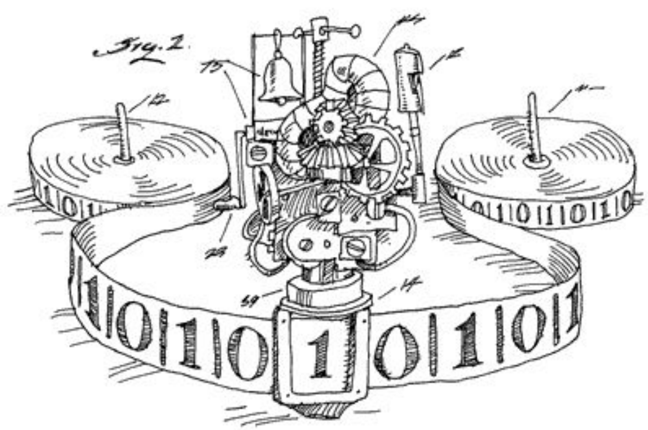
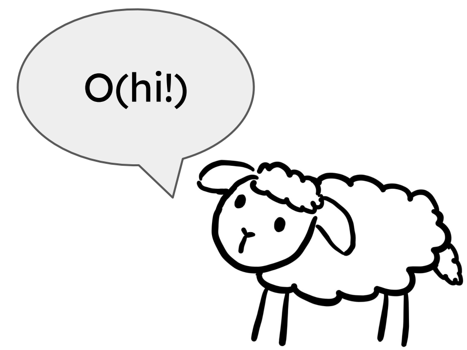
EECS 376: Foundations of Computer Science

Nicole Wein ("wine")

she/her





"Of all the courses I've ever taken, this was one of them"
- New York Times Review of CS Courses

"We cannot recommend this course too highly"
- The Michigan Daily

Sheep art by Melissa Zhang: my sister-in-law and mathematics professor at UC Davis

Course Staff

Instructors: 4



Nicole Wein (me), Chris Peikert, Thatchaphol Saranurak, Mark Brehob

Admin Support: 1, Rose Sherry

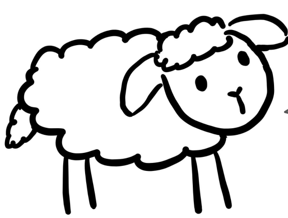
GSIs: 5

IAs: 17

Graders: ~20

Mascots: 1

You **can** attend any lecture(s) and any discussion(s)

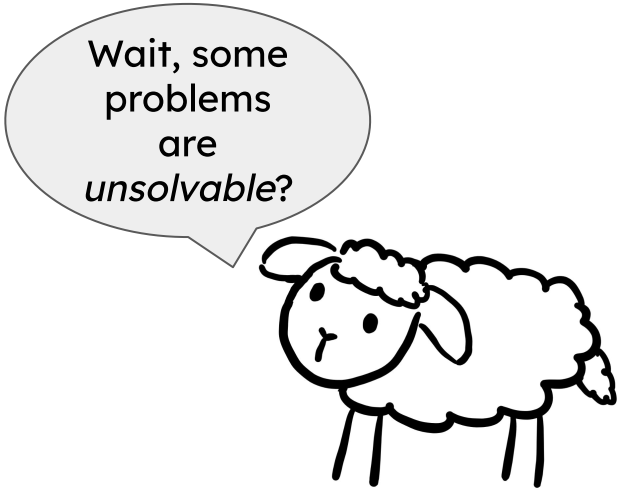


Hi! My name is λ.
I'm a lamb, duh!

Learning Objectives

1. Given a challenging computational problem:
   - Determine whether or not a computer can solve the problem.

   If so:
   - Design an algorithm for it
   - Derive its worst-case running time
   - Prove the correctness of the algorithm



Learning Objectives

2. Get comfortable with major techniques and paradigms:
    - Divide-and-conquer, greed, dynamic programming, the power of randomness

      Problems that are **easy** for a computer
    - Computability

      Problems that are **impossible** for a computer
    - NP-completeness and approximation algorithms

      Problems that are **"probably hard"** for a computer
    - Cryptography

      Using "probably hard" problems for our benefit (hiding secrets)

3. Approach computational problems in a principled and rigorous way
    - In this course you will be asked to think in ways that you have never thought before

Course Outline

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

Is this an EECS 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 is to define mathematical models and apply a rigorous, "proof-based" methodology to the questions.

