

CS 161

Design and Analysis of Algorithms

Lecture 1:

Logistics, introduction, and multiplication!

Slides originally created by Mary Wootters.

How was your break?

The big questions

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



Who are we?

Instructors

EthiCS



Moses
Charikar



Nima
Anari



Diana
Acosta-Navas



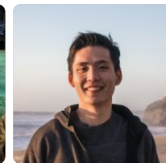
Akash
Velu



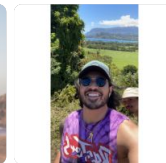
Amrita
Palaparthi



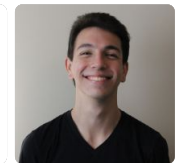
Apoorva
Dixit



Austin
Wang



Bharath
Namboothiry



Felipe
Godoy

Course Coordination

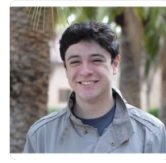


Amelie
Byun



John
Cho

Awesome TAs



Ivan
Villa-Renteria



Jeff
Z.
HaoChen



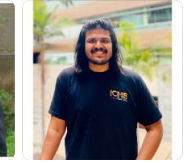
Jeffrey
Hu



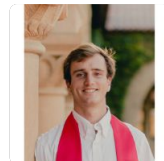
Lucy
Lu



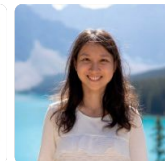
Rishi
Agarwal



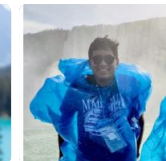
Rishu
Garg



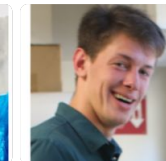
Robert
Thompson



Ruiqi
Wang



Shubham
Anand
Jain



Toby
Frager

Who are you?

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

Concentrating in:

- Art Practice
- Bioengineering
- Biology
- Biomedical Informatics
- Chemical Eng.
- Civil & Env. Eng.
- Comparative Literature
- Computation and Mathematical Eng.
- Computer Science
- Earth Systems
- East Asian Studies
- Economics
- Electrical Eng.
- Engineering
- Energy Resources Eng.
- History
- Material Sci & Eng.
- Math
- Math & CS
- Mechanical Eng.
- MS&E
- Philosophy
- Physics
- Political Science
- Psychology
- Sociology
- Spanish
- Statistics
- Symbolic Systems
- Theater & Performative Studies
- Undeclared

Where are you?

Why are we here?

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

Yay Algorithms!

You are better equipped to
answer this question, but I'll give
it a go ...

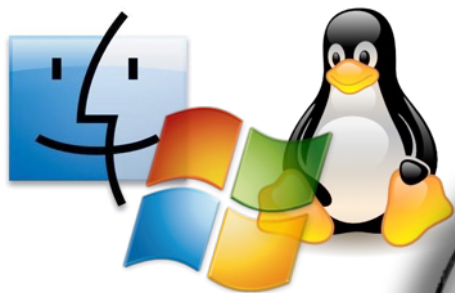
Why are you here?

- 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)

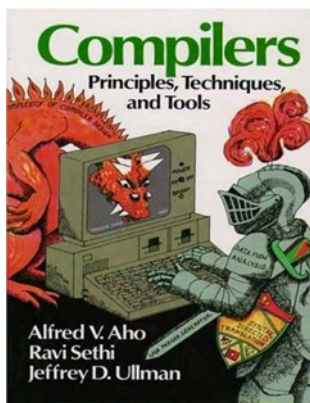


The
Algorithmic
Lens

229)



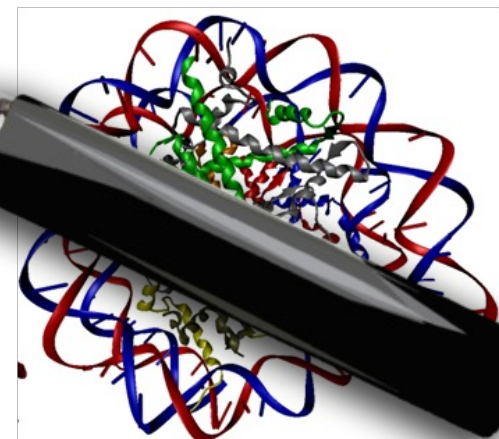
Cryptography (CS 255)



Compilers (CS 143)



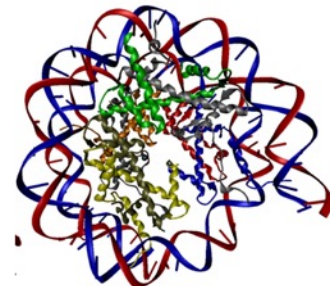
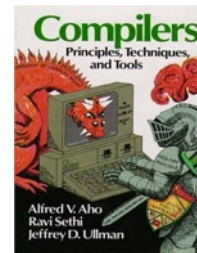
Networking (CS 144)



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!**
- Many **exciting research questions!**

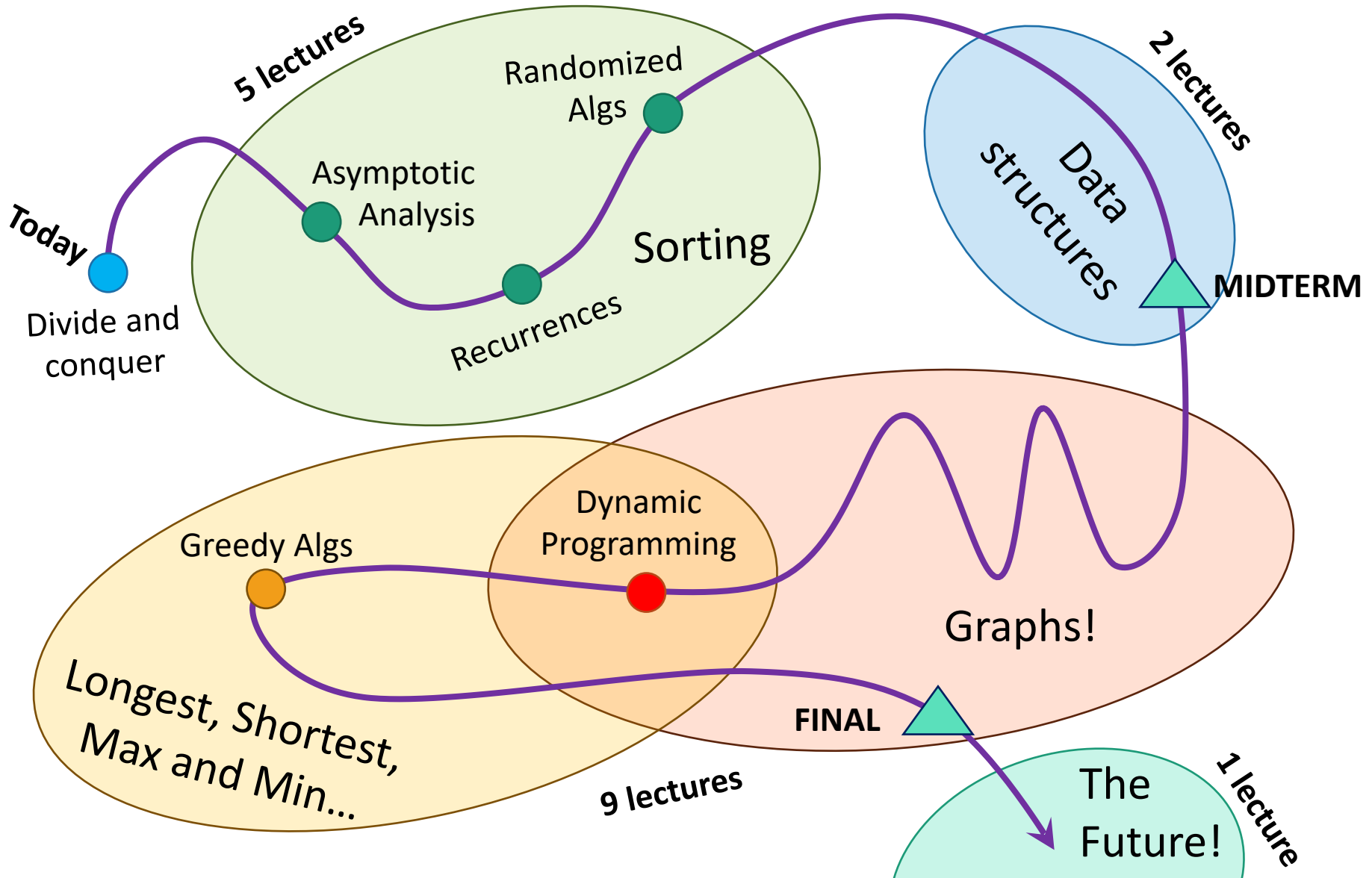
What's going on?

- Course goals/overview
- Logistics

Course goals

- The **design and analysis** of algorithms
 - They 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

Roadmap

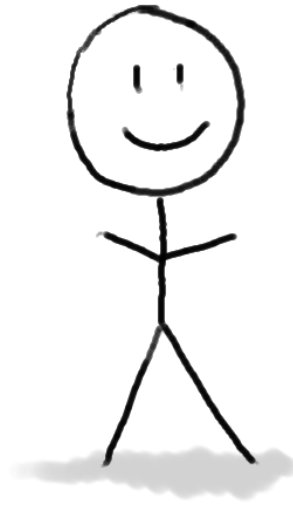


Our guiding questions:

Does it work?

Is it fast?

Can I do better?



Our internal monologue...

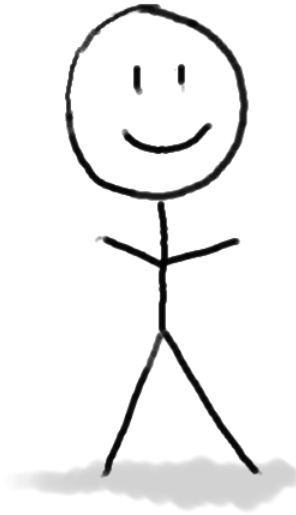
What exactly do we mean by better? And what about that corner case? Shouldn't we be zero-indexing?



Plucky the Pedantic Penguin

Detail-oriented
Precise
Rigorous

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



Lucky the Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavy

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!

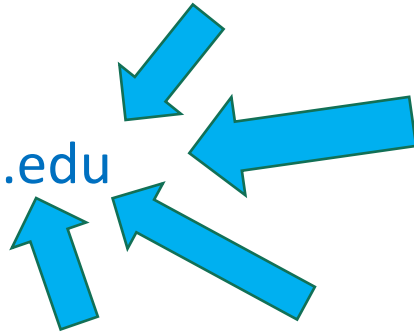
Both sides are necessary!

Aside: the bigger picture

- Does it work?
- Is it fast?
- Can I do better?
- **Should it work?**
- **Should it be fast?**
- We want to reduce crime.
- It would be more “efficient” to put cameras in everyone’s homes/cars/etc.
- We want advertisements to reach to the people to whom they are most relevant.
- It would be more “efficient” to make everyone’s private data public.
- We want to design algorithms, that work well, on average, in the population.
- It would be more “efficient” to focus on the majority population.

