Lecture 7 (more or less)
Algorithms

Software: how we tell a computer what to do

- **hardware: a general purpose computer**
  - capable of doing instructions repetitively and very fast
  - doesn't do anything itself unless we tell it what to do

- **software: the instructions we want it to perform**
  - different set of instructions
    => different program
    => machine behaves differently
  - program and data are stored in the same memory and manipulated by the same instructions

- **to tell a computer what to do,**
  - we have to spell out the steps in excruciating detail
  - programming languages help handle a lot of the details

Software roadmap

- algorithms
  - precise but abstract descriptions of how to do a task
  - how to describe algorithm speed or efficiency
- programs
  - complete concrete descriptions of how to do a task on a real computer
- programming languages
  - precise notations for describing how to do tasks on a computer
    e.g., Toy, Javascript, Python
- real programs (big software)
  - operating systems
  - applications

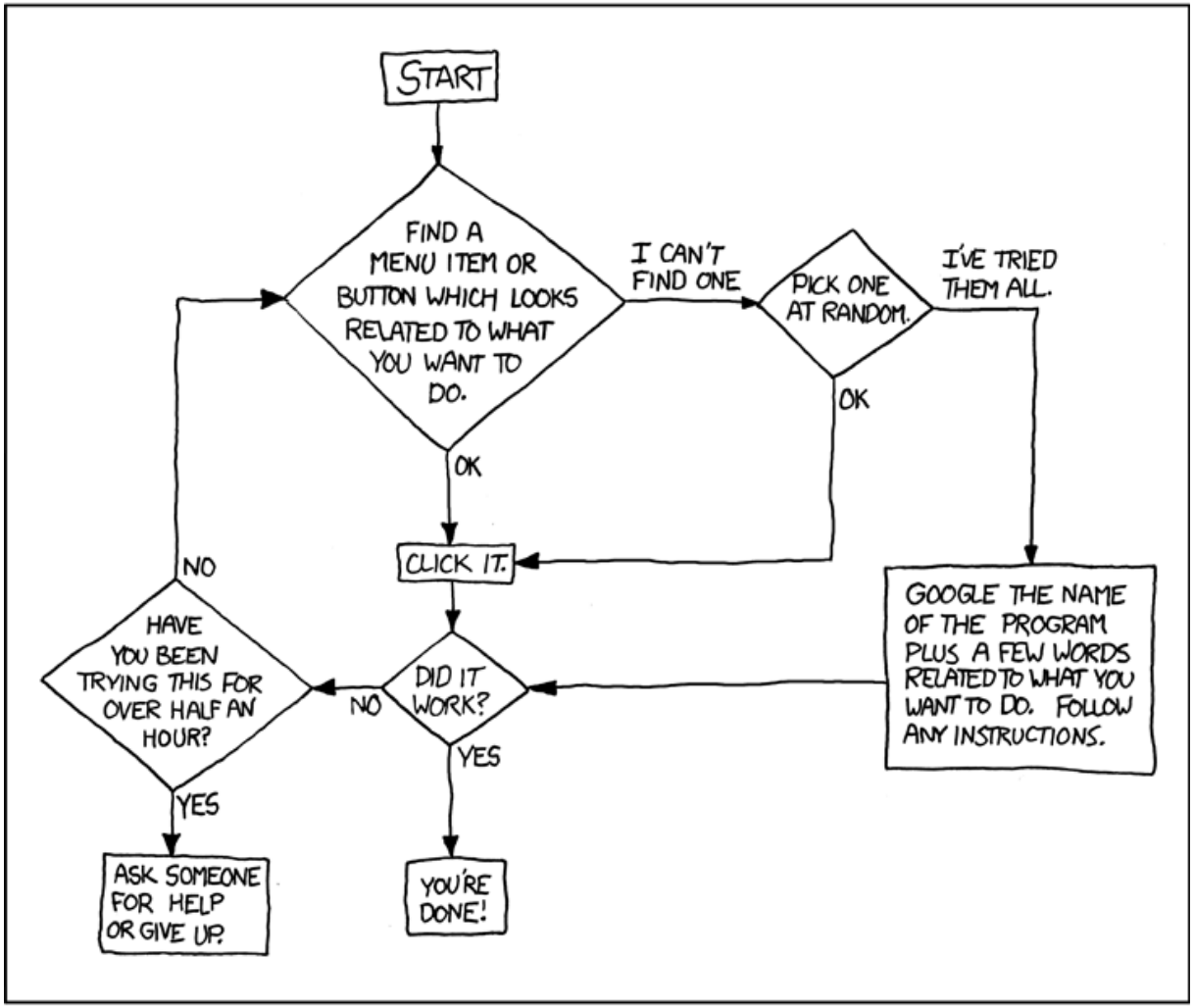
- social / political / economic / legal issues
  - intellectual property: patents, copyrights, interfaces
  - standards
  - open source

