CS-218 Data Structures and Algorithms

LECTURE 1
BY UROOJ AINUDDIN

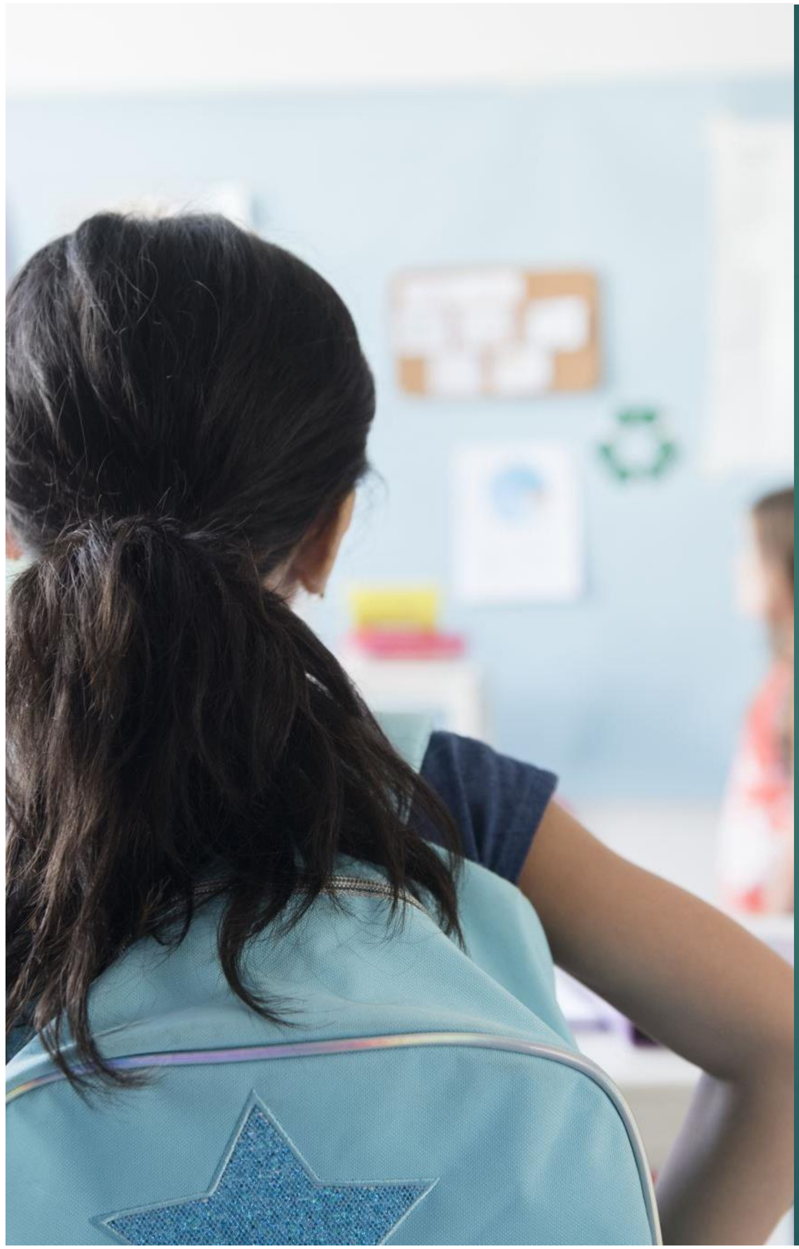
Fall Semester 2020

2nd Year Computer Systems Engineering 2019

NED University of Engineering and Technology

About Instructor

- Name: Urooj Ainuddin
- Designation: Assistant Professor CIS NEDUET
- Office: Research Lab
  - Extension: 2350
- Email: uroojain@neduet.edu.pk



Course Contents, Structure and Evaluation

THE WAYS AND MEANS TO HELP YOU UNDERSTAND WHAT DATA STRUCTURES AND ALGORITHMS ARE…

Course Plan

Topic	**Lectures**
Introduction and Motivation	1
Analysis of Algorithms	2
Asymptotic Analysis	3
Arrays and Lists	3
Stacks and Recursion	3
Queues	3
Linked Lists	3
Trees	6
Graphs	3
Sorting and Searching	3
Total lectures	30

