Analysis of Algorithms

Lecture 1:

Logistics, introduction, and multiplication!

Today

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

Why are you here?

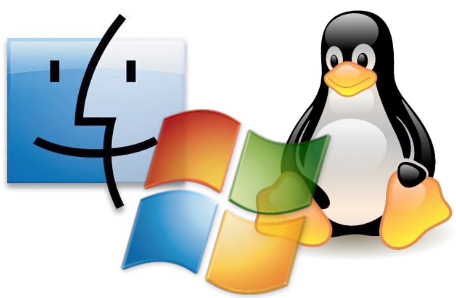
You are better equipped to answer this question than I am, but I'll give it a go anyway…

- Algorithms are fundamental.
- Algorithms are useful.
- Algorithms are fun!
- CE 323 is a required course.

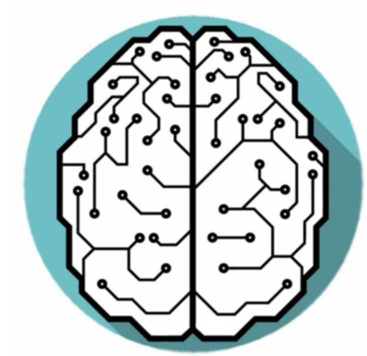
Why is CE 323 required?

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

Algorithms are fundamental



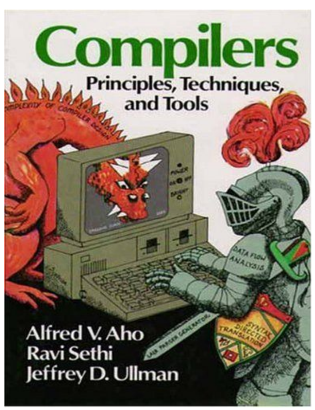
Operating Systems



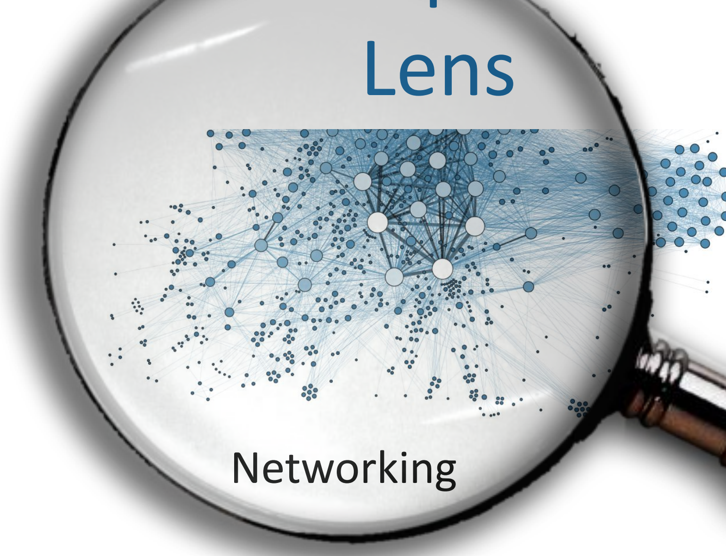
Machine learning

Cryptography

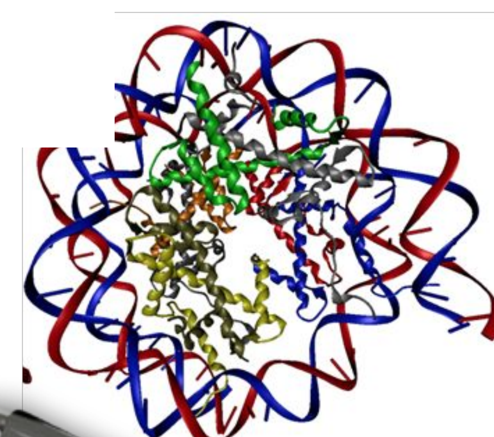
The Computational Lens



Compilers



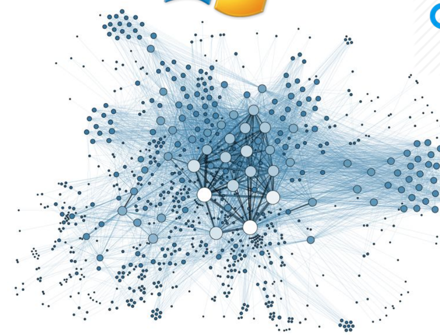
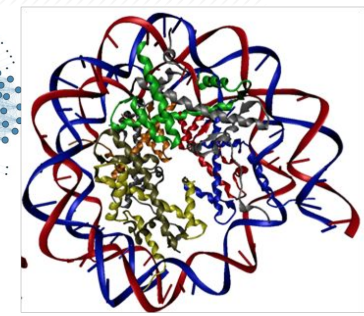
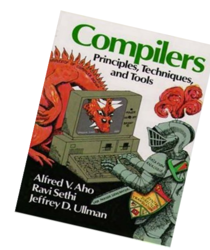
Networking

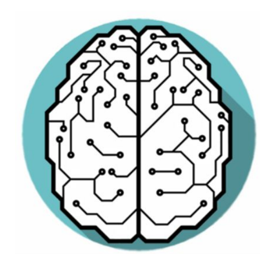


C[illegible]nal Biology

Algorithms are useful

- All those things, without CE 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.

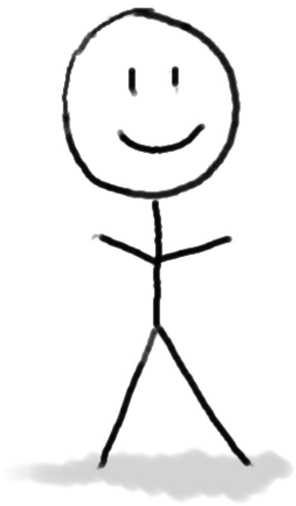
Course goals

- 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…

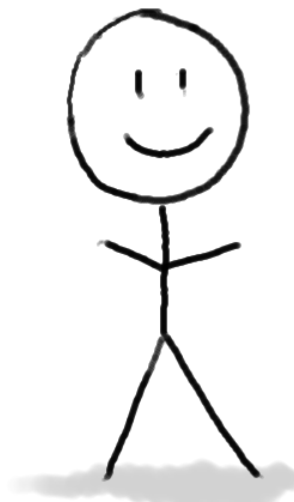
What exactly do we mean by better? And what about that corner case? Shouldn’t we be zero-indexing?

