COMSATS University, Islamabad

Islamabad Campus

Department of Computer Science

Read before Attempt

Assignment No. 3: NON LINEAR DATA STRUCTURE	
Course code and Title: CSC211, Data Structure	
Instructor:	Atique Ahmed
Assigned Date: Nov 22, 2025	**Due Date: Dec 2, 2025**
Total Marks: --	
CLO-3(Theory): Use non-linear data structures to solve computing problems. **CLO -4(Lab): Implement data structures and algorithms.**	

Instructions:

1. This is an individual assignment. You will submit your work individually through your logins on MS Team
2. Try to get the concepts, consolidate your concepts and ideas from these questions
3. You should read **recommended books** to clarify your concepts as lecture slides are not sufficient.
4. **Try to make the solution by yourself and protect your work from other students. If I found the solution files of some students are same, then I will reward zero marks to all those students.**
5. Deadline for this assignment is **December 6, 2024.** This deadline will not be extended.

Performance Indicators for Every Assignment (Grading rubric)

Each assignment will be assigned a letter grade of either A, B, C, D, or F based on its submission.
This five-point (discrete) scale is described as follows:

Grade	Status	Description
A	Exemplary (=60-70%)	Solution presented solves the problem stated correctly and meets all require ments of the problem.
B	Capable (=50-60%)	Solution is mostly correct, satisfying most of the above criteria under the exemplary category, but contains some *minor* pitfalls, errors/flaws or limit ations.

C	Needs Improvement (=40-50%)	Solution demonstrates a viable approach toward solving the problem but contains some *major* pitfalls, errors/flaws or limitations.
D	Unsatisfactory (=20-40%)	-Critical elements of the solution are missing or significantly flawed and does not demonstrate sufficient understanding of the problem.
F	Not attempted	Late/no submission

To discourage copying/cheating, 30% marks may be awarded based on the marks obtained either in the corresponding quiz or a question(s) in the Midterm exam that will be used as multiplying factor. Hence, the final grades (out of 100) will be calculated as follows:

Submission marks (given by the letter grades shown above) + 30 * multiplying factor

Question # 1

PART – A: THEORY SECTION

A technology company, **TechTree Inc.**, has grown significantly and wants to digitally represent its hierarchical structure as a **Full Binary Tree** to optimize internal operations.

- The CEO is at the root of the hierarchy.
- Each Manager supervises **exactly two Team Leads**.
- Each Team Lead supervises **exactly two Developers**.
- This creates a **Full Binary Tree**, where every non-leaf node has exactly **two children**.

TechTree Inc. provides two data files for reconstructing the hierarchy:

1. **Preorder Traversal File**: Lists employees starting from the root (CEO) and traversing top-down, left-to-right.
   Example content: ["CEO", "Manager1", "TeamLead1", "Developer1", "Developer2", "TeamLead2", "Developer3", "Developer4", "Manager2", ...]
2. **Postorder Traversal File**: Lists employees based on project completion, traversing bottom-up, left-to-right.
   Example content: ["Developer1", "Developer2", "TeamLead1", "Developer3", "Developer4", "TeamLead2", "Manager1", "Manager2", "CEO"]

Q1 (Theory): Reconstruct Full Binary Tree from Traversals

(a) Write a **step-by-step algorithm** to reconstruct the **Full Binary Tree** from the given **preorder** and **postorder** traversals.
(b) Ensure your algorithm **enforces the Full Binary Tree property**, i.e., every node has either **0 or 2 children**.
(c) Explain how the reconstructed tree represents the **hierarchical structure** of TechTree Inc., including CEO, Managers, Team Leads, and Developers.

Note: Only algorithm design and theoretical explanation are required. **No coding** is needed in this part.

PART – B : LAB / PROGRAMMING SECTION

Q2 (Lab): C++ Program – Full Binary Tree Reconstruction

Write a **C++ program** that:

1. Reads the **preorder** and **postorder** traversal data of TechTree Inc.'s hierarchy.
2. Reconstructs the **Full Binary Tree** using the provided traversals.
3. Ensures the resulting tree satisfies the **Full Binary Tree property** (every non-leaf node has exactly 2 children).
4. Displays the reconstructed tree in **level-order traversal** or any clear tree format showing the hierarchy (CEO → Managers → Team Leads → Developers).

