Name: Gobinda Narayan Pradhan **UID: 22BCS16759**

Subject: Advance Programming Lab – II **Section: 22BCS-IoT_626 [B]**

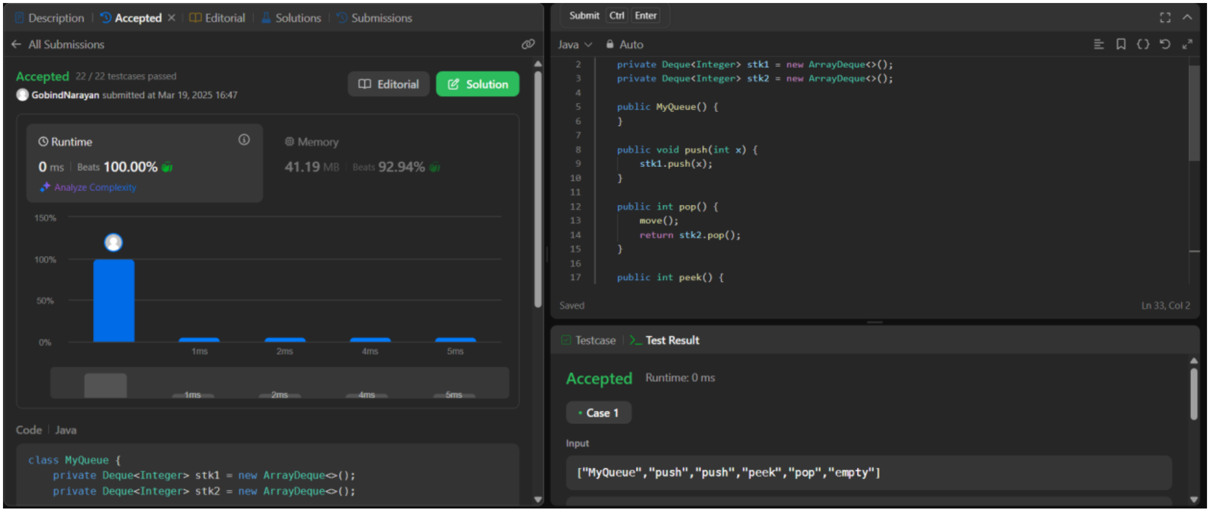
Subject Code: 22CSP-351 **Date of Submission: 19th Mar**

Assignment – 6

1. Implement Queue using Stack

```
class MyQueue {
   private Deque<Integer> stk1 = new ArrayDeque<>();
   private Deque<Integer> stk2 = new ArrayDeque<>();

   public MyQueue() {
   }

   public void push(int x) {
      stk1.push(x);
   }

   public int pop() {
      move();
      return stk2.pop();
   }

   public int peek() {
      move();
      return stk2.peek();
   }

   public boolean empty() {
      return stk1.isEmpty() && stk2.isEmpty();
   }

   private void move() {
      while (stk2.isEmpty()) {
         while (!stk1.isEmpty()) {
            stk2.push(stk1.pop());
         }
      }
   }
```

}



Accepted 22 / 22 testcases passed

GobindNarayan submitted at Mar 19, 2025 16:47

Runtime: 0 ms | Beats 100.00%

Memory: 41.19 MB | Beats 92.94%

Code | Java

```
class MyQueue {
    private Deque<Integer> stk1 = new ArrayDeque<>();
    private Deque<Integer> stk2 = new ArrayDeque<>();
```

```
    private Deque<Integer> stk1 = new ArrayDeque<>();
    private Deque<Integer> stk2 = new ArrayDeque<>();

    public MyQueue() {
    }

    public void push(int x) {
        stk1.push(x);
    }

    public int pop() {
        move();
        return stk2.pop();
    }

    public int peek() {
```

