Department of MCA

SYLLABUS of FY MCA 2024-25 (Autonomous)

SEMESTER-1

COURSE TITLE		DATA STRUCTURES & ALGORITHMS LAB		CREDITS	2
COURSE CODE	MCA102L	COURSE CATEGORY	SEC	L-P-T	0-4-0
Version	1.0	Approval Details	07-2024		

ASSESSMENT SCHEME			
Assignments	Internal Exam	Attendance	ESE
40%	40%	20%	--

Practical Assignment Questions

MODULE 1: Arrays

1: Dynamic Inventory Management System: Design an inventory management system for a warehouse using a **1-D array**. The array should store product details (ID, name, quantity, and price). Implement operations for adding new products, updating inventory levels, and calculating the total inventory value dynamically.

2: Real-time Seating Arrangement: Create a **2-D array** to represent seating arrangements for a movie theater. The system should handle seat booking and cancellation requests dynamically. Add constraints for group bookings, ensuring they are seated together.

3: Traffic Data Analysis: Collect real-time traffic data (e.g., car counts, speed, and entry/exit times) on multiple road lanes using **a dynamic array**. Analyze traffic flow patterns for peak hours using array operations like sorting, searching, and aggregation.

4: Multi-Dimensional Data Representation: Design a **multi-dimensional array** to represent geographical data (e.g., latitude, longitude, altitude) for a drone-based mapping system. Implement functions to extract and manipulate data for different regions dynamically.

MODULE 2: Linked List

1: Task Scheduler Simulation: Implement a **priority-based task scheduler** using a doubly linked list. Each task has a priority and duration. When new tasks are added, they should be inserted at the correct position based on priority. Simulate task execution and removal once completed.

2: Train Route Simulation (Circular Linked List): Simulate a train system where each station is represented as a node in a **circular linked list**. The train moves around the stations in a loop. Add operations to dynamically insert new stations or remove old ones, with proper memory management.

3: Real-Time Collaborative Editor: Build a **version control system** for a real-time collaborative text editor using a doubly linked list. Each edit is a node, allowing users to navigate forwards and backwards through changes. Include an undo/redo feature, tracking the position of multiple collaborators.

4: Browser Tabs Management: Implement browser tab management where each open tab is a node in a **singly linked list**. Users can open new tabs, close specific ones, and switch between them. Optimize the solution for handling thousands of open tabs efficiently.

5: Music Streaming Queue: Develop a **dynamic playlist system** where songs are represented as nodes in a **circular doubly linked list**. The user can queue songs, skip, replay, and move to the next song seamlessly. Ensure the playlist can handle thousands of songs with minimal memory overhead.

6: Memory Block Allocation (Garbage Collection): Simulate **dynamic memory block allocation and deallocation** using a **singly linked list**, where each node represents a memory block. Implement garbage collection to identify and release unreferenced blocks periodically

MODULE 3: Stacks and Queues

1: Expression Evaluation (Infix to Postfix Conversion): Implement a **calculator** that converts **infix expressions to postfix notation** using stacks. Evaluate the postfix expression to return the result. Handle complex expressions with parentheses and operator precedence efficiently.

2: Online Ticketing System (Priority Queue): Design an **online ticketing system** using a **priority queue** where VIP customers are served first. Regular customers are served based on their order of arrival. Simulate ticket booking, cancellation, and serve operations, ensuring the system works under heavy traffic conditions.

3: Undo-Redo Functionality for a Code Editor: Create an **undo-redo feature** using two stacks to track changes made in a code editor. As the user performs actions (e.g., writing, deleting text), track each action and allow them to undo or redo changes.

4: Job Queue System: Simulate a **job processing system** where jobs (like printing documents) are queued. Implement the queue with the ability to dynamically prioritize certain jobs (e.g., emergency print requests) using a **priority queue**.

5: Stock Span Problem: Solve the **Stock Span Problem** using a **stack**, where for each day's stock price, you calculate the number of consecutive days the price was less than or equal to today's price.

6: Bank ATM Queue Simulation: Implement a **bank ATM queue** where customers are queued for transactions. Simulate different types of transactions (deposit, withdrawal, balance check) with varying processing times. Use a **deque** (double-ended queue) to allow priority transactions at either end

MODULE 4: Trees and Graphs

1: Organizational Hierarchy Management System: Implement an **organization's hierarchy** using a **tree structure** where each node represents an employee. Simulate promotions, new hires, and removals dynamically, ensuring the tree stays balanced.

