CS 161

Design and Analysis of Algorithms

Lecture 1:

Logistics, introduction, and multiplication!

Slides originally created by Mary Wootters

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?

- Professors:
  - Ellen Vitercik and Moses Charikar
- Course manager:
  - Amelie Byun


Ellen


Moses

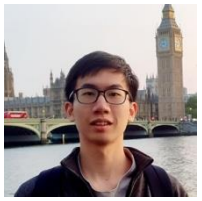

Amelie


Ziyi


Illia


Karan


Michael


Mingwei


Nikil


Ruiquan


Simon


Will


Yash


Zoe


Auddithio

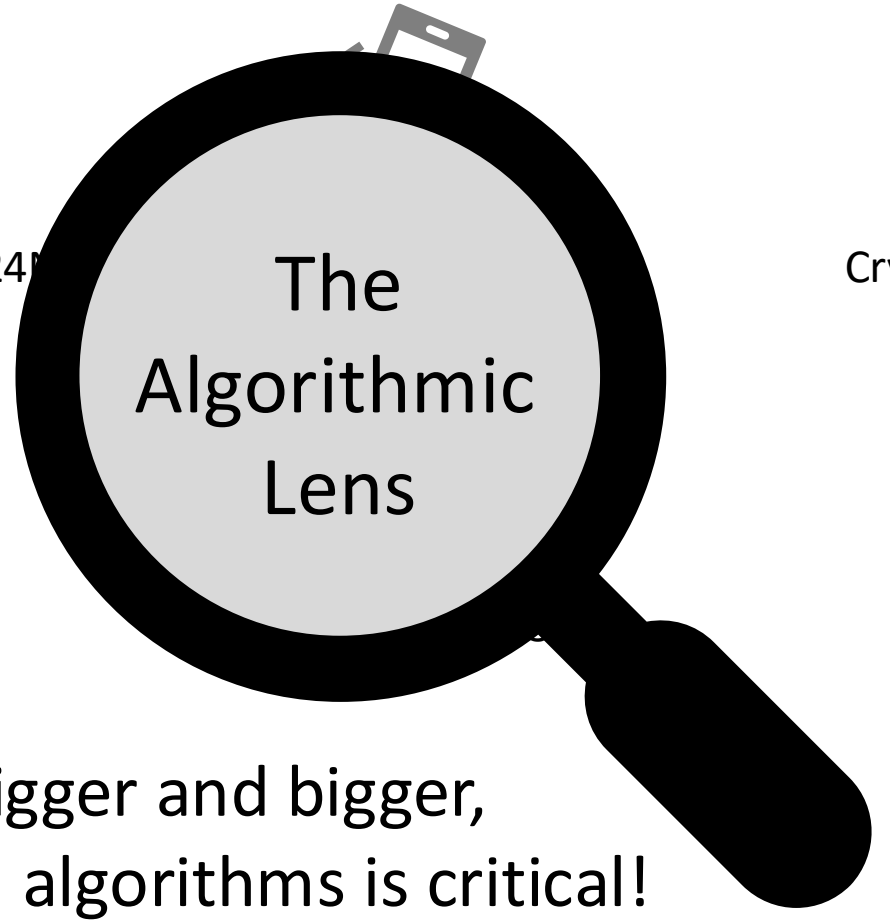
- Awesome CAs!
  - Ziyi Ding (head CA)
  - Illia Shkirko
  - Karan Bhasin
  - Michael Rybalkin
  - Mingwei Yang
  - Nikil Selvam
  - Ruiquan Gao
  - Simon Kim
  - Will Fang
  - Yash Dave
  - Zoe Wefers
  - Auddithio Nag (ACE CA)

Why are we here?

- Moses and I are here because we love algorithms

- Why are you here?
  - Algorithms are fundamental.
  - Algorithms are useful.
  - Algorithms are fun!
  - CS161 is a required course.

Algorithms: Fundamental & useful



Language models (CS224N)

Cryptography (CS255)

Robotics (CS237A)

Computational biology (CS262)

As inputs get bigger and bigger,
having good algorithms is critical!

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

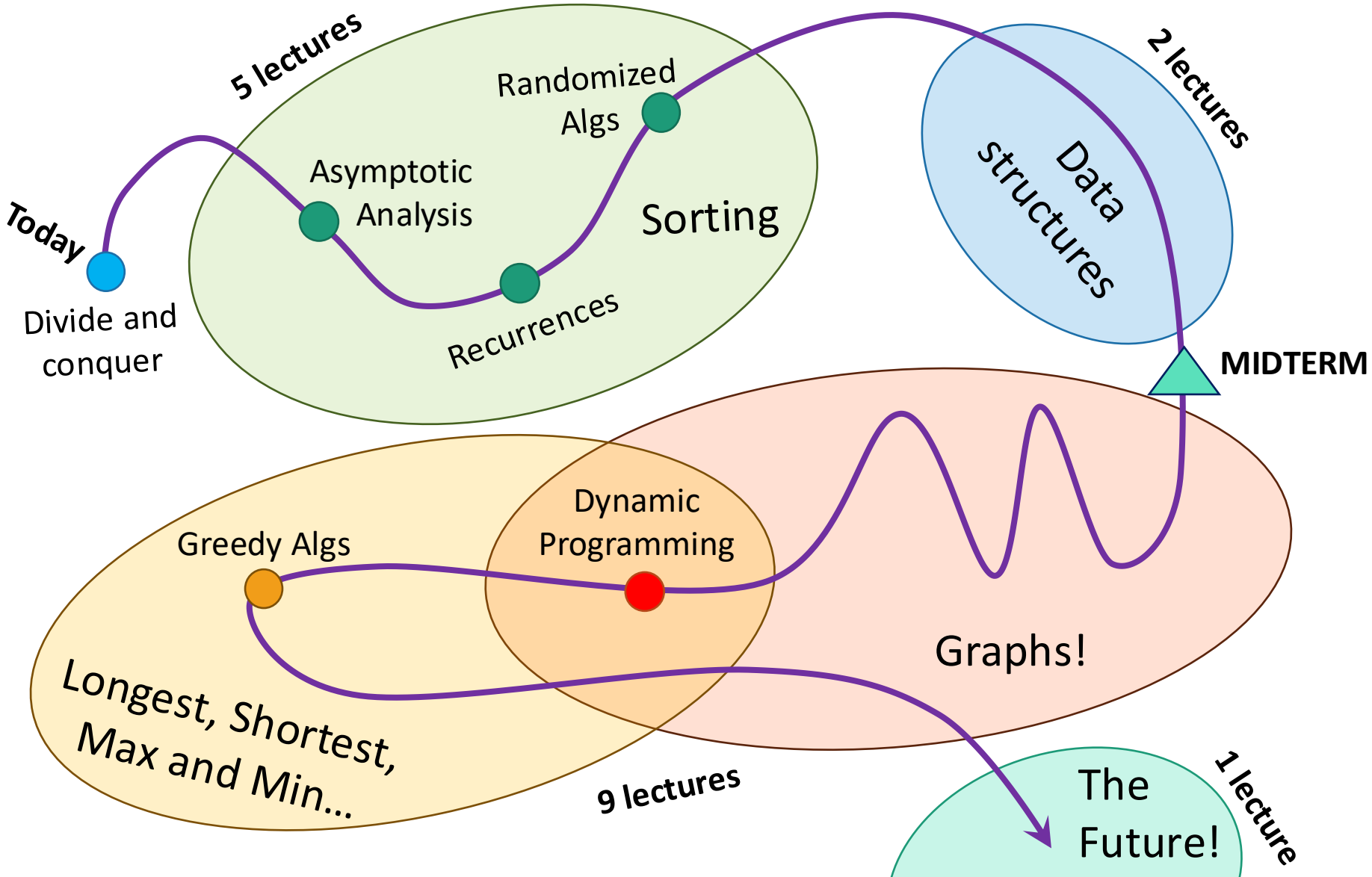
Course goals

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

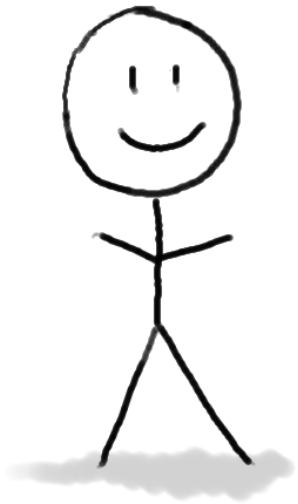
- In this course you will learn:
  - Design: Flesh out an "algorithmic toolkit"
  - Analysis: Learn to think analytically about algorithms
  - Communication: Learn to communicate clearly about algorithms

More detailed schedule on the website!

Roadmap



Our guiding questions:



Does it work?

Is it fast?

Can I do better?

