(Computational) Complexity:
In every day life?

Madhu Sudan
MIT

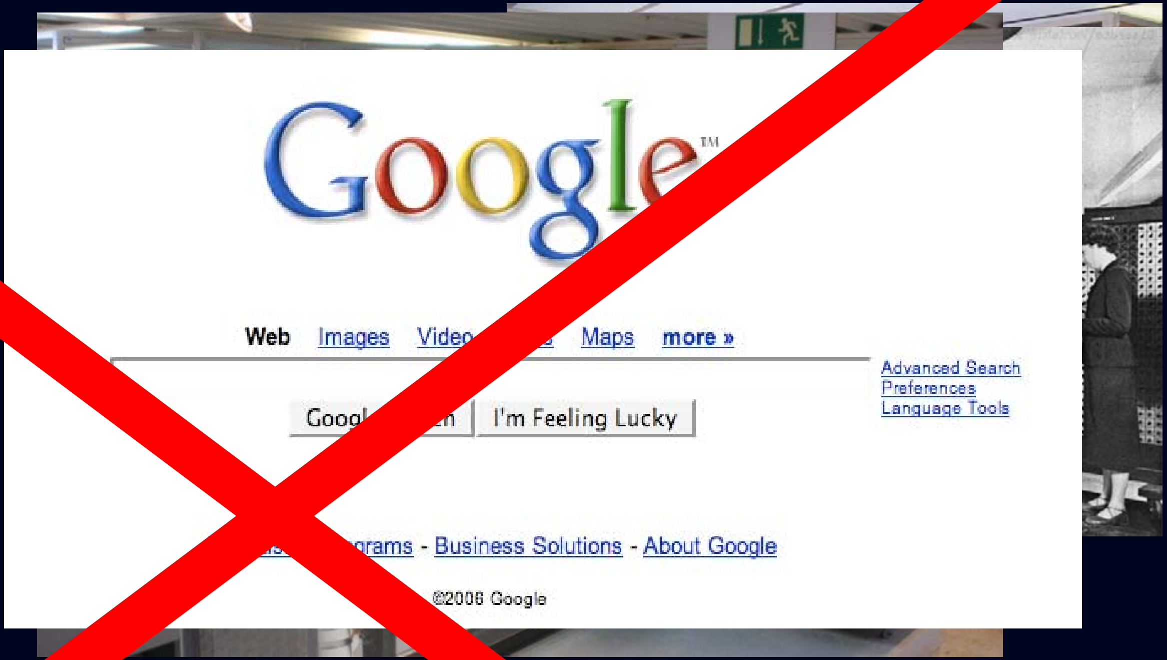
Theory of Computing?

- Part I: Origins: Computers and Theory
- Part II: Modern Complexity
- Part III: Implications to everyday life.
- Part IV: Future of computing

Origins of Computation

History of Computing

- Born: 1941?

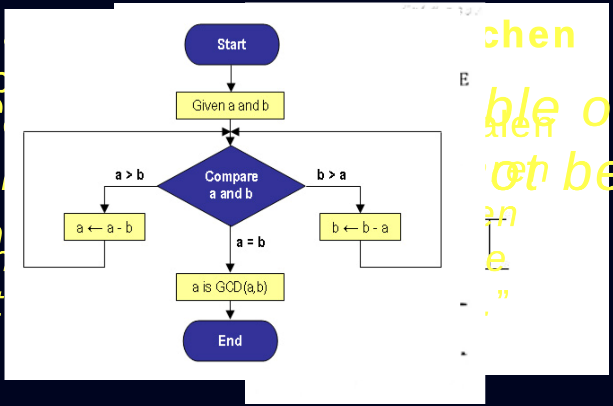


- 1946:
- 1950s-2000: Mainframes, PCs, Internet, WWW
- Died: 2005?

Tracing Computing Backwards

“Entscheidung der L�sb... **chen Gleichung.** Eine diop... irgendwelchen Unbekannt... ...ahlen Zahlkoefficienten sei vorg... angeben, nach welchen ... Anzahl von Operationen ... *Gleichung in ganzen rat*...

“Any effectively gene... ...ble of expressing elementa... ...ot be both consisten...”



- Turing (1936): Universal Computer (Model)
- Gödel (1931): Logical predecessor.
- Hilbert (1900): Motivating questions/program.
- Gauss (1801): Efficient factoring of integers?
- Euclid (-300): Computation of common divisors!
- Prehistoric!! (adding, subtracting, multiplying, thinking (at least logically) are all computing!)

Tracing Computing Forwards

"Rumors of its demise are greatly exaggerated …"

… More later.

