CS 161

Design and Analysis of Algorithms

Lecture 1:

Logistics, introduction, and multiplication!

How was your break?

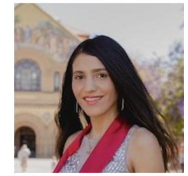
The big questions

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

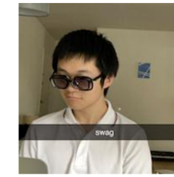
Who are we?


Amrita


Andre


Goli


Jerry


Jiazheng

- Instructors:
  - Moses Charikar
  - Nima Anari
- Course Coordinator:
  - Amelie Byun


Amelie


Jose


Manda


Nash


Peter


Sam


Samar

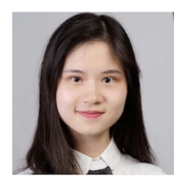
- Awesome CAs:
  - Ziang Liu (Head CA)
  - Peter Boennighausen
  - Andre Turati
  - Amrita Palaparthi
  - Seiji Eicher
  - Jiazheng Zhao
  - June Vuong
  - Yuchen Wang
  - Emily Wen
  - Samar Khanna
  - Avery Wang
  - Sam Lowe
  - Nash Luxsuwong
  - Shubham Jain
  - Andrew Yang
  - Jose Francisco
  - Tim Chirananthavat
  - Jerry Hong
  - Teresa Noyola
  - Goli Emami
  - Manda Tran


Seiji


Shubham


Teresa


Tim


Yuchen


Ziang

Who are you?

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

Concentrating in:

- Aero/Astro
- Archaeology
- Art Practice
- Bioengineering
- Biology
- Biomedical Informatics
- Biophysics
- Chemical Eng.
- Chemistry
- Chinese
- Civil & Env. Eng.
- Classics
- Communication
- Comparative Lit.
- Computer Science
- Creative Writing
- Earth Systems
- East Asian Studies
- Economics
- Education
- EE
- Energy Resources Eng.
- Engineering
- English
- Epidemiology
- Ethics in Society
- Geophysics
- History
- Human Biology
- Human Rights
- Immunology
- International Relat
- Material Sci & Eng
- Math & CS
- Math
- Mech. Eng.
- MS&E
- Music
- Philosophy
- Philosophy & Rel Stud
- Physics
- Political Science
- Psychology
- Science, Tech. and Society
- Slavic Lang & Lit
- Sociology
- Spanish
- Statistics
- Symbolic Systems
- Undeclared

Where are you?

Why are we here?

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

Yay Algorithms!

Why are you here?

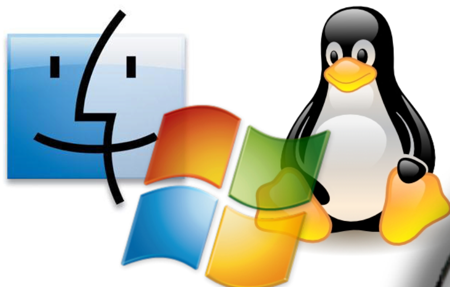
You are better equipped to answer this question than I am, but I'll give it a go anyway…

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

Why is CS161 required?

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

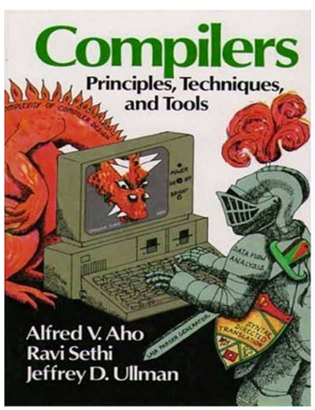
Algorithms are fundamental



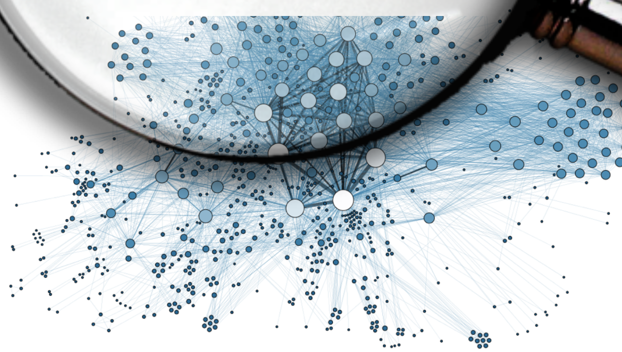
Operating Systems (CS 140)

229)

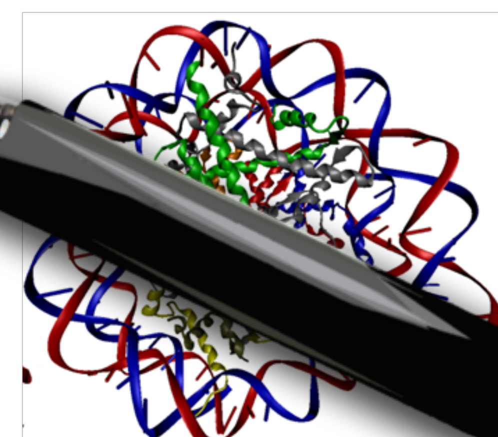
Cryptography (CS 255)



Compilers (CS 143)



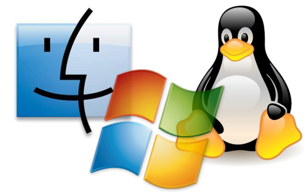
Networking (CS 144)

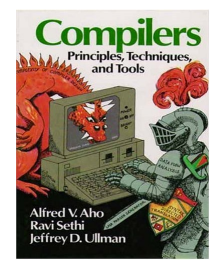


Computational Biology (CS 262)

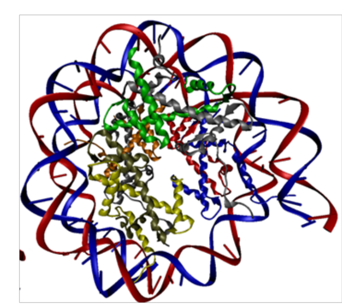
Algorithms are useful

- 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?