Requirements / Restrictions:

- Use **structs or classes** for tree nodes.
- Use **pointers** for left and right children.
- Clearly comment the code and provide **sample output** showing the hierarchy.

Question # 2

PART – A: THEORY SECTION

A real-estate management company, **HomeLink Solutions**, manages a large database of properties. Each node in their current **binary tree** represents a property with the following details:

- **Property ID**: A unique integer identifier for the property
- **Location**: City or state of the property
- **Price**: Current market price

The binary tree is organized to allow efficient searching based on **Property ID** (smaller IDs in the left subtree, larger IDs in the right subtree).

As the company scales, they want a simpler way to generate **sorted property listings** and perform **range queries** (e.g., find properties within a price range). To achieve this, they plan to **convert the binary tree into a Doubly Linked List (DLL)** following **in-order traversal**, so that the properties are arranged in **ascending order of Property ID**.

Q1 (Theory): Binary Tree to Doubly Linked List Conversion

(a) Write a **step-by-step algorithm** to convert the given binary tree into a **Doubly Linked List (DLL)** using **in-order traversal**.

(b) Explain how the algorithm maintains the **sorted order** based on **Property ID**.

(c) Describe how the DLL can be traversed in both **forward** and **reverse directions** to print all property details in ascending and descending order.

Note: Only algorithm design and theoretical explanation are required. No coding needed in this part.

PART – B : LAB / PROGRAMMING SECTION

Q2 (Lab): C++ Program – Binary Tree to Doubly Linked List

Write a **C++ program** that:

1. Builds the **binary tree** with properties (Property ID, Location, Price).
2. Converts the binary tree into a **Doubly Linked List (DLL)** using **in-order traversal**.
3. Ensures the **DLL maintains sorted order** based on **Property ID**.
4. Implements functions to **print the DLL**:
    - **Forward traversal**: Display all property details in ascending order of Property ID
    - **Reverse traversal**: Display all property details in descending order of Property ID

Requirements / Restrictions:

- Do **not** use STL list or pre-built DLL libraries; implement DLL manually using structs or classes.
- Use pointers for left/right or previous/next links.
- Include **sample outputs** showing forward and reverse traversals of the DLL.

Question # 3

PART – A: THEORY SECTION

A company processes real-time sales data. Each transaction contains the following fields: transactionID, productName, amount, and timestamp.

A subset of these transactions is stored in a **Binary Search Tree (BST)** using the **amount** as the key.

The BST is constructed using the following transaction amounts:

4500, 2200, 5000, 1800, 3000, 4700, 5100, 2600, 1900

Q1 (Theory): BST Deletion and AVL Conversion
Due to a customer refund and a manual correction, the transactions having amounts **3000** and **5000** must be deleted from the BST.

(a) Draw the **original BST** using the provided transaction amounts.
(b) Delete the nodes with **3000** and **5000**, and draw the **updated BST**.
(c) Convert the updated BST into a **balanced AVL Tree** (theory only).
Draw the final AVL Tree structure after applying the necessary rotations.

Note: AVL implementation in code is not required. Only theoretical diagrams are needed.

PART – B: LAB / PROGRAMMING SECTION
Q2 (Lab): BST Implementation in C++
Write a C++ program to:

- Build the BST using the given transaction amounts
- Delete the nodes with amounts **3000** and **5000**
- Display the BST before and after the deletion operations

Q3 (Lab): Max Heap and Descending Sort in C++
To analyze high-value transactions, the company wants the transaction amounts arranged in **descending order**.
Write a C++ program to:

- Build a **Max Heap** using the transaction amounts remaining after the BST deletions
- Perform **Heap Sort** to arrange the values in descending order
- Display the sorted transaction amounts

Restrictions:

- Do not use STL sort()
- Do not use priority_queue or STL heap functions
- Implement heap operations manually (heapify, insert, delete, heap-sort)