Algorithm for becoming a computer expert

(xkcd.com/627)

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER "NOT COMPUTER PEOPLE."

WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:



PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN. CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

A real-world algorithm

50	Enter your qualified 5-year gain, if any, from Schedule D (Form 1040), line 35 (as refigured for the AMT, if necessary) (see page 8 of the instructions)	50		
51	Enter the **smaller** of line 49 or line 50		51	
52	Multiply line 51 by 8% (.08) ▶			52
53	Subtract line 51 from line 49		53	
54	Multiply line 53 by 10% (.10) ▶			54
55	Subtract line 47 from line 46		55	
56	Subtract line 45 from line 44		56	
57	Enter the **smaller** of line 55 or line 56		57	
58	Multiply line 57 by 15% (.15) ▶			58
59	Subtract line 57 from line 56		59	
60	Multiply line 59 by 20% (.20) ▶			60
	If line 38 is zero or blank, skip lines 61 and 62 and go to line 63. Otherwise, go to line 61.			
61	Subtract line 44 from line 40		61	
62	Multiply line 61 by 25% (.25) ▶			62
63	Add lines 42, 48, 52, 54, 58, 60, and 62			63
64	If line 36 is $175,000 or less ($87,500 or less if married filing separately), multiply line 36 by 26% (.26). Otherwise, multiply line 36 by 28% (.28) and subtract $3,500 ($1,750 if married filing separately) from the result			64
65	Enter the **smaller** of line 63 or line 64 here and on line 31			65

Algorithms

- an algorithm is the computer science version of a
  really careful, precise, unambiguous recipe or procedure

- a sequence of steps that performs some computation

- each step is expressed in terms of basic operations whose meaning is completely specified
  - basic operations or "primitive operations" are given
    e.g., arithmetic operations
- all possible situations are covered
  - the algorithm never gets to a situation where it doesn't know what to do next
- guaranteed to stop
  - does not run forever

Linear time algorithms

- lots of algorithms have this same basic form:

  look at each item in turn

  do the same simple computation on each item:

  does it match something (looking up a name in a list of names)

  count it (how many items are in the list)

  count it if it meets some criterion (how many of some kind in the list)

  remember some property of items found (largest, smallest, …)

  filter it (preserve items with some property)

  transform it in some way (limit size, convert case of letters, …)

- amount of work (running time) is proportional to amount of data
  - twice as many items will take twice as long to process
  - computation time is linearly/directly proportional to length of input

