CS 161

Design and Analysis of Algorithms

Lecture 1:

Logistics, introduction, and multiplication!

Welcome to CS161!

Who are we?

- Instructor:
  - Mary Wootters
- Awesome TAs:
  - Michael Chen
  - Steven Chen
  - Shawn Hu
  - Sam Kim
  - Dana Murphy
  - Jessica Su (Head TA)

Who are you?

- CS majors…
- Physics
- Applied Physics
- BioE
- Biomedical informatics
- Civil + Env. Engineering
- CME
- EE
- Materials Science
- Econ
- Linguistics
- Mech E
- MS&E
- Math
- Stats
- Music
- Biology
- English
- Comp. Lit.
- International Policy Studies
- History
- Philosophy
- Symbolic Systems

Today

- Why are you here?
- Course overview, logistics, and how to succeed in this course.
- Some actual computer science.

Why are you here?

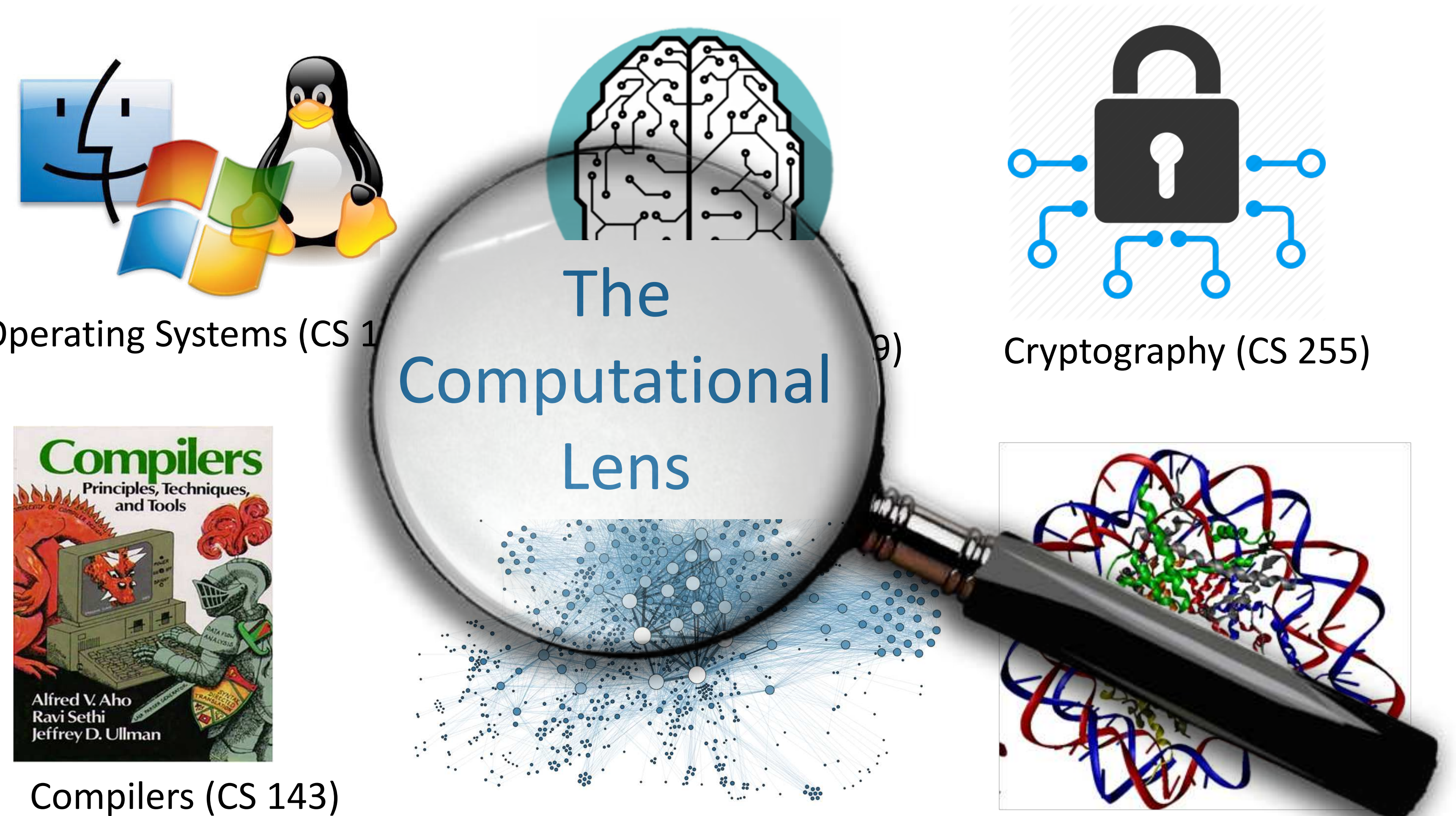
You are better equipped to answer this question than I am, but I'll give it a go anyway…

- 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 1

9)

Cryptography (CS 255)

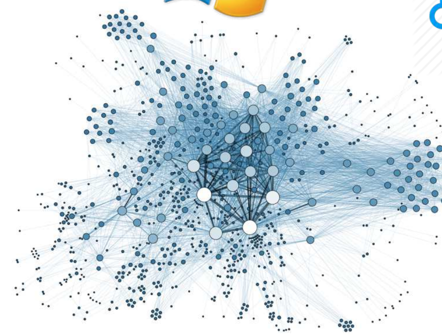
Compilers (CS 143)

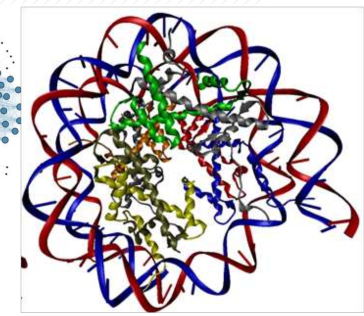
Networking (CS 144)

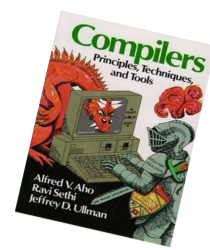
Computational Biology (CS 262)

Algorithms are useful

- All those things, without Stanford CS class numbers
- As we get more and more data and problem sizes get bigger and bigger, algorithms become more and more important.
- Will help you get a job.







Algorithms are fun!

- Algorithm design is both an art and a science.
- Many surprises!
- A young field, lots of exciting research questions!

Today

- Why are you here?
- Course overview, logistics, and how to succeed in this course.
- Some actual computer science.

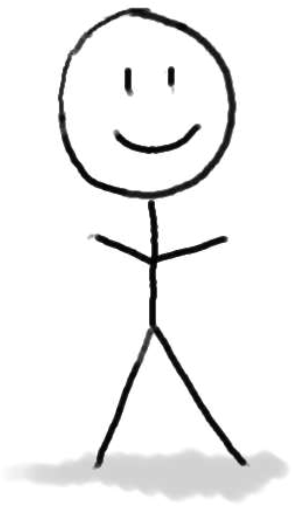
Course goals

- 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

The algorithm designer's question

Can I do better?



Algorithm designer

The algorithm designer's internal monologue…

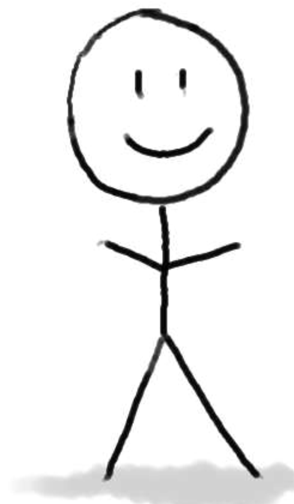
What exactly do we mean by better? And what about that corner case? Shouldn't we be zero-indexing?

