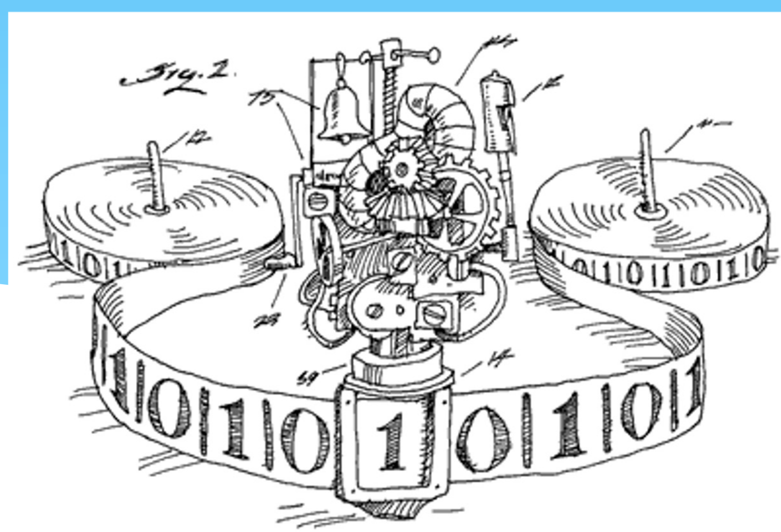
EECS 376: Foundations of Computer Science

Nikhil Bansal

Michal Derezinski

Ben Fish

Seth Pettie



Today’s Agenda

* Introduction
* Big questions/Outline
* Administration
* Algorithm Design and Analysis
  * Potential Function arguments

Introduction

* I'm Seth
* I've been a Professor at Michigan since 2006.
    * Visiting prof. at various places: Germany, Denmark, Israel, China.

* I teach math, theory, and algorithms courses (EECS 203, 376, 477, 586, 598 seminars, etc.)

* **Research:** Combinatorics/discrete math, data structures, distributed computing, statistics, graph algorithms.

Why are we here?

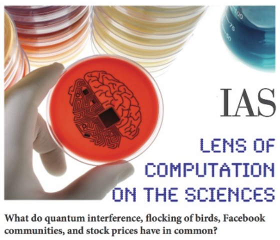
Computer science is no more about computers than astronomy is about telescopes. --- Edsger Dijkstra

Foundations: What is computation? (just number-crunching?)

- Is every problem solvable on a computer?
- Can every solvable problem be solved quickly?
- Are there general algorithmic techniques?
- …

Why is this useful to me?

Computational Thinking

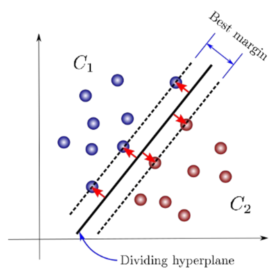


Natural processes

Computational Biology

Algorithmic Finance

Robotics



Machine learning
Big Data

Quantum Computation

Outline

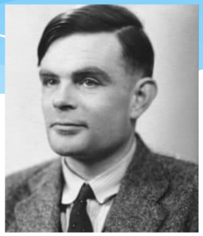
Design and Analysis of Algorithms	Mon 28 Aug	1	Introduction, The Potential Method
	Wed 30 Aug	2	Divide and Conquer 1
	Discussion	*1*	*Review: Proofs, Asymptotic Notation, Induction*
	Mon 4 Sep		***No class — Labor Day***
	Wed 6 Sep	3	Divide and Conquer 2
	Discussion	*2*	*Divide and Conquer*
	Mon 11 Sep	4	Dynamic Programming
	Wed 13 Sep	5	Dynamic Programming 2
	Discussion	*3*	*Dynamic Programming*
	Mon 18 Sep	6	Greedy Algorithms
Computability	Wed 20 Sep	7	Formal Languages and Finite Automata
	Discussion	*4*	*Greedy Algorithms and Finite Automata*
	Mon 25 Sep	8	Turing Machines and Decidability
	Wed 27 Sep	9	Diagonalization
	Discussion	*5*	*Turing Machines and Diagonalization*
	Mon 2 Oct	10	The Acceptance and Halting Problems
	Wed 4 Oct	11	Reducibility
	Discussion	*6*	*Acceptance and Reducibility*
	Mon 9 Oct	12	Rice's Theorem & Kolmogorov Complexity
Midterm	Wed 11 Oct	13	*Midterm Review*
	Discussion	*MR*	*Midterm Review*
	Mon 16 Oct		***No class — Fall Break***
	Wed 18 Oct		***Midterm, 7-9pm ET***
	Discussion		***No discussion***