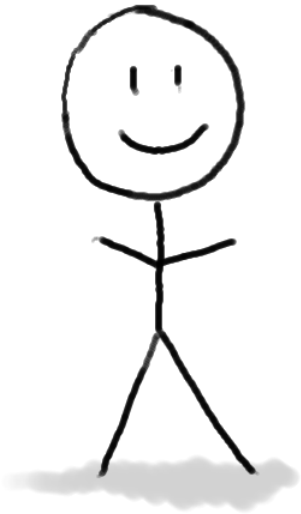
Our internal monologue…

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

Plucky the
Pedantic Penguin

Detail-oriented
Precise
Rigorous

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



Okay, this is basically the same as last time. If we just do the same thing again, it should probably work and run pretty fast.

Lucky the
Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

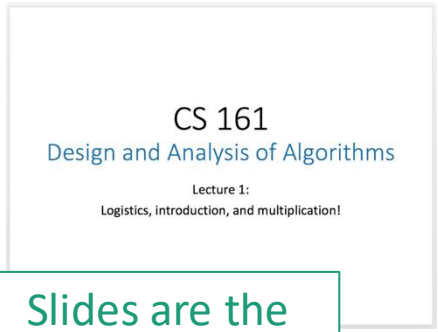
Both sides are necessary!

Course elements and resources

- **Course website:**
  - **stanford-cs161.github.io/winter2026**
- Lectures
- Homework
- Exams
- Office hours, Sections, and Ed

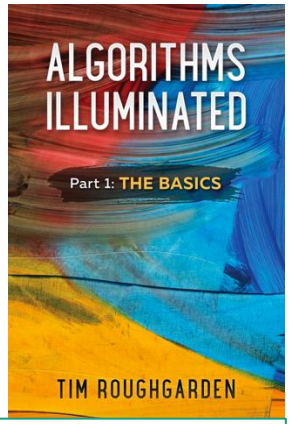
Lectures

- Here (STLC 111), M/W, 1:30-2:50
- Resources available:
  - Slides, Videos, Book, IPython notebooks

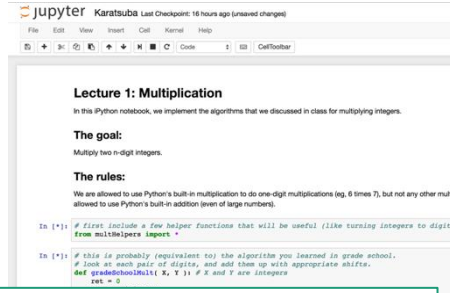


Slides are the slides from lecture.

Videos from lecture are available!



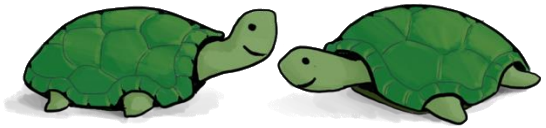
Textbook (and occasional hand-outs) have mathy details that slides may omit



IPython notebooks have implementation details that slides may omit.

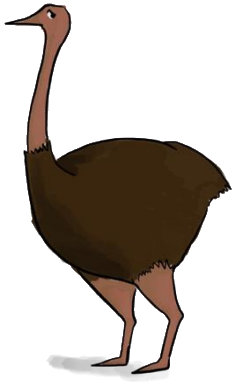
How to get the most out of lectures

- **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.**

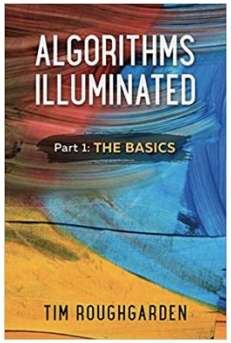
Homework!

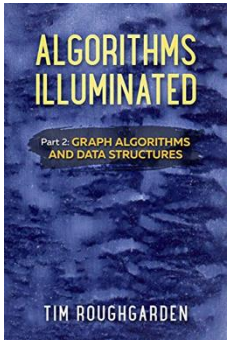
- Pre-req quiz: Self-graded w/ solutions on course page
- Weekly assignments HW1-HW8
- Due Wednesdays at 11:59pm, HW1 due Jan 14
- Homeworks 1-3 will be individual submissions
  - Means you should type up their own homework
- From Homework 4 onwards:
  - Paired submissions are permitted
  - You can make a single submission for groups of ≤2
- We only accept typed submissions. Handwritten submissions lose points on homework 1, and receive ZERO points from homework 2 onwards.

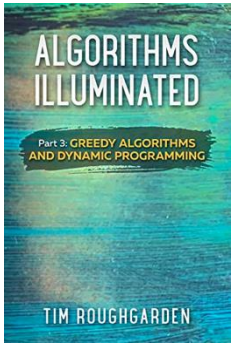
Late days

- You have six late days to use on HW1–HW8
  - See website for more details
- LATE DAYS ARE FOR EMERGENCIES.
  - Don’t ask us for an extension if you have an emergency. That’s what late days are for.

Textbook

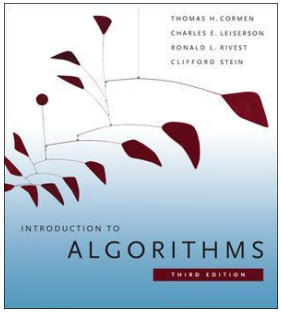




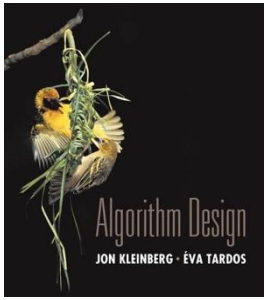


- **Algorithms Illuminated**, Vols 1, 2 and 3 by Tim Roughgarden
- Additional resources at algorithmsilluminated.org

- We may also refer to to the following (optional) books:



”CLRS”: Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein. Available FOR FREE ONLINE through the Stanford library.



“Algorithm Design” by Kleinberg and Tardos

Exams

- Two exams
  - Midterm: Wednesday 2/11, 6-9pm
  - Final Exam: Wednesday 3/18, 3:30-6:30pm

- We will not have a scheduled alternate final.
  - If you know you can't take the final, you should drop this class and take it in a different quarter.

- We are participating in the AIWG proctoring pilot
  - See website for details

Sections

- Taught by your amazing CAs and will
  - recap lecture
  - show you how to apply the ideas you learned in lecture
  - can occasionally cover new material
- We'll sharing locations & times in next few days
  - Go to whichever fits schedule
- Sections are as "mandatory" as lectures:
  - we will not track attendance, but
  - sections (practice, practice, practice) are the best way to learn the material in CS 161
  - also, a good place to find community

Review sessions

- Friday, 1:30-2:50, STLC 111
- This week: review of induction and week 1 material
- Future weeks: reviews before exams

ACE Section

- CS161ACE: 1-unit supplementary section for CS161
  - Algorithm design + problem-solving practice
- Meets Thursdays 3:00–4:50 PM
- **Attendance** mandatory
- Still attend CS161 lectures and regular sections
- Apply by Jan 9, 2026; waitlist after deadline
- Questions: ACE CA Auddithio Nag (aunag@stanford.edu)

Talk to us!

- Ed discussion forum:
  - Link on top of the course website
  - Course announcements will be posted there
  - Discuss material with TAs and your classmates

- Office hours:
  - See course website for schedule
  - They start next week

Course elements and resources

- **Course website:**
  - **stanford-cs161.github.io/winter2026**
- Lectures
- Homework
- Exams
- Office hours, Sections, and Ed

Course Policies

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

Bug bounty!

- We hope all PSETs and slides will be bug-free.
- Howover, we 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 or tell a CA).
  - The first person to catch a serious bug might get a good citizenship bonus point.
  - Good citizenship points: we might consider bumping letter grades for people who end up near boundaries and have lots of good citizenship points.

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 CGOE Students

- There will be some online office hours.
- One of the sections will be recorded.
- See the website for more details! (coming soon)

OAE forms

- Please send OAE forms to
  cs161-staff-win2526@cs.stanford.edu

- If you plan to use your OAE-approved exam accommodations for a specific exam:
  Must send letter 10 days before exam date

Feedback

- We will have high-resolution feedback throughout the course (subset of you randomly asked each week, starting week 2).
- Please help us improve the course!

Everyone can succeed in this class!

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

The big questions

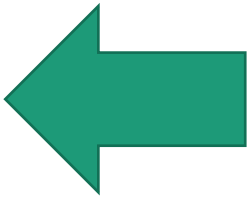
- Who are we?
  - Professor, TA's, students?
- Why are we here?
  - Why learn about algorithms?
- What is going on?
  - What is this course about?
  - Logistics?
- Can we multiply integers?
  - And can we do it quickly?

