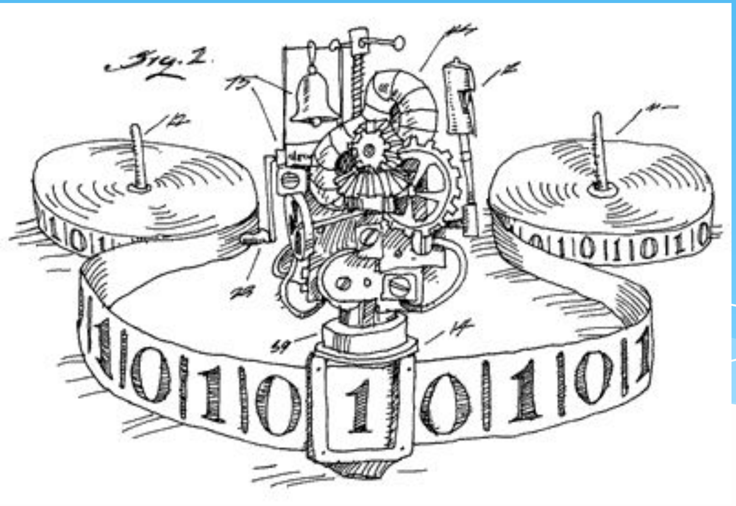
EECS 376: Foundations of Computer Science

Euiwoong Lee

Chris Peikert

Quentin Stout

Jimmy Zhu



Today's Agenda

- Introduction
- Big questions/Outline
- Administration
- Algorithm Design and Analysis
  - Greatest Common Divisor: naïve and clever

Introduction

- I'm Chris
- Research: cryptography, lattices, coding, theory
- MI ('XX)->MA ('96)->CA ('06)->GA ('09)->MI ('15)
- Fun fact: >=5 close family members went to U-M
- (and 0 to Ohio State)

Why are we here?

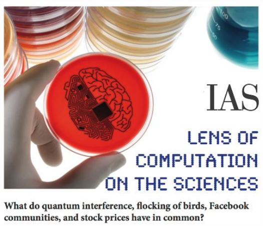
Computer science is no more about computers than astronomy is about telescopes.
--- Edsger Dijkstra

Foundations: What is computation?

Is every problem solvable on a computer?
Can every solvable problem be solved "efficiently"?
Do a finite number of algorithmic techniques solve every solvable problem?
…

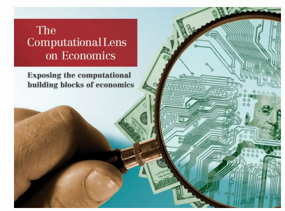
Why is this useful to me?
(fundamental knowledge; rigorous thinking;
edge in solving new problems; interview questions!)

Computational Thinking



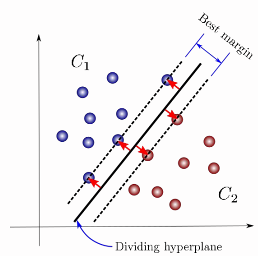
Natural processes

Computational Biology

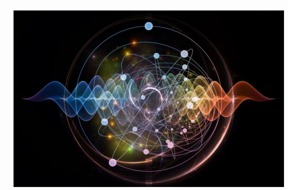


Algorithmic Finance

Robotics



Machine learning
Big Data



Quantum Computation

Course Outline

- Algorithm Design & Analysis (5 lectures)
- Computability theory (7 lectures)
- Complexity theory (6 lectures)
- Randomized algorithms (3 lectures)
- Cryptography (2 lectures)
- Special topics (1 lecture)
- Review (2 lectures)
- **Total:** 25 lectures

  (see schedule on website for more details)

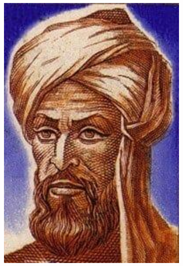
Basic introduction to topics (relatively fast paced)

Several advanced courses on these topics at UM

EECS ECE CSE

Algorithms

Babylonians (2000 BC), compute square-roots
- Greeks, Arabic, Indian, Chinese, … (astronomy, geometry)



The word: Muhammad al-Khwarizmi (780-850 AD)
-Wrote several books including "al-jabr"

1900s: First computing machines appear
(Slow, small memory. Need various tricks to compute usefully.)

Computability

Hilbert (ICM 1900): Is arithmetic “consistent”?
Can every true statement be proved (from axioms)

Godel: No! (famous "incompleteness theorems”)
Turing: Formal model of computer (Turing machine)

Computability: (1930s-60s)
What can/cannot be done (in finite time)?

E.g. Is there a program that tests if two given programs in C/C++ have the same functionality?
Answer: No!

E.g. Hilbert’s 10 problem. Is there an algorithm to determine if an equation has integer solutions? ()
Yuri Matiyasevitch (1975): No!

