Week 3 DSA: Stack & Queue 📚

1. Fundamentals

Stack (LIFO - Last In, First Out)

Imagine a stack of plates; you add to the top and take from the top.

- **Push:** Add an element to the top.
- **Pop:** Remove the top element.
- **Peek/Top:** View the top element without removing it.
- **IsEmpty:** Check if the stack is empty.
- **Time Complexity:** All basic operations are $O(1)$.

Real-world Applications:

- Undo/Redo operations in editors.
- Expression evaluation (Infix to Postfix).
- Backtracking (e.g., finding a path in a maze).
- Function call stack in recursion.

Queue (FIFO - First In, First Out)

Imagine a line at a ticket counter; the first person to enter is the first to leave.

- **Enqueue:** Add an element to the rear (end).
- **Dequeue:** Remove an element from the front.
- **Front:** View the first element.
- **Rear:** View the last element.
- **Time Complexity:** All basic operations are $O(1)$.

Real-world Applications:

- Task scheduling (CPU scheduling, IO buffers).
- Breadth-First Search (BFS) in graphs/trees.
- Handling website traffic (Load balancing).

2. LeetCode Problem Breakdown

Problem 1: Valid Parentheses (LeetCode #20)

Goal: Given a string containing just the characters `(`, `)`, `{`, `}`, `[` and `]`, determine if the input string is valid.

Logical Approach (Using Stack):

1. Initialize an empty stack.
2. Traverse the string character by character:
   - If it's an **opening bracket**, `push` it onto the stack.
   - If it's a **closing bracket**:
     - Check if the stack is empty. If yes, it's invalid.
     - `pop` the top element and check if it matches the current closing bracket. If not, it's invalid.
3. After the loop, if the stack is empty, return `true`; otherwise, return `false`.

Key Insight: Stacks are perfect for matching pairs because the last opened bracket must be the first one closed.

Problem 2: Implement Queue using Stacks (LeetCode #232)

Goal: Implement a FIFO queue using only two LIFO stacks.

Logical Approach:

- **Stack 1 (input):** Used for pushing new elements.
- **Stack 2 (output):** Used for popping/peeking elements.

Operations:

1. **Push:** Simply `push` the element into `Stack 1`.
2. **Pop/Peek:** - If `Stack 2` is empty, `pop` all elements from `Stack 1` and `push` them into `Stack 2`. This reverses the order, making it FIFO.
   - `pop` / `peek` from `Stack 2`.

Efficiency: While a single `pop` might take $O(n)$ in the worst case (when moving elements), the **amortized** time complexity is $O(1)$.

3. Comparison Summary

Feature	Stack	Queue
Principle	LIFO (Last In, First Out)	FIFO (First In, First Out)
Main Ops	Push / Pop	Enqueue / Dequeue
Pointers	Only Top	Front and Rear
Key Use	Recursion, Reversal	Buffering, Scheduling