Data Structures Question Paper

Arrays

1. Write a C program to find the longest increasing subsequence in an array of integers. The program should return both the length of the subsequence and the elements that form it.

2. Implement a function that takes two sorted arrays and merges them into a single sorted array without using any extra space (in-place merge).

3. Given an array of integers, write a program to find the maximum sum of a contiguous subarray (Kadane's algorithm). Extend the program to handle circular arrays as well.

4. Create a function that rotates a 2D square matrix by 90 degrees clockwise without using any extra space.

5. Implement a sparse matrix using a compressed row storage (CRS) format. Write functions to perform matrix addition and multiplication efficiently.

Stacks

1. Implement a min-stack data structure that supports push, pop, top, and retrieving the minimum element in constant time.

2. Write a program to evaluate a postfix expression using a stack. The program should handle multi-digit numbers and the operators +, -, *, /, and ^.

3. Implement a stack that can keep track of its maximum element at any point. All operations (push, pop, and get_max) should have O(1) time complexity.

4. Create a program that uses two stacks to implement a queue data structure efficiently.

5. Write a function to determine if a given sequence of parentheses, brackets, and braces is balanced using a stack.

Queues

1. Implement a circular queue using an array. The program should efficiently handle queue overflow by resizing the array when necessary.

2. Create a double-ended queue (deque) data structure that supports insertion and deletion at both ends in O(1) time.

3. Implement a priority queue using a binary heap. The program should support insertion, deletion, and retrieval of the highest priority element efficiently.

4. Write a program to simulate a call center using a queue. The program should handle customer calls, assign them to available representatives, and maintain wait times.

5. Implement a sliding window maximum algorithm using a deque. Given an array and a window size, find the maximum element in each window as it slides through the array.

Linked Lists

1. Implement a skip list data structure for efficient searching in a sorted linked list. Include functions for insertion, deletion, and search operations.

2. Write a program to detect and remove cycles in a linked list. The program should handle both self-loops and larger cycles efficiently.

3. Create a function to perform an in-place merge sort on a singly linked list.

4. Implement a polynomial addition function using linked lists to represent polynomials. Each node should contain the coefficient and exponent of a term.

5. Write a program to reverse a linked list in groups of K nodes. If the list length is not a multiple of K, the remaining nodes should be left as is.

6. Create a function to find the intersection point of two linked lists. The function should handle cases where the lists may or may not intersect.