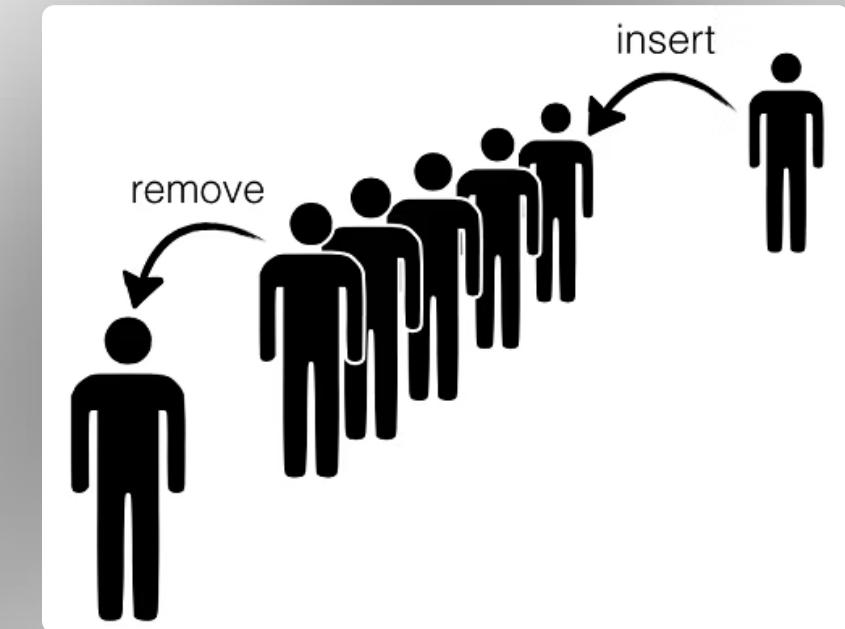


# DSA Session: Queue, Circular Queue & Deque



# Understanding the Queue

## FIFO Principle

A linear data structure where elements are inserted from the rear and deleted from the front, following a First-In, First-Out order.

## Real-Life Analogy

Imagine a queue at a ticket counter – the first person in line is the first to be served.

## Key Operations

Enqueue (add), Dequeue (remove), Front (peek first), Rear (peek last), isEmpty, isFull.

## Efficiency

Operations like Enqueue and Dequeue have a time complexity of  $O(1)$ , making them very fast.



# Circular Queue: Optimizing Space

## Space Reuse

Addresses wasted space in simple array-based queues by connecting the rear to the front.

## O(1) Efficiency

Maintains constant time complexity for Enqueue and Dequeue operations.



## Circular Logic

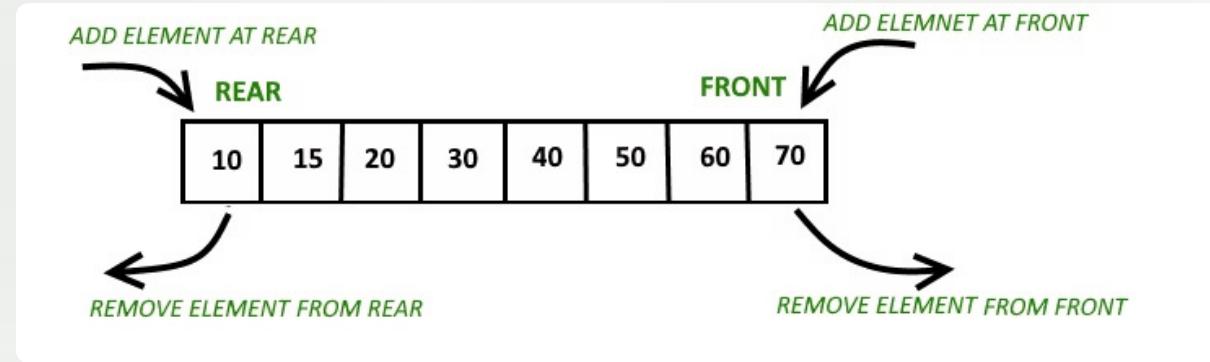
Allows elements to wrap around, effectively reusing previously freed memory slots.

## Full Condition

Determined by the formula:  $(\text{rear} + 1) \% \text{ size} == \text{front}$ .

## Empty Condition

Indicated when  $\text{front} == -1$ .



# Deque: Double-Ended Flexibility

A Deque, or Double-Ended Queue, is a versatile data structure allowing insertions and deletions from both the front and rear ends.

This dual functionality enables it to mimic the behavior of both a traditional queue and a stack.

- **Input-restricted deque:** Insertion only at one end, deletion at both.
- **Output-restricted deque:** Deletion only at one end, insertion at both.

All operations in a Deque, including insertion and deletion at either end, boast an optimal time complexity of O(1).

# Queue, Circular Queue & Deque: A Comparison

Insert	Rear only	Rear (circular)	Both ends
Delete	Front only	Front (circular)	Both ends
Space usage	Wasted space	Efficient	Efficient
Flexibility	Low	Medium	High

JS utils.js

JS index.js

JS blog

```

src ▶ components ▶ JS blog-post.js ▶ ↗ <fun
1 import { graphql } from "gatsby"
2 import React from "react"
3 import Image from "gatsby-image"
4
5 export default ({ data }) => {
6   const blogPost = d
7   return (
8     <div>
9       {blogP
10      blog
11      blog
12      <I
13      )>
14      <h1>{b
15      <div>P
16      <div d
17      </div>
18    )
19  }
20

```

PROBLEMS TERMINAL ...