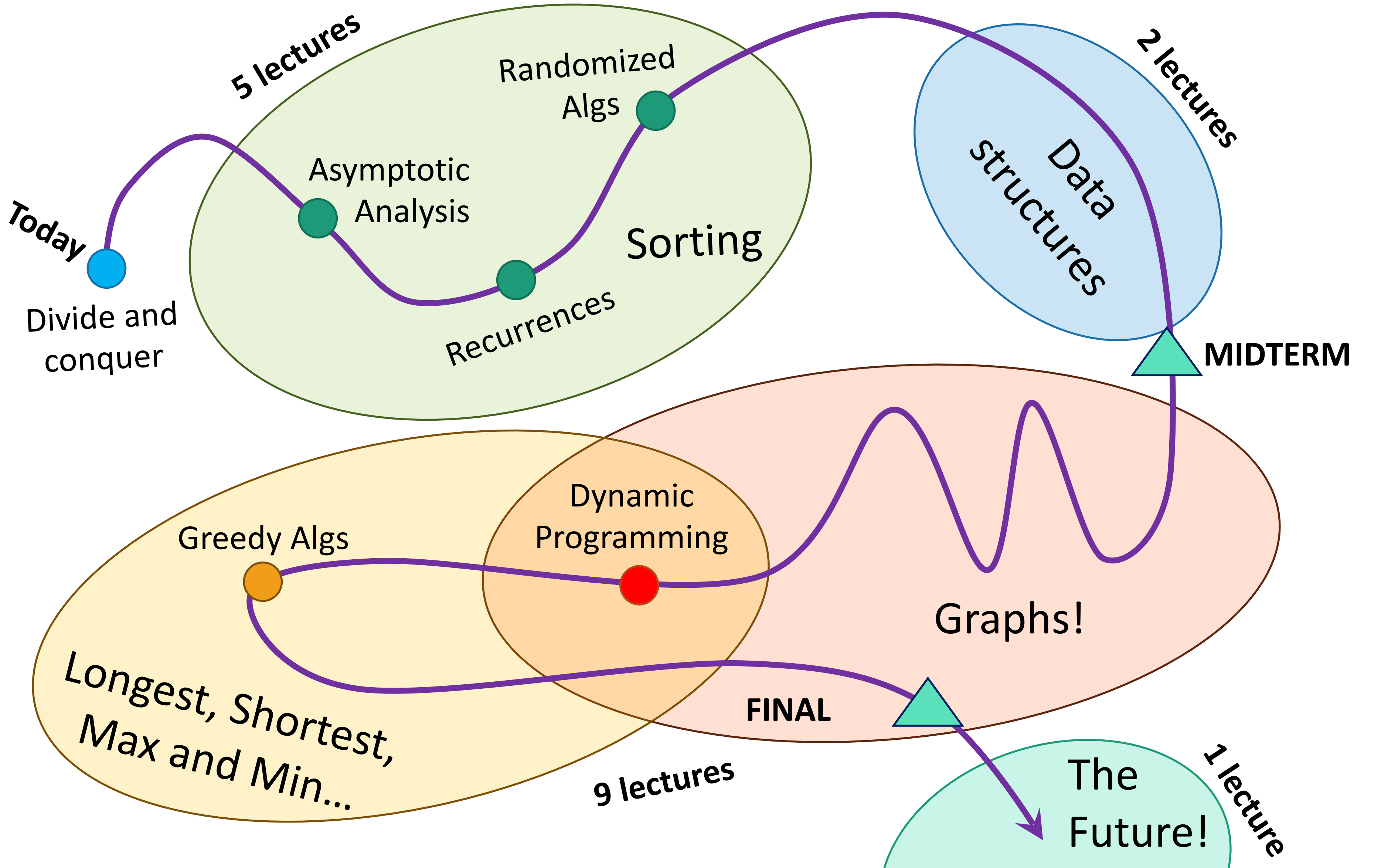
- Course goals/overview
- Logistics

Course goals

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

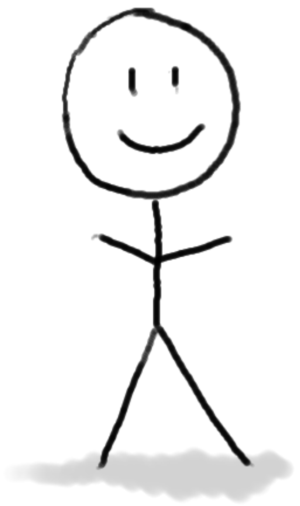
More detailed schedule on the website!

Roadmap



Our guiding questions:

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



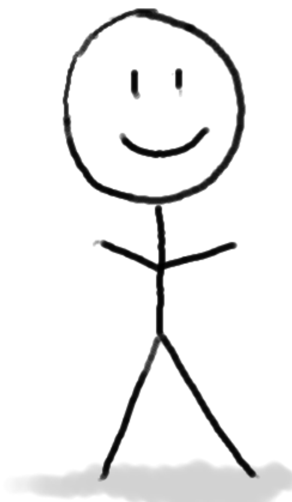
Our internal monologue…

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

Plucky the
Pedantic Penguin

Detail-oriented
Precise
Rigorous

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



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

Lucky the
Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

Both sides are necessary!

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.

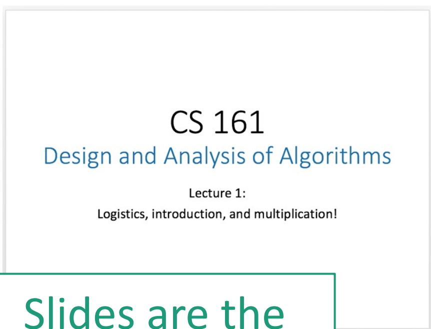
Course elements and resources

- Course website:
  - cs161.stanford.edu

- Lectures
- References
- IPython Notebooks
- Concept Check questions
- Homework
- Exams
- Office hours, recitation sections, and Ed

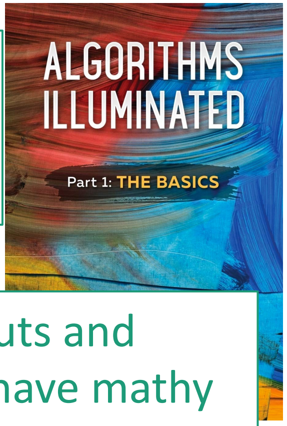
Lectures

- Mon/Wed, 9:45-11:15
  First 2 weeks: On Zoom (link on canvas)
  Later: In person (Nvidia auditorium)
- Resources available:
  - Slides, Videos, Notes, IPython notebooks, concept check qns

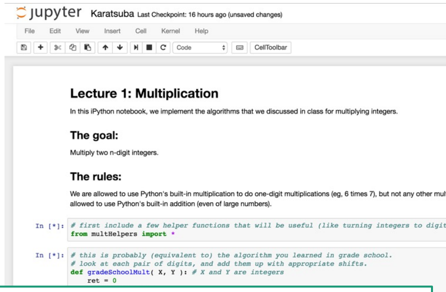


Slides are the slides from lecture.

Videos from lecture are available!



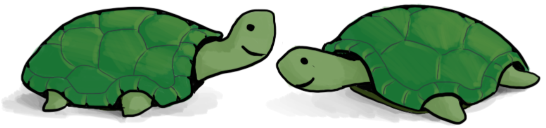
Hand-outs and references have mathy details that slides may omit



IPython notebooks have implementation details that slides may omit.

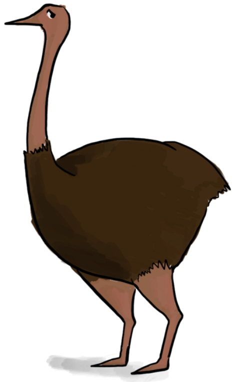
How to get the most out of lectures

- **During lecture:**
  - 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)

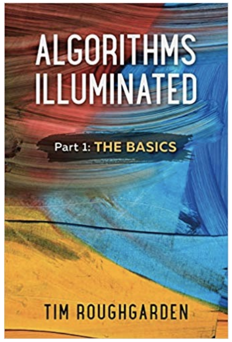
Siggi the Studious Stork
(recommended exercises)

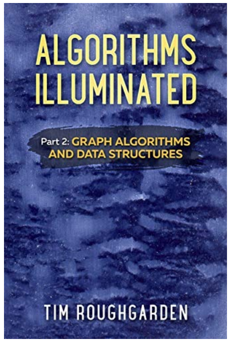


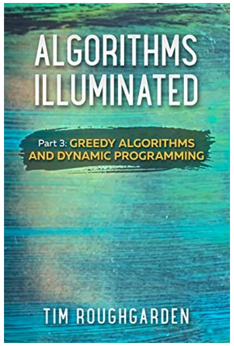
Ollie the Over-achieving Ostrich
(challenge questions)

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

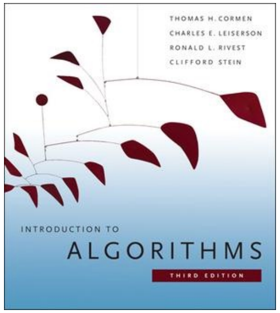
Optional References



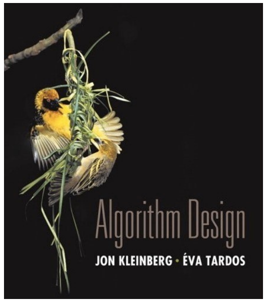




- **Algorithms Illuminated**, Vols 1,2 and 3 by Tim Roughgarden
- Additional resources at algorithmsilluminated.org
- We may also refer to to the following (optional) books:



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



“Algorithm Design” by Kleinberg and Tardos

IPython Notebooks

- Lectures will occasionally use IPython notebooks (but not homeworks)
  - 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.

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

Homework!

- Weekly assignments, posted Wednesday by 12:30pm, due the next Wednesday 11:59pm.
- First HW posted this Wednesday!

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

Exams

- There will be a **midterm** and a **final**
  - **Midterm:** Mon Feb 7 - Tue Feb 8 (48 hr window)
  - **Final:** Wed Mar 16, 3:30pm – 6:30pm

- 8 homeworks, lowest score dropped

- Weighting: **HW** (50%), **Midterm** (20%), **Final** (30%)

- If you have a conflict with the midterm time, email cs161-win2122-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 (on Nooks):
  - See course website for schedule
- Recitation sections (on Zoom):
  - Thursdays and Fridays.
  - See course website for schedule
  - Technically optional, but ***highly recommended***!
  - Extra practice with the material, example problems, etc.

Talk to each other!

- Answer your peers' questions on Ed!
- We will host Homework Parties (on Nooks).

Course elements and resources

- Course website:
  - cs161.stanford.edu
- Lectures
- References
- IPython Notebooks
- Homework
- Exams
- Office hours, recitation sections, 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)

Collaboration

- We encourage collaboration on homeworks (but strongly recommend you do exercises on your own)
- Valid and invalid modes of collaboration detailed on the course website.
  - Briefly, you can exchange ideas with classmates, but must write up solutions on your own.
- You must cite all collaborators, as well as all sources used (outside of course materials).

Honor code

- Updated last year: **“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**.”

  https://communitystandards.stanford.edu/bja-guidance-remote-teaching-and-learning-environment

- Course policy for Homeworks: “**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 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 bug gets a bonus point.

Bug Bounty Hunter

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

For SCPD Students (and all students)

- All/some office hours held online (on Nooks)
- One of the recitation sections will be recorded.
- See the website for more details! (Coming soon…)

Stanford | Center for Professional Development

OAE forms

- Please send OAE forms to
  cs161-win2122-staff@lists.stanford.edu

Feedback!

- We will have an anonymous feedback form on the course website (top of the main page).
- Please help us improve the course!

How are you approaching CS 161?