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

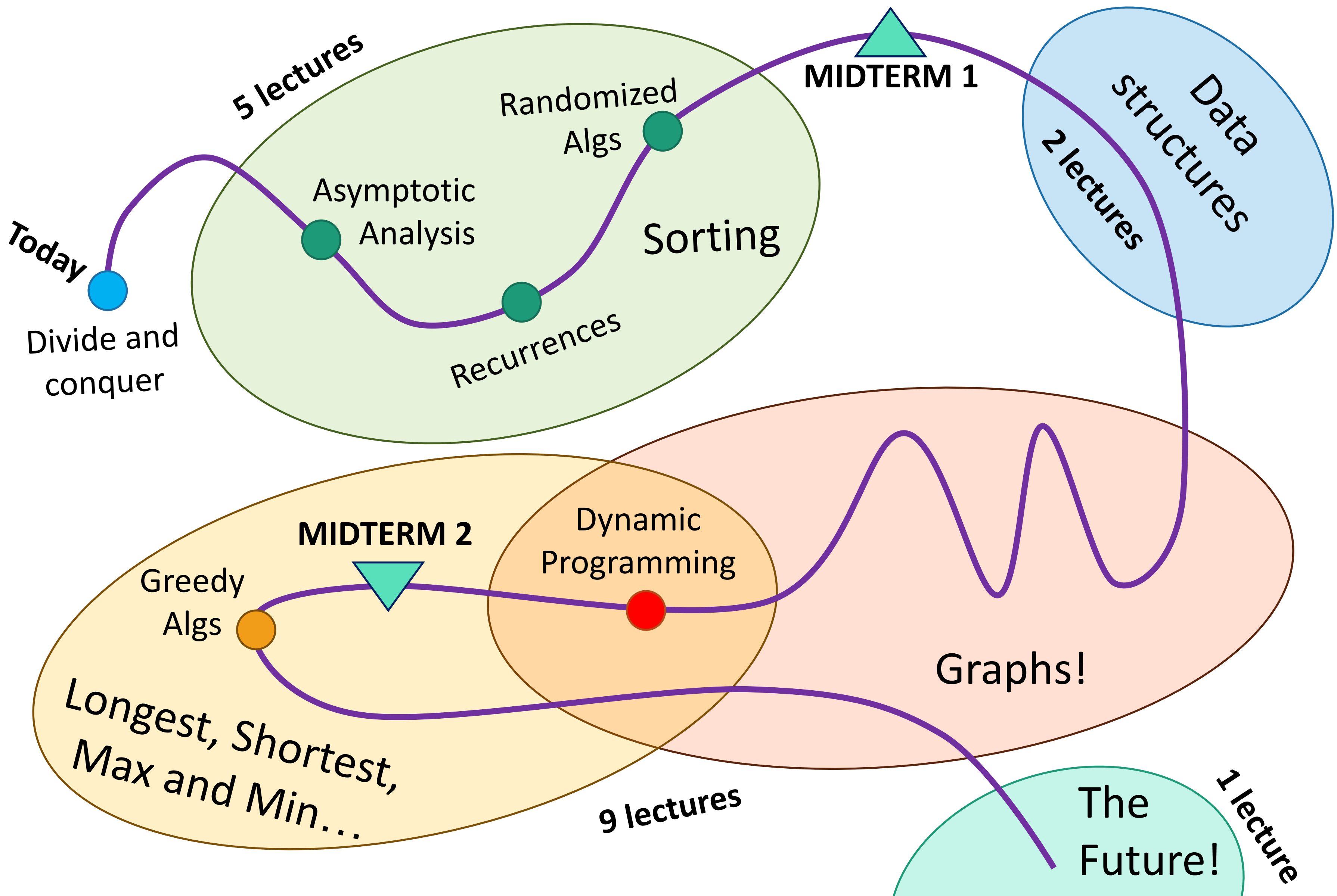
Lucky the
Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

Both sides are necessary!

More detailed schedule on the website!

Roadmap



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

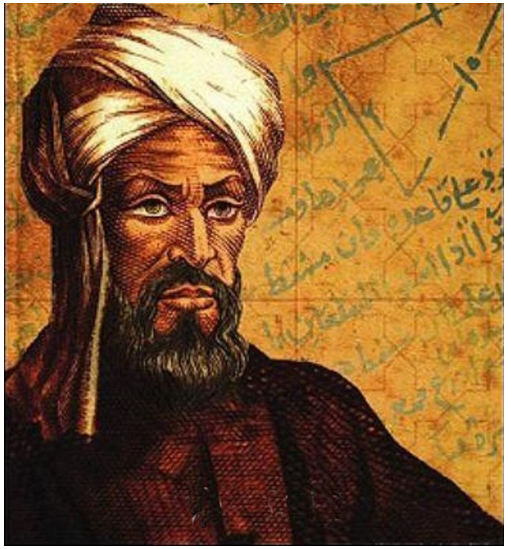
Today's goals

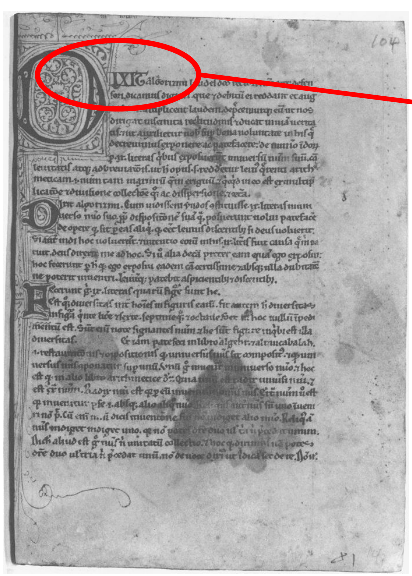
- Karatsuba Integer Multiplication
- Technique: Divide and conquer
- Meta points:
  - How do we measure the speed of an algorithm?

Let’s start at the beginning

Etymology of “Algorithm”

- Al-Khwarizmi 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.





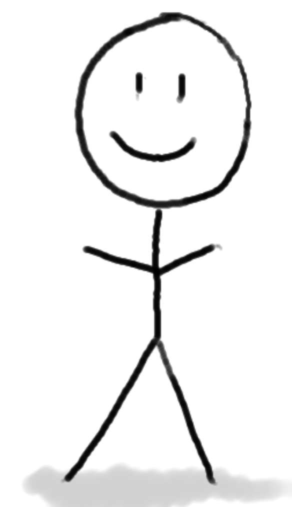
Dixit algorizmi
(so says Al-Khwarizmi)

- 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

$$
\begin{array}{r}
12345678895931413 \\
\times \quad 45638235520395533 \\
\hline
\end{array}
$$

Integer Multiplication

$$
\begin{array}{rr}
 & \overbrace{1233925720752752384623764283568364918374523856298}^{n} \\
\times & 4562323582342395285623467235019130750135350013753 \\
\hline
\end{array}
$$

???

How long would this take you?

About n^2 one-digit operations

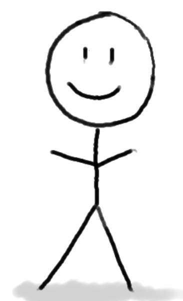
At most n^2 multiplications,
and then at most n^2 additions (for carries)
and then I have to add n different 2n-digit numbers…

Is that a useful answer?

- How do we measure the runtime of an algorithm?