Why study CS foundations?

(aka theoretical computer science)

Top 7 Reasons

1. Theoretical ideas underlie a lot of software (e.g. Akamai, RSA cryptography, many more)

   "Everyone knows Moore's Law — a prediction made in 1965 by Intel co-founder Gordon Moore that the density of transistors in integrated circuits would continue to double every 1 to 2 years…in many areas, performance gains due to improvements in **algorithms** have vastly exceeded even the dramatic performance gains due to increased processor speed." -Report to the President and Congress: Designing a Digital Future (2010)

2. It's very easy to write code that is either incorrect or will run for millions of years

   "For every complex problem there is an answer that is clear, simple, and wrong." -H. L. Mencken

Why study CS foundations?

(aka theoretical computer science)

Top 7 Reasons

3. In your job you might be asked you to solve a problem that's provably impossible

4. It will help you ace interviews

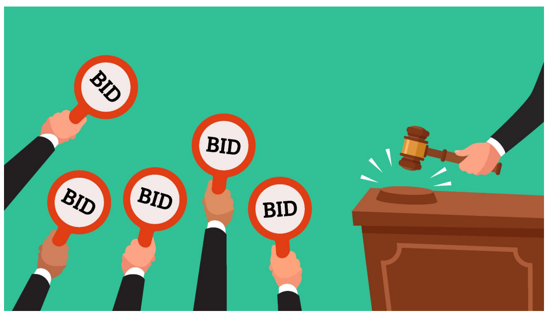
5. Fundamental, machine-independent

Why study CS foundations?

(aka theoretical computer science)

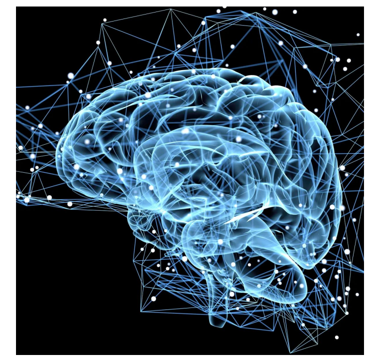
Top 7 Reasons

6. Computation is everywhere (not just in computers)



Economics / Game Theory
e.g. auctions, matching markets

Collective animal behavior



Your brain

7. Mathematical beauty

Struggling is good

This class asks you to comprehend deep ideas and solve challenging problems. You are supposed to struggle!

It is important to have wrong ideas. The creators of every idea in this course had many wrong ideas before having the right idea.

We are here to help! You will likely do better in this course if you ask questions during class and at office hours.

An example of a good question to ask in class: "Can you repeat what you just said?"

Administration

- **Website:** eecs376.org (Syllabus, Schedule, OH, links to the following)
- **Text:** https://eecs376.github.io/notes/
  - Written by Amir Kamil for this course. Follows the lectures quite closely.
- **Canvas/Drive:** HWs, lecture slides+recordings, announcements, etc.
- **Gradescope:** exam and HW submission
- **Piazza:** questions (private post if sensitive)
- **Administrative requests** (e.g. exam conflicts, SSD accommodations, prolonged illness, etc.): fill out the appropriate form (see syllabus)
- **My Proffice hours:**
  - Mondays 2-4pm
  - I will be here 15 minutes before class starts
  - I will stay after class to answer questions about lecture
- **All office hours:** see website, first office hours happen tomorrow

Administration

- 11 weekly HW assignments, due Wednesdays 8pm
  - **No Late Submissions after 9:59pm** (Staff only help until 8pm)
  - Two lowest scores will be dropped
  - Solutions published shorty after the deadline
  - HW 1 will be posted by tomorrow morning, and is due next Wednesday at 8pm

- **Midterm:** Wednesday March 6, 7-9pm
- **Final:** Wednesday May 1, 7-9pm

Collaboration

We encourage collaboration! See syllabus for more details

- When writing your solution, the **write-up must be done individually and entirely on your own**, and **you may not look at anyone else's write-up, including your collaborators'**.
- When turning in work that benefited from a collaboration, **you must acknowledge your collaborators** (as your answer to question 0 on the assignment).