Everyone can succeed in this class!

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

4. CS 161A
   one unit supplementary
   class (deadline to apply:
   Fri Jan 7, 5pm PST)

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

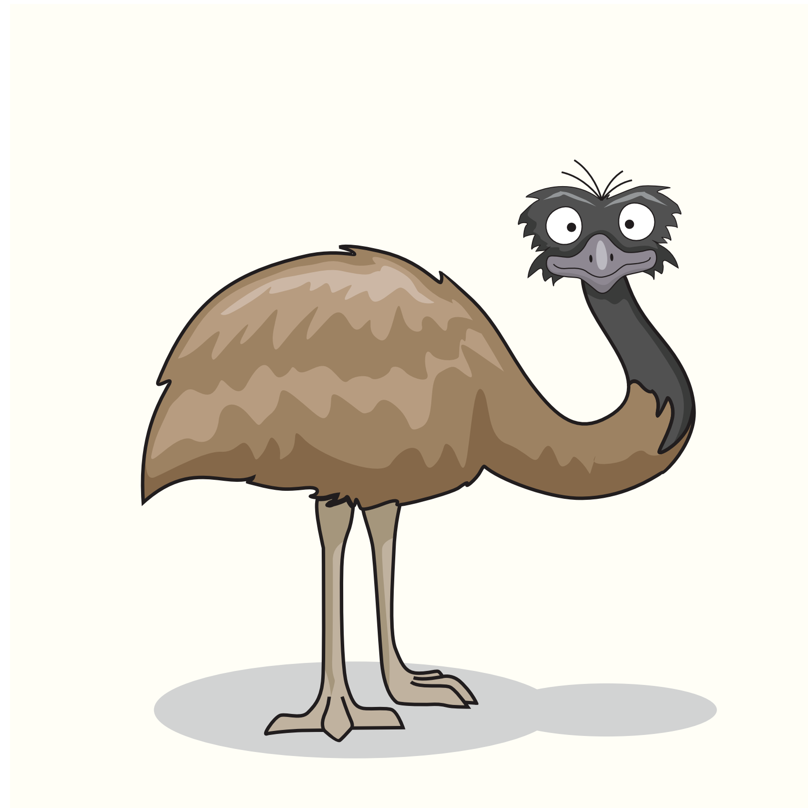
I taught ethics at the Harvard Kennedy School of Government

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

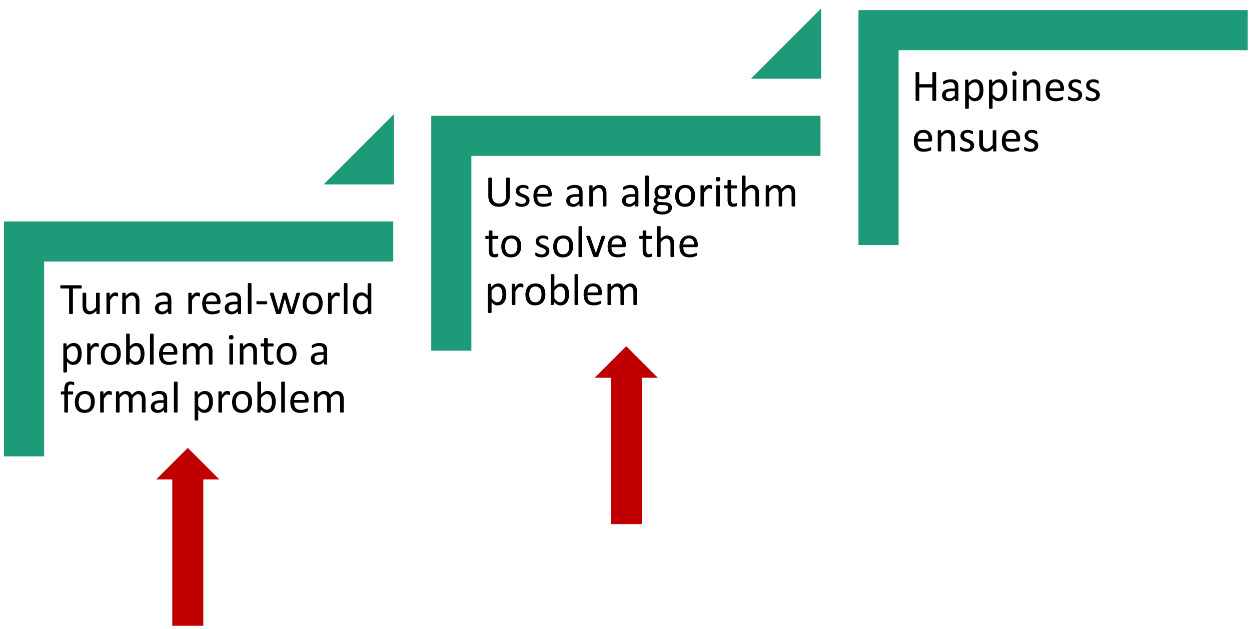
Algorithms & the Good

Turn a real-world problem into a formal problem

Use an algorithm to solve the problem

Happiness ensues

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.

Turn a real-world problem into a formal problem

Use an algorithm to solve the problem

Happiness ensues

Real-world problem? I don't see you all the way over there …

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

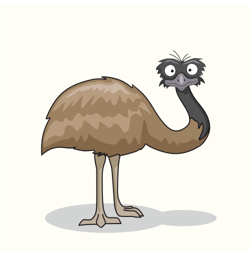
Use an algorithm to solve the problem

Happiness ensues

Turn a real-world problem into a formal problem

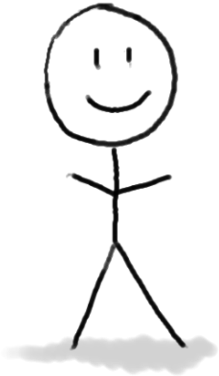
Use an algorithm to solve the problem

Happiness –or something– ensues



Did we achieve what we set out achieve in the first place? What did we lose along the way? Is this a desirable outcome?

Our guiding questions:



Does it work?

Is it fast?

Can I do better?

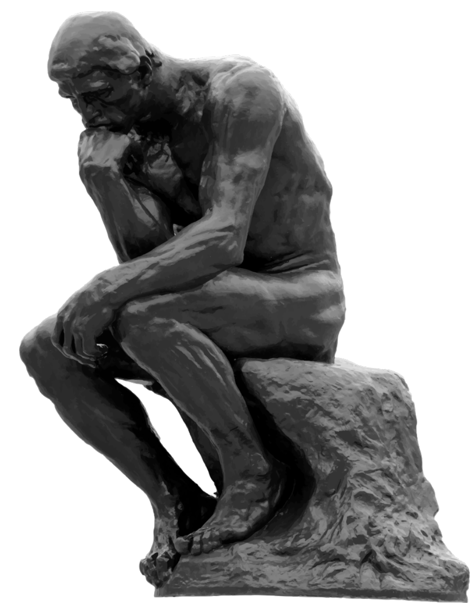
Can I do it right?

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

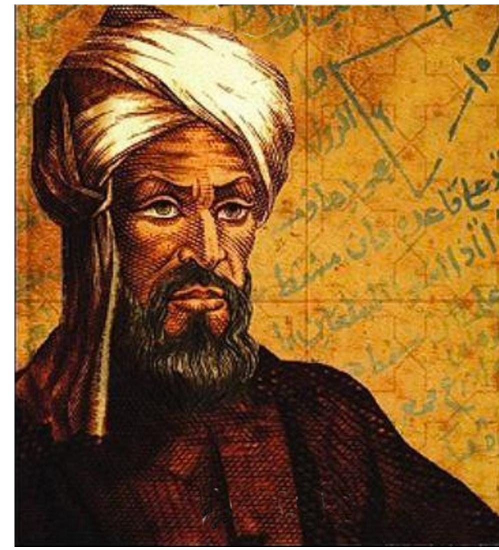
Today's goals

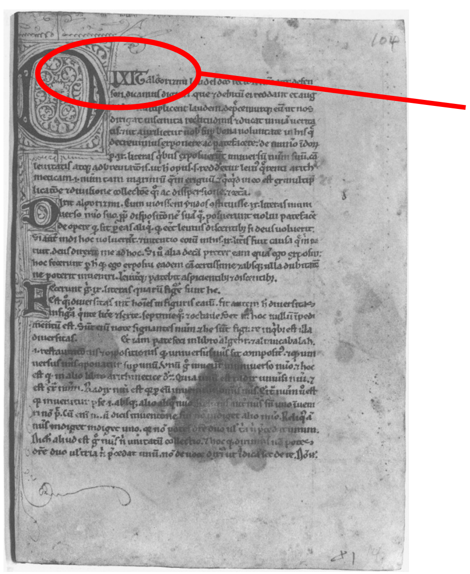
- Karatsuba Integer Multiplication
- Algorithmic Technique:
  - Divide and conquer
- Algorithmic Analysis tool:
  - Intro to asymptotic analysis

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 Abbassid Caliphate.
- Among many other contributions in mathematics, astronomy, and geography, he wrote a book about how to multiply with Arabic numerals.
- His ideas came to Europe in the 12th century.