All running the same algorithm…

python™

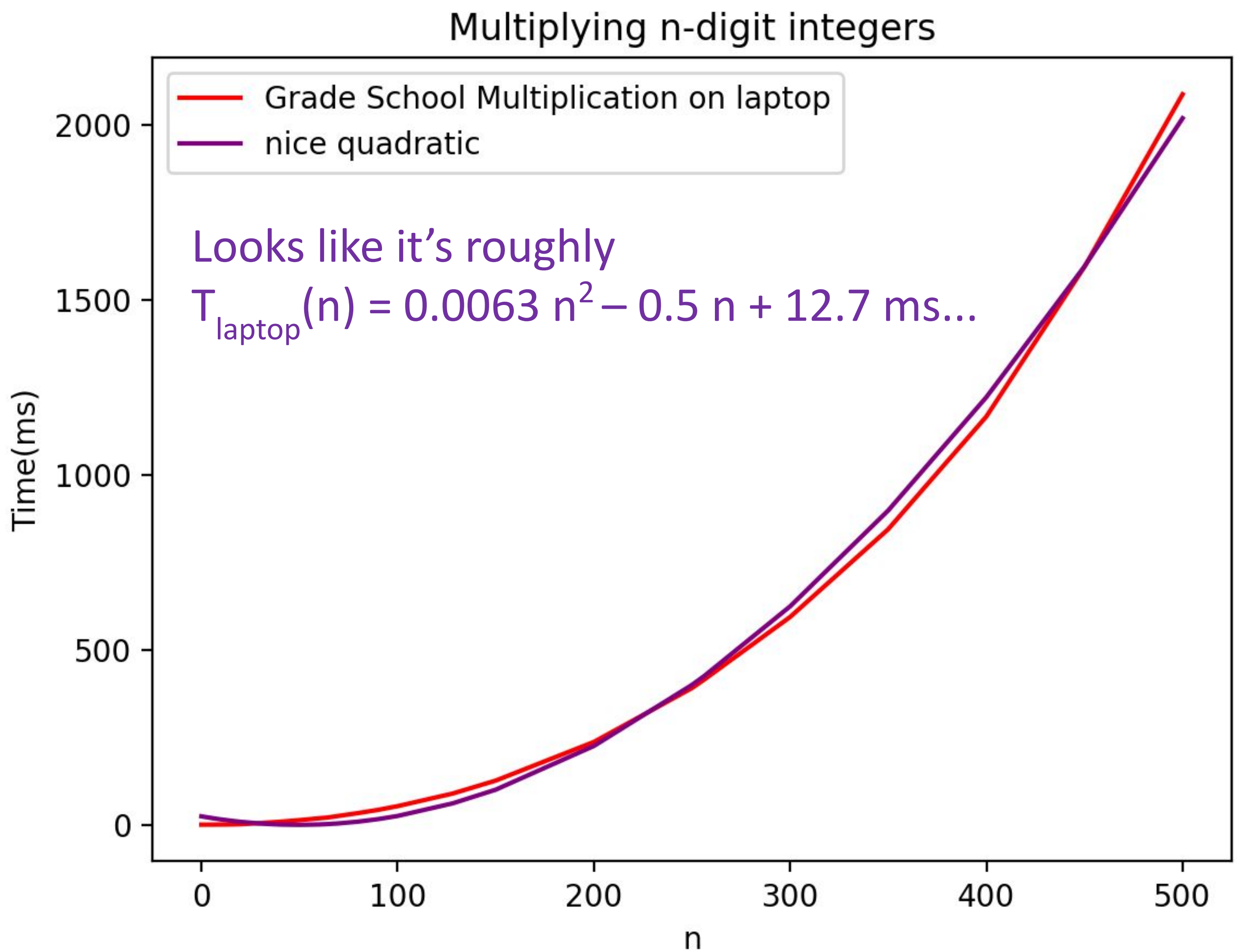


C++

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

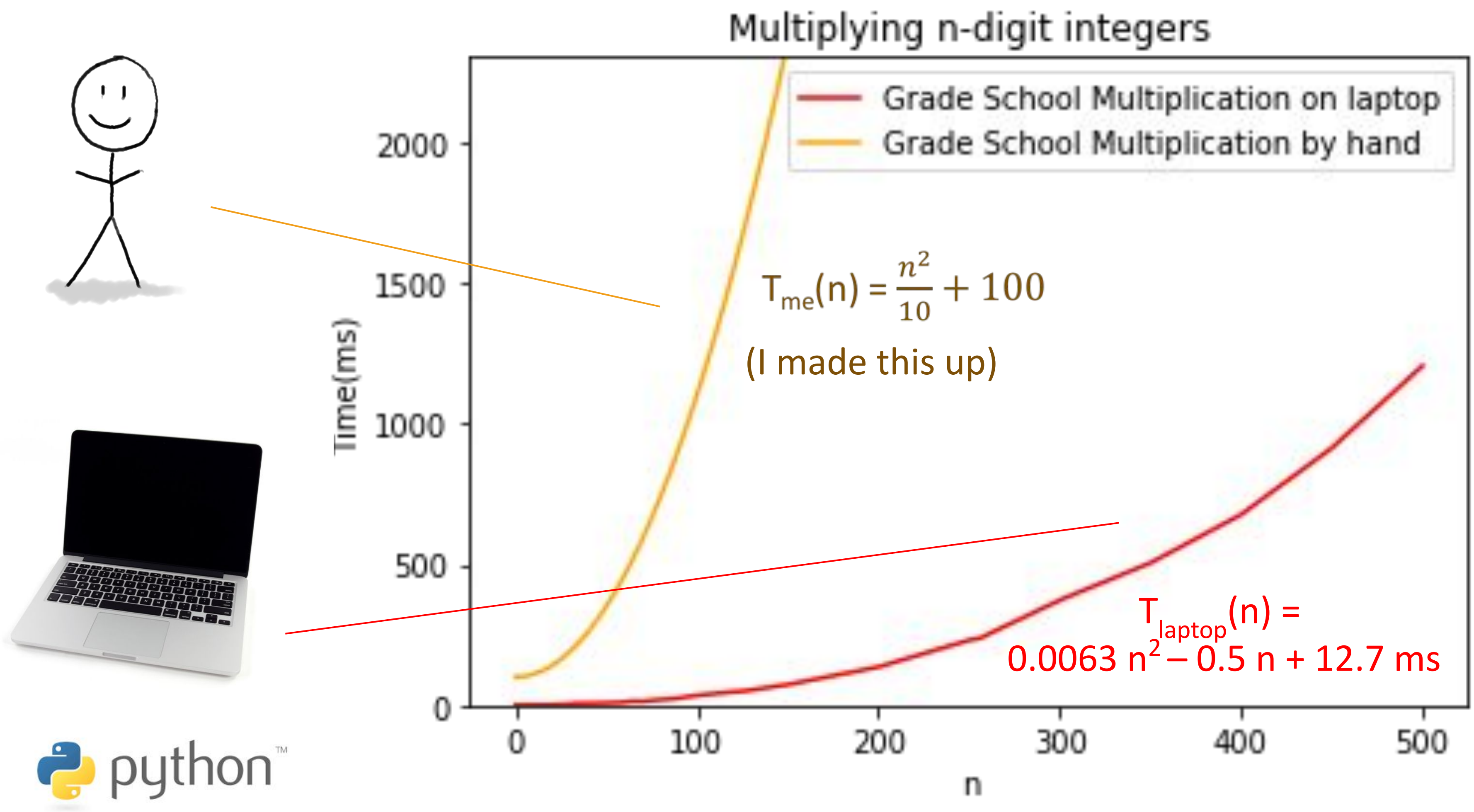
For grade school multiplication,

with python, on your laptop…



I am a bit slower than my laptop

But the runtime scales like n^2 either way.



Asymptotic analysis

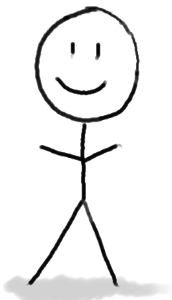
- How does the runtime scale with the size of the input?
  - Runtime of grade school multiplication scales like $\mathbf{n^2}$