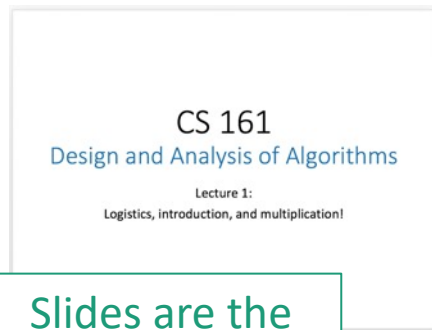
Course elements and resources

- Course website:
 - cs161.stanford.edu
- Lectures
 - Pre-Lecture Exercises
 - Lecture Notes
 - IPython Notebooks
 - Concept Check Questions
- References
- Sections
- Homework
- Exams
- Office Hours and Ed



Lectures

- Mon/Wed 9:30am-10:50am (Nvidia Auditorium)
 - Live on Zoom (you can ask questions)
 - Note: lecture ends at 10:50am and NOT 11:20am
- Resources available:
 - Pre-lecture exercises
 - Slides, Videos, Notes, IPython notebooks, Concept Check Qns



CS 161
Design and Analysis of Algorithms

Lecture 1:
Logistics, introduction, and multiplication!

Slides are the slides from lecture.



CS 161 (Stanford, Winter 2023) Lecture 1

Williams' lecture notes. Additional credits: J. Su, W. Yang, Gregory Avid Rubinstein, and mistakes to Nima Anari and Moses Charikar.

Introduction

<https://cs161.stanford.edu>. All course information is available

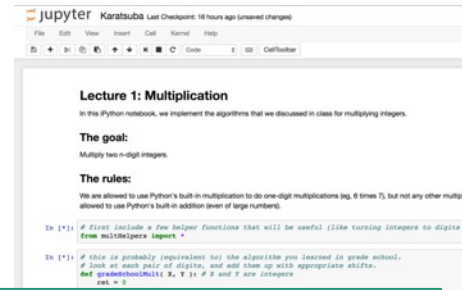
2 Why are you here?

Many of you are here because the class is required. But why is it required?

1 Algorithms are fundamental to all areas of CS: Algorithms are the backbone of...

Videos from lecture are available!

Hand-outs and references have mathy details that slides may omit



jupyter Karatsuba Last checkpoint: 16 hours ago (unreset changed)

File Edit View Insert Cell Kernel Help

Lecture 1: Multiplication

In this IPython notebook, we implement the algorithms that we discussed in class for multiplying integers.

The goal:
Multiply two n-digit integers.

The rules:
We are allowed to use Python's built-in multiplication to do one-digit multiplications (eg, 6 times 7), but not any other multiplication. We are not allowed to use Python's built-in addition (event of large numbers).

```
In [1]: # first, include a few helper functions that will be useful (like turning integers to digits from multipeers import *

In [1]: # this is probably (equivalent to) the algorithm you learned in grade school.
# look at each pair of digits, and add them up with appropriate shifts.
def gradeSchoolMult(x, y):
    # x and y are integers
    ret = 0
```

IPython notebooks have implementation details that slides may omit.

Embedded EthiCS Lectures

- Pre-recorded videos (by Diana Acosta-Navas) with embedded concept check questions.
 - You are responsible for watching them.
- EthiCS questions will also be in your homework and your exams.
- More on EthiCS in a bit ...

How to get the most out of lectures

- **During lecture:**

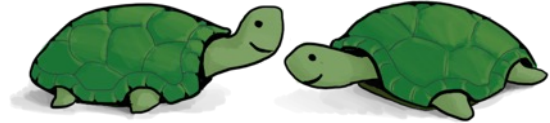
- Participate live (if you can), ask questions.
- Engage with in-class questions.

- **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.**


IPython Notebooks

- Lectures will occasionally be accompanied by IPython notebooks (but not homework)
 - For the next lecture, the *pre-lecture exercise* is to get started with Jupyter Notebooks and with Python.
 - See the course website for details.
- The goal is to make the algorithms (and their runtimes) more tangible.

Concept Check Questions

- Not part of grade; will not be graded
- Links to question sets part of resources for each lecture (via Lectures tab on website)

Lecture resources

- Lecture notes: [[PDF](#)]
 - Slides: [[PDF](#)] [[PowerPoint](#)]
 - Python notebook: [[Colab](#)] [[Zip](#)]
 - Concept check questions: [[Interactive SVG](#)] [[Solved PDF](#)]
- 

Multiplication Algorithms

[Reset Progress](#)

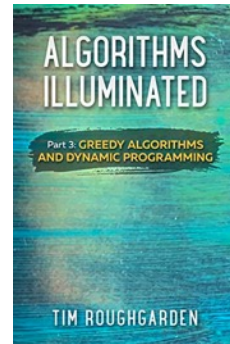
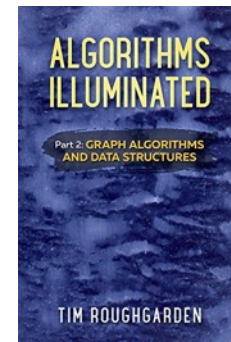
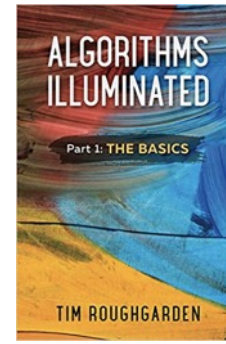
[Reveal Solutions](#)

1 Grade-school multiplication

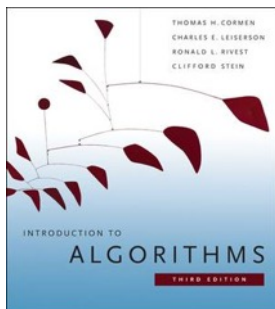
Suppose we multiply two n -digit integers $(x_1x_2\dots x_n)$ and $(y_1y_2\dots y_n)$ using the grade-school multiplication algorithm. How many pairs of digits x_i and y_j get multiplied in this algorithm?

- n^3
- $2n - 1$
- n^2

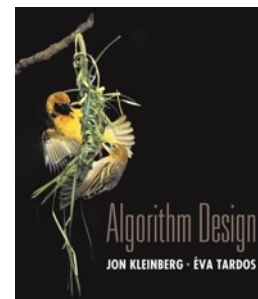
Optional References



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

Sections

Taught by your **amazing TAs** and will

- recap lecture
- show you **how to apply** the ideas you learned in lecture
- can occasionally cover new material

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

Homework

- Weekly assignments, posted Wednesday by 12:30pm, due the next Wednesday 11:59pm.
- First HW will be posted this Wednesday
- There are 8 total (no HW the week before midterm)
 - HW1, HW2, HW3: solo submissions
 - HW4, HW5, HW6, HW7, HW8: solo/pair submissions

How to get the most out of homework

- HW has two parts: exercises and problems.
- Do the exercises on your own.
- Try the problems on your own **before** discussing it with classmates.
- If you get help from a CA during 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.**

Exams

- There will be a **midterm** and a **final**
 - **Midterm:** Thu Feb 16, 6:00pm–9:00pm
 - Covers lectures 1-7
 - **Final:** Thu Mar 23, 8:30am–11:30am
 - Covers everything, but more focus on lectures 8-18
- 8 homeworks, lowest score dropped
- Weighting: **HW** (50%), **Midterm** (20%), **Final** (30%)
- If you have a conflict with the midterm time, email cs161-win2223-staff@lists.stanford.edu **ASAP!!!!!!**

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
 - Some online, some in-person
 - They start later this week

Talk to each other!

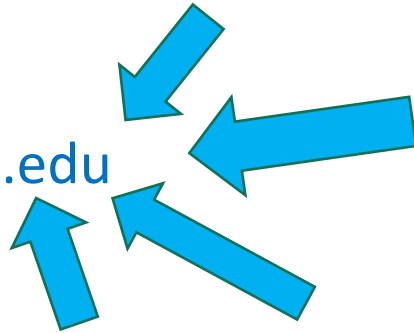
ed

CS 161 – Discussion

- Answer your peers' questions on Ed!

Course elements and resources

- Course website:
 - cs161.stanford.edu
- Lectures
 - Pre-Lecture Exercises
 - Lecture Notes
 - IPython Notebooks
 - Concept Check Questions
- References
- Sections
- Homework
- Exams
- Office Hours and Ed



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 (except for a couple of slides on collaboration and the honor code)

RULES

Collaboration

- We encourage collaboration on homework (but strongly recommend you do exercises on your own)
- Valid and invalid modes of collaboration are detailed on the course website.
 - Briefly, you can exchange ideas with classmates but must write up solutions on your own (except for pair submissions from HW4 onwards).
 - For pair submissions (HW4 onwards), each person must understand and contribute to the solution for **each individual problem**.
- You must cite all collaborators, as well as all sources used (outside of course materials).

Honor code

- Updated two years ago: **“In all cases, it is not permissible for students to enter exam questions into any software, apps, or websites. Accessing resources that directly explain how to answer questions from the actual assignment or exam is a violation of the Honor Code.”**
- Course policy for homework: **“In all cases, it is not permissible for students to enter *homework* questions into any software, apps, or websites. Accessing resources that directly explain how to answer questions from the actual assignment or exam is a violation of *course policy*.”**

Bug bounty!



- We hope all PSETs and slides will be bug-free.
- **However, we sometimes make 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.



Bug Bounty Hunter

*So, typos like **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 (and all students)

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

OAE forms

- Please send OAE forms to

cs161-win2223-staff@lists.stanford.edu

Feedback!

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



Canvas

Ed

Gradescope

Feedback

Analysis of Algorithms

Winter 2023

Pravesh Charikar

How are you
approaching CS 161?

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?
 - Embedded Ethics?
- Can we multiply integers?
 - And can we do it quickly?



Introducing Embedded Ethics

Diana Acosta-Navas

Postdoctoral Fellow, *McCoy Family Center for Ethics in Society*
and *Institute for Human Centered Artificial Intelligence*

Other big questions

- Who am I?
- Why am I here?
- What is Embedded Ethics?
- What has ethics got to do with algorithms?



Who am I?

I'm Diana, a post-doctoral fellow at the *McCoy Family Center for Ethics in Society* and *Stanford Institute for Human-Centered Artificial Intelligence*

I finished my Ph.D. in Philosophy at Harvard University

I taught ethics at the Harvard Kennedy School of Government

I also became part of Embedded EthiCS@Harvard

Now I'm helping Stanford to develop our Embedded Ethics program

What is 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.



Elan the Ethical Emu

What is Embedded Ethics?

The Vision

- Responsible ethical reasoning is a highly valuable skill.
- One that needs to be integrated with technical, managerial, and other skills we apply in our professional lives
- Successfully integrating these skills requires a distributed pedagogy approach



Elan the Ethical Emu

What is Embedded Ethics?

What 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.



Elan the Ethical Emu

Where is Embedded Ethics?

