# CHENNAI INSTITUTE OF TECHNOLOGY

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**
**CHENNAI - 600 069**

B.E. / B.Tech. DEGREE END SEMESTER EXAMINATIONS
APR / MAY 2025
Second Semester
CS4202 – DATA STRUCTURES USING C++
(Common to CSE / IT / AIDS / CSBS / CSE-AIML / CSE-CZ)
(Regulations 2024)

Time: Three Hours Maximum Mark

Answer ALL Questions

RBT Level : L1- Remembering, L2 – Understanding, L3 – Applying, L4 – Analyzing, L5 – Evaluating, L6 – Cre

PART – A **(10x2=20 Marks)**

1. Define an Abstract Data Type (ADT), and illustrate its importance in the development of modular and reusable data structures.
2. Demonstrate how a circular linked list facilitates efficient memory management, and provide a real-time scenario to support its usage.
3. Differentiate between LIFO and FIFO principles by associating them with appropriate examples in computing systems.
4. Justify the need for queues in managing function calls, particularly in recursive and parallel processing environments.
5. Interpret the role of tree height in determining the computational complexity of common tree operations.
6. Identify the balancing condition maintained in an AVL tree and discuss how it affects the performance of dynamic data insertions.
7. Evaluate the role of indexing within data structures and assess its effect on data retrieval and storage performance.
8. Outline the structural requirements of a bi-connected graph and analyze their relevance in ensuring network stability.
9. Define sorting in the context of algorithm design and justify its necessity in optimizing data processing tasks.
10. Compare separate chaining and open addressing as collision resolution techniques in hash tables, highlighting their operational efficiency and memory usage.

11. a) i) Design and implement list structures using both array-based and singly linked list models. Perform insertion of three elements into each, then traverse and display the contents. Critically evaluate the internal mechanics of element addition in both approaches, focusing on memory allocation, time complexity, and structural flexibility.

ii) Deconstruct the architecture of a singly linked list and analyze how dynamic memory allocation impacts runtime efficiency, memory fragmentation, and scalability in large-scale applications.

(OR)

b) i) Construct a program that initializes a doubly linked list, inserts a new element at the beginning, and displays the resultant list. Assess the influence of bidirectional node connectivity on traversal, insertion, and deletion efficiency.

ii) Apply Radix Sort to the unsorted list: 170, 45, 75, 90, 802, 24, 2, 66. Clearly illustrate each pass of the algorithm.

12. a) i) Design and implement an algorithm that converts an infix expression to its equivalent postfix form using a stack. Demonstrate your solution with a trace for the expression: (A + B) * (C - D). Justify each step with reference to operator precedence and stack behavior.

ii) Analyse the differences between stack operations (push, pop) and queue operations (enqueue, dequeue). Provide an example for each.

(OR)

b) i) Compare and contrast the operations and usage of linear queues and circular queues. Illustrate with simple examples.

ii) Develop and test a program that implements a double-ended queue (DeQueue). Demonstrate the functionality by performing insertions and deletions from both the front and rear ends. Critically assess how DeQueues enhance flexibility in data processing compared to standard queue models.

13. a) i) Compare the traversal strategies of a binary tree—in-order, pre-order, and post-order—by demonstrating how each processes a common binary tree structure.

ii) Define the characteristics of a Binary Search Tree (BST) and design a program to construct a BST by inserting a set of user-defined elements. Implement an in-order traversal to display the sorted sequence. Justify how in-order traversal inherently reveals the BST's ordered structure.

(OR)

b) Define the structure and properties of a Binary Heap. Evaluate how min-heaps and max-heaps are utilized in implementing efficient priority queues. Develop a program to insert the elements 15, 10, 30, 5 into a min-heap and represent the heap structure visually (as a tree or array). Comment on the time complexity of insertion and heapify operations.

14. a) Delve into the concepts of Breadth-First Search (BFS) and Depth-First Search (DFS) in graph traversal. Compare their behavior in terms of memory usage, time complexity, and application suitability in various real-world scenarios, such as finding the shortest path or topological ordering. Provide examples where each traversal method is distinctly preferred.

(OR)

b) Define topological sorting in directed acyclic graphs (DAGs). Discuss its applications and why it is critical in scheduling problems, such as task scheduling in operating systems or compiling software. Write a program or pseudocode to perform a topological sort on a directed graph with 4 vertices, ensuring that each node is visited in the correct order as per DFS traversal.

15. a) You are given a sorted array of integers that has been rotated at some unknown pivot. The array was originally in increasing order but has been rotated. Your task is to implement a function that finds the index of a given target value in this rotated array using binary search. The key challenge is to optimize the binary search for a rotated sorted array, leveraging the properties of the rotation.
For example:
- Input: nums = [6, 7, 9, 15, 19, 2, 3], target = 15
- Output: 3 (since 15 is at index 3)

(OR)

b) i) Implement the Quick Sort algorithm to sort an array of distinct integers using a random pivot at each partitioning step. The algorithm should sort the array in-place (i.e., without using additional data structures except for the recursion stack).
Quick Sort Algorithm:
  - Implement the Quick Sort algorithm using a random pivot at each partitioning step.
  - The solution should be in-place, meaning no extra arrays or data structures (except for the recursion stack) are allowed.
  - After sorting the array, output the sorted array.

Additional Constraints:
  - The array size, n, can be as large as 10^6.
  - The solution must maintain an average-case time complexity of O (n log n) and space complexity of O (log n) due to the recursion stack.

Input:
- nums [] : An array of distinct integers of size n.

Output:
- The sorted array in ascending order.

Example 1:

Input:
- nums = [12, 3, 5, 7, 19, 6, 2].

Output:
- Sorted Array: [2, 3, 5, 6, 7, 12, 19].