Encouraged Collaboration	Unacceptable Collaboration
Brainstorming solution ideas, e.g., whether to use a reduction for a problem and ideas about what a suitable language for the reduction might look like	Walking through an important piece of the solution step-by-step, sharing solution sketches or drafts, or otherwise directly giving away your solution to someone, whether written or verbally
Helping others understand the problem statement and nuances of the problem	Providing your solution as a reference

⋮ (see syllabus)

Grading

Assignment	Weight
Homework assignments	40%
Midterm exam	29%
Final exam	30-31%
Course evaluations	0-1%
Total	100%

Total Weighted Score	Letter Grade
≥55%	C or better
≥76%	B or better
≥93%	A

Additionally, you need at least 45% on exams to pass

Now onto the course content…

Topic 1: Algorithms

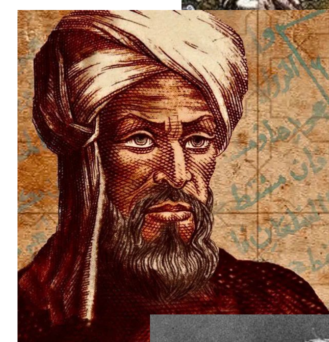
+

Some historical events

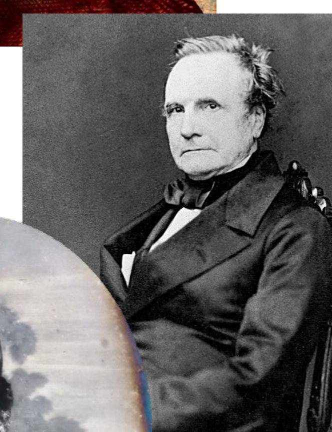
TODAY

300BC: Euclid describes algorithms (for problems such as Greatest Common Divisor) that are still used today

825: Muhammad ibn Musa al-Khwarizmi writes “On the Calculation with Hindu Numerals.” The word “algorithm” will be named for al-Khwarizmi.



1837: Charles Babbage describes plans for Analytical Engine (mechanical general-purpose computer, never completed)



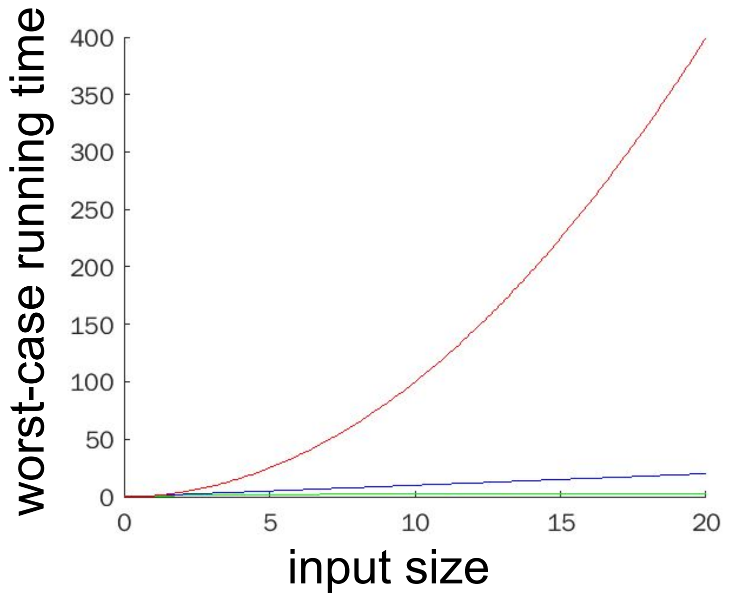
1842: In her notes on the Analytical Engine, Ada Lovelace writes the first algorithm specifically tailored for implementation on a computer (it calculated Bernoulli numbers)

(more history here: scottaaronson.blog/?p=524)

Review: Running Time

- We measure the efficiency of an algorithm by how its **worst-case running time** scales with the **input size**

- We express this asymptotically using Big-O notation: e.g., O(log n), O(n), O(n^2) etc, where n is the **input size**.



