INSTITUTE OF ENGINEERING & MANAGEMENT, SALT LAKE CAMPUS
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Course Title - Data Structure & Algorithms Lab
Course Code - BTHPCCCS391

ASSIGNMENT LIST

ARRAY

1. You are given two sorted arrays, arr1 and arr2, each of size m and n, respectively. Design a function to merge these two arrays into a single sorted array without using extra space.
2. You are given an array containing n-1 unique integers in the range of 1 to n. One integer is missing from the array. Design a function to find the missing number.
3. Implement a program that allows the user to perform various operations on an array:
    a. Create an array of size n and populate it with elements.
    b. Insert an element at a specified position in the array.
    c. Delete an element from a specified position in the array.
    d. Search for an element in the array and return its index.
    e. Find the maximum and minimum elements in the array.
    f. Display the array.
4. Given an array of integers, design a program to find if there exists a subarray with a given sum. If such a subarray exists, print the starting and ending indices of the subarray. If multiple subarrays with the same sum exist, display any one of them.
5. Given an array of integers, write a program to rearrange the elements such that all positive numbers come before negative numbers while maintaining the relative order of positive and negative numbers in the original array.

CASE STUDY BASED PROBLEM:

6. You are tasked with implementing an inventory management system for a retail store. The system should handle adding, updating, and querying products in the inventory using arrays.

Lab Assignment Tasks:

a. Implement a function to add a new product to the inventory.
b. Implement a function to update the quantity of a product based on its ID.
c. Implement a function to find a product by its ID and return its details.
d. Implement a function to calculate the total value of the inventory (the sum of the price * quantity for all products).

STACK and QUEUE

7. Design a stack data structure using two queues. Implement the push, pop, and top operations for the stack.
8. Design a queue data structure using two stacks. Implement the enqueue, dequeue, and front operations for the queue.
9. Write a program to reverse the order of elements in a given queue using a stack. The program should take a queue as input and output the reversed queue.
10. You are given a string containing parentheses, curly brackets, and square brackets. Design a function to check whether the given string has balanced parentheses. The string is considered balanced if each opening parenthesis is closed by the correct corresponding closing parenthesis.
11. Implement a function to convert an infix expression to a postfix expression using stacks. The infix expression contains operands, operators, and parentheses. The output should be a postfix expression with the same order of operations as the original expression.
12. Design a function to evaluate a postfix expression using a stack. The postfix expression contains operands and operators. The function should return the result of the expression.

CASE STUDY BASED PROBLEM:

13. Given an array of integers, design a function to find the next greater element for each element in the array. The next greater element for an element is the first greater element to its right. If no greater element exists, output -1 for that element.
14. Given an array of integers, design a function to find the next greater element for each element in the array. The next greater element for an element is the first greater element to its right. If no greater element exists, output -1 for that element.

LINKED LIST

15. Implement a singly linked list data structure that supports the following operations:
    a. Insert an element at the beginning of the list.
    b. Insert an element at the end of the list.
    c. Delete an element from the list.
    d. Search for a given element in the list.
    e. Display the elements of the list.
16. Write a program to reverse a given singly linked list. Implement two versions of the algorithm - one using an iterative approach and another using a recursive approach.
17. Write a program to check if a given singly linked list is a palindrome. A linked list is a palindrome if the elements read the same backward as forward.

CASE STUDY BASED PROBLEM:

18. Given a singly linked list, design a function to detect if there is a cycle in the list. If a cycle is present, find the starting node of the cycle.
19. You are given two sorted singly linked lists. Implement a function to merge these two lists into a single sorted list.

TREE and GRAPH

20. Implement a Binary Tree data structure that supports the following operations:
    a. Insertion of a new node.
    b. Deletion of a node.
    c. Search for a key in the tree.
    d. Finding the minimum and maximum elements in the tree.
21. Implement a Binary Search Tree (BST) data structure that supports the following operations:
    a. Insertion of a new node.
    b. Deletion of a node.
    c. Search for a key in the tree.
    d. Finding the minimum and maximum elements in the tree.
    e. Traversals: In-order, Pre-order, and Post-order.
22. Given two binary trees, design a function to check if they are structurally identical, meaning they have the same structure and corresponding nodes have the same values.
23. Given a sorted array, design a function to construct a Balanced Binary Search Tree (BST) from it.
24. Write functions to calculate the height of the AVL tree and the balance factor of each node in the tree. The balance factor of a node is the difference between the heights of its left and right subtrees.
25. Implement a threaded binary search tree (TBST) data structure that supports insertion, deletion, in-order traversal, and finding the in-order successor and predecessor of a given node. Ensure the TBST maintains the binary search tree property and the threaded connections.

CASE STUDY BASED PROBLEM:

26. Implement a graph data structure that supports both directed and undirected graphs. Implement functions for:
    a. Adding a vertex to the graph.
    b. Adding an edge between two vertices.
    c. Performing depth-first search (DFS) traversal of the graph.
    d. Performing breadth-first search (BFS) traversal of the graph.
27. Given a weighted undirected graph, design a function to find the Minimum Spanning Tree (MST) using Kruskal's or Prim's algorithm.

SORTING and SEARCHING

28. Implement a linear search algorithm to find the position of a given element in an array. If the element is present multiple times, the function should return the positions of all occurrences.
29. Implement a binary search algorithm to find the position of a given element in a sorted array. If the element is present multiple times, the function should return the position of any occurrence.
30. You are given a 2D matrix sorted row-wise and column-wise. Implement a function to search for a given element in the 2D array efficiently.
31. Implement the Bubble Sort algorithm in your preferred programming language. Test it with various arrays of integers to verify its correctness.
32. Write a function to implement the Insertion Sort algorithm in your preferred programming language. The function should take an array of integers as input and sort it in ascending order using the Insertion Sort technique.

CASE STUDY BASED PROBLEM:

33. Implement one or more other sorting algorithms, such as Bubble Sort, Insertion Sort, Merge Sort, Selection Sort, Quick Sort and Radix sort. Write functions for each of these sorting algorithms and measure their execution time using the same set of input arrays. Generate arrays of random integers with varying lengths, starting from small arrays (e.g., 100 elements) up to reasonably large arrays (e.g., 10,000 elements or more). Use a suitable method to record the execution time for each input size.
34. Create a class representing a custom object (e.g., a Person with attributes like name, age, and address). Implement a sorting algorithm that can sort a list of these custom objects based on different attributes (e.g., sorting by name, age, or address). Test the sorting algorithm with a list of objects and verify that the objects are sorted correctly.

HASHING

35. Implement a Hash Table data structure in your preferred programming language. The Hash Table should support basic operations such as inserting a key-value pair, retrieving the value associated with a given key, and deleting a key-value pair. Handle collisions using a suitable collision resolution method, such as chaining (linked lists) or open addressing (linear probing, quadratic probing, or double hashing).

CASE STUDY BASED PROBLEM:

36. Create test cases to verify the correctness of your Hash Table implementation with quadratic probing. Test it with various scenarios, including inserting, retrieving, and deleting elements, handling collisions, and resizing the Hash Table when it reaches a load factor threshold. Include test cases for edge cases, such as an empty table, a table with only one element, and situations where the table becomes full.