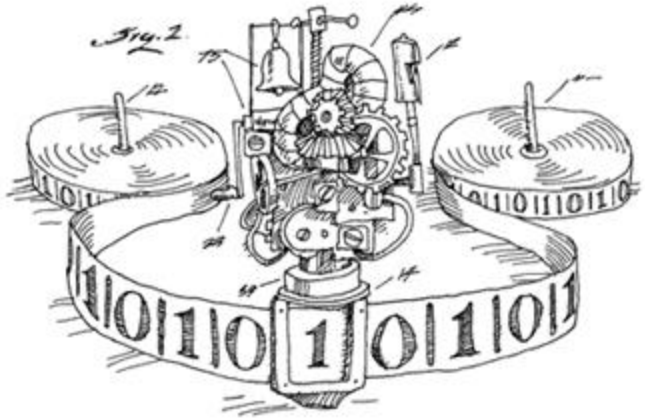
EECS 376: Foundations of Computer Science

Lecture 01 - Introduction



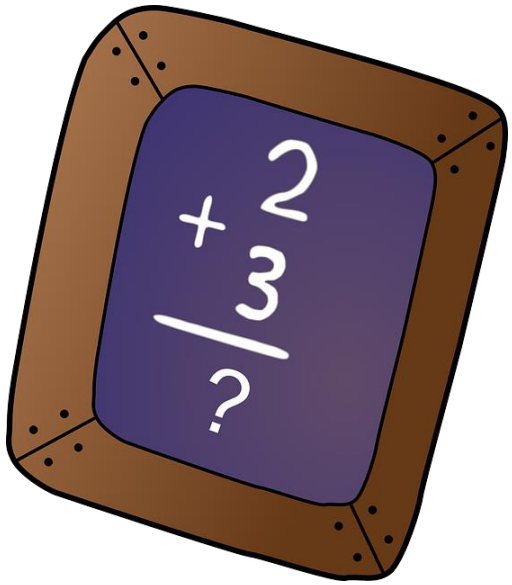
Course Staff (Tentative)

- Instructor: Ali Movaghar (movaghar@umich.edu)
- GSI: Junghwan Kim (kimjhj@umich.edu)
- IA: Eric Khiu (erickhiu@umich.edu)
- **To be updated shortly.**

The Instructor

- Ali Movaghar, movaghar@umich.edu
  - B.S. in Electrical Engineering (EE), University of Tehran, 1977
  - M.S. and Ph.D. in Computer, Information, and Control Engineering (CICE), The University of Michigan, 1979 and 1985
  - Professor at Sharif University of Technology, 1994 -
  - Visiting Professor at The University of Michigan, 2023 -

What is this class about (practically)?



Given a **"problem"**, can I write a **"program"** to solve it on a **"computer"**?

How **"efficient"** is my program? Can I make it more efficient?

What **"algorithmic techniques"** could I try?

How do I *present/explain my program to other people* and *convince them* it's **"correct"**?

Topics

Algorithms

Goal: Pass the coding interview

divide and conquer, DP, greedy

Computability

Goal: See limits of "computation"

Turing machines, undecidability

Complexity

Goal: Recognize "hard" problems

P vs NP, NP-Completeness

Advanced Algorithms

Goal: See what's "out there"

Approximate/randomized algorithms, cryptography

At its heart, this course is about "ideas",
the **art** of computer science.