- We'll see a more formal definition on the next class

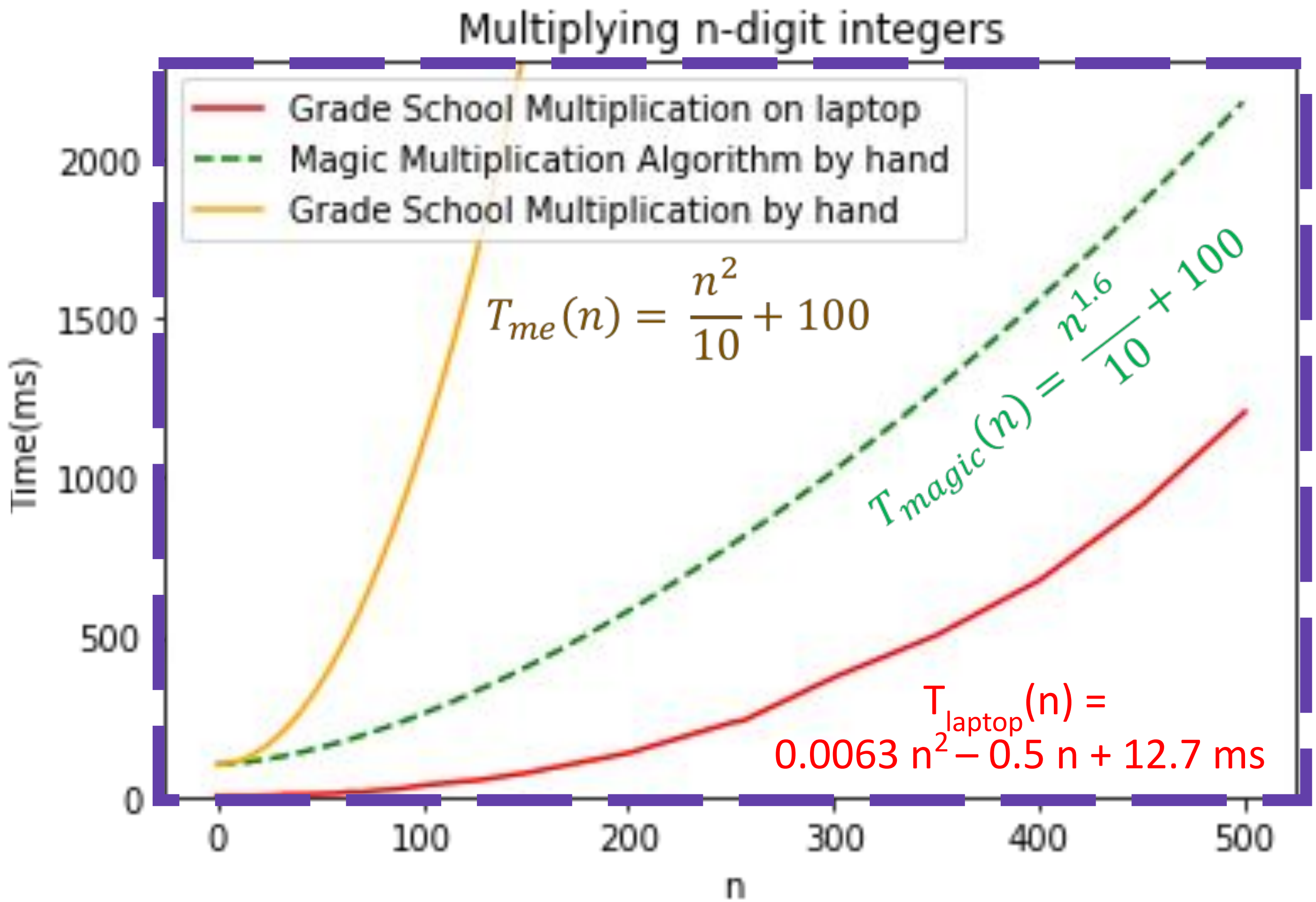
Is this a useful answer?

Hypothetically…

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

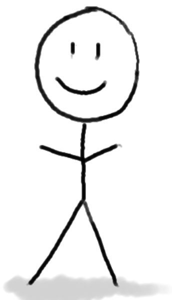


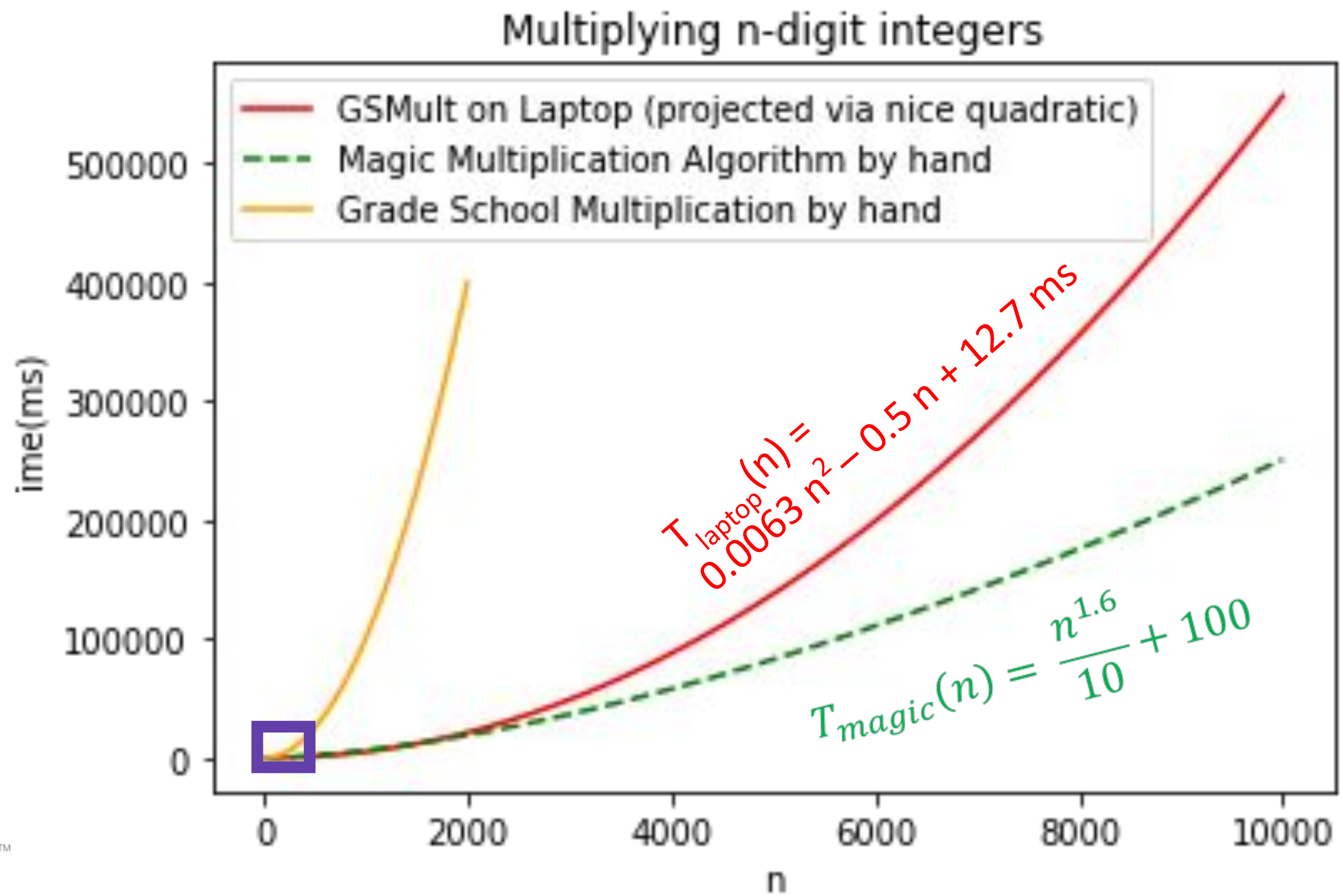
python™



Let n get bigger…

For large enough n, **it would be faster to do the magic algorithm by hand than the grade school algorithm on a computer!**