Outline

Complexity	Mon 23 Oct	14	The Classes P and NP
	Wed 25 Oct	15	The Cook-Levin Theorem
	Discussion	7	*NP Overview*
	Mon 30 Oct	16	Reductions and NP-Completeness
	Wed 1 Nov	17	NP-Complete Problems 2
	Discussion	*8*	*NP-Completeness*
	Mon 6 Nov	18	Search and Approximation Algorithms
	Wed 8 Nov	19	Approximation Algorithms 2
	Discussion	*9*	*Search and Approximation*
Randomness in Computation	Mon 13 Nov	20	Probability, Randomness in Computation
	Wed 15 Nov	21	Randomness in Computation 2
	Discussion	*10*	*Randomness and Modular Arithmetic Review*
	Mon 20 Nov	22	Randomness in Computation 3
Thanksgiving	Wed 22 Nov	23	***No Class — Thanksgiving Break***
	Discussion	*11*	***No Class — Thanksgiving Break***
Cryptography	Mon 27 Nov	24	One-time Pad, Diffie-Hellman, and Discrete Logarithm
	Wed 29 Nov	25	RSA and Factoring
	Discussion	*12*	*Intro to Cryptography*
Special Topics	Mon 4 Dec	26	Special Topcs (Untested Material)
	Wed 6 Dec	27	Special Topcs (Untested Material)
Final Exam			
	Tue 12 Dec		**Final Exam, 7–9pm**

Computability

Turing(1936): Formal model of computer (Turing machine)



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

E.g. Can one write a program that tests if two given programs in C/C++ have the same functionality? Answer: No

Will see: Various other basic problems are "uncomputable"

Complexity

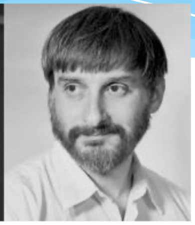
* ***Computability***: what can/cannot be computed (in finite time)?
* Finite time is not good enough; we need to solve <u>fast</u>!

* Example 1: Sorting n numbers
  * Naïve algorithm: *try all* n! *permutations*.
  * Any better solutions?

* Example 2: Travelling salesperson problem.
  * Find shortest tour that visits all n cities
  * Naïve algorithm: *try all* n! *permutations*.

* Is there a ***polynomial-time algorithm***? $(O(n^2), O(n^3), \ldots)$

Complexity


Cook Levin Karp

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

No efficient algorithm for TSP *unless* P=NP

One of the biggest questions of our time (is P=NP?) Million dollar prize

We will explore P vs NP in detail later.

Approximation Algorithms

Most problems turn out to be NP-hard (no efficient way to solve exactly)

But can try to solve approximately (99% solution may be good enough)

EECS ECE CSE

Hardness is Bad, Right?

* **Question:** Suppose you know that a certain problem is notoriously "hard". Is it necessarily a bad thing?

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 code in advance

Digital signatures

…

Administration

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

* **Canvas:** HWs, lecture slides, discussion material, OHs, etc..
* **Piazza:** questions about homework, lectures; search for teammates
* **Gradescope:** for exams and HW submission

* *You may attend any lecture/discussion*
* **Discussion:** *highly encouraged* to attend

* **Note:** No lecture recordings

  (recordings discourage active learning)

Administration

* 12 weekly HW assignments, due Wednesdays 8pm (Eastern)
  * **No Late Submissions after 9:59pm!**
  * However, **two lowest scores** will be dropped
  * Solutions published shortly after the deadline
  * HW = 40% of grade

* **Midterm:** October 18, 2023, 7-9pm. 29% of grade.
* **Final:** December 12, 2023, 7-9pm. 30% of grade.
* **Course Evaluation.** 1% of grade.