Books

01

Rance D. Necaise, Data Structures and Algorithms Using Python, 1st edition, John Wiley and Sons, Inc., 2011

02

Mark Allen Weiss, Data structures and algorithm analysis in C++, 4th edition, Pearson, 2014

03

Larry Nyhoff, ADTs, Data Structures and Problem Solving with C++, 2nd edition, Pearson Prentice Hall, 2005

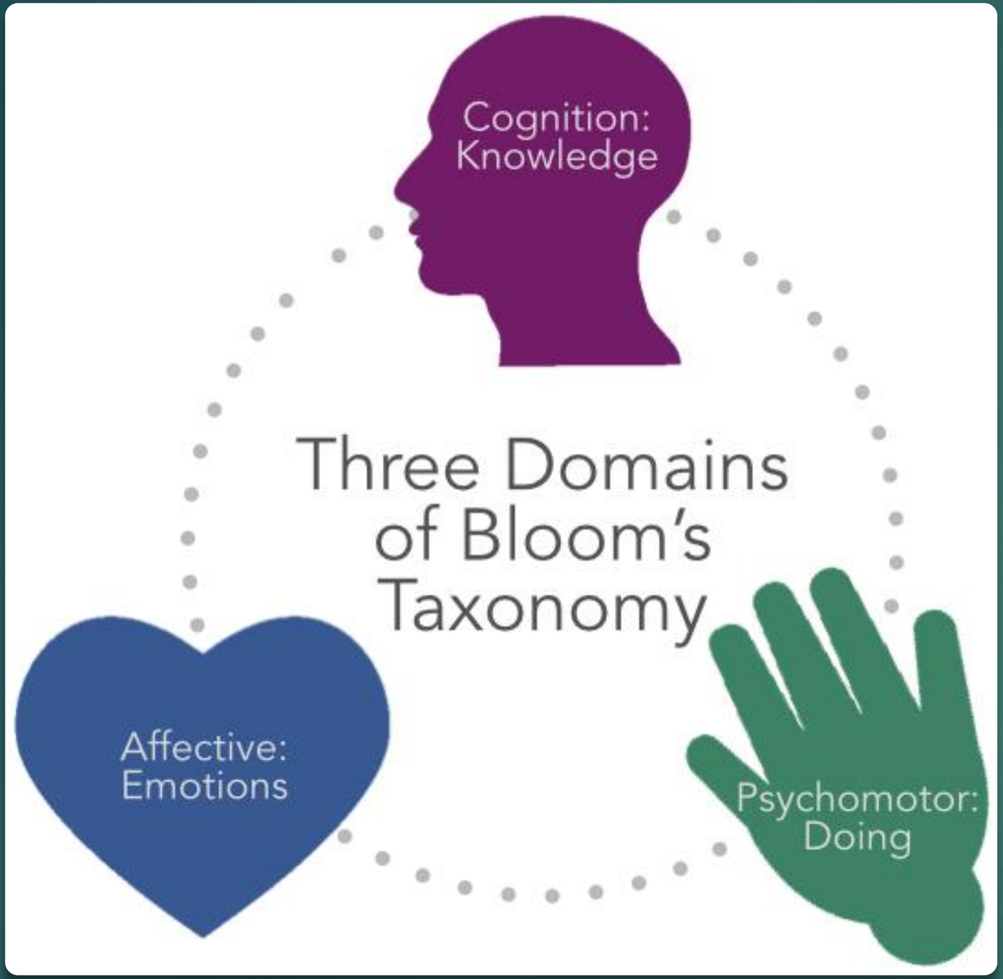
04

Seymour Lipschutz, Data Structures in C, 1st edition, Tata McGraw-Hill, 2011

About OBE

CIS FRAMEWORK

HTTPS://CIS.NEDUET.EDU.PK/OBE-SYSTEM



Learning domains

Levels of learning domains

Cognitive

- learners' ability to process information in a meaningful way
- **Categories:**
  - Knowledge
  - Comprehension
  - Application
  - Analysis
  - Synthesis
  - Evaluation