Complexity

Computability: What can/cannot be done (in finite time)?

Finite time not enough—need to solve fast!

1960's: Focus on clever/fast algorithms
(multiplication in almost-linear time)

Example: Traveling salesperson problem:
Shortest tour through n cities.

Brute force: n! Permutations
n=100 => essentially infinite

Need a smarter way
Is there a poly-time algo?

Complexity

1971-72: Efficient computation (polynomial time)
notion of P vs NP (STOC'71)


Cook


Levin


Karp

No efficient algorithm for TSP unless P=NP

One of the biggest questions of our time: is P=NP?

Million dollar prize (one of 7 questions by Clay Math Inst. 1 already solved)

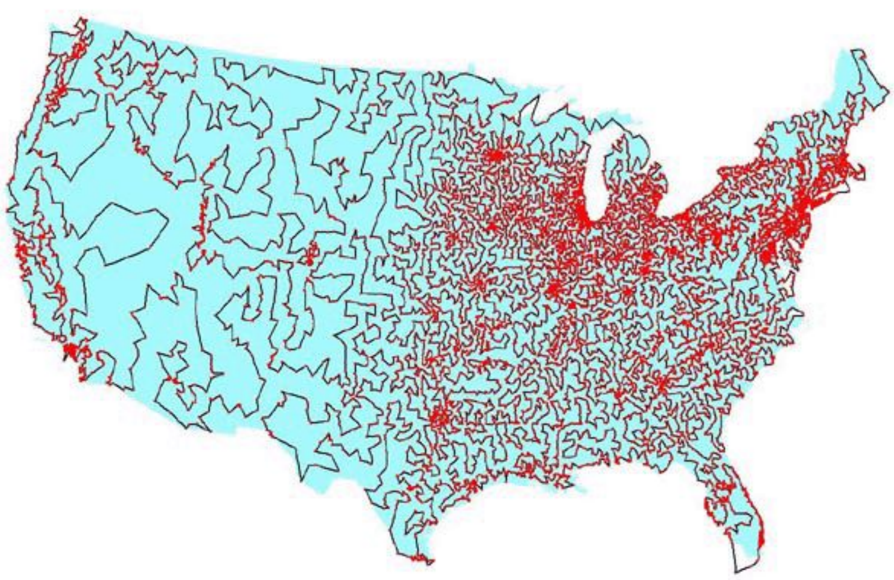
Surprising fact: Good algorithms exist for closely related problems

- Shortest s-t path.
- Chinese postman problem (must pass through each "street")
- …

Approximation Algorithms

Many problems turn out to be NP-hard to solve exactly

But we can solve them approximately, or even exactly in some useful cases.



Class Overview

- **Question:** Suppose that a certain problem seems notoriously "hard". Can we make this a positive?

Cryptography

Can do miraculous things! Modern magic

Each time you browse: you are sending data over (many) public networks

Public key cryptography: Can send a secret message to another person, without arranging a secret code in advance

Digital signatures

Coin-flipping over a telephone
…

Lots of topics

- Algorithms Design & Analysis (5 lectures)
- Computability theory (7 lectures)
- Complexity theory (6 lectures)
- Randomized algorithms (3 lectures)
- Cryptography (2 lectures)
- Special topics (1 lecture)

Some tips:

1) Memorizing vs. Understanding
2) No one understands pseudocode
   formalism often obscures the key insight
3) Practice, practice, practice
4) How can someone find such a brilliant idea (I must be dumb …)

Learning music:

1) Rote memorization of songs doesn't work:
   must learn to read and "feel" music
2) Can't "feel" music by looking at symbols
3) Practice, practice, practice
4) How did Mozart come up with
   this symphony
   (I must be dumb … ??)

Administration

- Website: eecs376.org (Syllabus, Schedule)
- **Text:** https://eecs376.github.io/notes/ (encouraged to read)

- Canvas/Drive: HWs, lecture slides, discussion material, OHs,…
- Piazza: questions (private post if sensitive); search for teammates
- Gradescope - for exams and HW submission

- **Can** attend any lecture/discussion
- **Discussion:** *highly encouraged* to attend

Administration

- 11 weekly HW assignments, due Wednesdays 8pm (Eastern)
  - **No Late Submissions after 11:59pm!** (Staff only help until 8pm)
  - Two lowest scores will be dropped
  - Solutions published shorty after the deadline

- **Midterm:** Mon Feb 20, 7-9pm (tentatively)
- **Final:** Wed Apr 26, 7-9pm

- Participation is important!
  - Questions are welcome!
  - There is no such thing as a “bad question”.

Is this an EEECS class?

- **Question:** Wolverine Access says it is an EECS class. Why does it feel like a math class?

- **Answer:** It's both!

  The only way to answer the questions we raise (and others) is to define mathematical models and apply a rigorous, "proof-based" methodology to the questions.

Is this an EEcs class?

