CS 161
Design and Analysis of Algorithms

Lecture 1:
Logistics, introduction, and multiplication!
Welcome to CS161!

Who are we?
- Instructor: Aviad Rubinstein
- Awesome CAs:
  - Richard Mu (head CA)
  - Dana Murphy (head CA)
  - Stefanie Baby
  - Jon Deaton
  - Ingerid Fosli
  - Anchit Gupta
  - Haojun Li
  - Deepak Narayanan
  - Jayden Navarro
  - Eric Redondo
  - Annie Shi
  - Abraham Starosta

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

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

- 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)

Compilers (CS 143)

Networking (CS 144)

Cryptography (CS 255)

Machine learning (CS 229)

Computational Biology (CS 262)

The Computational Lens
Algorithms are useful

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

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

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

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

• In this course you will:
  • Learn to **think analytically** about algorithms
  • Flesh out an “**algorithmic toolkit**”
  • Learn to **communicate clearly** about algorithms
The algorithm designer’s question

Can I do better?

Algorithm designer
The algorithm designer’s internal monologue...

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

Can I do better?

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

Plucky the Pedantic Penguin

Detail-oriented
Precise
Rigorous

Algorithm designer

Lucky the Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

Both sides are necessary!
Roadmap

- **Today**
  - Divide and conquer

- **5 lectures**
  - Asymptotic Analysis
  - Randomized Algs
  - Recurrences

- **Sorting**

- **Randomized Algs**

- **MIDTERM 1**

- **2 lectures**
  - Data structures

- **MIDTERM 2**
  - Greedy Algs
  - Longest, Shortest, Max and Min...

- **9 lectures**
  - Dynamic Programming

- **Graphs!**

- **The Future!**

More detailed schedule on the website!
Course elements and resources

- Course website:
  - cs161.stanford.edu
- Lectures
- Sections
- Homework
- Exams
- Office hours, Piazza, textbooks, etc
Lectures

• Right here, Tu/Th, 3-4:20!

• Resources available:
  • Slides, Lecture Notes

• Goal of lectures:
  • Hit the most important points of the week’s material
    • Sometimes high-level overview
    • Sometimes detailed examples
How to get the most out of lectures

• **During lecture:**
  • Show up, ask questions, put your phone away.
  • May be helpful: take notes on printouts of the slides.

• **After lecture:**
  • Go through the exercises on the slides.

• **Between lectures – more resources:**
  • Go to sections
  • Read textbook/notes
  • Do the homework

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

Siggi the Studious Stork (recommended exercises)

Ollie the Over-achieving Ostrich (challenge questions)
Sections

Sections are “mandatory”.

• OK, actually...
  • we will *not* actually take attendance;
  • no sign up needed
    • try to go to same section if possible (get to know your CA!)
  • but *sections are the best way to learn the material in CS161*
    • ... so likely to (indirectly) affect your grade after all.

• **Goal of sections:**
  • Show you how to apply the ideas you learnt in lecture
  • NOT for HW help! (→ Piazza, office hours).
Homework

- Assignment will be posted **TUESDAY** after lecture.
  - On the course website.
- Due the next **TUESDAY** before lecture. (3pm)
  - On GradeScope.
- Solutions are posted the following **THURSDAY**.
  - On the course website.

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW DUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW ASSIGNED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW RETURNED (target)</td>
<td></td>
<td></td>
<td>SOLUTIONS RELEASED</td>
<td></td>
</tr>
</tbody>
</table>

HW1 will be posted today!
Exams

• There will be **two midterms** and a **final**.
  • **MIDTERM 1:** 10/16, in class
  • **MIDTERM 2:** 11/6, in class
  • **FINAL:** 12/11, 3:30-6:30pm

• **If you have a conflict with these exams, contact head CAs (rmu, dkm0713) ASAP!**
Talk to us

• Sign up for Piazza:
  • Course announcements will be posted there
  • Discuss material with TAs and your classmates

• Office hours:
  • See website for schedule
  • Suggestion: do not go to office hours for nonspecific “homework help.” Go with a specific question.
Bug bounty!

• We hope all PSETs and slides will be bug-free.

• However, I sometimes maek typos.

