

CS 199 Lectures 27-29

Limitations of Computing

Limitations of Computation

- Hardware precision
- Software - we're human
- Problems
 - complex problem formulation (weather?)
 - complexity
 - inherent uncomputability

Hardware

- Precision errors
- Hardware Malfunction
- Design Errors (See famous Intel Pentium division bug)
- Read Text

Software

- Software becomes more and more complex. When programs have millions of lines of code, it's easy for errors to occur.
- Poor Design
- Human errors (Software bugs)

Problems

- Complex problems

- many factors and variables
- difficulty modeling
- no “closed form”

*temp tomorrow = temp. today * 1.1 - windchill...*

- solved via large-scale simulation with lots of math dictating behavior of “cells” based on adjacent cells. turn crank, see what happens.
- as moments increase, accuracy decreases

Problems

Computational Complexity

- Running time of an algorithm is a function of input size
- Some functions grow faster than others
- WAY faster
- For some problems, we have efficient algorithms (functions that grow moderately).
- For others, the best known algorithms have running times that grow so fast that they are impractical, and always will be.
- Some problems provably cannot be solved with a faster algorithm

Running Times

- Computational Complexity

Recall bubble-sort and insertion-sort ?

$$\begin{aligned}\text{num. comparisons} &= 1 + 2 + \dots + n-1 = n(n-1)/2 \\ &= (1/2)n^2 - (n/2)\end{aligned}$$

“big-Oh” notation.... say $O(n^2)$ instead.

- ignore constants, take highest degree of polynomial

Running Times

- $O(1)$: constant time.... doesn't depend on input
- $O(\log n)$: binary search
- $O(n)$: linear search, adding n numbers, etc.
- $O(n \log n)$: mergesort, quicksort on average
- $O(n^2)$: bubblesort, insertion, selection sort
- $O(n^3)$: slower
- $O(n^4)$: even slower

- $O(2^n)$: impractically slow algorithm
- $O(n!)$: even more impractically slow

Spreadsheet of running times

- and execution times at 1 billion steps/sec

Problems

- graph coloring (naïve algorithm: _____)
 - traveling salesperson (naïve algorithm: _____)
 - subset sum (naïve algorithm: _____)
 - steiner tree (compare to mst). _____)
 - circuit satisfiability

 - hundreds of others

 - \$\$\$\$ MILLION DOLLAR PRIZE \$\$\$\$
 - <http://www.claymath.org/millennium/>
- Find an efficient method for solving any one of these

UNCOMPUTABLE problems

- previous category were problems that could be solved, but known methods took too long
- this category are problems for which we know:

THEY CANNOT BE SOLVED BY

ANY ALGORITHM WHATSOEVER

History

- Hilbert's "program"
- Gödel's undecidability theorem
- Turing's halting problem
- Natural problems and consequences

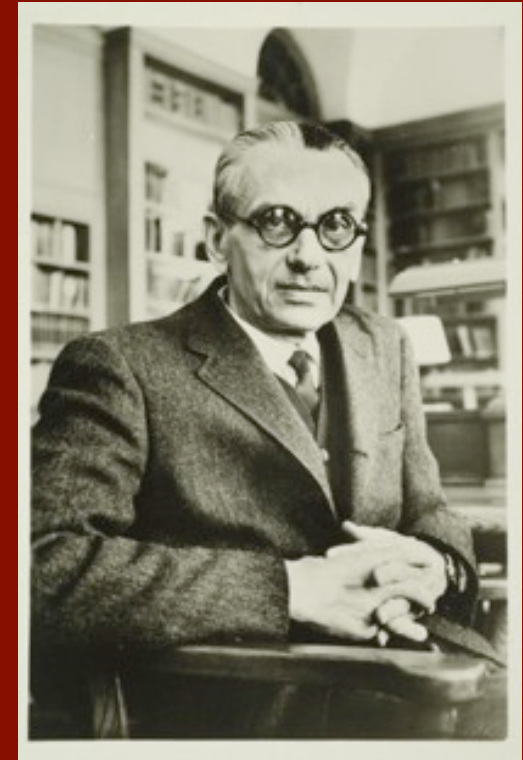
David Hilbert

- famous mathematician
- 23 problems
- founder of “metamathematics”
- goal to “mechanize” mathematics



Kurt Gödel

- Logician
- Proved in 1931...(age 25)
“there are true propositions that cannot be proved” (i.e., “no” to Hilbert’s program)
- Shook the foundations of
 - mathematics
 - philosophy
 - science
 - everything



Alan Turing

- British Mathematician
- Seminal work in Computation, AI, Cryptanalysis
- Subject of several books, movies



A Halting Problem

$x = \text{input}$ ("please enter a positive number")

WHILE $x > 1$:

IF x is even:

$$x = x/2$$

ELSE:

$$x = 3x + 1$$

The Halting Problem

- Given: Program, Input to program
Does the program on the input ever halt?
- Critical question that pervades all of computation
- Has much more broad consequences than it at first appears

Hilbert's 10th problem

- Finding integer solutions to a polynomial with integer coefficients:

$$15x^4y^2 - 3xy^3z^2 + 2x^2z = 1$$

- No algorithm exists for finding such solutions, or determining whether there are any.

“Proof” of Halting problem

- Suppose there was an algorithm that could tell whether a program halted on given input.....