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



Algorithm designer

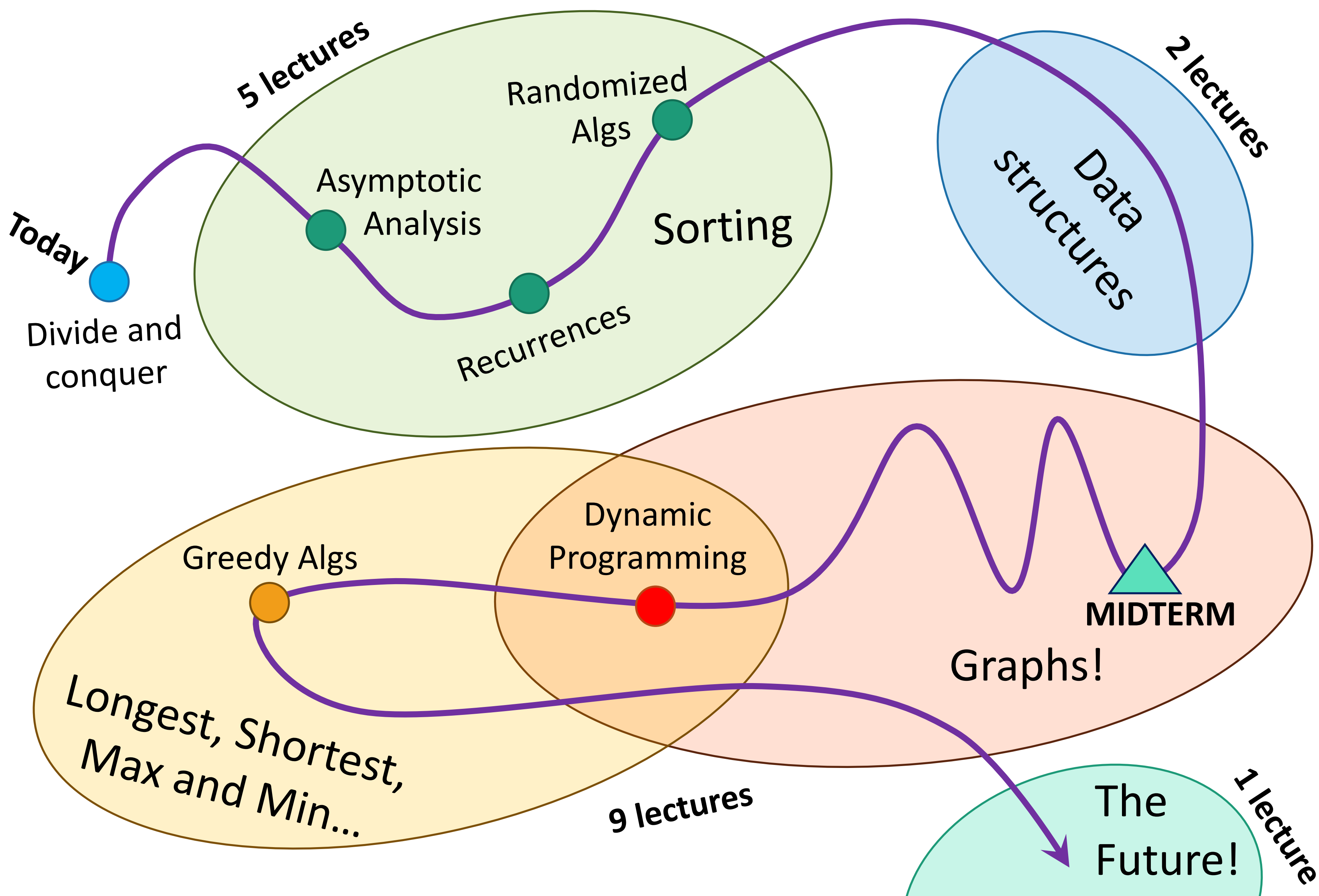
Lucky the Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

Both sides are necessary!

More detailed schedule on the website!

Roadmap



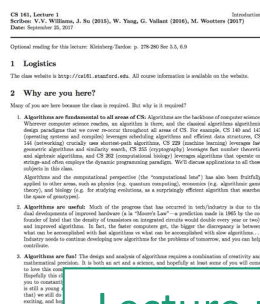
Course elements and resources

- Course website:
  - cs161.stanford.edu
- Lectures
- Textbook
- Homework
- Exams
- Office hours, recitation sections, and Piazza
- ***New this quarter!*** A bit of Python throughout!

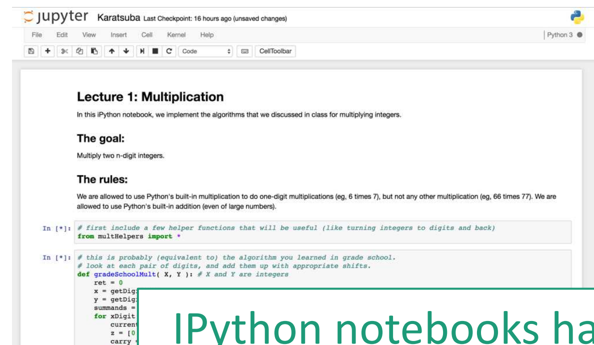
Lectures

- Right here, M/W, 1:30-2:50!
- Resources available:
  - Slides, Lecture Notes, IPython notebooks

Slides are the slides from lecture.



Lecture notes have mathy details that slides may omit



IPython notebooks have implementation details that slides may omit.

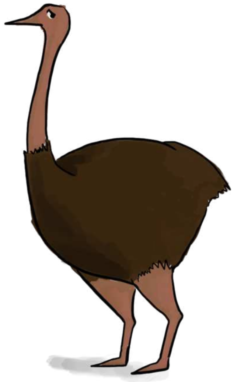
- Goal of lectures:
  - Hit the most important points of the week's material
    - Sometimes high-level overview
    - Sometimes detailed examples

How to get the most out of lectures

- **During lecture:**
  - Show up, ask questions, put your phone away.
  - May be helpful: take notes on printouts of the slides.
- **Before lecture:**
  - Do the ***pre-lecture exercises*** listed on the website.
- **After lecture:**
  - Go through the exercises on the slides.

These guys will pop up on the slides and ask questions – those questions are for you!

Siggi the Studious Stork (recommended exercises)



Ollie the Over-achieving Ostrich (challenge questions)

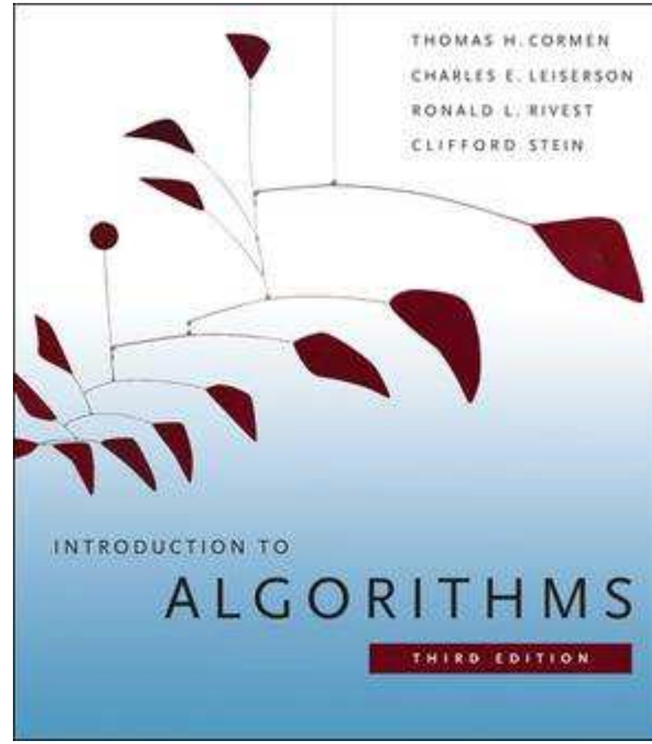
- ***Do the reading***
  - either before or after lecture, whatever works best for you.
  - **do not wait to "catch up" the week before the exam.**

Textbook

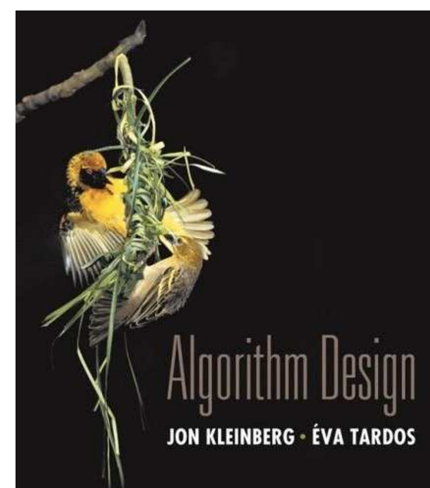
- **CLRS**:
  - Introduction to Algorithms, by Cormen, Leiserson, Rivest, and Stein.
- Available ***FOR FREE ONLINE*** via

STANFORD
UNIVERSITY
LIBRARIES

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



We will also sometimes refer to Kleinberg and Tardos



Homework!

Weekly assignments in two parts:

1. Exercises:
   - Check-your-understanding and computations
   - Should be pretty straightforward
   - ***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.
    - You **must** write up your solutions on your own.

- If you get help from a TA at office hours:
    - **Try the problem first.** And then try a few more times.
    - 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.

Homework logistics

- Assignment will be posted FRIDAY afternoon.
  - On the course website.
- Due the next FRIDAY afternoon. (3pm)
  - On GradeScope.
- Solutions are posted the following MONDAY.
  - One the course website.

Monday	Tuesday	Wednesday	Thursday	Friday
				HW ASSIGNED
				HW DUE
SOLUTIONS RELEASED				HW RETURNED (target)

Exams

- There will be a **midterm** and a **final**.
  - **MIDTERM:** 10/30, in class
  - **FINAL:** 12/13, 3:30-6:30pm

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

- **If you have a conflict with these exams, contact me (marykw) and Jessica Su (jtysu) ASAP!!!!!**

Talk to us

- Sign up for Piazza:
    - Course announcements will be posted there
    - Discuss material with TAs and your classmates
- Office hours:
    - See website for schedule
    - Suggestion: do not go to office hours for nonspecific "homework help." Go with a specific question.
- Recitation sections:
    - Optional, for your benefit only
    - Extra practice with the material, example problems, etc.

New this quarter! A bit of Python.

- Lectures and homework will occasionally use IPython notebooks.
  - Mostly, 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).

Course elements and resources

- Course website:
  - cs161.stanford.edu
- Lectures
- Textbook
- Homework
- Exams
- Office hours, recitation sections, and Piazza
- ***New this quarter!*** A bit of Python throughout!