Affective

- learners' attitudes and feelings that are a result of the learning process
- **Categories:**
  - Receiving
  - Responding
  - Valuing
  - Organizing
  - Characterizing

Psychomotor

- learners' ability to use motor skills to learn
- **Categories:**
  - Perception
  - Set
  - Guided response
  - Mechanism
  - Complex overt response
  - Adaptation
  - Origination

Levels of the cognitive domain

Knowledge

- recognizing or remembering without necessarily understanding

Comprehension

- demonstrating understanding of facts and ideas by organizing, comparing

Application

- solving problems in new situations by applying acquired knowledge

Analysis

- examining and breaking information into component parts

Synthesis

- building a structure or pattern from diverse elements

Evaluation

- presenting and defending opinions by making judgments about information

Program Learning Outcomes (PLOs)

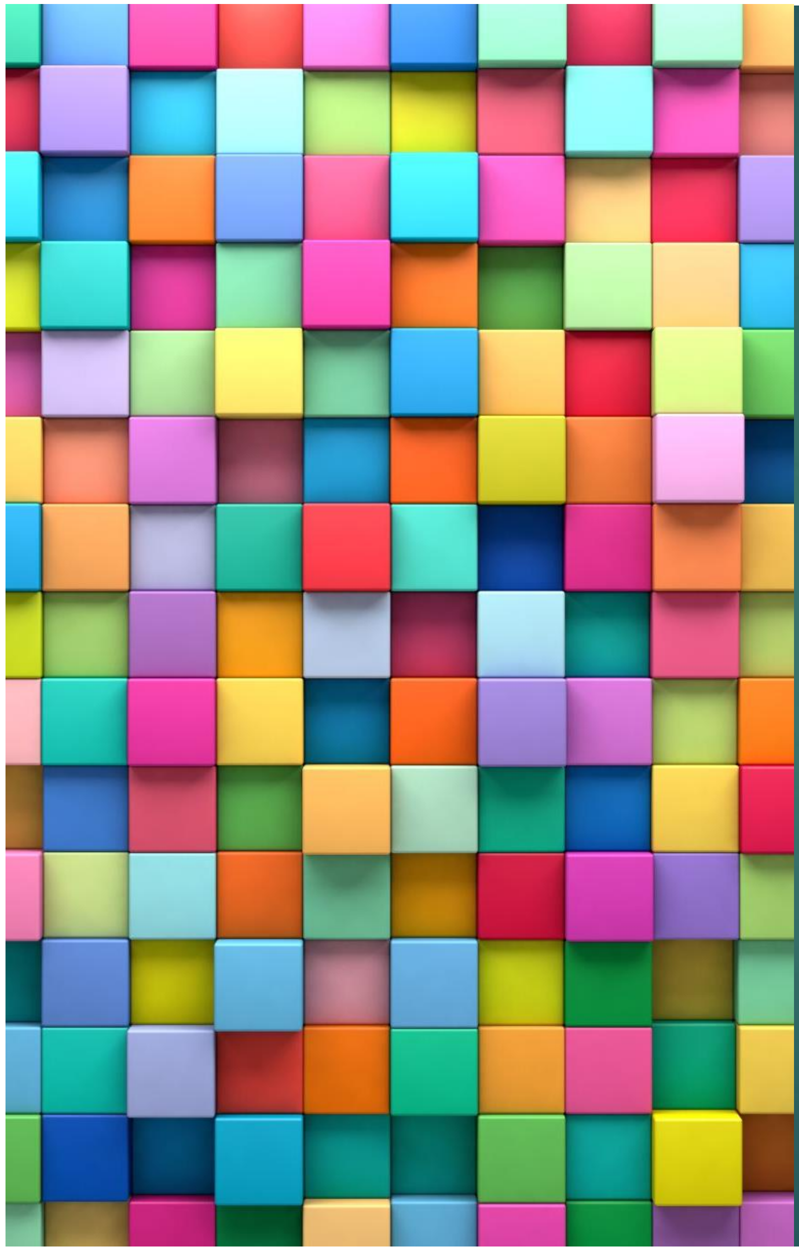
- PLO-1 Engineering Knowledge
- PLO-2 Problem Analysis
- PLO-3 Design/Development of Solutions
- PLO-4 Investigation
- PLO-5 Modern Tool Usage
- PLO-6 The Engineer and Society
- PLO-7 Environment and Sustainability
- PLO-8 Ethics
- PLO-9 Individual and Teamwork
- PLO-10 Communication
- PLO-11 Project Management
- PLO-12 Lifelong Learning