Course Outline

- Algorithm Design & Analysis (7 lectures)
- Computability theory (5 lectures)
- Complexity theory (6 lectures)
- Randomized algorithms (3 lectures)
- Cryptography (2 lectures)
- Review (1 lecture)
- **Total:** 23 lectures + review

Administration

- **Website:** eecs376.org (Syllabus, Schedule, 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, discussion material, OHs,…
- **Piazza:** questions (private post if sensitive)
- **Gradescope:** exam and HW submission
- **My office hours (student hours):**
  - Tuesday 3:00-5:00 pm (Dow 1031)
- **All office hours:** will be announced soon

Administration

- 5 weekly HW assignments, due Thursdays at 8 pm (Eastern)
  - **late submissions until 9:59 pm with a 5% penalty.**
  - The lowest score will be dropped
  - Solutions published shortly after the deadline

- **Midterm:** Thursday, May 30, 2024, 7-9 pm
- **Final:** Monday, June 24, 7-9 pm

Now onto the course content…

Topic 1: Algorithms

Historical events in antiquity and medieval period

TODAY

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

825: Muhammad ibn Musa al-Khwarizmi, often simply referred to as Al-Khwarizmi, was a Persian polymath who made significant contributions to the development of algebra and arithmetic during the Islamic Golden Age.

He is best known for his works that introduced the Hindu-Arabic numeral system to the Arab world and later to Europe.

In the 12th century, Latin-language translations of al-Khwarizmi's textbook on Indian arithmetic (*Algorithmo de Numero Indorum*), which codified the various Indian numerals, introduced the decimal-based positional number system to the Western world.

Likewise, *Al-Jabr*, translated into Latin by the English scholar Robert of Chester in 1145, was used until the 16th century as the principal mathematical textbook of European universities.

Historical events in Early Middle Ages

- Various mathematicians developed the algorithms for basic arithmetic operations within the decimal-based positional number system over time across different cultures.
- However, a significant milestone in this development is attributed to Al-Khwarizmi.
- His name gave rise to the English terms *algorism* and *algorithm*; the Spanish, Italian, and Portuguese terms *algoritmo*; and the Spanish term *guarismo* and Portuguese term *algarismo*, both meaning "digit".

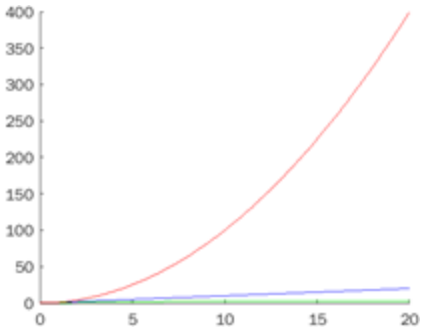
Roman numerals

Roman numerals are a numeral system that originated in ancient Rome and remained the usual way of writing numbers throughout Europe well into the Late Middle Ages.

Individual decimal places				
	Thousands	Hundreds	Tens	Units
1	M	C	X	I
2	MM	CC	XX	II
3	MMM	CCC	XXX	III
4		CD	XL	IV
5		D	L	V
6		DC	LX	VI
7		DCC	LXX	VII
8		DCCC	LXXX	VIII
9		CM	XC	IX

Review: Running Time

- Measure of efficiency of algorithms:
  how **worst-case running time** scales with **input size**



- We express this asymptotically using Big-O notation:
  e.g., O(log n), O(n), O(n^2) 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(size of integer))
  - **Rule of thumb:** size = # bits to represent input