Course goals

- Design: Flesh out an "algorithmic toolkit"
- Analysis: Think analytically about algorithms
- Communication: Learn to communicate clearly about algorithms

Today's goals

- Karatsuba Integer Multiplication

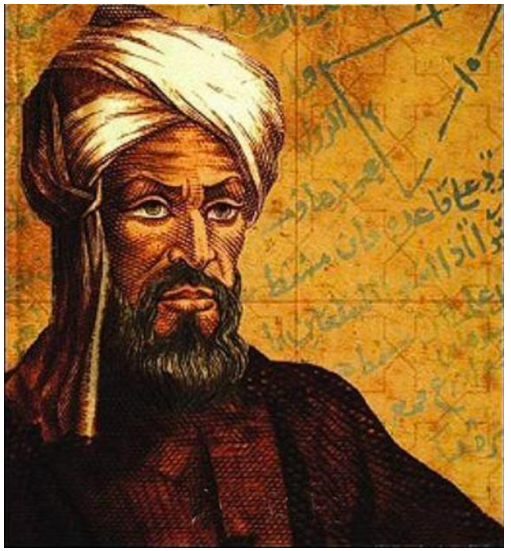


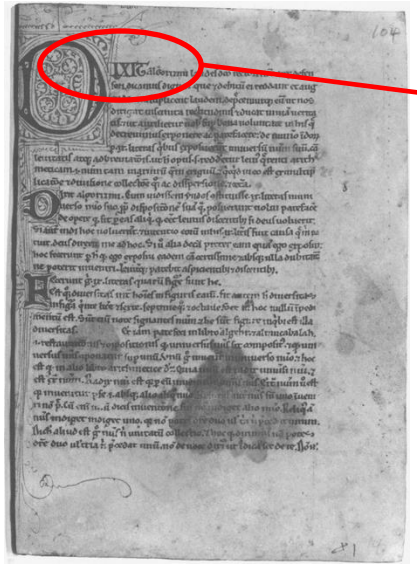
- Algorithmic Technique:
  - Divide and conquer
- Algorithmic Analysis tool:
  - Intro to asymptotic analysis

Let’s start at the beginning

Etymology of “Algorithm”

- Al-Khwarizmi was a 9th-century scholar from Uzbekistan
- 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

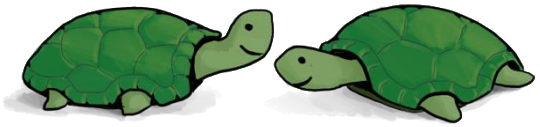
n

```
  12339257207527523846237642835683649183745238562 98
x 45623235823423952856234672350191307501353500137 53
```

???

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 $O(n^2)$"

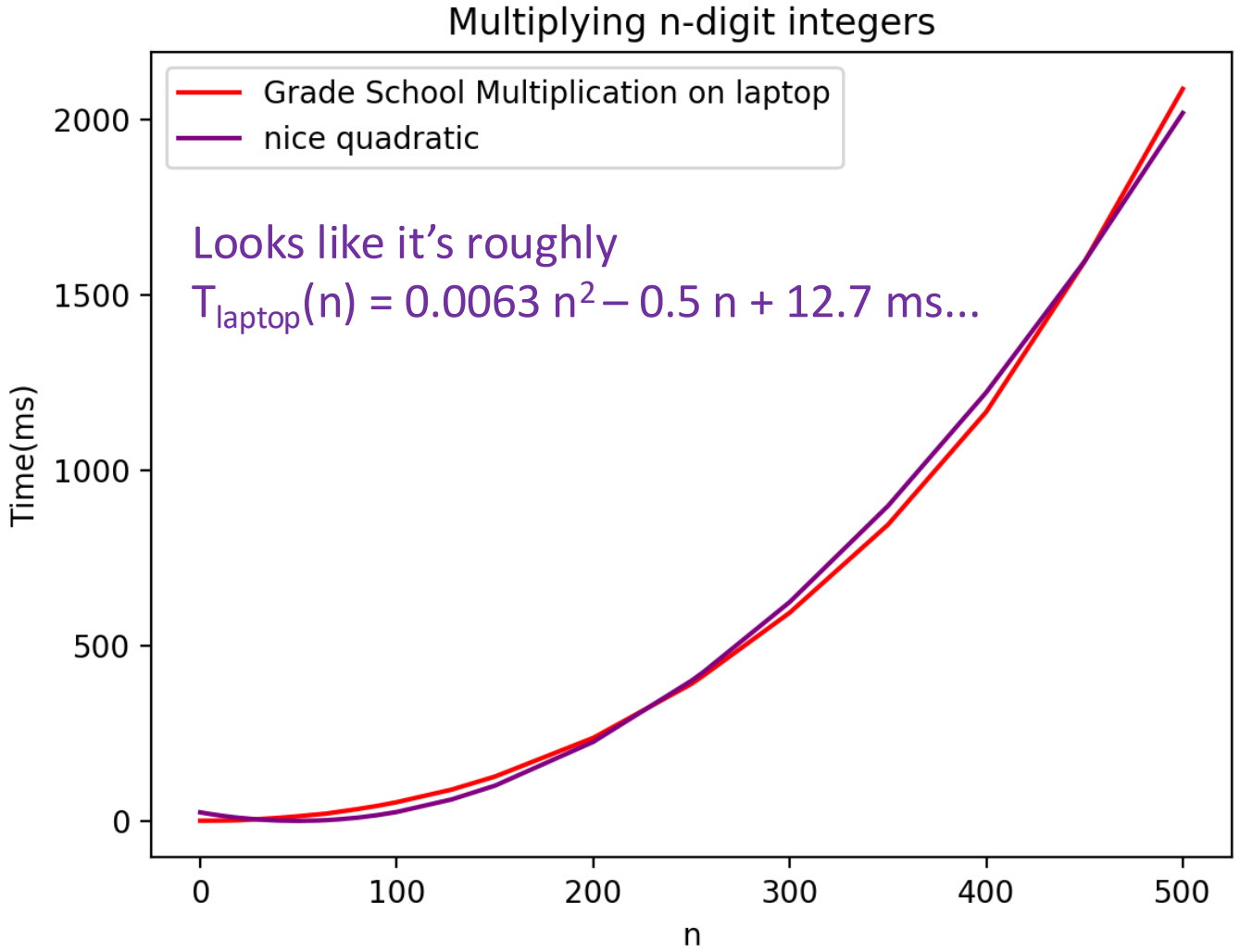
- Formal definition coming Wednesday!
- Informally, big-Oh notation tells us how the running time scales with the size of the input.