Question # 4

A city is planning the development of a **smart traffic management system**. The traffic control unit wants to model the network of intersections (nodes) and roads (edges) to optimize travel, connectivity, and cost. Each intersection is identified by a unique alphabetic ID (A-F), and the road between them has a cost (in minutes) that reflects congestion and signal delays. The aim is to ensure efficient communication and optimal routing for emergency vehicles.
The road network is as follows:

From	To	Time (minutes)
A	B	2
A	C	5
B	C	1
B	D	4
C	E	3
D	E	2
D	F	6
E	F	1

To deploy an emergency control update across all traffic intersections, the smart city system must send signals from intersection **A** to all others as **quickly and reliably** as possible.

PART – A : THEORY

4.1 Emergency Broadcast Order Using BFS and DFS

To deploy an emergency control update across all traffic intersections, the system must send signals from intersection **A** to all others as efficiently as possible.

(a) Apply **BFS** starting from A and show:
- Step-by-step queue updates
- Visited nodes at each iteration
- Final BFS visitation order

(b) Apply **DFS** starting from A and show:
- Step-by-step stack/recursive steps
- Visited nodes at each iteration
- Final DFS visitation order

(c) Based on your results, justify **which traversal (BFS or DFS)** is more suitable for emergency broadcast where intersections must be reached using *minimum number of edges*.
Give a clear explanation comparing both.

4.2 Minimum Spanning Tree Using Kruskal's Algorithm
Construct the MST for the road network using Kruskal's Algorithm.
Show all required steps:
- List all edges sorted by increasing weight
- For each edge, state whether it is accepted or rejected (cycle check)
- Show component/union-find updates
- Draw or list the final MST
- Calculate the total MST cost (sum of selected edge weights)

4.3 Shortest Travel Times Using Dijkstra's Algorithm
Apply Dijkstra's Algorithm starting from intersection **A**.
Show:
- Node extracted in each step
- Relaxation (update) steps for neighbors
- Distance table after each iteration
- Final minimum travel times from A to: B, C, D, E, and F

PART – B : LAB / PROGRAMMING (C++)

4.4 Implement BFS and DFS (Lab)
Write a C++ program to:
- Build the graph using an adjacency list
- Implement BFS and DFS starting from node A
- Print step-by-step states (queue/stack and visited lists)
- Display the final order for both BFS and DFS

Question # 5

PART – A : THEORY SECTION

You are planning a power distribution network to connect power stations to multiple consumption points. The goal is to **minimize the total distance of power lines**, which affects the overall cost of the network (including terrain and infrastructure).

The connections and distances between points are as follows:

From	**To**	**Distance (miles)**
A	C	15
B	C	20
B	D	25
C	E	30
D	E	35
E	F	40
E	G	45
F	G	50

Q1 (Theory): Graph Representation & Minimum Spanning Tree

(a) Generate the **Undirected Graph** from the above information. Clearly label nodes and edge weights.

(b) Using **Kruskal's Algorithm**, construct the **Minimum Spanning Tree (MST)**:

- List all edges sorted by increasing weight.
- Show step-by-step selection of edges: which edges are **accepted** or **rejected** (avoid cycles).
- Draw the **final MST** and calculate the **total distance** of the MST.

Deliverables:

- Graph diagram
- Edge list table (sorted by weight)
- Stepwise Kruskal's process
- MST diagram with total distance

PART – B : LAB / PROGRAMMING SECTION

Q2 (Lab): Kruskal's Algorithm Implementation in C/C++

Write a **C/C++ program** to:

1. Represent the network as an **undirected weighted graph** using an **edge list** or adjacency list.
2. Implement **Kruskal's Algorithm** to find the **Minimum Spanning Tree (MST)**.
3. Output:
   - Edges sorted by weight
   - Accept/reject decisions for each edge
   - Final MST edges and **total distance**

Requirements / Restrictions:

- Use **structs or classes** to represent edges and sets (for union-find).
- Implement **Union-Find (Disjoint Set)** for cycle detection.
- Include **comments** to explain key steps.
- Display **example output** showing MST edges and total cost.