Outside Stanford

- Harvard (2017)
- Georgetown (2017)
- Brown (2019)
- Northeastern (2019)
- MIT (2020)
- Andover (2021)
- ... and many other places

At Stanford

- CS106A
- CS106B
- CS107
- CS109
- CS221
- CS247
- CS147
- ... and more over the next two years

And what does ethics have to do with algorithms?

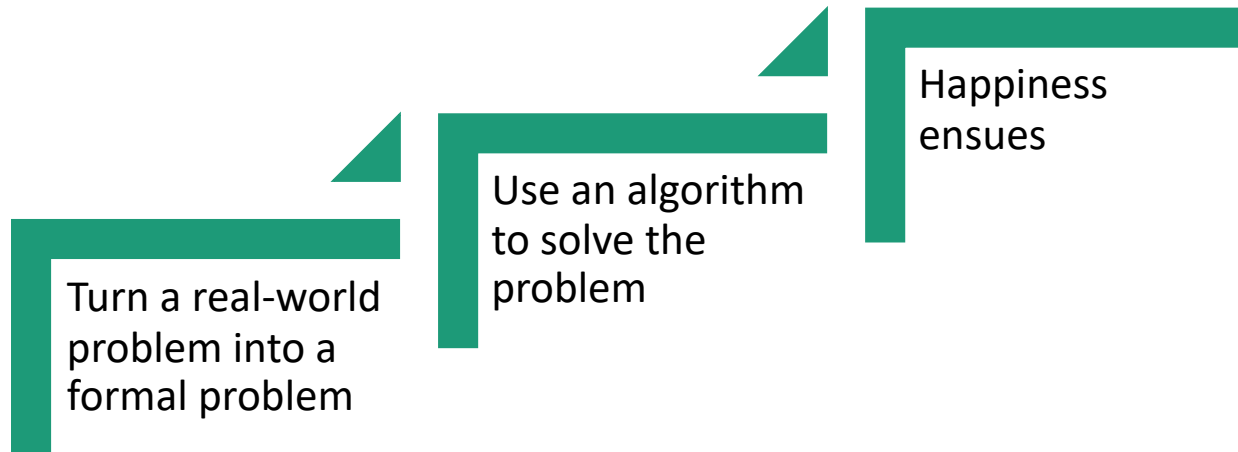


If algorithms are fundamental (which they are) ...

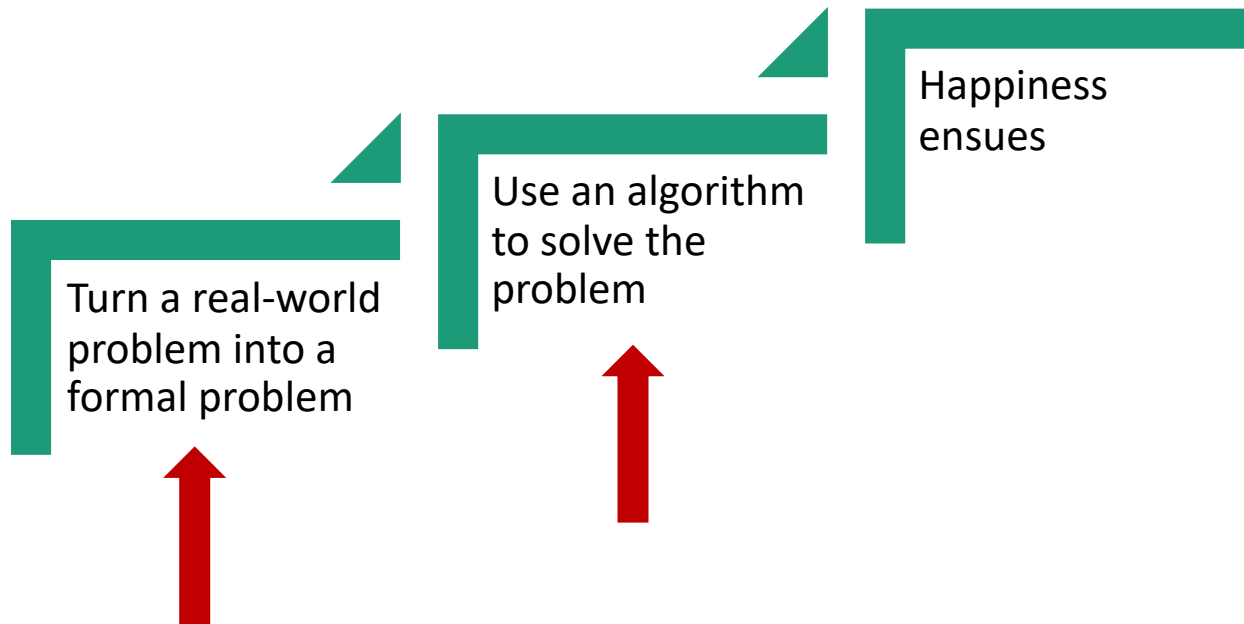
Then some of the most consequential choices you will make as computer scientists are:

- Deciding *which problem* to solve
 - This often is a huge ethical question
- Deciding how to turn that problem into something *algorithmically tractable*
 - This also can involve serious ethical decisions
- Deciding *which algorithm* to use to solve it and what tradeoffs to accept
 - Which often requires ethical reasoning

Algorithms & the Good



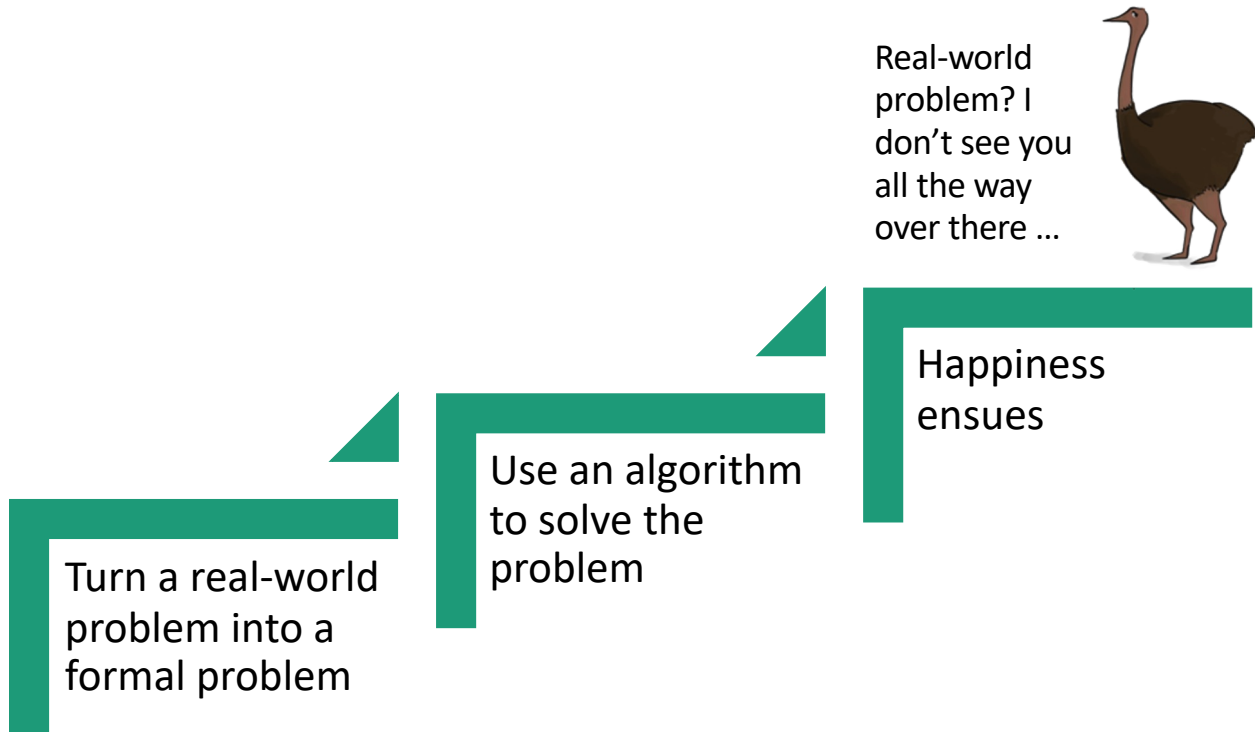
Algorithms & the Good



Some (potentially impactful) decisions:

We often need to ignore or change aspects of a real-world situation in order to turn it into an algorithmically solvable problem. For example, we can write an algorithm that sorts a numbered list without knowing what the numbers are numbers of.

- **Abstraction** is when we omit details of the real-world situation.
 - Omit the kind of thing being sorted by our algorithm, or what condition it is in, or what color it is, or how long it has been in the list.
- **Idealization** is when we deliberately change aspects of the real-world situation.
 - Round the numbers being sorted to make them whole numbers.



So how do we make sure we aren't losing important features of the real-world problem when we formalize it?

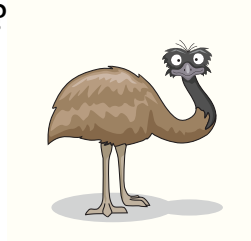


Turn a real-world problem into a formal problem

Use an algorithm to solve the problem

Happiness ensues

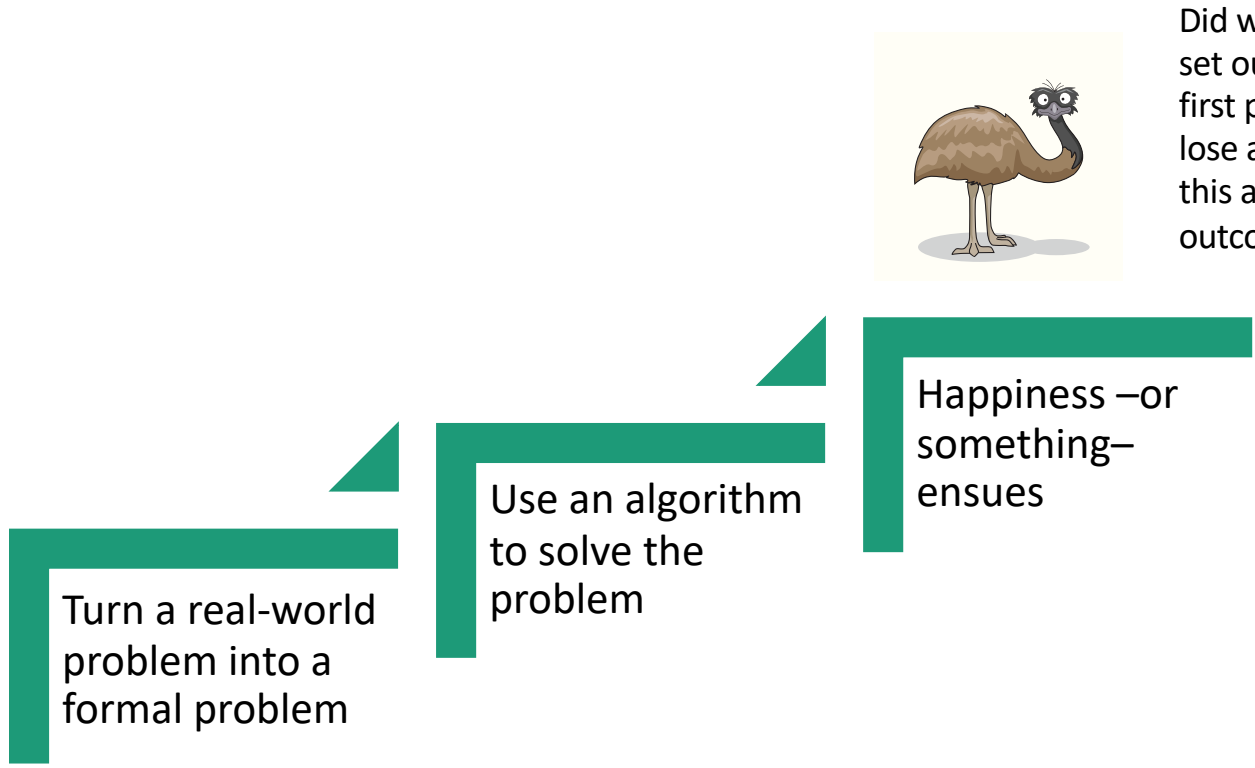
By the time you finish
161 you will have an
“algorithmic tool kit”–
which one is right for the
job?