Course Learning Outcomes

Sr. No.	CLOs	Taxonomy level	Programme learning outcome (PLO)	Assessment Tool
1	**Elaborate** fundamental data structures	Cognitive C2	PLO-1 Engineering Knowledge	Quiz Assignment Midterm Final Exam
2	**Analyze** time and space complexity of algorithms	Cognitive C4	PLO-2 Problem Analysis	Quiz Assignment Midterm Final Exam
3	**Practice** with algorithms for widely used computing operations (Theory & Lab work)	Cognitive C3	PLO-3 Design / Development of Solutions	Rubrics for CEA Rubrics (for lab work) Midterm Final Exam

Why do we need this course?

- DSA helps us to:
  - Write fast and memory efficient algorithms.
  - Create optimized solutions to problems.



Introduction to DSA

BOOK 1 CHAPTER 1

Data types

- In programming languages, we are provided with a few different ways in which we can collect and characterize data. These ways are called **data types**.
- In the popular programming language, C, we have the *int* data type for storing integers, *float* for numbers with fractional values, *char* for alphabetical and special symbols. Each data type takes up fixed space in memory. An *int* data item takes up 2 bytes of memory. A *float* takes up 4 bytes, and a *char* takes up a single byte.
- The *int*, *float* and *char* data types are **simple data types** because they cannot be decomposed into smaller parts.
- **Complex data types** are constructed of multiple components consisting of simple types or other complex types. In Python, we see many complex data types like *list* and *dictionary*.

Algorithms

- An **algorithm** is a sequence of clear and precise step-by-step instructions for solving a problem in a finite amount of time.
- An algorithm is a sequence of steps that halts somewhere.

Abstraction

- An **abstraction** is a mechanism for separating the properties of an object and restricting the focus to those relevant in the current context.
- The user of the abstraction does not have to understand all of the details in order to utilize the object, but only those relevant to the current task or problem.
- Two common types of abstractions encountered in computer science are **procedural abstraction** and **data abstraction**.
- **Procedural abstraction** is the use of a function or method knowing what it does but ignoring how it's accomplished.
- **Data abstraction** is the separation of the properties of a data type (its values and operations) from the implementation of that data type.

Abstract data type

- An **abstract data type** (or **ADT**) is a programmer-defined data type that species a set of data values and a collection of well-defined operations that can be performed on those values.
- Abstract data types are defined independent of their implementation, allowing us to focus on the use of the new data type instead of how it's implemented.
- This separation is typically enforced by requiring interaction with the abstract data type through an interface or defined set of operations. This is known as **information hiding**.
- Using information hiding, we can work with an abstraction and focus on what functionality the ADT provides instead of how that functionality is implemented.

Operations on ADTs

- ADTs can be viewed like black boxes.
- The set of operations defined on an ADT can be grouped into four categories:
  - Constructors: Create and initialize new instances of the ADT.
  - Accessors: Return data contained in an instance without modifying it.
  - Mutators: Modify the contents of an ADT instance.
  - Iterators: Process individual data components sequentially.