Bug bounty! 

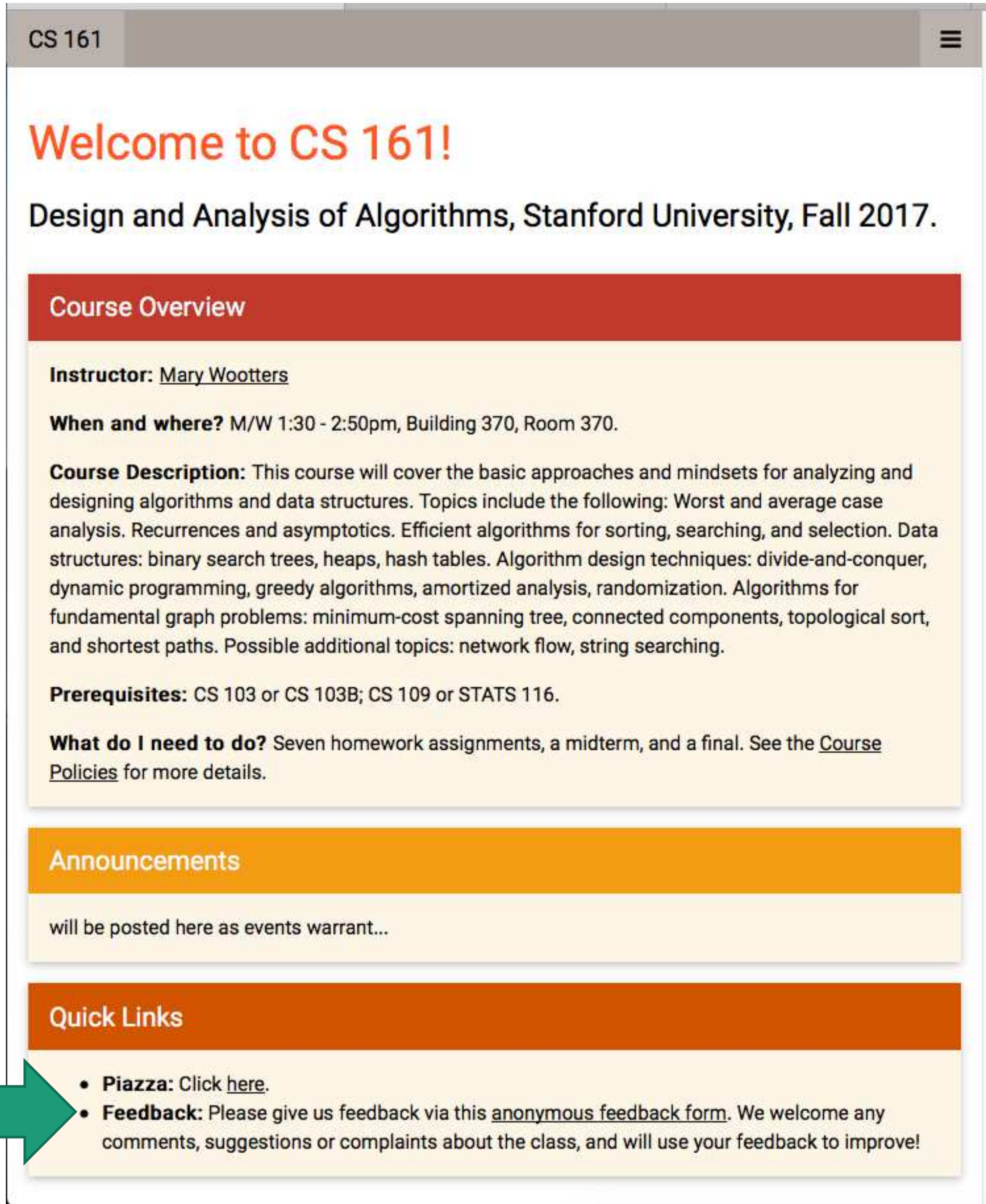
- We hope all PSETs and slides will be bug-free.
- However, I sometmes maek typos.
- **If you find a typo** (that affects understanding*) on slides, IPython notebooks, lecture notes, or PSETs:
  - Let us know! (Email jtysu and marykw, or post on Piazza).
  - The first person to catch a bug gets a bonus 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.

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. Ask for help
3. Work hard

Today

- Why are you here?
- Course overview, logistics, and how to succeed in this course.
- Some actual computer science.

Course goals

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

Today's goals

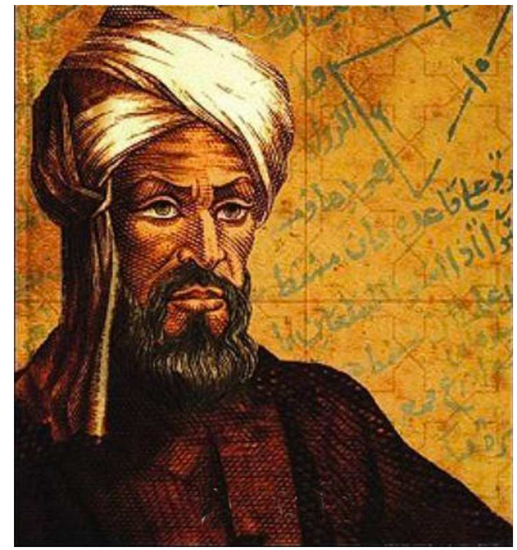
- Karatsuba Integer Multiplication
- Technique: Divide and conquer
- Meta points:
    - How do we measure the speed of an algorithm?

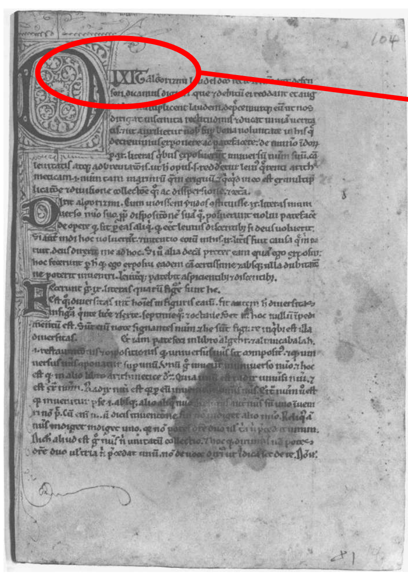
Let’s start at the beginning

Etymology of “Algorithm”

- Al-Khwarizmi (Persian mathematician, lived around 800 AD) wrote a book about how to multiply with Arabic numerals.

- His ideas came to Europe in the 12th century.





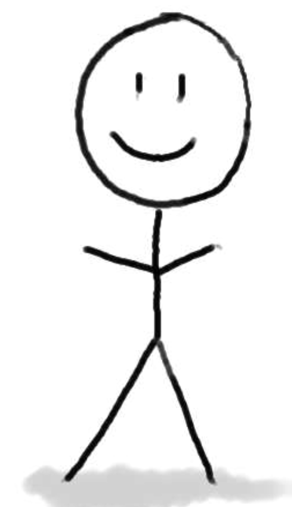
Dixit algorizmi
(so says Al-Khwarizmi)

- 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

$$
\begin{array}{rr}
 & 1234567895931413 \\
\times & 4563823520395533 \\
\hline
\end{array}
$$

Integer Multiplication

n

  1233925720752752384623764283568364918374523856298
x 4562323582342395285623467235019130750135350013753

???

How long would this take you?

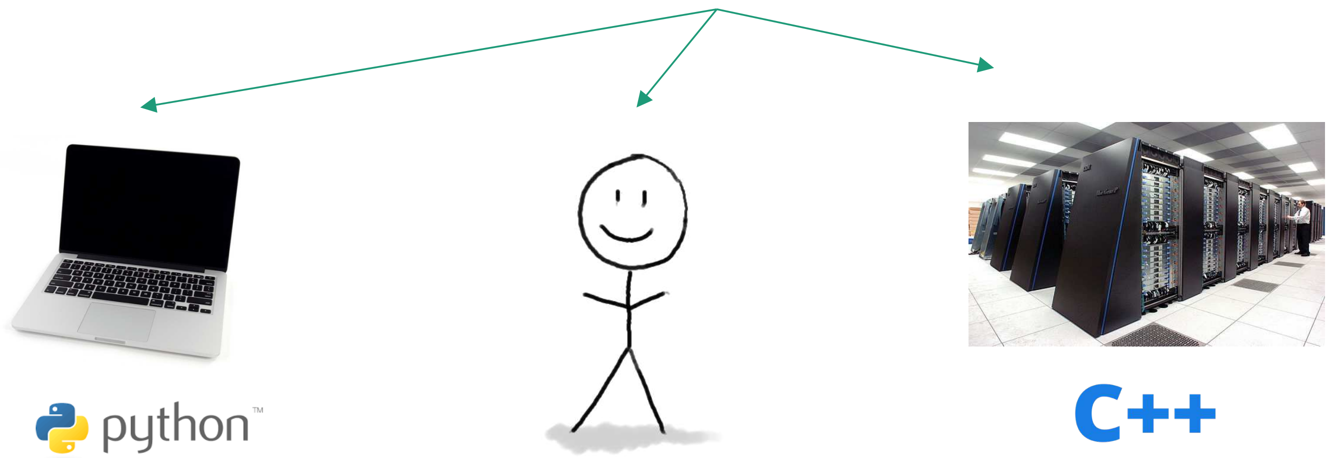
About n^2 one-digit operations

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…

Is that a useful answer?

