What is an Algorithm?

- not a beat that Al Gore came up with!
- Origin of word: 9th century Persian scholar Muhammad ibn Musa al-Khwarizmi
  - also the origin of the word "algebra", the decimal system for representing numbers, and invented the number zero (also independently defined in India)
  - his name was combined with the Greek arithmos (number) and algos (pain).
- Until very recently, the word algorithm referred to techniques for auditable arithmetic procedures. People trained to execute these procedures were called algorists, computators, or computers.
- Informal definition: solicit from students
- Workshop informal definition to formal definition
- Formal definition: an
  1. explicit
  2. precise
  3. unambiguous
  4. mechanically-executable
  5. finite sequence of
  6. elementary instructions

Explicit but not precise or unambigous: "my name is not Bob Ross"

Precise but not explicit or elementary: "choose the best edge in the graph"

Mechanical execution

- Turing machine model
- Human computers (Apollo)
- Crab computer
- Liquid computers - immune system
- Compass and straightedge
- human does not need to intervene and use best judgement

Not an algorithm

"Martin's algorithm"

```
BeAMillionaireAndNeverPayTaxes():
  1. Get a million dollars
  2. If the tax man comes to your door and says, "You have never paid taxes!", then
     - Say "I forgot."
```

Not explicit enough for most of us to execute!

Good example of a reduction: reduces a challenging problem by using a "solved" problem (getting one million dollars).

Example of Algorithm

```
BottlesOfBeer(n):
  For i in [n..1]
    Sing "i bottles of beer on the wall, i bottles of beer"
    Sing "Take one down, pass it around, i-1 bottles of beer on the wall."
  Sing "No bottles of beer on the wall, no bottles of beer,"
  Sing "Go to the store, buy some more, n bottles of beer on the wall"
  Optional: repeat BottlesOfBeer(n)
```

Is this a finite set of instructions?

If the recursive call is implemented, is this an algorithm that will ever terminate?

What assumptions are made in this algorithm description? (n must be non-negative)

Describing algorithms

- To design or analyze an algorithm, one must first learn how to describe algorithms.
- Human brain hardware is not as precise as the typical digital computer; we must learn to think rigorously
- Learning to describe algorithms and their properties is an essential skill for computer scientists.
- Always assume your audience is made up of "skeptical novices": people less knowledgeable and clever than you, and who will interpret your writing extremely literally.
  - Anyone done the peanut butter sandwich instruction assignment?
- Primary job as a developer is not to write awe-inspiring code. You much be able to teach other people how and why your algorithms work!

Components of a complete algorithm description

- What: Precise specification of the problem solved by the algorithm.
- How: Precise description of the algorithm itself
- Why: A proof that the algorithm solves the problem it is supposed to solve
- How well: An analysis of running time (and/or space) of the algorithm

Not often developed in this order - more often all pieces are developed simultaneously with lots of iterative feedback loops.

Why analyze algorithms?

1. Determine performance characteristics and predicting resource requirements

- Time, memory, bandwidth, etc

2. Determine optimality: since algorithms are executed mechanically, there are physical limitations on the resource requirements. Can we find the "best" algorithm with respect to different resources?
3. Correctness: some bugs are extremely subtle (merge sort example).

- Even a provably correct algorithm can be implemented incorrectly, of course! See: formal methods.
- Often use induction to prove that algorithms are correct on all possible inputs!

4. Deep questions of computability and information processing in the universe

- Millenium problems

Formalizing resource usage

- Some computers are faster than others
- Runtime expression should be machine-independent, so we need to choose a "hypothetical computer"
- RAM model:
  - single processor, sequential execution
  - constant-time elementary instructions (arithmetic, data movement, control)
  - runtime cost is uniform (1 time unit) for all simple instructions
  - memory is unlimited and "flat" : no hierarchy, accessing a variable in memory takes 1 time unit.
- Running time of an algorithm is always given in terms of input size (usually N).
  - Input can have other structure, eg: sorted or unsorted, cyclic or acyclic graphs
- Also known as the runtime *complexity* of the algorithm. Algorithms belong to *complexity classes* defined by the *asymptotic complexity* as N becomes large.

(Compare growth rates slide)