- Common interpretations of input size:
  - size of array = # elements
  - size of graph = # vertices + # edges
  - size of integer = # **digits** = O(log(magnitude of integer))
  - **Rule of thumb:** size = # bits to represent input

"Efficient": running time <u>polynomial</u> in input size

Exponential vs. Polynomial

The λ-O-Matic performs 10^{11} operations/sec

	n=10	n=35	n=60	n=85
n^2	100 < 1 sec	1225 < 1 sec	3600 < 1 sec	7225 < 1 sec
n^3	1000 < 1 sec	43k < 1 sec	216k < 1 sec	614k < 1 sec
2^n	1024 < 1 sec	34×10^9 < 1 sec		

"*Efficient*": running time <u>polynomial</u> in input size

Exponential vs. Polynomial

An experiment to try with a child (or adult):

If I pay you every day for a month, would you rather receive:

1. On the 1st day 1 penny, on the 2nd day 2 pennies, on the 3rd day 4 pennies, on the 4th day 8 pennies, etc. doubling each day

   OR

2. \$100 per day

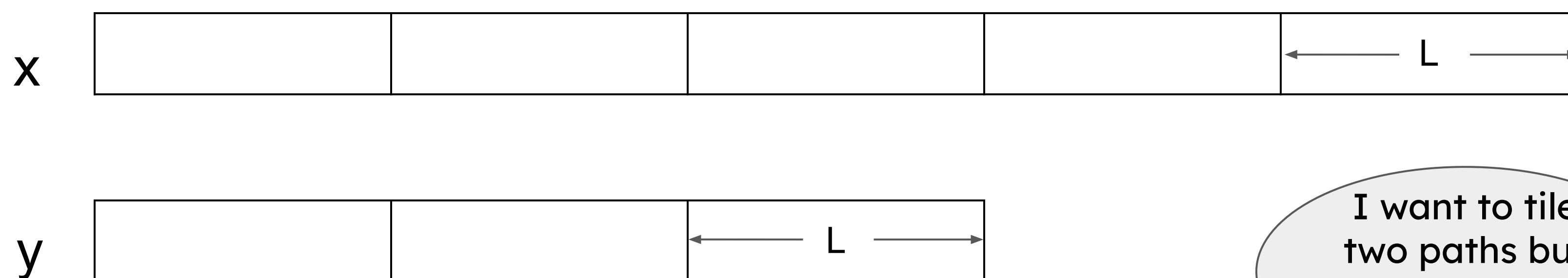
"Efficient": running time polynomial in input size

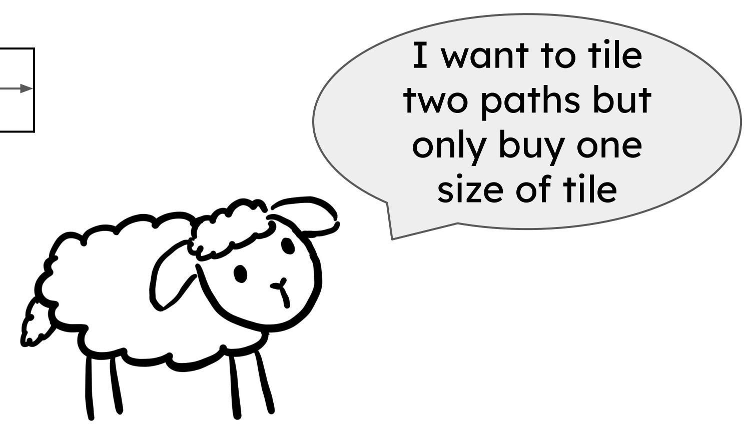
Now back to the oldest algorithm…

The Tiling problem (aka gcd)

Input: n-bit integers x ≥ y ≥ 0, but not both =0.

