Design and Theory of Algorithms
Introduction

Outline

- What is the course about?
- What is an Algorithm?
- Course Objectives
- Definitions
- Analysis of Algorithms

The Course

- Purpose: a rigorous introduction to the design and analysis of algorithms
  - *Not a lab or programming course*
  - *Not a math course, either*
- Required Textbook:

  Introduction to Algorithms, Cormen, Leiserson, Rivest, Stein

The Course – contd.

- Grading policy:
  - Assignments: 35 %
  - Midterm Examination: 25 %
  - Final Examination: 35 %
  - Class Participation: 5 %

Outline

- What is the course about?
- What is an Algorithm?
- Course Objectives
- Definitions
- Analysis of Algorithms

What is an Algorithm?

- Algorithm
  - is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
  - is thus a sequence of computational steps that transform the input into the output.
  - is a tool for solving a well - specified computational problem.
  - Any special method of solving a certain kind of problem (Webster Dictionary)

What is an Algorithm? - contd.

Algorithms are the ideas behind computer programs.

An algorithm is the thing that stays the same whether the program is in C++ running on a Cray in New York or is in BASIC running on a Macintosh in Alaska!

To be interesting, an algorithm has to solve a general, specified problem.

What's More Important than Performance?

- Modularity
- Correctness
- Maintainability
- Functionality
- Robustness
- User-friendliness
- Programmer time
- Simplicity
- Extensibility
- Reliability

Why study algorithms and performance?

- Algorithms help us to understand ***scalability.***
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** *for talking about program behavior*.
- Performance is the ***currency*** *of computing*.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

Outline

- What is the course about?
- What is an Algorithm?
- Course Objectives
- Definitions
- Analysis of Algorithms

Course Objectives

1. Details of classic algorithms
2. Methods for designing algorithms
3. Validate/verify algorithm correctness
4. Analyze algorithm efficiency
5. Prove (or at least indicate) no correct, efficient algorithm exists for solving a given problem
6. Writing clear algorithms and proofs

Course Objectives – contd.

- This course introduces students to the analysis and design of computer algorithms. Upon completion of this course, students will be able to do the following:
  - Analyze the asymptotic performance of algorithms.
  - Demonstrate a familiarity with major algorithms and data structures.
  - Apply important algorithmic design paradigms and methods of analysis.
  - Synthesize efficient algorithms in common engineering design situations.

Outline

- What is the course about?
- What is an Algorithm?
- Course Objectives
- Definitions
- Analysis of Algorithms

What is a Program?

- A program is the expression of an algorithm in a programming language
- a set of instructions which the computer will follow to solve a problem

What is a Problem?

- **Problem**:
  - Description of Input-Output relationship
- **Algorithm**:
  - A sequence of computational step that transform the input into the output.
- **Data Structure:**
  - An organized method of storing and retrieving data.
- **Our task:**
  - Given a problem, design a ***correct*** and ***good*** algorithm that solves it.

What is a problem? – contd.

- Definition
  - A mapping/relation between a set of input instances (domain) and an output set (range)
- Problem Specification
  - Specify what a typical input instance is
  - Specify what the output should be in terms of the input instance
- Example: Sorting
  - **Input**: A sequence of N numbers $a_1 \ldots a_n$
  - **Output**: the permutation (reordering) of the input sequence such that $a_1 \leq a_2 \leq \ldots \leq a_n$.

Types of Problems

Search: find X in the input satisfying property Y

Structuring: Transform input X to satisfy property Y

Construction: Build X satisfying Y

Optimization: Find the best X satisfying property Y

Decision: Does X satisfy Y?

Adaptive: Maintain property Y over time.

Types of Problems – contd.

- Learn general approaches to algorithm design
  - Divide and conquer
  - Greedy method
  - Dynamic Programming
  - Basic Search and Traversal Technique
  - Graph Theory
  - Linear Programming
  - Approximation Algorithm
  - NP Problem

Types of Problems – contd.

- Examine methods of analyzing algorithm correctness and efficiency
  - Recursion equations
  - Lower bound techniques
  - O,Omega and Theta notations for best/worst/average case analysis
- Decide whether some problems have no solution in reasonable time
  - List all permutations of n objects (takes n! steps)
  - Travelling salesman problem
- Investigate memory usage as a different measure of efficiency

Outline

- What is the course about?
- What is an Algorithm?
- Course Objectives
- Definitions
- Analysis of Algorithms

The study of Algorithm

- How to devise algorithms
- How to express algorithms
- How to validate algorithms
- How to analyze algorithms
- How to test a program

Algorithm design methods

- Something of an art form
- Cannot be fully automated
- We will describe some general techniques and try to illustrate when each is appropriate

Two desired properties of algorithms

- Correctness
  - Always provides correct output when presented with legal input
- Efficiency
  - Computes correct output quickly given input

Algorithm correctness

- Proving an algorithm generates correct output for all inputs
- One technique covered in textbook
  - Loop invariants
- We will do some of this in the course, but it is not emphasized as much as other objectives

Algorithmic Correctness – contd.

