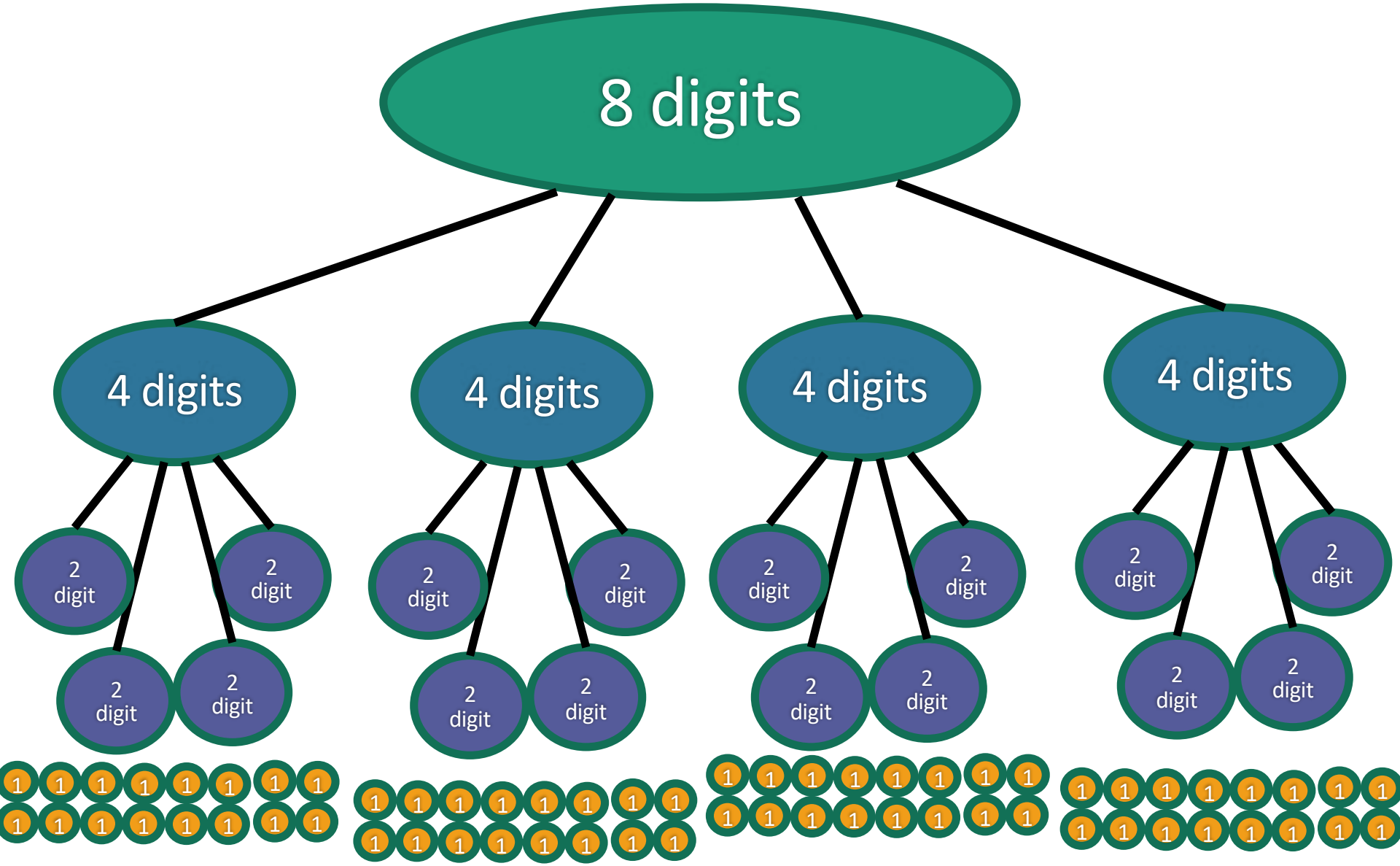
Think-Pair-Share:

- 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 one-digit multiplies!

