**4.1 Queue**

Easy

1. Reverse First k Elements of Queue
2. Implement a Queue using an Array
3. Print all elements of a queue in a new line
4. Level with maximum number of nodes
5. Breadth First Search or BFS for a Graph
6. Find Minimum Depth of a Binary Tree

Medium

1. Implement a Deque
2. Implement a Circular Queue
3. Check if a queue can be sorted into another queue using a stack
4. Implement Stack using Queues
5. Implement Stack using Two Queues
6. Implement Queue using Two Stacks
7. Design a Queue data structure to get minimum or maximum in O(1) time
8. Check whether a given graph is Bipartite or not
9. Print Right View of a Binary Tree
10. An Interesting Method to Generate Binary Numbers from 1 to n
11. Implement a Queue using a Stack
12. Reverse a queue using recursion
13. Implement Priority Queue using Linked List
14. Implement Queue using Deque
15. Flatten a multilevel linked list
16. Find next right node of a given key
17. Detect cycle in an undirected graph using BFS
18. Minimum steps to reach target by a Knight
19. Islands in a graph using BFS
20. Flood Fill Algorithm
21. Minimum steps to reach target by a Knight
22. First negative integer in every window of size k
23. Level order traversal in spiral form
24. Minimum time required to rot all oranges
25. Queue based approach or first non-repeating character in a stream
26. Shortest distance in a maze
27. Geek in a Maze
28. Find shortest safe route in a path with landmines
29. Find the first circular tour that visits all petrol pumps
30. Connect Nodes at Same Level

Hard

1. Find the first non-repeating character from a stream of characters
2. Maximum of all subarrays of size k using a queue
3. Implement LRU Cache using Queue
4. Design a Queue data structure to get the maximum or minimum of sliding window
5. Find if there is a path between two vertices in a directed graph
6. Design a Data Structure for LRU Cache
7. Trapping Rain Water
8. Maximum cost path from source node to destination
9. Trapping Rain Water
10. Maximum cost path from source node to destination
11. Snake and Ladder Problem
12. Minimum Cost Path in a directed graph via given set of intermediate nodes
13. Turn a Queue into a Priority Queue
14. Interchange elements of Stack and Queue without changing order

Link: https://www.geeksforgeeks.org/top-50-problems-on-queue-data-structure-asked-in-sde-interviews/

**Queue problem given by lab teacher.**

**Class Example :**

**Question 1 - Queue Implementation with Array:**

You are required to implement a queue data structure using an array in C. The queue should

have the following functionalities:

a) void enqueue(int item): This function should enqueue an integer item into the queue.

b) int dequeue(): This function should remove and return the front item from the queue.

c) int front(): This function should return the front item from the queue without removing it.

d) int isEmpty(): This function should return 1 if the queue is empty, and 0 otherwise.

Write the C code for the queue implementation with an array and demonstrate its usage by

enqueuing a few elements, dequeuing elements, and checking whether the queue is empty.

**Question 2 - Queue Implementation with Linked List:**

You are required to implement a queue data structure using a singly linked list in C. The queue

should have the following functionalities:

a) void enqueue(int item): This function should enqueue an integer item into the queue.

b) int dequeue(): This function should remove and return the front item from the queue.

c) int front(): This function should return the front item from the queue without removing it.

d) int isEmpty(): This function should return 1 if the queue is empty, and 0 otherwise.

Write the C code for the queue implementation with a linked list and demonstrate its usage by

enqueuing a few elements, dequeuing elements, and checking whether the queue is empty.

Please ensure that your implementations handle edge cases appropriately and provide the

correct output for different scenarios.

**Question 3 - Circular Queue Implementation with Array:**

You are required to implement a circular queue data structure using an array in C. The circular

queue should have the following functionalities:

a) void enqueue(int item): This function should enqueue an integer item into the circular queue.

b) int dequeue(): This function should remove and return the front item from the circular queue.

c) int front(): This function should return the front item from the circular queue without removing

it.

d) int isEmpty(): This function should return 1 if the circular queue is empty, and 0 otherwise.

e) int isFull(): This function should return 1 if the circular queue is full, and 0 otherwise.

Write the C code for the circular queue implementation with an array and demonstrate its usage

by enqueuing a few elements, dequeuing elements, checking whether the queue is empty, and

verifying if the queue is full.

Ensure that your implementation handles the circular nature of the queue correctly and provides

the correct output for various scenarios.

**Question 4 - Implement a Queue using Stacks:**

You are tasked with designing a queue data structure using two stacks. Each stack should have

push, pop, and isEmpty functions. The queue should support the following operations:

enqueue(int x): Add element x to the back of the queue.

dequeue(): Remove and retrieve the element from the front of the queue.

Dequeue constant : https://www.geeksforgeeks.org/queue-using-stacks/

Enqueue constant : Code uploaded

**Practice problem from different section**

1. Implement a stack using queue.
2. Reverse the elements of a queue.

**Input**: q[ ] = {10, 20, 30, 40, 50, 60}

**Output**: q[ ] = {60, 50, 40, 30, 20, 10}

**3.** Sort the elements of a queue without using any extra space.

**Input**: q[ ] = {20, 40, 10, 60, 50, 30}

**Output**: q[ ] = {10, 20, 30, 40, 50, 60}

**4.** Given a queue q[ ] and an integer K, remove the integer K from the queue. If

multiple same elements exist, remove the first one.

**Input**: q[ ] = {10, 20, 30, 40, 50, 60}, K = 30

**Output**: {10, 20, 40, 50, 60}

**Input**: q[] = {1, 2, 3, 3}, K = 3

**Output**: {1, 2, 3}

**4.1 Queue in Data Structure | Introduction to Queue**