Log n algorithms

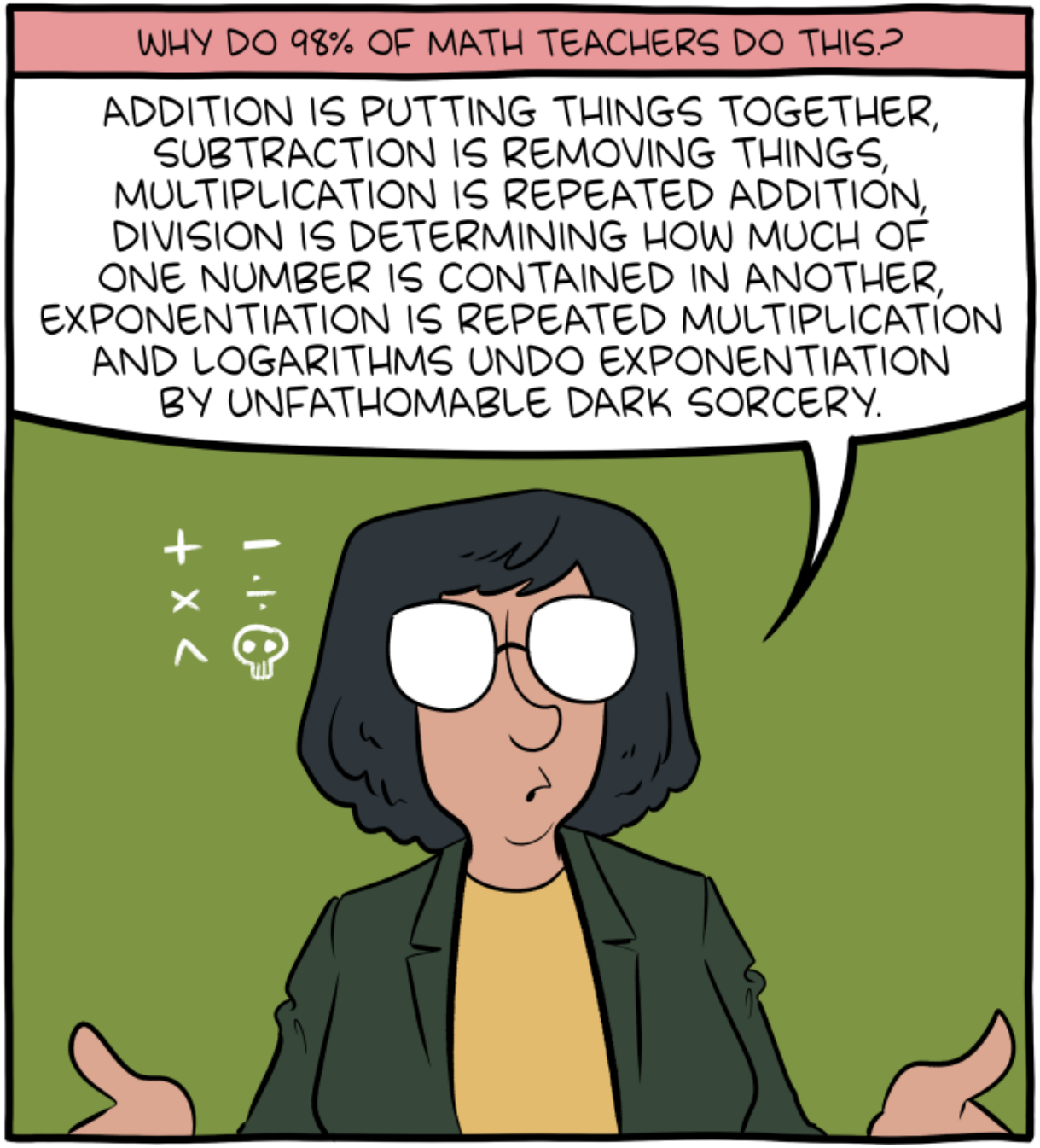
- how do we find a name in a phone book?
  - linear search requires looking at all the names

- if the names are sorted into alphabetical order,
  we can use binary search, which is much faster than linear
  - an example of a "divide and conquer" algorithm

- data has to be sorted
  - have to be able to access any data item equally quickly
  - "random access"