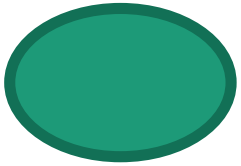


2. Try to understand the running time analytically

Claim:

The running time of this algorithm is AT LEAST n^2 operations.

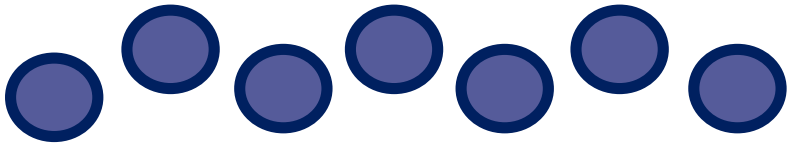
There are n^2 1-digit problems



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!

...

$\frac{n^2}{\text{}}$ problems
of size 1

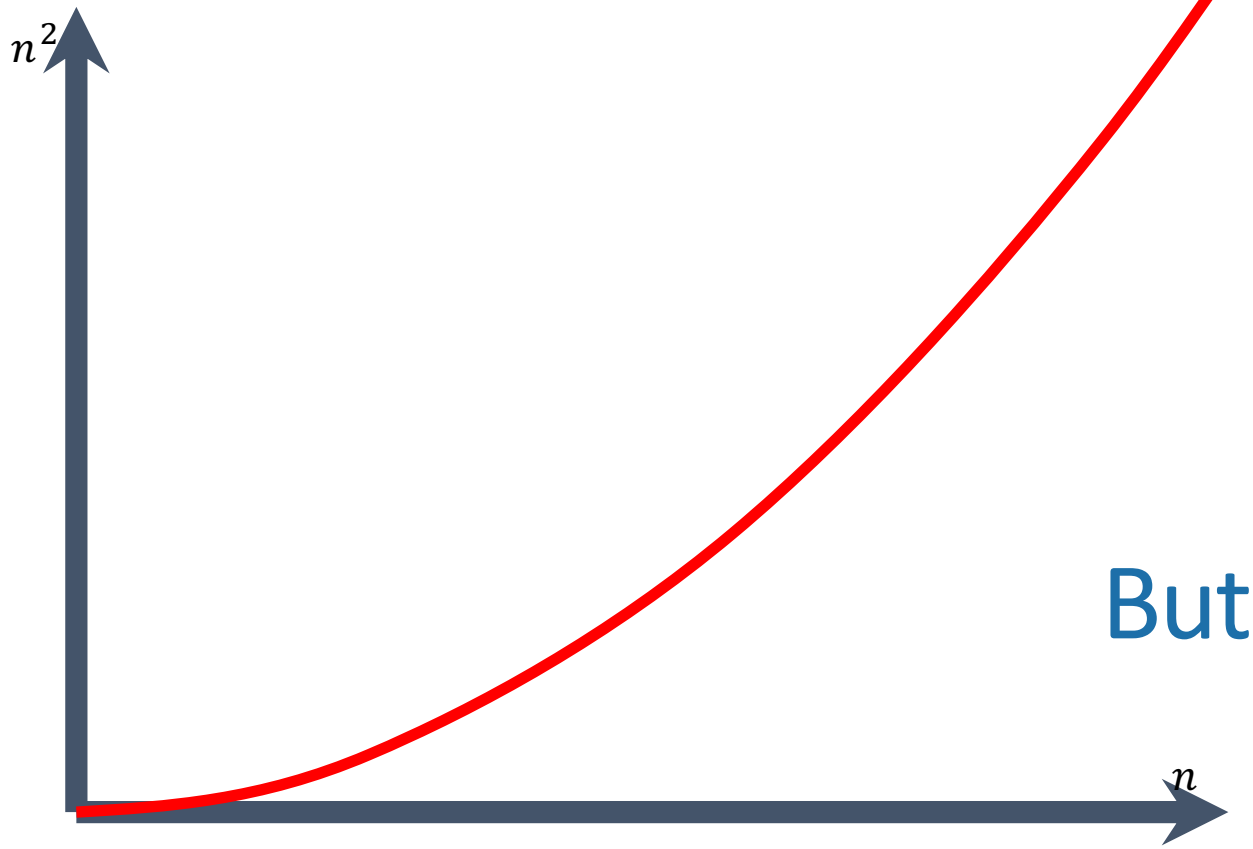
- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So we do this $\log_2(n)$ times and get…