1:

```

info i [wdm]: Compiled successfully.
info changed file at
WAIT Compiling...
9:51:57 AM

```

```

info i [wdm]: Compiling...
DONE Compiled successfully in 63ms
9:51:58 AM

```

```

info i [wdm]:
info i [wdm]: Compiled successfully.

```

Ln 6,

# Hands-On Practice: Implementation Challenges



## Array-Based Queue

Implement a standard queue using an array.



## Linked List Queue

Build a queue using a linked list for dynamic sizing.



## Circular Queue

Develop a circular queue to optimize space utilization.



## Deque (No STL)

Create a deque from scratch without using Standard Template Library.



# LeetCode Problems: Queues in Action

1

## Time Needed to Buy Tickets

Solve a problem involving queue processing for ticket purchases.

2

## Students Unable to Eat Lunch

Address a scenario where students try to get lunch from a queue.

3

## Winner of Circular Game

Determine the winner in a circular game using queue concepts.

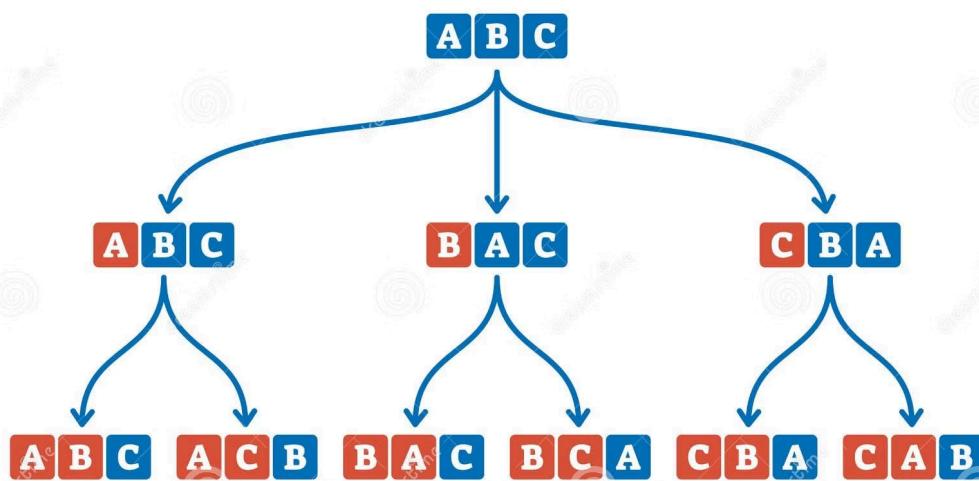
4

## Implement Stack using Queues

Challenge yourself to simulate stack behavior using only queues.

# Advanced LeetCode: Permutations

## PERMUTATION



### Permutations (Backtracking)

Explore the concept of permutations, a classic problem often solved using backtracking algorithms.

This problem tests your understanding of recursive approaches and state management.

It's a great way to apply your knowledge of data structures in more complex algorithmic scenarios.

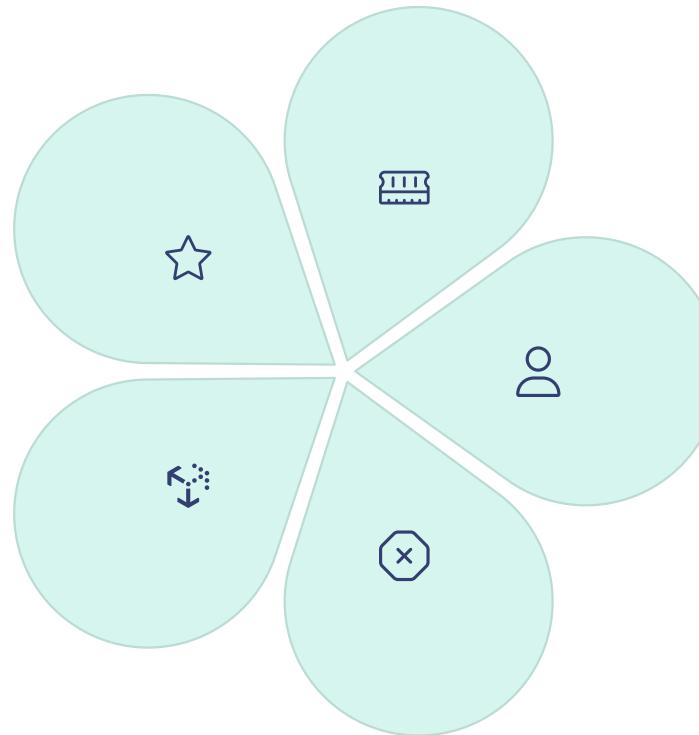
# Why Master Queues and Deques?

## Optimized Performance

Achieve  $O(1)$  time complexity for critical operations.

## Real-World Apps

Used in operating systems, network buffers, and more.



## Efficient Memory

Circular queues prevent wasted space in array implementations.

## Versatile Tools

Deques offer flexibility, acting as both queues and stacks.

## Problem Solving

Essential for tackling various algorithmic challenges.

# Next Steps: Continue Your DSA Journey



## Review Concepts

Revisit the definitions and operations of queues, circular queues, and deques.



## Code Implementations

Practice implementing each data structure from scratch.



## Solve More Problems

Tackle additional LeetCode problems to solidify understanding.



## Explore Advanced Topics

Look into priority queues and other queue variations.