Asymptotic analysis
is a useful notion…

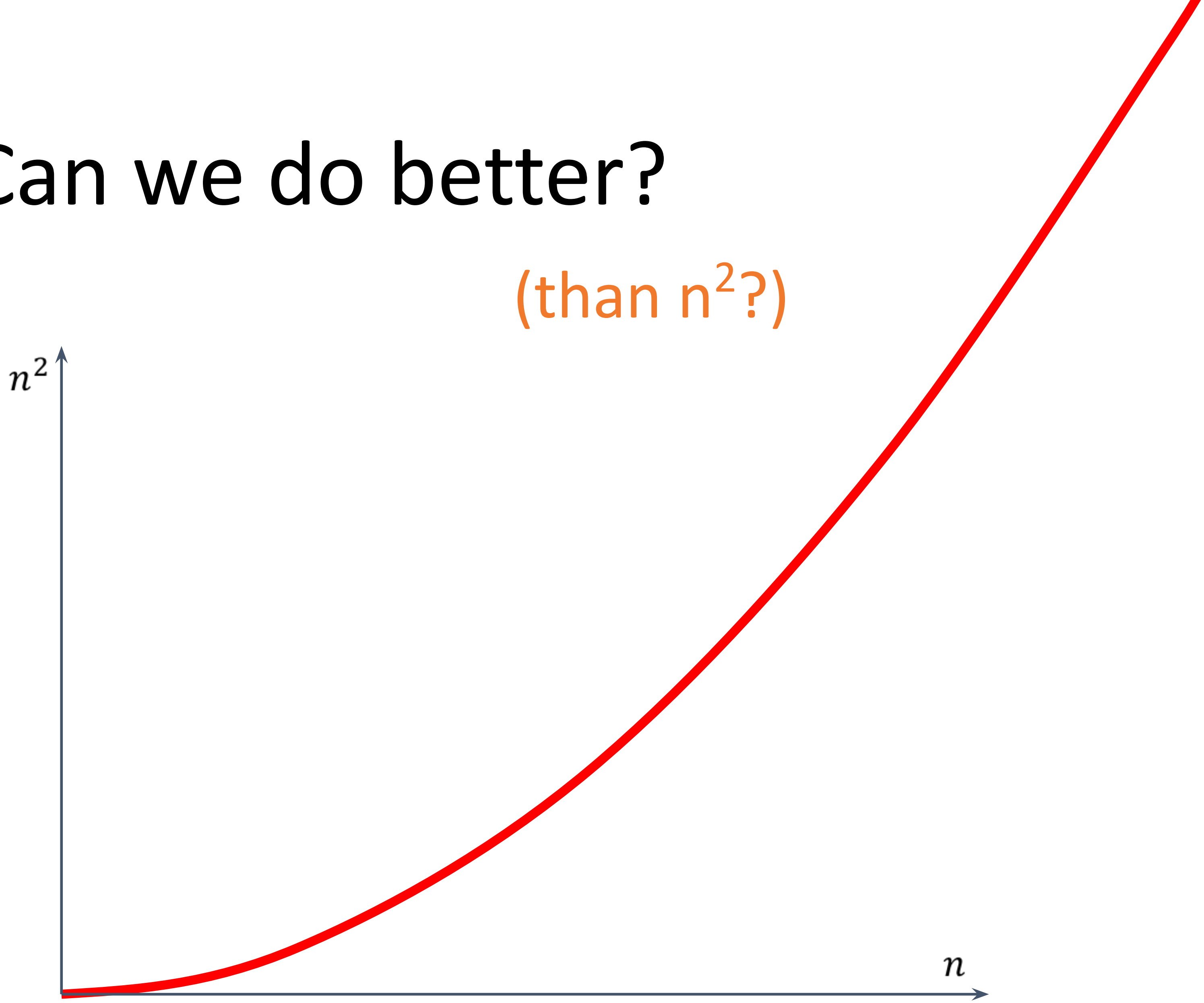
- How does the runtime scale with the size of the input?

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

- So the question is…

Can we do better?

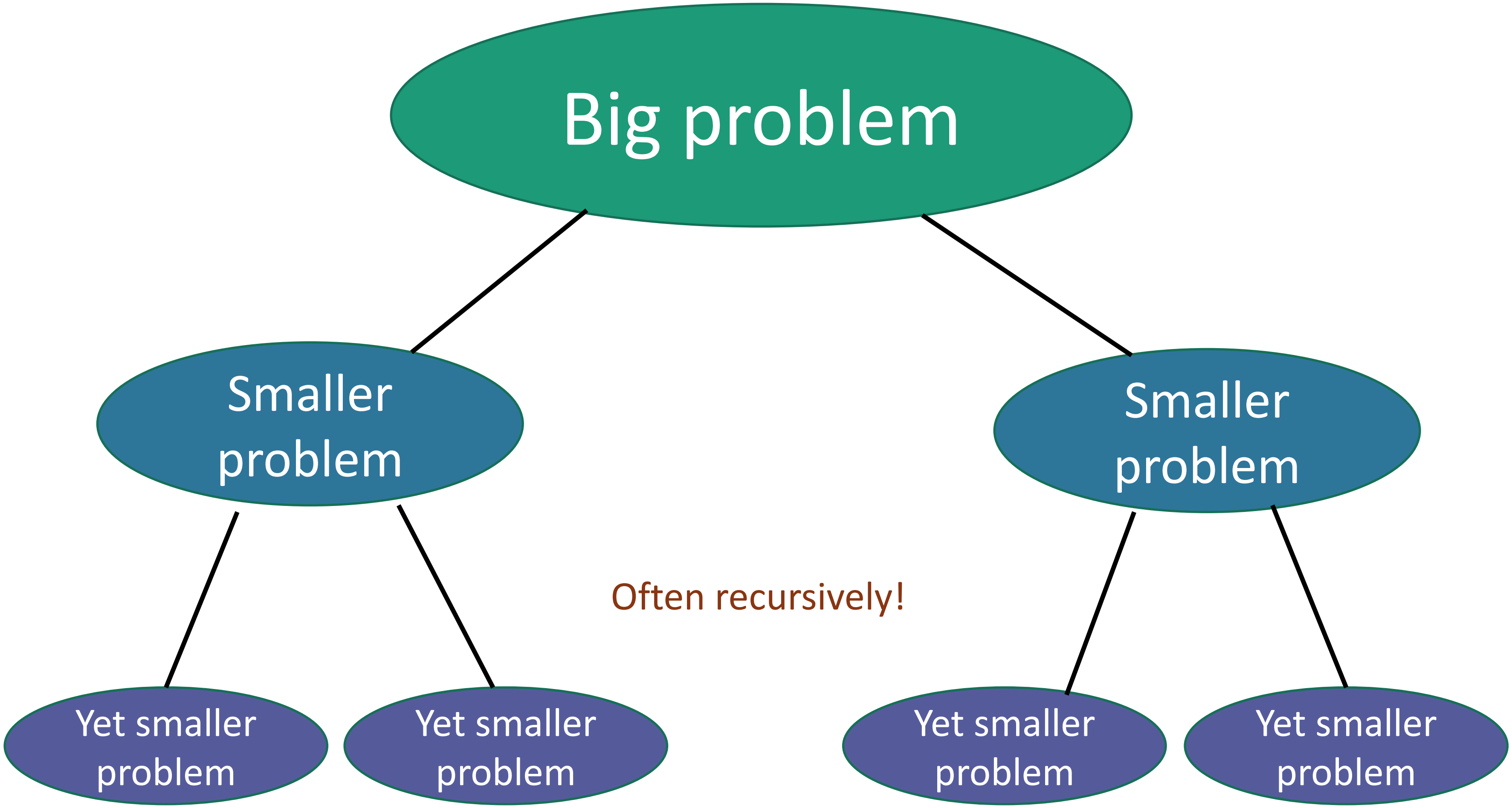
(than n^2?)