"Efficient": running time polynomial in input size

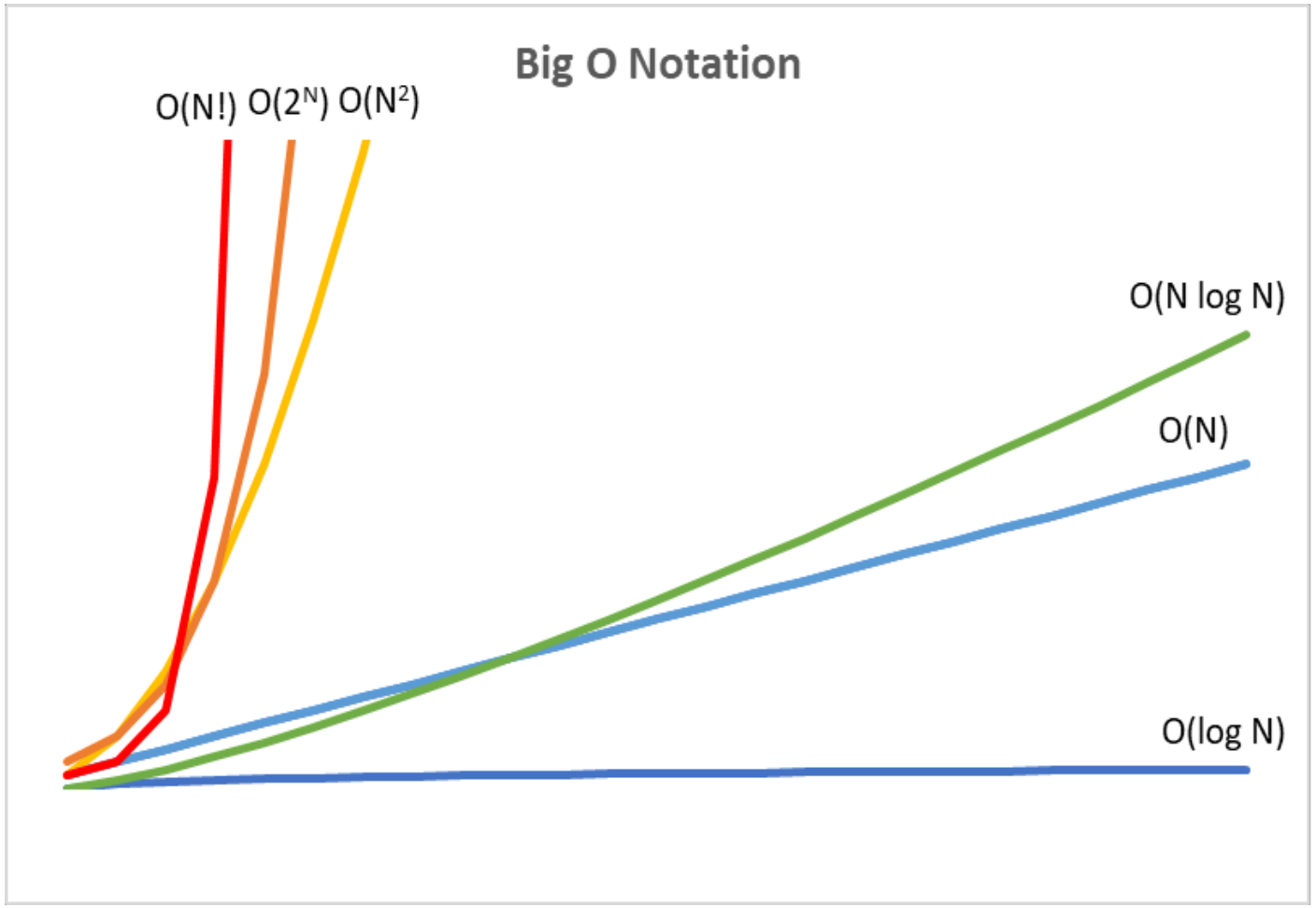
Big O Notation

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\,g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

Examples: Big O Notation



Exponential vs. Polynomial

	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 polynomial 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.
   OR