Computation and Complexity

Example: Integer Addition

- Addition: Suppose you want to add two ten-digit numbers. Does this take about 10 steps? Or about 10 x 10 steps?

$$
\begin{array}{ccccccccccc}
 & & \boxed{1} & \boxed{1} & & \boxed{1} & & & \boxed{1} & \boxed{1} & \\
 & 2 & 3 & 1 & 5 & 6 & 7 & 5 & 6 & 8 & 9 \\
+ & & 5 & 8 & 9 & 1 & 4 & 3 & 2 & 2 & 6 \\
\hline
 & 2 & 9 & 0 & 4 & 8 & 1 & 8 & 9 & 1 & 5
\end{array}
$$

- ~10 steps! Linear time!

Computation!

- What we saw was a computational procedure (algorithm) to add integers.
- In general Algorithm =
  - Sequence of steps
  - Each step very simple (finite + local)
  - Every step of sequence determined by previous steps.
- Formalization:
  - Turing Machine/Computer Program/Computer!

- Moral: Computation is ancient! Eternal!!

First Law of Computation [Turing]

- Universality: There is a single computer which can execute every algorithm.

- Obvious today

  … we all own such a "single computer".

- Highly counterintuitive at the time of Turing.

- Idea made practical by von Neumann.

Example 2: Multiplication

- Multiplication: Suppose you want to multiply two n-digit numbers. Does this take about n steps? Or n x n steps?

```
            2 3 1 5 6 7
      x       5 8 9 1 4
  ___________________________
            9 2 6 2 6 8
          2 3 1 5 6 7
      2 0 8 4 1 0 3
    1 8 5 2 5 3 6
  1 1 5 7 8 3 5
  ___________________________
  1 3 6 4 2 5 3 8 2 3 8
```

- Above process: n^2 steps. Best?

Complexity

- Adding/Multiplying n-digit numbers
- Addition: ~n steps; Multiplication: ~n^2 steps.
- Is addition really easier than multiplication?
- Can we prove multiplying requires n^2 steps ? (Needed to assert addition is easier!)
  - Unfortunately, NO!
  - Why?
    - Answer 1: Proving "every algorithm must be slow" is hard!
    - Answer 2: Statement is incorrect!
      - Better algorithms (running in nearly linear time) exist!

Computation and Complexity

- Broad goal of Computational Research:
  - For each computational task
    - Find best algorithms [Algorithm Design]
    - Prove they are best possible [Complexity]

- Challenges to the field:
  - Algorithms: Can be ingenious
    - (in fact they model ingenuity!)
  - Complexity: Elusive, Misleading

Example: Integer Arithmetic

- Addition: Linear!
- Multiplication: Quadratic! Fastest? Not-linear
- Factoring? Write 13642538238 as product of two integers (each less than 1000000)
- Inverse of multiplication.
  - Not known to be linear/quadratic/cubic.
  - Believed to require exponential time.

Computation and Complexity

- Broad classification of Computational Problems
  - Easy
    - Those that can be solved in polynomial amount of time (n, n^2, n^{10} …)
    - Doubling of resources increases size of largest feasible problem by *multiplicative* factor.
  - Hard
    - Those that require exponential time (2^n, 10^n, …)
    - Doubling of resources increases size of largest feasible problem by *additive* factor.

Computation and Complexity

- Broad classification of problems
  - Easy: Doubling of resources increases size of largest feasible problem by multiplicative factor.
  - Hard: Doubling of resources increases size of largest feasible problem by additive factor.

- Computer Science

  = (Mathematical) Study of Easiness.

  = (Mathematical) Study of Complexity.

Reversibility of Computation?

- Recall: Multiplication vs. Factoring
  - Factoring reverses Multiplication
  - Multiplication Easy
  - Factoring seems Hard

- **P** = Class of Easy Computational Problems.
  - Problem given by function f: input → output.

- **NP** = Reverses of **P** problems.
  - Given function f in **P**, and output, give (any) input such that f(input) = output.

- Open: Is **P=NP?**

Second Law of Computation? [Unproven]

- **Irreversibility Conjecture:** Computation can not be *easily* reversed. (**P ≠ NP**).

- The famed "P = NP?" question
  - Financially Interesting:
    - Clay Institute offers US$ 1.000.000.
  - Mathematically interesting:
    - Models essence of theorems and proofs.
  - Computationally interesting:
    - Captures essential bottlenecks in computing.
  - Interesting to all:
    - Difference between goals and path to goals.

NP-completeness and consequences

Hardest problem in NP

- Even though we don't know if **NP = P**, we know which problems in **NP** may be the hardest. E.g.,
  - Travelling Salesman Problem
  - Integer Programming
  - Finding proofs of theorems
  - Folding protien sequences optimally
  - Computing optimal market strategies
- These problems are **NP-**complete.
  - If any one can be easily solved, then all can be easily solved.