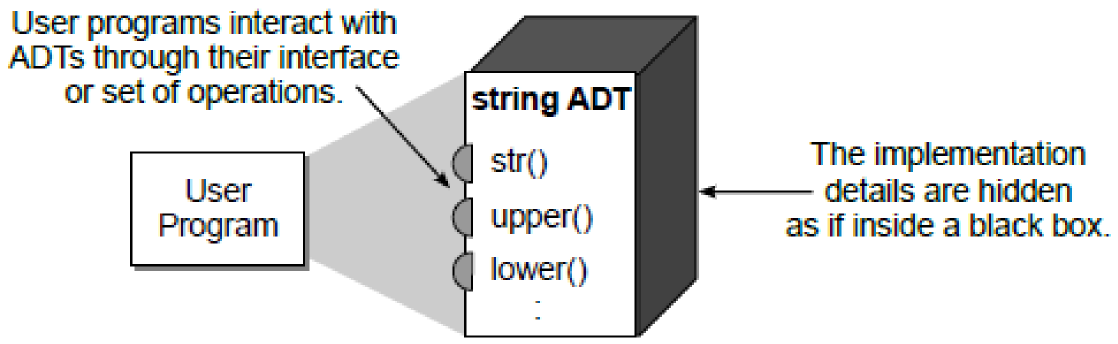

User programs interact with ADTs through their interface or set of operations.
User Program
string ADT
str()
upper()
lower()
:
The implementation details are hidden as if inside a black box.

Figure 1.2: Separating the ADT definition from its implementation.

Advantages of ADTs

- We can focus on solving the problem at hand instead of getting bogged down in the implementation details.
- We can reduce logical errors that can occur from accidental misuse of storage structures and data types by preventing direct access to the implementation.
- The implementation of the abstract data type can be changed without having to modify the program code that uses the ADT.
- It's easier to manage and divide larger programs into smaller modules, allowing different members of a team to work on the separate modules.

Types of ADTs

- Abstract data types can be **simple** or **complex**.
- A **simple ADT** is composed of a single or several individually named data fields such as those used to represent a date or a rational number.
- The **complex ADTs** are composed of a collection of data values such as the Python *list* or *dictionary*.
- Complex abstract data types are implemented using a particular data structure.

Data structures

- A **data structure** is a container for data.
- It helps to organize data.
- It helps to retrieve data.
- A suitable container is important for storage and retrieval.

Operations on data structures

Insertion	• Adding data to the data structure at a specified position.
Deletion	• Deleting data from the data structure at a specified position.
Traversal	• Moving through the whole data structure, visiting each data item exactly once.
Searching	• Finding the position of a specified data item in the data structure.
Sorting	• Ordering data items in the data structure.
Merging	• Joining two sorted data structures to create a single sorted data structure.

Insertion

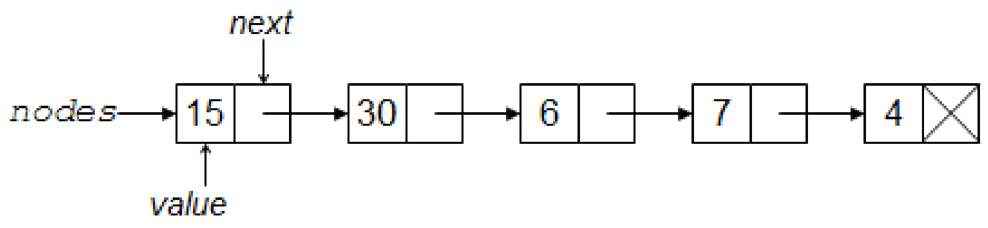
- We must never add data to a data structure that is already full and does not contain any empty space.
- Adding to a full data structure results in an error known as the **overflow**.
- Overflow leads to loss of data, as an old data item gets written over by a new data item.