highly non-optimized

Implemented in Python, on my laptop

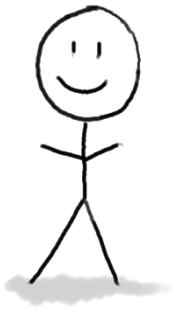
The runtime "scales like" n^2

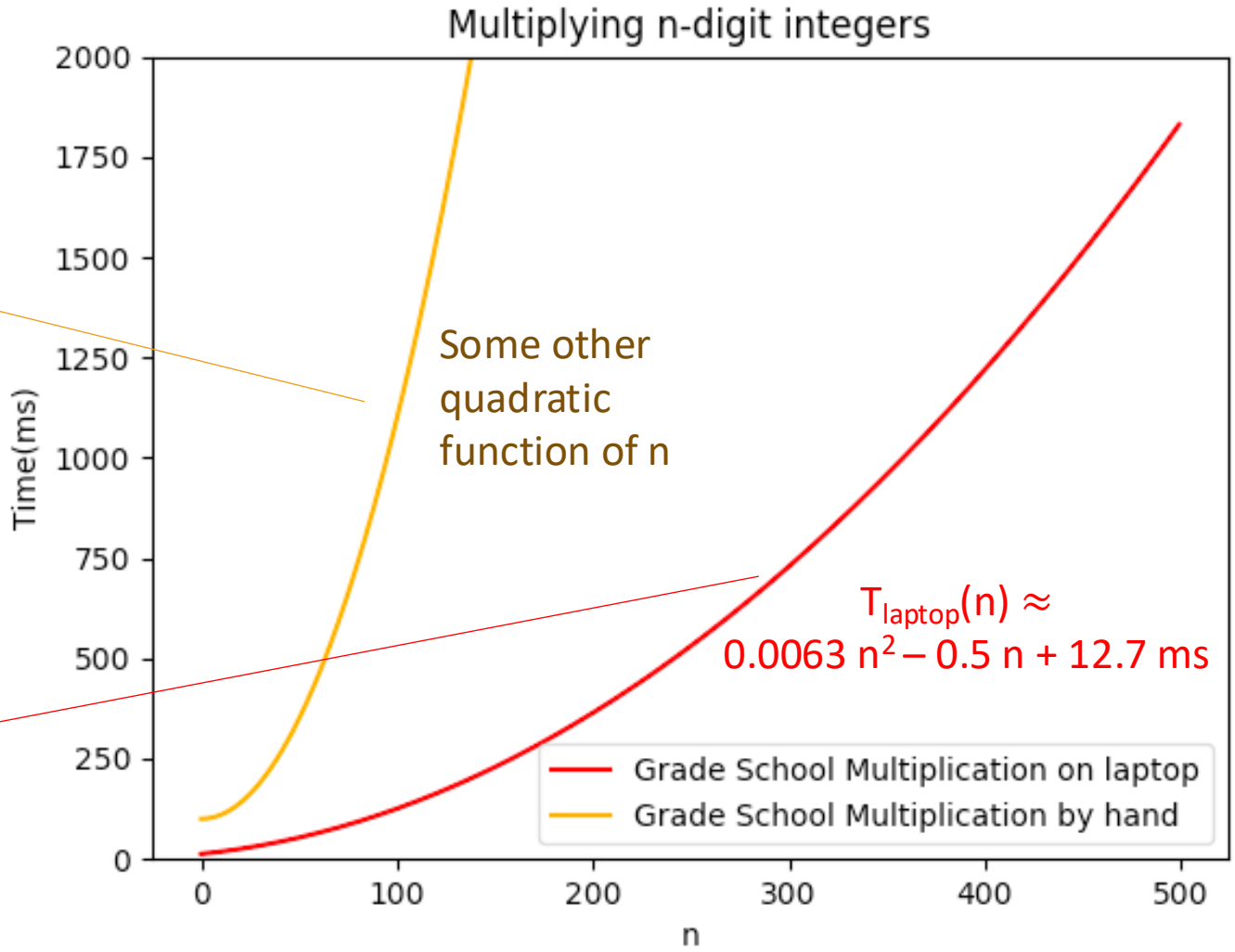
python™



Implemented by hand

The runtime still "scales like" n^2

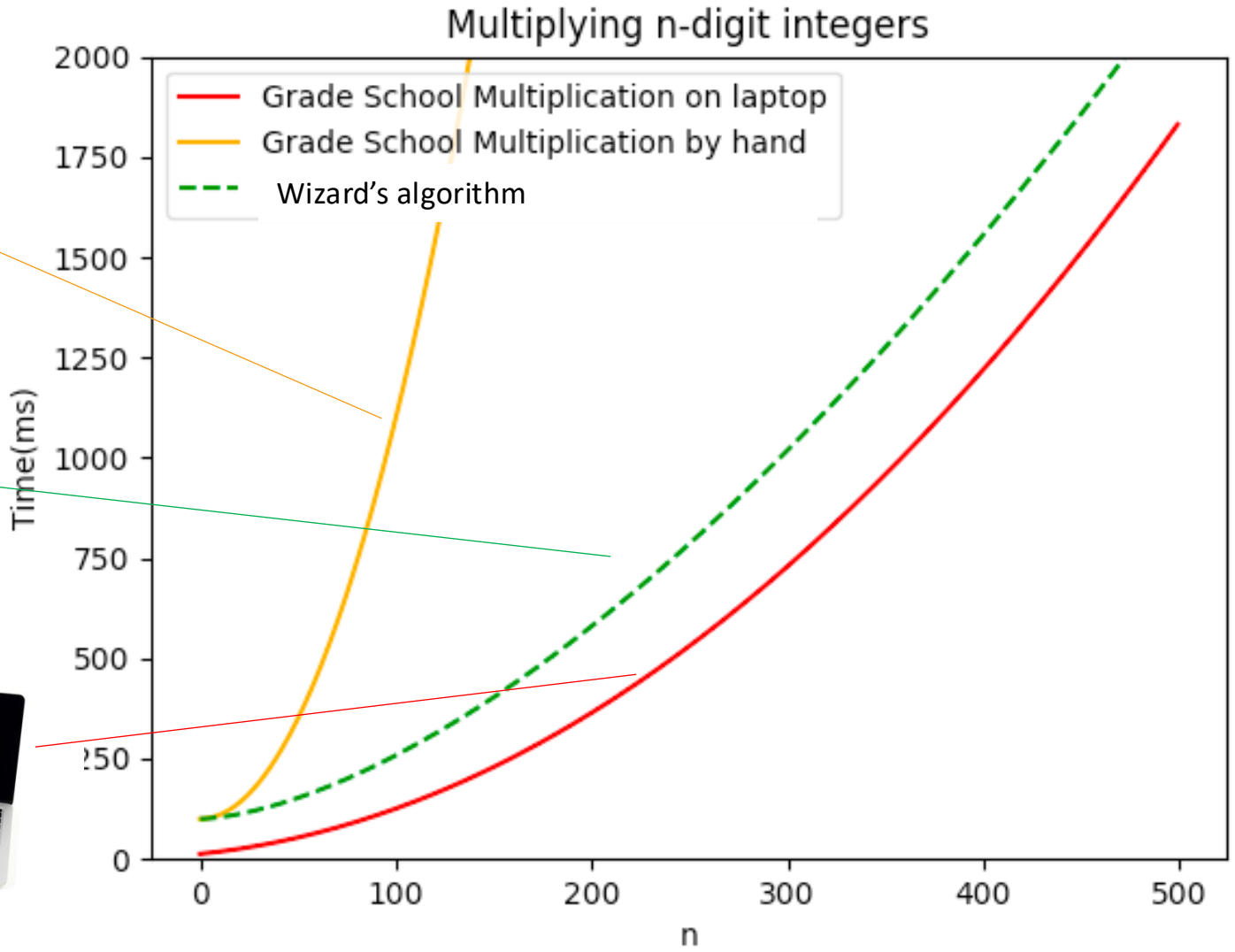




Why is big-Oh notation meaningful?