An NP-complete Problem: Divisor?

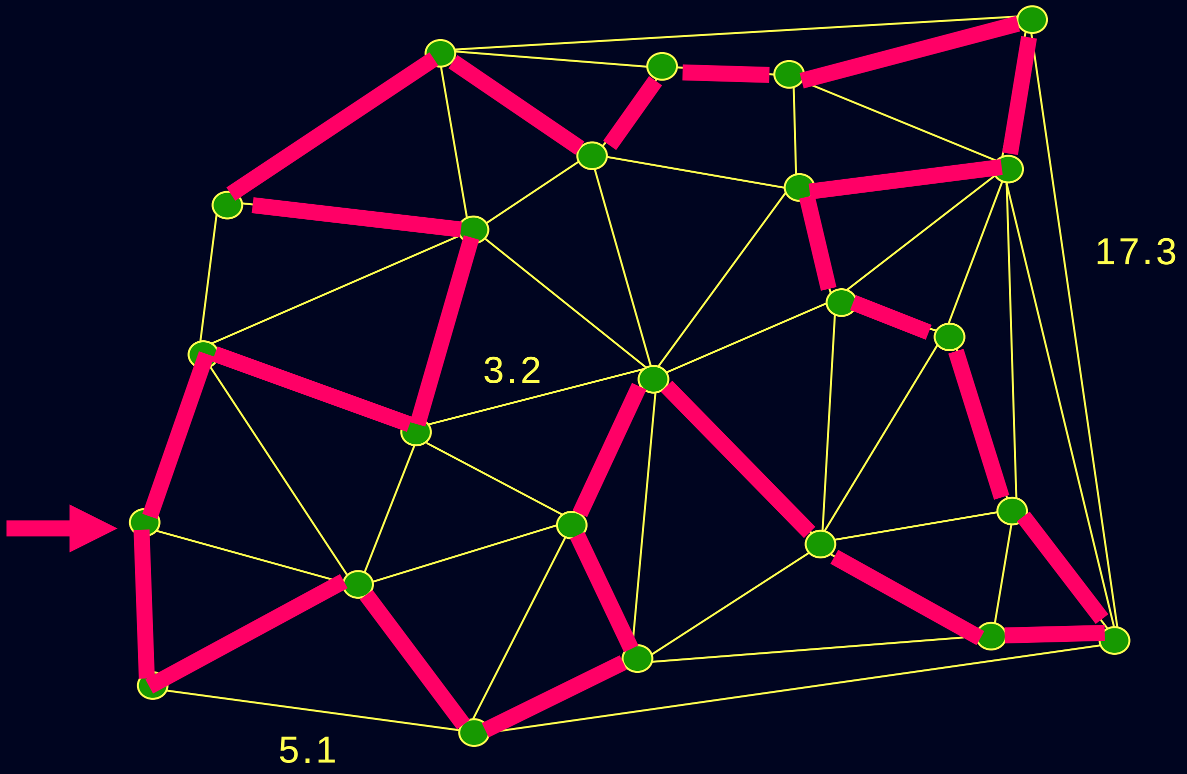
- Given n-digit numbers A, B, C, does A have a divisor between B and C?
  - (Does there exist D such that B < D < C and D divides A?)

Example:

- Q: Does 3190966795047991905432 have a divisor between 25800000000 and 25900000000.
- A: YES
- Proof: 25846840632.

Example 2: Travelling Salesman Problem

- Many cities;
- Want to visit all and return home;
- Can he do it with < 125 hours of driving?



#Hours so far

12[illegible]

Easy to verify if answer is YES.

Can you prove if answer is NO?

Theorems and Proofs

- 1900-2000: Mathematical formalization of Logic
  - [Hilbert, Gödel, Church, Turing ...]
  - Logic = Axioms + Deduction Rules
  - Theorem, Proofs: Sentences over some alphabet.
    - Theorem: Valid if it follows from axioms and deduction rules.
    - Proof: Specifies axioms used and order of application of deduction rules.
- Computational abstraction:
    - (Theorem,Proof) easy to verify.
    - Finding a proof for proposed theorem is hard.
- Theorem: Finding short proofs is **NP**-complete.

Theorems: Deep and Shallow

- A Deep Theorem: $\forall x, y, z \in \mathbb{Z}^+, n \geq 3$

$$x^n + y^n \neq z^n$$

  - Proof: (too long to fit in this section).

- A Shallow Theorem:
  - The number 3190966795047991905432 has a divisor between 25800000000 and 25900000000.
  - Proof: 25846840632.