- How do we measure the runtime of an algorithm?

All running the same algorithm…

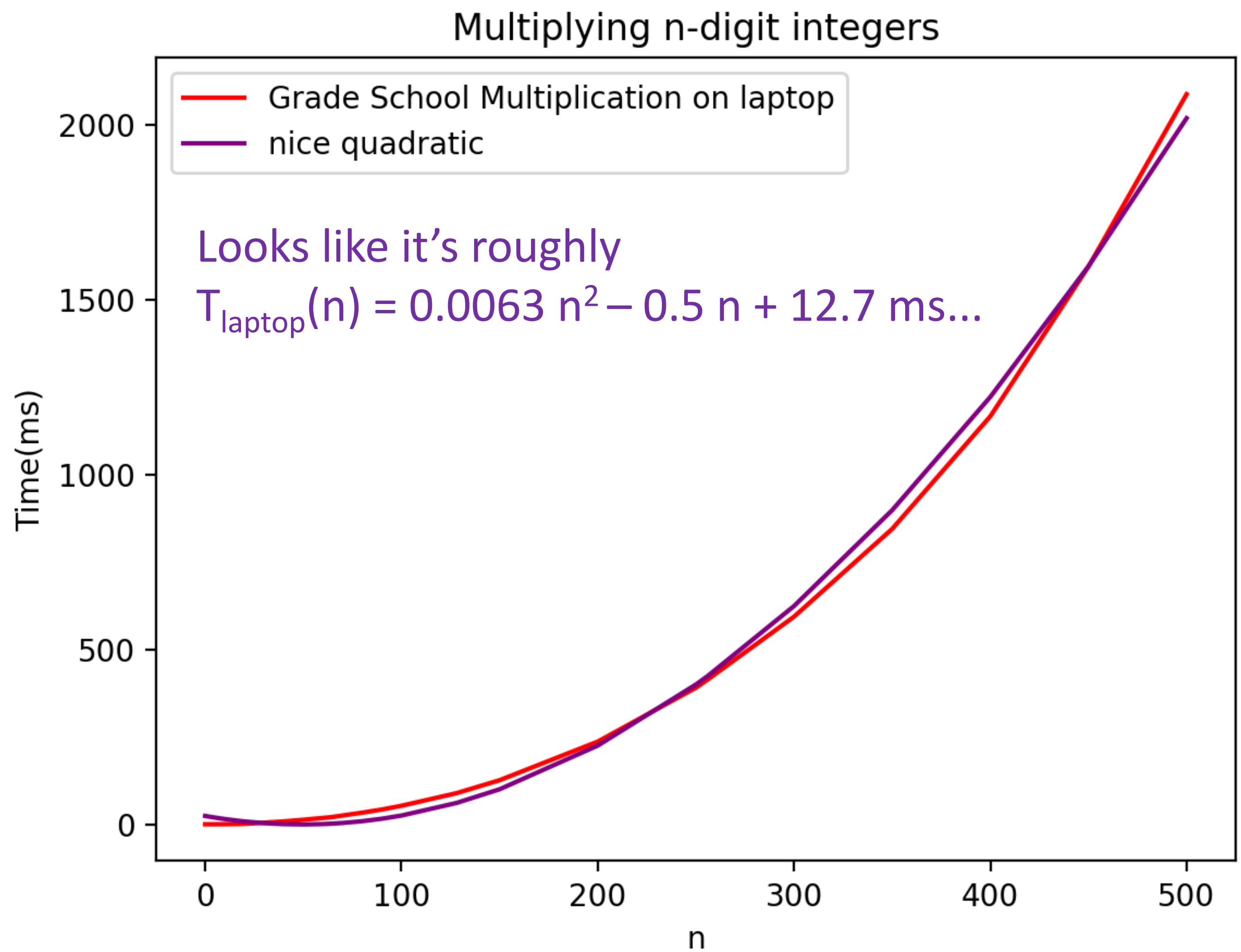


- We measure how the runtime scales with the size of the input.

For grade school multiplication,

with python, on my laptop…

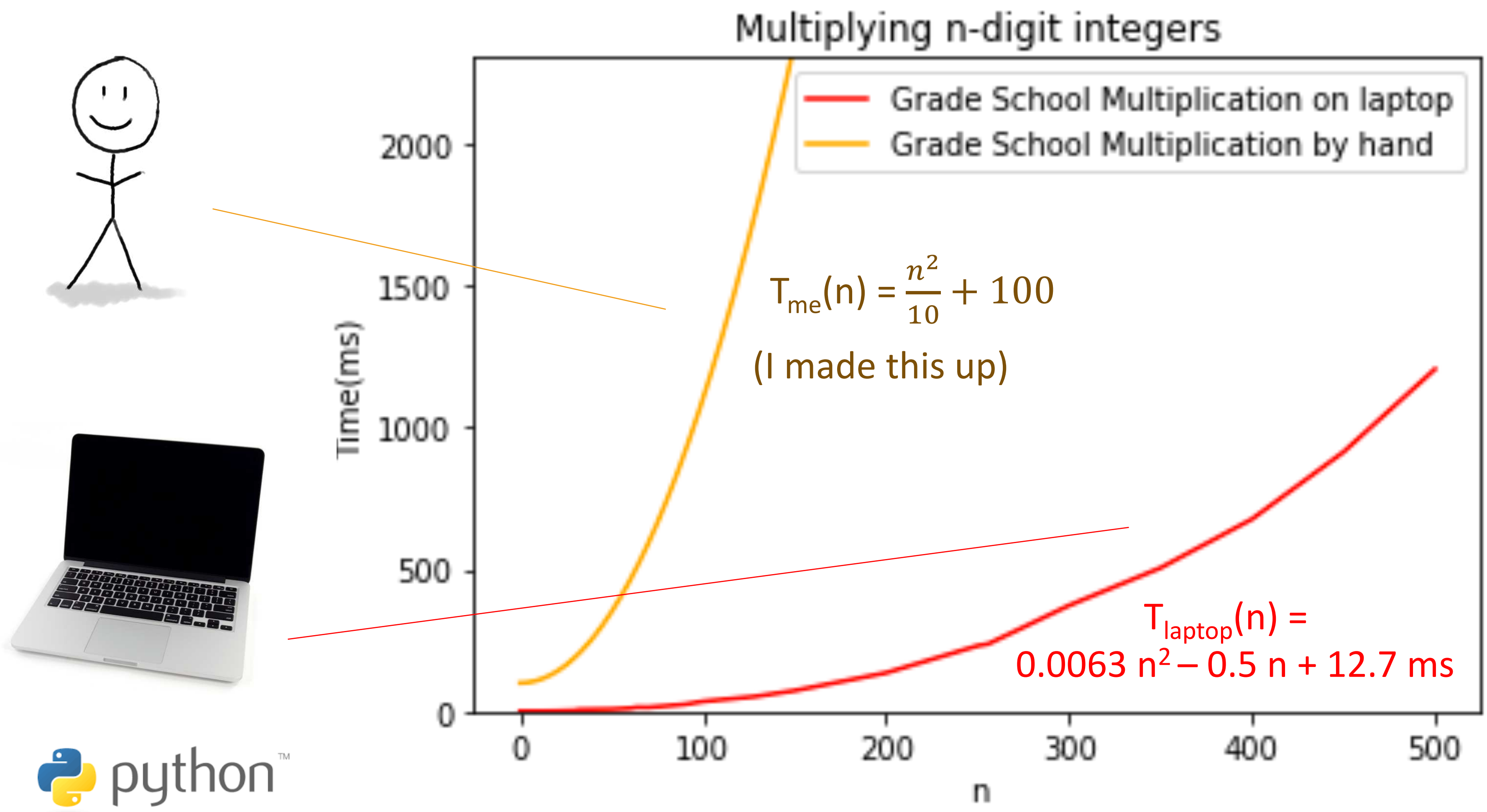
highly non-optimized



python™

I am a bit slower than my laptop

But the runtime scales like n^2 either way.