Saved

Ln 33, Col 2

Accepted Runtime: 0 ms

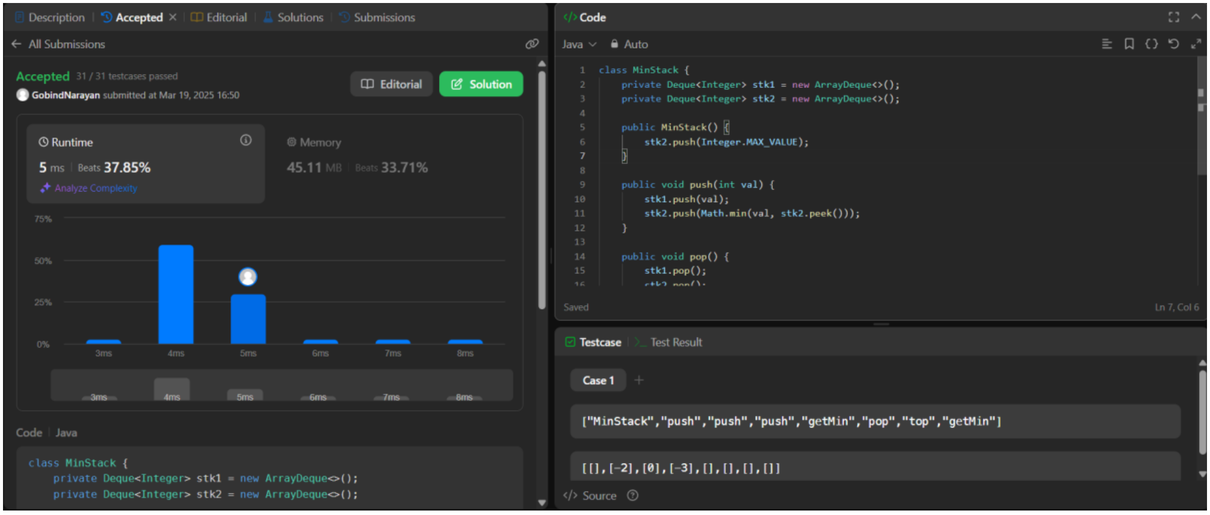
Input

```
["MyQueue","push","push","peek","pop","empty"]
```

2. Implement Min Stack using Two Stacks

```
class MinStack {
    private Deque<Integer> stk1 = new ArrayDeque<>();
    private Deque<Integer> stk2 = new ArrayDeque<>();

    public MinStack() {
        stk2.push(Integer.MAX_VALUE);
    }

    public void push(int val) {
        stk1.push(val);
        stk2.push(Math.min(val, stk2.peek()));
    }

    public void pop() {
        stk1.pop();
        stk2.pop();
    }

    public int top() {
        return stk1.peek();
    }

    public int getMin() {
        return stk2.peek();
    }
```

}



Accepted 31 / 31 testcases passed

GobindNarayan submitted at Mar 19, 2025 16:50

Runtime 5 ms | Beats 37.85%

Memory 45.11 MB | Beats 33.71%

Code | Java

```
class MinStack {
    private Deque<Integer> stk1 = new ArrayDeque<>();
    private Deque<Integer> stk2 = new ArrayDeque<>();
```