Output: largest integer L that divides both x and y (aka greatest common divisor)
In other words: largest integer tile size that can exactly tile a path of length x and a path of length y

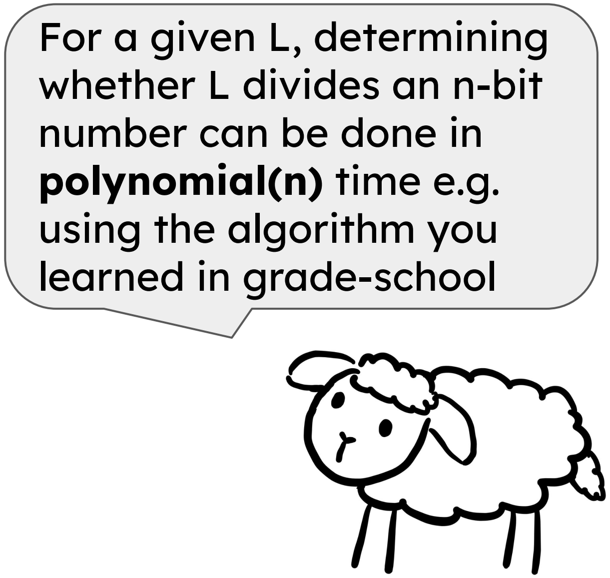




What is the simplest brute force algorithm you can think of?

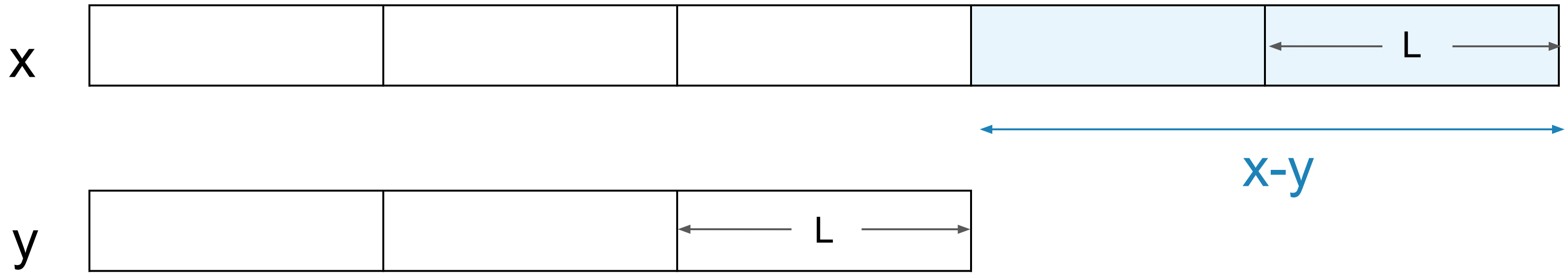
Algorithm:

Worst-case running time in terms of n:



Euclid's idea! (actually it was known before Euclid)

The answer is the same for x, y as it is for x-y, y



Proof.

1. Suppose x-y, y can be L-tiled.
   Why does this mean x,y can be L-tiled?

2. Suppose x, y can be L-tiled.
   Why does this mean x-y,y can be L-tiled?

Euclid's idea! (actually it was known before Euclid)

The answer is the same for:

- x, y
- x-y, y
- x-2y, y
- x-ky, y for any k such that $x-ky \geq 0$
- x mod y, y

Now we have a smaller instance of the same problem.
What should we do?

