❖ ASSIGNMENT NUMBER 1:

	Objective: The objective of this lab assignment is to implement fundamental data structures (Stack, Queue, List, Tree, Hash Table, Graph) and analyse their performance using basic algorithms.	**CO**	**BL**
	Lab Instructions: • Implement each data structure from scratch (without using built-in libraries like collections.deque or queue.Queue). • Demonstrate basic operations for each data structure. • Perform algorithm analysis using time complexity notation. • Provide a small test case for each implementation.		
Q.1	**Implement a Stack (LIFO)** 1. **Implement a stack using an array with the following operations:** • **push():** Insert an element. • **pop():** Remove and return the top element. • **peek():** Get the top element without removing it. • **is_empty():** Check if the stack is empty. 2. **Implement a stack using a linked list with the same operations.** 3. **Test Case:** • Push 5 elements onto the stack. • Pop 2 elements and print the stack state. • Display the final stack.	**CO1**	**BL1**
Q.2	**Implement a Queue (FIFO) and Circular Queue** 1. **Implement a queue using an array with the following operations:** • **enqueue():** Insert an element. • **dequeue():** Remove an element. • **front():** Get the front element. • **is_empty():** Check if the queue is empty. 2. **Implement a circular queue using an array.** 3. **Test Case:** • Enqueue 5 elements, then dequeue 2 elements. • Display the final queue state.	**CO1**	**BL1**

Q.3	**Implement a Singly and Doubly Linked List** 1. **Implement a singly linked list with the following operations:** • insert_at_head(), insert_at_tail(), delete_node(), search(), display(). 2. **Extend your implementation to a doubly linked list with prevpointers.** 3. **Test Case:** • Insert 3 elements at the head and 3 at the tail. • Delete one element and print the list forward and backward.	**CO1**	**BL1**
Q.4	**Implement a Binary Search Tree (BST)** 1. **Implement a Binary Search Tree(BST) with the following operations:** • insert(), delete(), search(), inorder(), preorder(), postorder(). 2. **Test Case:** • Insert 50, 30, 70, 20, 40, 60, 80. • Perform inorder, preorder, and postorder traversals. • Delete node 30 and print the updated tree.	**CO1**	**BL1**
Q.5	**Implement a Hash Table with Chaining** 1. **Implement a hash table using:** • A simple modulo-based hash function. • Chaining (linked list at each index) for collision resolution. 2. **Implement insert(), search(), and delete() operations.** 3. **Test Case:** • Insert 10, 20, 30, 40, 50 into a hash table of size 7. • Search for 30, then delete it and display the hash table.	**CO1**	**BL1**
Q.6	**Implement a Graph Using Adjacency List & Matrix** 1. **Implement a graph using:** • Adjacency Matrix representation. • Adjacency List representation. 2. **Implement Breadth-First Search and Depth-First Search .** 3. **Test Case:** • Create a graph with 5 vertices and edges (1,2), (1,3), (2,4), (3,5). • Perform BFS and DFS starting from vertex 1.	**CO1**	**BL1**
Q.7	**Implement and Compare Sorting Algorithms** • Implement Bubble Sort, Insertion Sort, and Merge Sort. • Generate an array of 10 random integers and sort them using each algorithm. • Compare the execution time of each sorting algorithm.		

Q.8	**Implement Dijkstra's Algorithm to find the shortest path in a weighted graph.** • Test it with a sample graph and show step-by-step execution		