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

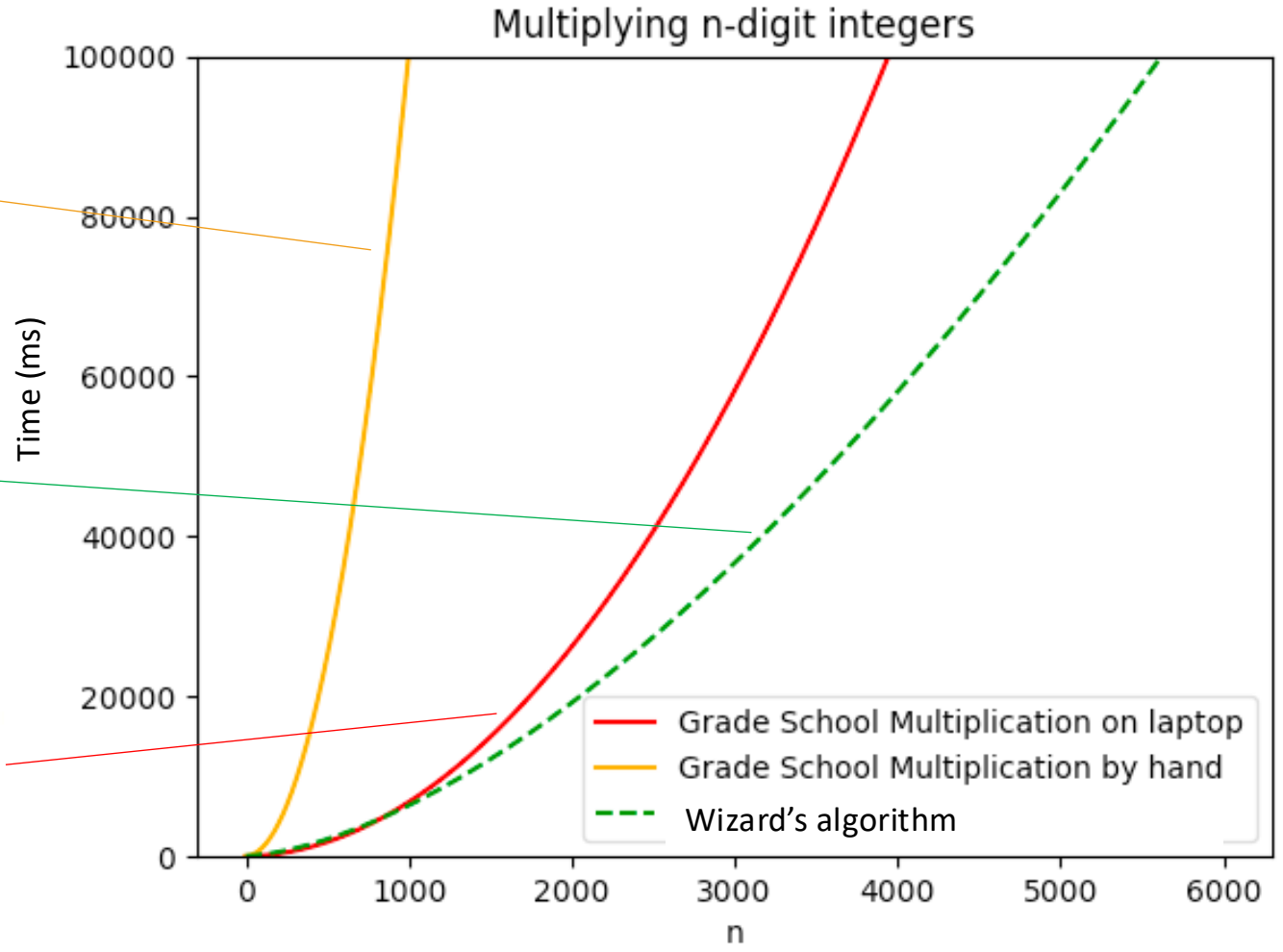
$\approx .0063n^2$



Let n get bigger…

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

$\approx .0063n^2$



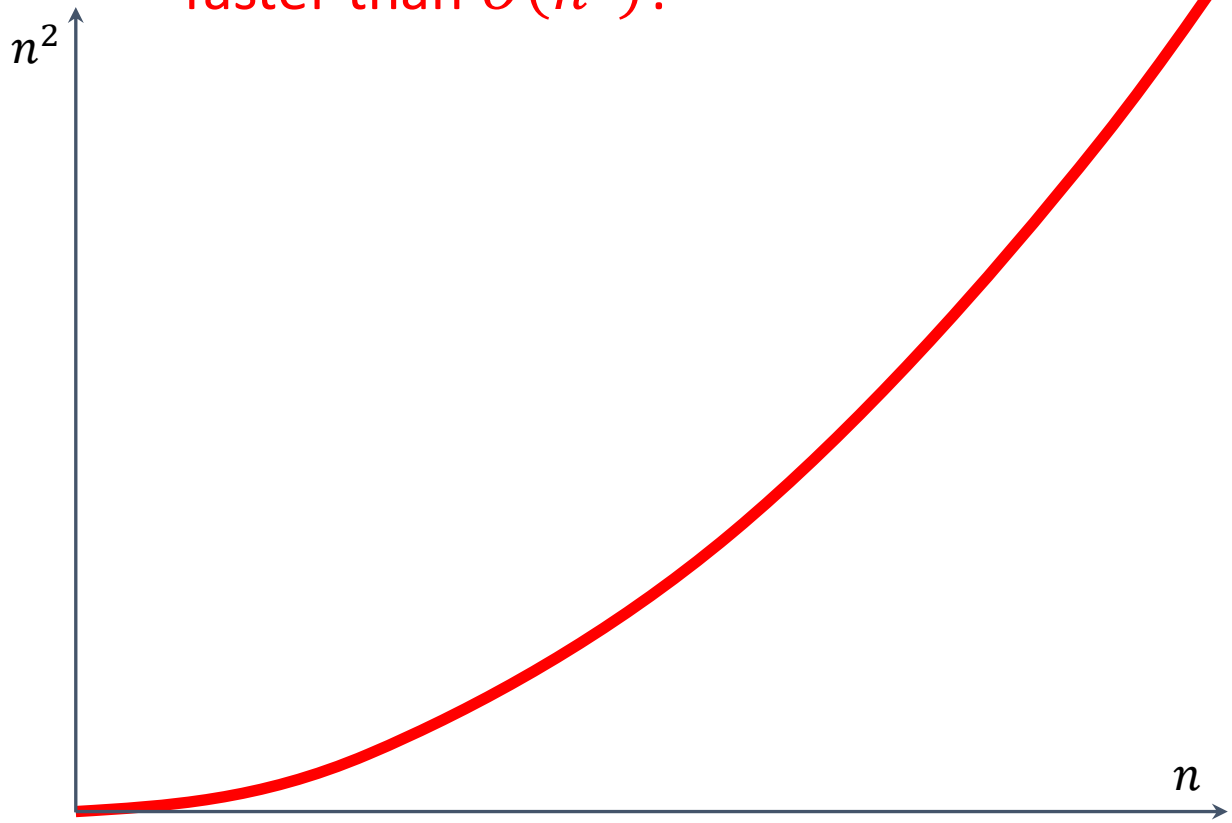
Take-away

- 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?

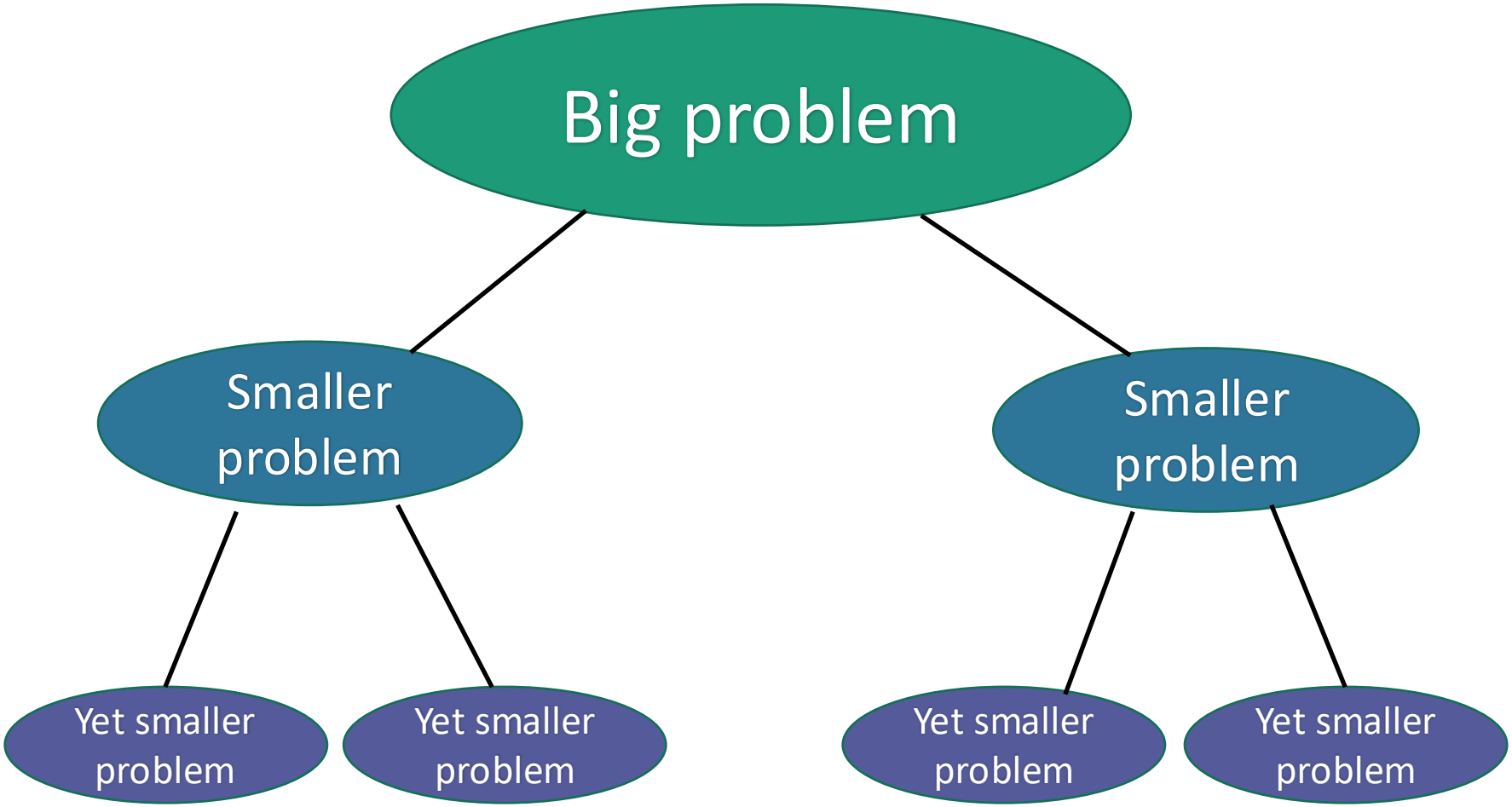
Can we multiply n-digit integers faster than $O(n^2)$?



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:

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

$$1234 \times 5678$$

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

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

One 4-digit multiply → Four 2-digit multiplies

More generally

Suppose n is even

Break up an n-digit integer:

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

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

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

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):

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

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

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

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!