- why is binary search faster than linear searching?
  - each test / comparison cuts the number of things to search in half
- how much faster is it?
  - the number of comparison is approximately $\log_2 n$ for n items

smbc-comics.com



Logarithms for COS 109

- all logs in 109 are base 2
- all logs in 109 are integers

- if N is a power of 2 like 2^m, $\log_2$ of N is m
- if N is not a power of 2, $\log_2$ of N is
  the number of bits needed to represent N
  the power of 2 that's bigger than N
  the number of times you can divide N by 2 before it becomes 0

- you don't need a calculator for these!
  - just figure out how many bits or what's the right power of 2

- logs are related to exponentials: $\log_2 2^N$ is N

- it's the same as decimal, but with 2 instead of 10

Algorithms for sorting

- binary search needs sorted data

- how do we sort names into alphabetical order?
- how do we sort numbers into increasing or decreasing order?
- how do we sort a deck of cards?

- how many comparison operations does sorting take?

- "selection sort":
  - find the smallest/earliest
    using a variant of the "find the largest" algorithm
  - repeat on the remaining names
  - this is what bridge players typically do when organizing a hand

- what other algorithms might work?

How fast do these run?

- searching an unordered/unsorted list of names
  - time is proportional to length of the list
    because you might have to go all the way to the end
  - twice as many items takes twice as long to search
- searching a sorted list of names with binary search
  - time is much faster (proportional to logarithm of length)
    because you can use divide-and-conquer to narrow the search
  - twice as many items needs only one more probe

- sorting n items takes time proportional to n^2 with simple sorting algorithms like selection sort
  - twice as many items takes 4 times as long to sort

- there are much faster sorting algorithms (e.g., Quicksort)
  - time proportional to n log n

Quicksort: an n log n sorting algorithm

- make one pass through data, putting all small items in one pile and all large items on another pile
  - there are now two piles, each with about 1/2 of the items
  - and each item in the first pile is smaller than any item in the second
- make a second pass; for each pile, put all small items in one pile and all larger items in another pile
  - there are now four piles, each with about 1/4 of the items
  - and each item in a pile is smaller than any item in later piles
- repeat until there are n piles
  - each item is now smaller than any item in a later pile

- each pass looks at n items
- each pass divides each pile about in half; stops when pile size is 1
  - number of divisions is log n
- n log n operations

Tower of Hanoi: an exponential algorithm

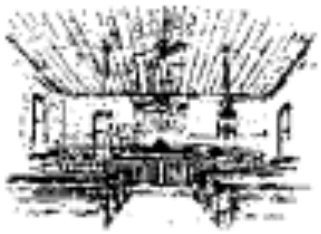
- 

Complexity hierarchy (or part of it)

$\log n$	logarithmic	(sub-polynomial?)
n	linear	polynomial
$n \log n$		polynomial
n^2	quadratic	polynomial
n^3	cubic	polynomial
2^n	exponential	(not polynomial)