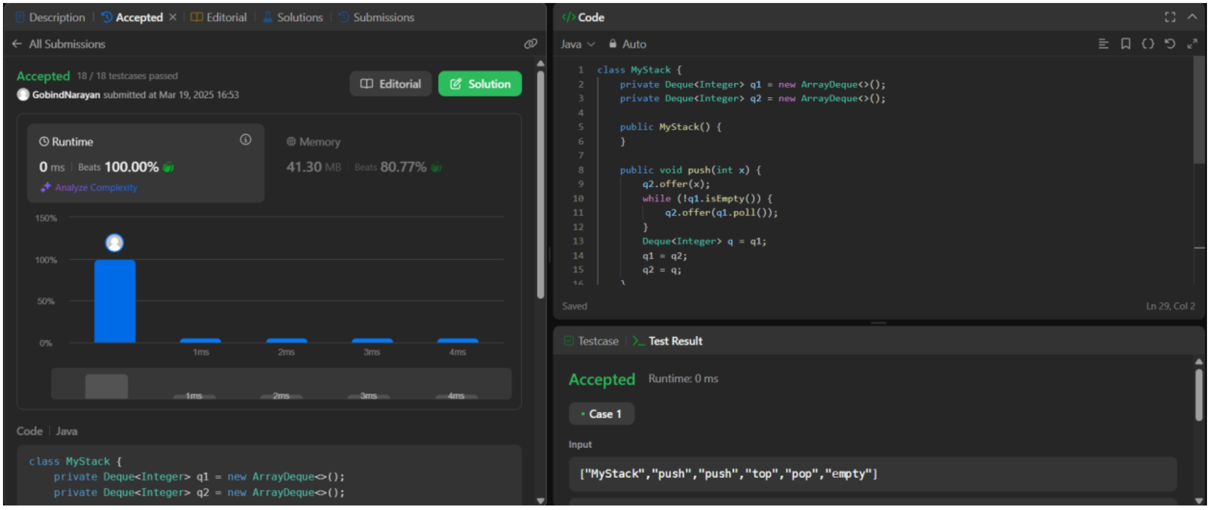
```
class MinStack {
    private Deque<Integer> stk1 = new ArrayDeque<>();
    private Deque<Integer> stk2 = new ArrayDeque<>();

    public MinStack() {
        stk2.push(Integer.MAX_VALUE);
    }

    public void push(int val) {
        stk1.push(val);
        stk2.push(Math.min(val, stk2.peek()));
    }

    public void pop() {
        stk1.pop();
        stk2.pop();
```

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
```

```
[[],[-2],[0],[-3],[],[],[],[]]
```

3. Implement Stack using Queue

```
class MyStack {
    private Deque<Integer> q1 = new ArrayDeque<>();
    private Deque<Integer> q2 = new ArrayDeque<>();

    public MyStack() {
    }

    public void push(int x) {
        q2.offer(x);
        while (!q1.isEmpty()) {
            q2.offer(q1.poll());
        }
        Deque<Integer> q = q1;
        q1 = q2;
        q2 = q;
    }

    public int pop() {
        return q1.poll();
    }

    public int top() {
        return q1.peek();
    }

    public boolean empty() {
        return q1.isEmpty();
    }
}
```



Accepted 18 / 18 testcases passed

GobindNarayan submitted at Mar 19, 2025 16:53

Runtime 0 ms | Beats 100.00%

Analyze Complexity

Memory 41.30 MB | Beats 80.77%

Code | Java

```
class MyStack {
    private Deque<Integer> q1 = new ArrayDeque<>();
    private Deque<Integer> q2 = new ArrayDeque<>();
```

Java

```
class MyStack {
    private Deque<Integer> q1 = new ArrayDeque<>();
    private Deque<Integer> q2 = new ArrayDeque<>();