• If you find a typo (that affects understanding*) on slides, IPython notebooks, lecture notes, or PSETs:
  • Let us know! (Email aviad, rmu, and dkm0713, or post on Piazza).
  • The first person to catch a bug gets a bonus point.

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

• Please help us improve the course!

• Give feedback early and often!

• There is an anonymous Google form on the course website.

• We’ll ask for specific feedback (“prompt of the week”) as part of your homework.
Everyone can succeed in this class!

1. Work hard
2. Ask for help
3. Help others
   (Think)-Pair-Share:
   2 min:
   Talk to person(s) next to you;
   tell them they’ll succeed in this class,
   and that you’re here help them!

4. OK... also work hard
Everyone can succeed in this class!

Most importantly:
Don’t just find the solution.

Make sure you really understand:
- Why is this the solution?
- What are the mental steps to coming up with the solution?
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.
• 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 = ?

44 x 97
Integer Multiplication

44
x 97

---
Integer Multiplication

\[
\begin{array}{c}
1234567895931413 \\
\times \\
4563823520395533
\end{array}
\]
Integer Multiplication

\[ \begin{array}{c}
1233925720752752384623764283568364918374523856298 \\
\times \\
4562323582342395285623467235019130750135350013753 \\
\end{array} \]

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...

- We measure how the runtime scales with the size of the input.
For grade school multiplication, with python, on your laptop...

Looks like it’s roughly

\[ T_{\text{laptop}}(n) = 0.0063 \ n^2 - 0.5 \ n + 12.7 \text{ ms...} \]
I am a bit slower than my laptop

But the runtime scales like $n^2$ either way.

$I_m(n) = \frac{n^2}{10} + 100$

(I made this up)

$T_{laptop}(n) = 0.0063 n^2 - 0.5 n + 12.7 \text{ ms}$
Asymptotic analysis

• How does the runtime scale with the size of the input?
  • Runtime of grade school multiplication scales like \( n^2 \)

• We’ll see a more formal definition on Thursday

Is this a useful answer?
Hypothetically...

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

$$T_{me}(n) = \frac{n^2}{10} + 100$$

$$T_{magic}(n) = \frac{n^{1.6}}{10} + 100$$

$$T_{laptop}(n) = 0.0063 n^2 - 0.5 n + 12.7 \text{ ms}$$
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!

Multiplying $n$-digit integers

$T_{\text{laptop}}(n) = 0.0063 n^2 - 0.5 n + 12.7 \text{ ms}$

$T_{\text{magic}}(n) = \frac{n^{1.6}}{10} + 100$
Asymptotic analysis is a useful notion...

• How does the runtime \textit{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

Big problem

- Smaller problem
  - Yet smaller problem
  - Yet smaller problem

- Smaller problem
  - Yet smaller problem
  - Yet smaller problem

Often recursively!
Divide and conquer for multiplication

Break up an integer:

\[ 1234 = 12 \times 100 + 34 \]

\[ 1234 \times 5678 = (12 \times 100 + 34)(56 \times 100 + 78) \]
\[ = (12 \times 56)10000 + (34 \times 56 + 12 \times 78)100 + (34 \times 78) \]

One 4-digit multiply

Four 2-digit multiplies
More generally

Break up an \( n \)-digit integer:

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

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

\[
= (a \times c)10^n + (a \times d + c \times b)10^{n/2} + (b \times d)
\]

One \( n \)-digit multiply \quad \rightarrow \quad Four \ (n/2)\)-digit multiplies
Divide and conquer algorithm

x, y are n-digit numbers

Multiply(x, y):

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

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 $10^n$”?
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?

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

Every pair of digits still gets multiplied together separately.

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

etc...
Another way to see this*

1 problem of size $n$

4 problems of size $n/2$

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

…

$\frac{n^2}{4^{\log_2(n)}}$ problems of size 1

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

- 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!
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
  • \((a+b)(c+d)\)

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

• Assemble the product:

\[xy = (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d) = ac \cdot 10^n + (ad + bc)10^{n/2} + bd\]
What’s the running time?

1 problem of size $n$

3 problems of size $n/2$

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

$\cdots$

$n^{1.6}$ 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?

• **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

- cs161.stanford.edu

- 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

BEFORE Next time

• Join Piazza!