1. Overview of Data Structures

1.1. Introduction to data structures

- A *data structure* is an arrangement of data in a computer's memory or even disk storage.
- An example of several common data structures are
  - Arrays
  - Linked lists
  - Queues
  - Stacks
  - Binary trees
  - Hash tables
- *Algorithms*, on the other hand, are used to manipulate the data contained in these data structures as in searching and sorting.
- Many algorithms apply directly to a specific data structures. When working with certain data structures you need to know how to insert new data, search for a specified item, and deleting a specific item.

1.1. Characteristics of Data Structures

Data Structure	Advantages	Disadvantages
Array	Quick inserts Fast access if index known	Slow search Slow deletes Fixed size
Ordered Array	Faster search than unsorted array	Slow inserts Slow deletes Fixed size
Stack	Last-in, first-out acces	Slow access to other items
Linked List	Q Slow search u i	L Quick inserts i Quick deletes n
Queue	First-in, first-out access	Slow access to other items

1.1. Characteristics of Data Structures cont…

Data Structure	Advantages	Disadvantages
Binary Tree	Quick search Quick inserts Quick deletes *(If the tree remains balanced)*	Deletion algorithm is complex
Red-Black Tree	Quick search Quick inserts Quick deletes *(Tree always remains balanced)*	Complex to implement
2-3-4 Tree	Quick search Quick inserts Quick deletes *(Tree always remains balanced)* *(Similar trees good for disk storage)*	Complex to implement

1.1. Characteristics of Data Structures cont…

Data Structure	Advantages	Disadvantages
Hash Table	Very fast access if key is known Quick inserts	Slow deletes Access slow if key is not known Inefficient memory usage
Heap	Quick inserts Quick deletes Access to largest item	Slow access to other items
Graph	Best models real-world situations	Some algorithms are slow and very complex

NOTE: The data structures shown above (with the exception of the array) can be thought of as Abstract Data Types (ADTs).

1.1. Abstract Data Types

- An *Abstract Data Type* (ADT) is a way of looking at a data structure: focusing on what it does.
- A stack or a queue is an example of an ADT.
- It is important to understand that both stacks and queues can be implemented using an array.
- It is also possible to implement stacks and queues using a linked list.
- This demonstrates the "abstract" nature of stacks and queues: how they can be considered separately from their implementation.
- To best describe the term Abstract Data Type, it is best to break the term down into "data type" and then "abstract".

1.1. Abstract Data Types cont…

- **Data type**
  - Primitive data types refer to two things: a data item with certain characteristics and the permissible operations on that data.
    - Eg: A short in Java, can contain any whole number value from -32,768 to 32,767.
    - It can also be used with the operators +, -, *, and /.
  - The data type's permissible operations are an inseparable part of its identity; understanding the type means understanding what operations can be performed on it.

1.1. Abstract Data Types cont…

- Data type cont..
  - In Java, any class represents a data type, in the sense that a class is made up of data (fields) and permissible operations on that data (methods).
  - By extension, when a data storage structure like a stack or queue is represented by a class, it too can be referred to as a data type.
  - A stack is different in many ways from an int, but they are both defined as a certain arrangement of data and a set of operations on that data

1.1. Abstract Data Types cont…

- **Abstract**
  - In Java, an Abstract Data Type is a class considered without regard to its implementation. It can be thought of as a "description" of the data in the class and a list of operations that can be carried out on that data and instructions on how to use these operations.
  - What is excluded though, is the details of how the methods carry out their tasks.
  - End user (or class user), should be told what methods to call, how to call them, and the results that should be expected, but not how they work.

1.1. Abstract Data Types cont…

- **Abstract cont…**
  - Can further extend the meaning of the ADT when applying it to data structures such as a stack and queue. In Java, as with any class, it means the data and the operations that can be performed on it. In this context, although, even the fundamentals of how the data is stored should be invisible to the user.
  - Users not only should not know how the methods work, they should also not know what structures are being used to store the data.

1.1. Abstract Data Types cont…

- **The Interface**
  - The ADT specification is often called an *interface*.
  - It's what the user of the class actually sees.
  - In Java, this would often be the public methods. Consider for example, the stack class - the public methods push() and pop() and similar methods from the interface would be published to the end user.

1.2. Practical data storage structures

- Many of the structures and techniques considered here are about how to handle real-world data storage.
- By real-world data, we mean data that describes physical entities external to the computer.

  Examples: A personnel record describes a actual human being, an inventory record describes an existing car part or grocery item, and a financial transaction record describes, say, an actual check written to pay the electric bill.

1.2. Practical data storage structures Cont…

- A non-computer example of real-world data storage is a stack of index cards.
- These cards can be used for a variety of purposes. If each card holds a person's name, address, and phone number, the result is an address book. If each card holds the name location, and value of a household possession, the result is a home inventory.

1.3. Programmer’s Tools for data storage

- Not all data storage structures are used to store real-world data.
- Typically, real-world data is accessed more or less directly by a program's user.
- Some data storage structures, however, are not meant to be accessed by the user, but by the program itself.
- A programmer uses such structures as tools to facilitate some other operation.

Eg: Stacks, queues, and priority queues are often used in this way.

1.3. Real- world Modeling for data storage

- Some data structures directly model a real-world situation.
- The most important data structure of this type is the graph.
- You can use graphs to represent airline routes between cities, connections in an electrical circuit, or tasks in a project.
- Other data structures, such as stacks, queues, and priority queues, may also be used in simulations.
- A queue, for example, can model customers waiting in line at a bank.