Dixit algorizmi
(so says Al-Khwarizmi)

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

This was kind of a big deal

XLIV × XCVII = ?

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

Integer Multiplication

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

Integer Multiplication

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

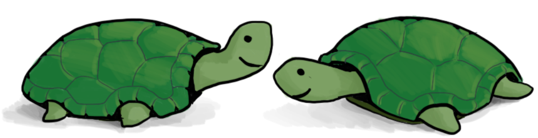
Integer Multiplication

$$\overbrace{\begin{array}{r} 1233925720752752384623764283568364918374523856298 \\ \times \quad 4562323582342395285623467235019130750135350013753 \end{array}}^{n}$$

???

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

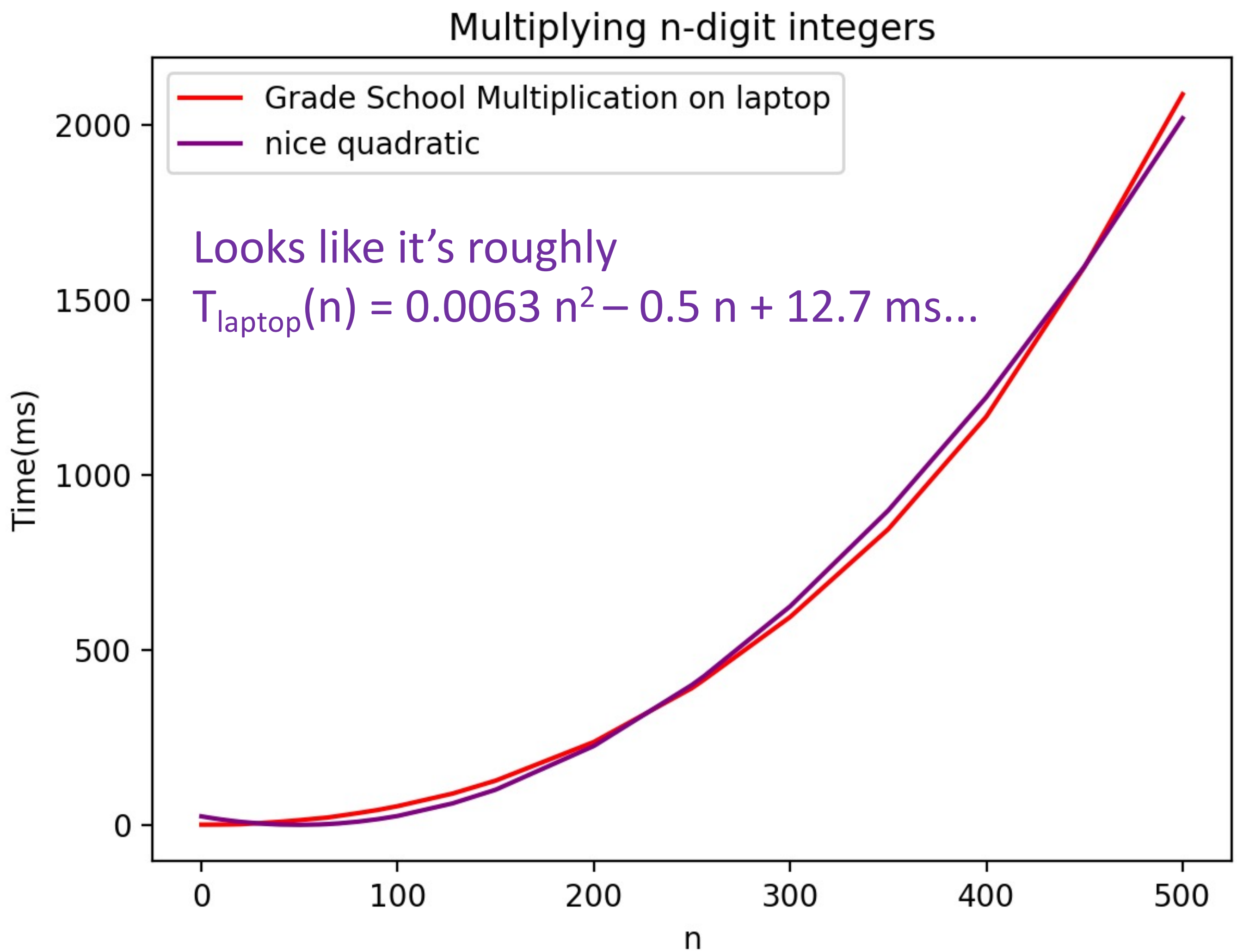
  "runs in time $O(n^2)$"