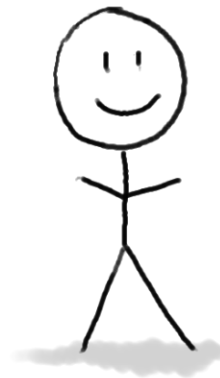
Turn a real-world
problem into a
formal problem

Use an algorithm
to solve the
problem

Happiness
ensues



Our guiding questions:



Does it work?

Is it fast?

Can I do better?

Can I do it right?

Thank you!

You can always email me at dacostan@stanford.edu

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?
 - Embedded Ethics?
- Can we multiply integers?
 - And can we do it quickly?

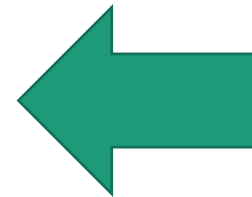


Course goals

- Think *analytically* about algorithms
- Flesh out an “*algorithmic toolkit*”
- Learn to *communicate clearly* about algorithms

Today’s goals

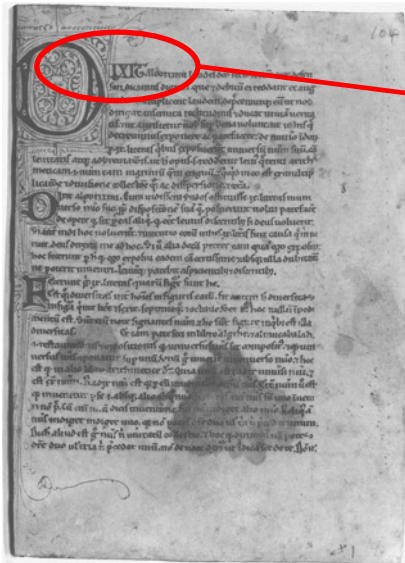
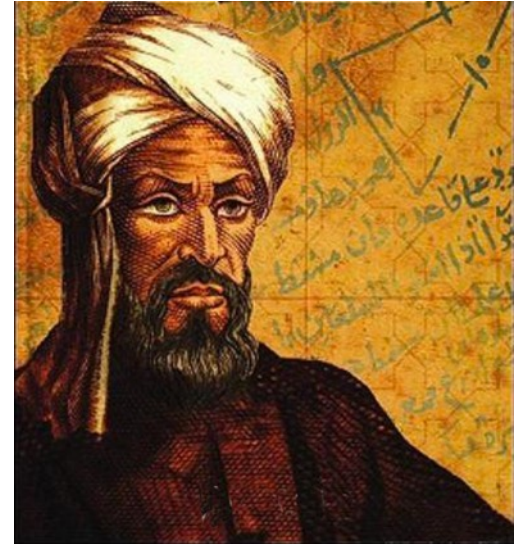
- Karatsuba Integer Multiplication
- Algorithmic Technique:
 - *Divide and conquer*
- Algorithmic Analysis tool:
 - *Intro to asymptotic analysis*



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 Abbasid 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

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

Integer Multiplication

n

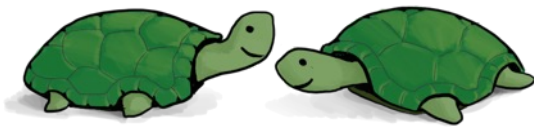
}

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...

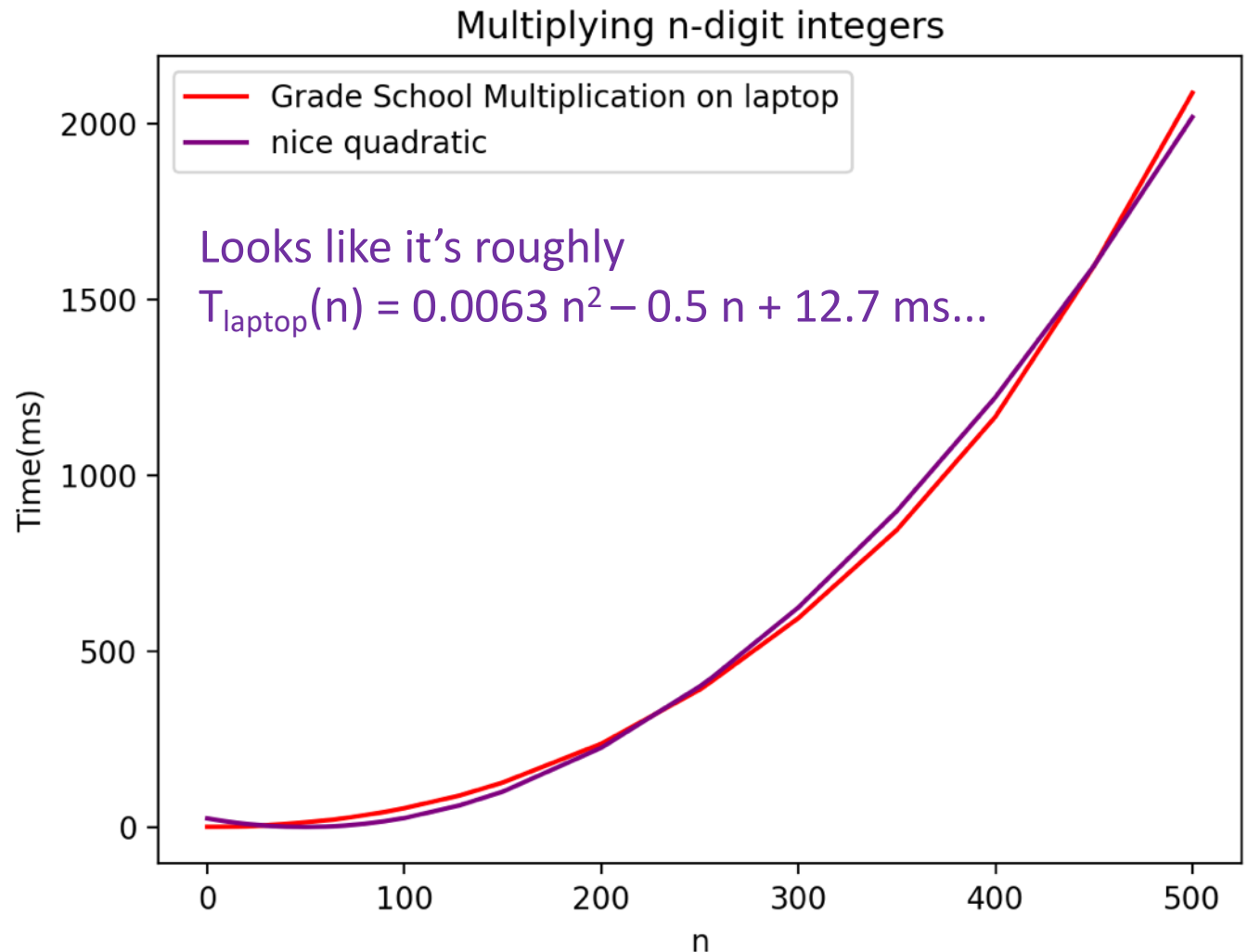
Big-Oh Notation

- We say that Grade-School Multiplication
“runs in time $O(n^2)$ ”
- Formal definition coming Wednesday!
- Informally, big-Oh notation tells us how the running time scales with the size of the input.