    public MyStack() {
    }

    public void push(int x) {
        q2.offer(x);
        while (!q1.isEmpty()) {
            q2.offer(q1.poll());
        }
        Deque<Integer> q = q1;
        q1 = q2;
        q2 = q;
```

Saved

Ln 29, Col 2

Accepted Runtime: 0 ms

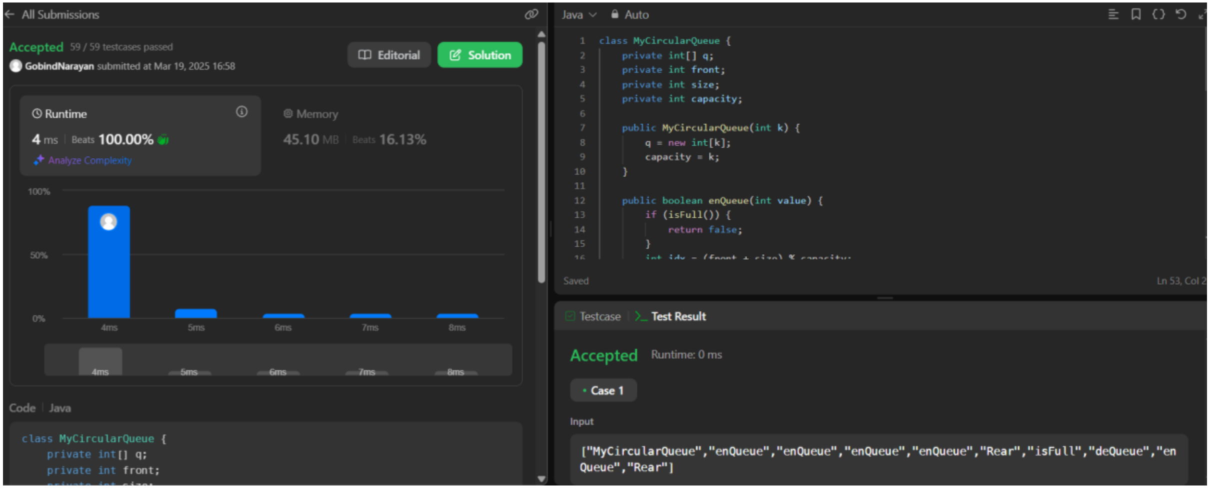
Case 1

Input

```
["MyStack","push","push","top","pop","empty"]
```

4. Implement Circular Queue using Queue

```
class MyCircularQueue {
    private int[] q;
    private int front;
    private int size;
    private int capacity;

    public MyCircularQueue(int k) {
        q = new int[k];
        capacity = k;
    }

    public boolean enQueue(int value) {
        if (isFull()) {
            return false;
        }
        int idx = (front + size) % capacity;
        q[idx] = value;
        ++size;
        return true;
    }

    public boolean deQueue() {
        if (isEmpty()) {
            return false;
        }
        front = (front + 1) % capacity;
        --size;
        return true;
    }

    public int Front() {
        if (isEmpty()) {
            return -1;
        }
        return q[front];
    }

    public int Rear() {
        if (isEmpty()) {
            return -1;
        }
        int idx = (front + size - 1) % capacity;
        return q[idx];
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == capacity;
    }
}
```



Accepted 59 / 59 testcases passed

GobindNarayan submitted at Mar 19, 2025 16:58

Runtime

4 ms | Beats 100.00%

Memory

45.10 MB | Beats 16.13%

Code | Java

```
class MyCircularQueue {
    private int[] q;
    private int front;
    private int size;
```

```
class MyCircularQueue {
    private int[] q;
    private int front;
    private int size;
    private int capacity;

    public MyCircularQueue(int k) {
        q = new int[k];
        capacity = k;
    }

    public boolean enQueue(int value) {
        if (isFull()) {
            return false;
        }
        int idx = (front + size) % capacity;
```

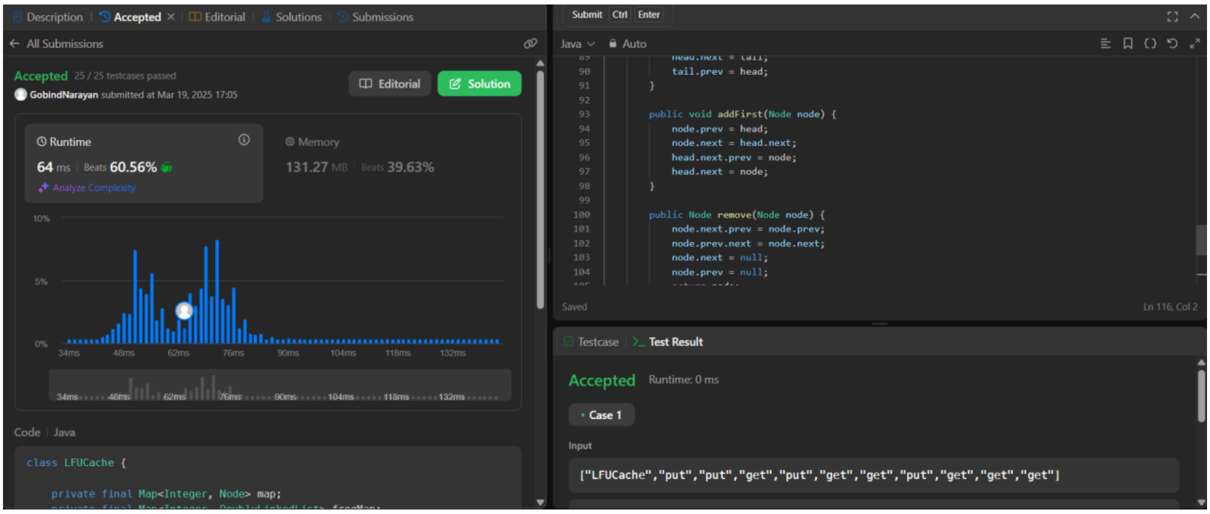
Accepted Runtime: 0 ms

Input