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

highly non-optimized

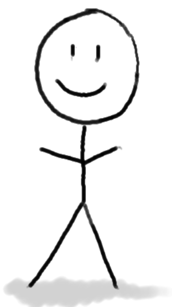
Implemented in Python, on my laptop

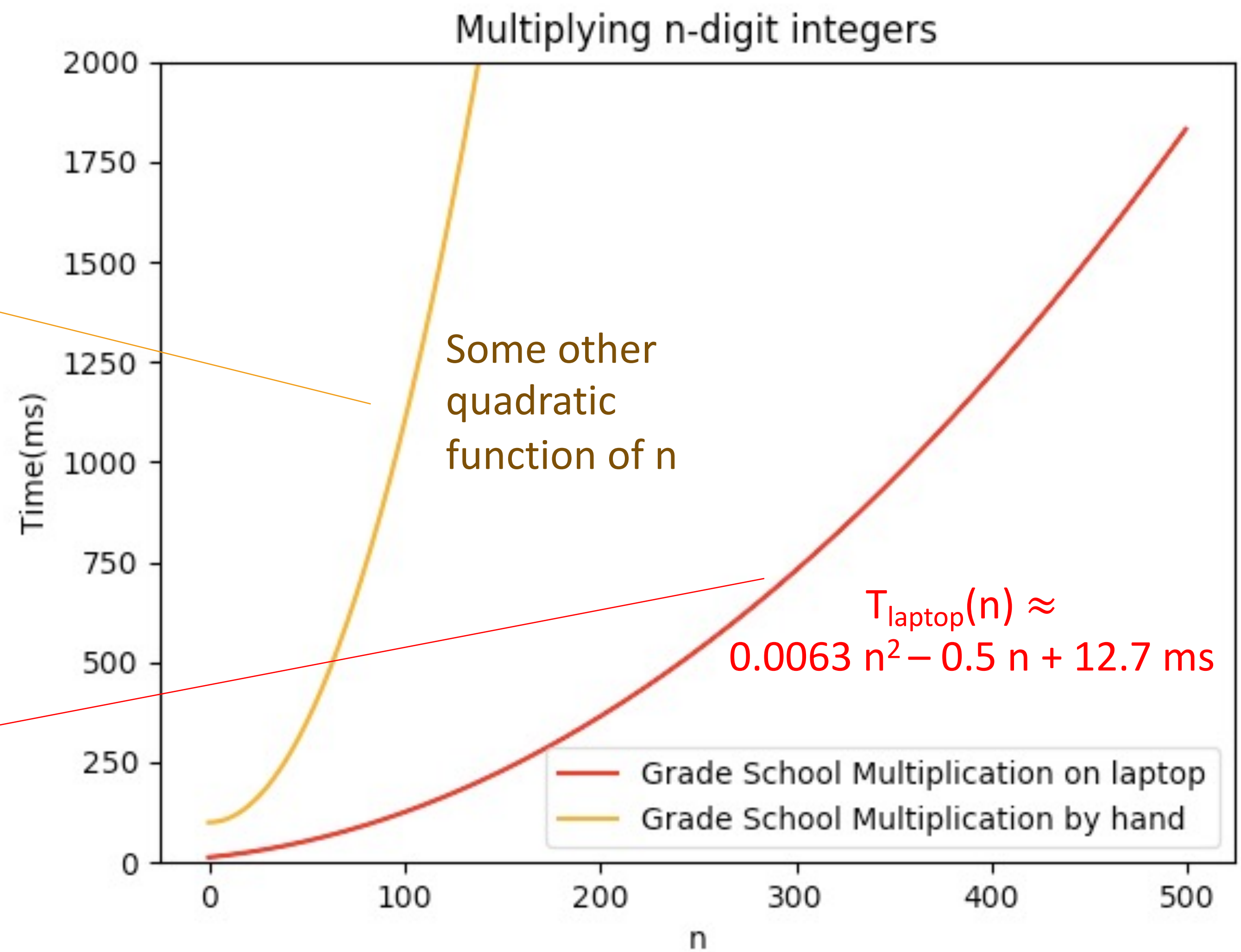
The runtime "scales like" n^2



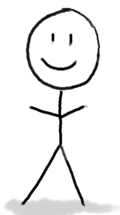
Implemented by hand

The runtime still "scales like" n^2



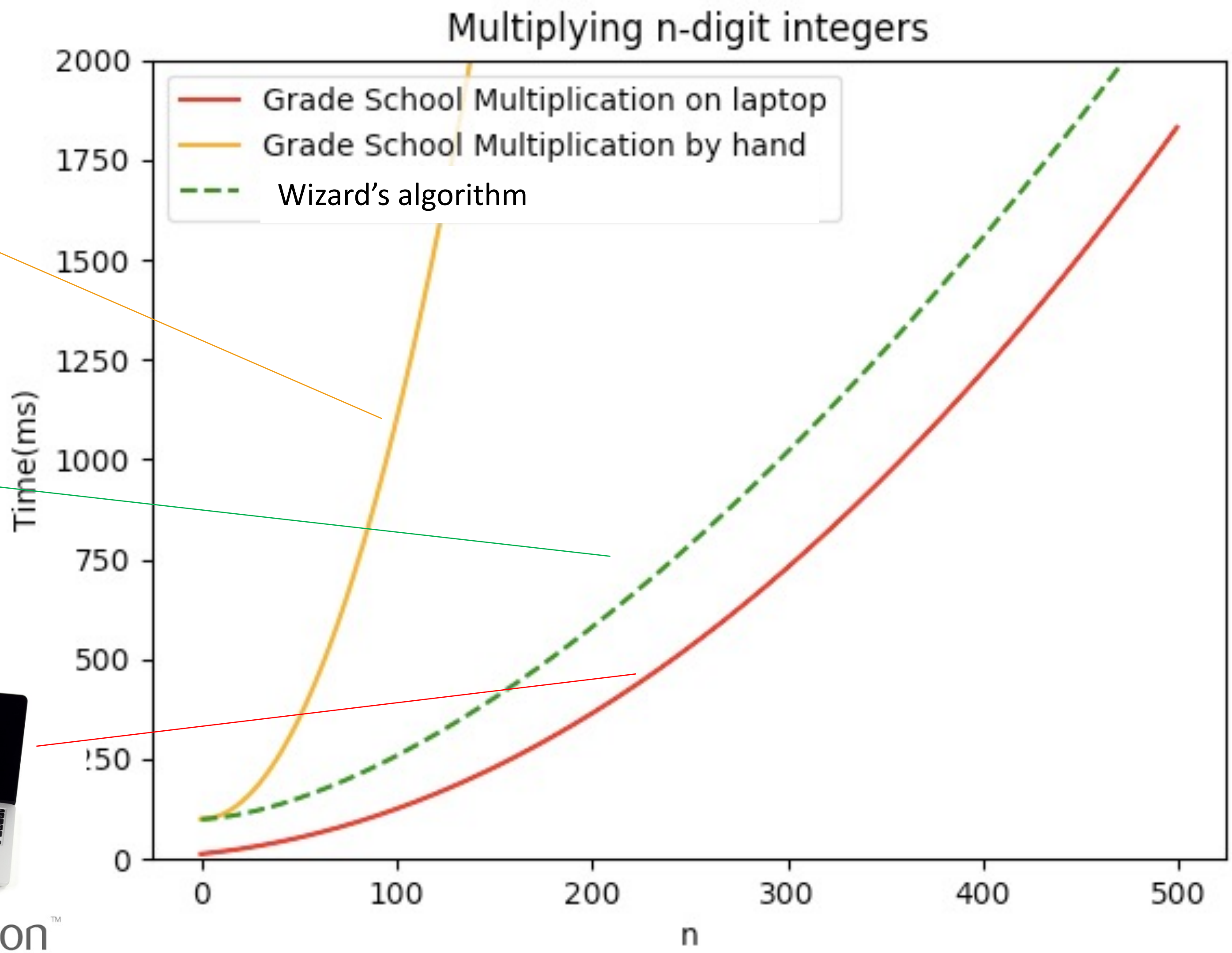


Why is big-Oh notation meaningful?