- Example: Traveling Salesperson Problem (TSP)
- **Input**: A sequence of N cities with the distances d_{ij} between each pair of cities
- **Output**: a permutation (ordering) of the cities $<c_{1'}, \ldots, c_{n'}>$ that minimizes the expression

  $\Sigma_{j=1 \text{ to } n-1} d_{j',j'+1} + d_{n',1'}$
- Which of the following algorithms is correct?
  - Nearest neighbor: Initialize tour to city 1. Extend tour by visiting nearest unvisited city. Finally return to city 1.
  - All tours: Try all possible orderings of the points selecting the ordering that minimizes the total length:

Efficiency

- Example: Odd Number Problem
- **Input**: A number n
- **Output**: Yes if n is odd, no if n is even
- Which of the following algorithms is most efficient?
  - Count up to that number from one and alternate naming each number as odd or even.
  - Factor the number and see if there are any twos in the factorization.
  - Keep a lookup table of all numbers from 0 to the maximum integer.
  - Look at the last bit (or digit) of the number.

Analyzing algorithms

- The “process” of determining how much resources (time, space) are used by a given algorithm
- We want to be able to make quantitative assessments about the value (goodness) of one algorithm compared to another
- We want to do this WITHOUT implementing and running an executable version of an algorithm

Proving hardness results

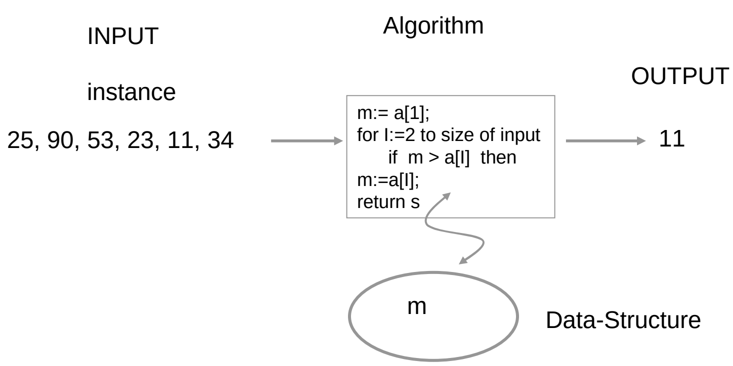
- We believe that no correct and efficient algorithm exists that solves many problems such as TSP
- We define a formal notion of a problem being hard
- We develop techniques for proving hardness results

Clear Writing

- Methods for Expressing Algorithms
  - Implementations
  - Pseudo-code
  - English
- Writing clear and understandable proofs
- My main concern is not the specific language used but the clarity of your algorithm/proof

Example: What is an Algorithm?

Problem: Input is a sequence of integers stored in an array.
Output the minimum.



Example Algorithm

Problem: The input is a sequence of integers stored in array. Output the minimum.

Algorithm A

$m \leftarrow a[1]$;
For $i \leftarrow 2$ to size of input;
    if $m > a[i]$ then $m \leftarrow a[i]$;
output m.

What do we need?

Correctness: Whether the algorithm computes the correct solution for **all** instances

Efficiency: Resources needed by the algorithm

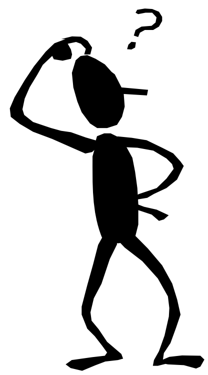
1. Time: Number of steps.
2. Space: amount of memory used.

Measurement “model”: Worst case, Average case and Best case.

Which algorithm is better?

The algorithms are correct, but which is the best?

- Measure the running time (number of operations needed).
- Measure the amount of memory used.
- Note that the running time of the algorithms increase as the size of the input increases.



Review: Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function call most statements roughly require the same amount of time
  - We can be more exact if need be
- Worst case vs. average case

What is Algorithm Analysis?

- How to estimate the time required for an algorithm
- Techniques that drastically reduce the running time of an algorithm
- A mathemactical framwork that more rigorously describes the running time of an algorithm