Algorithms in Systems Engineering
IE170

Lecture 3

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - CLRS Chapter 2
- References
  - D.E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (Third Edition), 1997.

Designing Algorithms

- We have already motivated the development of algorithms that are both correct and efficient.
- How do we know if an algorithm is correct and what do we mean by efficient?

Analyzing Algorithms

- The goal of analyzing an algorithm is to determine how quickly it will execute in practice.
- This can be done either *empirically* or *theoretically*.
- Empirical analysis involves implementing the algorithm and testing it on various instances.
- The difficulty is knowing which instances to test it on.
- What do we want to know?

Theoretical Analysis

- In general, the speed of execution of an algorithm depends on

- Theoretical analysis allows us to separate the effect of these factors.
- In a basic theoretical analysis, we try to determine how many "steps" would be necessary to complete the algorithm.
- We assume that each "step" takes a constant amount of time, where the constant depends on the hardware.
- We might also be interested in other resources required for the algorithm, such as memory.

Models of Computation

- In order to analyze the number of steps necessary to execute an algorithm, we have to say what we mean by a "step."
- To define this precisely is tedious and beyond the scope of this course.
- A precise definition depends on the exact hardware being used.
- Our analysis will assume a very simple model of a computer called a *random access machine* (RAM).
- In a RAM, the following operations take one step.

- This is a very idealized model, but it works in practice.
- We will sometimes need to simplify the model even further.

Running Time

- The number of steps required for an algorithm to solve a given instance of a problem is called the *running time* for that instance.
- The overall *running time* of an algorithm is the number of steps required to solve an instance of the problem in either

- Best case behavior is usually uninteresting.
- Average case behavior can be difficult to define and analyze.
- Worst case is easier to analyze and can yield useful information.
- Unless otherwise specified, running time is in the worst-case.

Evaluating a Polynomial

- Consider the problem of evaluating a polynomial.

  Input: The coefficients $a_0, \ldots, a_n$ and $x \in \mathbb{R}$.
  Output: The value $\sum_{i=0}^{n} a_i x^i$.

- What is the running time of the most obvious algorithm?

- Is there a more efficient algorithm?

The Input Size

- Notice that in the previous example, the worst-case running time depended only on the number of input values, or the *size of the input*.
- This is almost always the case.
- In reality, the running time could be affected by the size of the input *values* as well, but we'll ignore this for now.
- We are interested in how the running time grows generally as the input size grows.
- Any algorithm can be used to solve a small problem.
- It is the really large problems that require efficient algorithms.

Order of Growth

- Because we are mainly interested in how the running time grows as the instances become larger, we won't need "exact" running times.
- We will allow some "sloppiness" and ignore constants and low order terms.
- Because of our many simplifying assumptions, the low order terms may not be accurate anyway.
- Next time, we'll define these notions more precisely.