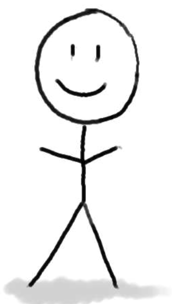
Asymptotic analysis

- How does the runtime scale with the size of the input?
  - Runtime of grade school multiplication scales like $\mathbf{n^2}$
- We'll see a more formal definition on Wednesday

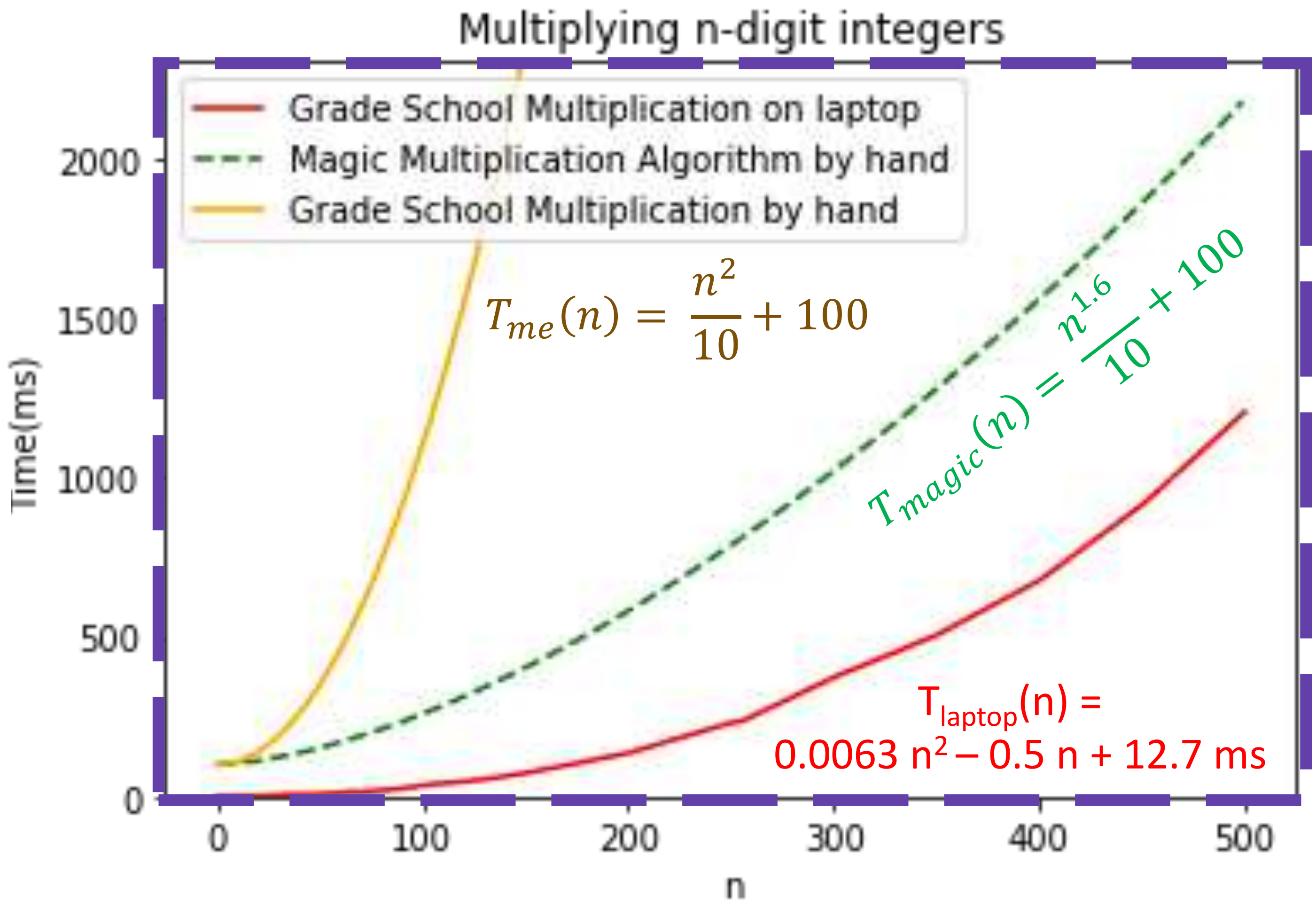
Is this a useful answer?

Hypothetically…

A magic algorithm that scales like $n^{1.6}$

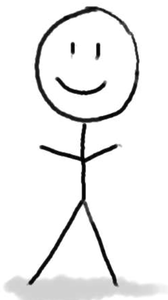


python™

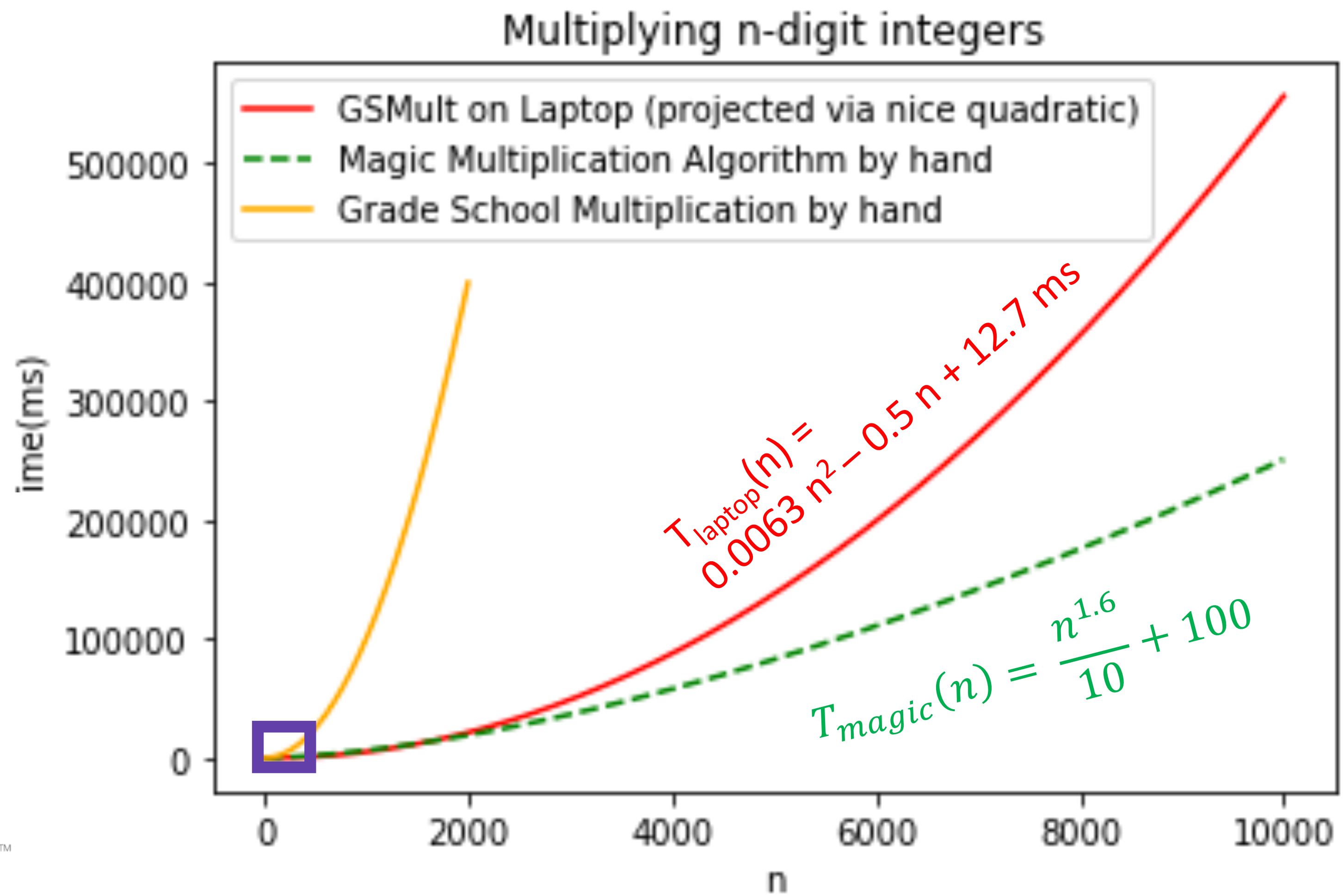


Let n get bigger…

No matter what the constant factors are, for large enough n, **it would be faster to do the magic algorithm by hand than the grade school algorithm on a computer!**