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

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

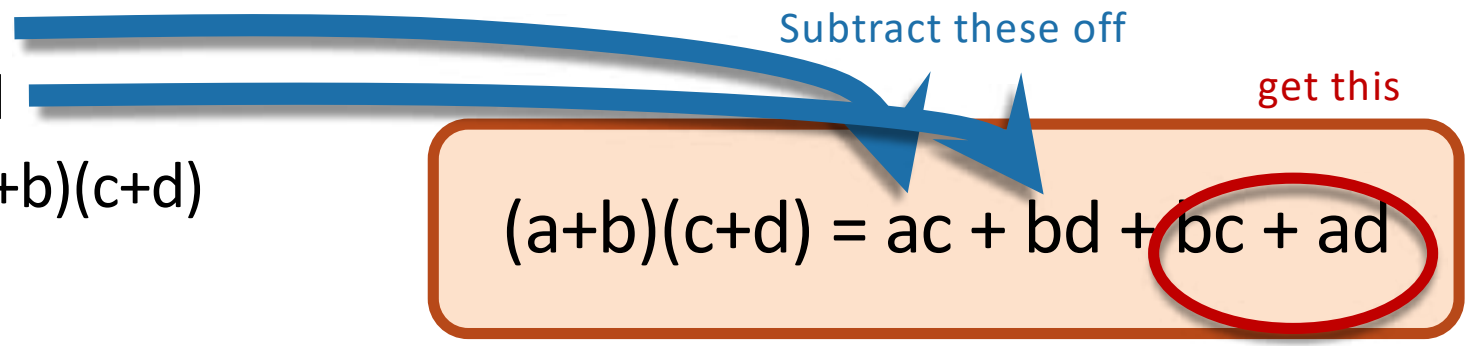
- Karatsuba figured out how to do this better!

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

- 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)



- Assemble the product:

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

THIS SLIDE ADDED FOR CLARIFICATION AFTER LECTURE

How would this work?

x,y are n-digit numbers

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

Multiply(x, y):

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

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

- ac = **Multiply**(a, c)
- bd = **Multiply**(b, d)
- z = **Multiply**(a+b, c+d)

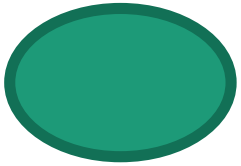
We can do the addition a+b and c+d in time O(n). This results in integers that are still roughly n/2 bits long.

- cross_terms = z - ac - bd

The quantity cross_terms is meant to be (ad + bc)

- xy = ac 10^n + (cross_terms) $10^{n/2}$ + bd
- Return xy

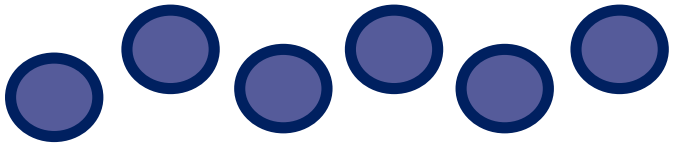
What's the running time?