```
["MyCircularQueue","enQueue","enQueue","enQueue","enQueue","Rear","isFull","deQueue","en
Queue","Rear"]
```

5. Implement LFU Cache using Hash Table + Min Heap

```
class LFUCache {

    private final Map<Integer, Node> map;
    private final Map<Integer, DoublyLinkedList> freqMap;
    private final int capacity;
    private int minFreq;

    public LFUCache(int capacity) {
        this.capacity = capacity;
        map = new HashMap<>(capacity, 1);
        freqMap = new HashMap<>();
    }

    public int get(int key) {
        if (capacity == 0) {
            return -1;
        }
        if (!map.containsKey(key)) {
            return -1;
        }
        Node node = map.get(key);
        incrFreq(node);
        return node.value;
    }

    public void put(int key, int value) {
        if (capacity == 0) {
            return;
        }
        if (map.containsKey(key)) {
            Node node = map.get(key);
            node.value = value;
            incrFreq(node);
            return;
        }
        if (map.size() == capacity) {
            DoublyLinkedList list = freqMap.get(minFreq);
            map.remove(list.removeLast().key);
        }
        Node node = new Node(key, value);
        addNode(node);
        map.put(key, node);
        minFreq = 1;
    }

    private void incrFreq(Node node) {
        int freq = node.freq;
        DoublyLinkedList list = freqMap.get(freq);
        list.remove(node);
        if (list.isEmpty()) {
            freqMap.remove(freq);
            if (freq == minFreq) {
                minFreq++;
            }
        }
        node.freq++;
        addNode(node);
    }

    private void addNode(Node node) {
        int freq = node.freq;
        DoublyLinkedList list = freqMap.getOrDefault(freq, new
DoublyLinkedList());
        list.addFirst(node);
        freqMap.put(freq, list);
    }

    private static class Node {
        int key;
        int value;
        int freq;
        Node prev;
        Node next;

        Node(int key, int value) {
            this.key = key;
            this.value = value;
            this.freq = 1;
        }
    }

    private static class DoublyLinkedList {

        private final Node head;
        private final Node tail;

        public DoublyLinkedList() {
            head = new Node(-1, -1);
            tail = new Node(-1, -1);
            head.next = tail;
            tail.prev = head;
        }

        public void addFirst(Node node) {
            node.prev = head;
            node.next = head.next;
            head.next.prev = node;
            head.next = node;
        }

        public Node remove(Node node) {
            node.next.prev = node.prev;
            node.prev.next = node.next;
            node.next = null;
            node.prev = null;
            return node;
        }

        public Node removeLast() {
            return remove(tail.prev);
        }

        public boolean isEmpty() {
            return head.next == tail;
        }
    }
}
```



Accepted 25 / 25 testcases passed

GobindNarayan submitted at Mar 19, 2025 17:05

Runtime

64 ms | Beats 60.56%

Memory

131.27 MB | Beats 39.63%

Code | Java