2: E-Commerce Recommendation System (Binary Search Tree): Build an **e-commerce recommendation system** where products are stored in a **binary search tree (BST)** based on customer ratings. Implement operations to find products within a specific rating range and suggest similar products.

3: Social Network Friend Recommendation (Graph): Use a **graph** to represent connections between users in a social network. Implement a **BFS algorithm** to suggest friend recommendations based on mutual connections.

4: Shortest Path in a City (Graph): Given a city represented as a **graph** with road networks (nodes for intersections, edges for roads), use **Dijkstra's algorithm** to find the shortest path between any two intersections.

5: File System Management (Tree): Simulate a **file system** where directories and files are stored in a **tree structure**. Implement operations like creating new files, deleting files, and listing files in different traversal orders (pre-order, post-order, in-order).

6: AVL Tree for Stock Price Management: Use an **AVL tree** to maintain stock prices. Ensure that after each insertion, the tree remains balanced by performing rotations.

7: Graph Coloring Problem (Greedy): Solve the **graph coloring problem** using a **greedy algorithm** to minimize the number of colors needed to color a graph such that no two adjacent nodes share the same color.

8: Minimum Spanning Tree for a Power Grid: Implement **Kruskal's algorithm** to find the **minimum spanning tree (MST)** for a power grid system connecting cities. Each city is a node, and each connection between cities has a cost.

9: Red-Black Tree for Dynamic Leaderboard: Implement a **red-black tree** to manage a dynamic gaming leaderboard. As players gain points, their rank in the tree adjusts in real time.

10: Cycle Detection in Graph: Implement a graph traversal algorithm (DFS) to detect cycles in a directed and undirected graph, simulating dependencies between software modules.

MODULE 5: Searching and Sorting

1: E-commerce Product Search with Binary Search: Implement a **binary search** algorithm to search for products in a sorted product catalog. Compare its performance against **linear search**.

2: Contact List Sorting (Merge Sort): Sort a large list of phone contacts using **merge sort** and compare the time complexity with **quick sort** when applied to smaller lists.

3: Event Ranking System (Heap Sort): Implement **heap sort** to rank participants in a large-scale competition based on their scores. Test your solution with large datasets.

4: Efficient Storage using Hash Tables: Design a **hash table** to store and retrieve employee records based on employee IDs. Implement different **hash functions** and collision handling techniques (chaining, open addressing).

5: Searching in a Rotated Sorted Array: Solve the problem of searching for a specific element in a **rotated sorted array** using a modified **binary search** algorithm.

6: Sorting a Music Library (Quick Sort): Implement **quick sort** to arrange songs in a music library by different parameters (duration, artist, genre). Optimize the algorithm for large datasets.

7: Caching using LRU Cache: Implement an **LRU (Least Recently Used) Cache** system using a combination of **hash maps** and **doubly linked lists** to store frequently accessed data efficiently.

8: Dictionary Implementation with Hashing: Create a **dictionary** where words are stored using a **hash table**. Implement efficient lookup, insertion, and deletion operations using custom **hash functions**.

9: Inventory Search using Interpolation Search: Implement an **interpolation search** algorithm for finding items in an inventory management system where the data distribution is uniform. Compare its performance with binary and linear search algorithms.

10: Sorting Patient Data in a Hospital: Design an algorithm to sort patient data based on emergency levels using **heap sort**. Ensure that the sorting happens in real-time for critical situations in an emergency room.

MODULE 6: Advanced Algorithms and Optimization

1: Delivery Route Optimization (Greedy Algorithm): Solve the **delivery route optimization** problem for a delivery service using a **greedy algorithm**. Minimize the total distance traveled by the delivery driver to deliver packages to multiple destinations.

2: Knapsack Problem (Dynamic Programming): Solve the **0/1 Knapsack Problem** using **dynamic programming**, where you are given a set of items, each with a weight and value, and must determine the most valuable combination that can fit within a weight limit.

3: Divide and Conquer Approach for Matrix Multiplication: Implement a **divide and conquer** algorithm (Strassen's algorithm) for **matrix multiplication**. Compare its performance with the standard matrix multiplication algorithm for large matrices.

4: Approximation Algorithms for NP-Complete Problems: Implement an **approximation algorithm** for solving the **traveling salesman problem**. Analyze how close the solution is to the optimal path and discuss the complexity of the algorithm