- **CS Example:** Show that there is no program that correctly tests if two given programs in C/C++ have the same functionality.
- **Wrong Approach:** Try all programs… (infinitely many!)
- **Right Answer:** Construct a model that captures any possible program and give a general "impossibility proof".

Design & Analysis of Algorithms

- **Algorithm Design:** A set of methods to create algorithms for certain types of problems.
- **Examples:** Dynamic Programming, Divide and Conquer, Greedy Algorithms
- **Algorithm Analysis:** Methods to prove correctness of algorithms and determine the amount of resources (time, memory, etc.) they need to run.
- **Examples:** potential function arguments, recurrences, Master Theorem, exchange arguments

Greatest Common Divisor

- **Definition:** Let x,y N (positive integers).
- The Greatest Common Divisor (gcd) of x and y is the largest z N that divides both x and y.
- If gcd(x,y)=1 then x and y are said to be **coprime**.

- **Examples:**
  - gcd(21,9) = ?
  - gcd(121,5) = ?
  - gcd(62615533, 62425477) = ?
  - "Put 62615533/62425477 in simplest form."

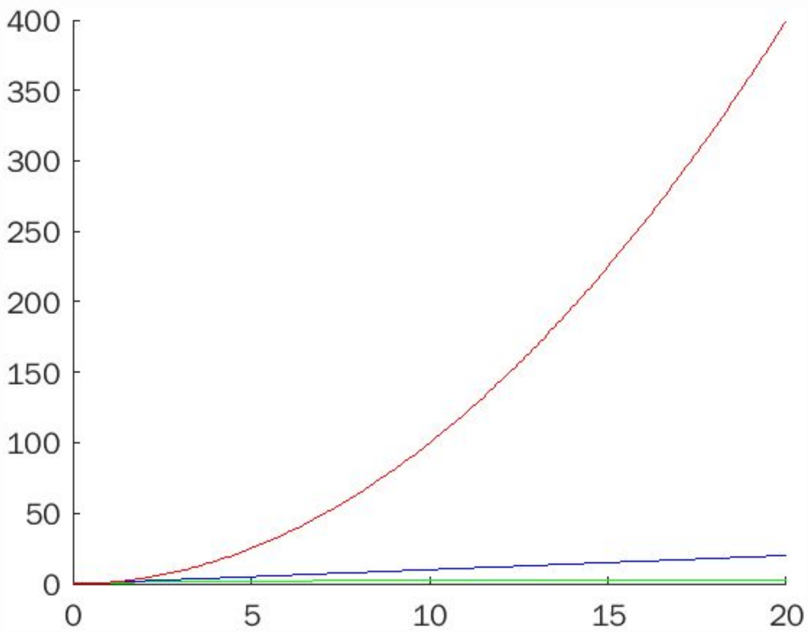
- **Algorithm 1:** For z=y down to 1: if z divides both x and y, return z.
- **Runtime:** O(y) division operations. Is this "efficient"?

Review: Running Time

- We measure the "efficiency" of an algorithm by how its (<u>*worst-case*</u>) ***runtime*** scales with the "input size."

- We express this asymptotically: e.g., etc, where is the **input size**.

- Common interpretations of "size":
  - size of array = # elements
  - size of graph = # vert. + # edges
  - size of integer = # **digits**
  - **Rule of thumb:** size = # bits of memory to store on a



Efficient "runtime <u>polynomial</u> in input size"

Step 2: Analyze runtime of the naïve solution

- **Q:** Suppose and each have digits. How large can they be?
  - Up to (in binary,)
  - The *value* of an integer is **exponential** in its *size*!
- **Q:** What's the runtime of the naïve O(y)-time algorithm?

  Recall: "size" of an integer is # digits

  - <u>Exponential</u> *in the input size n!* (<u>Not efficient</u>.)

Naïve():
for :
  if divides and ,
    return

Step 3: Think strategically

- **Strategy:** *Recursively* solve the problem, by reducing to *smaller* numbers.
- Suppose x >= y. Observe: gcd(x,y) = gcd(y, x-y).

Proof: Any d that divides both x and y, also divides x-y (and y, still).
Conversely, any d that divides y and x-y, also divides x (and y, still).
So the common divisors of x,y are *exactly* the common divisors of y,x-y.
Hence, their *greatest* common divisors are equal.

How far can we reduce?

- In general, we can reduce times until .
- **Q:** What is ? Hint: Think division.
  - the remainder of divided by
- Theorem: for , .

Step 4: Code it up

- We have just discovered the ***Euclidean Algorithm*** to compute the ***greatest common divisor*** of two integers.

  Example: gcd (21,9), gcd(13,8)

  [Calculator]

```
Euclid(): // for
if divides : return
return Euclid()
```

Euclid, 300 BCE

- What is the runtime of Euclid (as a function of input size)?
- Is it efficient?
- Next time: **potential** argument.