highly non-optimized

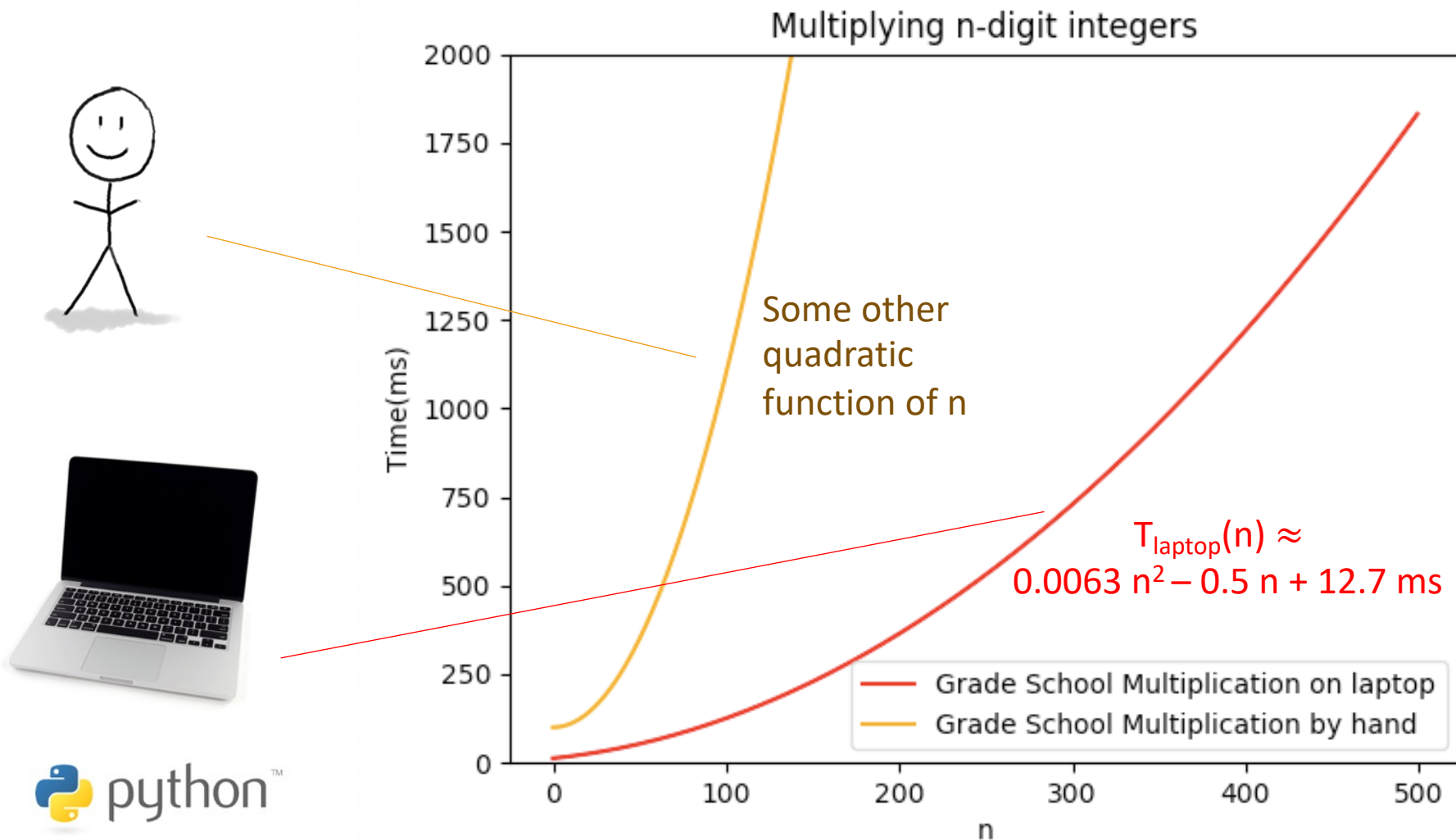
Implemented in Python, on a laptop

The runtime “scales like” n^2



Implemented by hand

The runtime still “scales like” n^2



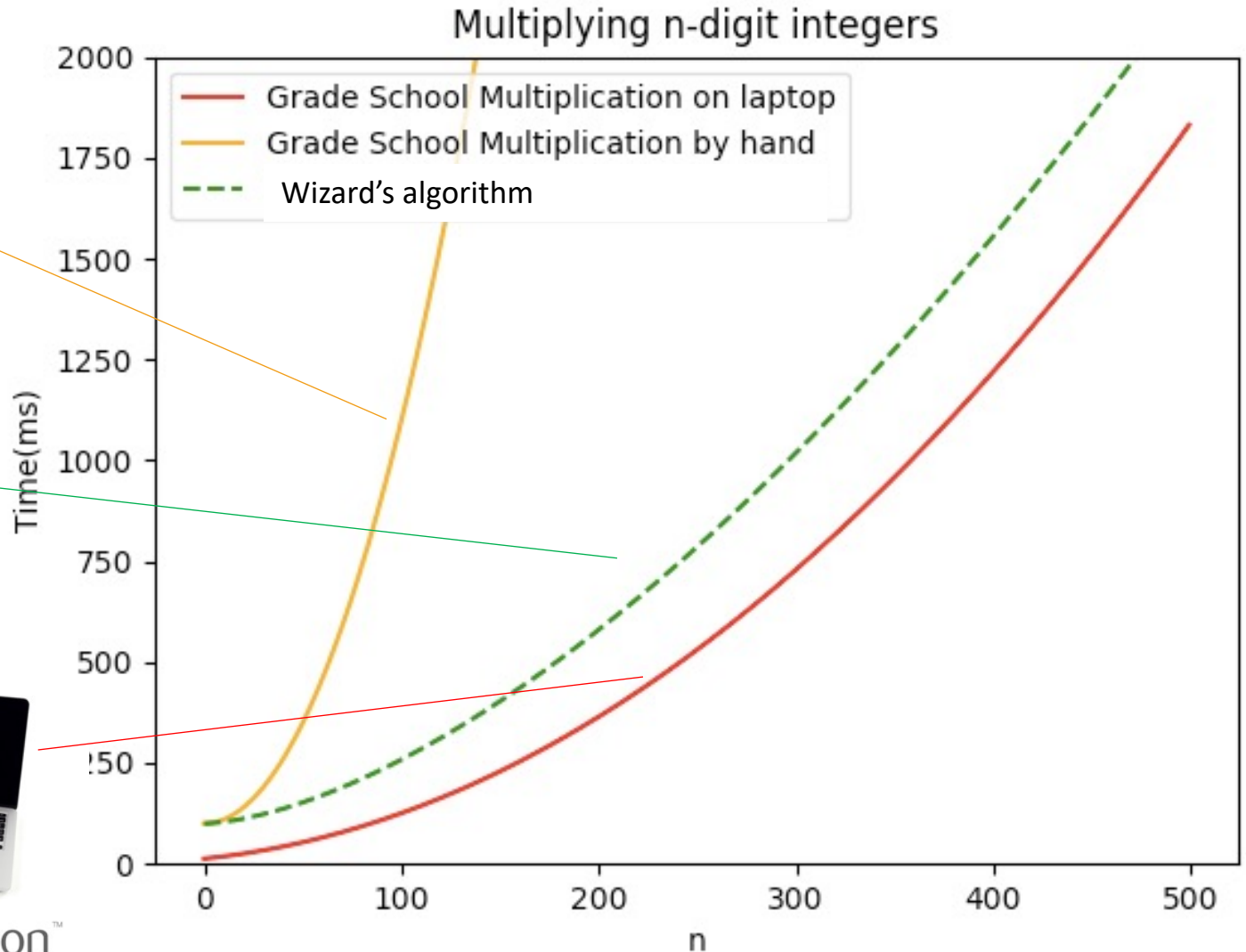
Why is big-Oh notation meaningful?



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




$$\approx .0063n^2$$




Let n get bigger...