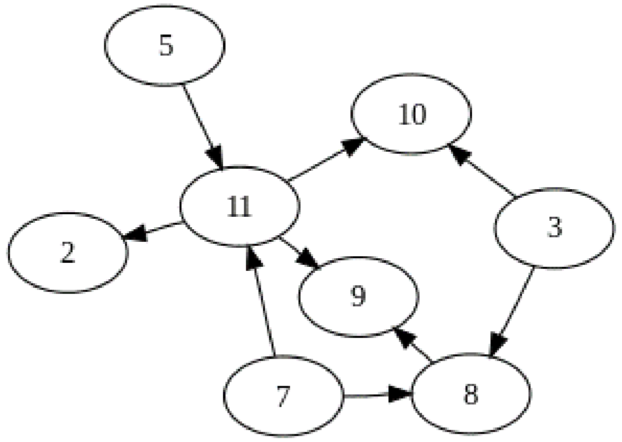
Deletion

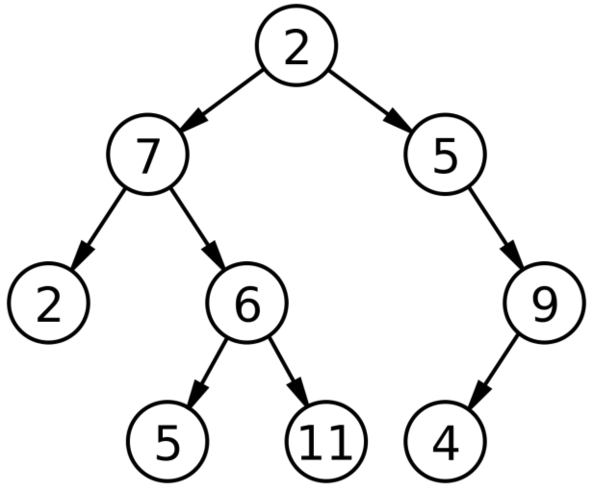
- Deletion of data from the data structure is retrieval of data.
- Deletion of data is done when the data item is required.
- We must never retrieve data from a data structure that is empty and does not contain any data.
- Deleting from an empty data structure gives us garbage values, which can never be useful.

Linearity

- Data structures may be of two kinds: **linear** and **non-linear**.
- If during traversal, at any position in a data structure, there is only one direction to move to, it is a linear data structure.
- In non-linear data structures, there exist one or more points at which, we have more than one directions to continue traversal.







Searching

- Searching is implemented by traversing the data structure.
- A search is **successful** when the data item we are looking for is present in the data structure. We return the location of this item to the calling program. We stop traversal when the data item is found.
- A search is **unsuccessful** when the item we are looking for is not in the data structure. We return **None** to the calling program.

Sorting

- Sorting is arranging items in an order. Following are some common orders:
  - Chronological
  - Alphabetical
  - Numeric
- Chronological ordering means arranging data items according to the time stamp attached to them.

Merging

- This operation can only be conducted on **two sorted** data structures.
- The challenge is to keep the sorted order intact in the final data structure.

Types of data structures

There are many common data structures, including **arrays**, **linked lists**, **stacks**, **queues**, and **trees**, to name a few.

All data structures store a collection of values but differ in how they organize the individual data items and by what operations can be applied to manage the collection.

The choice of a particular data structure depends on the ADT and the problem at hand.

Some data structures are better suited to certain problems.

Tasks

1. A mother compiles a baby book for her infant, in which she notes his important achievements. What order is she most likely to follow for the content?
2. You have 50 books. You can keep them in either a big bag, or in a cupboard with three shelves or in your elder brother's carefully catalogued library? Where will insertion be the easiest? Where will searching be the easiest? Where will sorting be the easiest?

Submit your response on a Google form in the assignment titled "Lecture 1 tasks" in Google Classroom.

So what did we learn today?

- We learned about the algorithm.
- We learned about abstraction and its types.
- We explored the abstract data type in detail.
- We were introduced to the idea of a data structure.
- We explored the operations performed on a data structure.
- We were introduced to different kinds of data structures.

Things to do

Read
- the book!

Submit
- your answers to the tasks in this lecture.

Note
- your questions and put them up in the relevant online session.

Email
- suggestions on content or quality of this lecture at uroojain@neduet.edu.pk