Course Outline

- Goals:
  1. Learn the commonly used data structures
  2. To learn how to measure the merits of these data structures
  3. To learn/reinforce the concept that every data structure has costs and benefits
- Methodology:
  - Algorithm analysis techniques
  - Algorithm design techniques
  - Implementation/analysis of data structures:
    - Lists, stacks, and queues
    - Trees
    - Hashing
    - Priority queues (heaps)
    - Sorting
    - Graphs

Why Study Data Structures?

Any organization for a collection of records can be searched, processed in any order, or modified.

- Data structures organize data:
  - Good choice: more efficient programs
  - Bad choice: poor program performance
    - The choice can make a difference between the program running in a few seconds or many days
- Justification: over time, there are
  - More powerful computers
  - More complex applications
  - Complex tasks unlike everyday experience
- Characteristics of a problem's solution
  - efficient: if it solves problem within resource constraints
    - time
    - space
  - Cost: amount of resources a solution consumes

Why Study Algorithms?

- Algorithms solve problems
  - Good choice: more efficient programs
  - Bad choice: poor program performance
  - Example:
    - Selection problem: find the largest k out of N integers
    - Easy algorithm: sort all integers, then list the first (or last) k
    - Better algorithm: sort first k then read through the list...
  - Different algorithms perform better on different inputs
  - Input *size* can also affect the performance

Algorithm Design Techniques

Most chapters focus on *implementation* of algorithms. The *design* of algorithms is also an important focus.

- Types of algorithms:
  - Greedy algorithms
  - Divide and Conquer
  - Dynamic programming
  - Randomized algorithms
  - Backtracking

Abstract Data Types

- Basic definitions:
  - **type**: a set of objects
  - **data item** or **element**: a piece of information or a record
  - **member**: a data item is said to be a *member* of a data type
  - **simple data item**: a data item containing no subparts.
  - **aggregate data item**: a data item that may contain several pieces of information
  - **abstract data type**: a type and a collection of operations to manipulate that type

    ADTs are mathematical abstractions; an ADT only mentions what is to be done, not how.

Data Structure

- A *data structure* is the physical implementation of an ADT.
  - Each ADT operation is implemented by one or more subroutines
  - Data structures are organizations for data in main memory
- File structures organize data on peripheral storage

Selecting a Data Structure

- Analyze the problem to determine its basic operations
- Quantify the resource constraints for each operation
- Select a data structure best meeting these requirements
- Some questions to consider:
  - At what time(s) in the program run do inserts occur
  - Are deletes allowed?
  - Is there any order to the data processing?

Algorithm/Data Structure Philosophy

- Each data structure requires:
  - space to store each item, including *overhead*
  - time to perform basic operations
  - programming effort
- Algorithms are closely related:
  - poor data structure choice can make *higher complexity* algorithm
  - good data structure choice can make the algorithm trivial

Problems, Algorithms, and Programs

What is the difference among these?

- Key questions that relate:
  - Can a problem be solved efficiently?
  - What is efficient?
  - Which algorithms are more efficient?
  - How to answer the above?
  - How to estimate the time required for a program
  - How to reduce the running time of a program
  - The consequences of careless use of recursion

Problems

- **Problem**: a task to be performed
  - One view: a set of inputs and matching outputs
  - Problem definition includes resource constraints
- Problems are analagous to mathematical functions
  - **Function**: mapping of inputs (*domain*) to outputs (*range*)
  - The input to a function can vary:
    - single number
    - multiple numbers
    - set of information
  - **Parameters**: the values making up an input
  - A given input must always map to the same output

Algorithms and Programs

Algorithm: a method or process followed to solve a problem

- Algorithm transforms the input of a problem to its output
- Algorithm properties:
  1. It must be correct
  2. It must be composed of a series of concrete steps
  3. There can be no ambiguity about which step is next
  4. It must be finite in length
  5. It must terminate
- **Program**: an instance of an algorithm, written in some programming language