python™



Asymptotic analysis
is a useful notion…

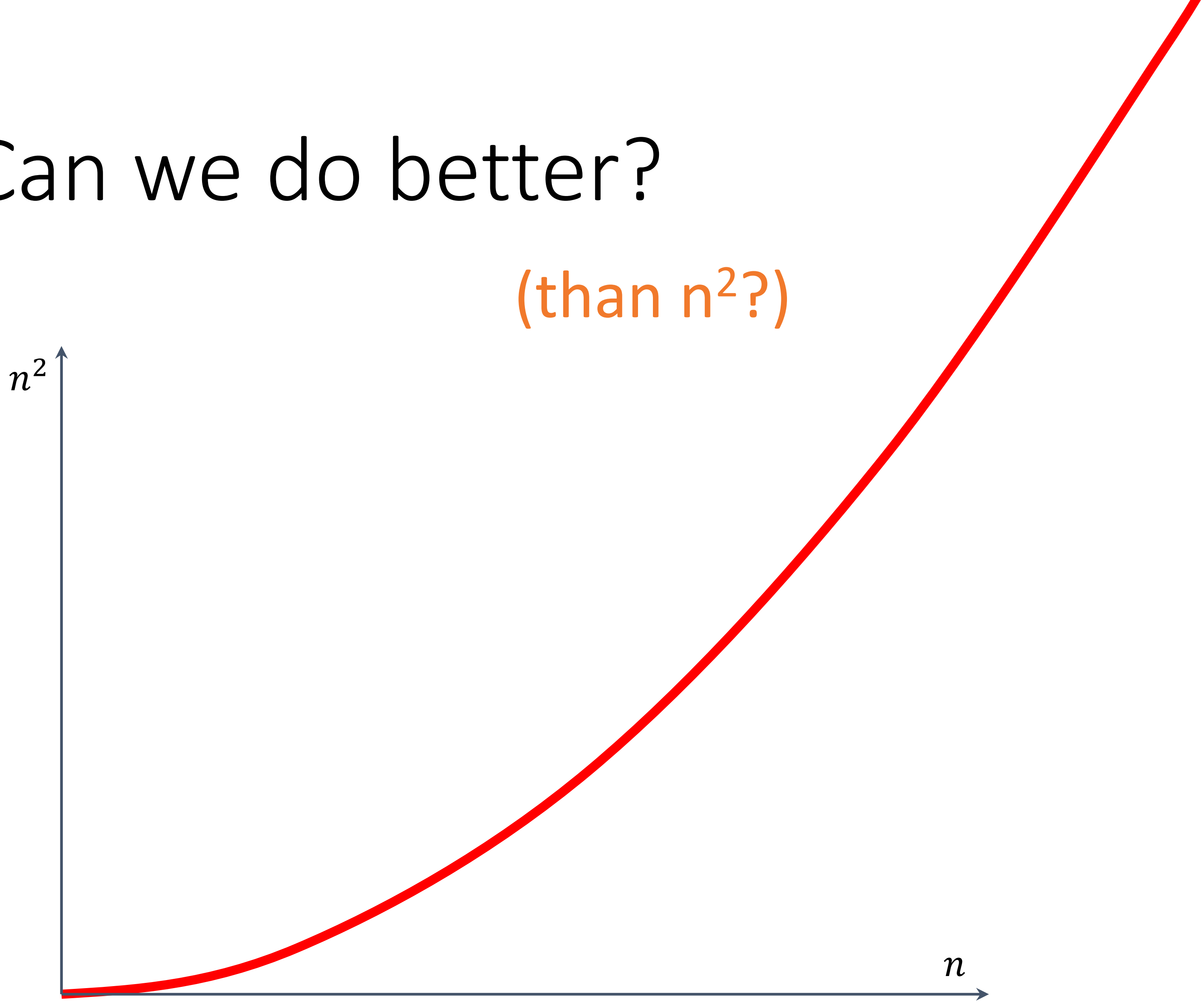
- How does the runtime scale with the size of the input?

- This is our measure of how "fast" an algorithm is.
- We'll see a more formal definition Wednesday

- So the question is…

Can we do better?

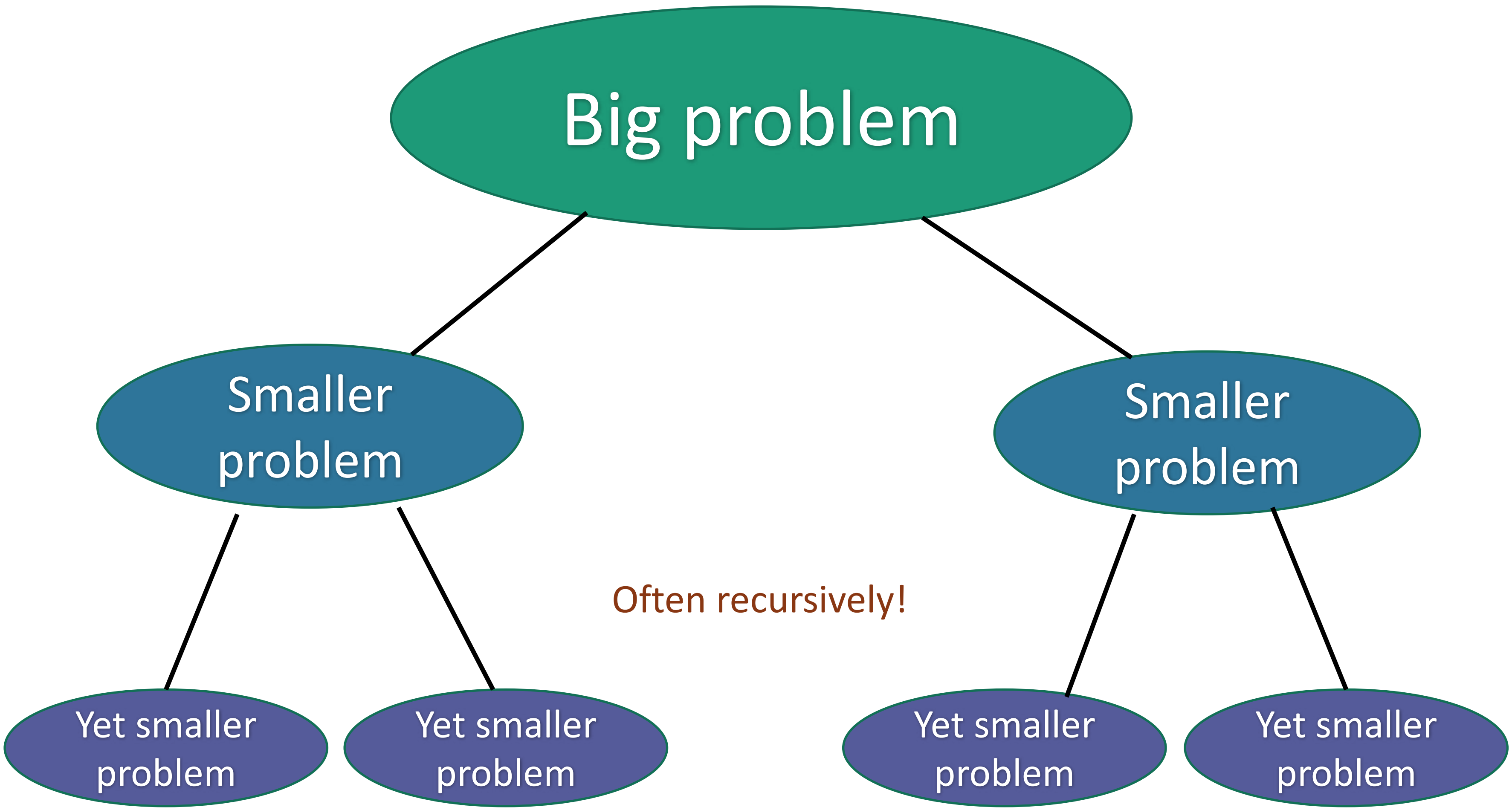
(than n^2?)



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:

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

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

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

Divide and conquer algorithm

not very precisely…

x,y are n-digit numbers

Multiply(x, y):

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

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

Say n is even…

- 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

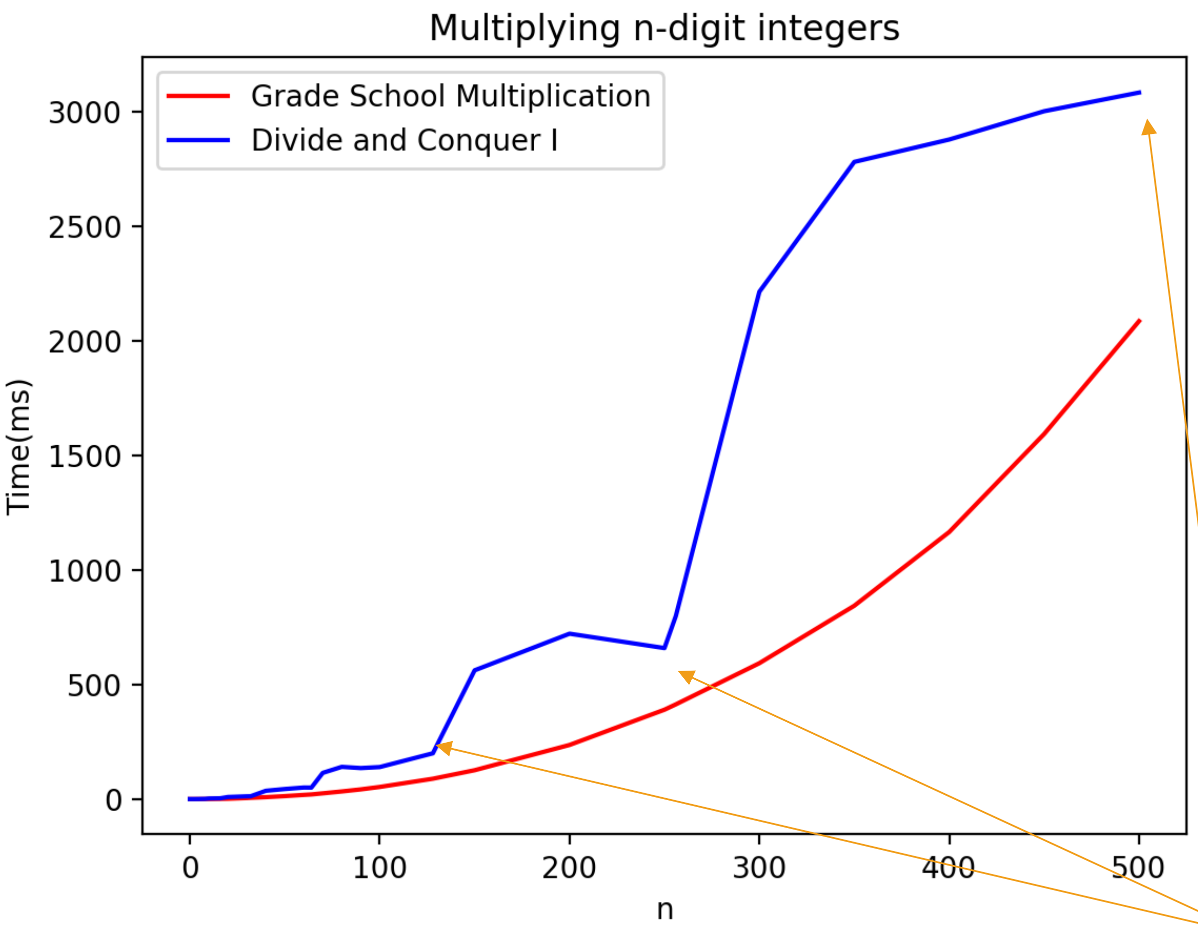
How long does this take?