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:

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

$$
\begin{aligned}
&1234 \times 5678 \\
&= (12 \times 100 + 34)\,(56 \times 100 + 78) \\
&= \underbrace{(12 \times 56)}_{1}\,10000 + \big(\underbrace{34 \times 56}_{2} + \underbrace{12 \times 78}_{3}\big)\,100 + \underbrace{(34 \times 78)}_{4}
\end{aligned}
$$

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 $\Rightarrow$ Four (n/2)-digit multiplies

Divide and conquer algorithm

x,y are n-digit numbers

Multiply(x, y):

- **If** n=1:
  - **Return** xy
- Write $x = a\ 10^{\frac{n}{2}} + b$
- Write $y = c\ 10^{\frac{n}{2}} + d$
- Recursively compute ac, ad, bc, bd:
  - ac = **Multiply**(a, c), etc…
- Add them up to get xy:
  - xy = ac 10^n + (ad + bc) $10^{n/2}$ + bd

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

Say n is even…

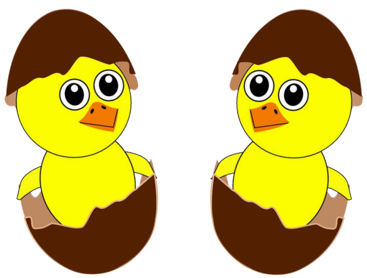
a, b, c, d are n/2-digit numbers

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

Siggi the Studious Stork

How long does this take?

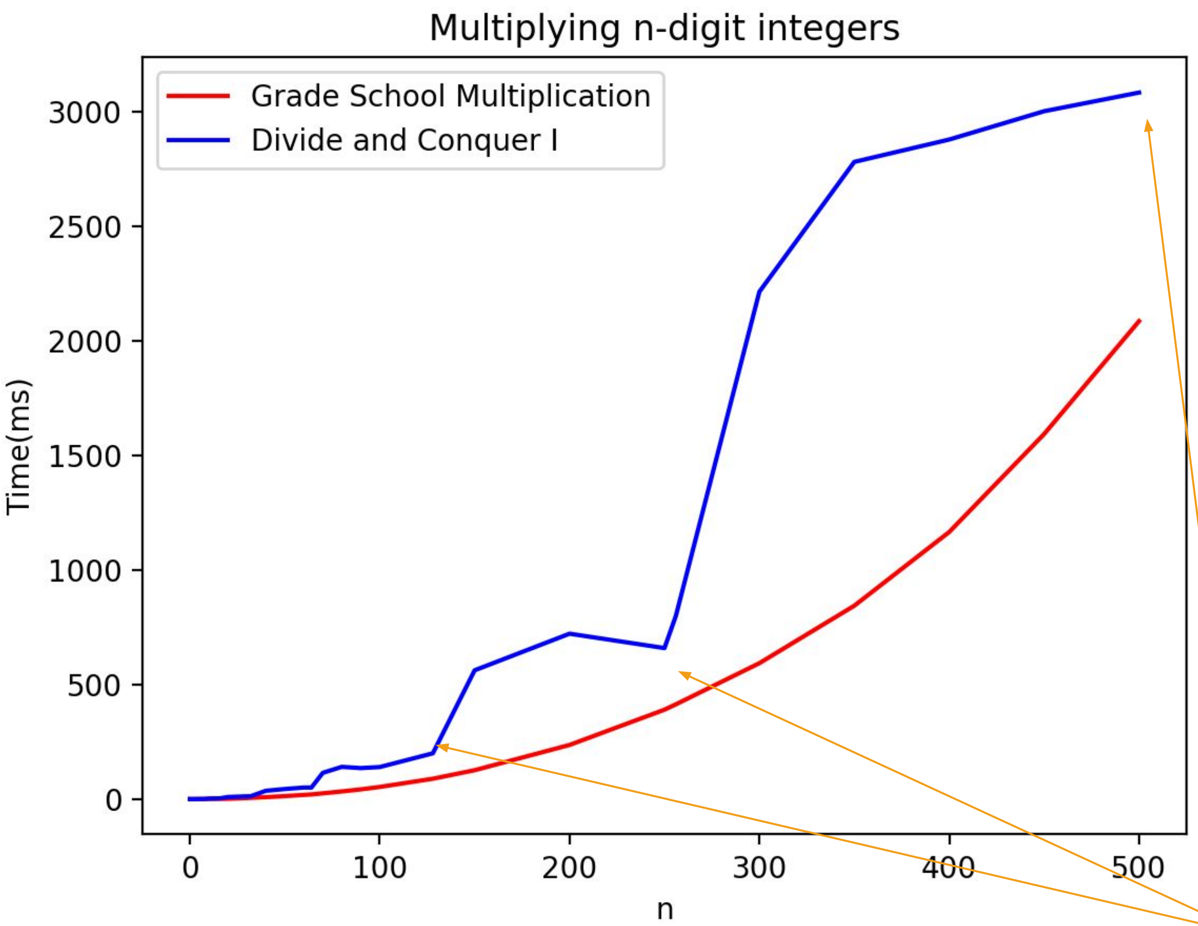
- Better or worse than the grade school algorithm?
  - That is, does the number of operations grow like n^2 ?
  - More or less than that?



Think-Pair-Share:
(2 min: try to think- how fast is our new algorithm?
2 min: what does the person next to you think? why?)

- 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…

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!