* Participation is important!
  * Questions are welcome!
  * There is no such thing as a "bad question".

Is this an EECS class?

* **Question to the Instructor:** Wolverine Access says it is an EECS class. Yet why does it feel like a math class?

* **Answer:** It is both. The only way to answer the questions we raised (and others) is to construct a ***computational model*** and apply a "proof-based" (*mathematical*) methodology.

Is this an EEECS class?

* **Example:** Show that there is no compiler that tests if two given programs in C/C++ have the same functionality.

* **Wrong Approach:** Try all the compilers… (infinitely many)

* **Right Answer:** Construct a ***computational model*** that captures all the compilers and give a "general impossibility argument".

EECS M ECE CSE

Next 3.5 weeks: 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 (e.g. time, memory) necessary to execute them.
* **Examples:** Potential function arguments, recurrences, Master Theorem, exchange arguments
* **Remark:** We describe algorithms in "Pseudo-Code" (human readable)

Greatest Common Divisor

* **Definition:** Let $x, y \in \mathbb{N}$ (natural numbers). The Greatest Common Divisor (gcd) of x and y is the largest $z \in \mathbb{N}$ that divides x and y.
  * If gcd(x,y) = 1 then x and y are called **coprime**.

* **Examples:** $gcd(21,9) = ?$ $gcd(121,5) = ?$
* $\text{gcd}(343473598722323,432029908279) = ?$
  * (gcd is a basic building block for cryptographic algorithms, typically > 100 digits)

* **Naïve Algorithm:**
  * For z from y down to 1:
    * If $((z|y) \wedge (z|x))$, return z.

* **Runtime:** $\boldsymbol{O(y)}$ operations. Is that good or bad?

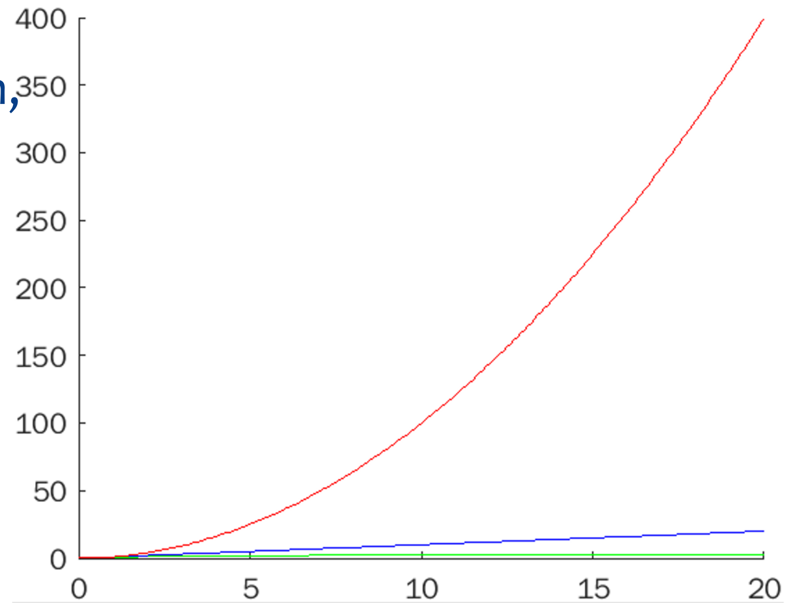
Review: Running Time

* We measure the "efficiency" of an algorithm by how its ***worst case*** runtime scales with the input size.

* We express this trade-off in asymptotic notation, e.g. $O(\log n)$, $O(n)$, $O(n^2)$, $etc.$, where n is the size of the input.

* Common interpretations of "size"
  * Size of an array: #array cells
  * Size of a graph: #vertices + #edges
  * Size of an integer: #digits.