Euclid's Algorithm

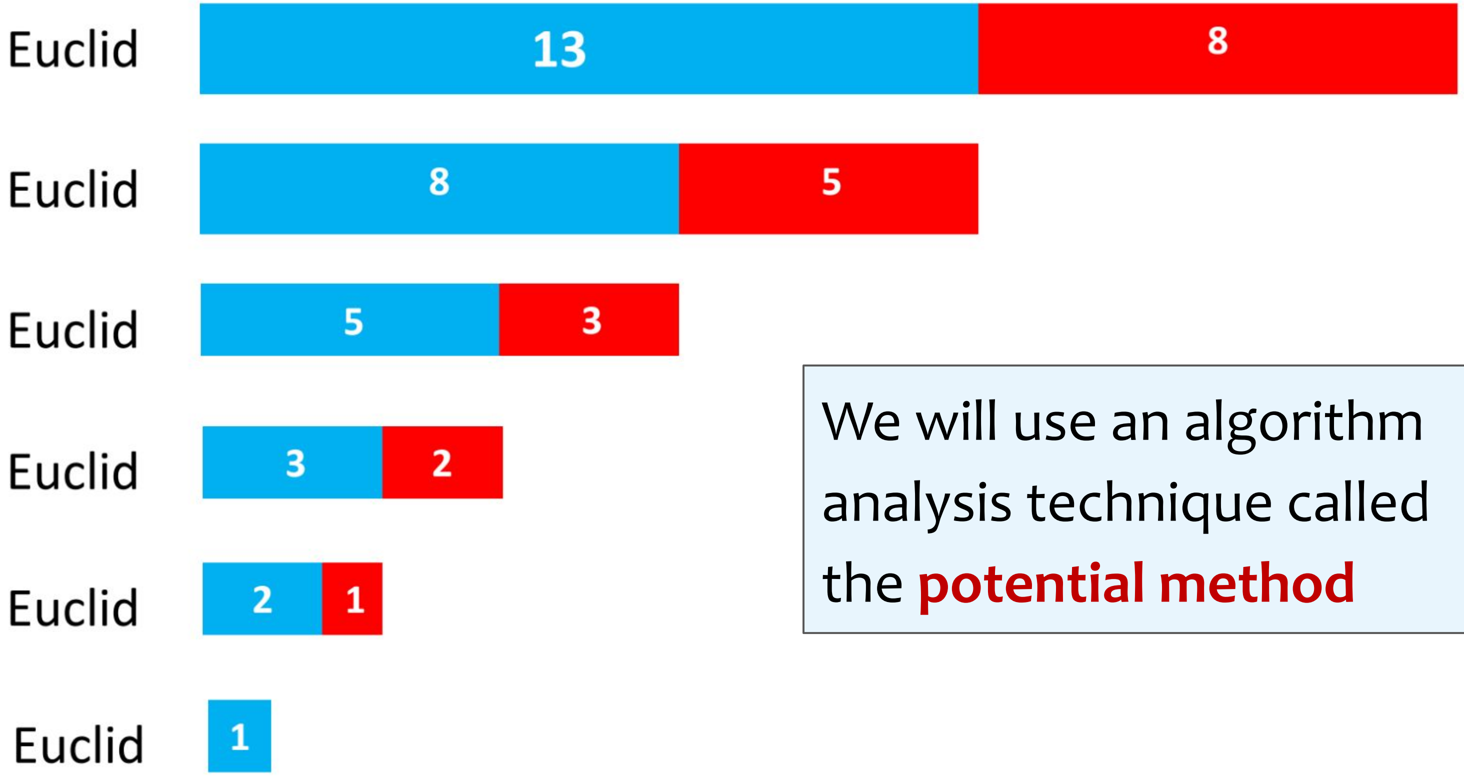
Euclid(x,y): // for integers $x \geq y \geq 0$

Base case: If $y = 0$, **return** __

Recursive case: Else **return** Euclid(   )

Next time: Analyzing Euclid's Algorithm

We will show that, in each recursive call to Euclid, the x, y arguments are *collectively* decreasing *very quickly*



We will use an algorithm analysis technique called the **potential method**