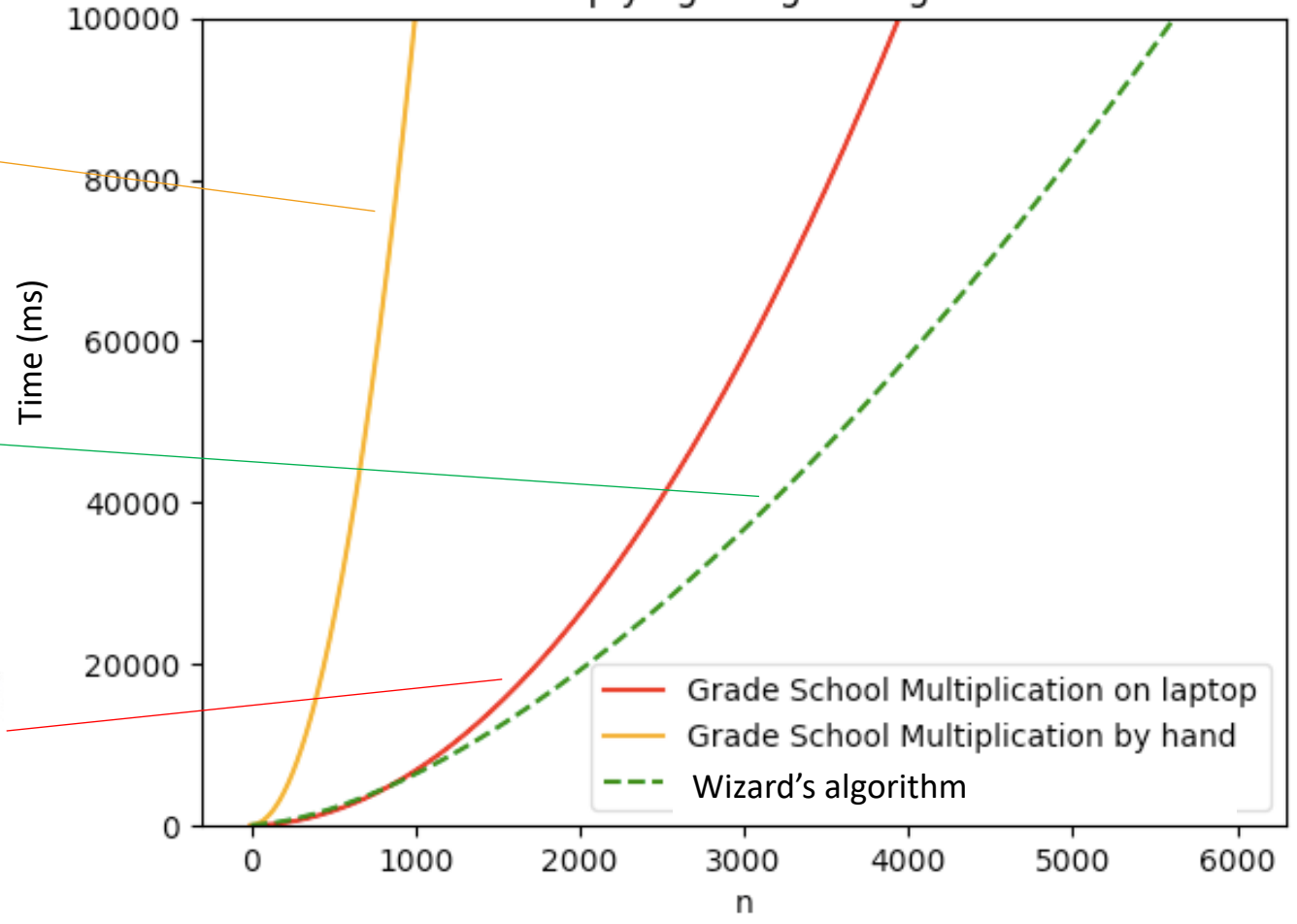
Multiplying n-digit integers



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



$\approx .0063n^2$

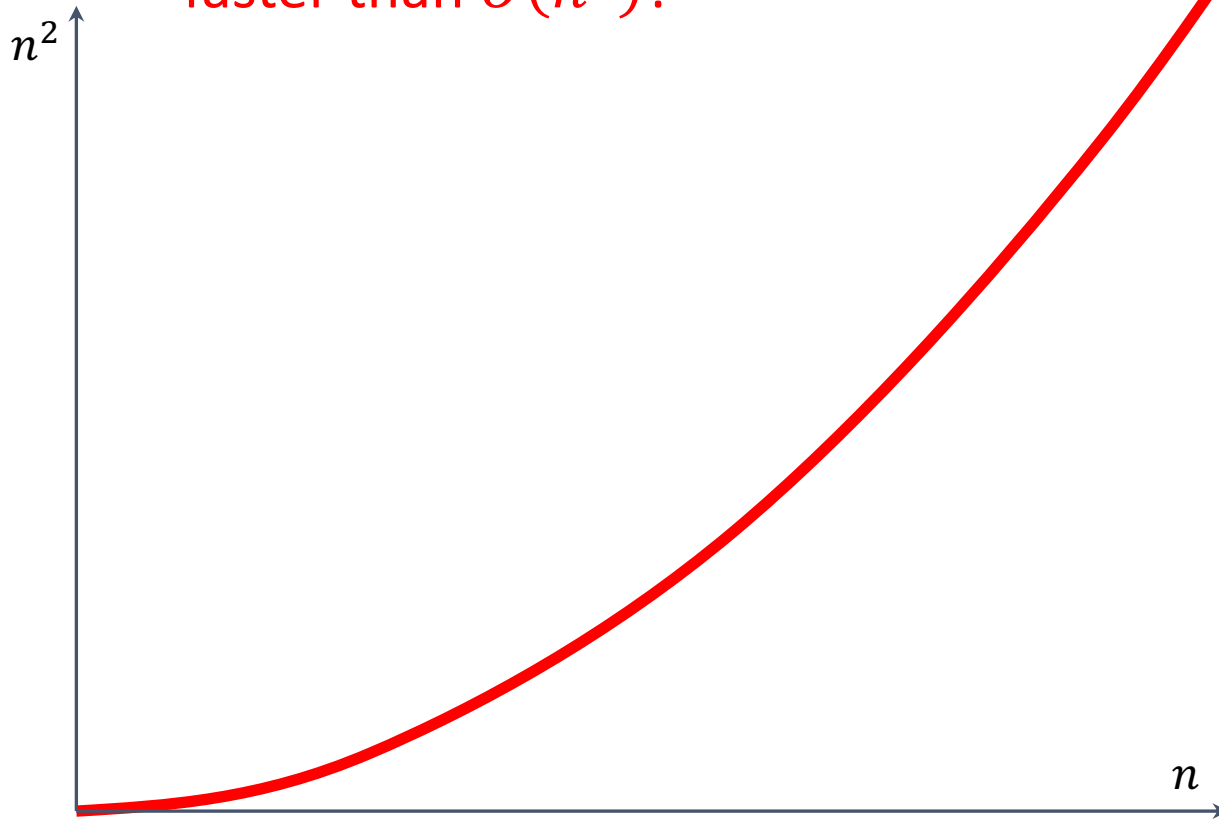


Take-away

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

Can we do better?

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

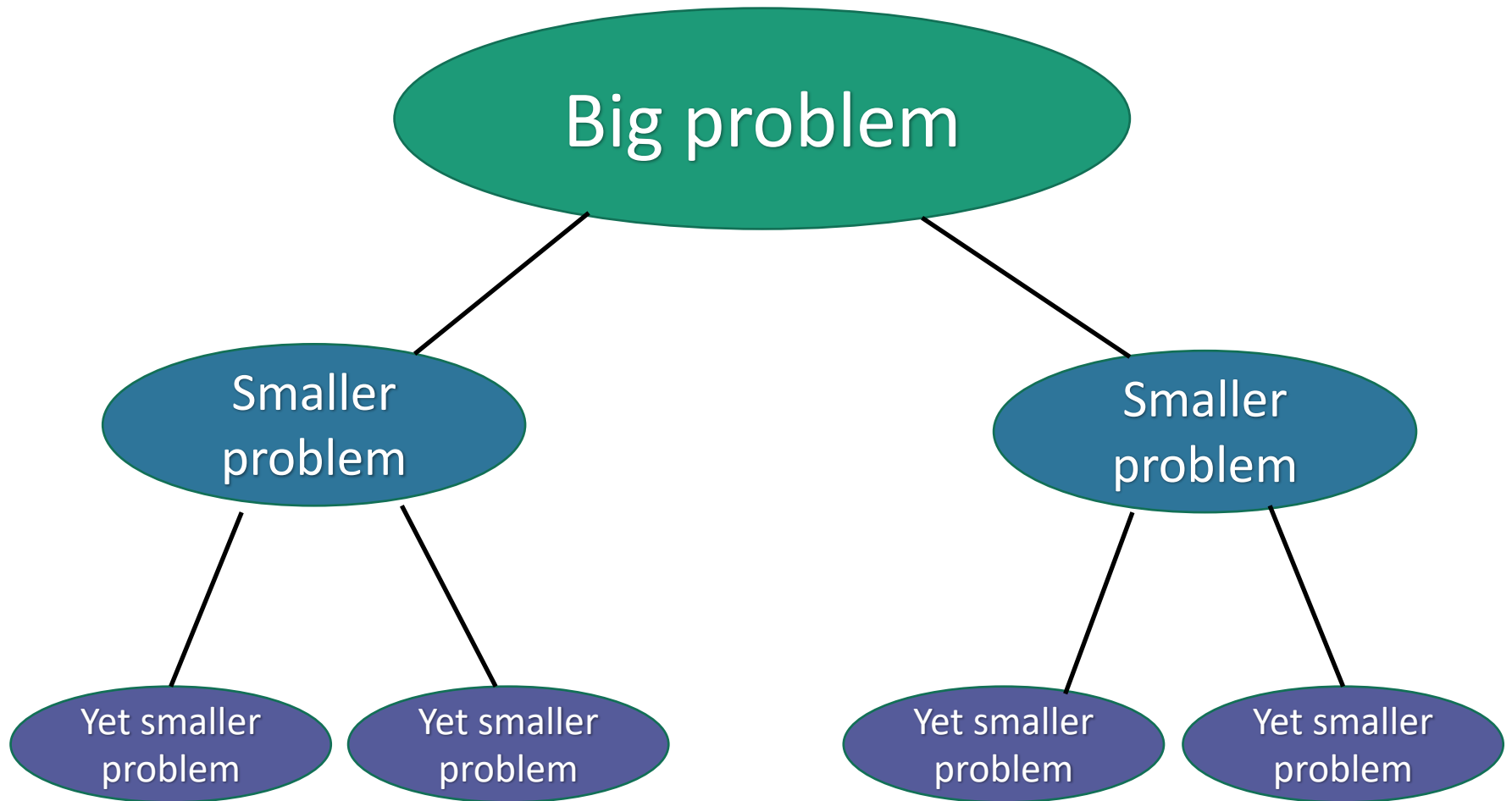


Let's dig into our algorithmic toolkit...



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:

$$[x_1x_2 \cdots x_n] = [x_1x_2 \cdots x_{n/2}] \times 10^{n/2} + [x_{n/2+1}x_{n/2+2} \cdots x_n]$$

$$\begin{aligned} x \times y &= (a \times 10^{n/2} + b)(c \times 10^{n/2} + d) \\ &= \underbrace{(a \times c)}_1 10^n + \underbrace{(a \times d + c \times b)}_2 10^{n/2} + \underbrace{(b \times d)}_4 \end{aligned}$$

One n -digit multiply



Four $(n/2)$ -digit multiplies



Divide and conquer algorithm

not very precisely...

(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 $x = a 10^{\frac{n}{2}} + b$

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

- Write $y = c 10^{\frac{n}{2}} + d$

- Recursively compute ac, ad, bc, bd :

- $ac = \mathbf{Multiply}(a, c)$, 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 "?

