Lab Exercise-3

Lab exercise (C/C++):

1. Implement Tower of Hanoi on Stack using a Linked list

2. Implement Parenthesis Matching on Stack using a Linked list.

3. Implement a Binary Tree and perform Insertion, Deletion, and Search operations:
   3.1. Arrays
   3.2. Linked data structure

Note: Delete a node by making sure that the tree shrinks from the bottom (i.e., the deleted node is replaced by the bottom-most and rightmost node)

4. Accepts input to create a linked list, then traverses the list and finds elements at specific positions. The positions should be dynamically determined based on the total length of the list, without explicitly counting the number of nodes (n). The program should find the elements at positions n/2, n/3, n/4, and so on, until n/k, where k is the total number of nodes in the linked list. The program should output the elements at these positions. Important constraints:

- No explicit counting of n is allowed.
- Only one loop (single traversal) is permitted.

5. Taking input expression from the user and perform Infix to Postfix Conversion: Convert an infix expression (e.g., A + B * (C - D)) to its postfix equivalent (e.g., A B C D - * +) using a stack.

6. Taking input expression from the user and performPostfix to Infix Conversion: Convert a postfix expression (e.g., ABC*D-+) back to its infix form (e.g., A + B * (C - D)) using a stack.

7. Implement the merge sort algorithm and print the sorted order (increasing) along with the number of comparisons and swaps.

Note: Optionally students can practice using large files of inputs having thousands to millions of numbers and compare the time taken by the algorithm (Q6).

8. To simulate a Ticket Booking System where customers can book tickets for an event. The system uses a circular queue to manage ticket requests. Customers are allocated tickets in the order of their arrival, and the system handles wraparound when the queue reaches its maximum capacity.

Specifications:

1. Queue Management: Use a circular queue to manage customer ticket requests.
2. Ticket Booking: When a customer requests a ticket, the system should allocate a ticket and display the customer's ID.
3. Queue Overflow Handling: If the queue is full, the system should inform the user that no more customers can be added until space is available.
4. Queue Underflow Handling: If there are no customers in the queue when attempting to allocate a ticket, the system should inform the user.
5. Circular Queue: Ensure that when the queue is full, and customers are removed, the queue continues to work efficiently by overwriting the earliest requests.
6. Simulate Ticket Allocation: Simulate the booking process where customers are served in the order they request tickets.