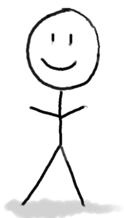


$$\approx \frac{n^{1.6}}{10} + 100$$

$$\approx .0063n^2$$

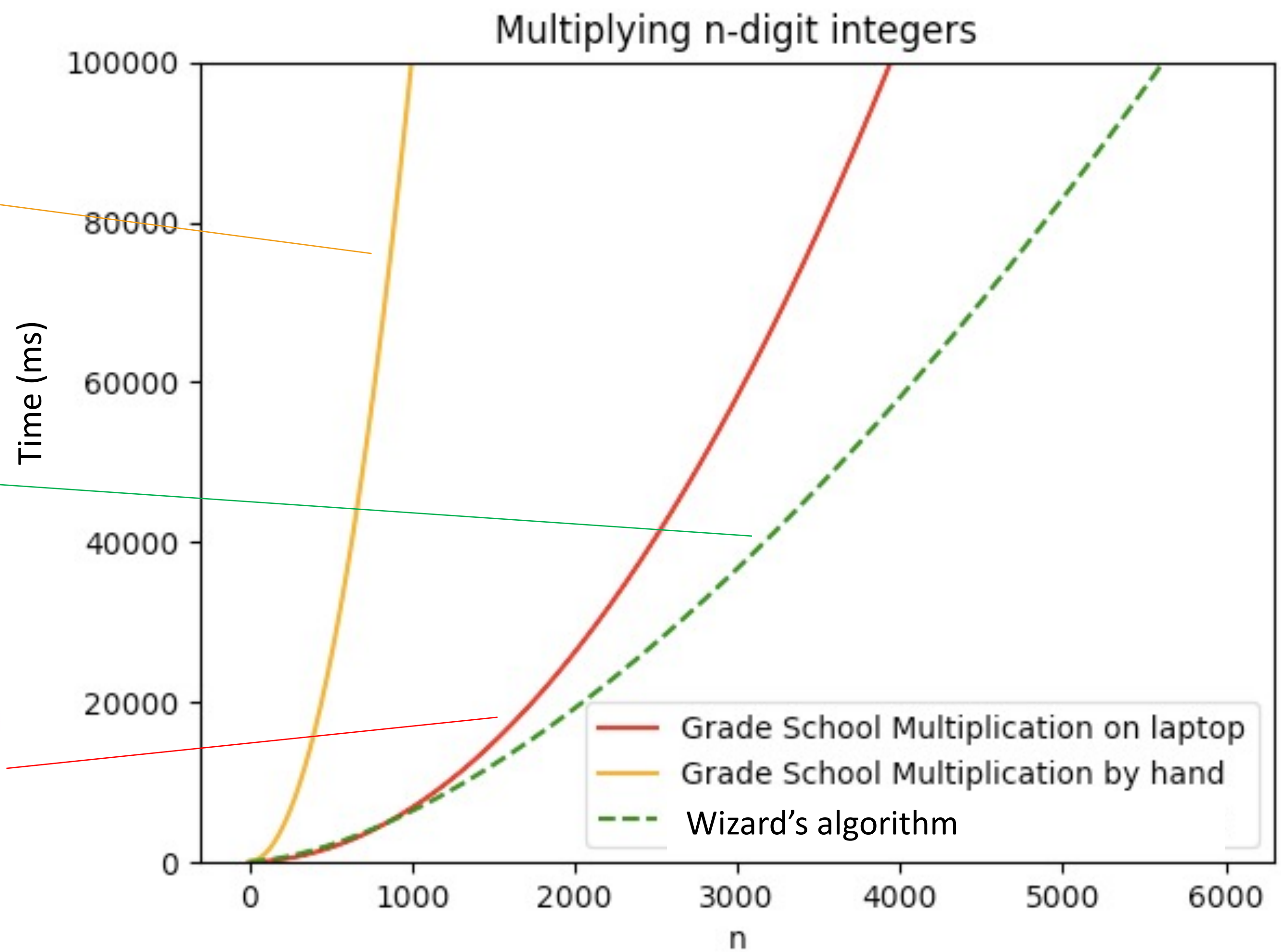


Let n get bigger…



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

$$\approx .0063n^2$$



Take-away

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

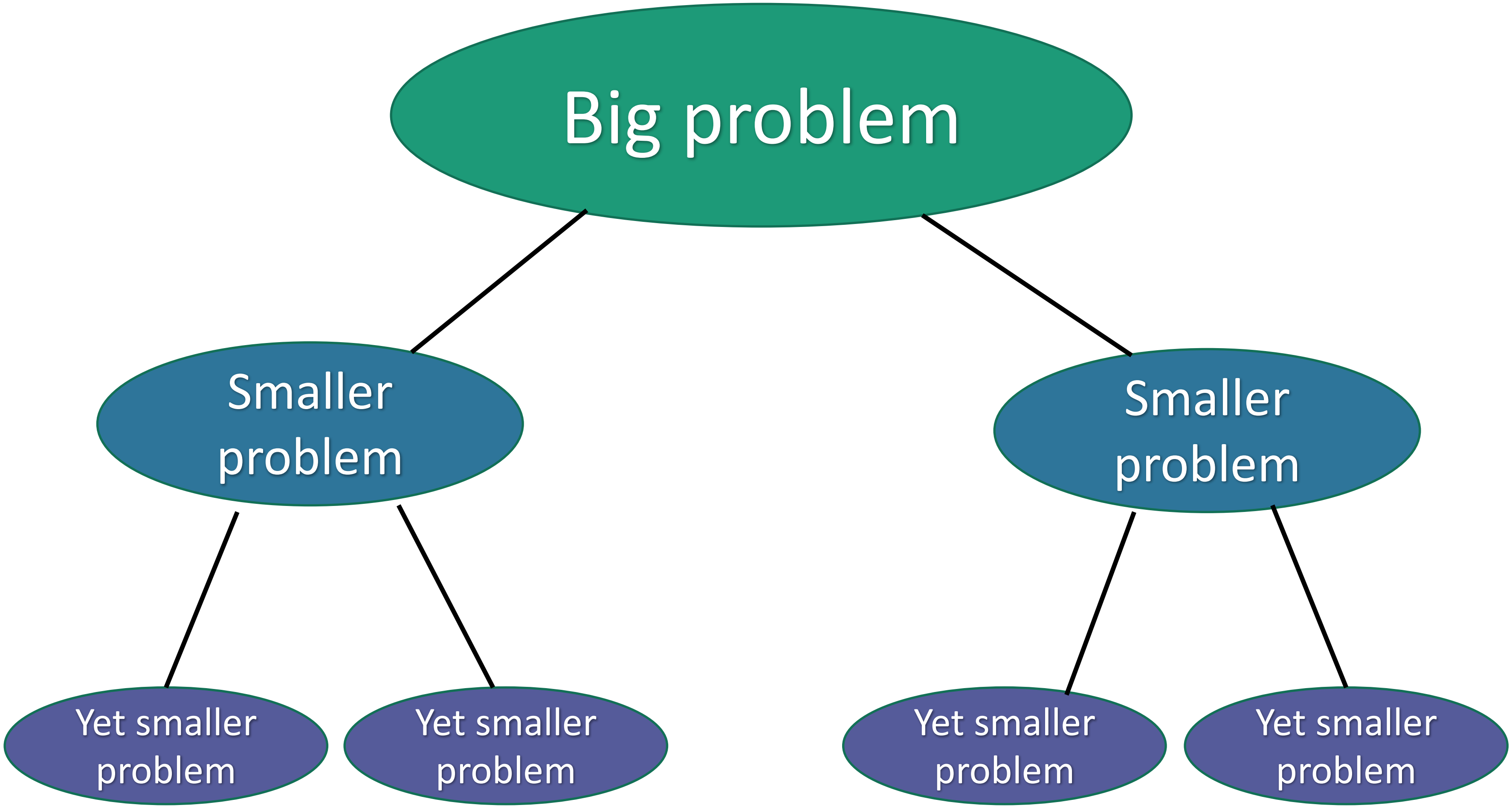
Can we do better?

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

Let’s dig in to our algorithmic toolkit…

Divide and conquer

Break problem up into smaller (easier) sub-problems



Divide and conquer for multiplication

Break up an integer:

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

$$1234 \times 5678$$

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

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

One 4-digit multiply → Four 2-digit multiplies

More generally

Suppose n is even

Break up an n-digit integer:

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

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

One n-digit multiply

Four (n/2)-digit multiplies

Divide and conquer algorithm

not very precisely…

(Assume n is a power of 2…)

x,y are n-digit numbers

Multiply(x, y):

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

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

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

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

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

See the Lecture 1 Python notebook for actual code!

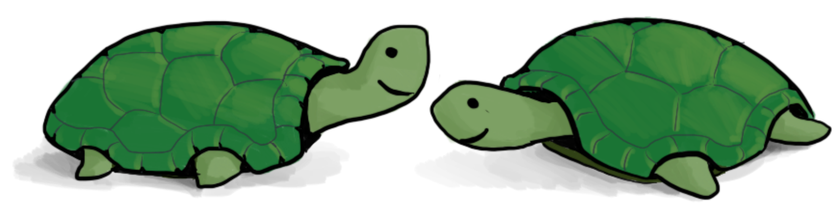
Siggi the Studious Stork

Think-Pair-Share

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

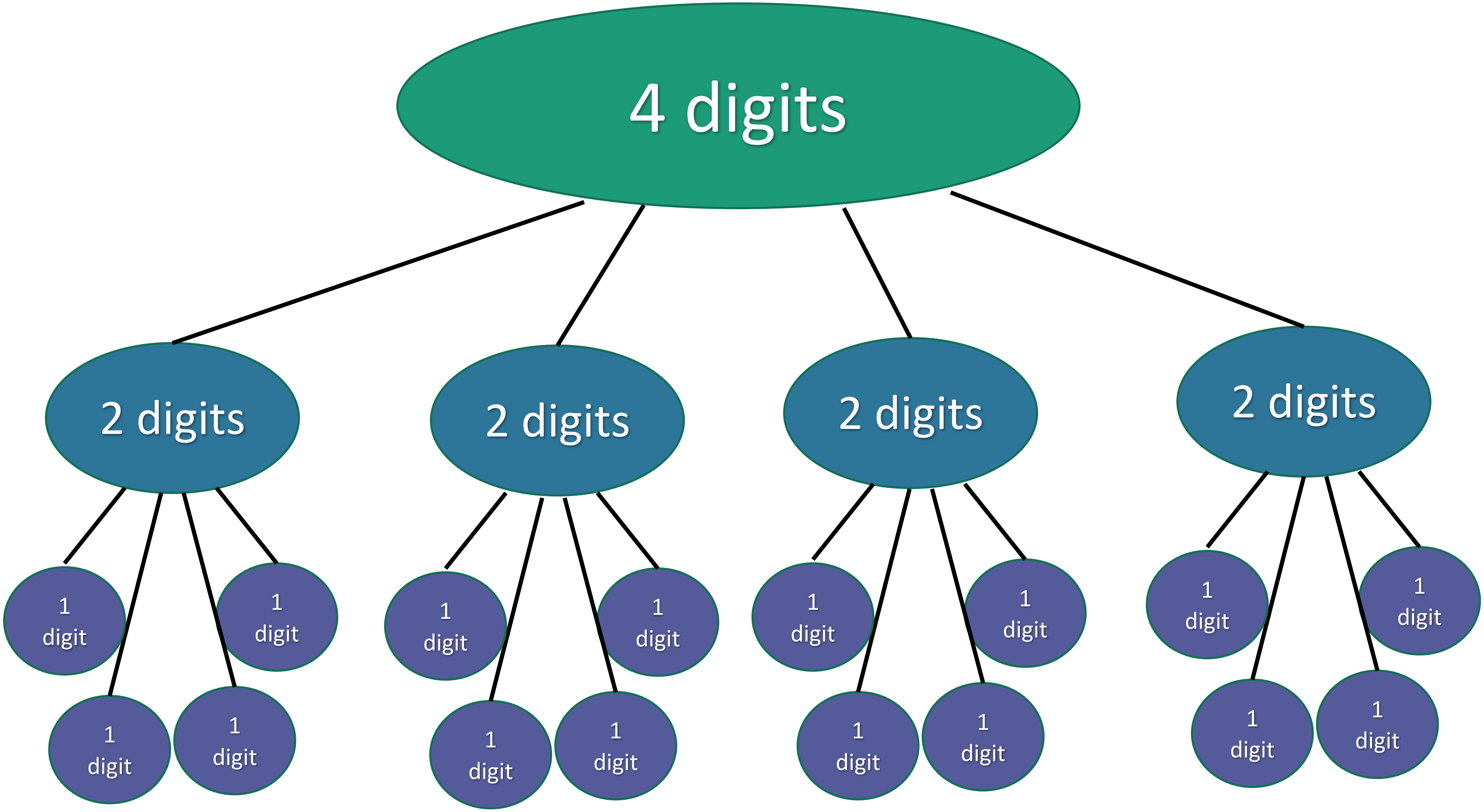
$$1234 \times 5678$$

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



Recursion Tree

16 one-digit multiplies!



What is the running time?