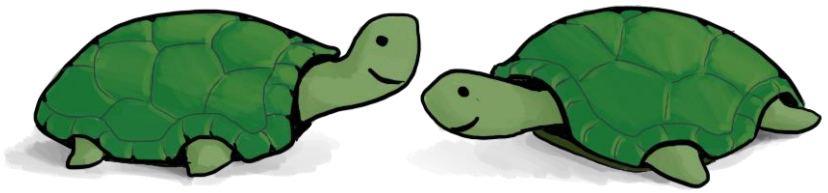
Siggi the Studious Stork

Think-Pair-Share

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

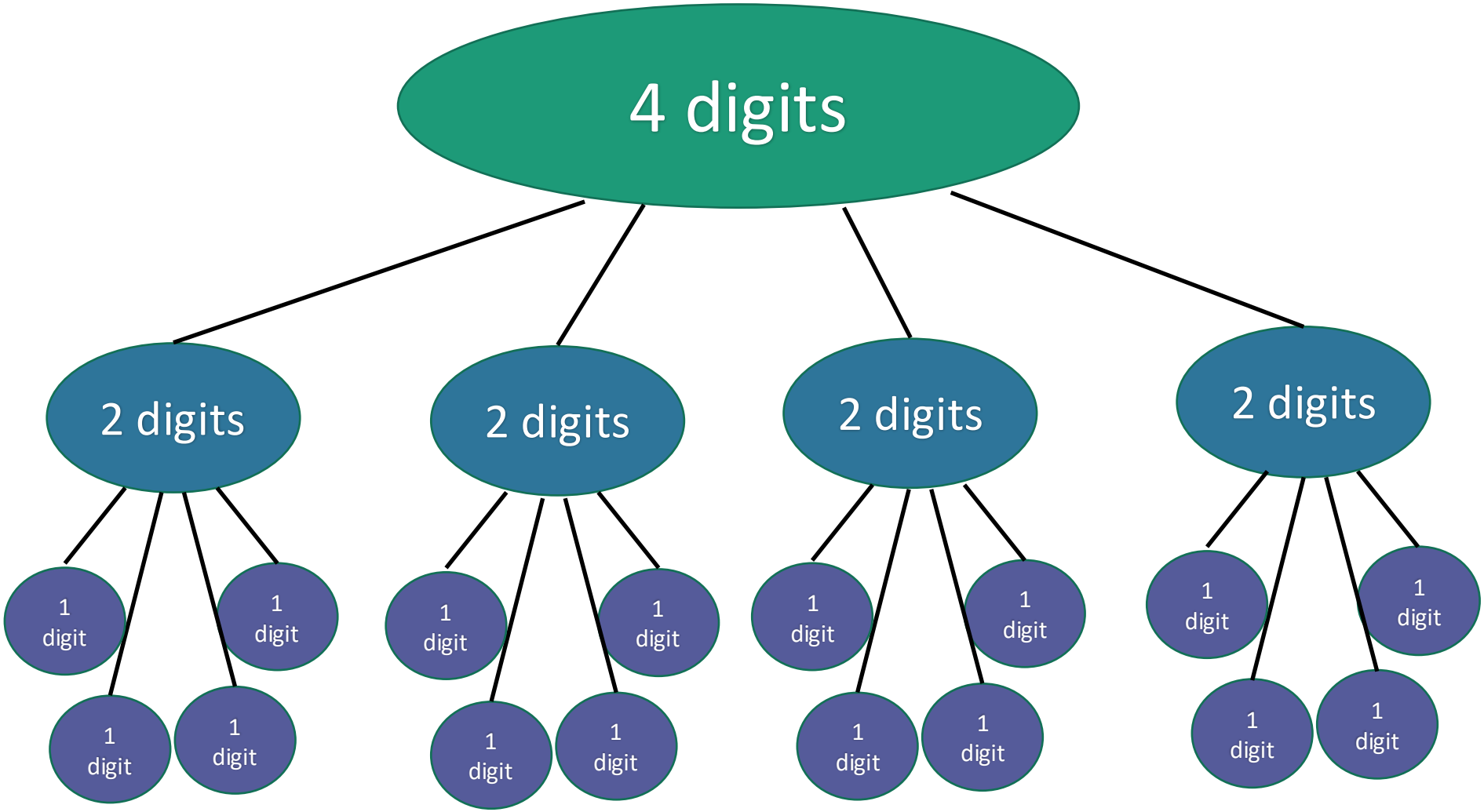
$$1234 \times 5678$$

- 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?

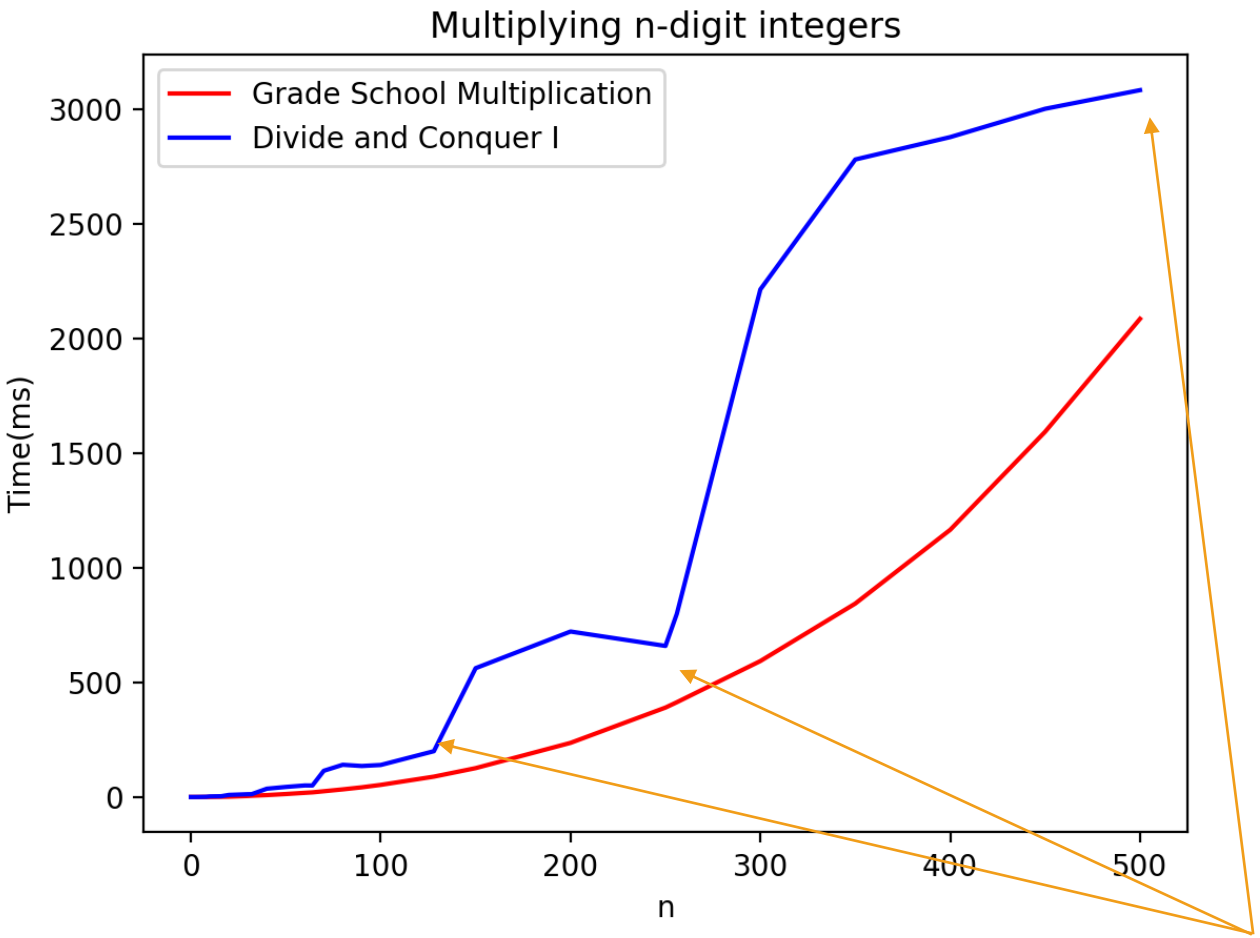
- Better or worse than the grade school algorithm?

- 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.

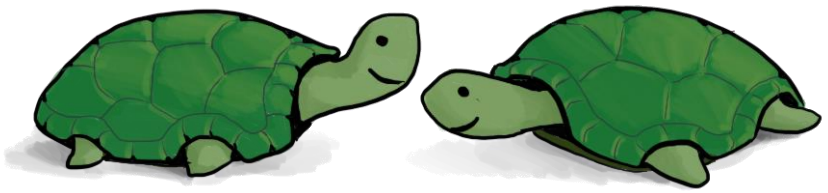
Not sound logic!