  * Rule of thumb: ≈ # bits needed to represent input on a computer.



Efficient ≡ "runtime is polynomial in size of input"

Step 2: Analyze runtime of the naïve solution

* Q: Suppose x and y each have n digits. How large can y be?
  * $10^n - 1$

* Note: the "size" of y is $\log y$.

* The runtime of the naïve algorithm is $O(y)$, which is ***exponential*** in the size of the input: $n = \log y$. *(This is not efficient!)*

```
NaiveGCD(x, y)
For z = y, y − 1, ..., 1
  If (z|x) ∧ (z|y) then
     Return(z)
```

EECS M ECE CSE

Step 3: Think about the "structure" of the problem.

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

Proof: If d divides both x and y, d also divides $x - y$.
Conversely, any d that divides both $x - y$ and y also divides x.

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 k times until $x - ky \; < \; y$.

* Q: What is $x - ky$?
  * $x \bmod y$ = the remainder of x divided by y.

* Thereom: $\gcd(x, y) = \gcd(y, x \bmod y)$

Step 4: Code it up

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

Euclid(x, y): // for $x \geq y > 0$
if $x \bmod y = 0$: return y
return **Euclid**$(y, x \bmod y)$

Let's do some examples:
gcd (21,9),
gcd(13,8)
gcd(42273,9516)
[Calculator]

Euclid, 300 BCE

Seems fast

Calculator

42273, 9516
-> 9516, 4209
-> 4209, 1098
-> 1098, 915
-> 915, 183
-> 183,0 (gcd = 183)

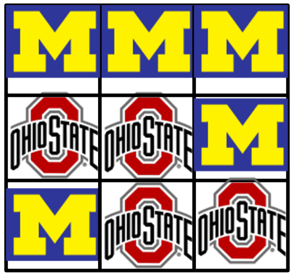
Euclid(x, y): // for $x \geq y > 0$
 if($x \bmod y = 0$), return y.
 else return **Euclid**$(y, x \bmod y)$

Euclid, 300 BCE

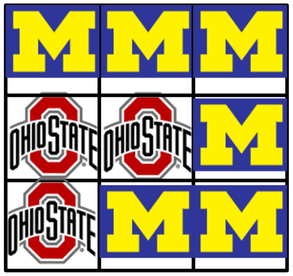
* "How can be bound the runtime of Euclid?"
* We need some tools…

Example: A Flipping Game

* 3 x 3 board covered with two-sided chips:
* Two players, R and C, alternately perform "flips"
  * R-flip (C-flip): flip a row (col) with # > #
* If no flip is possible, then the game ends.
* **Q:** Does the game always end?



R-flip (3)



C-flip (1)

Analysis Tool: Potential Function Argument

* Intuitively, a ***potential function argument*** says that:
* If I start with a ***finite*** amount of water in a ***leaky bucket***, then eventually water stops leaking out.

* 3 main ingredients of the argument:
  * Discrete units of *time* $t = 0,1,2,3,\ldots.$ (Loop iteration, recursion depth, etc.)
  * Measure how much water is in the bucket. Map to an integer $s_t \geq 0$.
  * How is the bucket leaking? Show $s_{t+1} < s_t$.