PEKIN
COURT HOUSE

LINCOLN & HERNDON LAW OFFICE SPRINGFIELD



PEKIN
COURT HOUSE

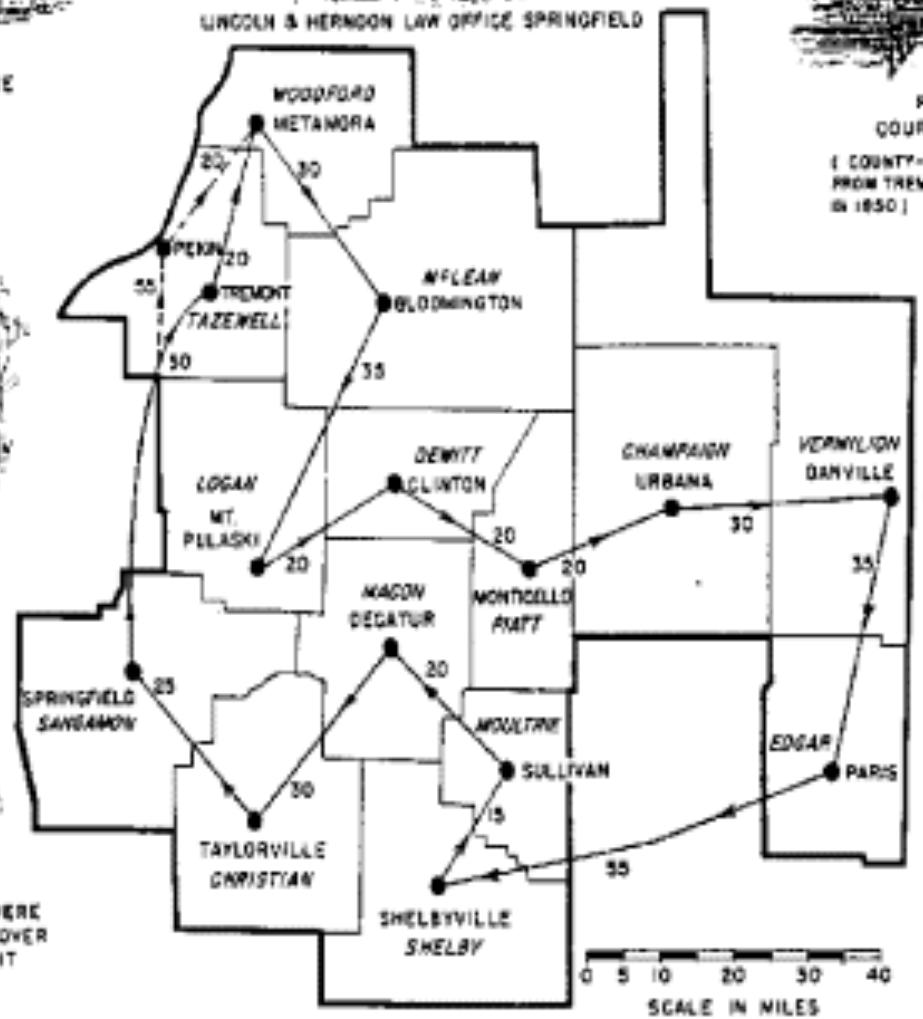
(COUNTY-SEAT CHANGED FROM TREMONT TO PEKIN IN 1850)

METAMORA
COURT HOUSE

LINCOLN & HERNDON
LAW OFFICE SPRINGFIELD

JUDGE DAVIS
DAVIS & LINCOLN WERE THE ONLY MEN TO COVER THE ENTIRE CIRCUIT IN 1850

MARKERS
ALONG 8TH CIRCUIT ABRAHAM LINCOLN TRAVELED THIS WAY AS HE RODE THE CIRCUIT OF THE EIGHTH JUDICIAL DISTRICT 1847-1859



COUNTIES IN ITALICS • COUNTY SEATS IN VERTICALS • LENGTH OF CIRCUIT ABOUT 450 MILES

THE 8TH CIRCUIT AS TRAVELED BY MR. LINCOLN IN 1850

5

P vs NP Problem

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

- The Millennium Problems
- Official Problem Description — Stephen Cook
- Lecture by Vijaya Ramachandran at University of Texas (video)
- Minesweeper

Algorithms in Computer Science

- study and analysis of algorithms is a major component of CS courses
  - what can be done (and what can't)
  - how to do it efficiently (fast, compact memory)
  - finding fundamentally new and better ways to do things
  - basic algorithms like searching and sorting
  - plus lots of applications with specific needs

- big programs are usually a lot of simple, straightforward parts, often intricate, occasionally clever, very rarely with a new basic algorithm, sometimes with a new algorithm for a specific task

Algorithms versus Programs

- **An algorithm is the computer science version of a really careful, precise, unambiguous recipe**
  - defined operations (primitives) whose meaning is completely known
  - defined sequence of steps, with all possible situations covered
  - defined condition for stopping

  - an idealized recipe

- **A program is an algorithm converted into a form that a computer can process directly**
  - like the difference between a blueprint and a building
  - has to worry about practical issues like finite memory, limited speed, erroneous data, etc.

  - a guaranteed recipe for a cooking robot