Plucky the Pedantic Penguin

2. Try to understand the running time analytically

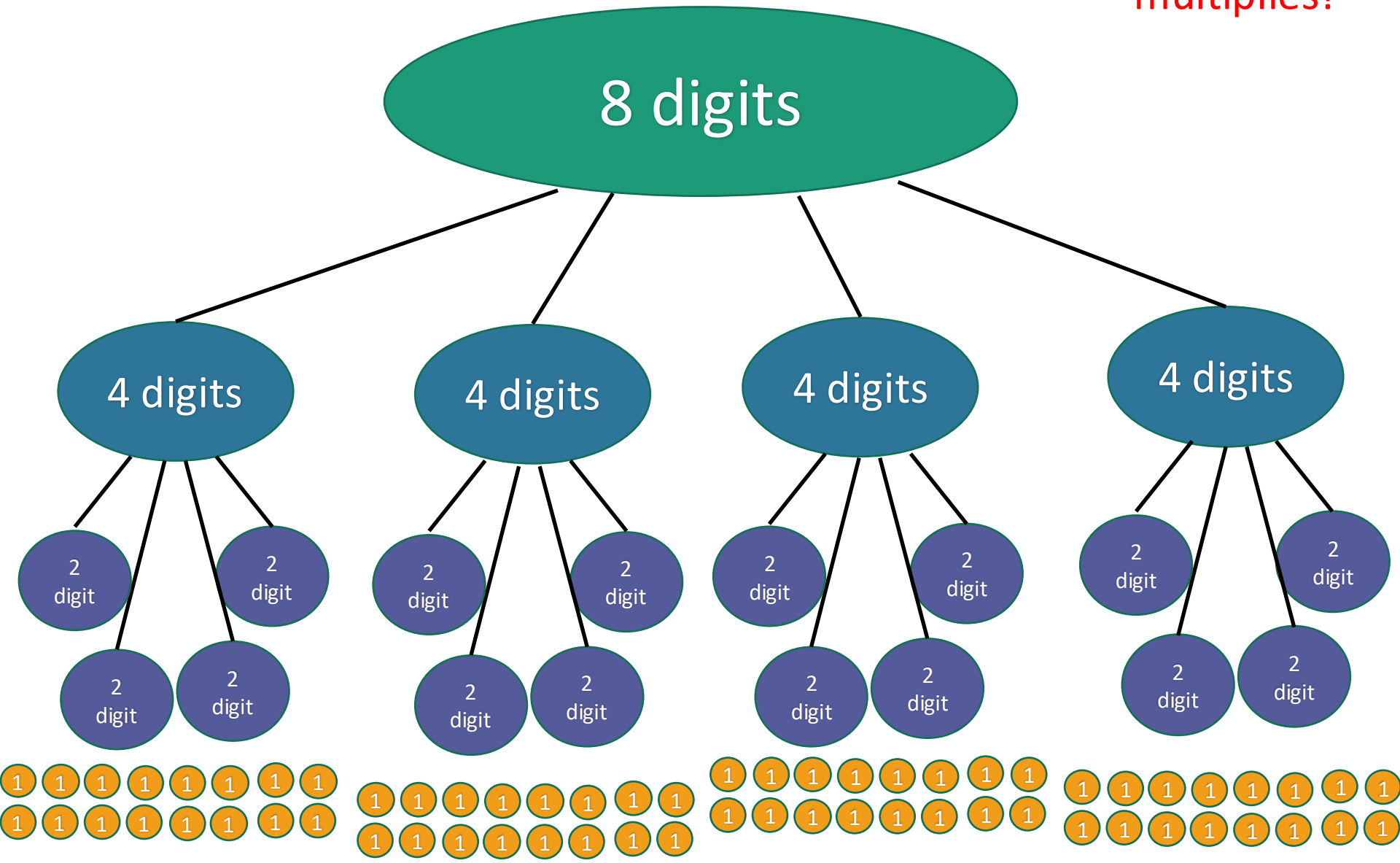
Think-Pair-Share:

- 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:

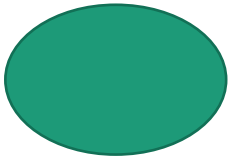
We end up doing about n^2 one-digit multiplications

$\Rightarrow$

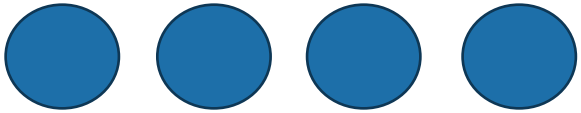
The running time of this algorithm is

AT LEAST n^2 operations.

There are n^2 1-digit problems

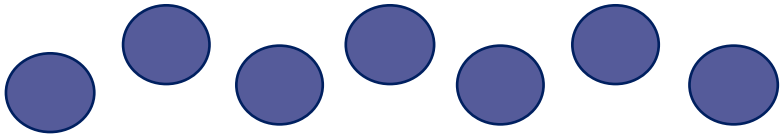


1 problem
of size n



4 problems
of size n/2

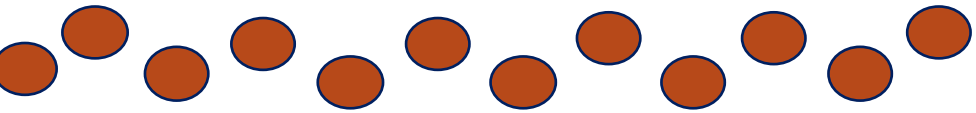
...



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

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

...



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

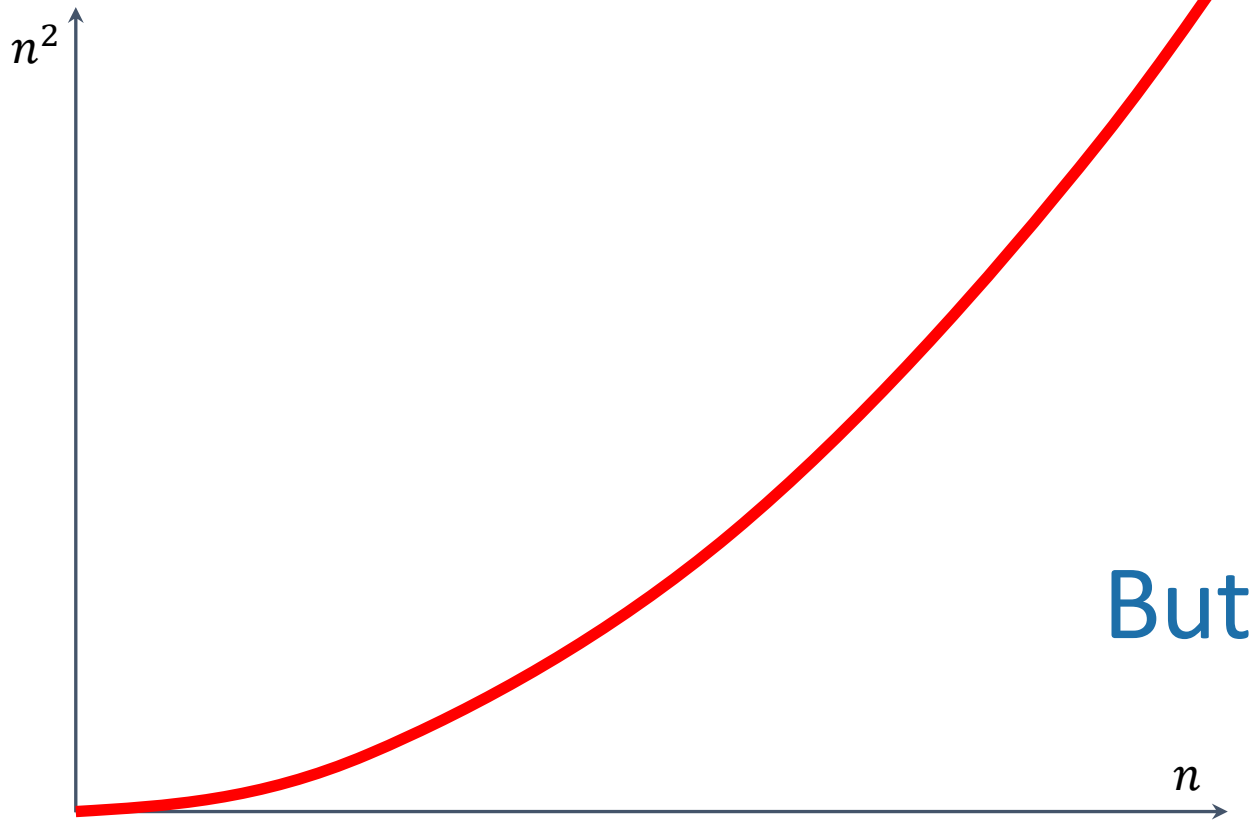
- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So at level $t = \log_2(n)$ we get…

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

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!

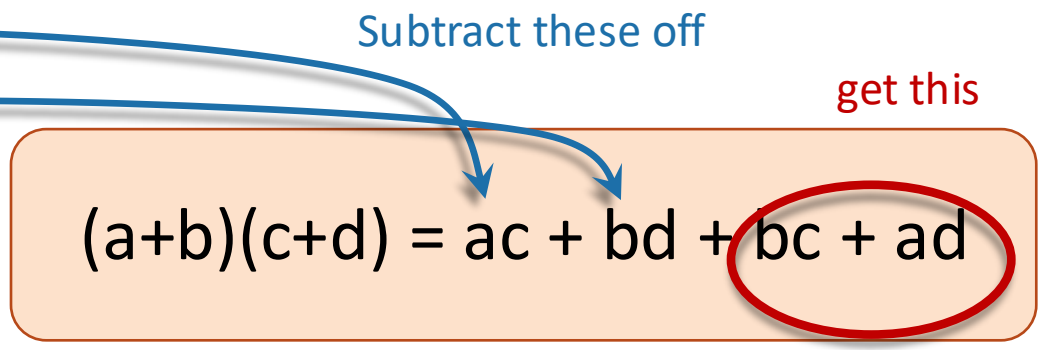
$$xy = (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d)$$
$$= ac \cdot 10^{n} + (ad + bc)10^{n/2} + bd$$

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)



- 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?

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.)

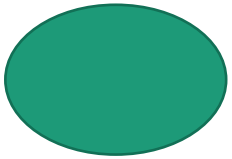
Multiply(x, y):

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

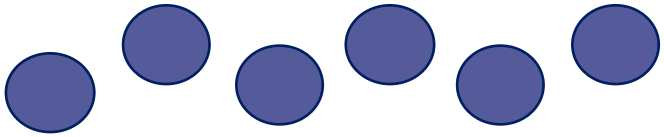
What's the running time?



