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

- **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)
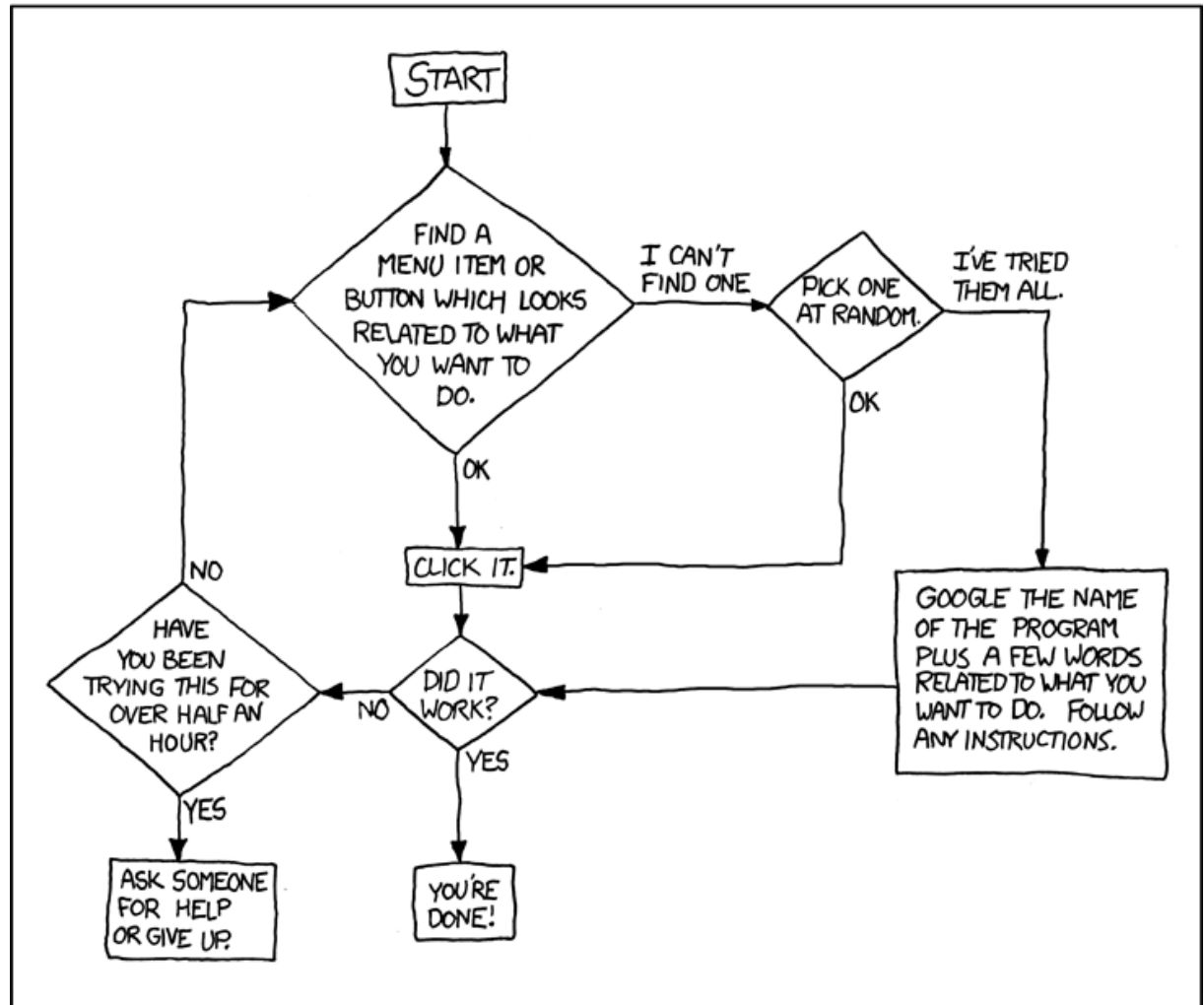
Algorithms

- an <u>algorithm</u> 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 <u>binary search</u>, 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

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 "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?

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

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)

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