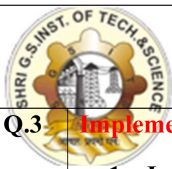




**ANALYSIS of DESIGN and ALGORITHM - LAB ASSIGNMENT**  
**MCA I year II Sem**

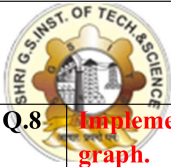
**ASSIGNMENT NUMBER 1:**

	<b>Objective:</b>  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
	<b>Lab Instructions:</b> <ul style="list-style-type: none"><li>• Implement each data structure from scratch (without using built-in libraries like collections.deque or queue.Queue).</li><li>• Demonstrate basic operations for each data structure.</li><li>• Perform algorithm analysis using time complexity notation.</li><li>• Provide a small test case for each implementation.</li></ul>		
<b>Q.1</b>	<b>Implement a Stack (LIFO)</b>  <b>1. Implement a stack using an array with the following operations:</b> <ul style="list-style-type: none"><li>• <b>push():</b> Insert an element.</li><li>• <b>pop():</b> Remove and return the top element.</li><li>• <b>peek():</b> Get the top element without removing it.</li><li>• <b>is_empty():</b> Check if the stack is empty.</li></ul> <b>2. Implement a stack using a linked list with the same operations.</b>  <b>3. Test Case:</b> <ul style="list-style-type: none"><li>• Push 5 elements onto the stack.</li><li>• Pop 2 elements and print the stack state.</li><li>• Display the final stack.</li></ul>	CO1	BL1
<b>Q.2</b>	<b>Implement a Queue (FIFO) and Circular Queue</b>  <b>1. Implement a queue using an array with the following operations:</b> <ul style="list-style-type: none"><li>• <b>enqueue():</b> Insert an element.</li><li>• <b>dequeue():</b> Remove an element.</li><li>• <b>front():</b> Get the front element.</li><li>• <b>is_empty():</b> Check if the queue is empty.</li></ul> <b>2. Implement a circular queue using an array.</b>  <b>3. Test Case:</b> <ul style="list-style-type: none"><li>• Enqueue 5 elements, then dequeue 2 elements.</li><li>• Display the final queue state.</li></ul>	CO1	BL1



**ANALYSIS of DESIGN and ALGORITHM - LAB ASSIGNMENT**  
**MCA I year II Sem**

<b>Q.3 Implement a Singly and Doubly Linked List</b>	<b>CO1</b>	<b>BL1</b>
<ol style="list-style-type: none"><li>1. Implement a singly linked list with the following operations:<ul style="list-style-type: none"><li>• insert_at_head(), insert_at_tail(), delete_node(), search(), display().</li></ul></li><li>2. Extend your implementation to a doubly linked list with prevpointers.</li><li>3. Test Case:<ul style="list-style-type: none"><li>• Insert 3 elements at the head and 3 at the tail.</li><li>• Delete one element and print the list forward and backward.</li></ul></li></ol>		
<b>Q.4 Implement a Binary Search Tree (BST)</b>	<b>CO1</b>	<b>BL1</b>
<ol style="list-style-type: none"><li>1. Implement a Binary Search Tree(BST) with the following operations:<ul style="list-style-type: none"><li>• insert(), delete(), search(), inorder(), preorder(), postorder().</li></ul></li><li>2. Test Case:<ul style="list-style-type: none"><li>• Insert 50, 30, 70, 20, 40, 60, 80.</li><li>• Perform inorder, preorder, and postorder traversals.</li><li>• Delete node 30 and print the updated tree.</li></ul></li></ol>		
<b>Q.5 Implement a Hash Table with Chaining</b>	<b>CO1</b>	<b>BL1</b>
<ol style="list-style-type: none"><li>1. Implement a hash table using:<ul style="list-style-type: none"><li>• A simple modulo-based hash function.</li><li>• Chaining (linked list at each index) for collision resolution.</li></ul></li><li>2. Implement insert(), search(), and delete() operations.</li><li>3. Test Case:<ul style="list-style-type: none"><li>• Insert 10, 20, 30, 40, 50 into a hash table of size 7.</li><li>• Search for 30, then delete it and display the hash table.</li></ul></li></ol>		
<b>Q.6 Implement a Graph Using Adjacency List &amp; Matrix</b>	<b>CO1</b>	<b>BL1</b>
<ol style="list-style-type: none"><li>1. Implement a graph using:<ul style="list-style-type: none"><li>• Adjacency Matrix representation.</li><li>• Adjacency List representation.</li></ul></li><li>2. Implement Breadth-First Search and Depth-First Search .</li><li>3. Test Case:<ul style="list-style-type: none"><li>• Create a graph with 5 vertices and edges (1,2), (1,3), (2,4), (3,5).</li><li>• Perform BFS and DFS starting from vertex 1.</li></ul></li></ol>		
<b>Q.7 Implement and Compare Sorting Algorithms</b>		
<ul style="list-style-type: none"><li>• Implement Bubble Sort, Insertion Sort, and Merge Sort.</li><li>• Generate an array of 10 random integers and sort them using each algorithm.</li><li>• Compare the execution time of each sorting algorithm.</li></ul>		



**ANALYSIS of DESIGN and ALGORITHM - LAB ASSIGNMENT**  
**MCA I year II Sem**

**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

