Chapter 1

Data Structures and Algorithms
Primary Goals

- Present commonly used data structures
- Introduce the idea of tradeoffs; reinforce the concept of costs and benefits associated with every data structure
- Measure the effectiveness of a data structure or algorithm
Computer Programming Goals

- There are two, sometime conflicting:
  1. To design an algorithm that is easy to understand, code and debug
  2. To design an algorithm that make efficient use of the computer’s resources.
- This course is mainly concerned with goal number 2.
What is a data structure?

- In the most general, a data structure is any data representation and its associated operations.
- More typically, a data structure is meant to be an organization or structuring for a collection of data items.
Efficiency and Cost

- A solution to a problem is **efficient** if it solves the problem within the required resource constraints.
  - Examples: total space to store data, time allowed to perform.
- The **cost** of a solution is the amount of resources that the solution consumes.
  - Often the cost is measured in terms of one key resource.
Efficiency and Cost

- A solution can also be efficient if it requires fewer resources than other known solutions.
- When solving problems, you will want to analyze the different data structures available and choose accordingly.
- You want to make sure you choose both efficient and cost-effective solutions.
- Conversely, you don’t need a sledge hammer to put in a thumbtack.
Choosing your Data Structure

- You should follow these steps
  1. Analyze your problem to determine the resource constraints that any solution must meet
  2. Determine the basic operations that must be supported and quantify the resource constraints for each operation
  3. Select the data structure that best meets these requirements
Operational Questions

- When choosing data structures, here are some questions to ask.
  - Are all data items inserted into the data structure at the beginning, or are the insertions interspersed with other operations?
  - Can data items be deleted?
  - Are all data items processed in some well-defined order, or is search for specific data items allowed?
Abstract Data Types and Data Structures

- **Type** a collection of values
- A **simple type** has no subparts
- An **aggregate type** or **composite type** contains several pieces of information
- A **data item** is a piece of information or a record whose value is drawn from a type.
  - It is said to be a **member** of a type
- A **data type** is a type together with a collection of operations to manipulate the type.
There is a distinction between the logical concept of a data type and its physical implementation. Example: Trees can be implemented both in a linked fashion or an array based fashion.

- An **Abstract Data Type (ADT)** is the realization of a data type as a software component.
- An ADT does not specify how the data type is implemented.
Abstract Data Types and Data Structures

- A **data structure** is the implementation for an ADT
- ADT’s allow programmers to manage complexity through abstraction
- Data items have both a logical and a physical form
  - The definition of the data item in terms of an ADT is the logical form
  - The implementation of the data item within the data structure is its physical form