1. Implement a Stack using Array

Implement a stack that supports:

- `push(x)` → add element
- `pop()` → remove top element
- `peek()` → view top element
- `isEmpty()` → check if empty

Handle **stack overflow** and **underflow** conditions.

2. Implement a Stack using Linked List

Build a stack using linked nodes instead of arrays.
Each node should contain `data` and `next`.
Perform push, pop, and display operations.

3. Reverse a Stack using Recursion (No Loops)

Reverse a given stack using recursion only (no arrays or loops).

Example:
Input: `[1, 2, 3, 4]`
Output: `[4, 3, 2, 1]`

4. Sort a Stack using Recursion

Sort a stack in ascending order using only **stack operations** and **recursion**. No loops or extra stack allowed.

Example:
Input: `[3, 5, 1, 4, 2]`
Output: `[1, 2, 3, 4, 5]`

5. Delete Middle Element of a Stack

Remove the middle element of a stack without using another data structure.

Example:
Input: [1, 2, 3, 4, 5]
Output: [1, 2, 4, 5]

Hint: Use recursion to pop until the middle is reached, then rebuild.

6. Check for Balanced Parentheses

Check if every opening bracket has a matching closing bracket.

Example:
Input: {[()()]} → Balanced
Input: {[(])} → Not Balanced

7. Next Greater Element

Find the next greater element for each element in an array. If no such number exists, use -1.

Example:
Input: [4, 5, 2, 10]
Output: [5, 10, 10, -1]

8. Check if Stack Elements are Palindrome

Check if the elements in a stack form a palindrome when read from top to bottom.

Example:

Input: [1, 2, 3, 2, 1]

Output: Palindrome