NP-Completeness & Logic

- Theory of **NP**-completeness:
  - Every (deep) theorem reduces to shallow one.

Given theorem T and bound n on the length (in bits) of its proof there exist integers $0 \le A, B, C \le 2^{n^c}$ such that A has a divisor between B and C if and only if T has a proof of length n.

- Shallow theorem easy to compute from deep one.
- Shallow proofs are not much longer.
- Every **NP**-complete problem = “format” for proofs.

“Of all the Clay Problems, this might be the one to find the shortest solution, by an amateur mathematician.”
Devlin, *The Millenium Problems*
(Possibly thinking of the case P=NP)

  - Cryptography might well be impossible (current systems all broken simultaneously)

“If someone shows P=NP, then they prove any theorem they wish. So they would walk away not just with $1M, but $6M by solving all the Clay Problems!”
Lance Fortnow, *Complexity Blog*

  - Proof would be very interesting.
  - Might provide sound cryptosystems.

P = NP? is Mathematics-Complete

- independent of Peano's axioms, Choice ...

Probabilistic Verification of Proofs

- NP-completeness implies many surprising effects for logic.

- Examples:
  - Proofs can be verified <u>interactively</u> much more quickly than in “published format”!
  - Proofs may reveal <u>knowledge</u> selectively!
  - Proofs need *not* be <u>fully read</u> to verify them!

- “Deep theorems” of computational complexity.

Computation and You?

Computation beyond Computers

- Computation is not just about computers:
  - It models all systematic processing …
    - Adding/Subtracting
    - Logical Deduction
    - Reasoning
    - Thought
    - Learning

    - Cooking (“Recipes = Algorithms”)
    - Shampoo’ing your hair.
    - Design, Engineering, Scientific …

Biological organisms compute

- Folded structure of protiens determines their action.
  - Common early belief: Protiens fold so as minimize their energy.
  - However …
    - Minimum Energy configuration hard to compute (NP-complete).
  - Implication:
    - Perhaps achievable configurations are not global minima.

NP-Completeness and Economics

- Economic belief:
  - Individuals act rationally, optimizing their own profit, assuming rational behavior on other’s part.
- However …
  - Optimal behavior is often hard to compute (NP-complete)
  - In such cases irrational (or bounded rationality) is best possible.
  - Alters behavior of market.

NP-Completeness and the Brain

- Axiom: Brain is a computer

  (Follows from Universality).
- Implications to Neuroscience:
  - What is the model of computing (neural network, other?)
- More significantly … to Education:
  - Education = Programming of the brain

    (without losing creativity)
  - What algorithms to "teach"
    - Why multiplication? What is the point of "rote"?
    - Do resources matter? How much?
    - How much complexity can a child's brain handle?

NP-Completeness and Life

- Life = Choices + Consequences
  - Which school should I go to?
  - What subjects should I learn?
  - How should I spend my spare time?
  - Which job should I take?
  - Should I insult my boss today? Or tomorrow?
  - Sequence of simple steps that add up …
  - Eventually we find out if we did the right thing!
- Life = (Non-deterministic) computation.
- P = NP? ⇔ Humans don’t need creativity/choice

Computation and You

- Eventually … humans are characterized by their intelligence.
- Intelligence is a “computational effect”.
- Inevitably “computation” is the “intellectual core of humanity”.
- Shouldn’t be surprised if it affects all of us.

Future of Computing

Tracing Computing Forwards

“Rumors of its demise are greatly exaggerated …”

- Computing thus far …
  - First Law: Universality
  - Second Law(?): Irreversibility.

- Just the <u>beginning</u> …
  - … of Micro-Computer Science (<u>one</u> computer manipulating information).

Future = Macro-Computer Science: The vast unknown

- What happens when many computers interact?
  - What determines long term behavior?
  - What describes long term behavior?
  - What capabilities do we have (as intelligent beings, society) to control and alter this long term behavior?
  - How do computers evolve?
- Questions relevant already: Internet, WWW etc.
- What scientific quests are most similar?
  - (Statistical) Physics? Biology? Chemistry (big reactions)?
  - Sociology? Logic?
  - Mathematics?

- Computation = Mathematics of the 21st Century.

Acknowledgments (+ Pointers)

- This talk is inspired by (and borrows freely from) …
- Christos Papadimitriou: The Algorithmic Lens
  - http://lazowska.cs.washington.edu/fcrc/Christos.FCRC.pdf
- Avi Wigderson: A world view through the computational lens
  - http://www.math.ias.edu/~avi/TALKS/
- Many colleagues: esp. Oded Goldreich, Salil Vadhan

Thank You!