1 problem
of size n

3 problems
of size n/2

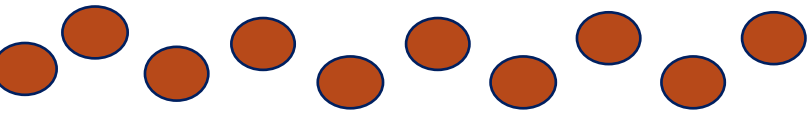
…



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

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

…



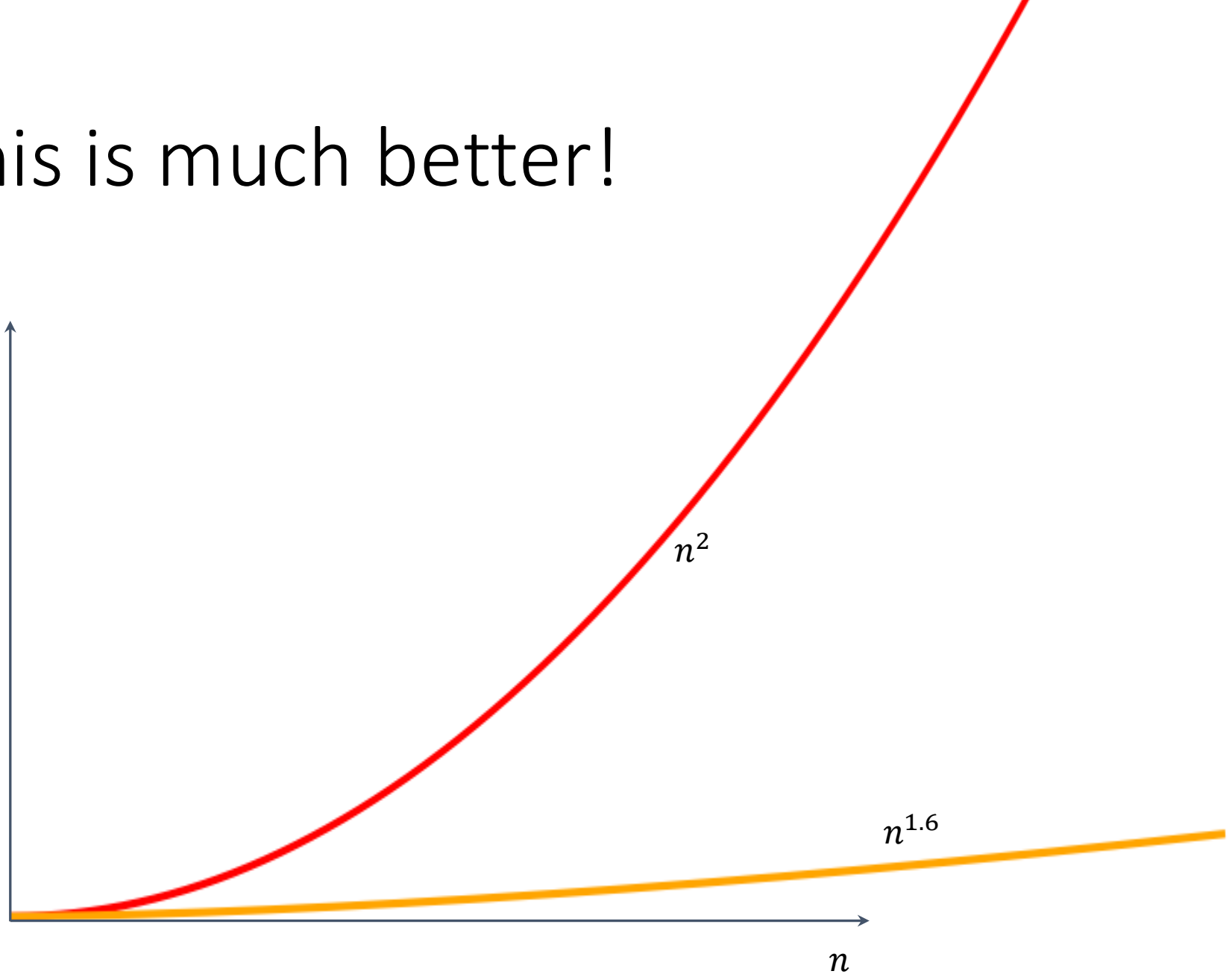
$\frac{n^{1.6}}{\_\_\_}$ problems
of size 1

- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So at level $t = \log_2(n)$ we get…

$3^{\log_2 n} = n^{\log_2 3} \approx n^{1.6}$
problems of size 1.

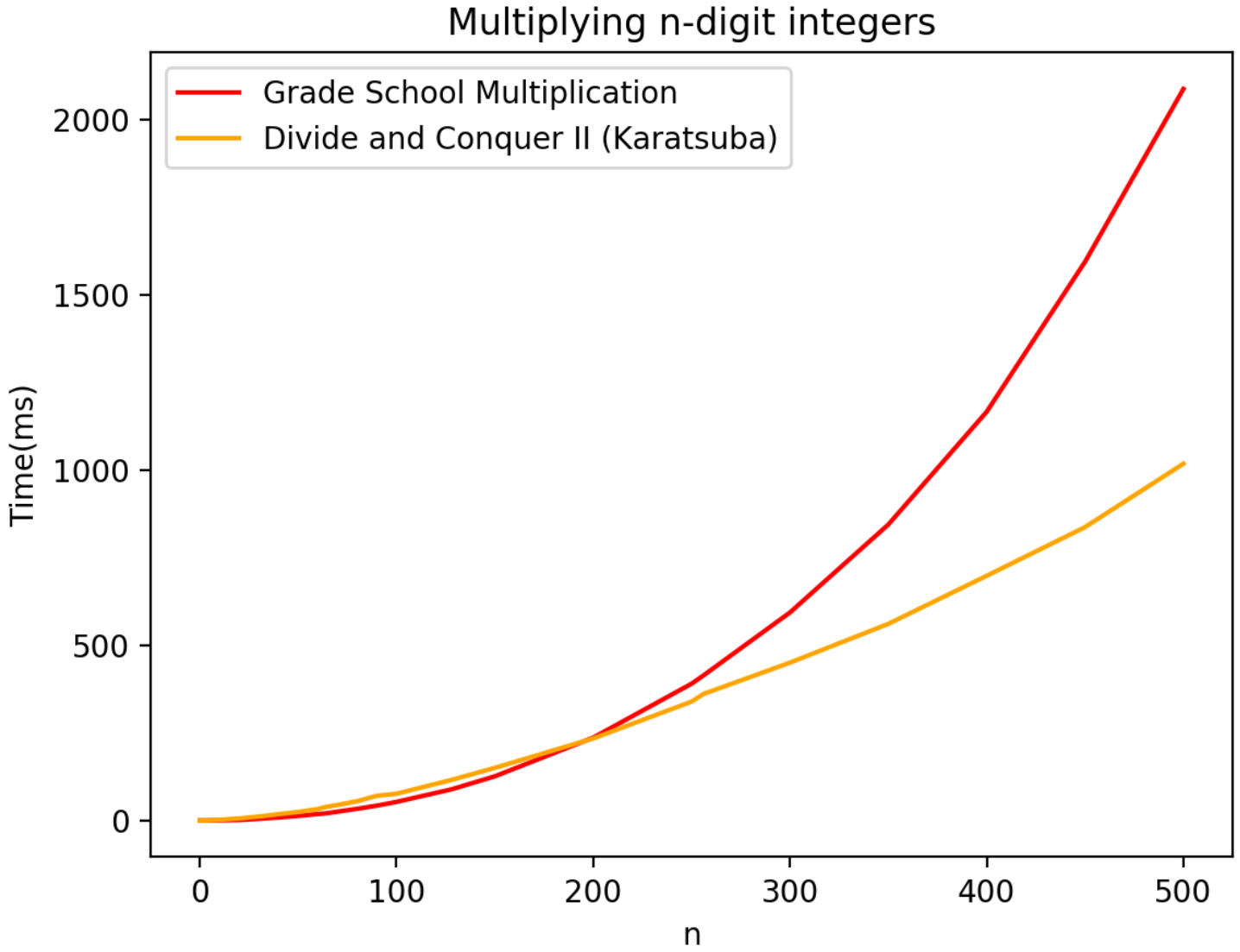
We aren't accounting for the work at the higher levels! But we'll see later that this turns out to be okay.

This is much better!



We can even see it in real life!

python™



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!]

Course goals

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

Today's goals

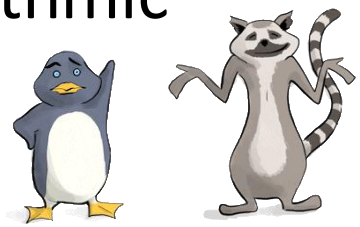
- Karatsuba Integer Multiplication
- Algorithmic Technique:
  - Divide and conquer
- Algorithmic Analysis tool:
  - Intro to asymptotic analysis

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

- **stanford-cs161.github.io/winter2026**
- 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!