Queue – Queue Using Stack

1. Introduction

A **Queue Using Stack** is an implementation of the queue data structure using **two stacks** instead of a direct queue structure.

Even though stacks follow **LIFO (Last In, First Out)**, a queue can be implemented by carefully arranging stack operations to achieve **FIFO (First In, First Out)** behavior.

This concept demonstrates how one data structure can be implemented using another.

---

2. Why Implement Queue Using Stack?

- To understand the flexibility of data structures
- To practice stack operations deeply
- Commonly asked in **interviews and exams**
- Shows how LIFO can simulate FIFO

---

3. Basic Idea

To implement a queue using stacks:

- Use **two stacks**:
  - **Stack 1 (Input Stack)**
  - **Stack 2 (Output Stack)**

By transferring elements between the two stacks, the order of elements is reversed, enabling FIFO behavior.

---

4. Principle Used

- Stack → LIFO

- Queue → FIFO

Using **two reversals**:

```
FIFO = Reverse(LIFO(Reverse(LIFO)))
```

---

5. Operations Supported

Queue using stack supports:

1. **Enqueue**
2. **Dequeue**
3. **Peek**
4. **isEmpty**

---

6. Enqueue Operation (Plain English)

Goal: Add element to the queue

Logic

1. Push the element into **Stack 1**
2. No other operation is needed

This is a simple and fast operation.

---

7. Dequeue Operation (Plain English)

Goal: Remove the first inserted element

Logic

1. If **Stack 2 is not empty**:
    - Pop from Stack 2
2. If **Stack 2 is empty**:
    - Move all elements from Stack 1 to Stack 2

- Pop the top element from Stack 2

3. If both stacks are empty:
    - Queue underflow

---

8. Peek Operation

Logic

1. If Stack 2 is not empty:
    - Return the top element of Stack 2
2. Else:
    - Move elements from Stack 1 to Stack 2
    - Return the top of Stack 2

---

9. isEmpty Operation

Queue is empty when:

```
Stack 1 is empty AND Stack 2 is empty
```

---

10. Step-by-Step Example

Operations:

```
Enqueue 10
Enqueue 20
Enqueue 30
Dequeue
```

Stack State

```
Stack 1: 10 20 30
Stack 2: (empty)
```

During dequeue:

```
Move all from Stack 1 → Stack 2
Stack 2: 30 20 10
Pop → 10
```

Queue behaves correctly as FIFO.

---

11. Visualization

```
Enqueue → Stack 1
Dequeue → Stack 2
```

Elements are reversed once during transfer, preserving queue order.

---

12. Time and Space Complexity

Operation	Time Complexity
Enqueue	O(1)
Dequeue	O(n) (worst case)
Peek	O(n) (worst case)

- **Space Complexity:** O(n)

---

13. Advantages of Queue Using Stack

- Demonstrates strong understanding of data structures
- Useful for interview questions
- No need for direct queue structure
- Flexible design

14. Limitations

- Slower dequeue and peek operations
- Uses extra memory
- More complex than direct queue implementation

15. Applications

- Algorithm design practice
- Interview problem solving
- Understanding data structure transformations
- Academic learning

16. Comparison with Queue Using Array

Feature	Queue Using Stack	Queue Using Array
Implementation	Complex	Simple
Space	Extra stacks	Fixed array
Performance	Slower dequeue	Faster
Flexibility	High	Limited

17. Summary

- Queue using stack uses two stacks
- Enqueue is simple push
- Dequeue uses stack transfer
- FIFO achieved using LIFO stacks
- Time complexity varies