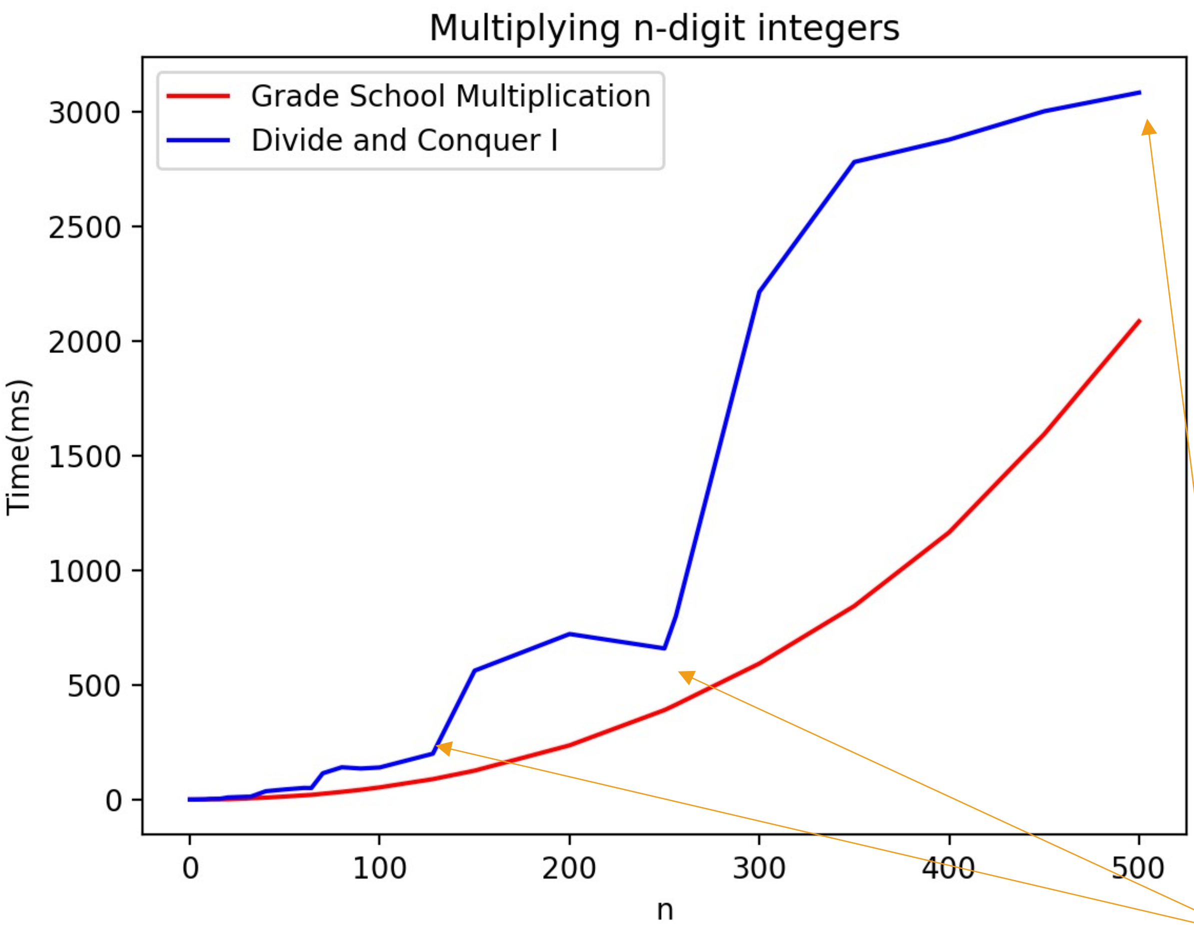
- Better or worse than the grade school algorithm?

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

Check out the Lecture 1 IPython Notebook

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.~~

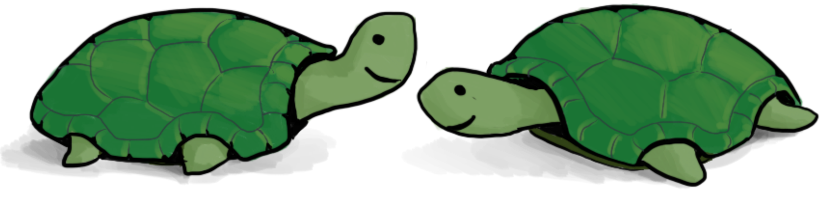
Not sound logic!

Plucky the Pedantic Penguin

2. Try to understand the running time analytically

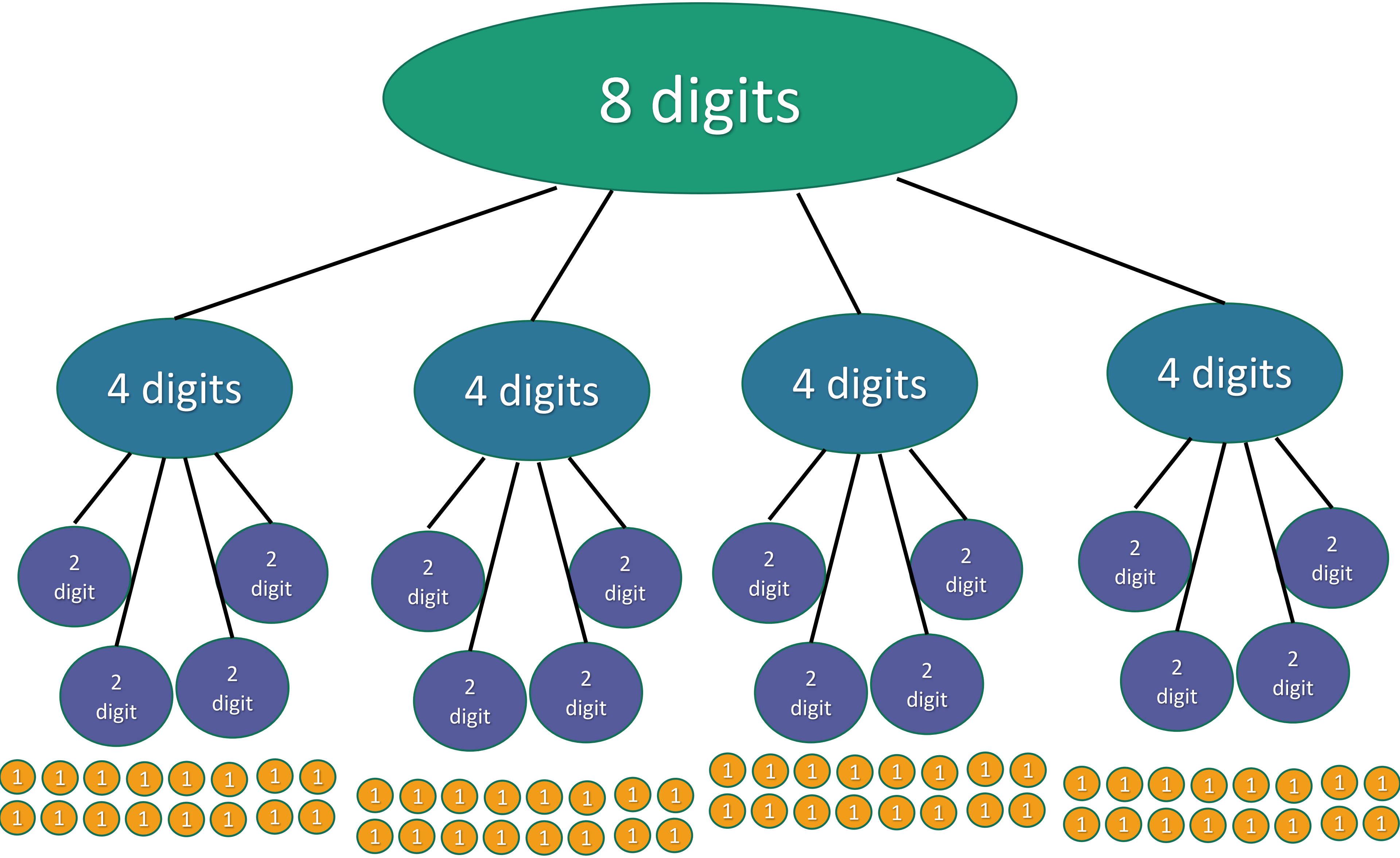
Think-Pair-Share:

- 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!

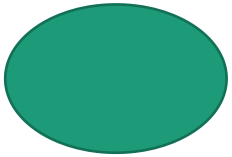


2. Try to understand the running time analytically

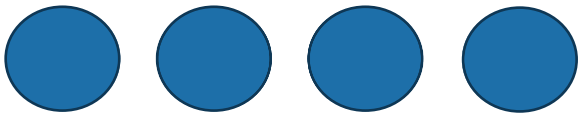
Claim:

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

There are n^2 1-digit problems

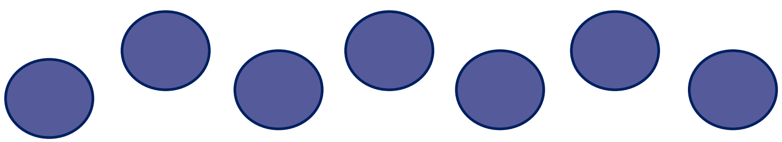


1 problem
of size n



4 problems
of size n/2

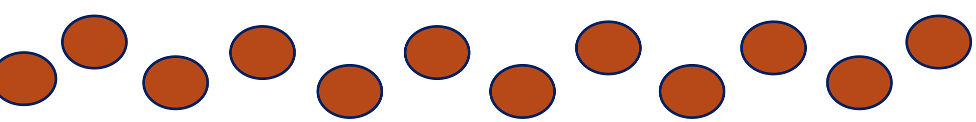
...