- Better or worse than the grade school algorithm?
  - That is, does the number of operations grow like n^2 ?
  - More or less than that?

- 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!

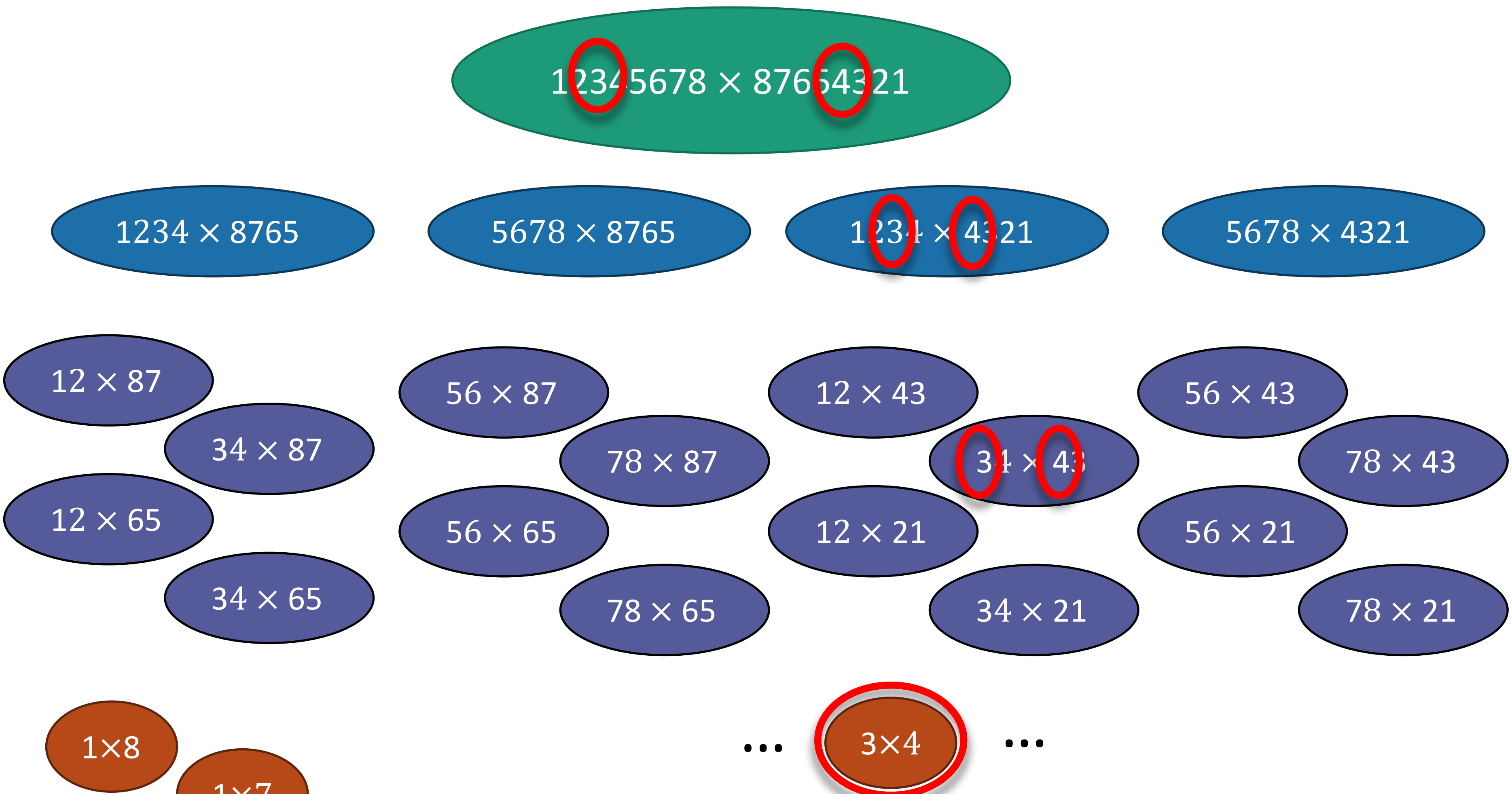
Plucky the Pedantic Penguin

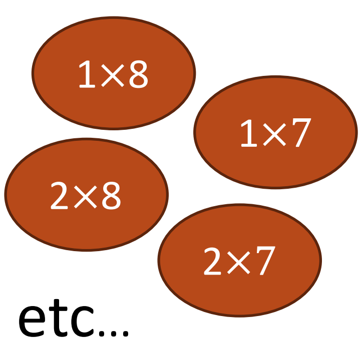
2. Try to understand the running time analytically

- Claim:

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

How many one-digit multiplies?





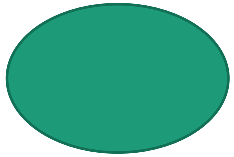
Claim: there are n^2 one-digit problems.

Every pair of digits still gets multiplied together separately.

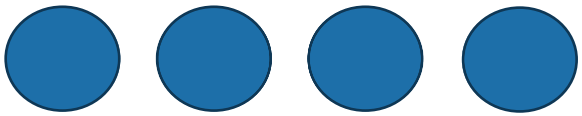
So the running time is still at least n^2.

Another way to see this*

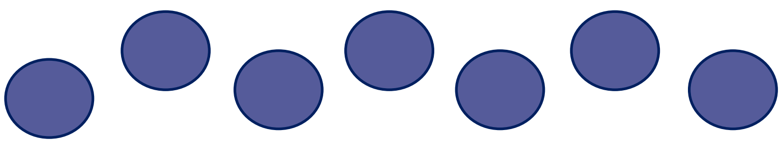
*we will come back to this sort of analysis later and still more rigorously.



1 problem
of size n

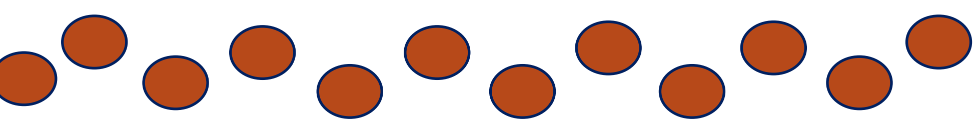


4 problems
of size n/2



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

…



$\underline{\quad n^2 \quad}$ 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.

This is just a lower bound – we're just counting the number of size-1 problems!

Another way to see this

This slide skipped in class, for reference only.