```
class LFUCache {

    private final Map<Integer, Node> map;
```

```
            tail.prev = head;
        }

        public void addFirst(Node node) {
            node.prev = head;
            node.next = head.next;
            head.next.prev = node;
            head.next = node;
        }

        public Node remove(Node node) {
            node.next.prev = node.prev;
            node.prev.next = node.next;
            node.next = null;
            node.prev = null;
```

Saved

Ln 116, Col 2

Accepted Runtime: 0 ms

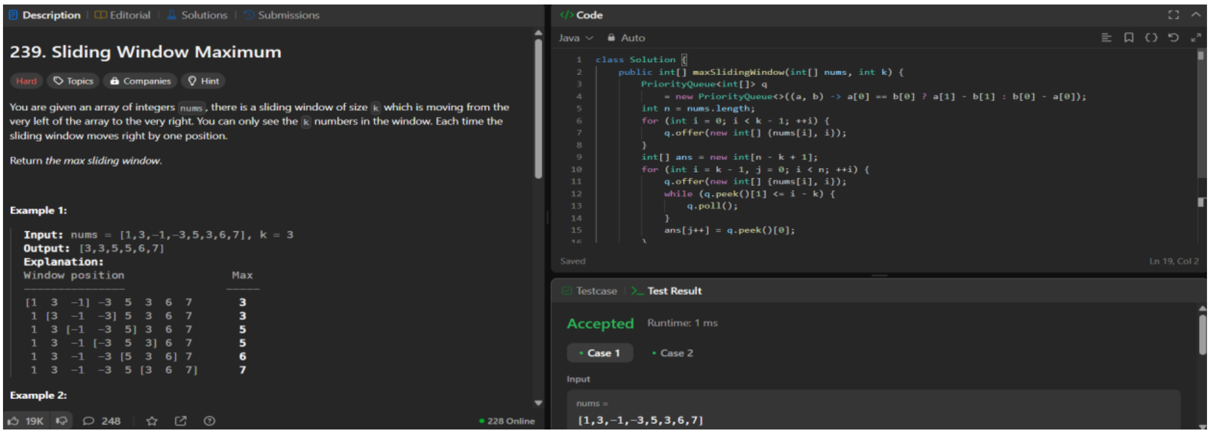
Case 1

Input

```
["LFUCache","put","put","get","put","get","get","put","get","get","get"]
```

6. <u>Implement Sliding Window Maximum using Deque</u>

```
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        PriorityQueue<int[]> q
            = new PriorityQueue<>((a, b) -> a[0] == b[0] ? a[1] - b[1] : b[0] - 
a[0]);
        int n = nums.length;
        for (int i = 0; i < k - 1; ++i) {
            q.offer(new int[] {nums[i], i});
        }
        int[] ans = new int[n - k + 1];
        for (int i = k - 1, j = 0; i < n; ++i) {
            q.offer(new int[] {nums[i], i});
            while (q.peek()[1] <= i - k) {
                q.poll();}
            ans[j++] = q.peek()[0];
        }
        return ans;
```



239. Sliding Window Maximum

You are given an array of integers `nums`, there is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position.

Return *the max sliding window.*

Example 1:

```
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Example 2:

```
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        PriorityQueue<int[]> q
            = new PriorityQueue<>((a, b) -> a[0] == b[0] ? a[1] - b[1] : b[0] - a[0]);
        int n = nums.length;
        for (int i = 0; i < k - 1; ++i) {
            q.offer(new int[] {nums[i], i});
        }
        int[] ans = new int[n - k + 1];
        for (int i = k - 1, j = 0; i < n; ++i) {
            q.offer(new int[] {nums[i], i});
            while (q.peek()[1] <= i - k) {
                q.poll();
            }
            ans[j++] = q.peek()[0];
```

Saved

Ln 19, Col 2

Accepted Runtime: 1 ms

Case 1 · Case 2

Input

nums =

```
[1,3,-1,-3,5,3,6,7]
```