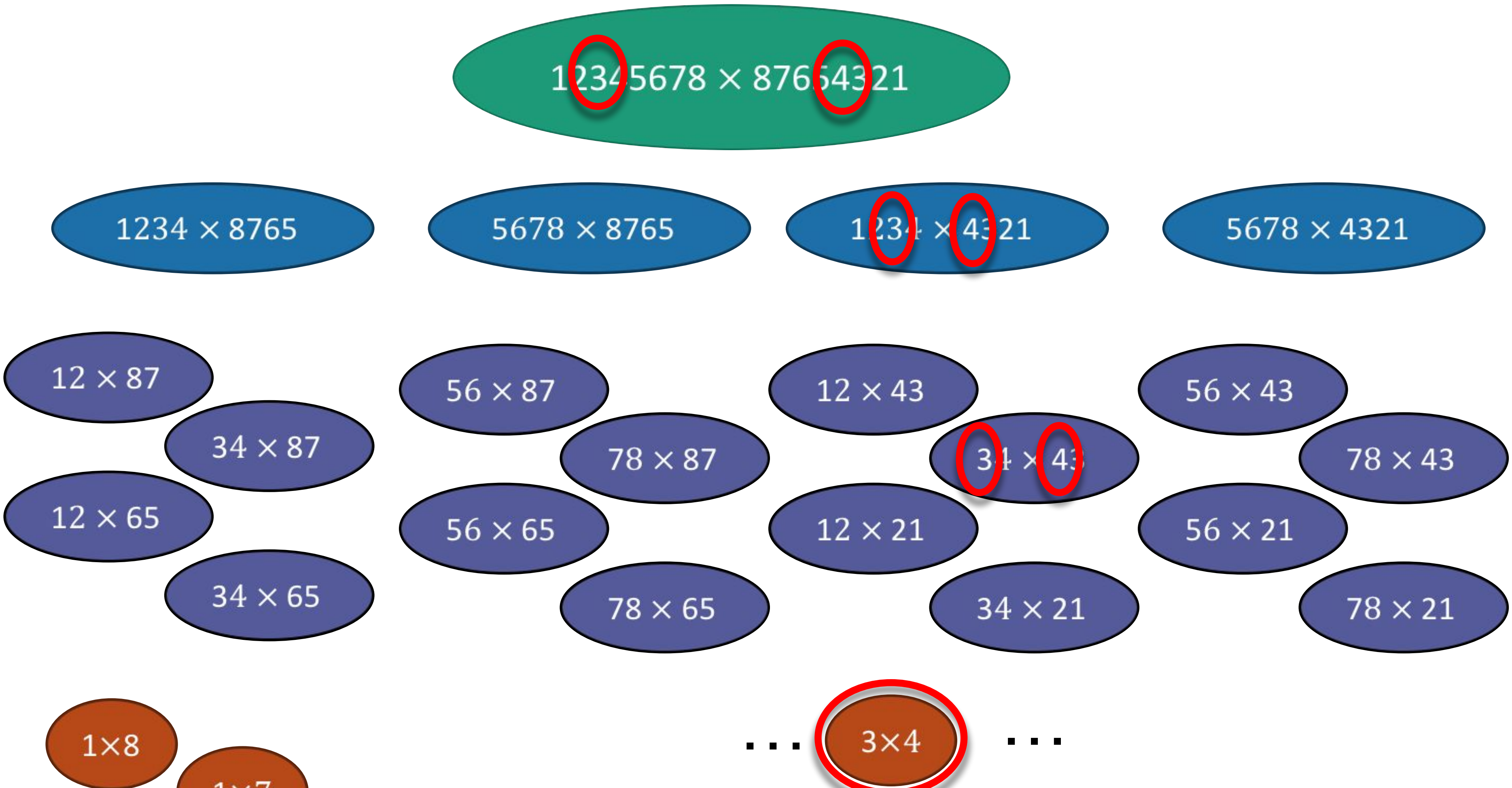
Plucky the Pedantic Penguin

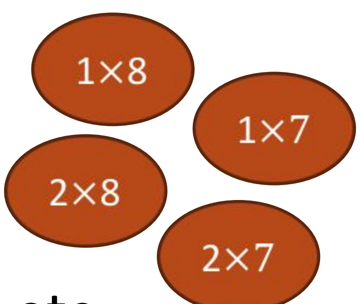
2. Try to understand the running time analytically

- Claim:

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

How many one-digit multiplies?





etc…

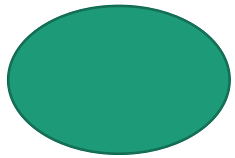
Claim: there are n² one-digit problems.

Every pair of digits still gets multiplied together separately.

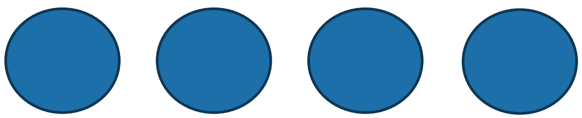
So the running time is still at least n^2.

Another way to see this*

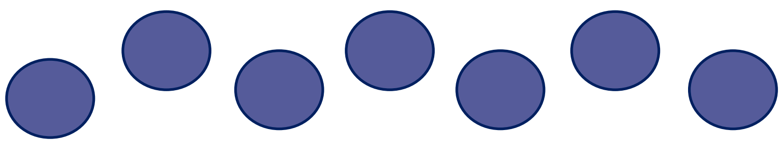
*we will come back to this sort of analysis later and still more rigorously.



1 problem of size n

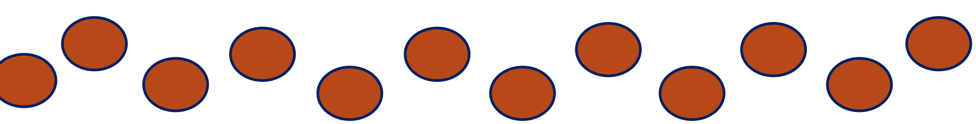


4 problems of size n/2



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

…



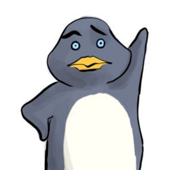
$\underline{\;n^2\;}$ problems of size 1

- 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.

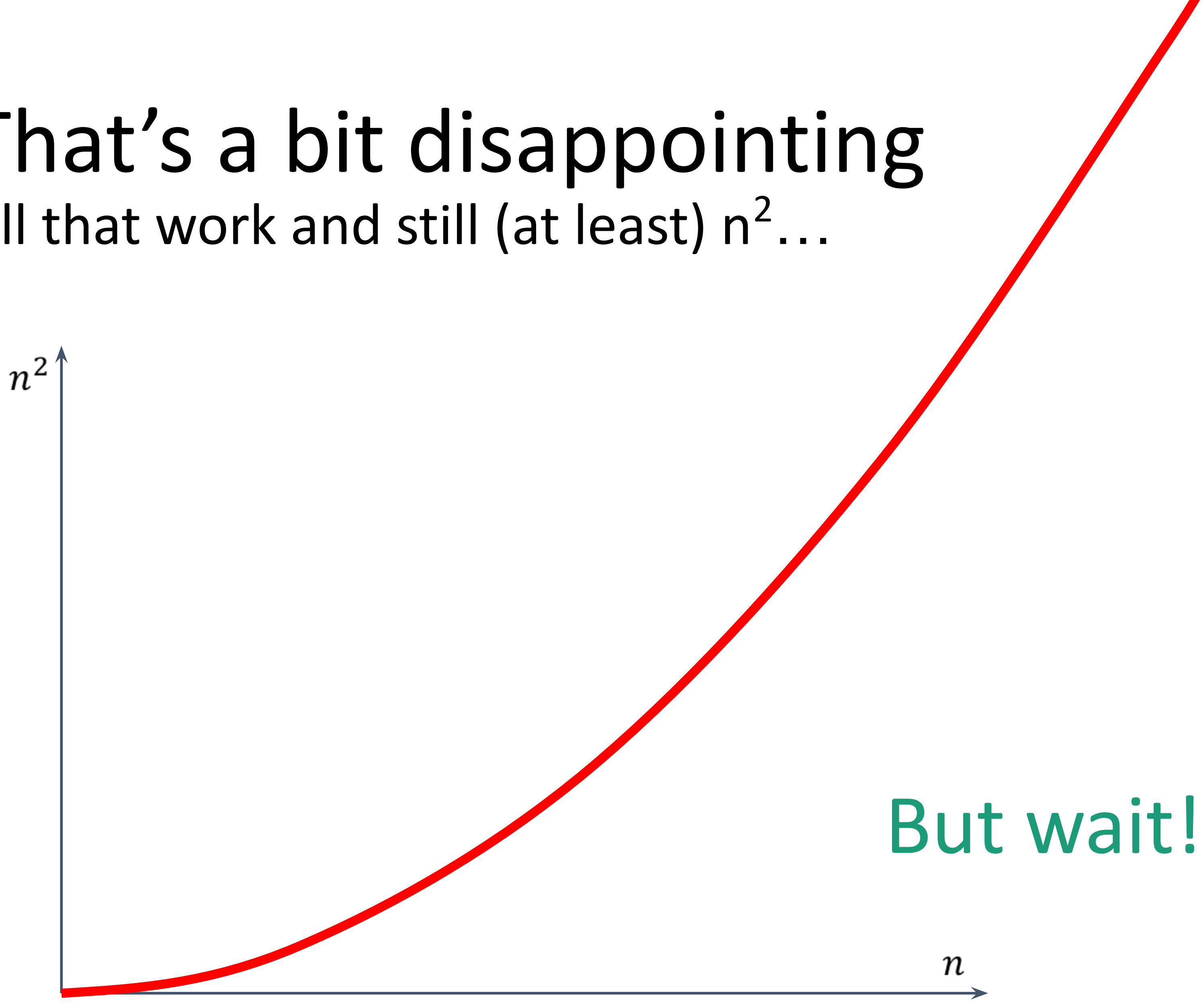
- Let T(n) be the time to multiply two n-digit numbers.
- Recurrence relation:
  - $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + \text{(about n to add stuff up)}$

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!