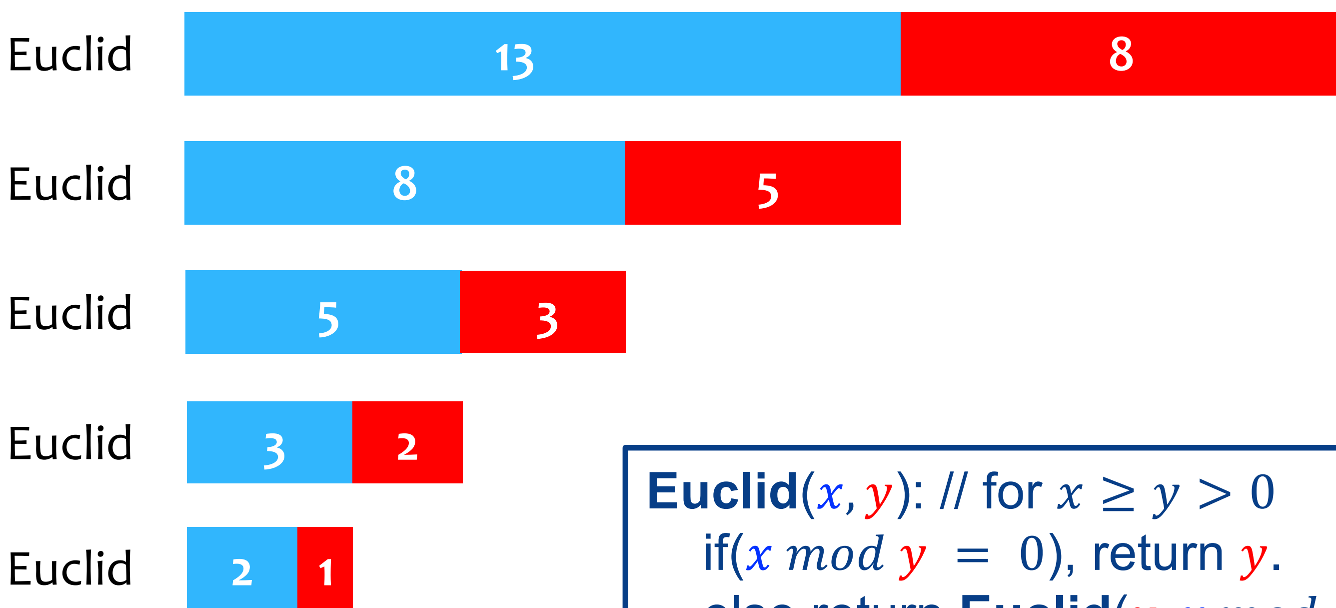
Example: A Flipping Game (As a Potential Function Argument)

* Let $s_t \geq 0$ be the number of Ohio chips after t flips.
  * Unit of time: 1 flip
  * Amount of water in the bucket: $s_t = \#$ Ohio chips.
  * **Claim:** Each flip decreases the number of Ohio chips, i.e., $s_{t+1} < s_t$.

* The game must end! There are no infinitely decreasing *integer* sequences $s_0 > s_1 > s_2 > \cdots > 0$.

Back to Euclid: Idea of analysis

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



Euclid(x, y): // for $x \geq y > 0$
  if($x \bmod y\ = 0$), return y.
  else return **Euclid**($y, x \bmod y$)

A good potential function

* The ***sum*** of the arguments to **Euclid** decreases quite rapidly.

Euclid(x, y): // for $x \geq y > 0$
  if($x \bmod y = 0$), return y.
  else return **Euclid**$(y, x \bmod y)$

* *Example*: $\text{Euclid}(13,8) \rightarrow \text{Euclid}(8,5) \rightarrow \text{Euclid}(5,3) \rightarrow \text{Euclid}(3,2) \rightarrow \text{Euclid}(2,1) = 1$

* Define x_t, y_t to be the arguments to the tth call to **Euclid**, where $x_t \geq y_t$.
* Define the ***potential*** to be $s_t = x_t + y_t$.

* **Claim.** $s_{t+1} < \frac{2}{3} s_t$.

A good potential function

Euclid(x, y): // for $x \geq y > 0$
  if($x \bmod y = 0$), return y.
  else return **Euclid**$(y, x \bmod y)$

* **Claim.** $s_{t+1} < \frac{2}{3} s_t$.
* Proof. Write $x_t = k_t y_t + r_t$, where $k_t \geq 1, r_t < y_t$
  * What is $x_{t+1} = ?$ $\boxed{y_t}$
  * What is $y_{t+1} = ?$ $\boxed{r_t}$
* $s_t = x_t + y_t = k_t y_t + r_t + y_t \geq 2y_t + r_t$
* $> 2y_t + r_t - \frac{y_t - r_t}{2} = \frac{3}{2}(y_t + r_t) = \frac{3}{2} s_{t+1}$.