2. $100 per day

1. 2^30 -1 pennies = $10,737,418.23 2. $3000

"Efficient": running time polynomial in input size

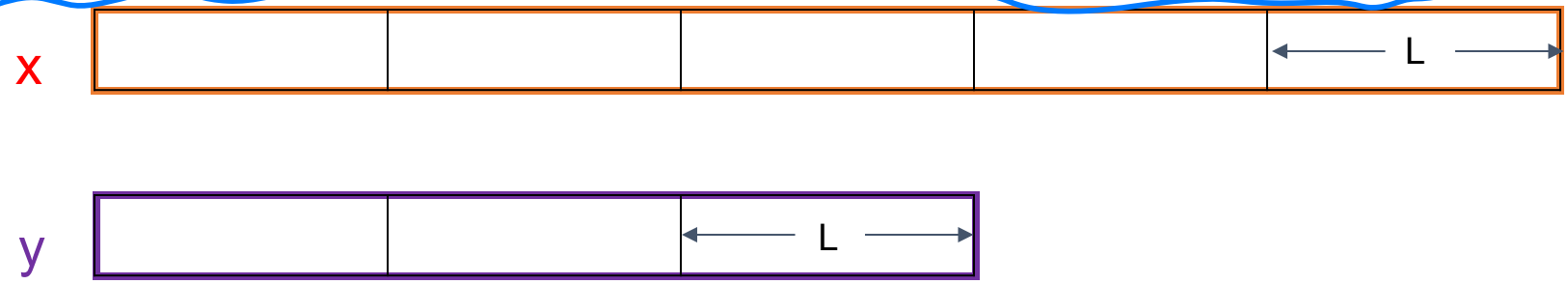
Now back to the oldest algorithm…

The Tiling problem (aka gcd)

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

Output: greatest common divisor **gcd(x,y)**
largest integer L that divides both x and y

In other words:
largest integer tile size for tiling **both paths** of length x and y



Example: Compute these

- gcd(25,15) 5
- gcd(12,12) 12
- gcd(10,13) 1
- gcd(10,1) 1
- gcd(10,0) 10

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

Algorithm:

Naïve(x, y):
1: **for** $\ell = y, y-1, \ldots, 1$:
2: **if** ℓ divides x *and* y:
3: **return** ℓ

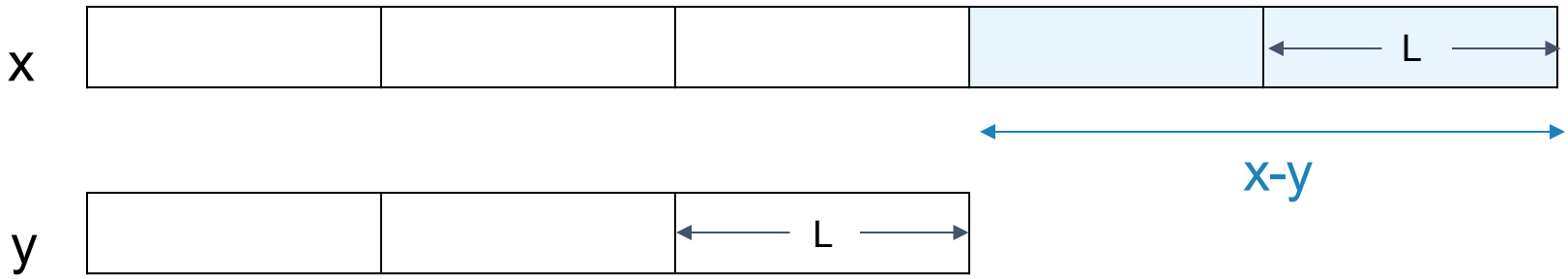
2 mod operations every time

Worst-case running time in terms of n:

$O(y)$ (linear)

Observation: We can reduce the size of problem

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



Lemma 5: $\forall m \in \mathbb{Z}$, $(x,y) = (x-my, y)$

Reason: (x,y) can be L-tiled ⇔ (x-y,y) can be L-tiled

- If L divides x and y, then L divides x-y (and y, of course)
- If L divides x – y and y, then L divides x (and y, of course)

⇒ 取 $m = \lfloor \frac{x}{y} \rfloor$

则 $x - my =$ x mod y

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 ≥ 0
- x mod y, y

Now we have a smaller instance of the same problem.

What should we do?