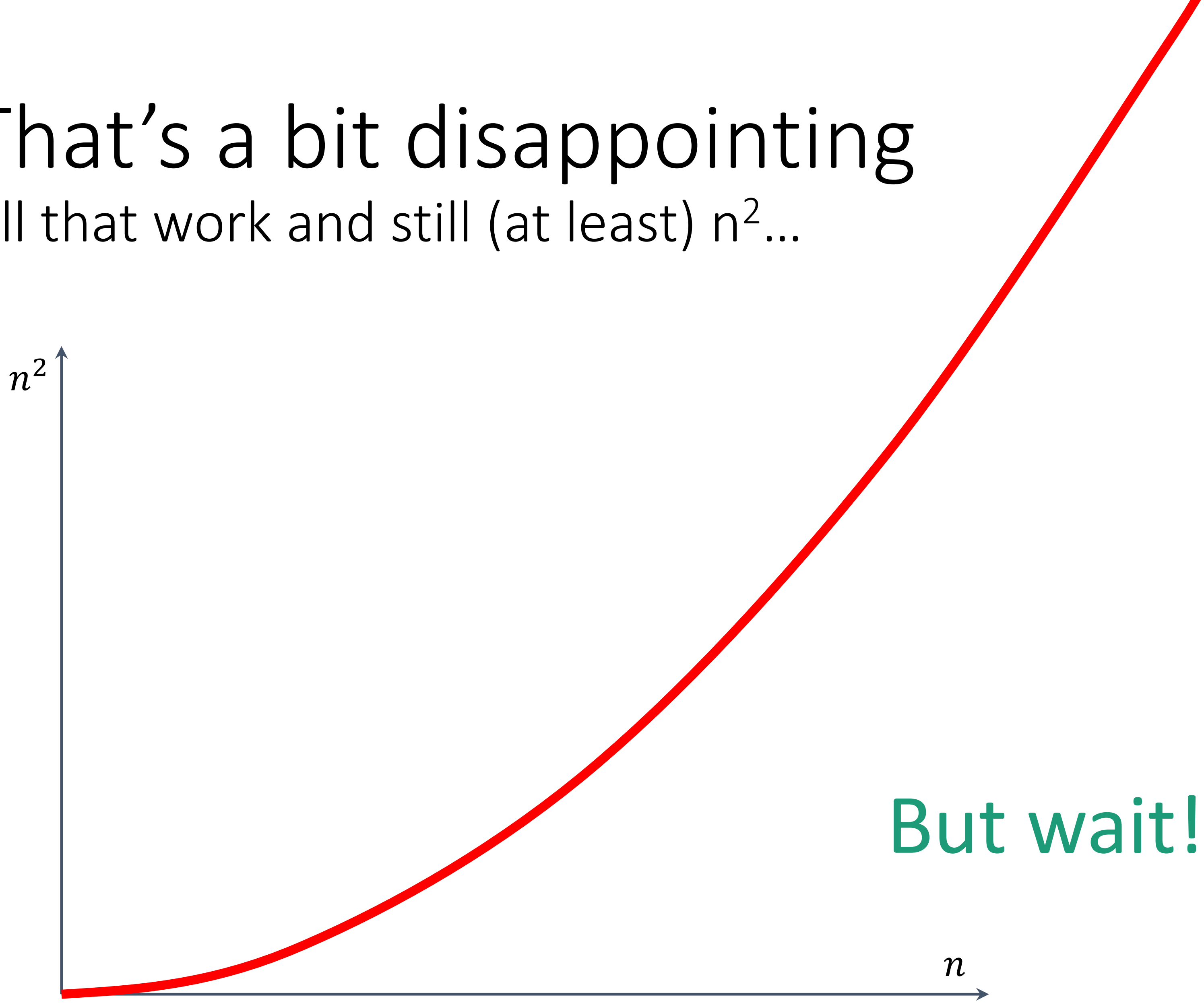
- Let T(n) be the time to multiply two n-digit numbers.
- Recurrence relation:
  - $T(n) = 4 \cdot T\left(\frac{n}{2}\right)$ + (about n to add stuff up)

Ignore this term for now…

$$
\begin{aligned}
T(n) &= 4 \cdot T(n/2) \\
&= 4 \cdot (4 \cdot T(n/4)) && 4^2 \cdot T(n/2^2) \\
&= 4 \cdot (4 \cdot (4 \cdot T(n/8))) && 4^3 \cdot T(n/2^3) \\
&\vdots \\
&= 2^{2t} \cdot T(n/2^t) && 4^t \cdot T(n/2^t) \\
&\vdots \\
&= n^2 \cdot T(1). && 4^{\log_2(n)} \cdot T(n/2^{\log_2(n)})
\end{aligned}
$$

That’s a bit disappointing

All that work and still (at least) 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 recurse three times instead of four…

Karatsuba integer multiplication

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

Subtract these off

get this

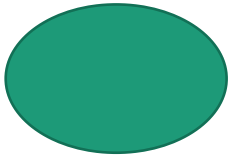
$$(a+b)(c+d) = ac + bd + bc + ad$$

- Assemble the product:

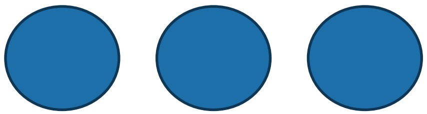
$$xy = (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d)$$
$$= ac \cdot 10^n + (ad + bc)10^{n/2} + bd$$

✓ ✓ ✓

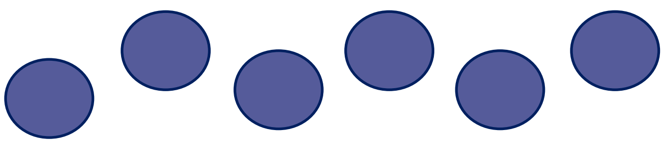
What’s the running time?



1 problem
of size n

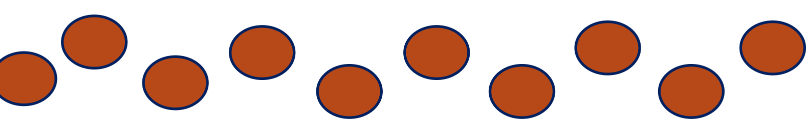


3 problems
of size n/2



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

…



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

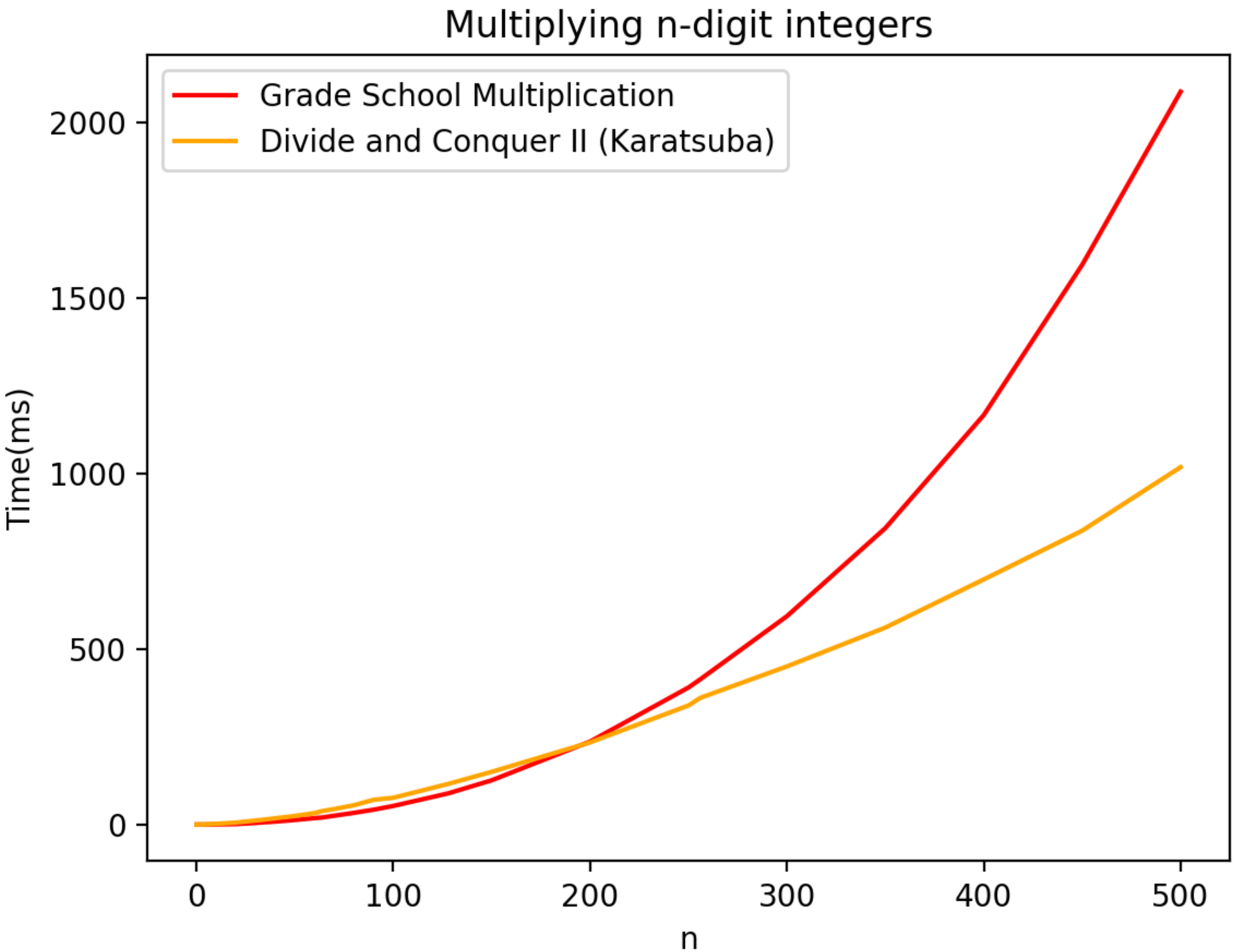
- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So we do this $\log_2(n)$ times and get…

$3^{\log\_2(n)} = n^{\log\_2(3)} \approx n^{1.6}$
problems of size 1.

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

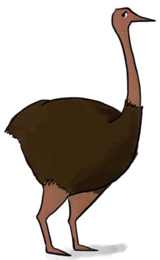
This is much better!

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.
  - This scales like $n^{1.465}$



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).**

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

Ollie the Over-achieving Ostrich

Siggi the Studious Stork

- **Schönhage–Strassen** (1971):
  - Scales like $n \log(n) \log\log(n)$
- **Furer** (2007)
  - Scales like $n \log(n)\ 2^{\log^*(n)}$

[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
- Technique: Divide and conquer
- Meta points:
    - How do we measure the speed of an algorithm?

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
  - **It might not be easy but it will be worth it!**
- 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
- Begin Asymptotics and Big-Oh notation

BEFORE Next time

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