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

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

...



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

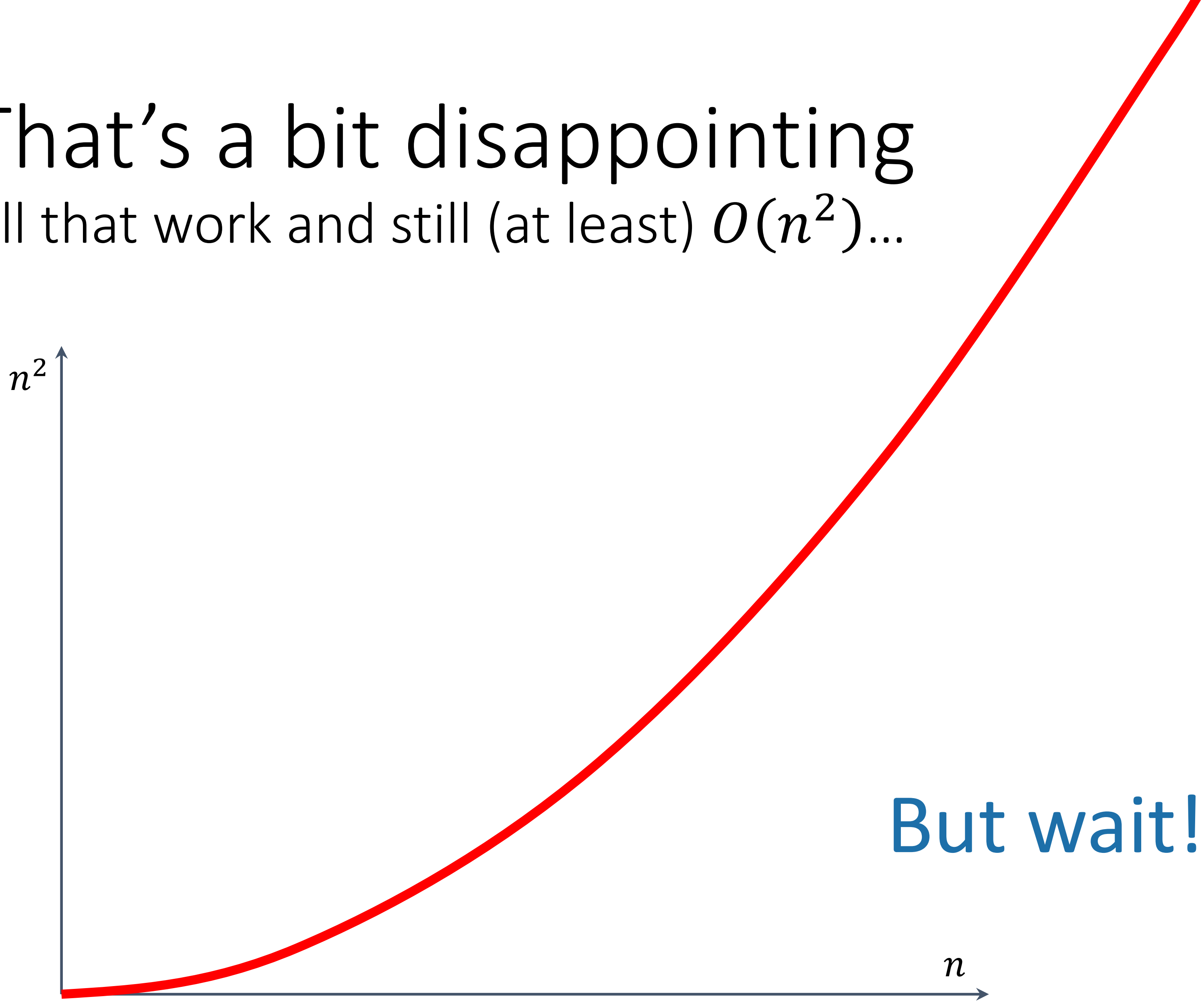
- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So at level $t = \log_2(n)$ we get...

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

problems of size 1.

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

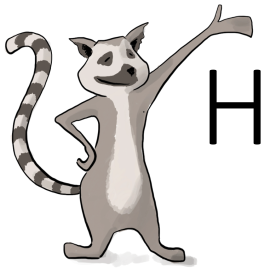
get this

(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?

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

x,y are n-digit numbers

Multiply(x, y):

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

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

What's the running time?

1 problem
of size n

3 problems
of size n/2

...

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

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

...

$\underline{n^{1.6}}$ 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...

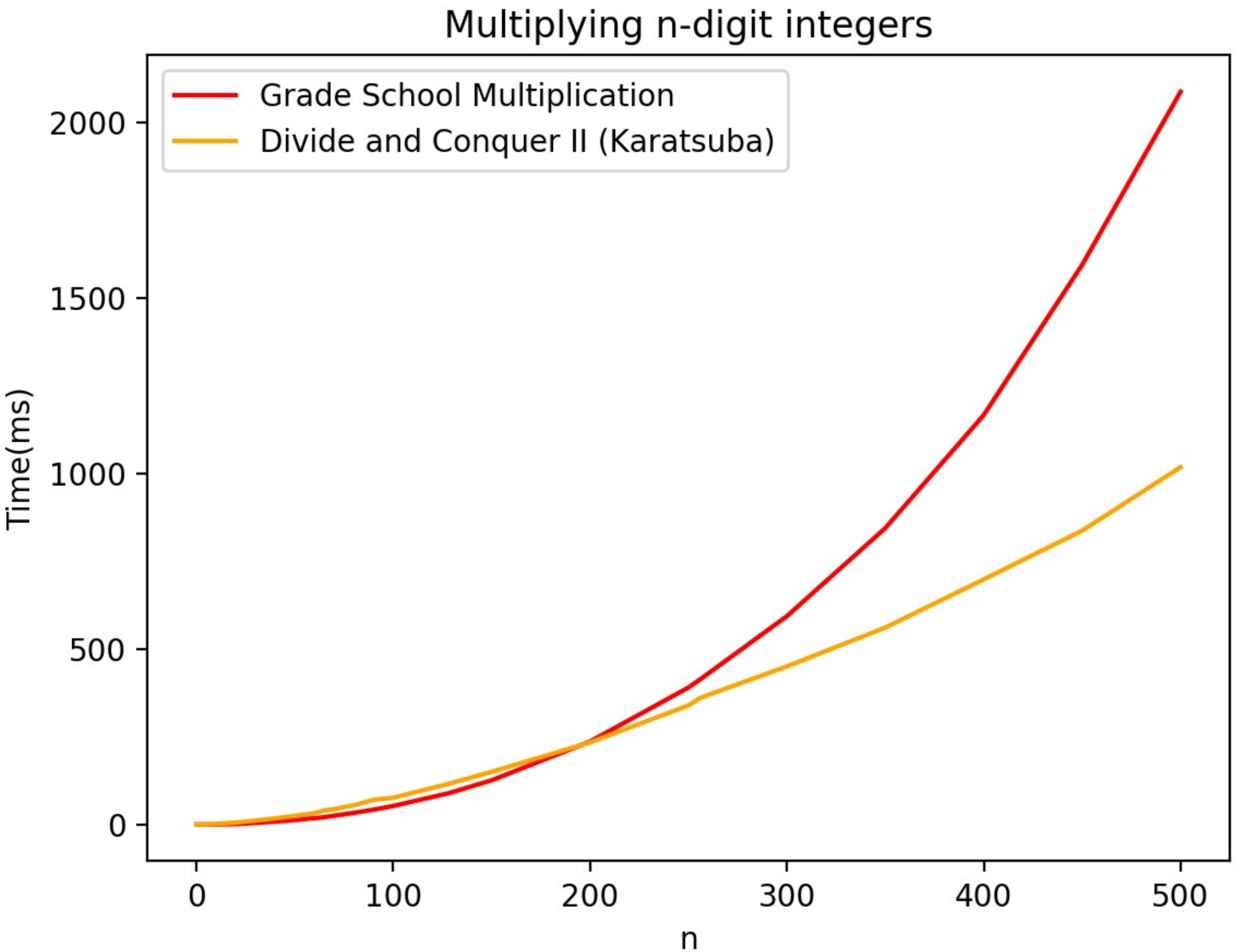
$$3^{\log_2 n} = n^{\log_2 3} \approx n^{1.6}$$

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.

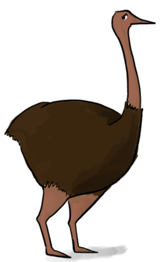
This is much better!

We can even see it in real life!



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

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!