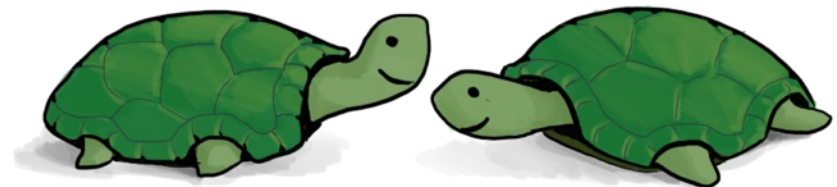


Think-Pair-Share

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

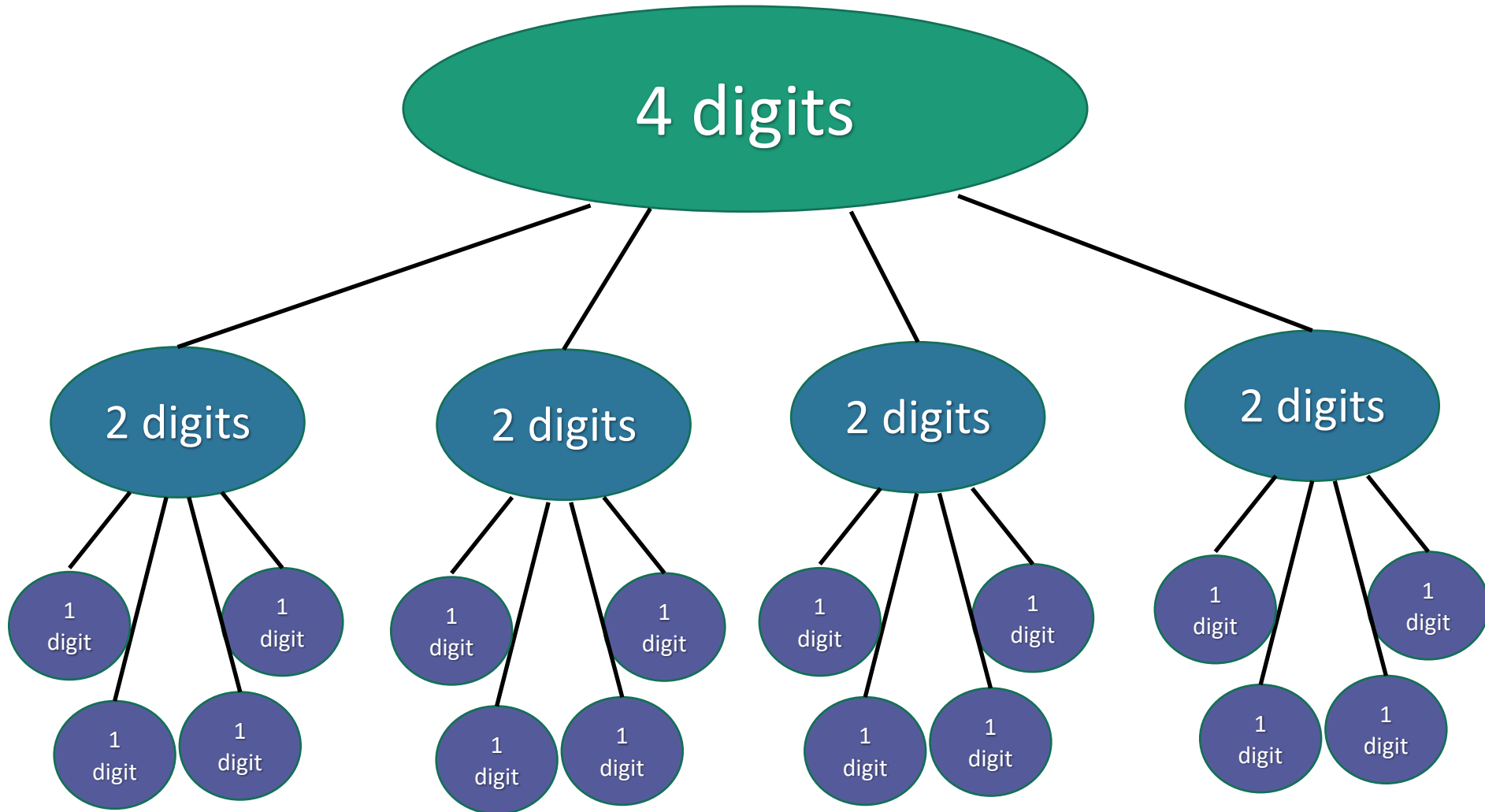
$$1234 \times 5678$$

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

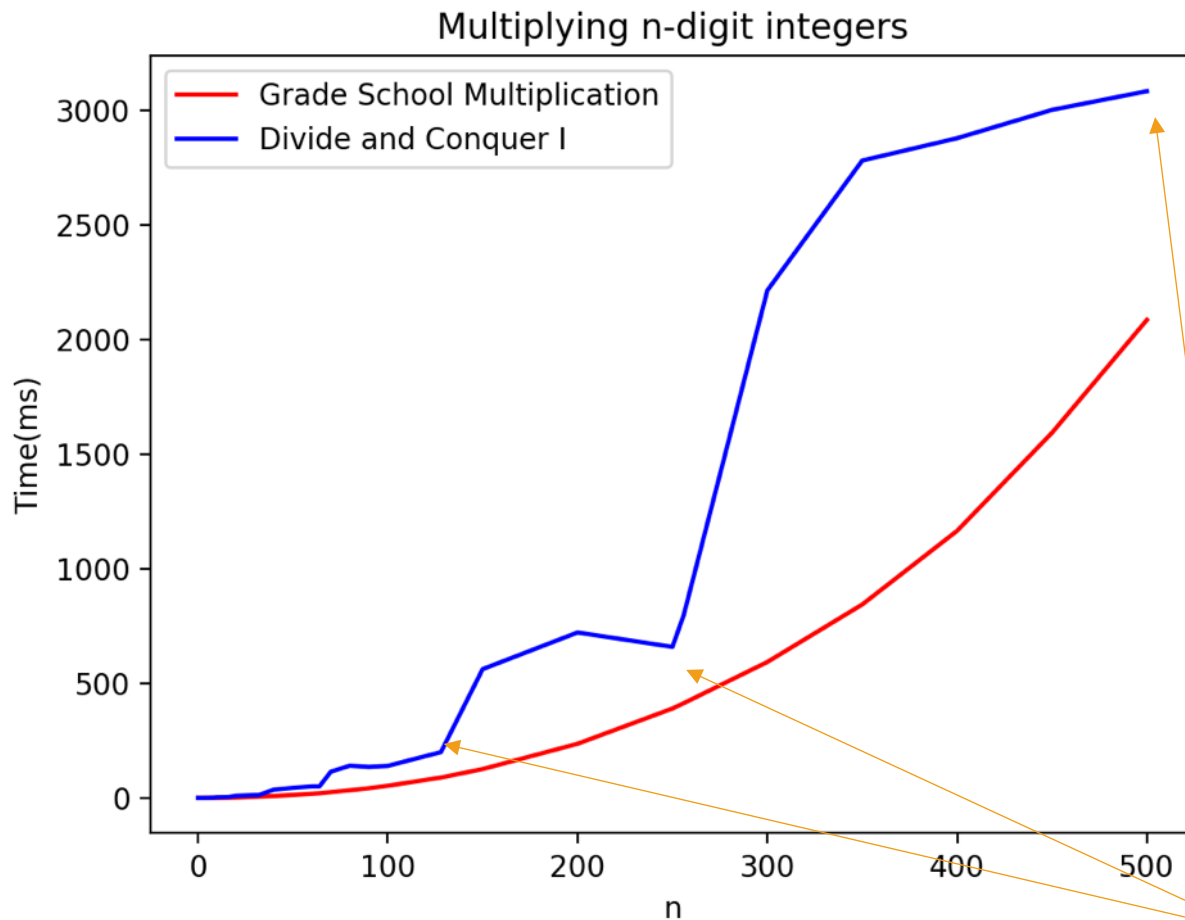
1. Try it.

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!

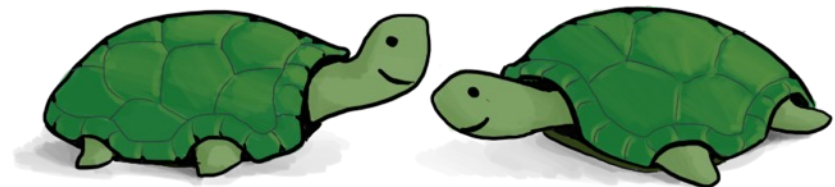


Plucky the Pedantic Penguin

2. Try to understand the running time analytically

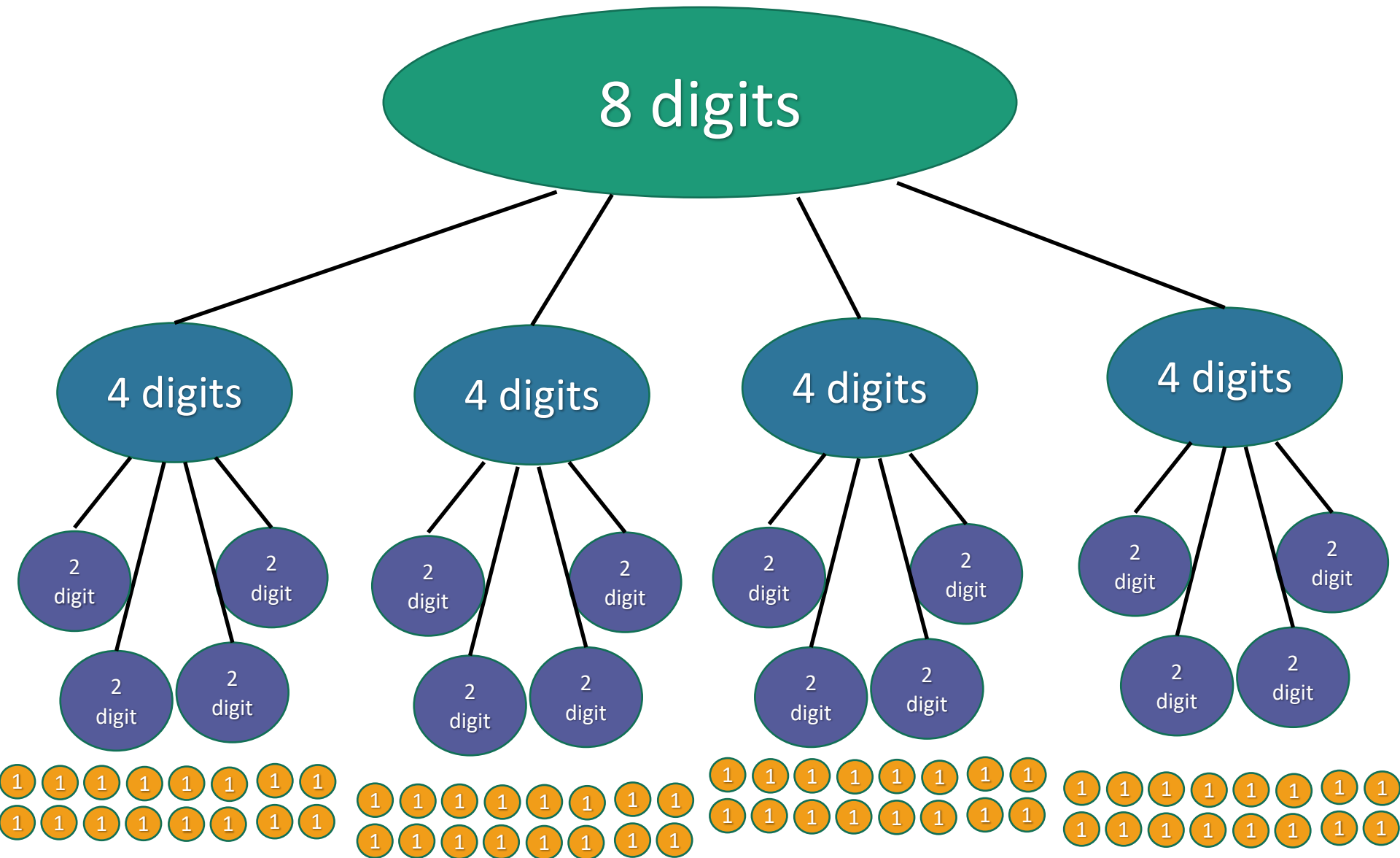
Think-Pair-Share:

- 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 one-digit multiplies!

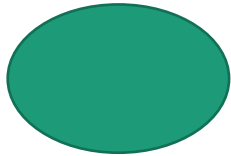


2. Try to understand the running time analytically

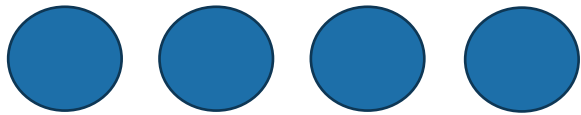
Claim:

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

There are n^2 1-digit problems

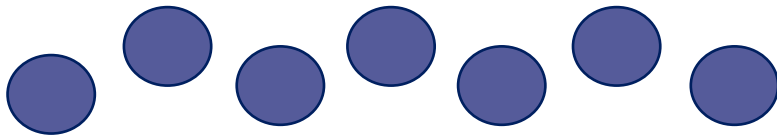


1 problem
of size n



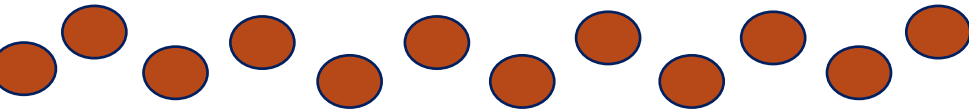
4 problems
of size $n/2$

...



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

...



n^2 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...

