COMP 355
Advanced Algorithms

Introduction &
Course Overview

Rhodes College
—1848—

What is an algorithm?

- An *algorithm is* any well-defined computational procedure that takes some values as *input* and produces some values as *output*.
- Provides a step-by-step method for solving a computational problem
- Unlike programs, algorithms are not dependent on a particular programming language, machine, system, or compiler.
- Mathematical entities, which can be thought of as running on some sort of *idealized computer* with an infinite random access memory and an unlimited word size
- Algorithm design is all about the mathematical theory behind the design of good programs.

Why study algorithm design?

- Internet. Web search, packet routing, distributed file sharing,…
- Biology. Human genome project, protein folding,…
- Computers. Circuit layout, databases, caching, networking, compilers,…
- Computer graphics. Movies, video games, virtual reality,..
- Security. Cell phones, e-commerce, voting machines,…
- Multimedia. MP3, JPG, DivX, HDTV, face recognition, …
- Social Networks. Recommendations, news feeds, advertisements,…
- Physics. N-body simulation, particle collision simulation,…

Why study algorithm design?

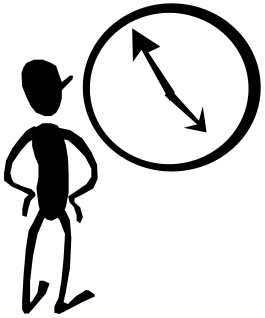
- Programming is a very complex task for many reasons.
  - Large programming projects are structurally complex (software engineering)
  - Need to store and access large data sets efficiently (data structures and databases)
  - Complex computational problems
    - Numerical data (numerical analysis course)
    - Discrete data (this course)

Why study algorithm design?

- Algorithms represent only a small fraction of the code generated in a large software system
- Very important to the overall success
- Bad idea! – design an inefficient algorithm and data structure to solve the problem and then fine-tune its performance
- Good idea! – design a correct and efficient algorithm to solve a problem

Course Overview

- Website:
  http://cs.rhodes.edu/welshc/CS355/F19/
  look here first for
  - News, hints, and helpful resources
  - Revisions, solutions, and corrections to problem sets
- Office Hours: Tues/Thurs 10-11:30am
- Grading
  Problem sets (worth 50%)
  Midterm Exam 1 (worth 15%)
  Midterm Exam 2 (worth 15%)
  Final Exam (worth 20%)
- Problem Sets
  - Roughly one a week
  - Most will include a short program to write
  - Programs will be written in Python



Course Overview

- Review of preliminary material
  - Asymptotics
  - Summations
  - Recurrences
  - Sorting
- Designing Optimization Algorithms
  - Dynamic Programming
  - Greedy Algorithms
- Graph Algorithms
  - Review BFS and DFS (connectivity in graphs)
  - Minimum Spanning Trees
  - Shortest Paths
  - Network Flows
- Intractable Problems

Course Topics

- Algorithm Analysis (Review)
- Recurrences and Master Theorem
- Greedy Algorithms
  - Interval Scheduling, Scheduling to minimize lateness, greedy graph algorithms
- Dynamic Programming
  - Weighted Interval Scheduling, Subset Sums, Knapsack, shortest path in a graph
- Network Flow
  - Network flows, bipartite matching, edge-disjoint paths
- NP & Computational Intractability
  - Polynomial-time reductions, definition of NP, NP-complete problems
- Approximation Algorithms
  - Greedy algorithms and bounds on the optimum, examples of approximation algorithms

Issues in Algorithm Design

- Mathematical objects (not as concrete as a computer program implemented in a particular language and executing on some machine)
- Must reason algorithmic properties mathematically
- Two fundamental issues to be considered:
  - Correctness
  - Efficiency

Correctness of an Algorithm

- Complex algorithms
  - Require careful mathematical proofs
  - May require proof of many lemmas and properties in the solution
- Simple algorithms
  - Short intuitive explanations based on algorithm's basic invariants are sufficient
  - Example: BubbleSort
    - The principle invariant is that on completion of the *i*th iteration, the last *i* elements are in their proper sorted positions.)

Efficiency of an Algorithm

- Establishing efficiency is more complex than establishing correctness
- Function of the amount of computational resources an algorithm uses
  - Execution time
  - Amount of space (memory)
- Consider efficiency in terms of input size
- We will usually focus on *worst-case analysis* in this course.

Presenting Algorithms

1. Present a clear, simple and unambiguous description of the algorithm (pseudo-code)
   - Keep it simple
   - **Example**: Say "Add X to the end of list L" rather than present code to do this or use some arcane syntax, such as "*L:insertAtEnd*(X)."
2. Present a justification or proof of the algorithm's correctness
   - A good proof provides an overview of what the algorithm does, and then focuses on any tricky elements that may not be obvious.
3. Present a worst-case analysis of the algorithms efficiency
   - Typically running-time, but also can include space if space is an issue.

Next Time

- Stable Matching Algorithm
- Read Section 1.1 in KT (your book)