Corollary: $(x, y) = (x \bmod y, y)$

Euclid's Algorithm

Euclid(x,y): // for integers x ≥ y ≥ 0

Base case: If y = 0, **return** x

Recursive case: Else **return** Euclid(y, x mod y)

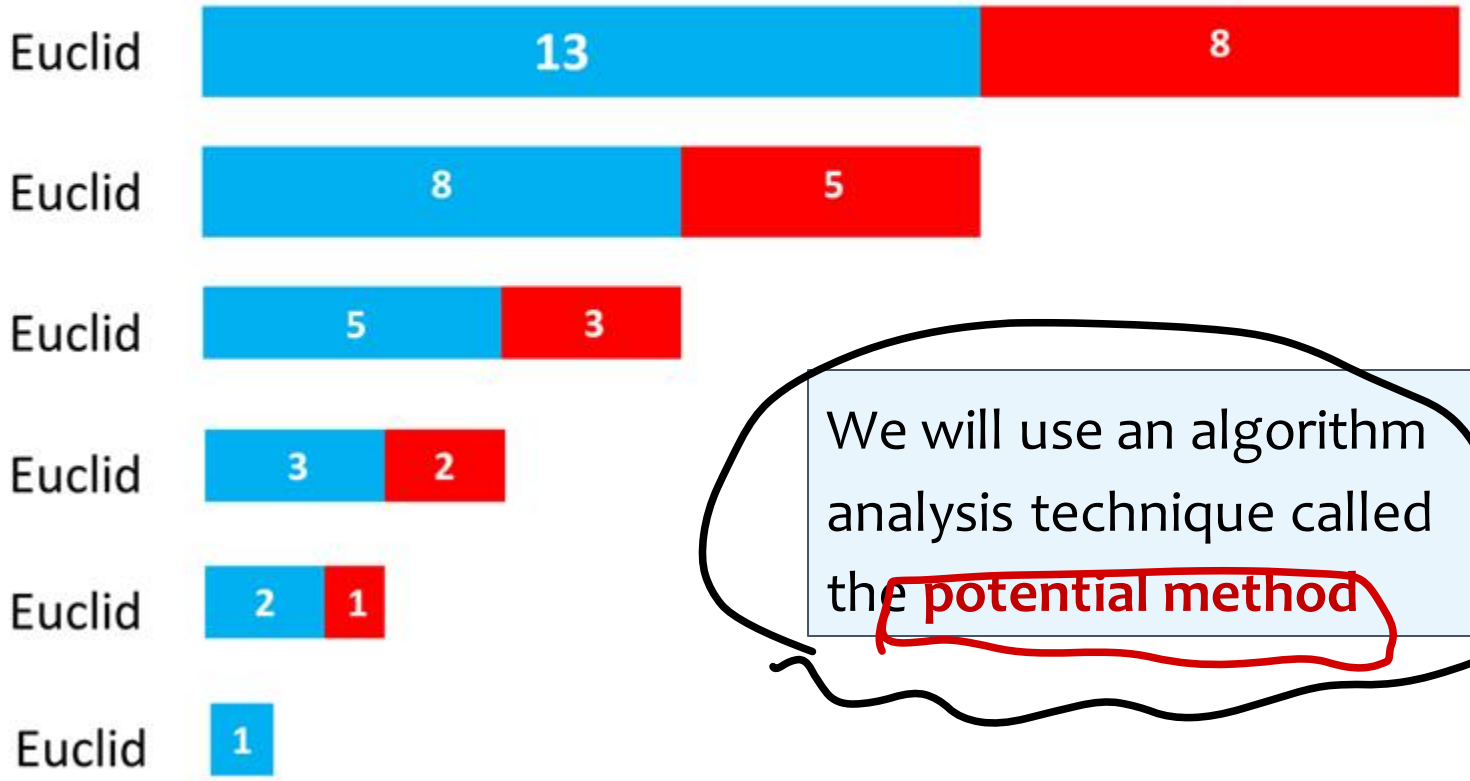
(since ∀x, (x,0) = x)

Next time: when x,y have n digits

- We will show this algo runs in $O(n)$ time
- instead of $2^{O(n)}$

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**