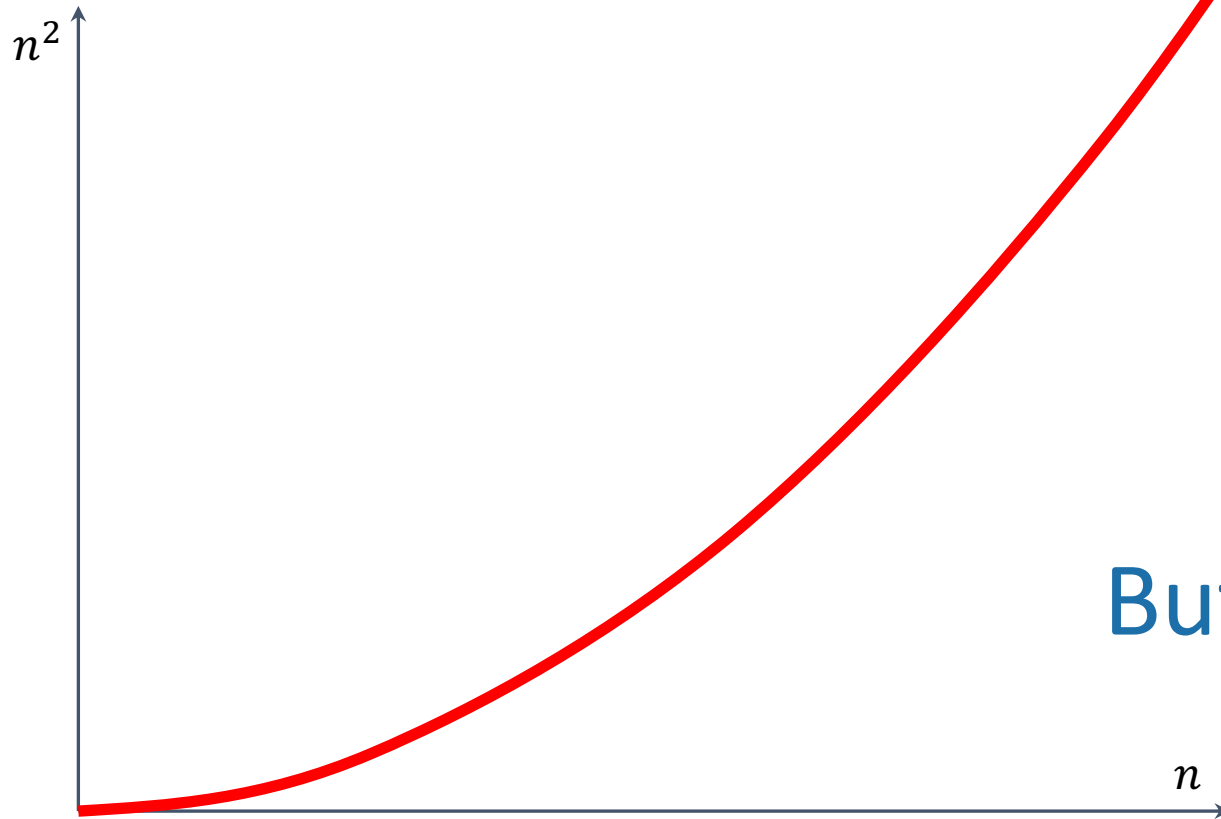
$$4^{\log_2 n} = n^{\log_2 4} = n^2$$

problems of size 1.

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

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:

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





How would this work?

x, y are n -digit numbers

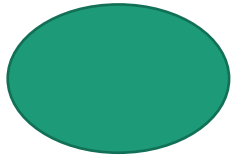
(Still not super precise, see IPython notebook for detailed code. Also, still assume n is a power of 2.)

Multiply(x, y):

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

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

What's the running time?

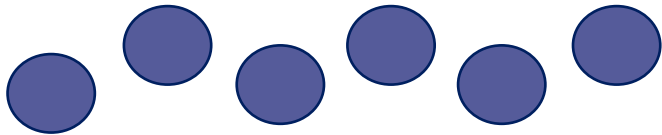


1 problem
of size n



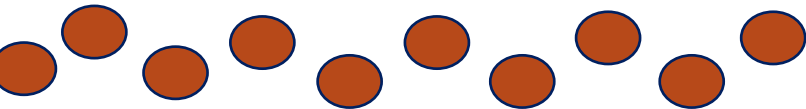
3 problems
of size $n/2$

...



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

...



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

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

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

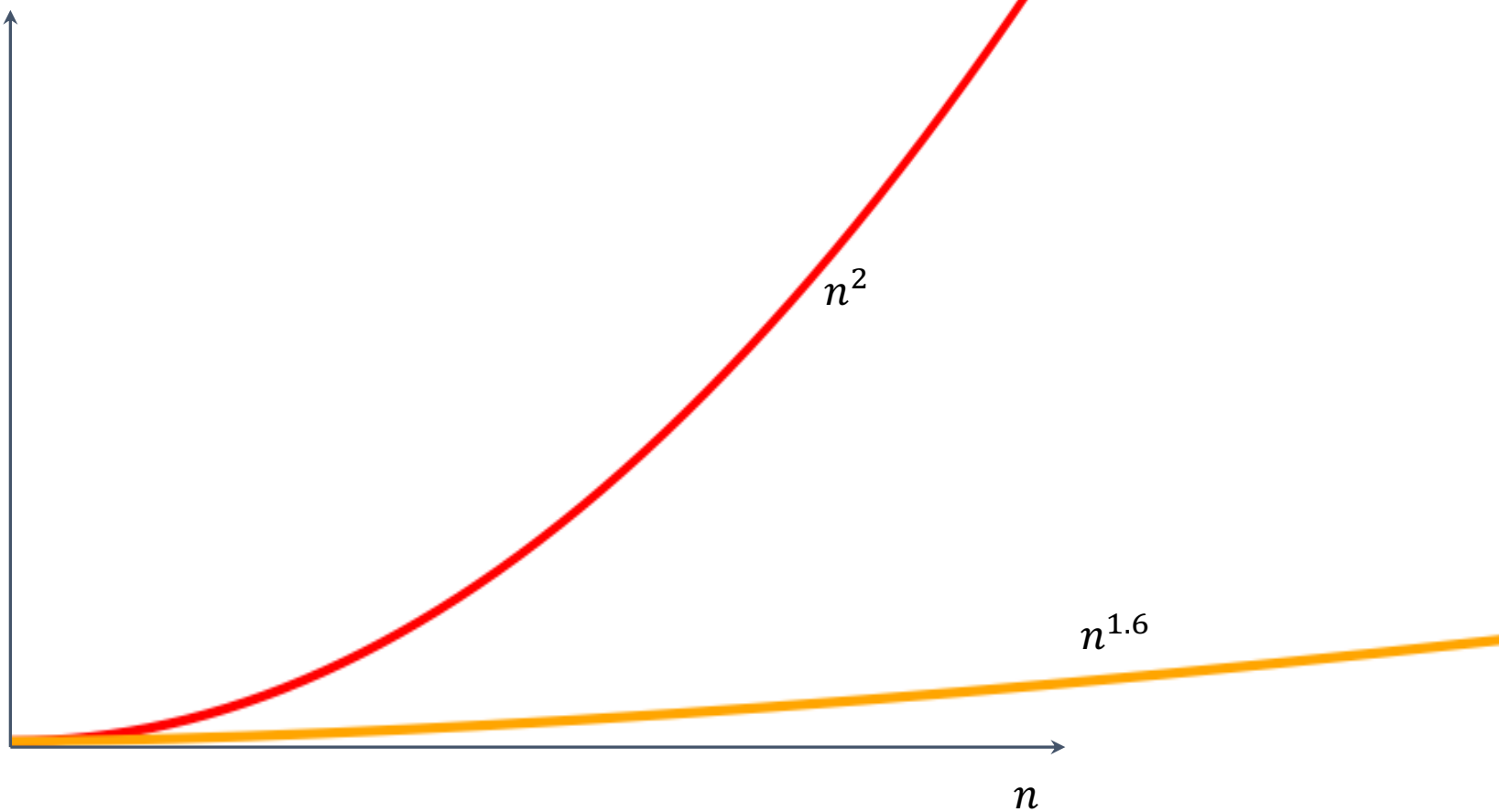
problems of size 1.

$n^{1.6}$
problems
of size 1

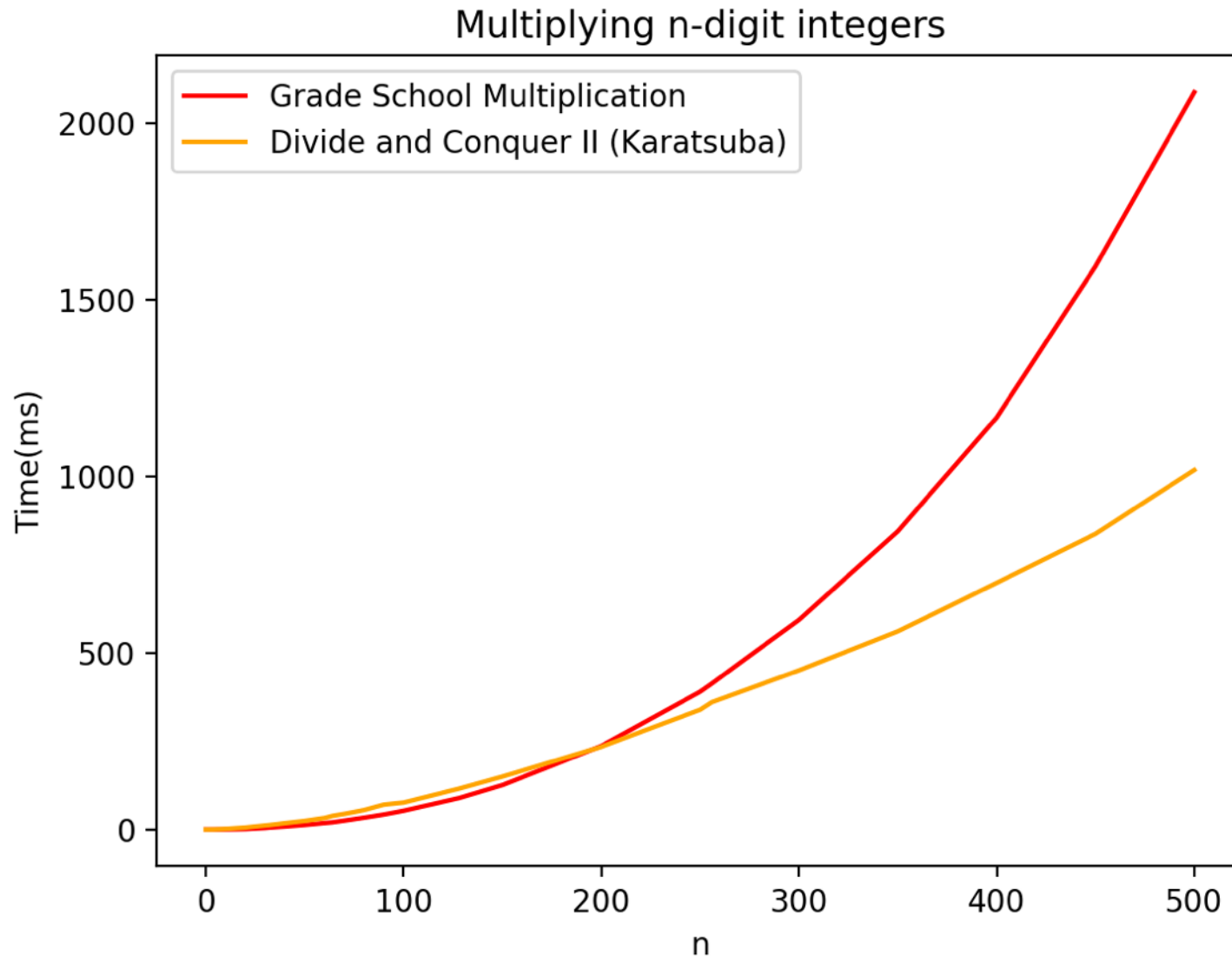
We aren't accounting for the work at the higher levels!
But we'll see later that this turns out to be okay.



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**).

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

Ollie the Over-achieving Ostrich

- **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

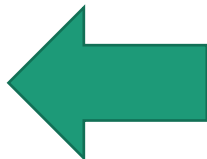
- Karatsuba Integer Multiplication
- Algorithmic Technique:
 - **Divide and conquer**
- Algorithmic Analysis tool:
 - **Intro to asymptotic analysis**



How was the pace
today?

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



BEFORE Next time

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