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

Need these three things

- If only we recurse three times instead of four…

Karatsuba integer multiplication

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

Subtract these off

Subtract these off

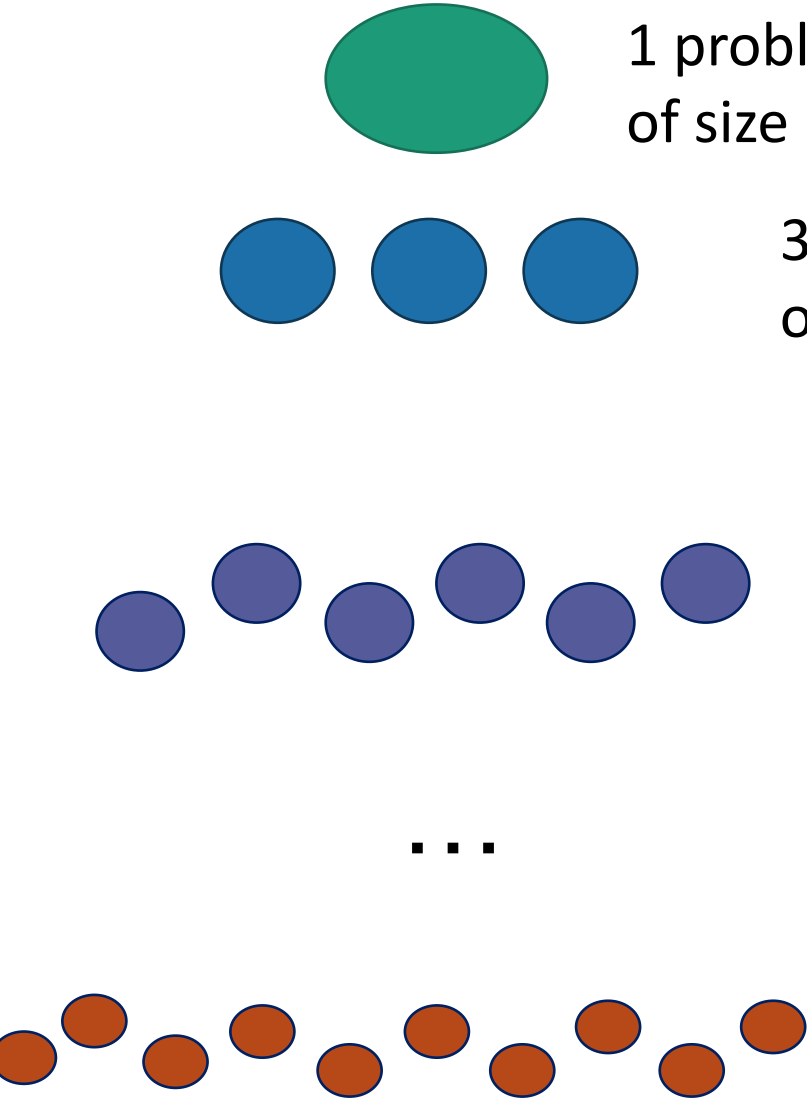
get this

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

- Assemble the product:

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

What's the running time?



1 problem
of size n

3 problems
of size n/2

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

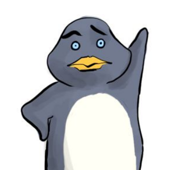
…

$n^{1.6}$ 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…

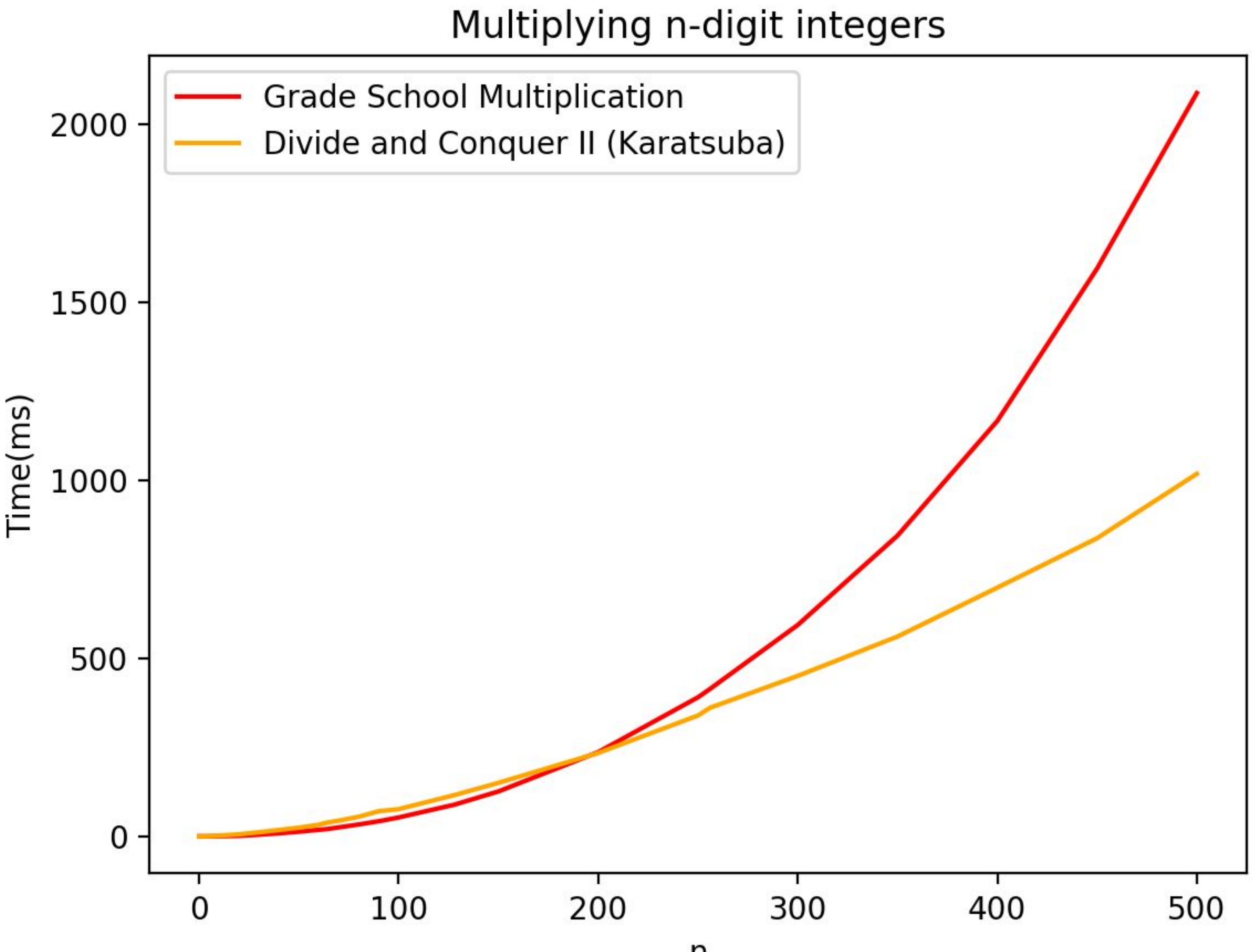
$3^{\log\_2(n)} = n^{\log\_2(3)} \approx n^{1.6}$
problems of size 1.

We still aren't accounting for the 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

- Algorithms are:
  - Fundamental, useful, and fun!
- In this course, we will develop both algorithmic intuition and algorithmic techniques
  - **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