A good potential function

Euclid(x, y): // for $x \geq y > 0$
  if$(x \bmod y = 0)$, return y.
  else return **Euclid**$(y, x \bmod y)$

* **Claim.** $s_{t+1} < \frac{2}{3} s_t$.

* Thus, if there are t calls to **Euclid,** $2 \leq s_t < \left(\frac{2}{3}\right)^t (x + y)$
* Which implies that $t < \log_{3/2}((x + y)/2)$

* I.e., $t = O(n)$, where $n = \log x + \log y$ is the size of the input.

Sprouts Game

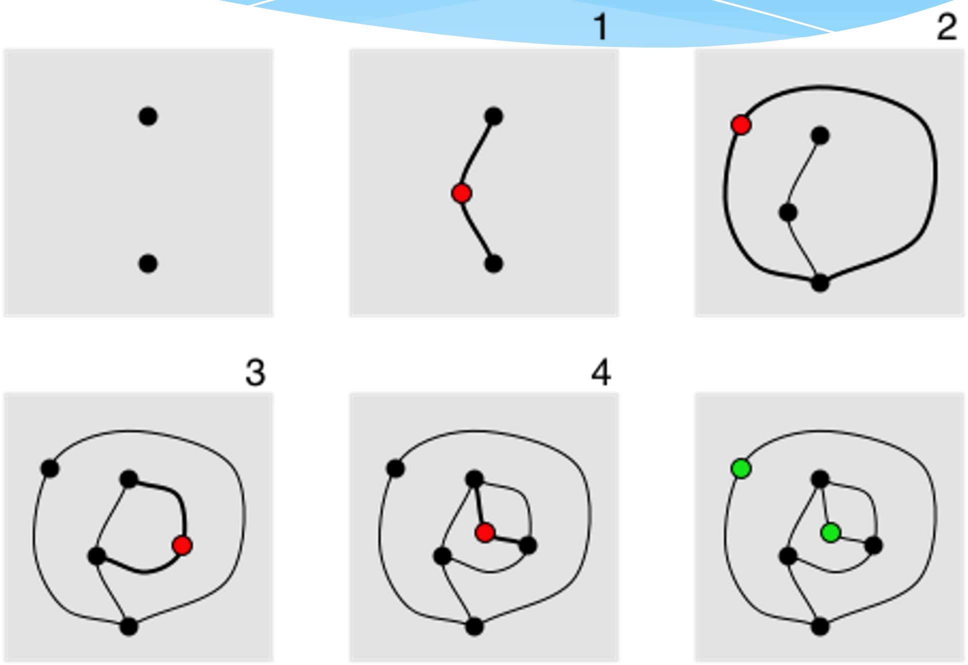
There are n points initially.

Each step: Connect any two points by a line (or curve) and introduce a new point on the line.

Rule: Each point can have at most 3 lines attached to it

(i.e. degree of point at most 3)

Given n points, can the game continue forever? If not, how many steps to terminate?



Sprouts

Let’s do a couple of more examples.

What’s a good potential? (something that decreases over time)

Hint: Consider modified game where we add two new points on the line.

Can this go on forever?

Can this tell us something about our original game?

Potential Function: Sprouts

At time t, let us call the deficit of a point p as
3 - # of lines attached to p

Note: By rules of the game, the deficit of each point at any time is >=0.

Consider potential s_t = sum of deficits of all points at time t.
Initially: $s_0 = 3n$

Claim: For each time step t >= 1, we have $s_t = s_{t-1} - 1$

Proof: Let us see what happens at time t.
(i) adding the new line decreases deficit by 1 for each of its two end points.
(ii) the new added point has deficit 3-2 = 1 (the new point has two attachments)

So overall change in potential $s_t - s_{t-1}$ = -2 + 1 = -1.

Theorem: The game stops after at most 3n-1 steps.
Proof: Initial potential s_0 = 3n.
Game must stop when deficit reaches 1 (as no two points available to connect)