1 problem of size n

3 problems of size n/2

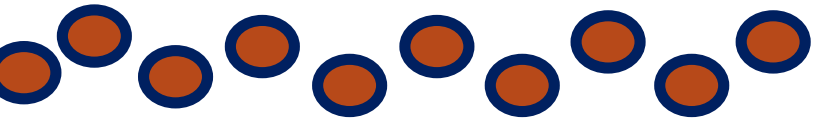
…



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

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

…



$\frac{n^{1.6}}{}$ problems of size 1

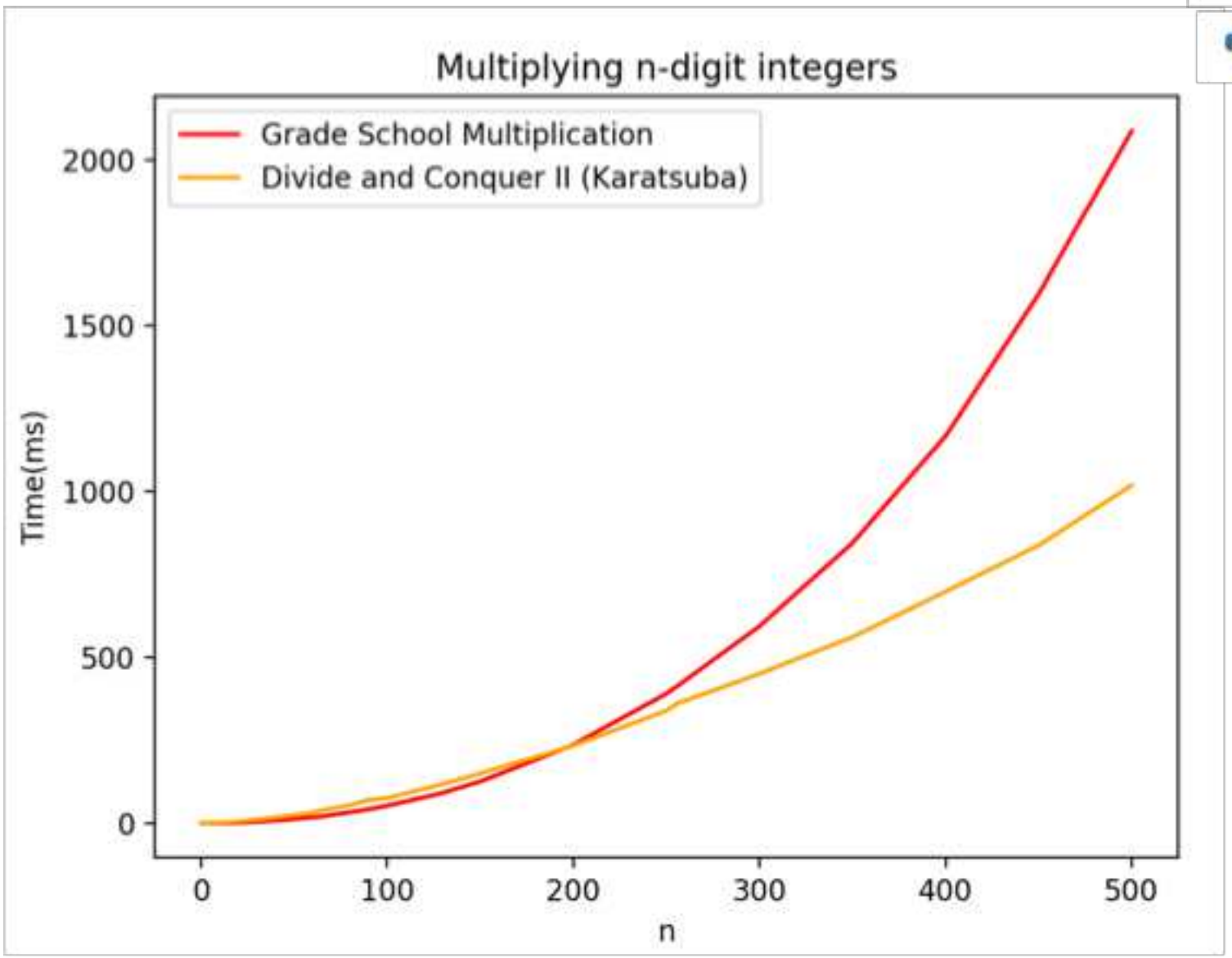
- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So we do this $\log_2(n)$ times and get…

$3^{\log_2 n} = n^{\log_2 3} \approx n^{1.6}$ 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 $n \log(n) \cdot 2^{O(\log^*(n))}$

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

Course goals

- Think analytically about algorithms
- Flesh out an "algorithmic toolkit"
- Learn to communicate clearly about algorithms

Today's goals

- 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

- cs161.stanford.edu
- 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!
- Divide and Conquer some more
- Asymptotics and (formal) Big-Oh notation

BEFORE Next time

- ***Pre-lecture exercise!*** On the course website!
- Join Piazza!