Algorithms
Problems, Algorithms, Programs

- Problem - a well defined task.
  - Sort a list of numbers.
  - Find a particular item in a list.
  - Find a winning chess move.
Algorithms

- A series of precise steps, known to stop eventually, that solve a problem.
- NOT necessarily tied to computers.
- There can be many algorithms to solve the same problem.
Characteristics of an Algorithm

- Precise steps.
- Effective steps.
- Has an output.
- Terminates eventually.
Trivial Algorithm

- Computing an average:
  - Sum up all of the values.
  - Divide the sum by the number of values.

(Precise enough?)
Problem - Finding the Greatest Common Denominator

Examples:
- $\text{gcd}(12, 2) = 2$
- $\text{gcd}(100, 95) = 5$
- $\text{gcd}(100, 75) = 25$
- $\text{gcd}(39, 26) = 13$
- $\text{gcd}(17, 8) = 1$
Possible Algorithm #1

- Assumption: $A > B \geq 0$
  - If $A$ is a multiple of $B$, then $\gcd(A, B) = B$.
  - Otherwise, return an error.

- Works for $\gcd(12, 2) = 2$
- But what about $\gcd(100, 95)$???
Possible Algorithm #2

– Start with 1 and go up to B.
– If a number if a common divisor of both A and B, remember it.
– When we get to B, stop. The last number we remembered is the gcd.

Works, but is there a better way?
Think about gcd(100, 95)
Euclid’s Algorithm

- Makes use of the fact that:
  - $\text{gcd}(A, B) = \text{gcd}(B, A \text{ rem } B)$
    - Note: $A \text{ rem } B$ refers to the remainder left when $A$ is divided by $B$.
    - Examples:
      - $12 \text{ rem } 2 = 0$
      - $100 \text{ rem } 95 = 5$
Euclid’s Algorithm

function gcd(A, B)
    If B == 0
        return A;
    else
        return gcd(B, A rem B);

Note – this algorithm is recursive.

Examples:
- gcd(12, 2) -> gcd(2, 0) -> 2
- gcd(100, 95) -> gcd(95, 5) -> gcd(5, 0) -> 5
Why do we care?

- Let’s say we want the gcd of 1,120,020,043,575,432 and 1,111,363,822,624,856.
- Assume we can do 100,000,000 divisions per second.
- Algorithm #2 will take about three years.
- Euclid’s Algorithm will take less than a second.
Programs vs. Algorithms

- Program: “A set of computer instructions written in a programming language”.
- We write Programs that implement Algorithms.
Problems vs. Algorithms vs. Programs

- There can be many algorithms that solve the same problem.
- There can be many programs that implement the same algorithm.
- We are concerned with:
  - Analyzing the difficulty of problems.
  - Finding good algorithms.
  - Analyzing the efficiency of algorithms.
Classifying Problems

- Problems fall into two categories.
  - Computable problems can be solved using an algorithm.
  - Non-computable problems have no algorithm to solve them.

- We don’t necessarily know which category a problem is in. (this is non-computable)
Classifying Computable Problems

- **Tractable**
  - There is an efficient algorithm to solve the problem.

- **Intractable**
  - There is an algorithm to solve the problem but there is no efficient algorithm. (This is difficult to prove.)
Examples

- Sorting: tractable.
- The traveling salesperson problem: intractable. (we think…)
- Halting Problem: non-computable. – (More on this later…)
Measuring Efficiency

- We are (usually) concerned with the time an algorithm takes to complete.
- We count the number of “basic operations” as a function of the size of the input.
  - Why not measure time directly?
  - Why not count the number of instructions executed?
Example: Computing an Average

```matlab
function avg = average(array)
    sum = 0;
    for i = 1:size(array),
        sum = sum + array(i);
    end
    avg = sum / size(array);
```

- The statement inside the for loop gets executed len(n) times.
- If the length is n, we say this algorithm is “on the order of n”, or, O(n).
- O(n)???
Some Mathematical Background

- Let’s see some examples in Matlab...
Big O

- When comparing algorithms, we are concerned with asymptotic run time.
  - How long does the algorithm take as the input gets very large?
- Big O: Worst case running time, discounting constants and lower order terms.
- Example:
  - $5n^3 + 2n$ is $O(n^3)$
Tractable vs. Intractable Problems

- Problems with polynomial time algorithms are considered tractable.
- Problems *without* polynomial time algorithms are considered intractable.
  - Eg. Exponential time algorithms.
  - (More later)
Big O Can Be Misleading

- Big O analysis is concerned with worst case behavior.
- Sometimes we know that we are not dealing with the worst case.
Searching an Array

Function found = search(array, key)
    found = false;
    for i = 1:size(array),
        if array(i) == key
            found = true;
        end
    end
end

- Worst case?
- Best case?
Binary Search

- If we are searching in a sorted list, we choose the half that the value would be in.
- We continue to cut the area we are searching in half until we find the value we want, or there are no more values to check.
- Worst case?
- Best case?
Algorithms Exercise...