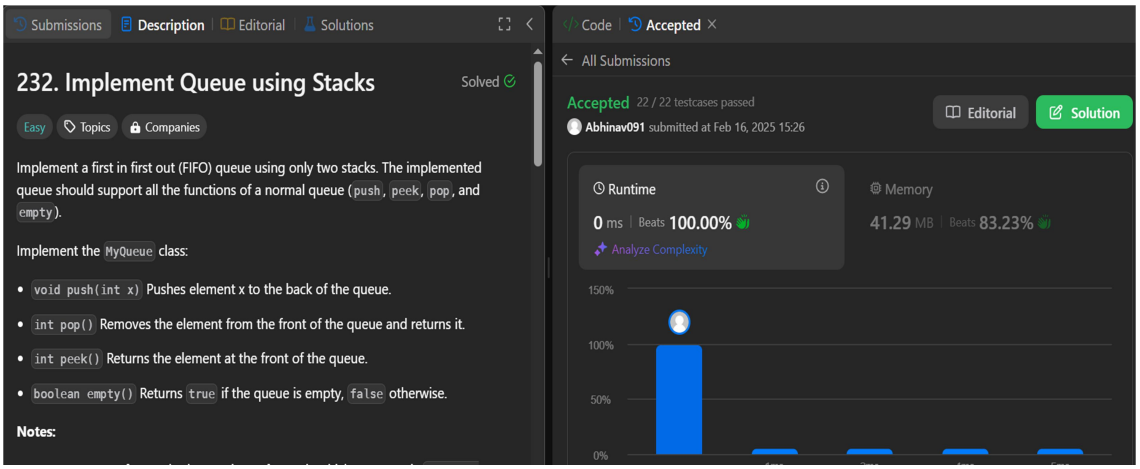
1. Implement Queue using Stacks

Code:

```
import java.util.Stack;

class CustomQueue {
   private Stack<Integer> inputStack;
   private Stack<Integer> outputStack;

   public CustomQueue() {
      inputStack = new Stack<>();
      outputStack = new Stack<>();
   }

   private void shiftStacks() {
      while (!inputStack.isEmpty()) {
         outputStack.push(inputStack.pop());
      }
   }

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

   public int pop() {
      if (outputStack.isEmpty()) {
         shiftStacks();
      }
      return outputStack.pop();
   }

   public int peek() {
      if (outputStack.isEmpty()) {
         shiftStacks();
      }
      return outputStack.peek();
   }

   public boolean empty() {
      return inputStack.isEmpty() && outputStack.isEmpty();
   }}
```

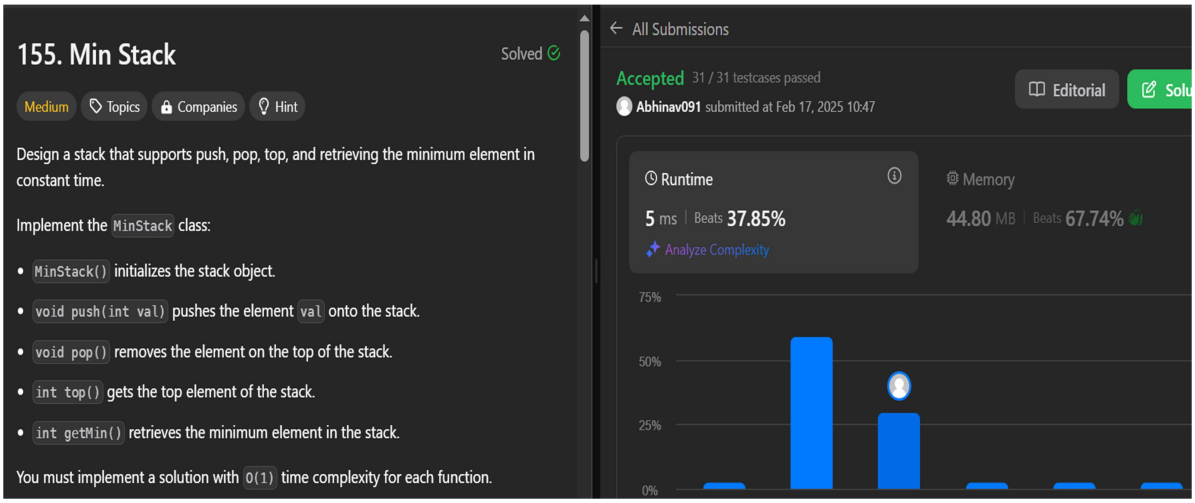


2. Implement Min Stack using Two Stacks

Code:

```
class MinStack {
   Stack<Integer> stack;
   Stack<Integer> MinStack;
   public MinStack() {
      stack=new Stack<>();
      MinStack=new Stack<>();
   }

   public void push(int val) {
     stack.push(val);
     if(MinStack.isEmpty() || val<=MinStack.peek()){
      MinStack.push(val);
     }
   }

   public void pop() {
      if(stack.isEmpty()){
         return;
      }
      int top= stack.pop();
      if(!MinStack.isEmpty()&&top==MinStack.peek()){
         MinStack.pop();
      }
   }

   public int top() {
```

```
        return stack.peek();
    }

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

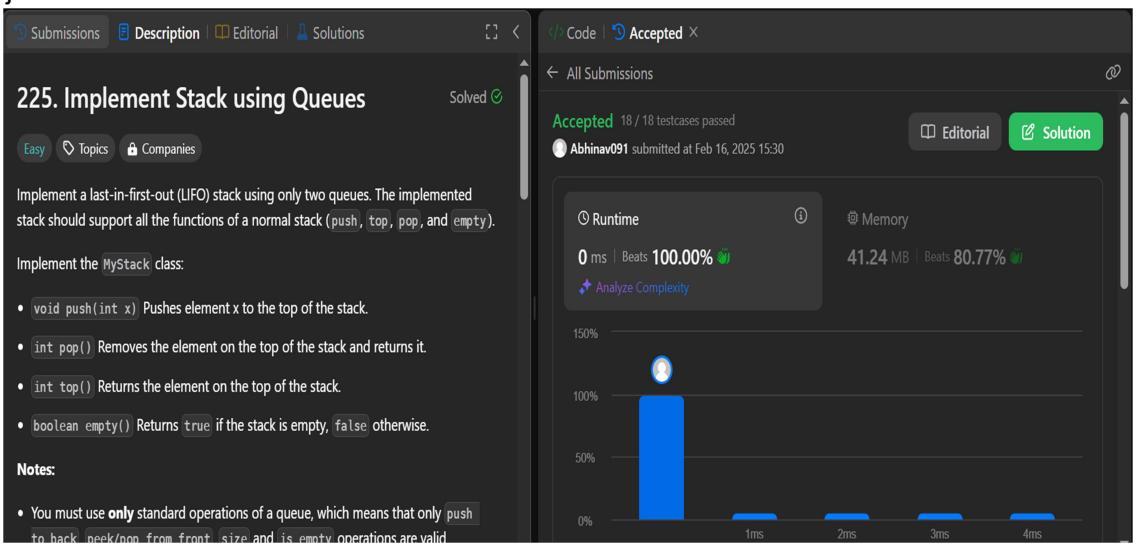


3. Implement Stack using Queue

Code:

```
class MyStack {
    private Queue<Integer> queue;

    public MyStack() {
        queue = new LinkedList<>();
    }

    public void push(int x) {
        queue.add(x);
        int size = queue.size();
        while (size-- > 1) {
            queue.add(queue.remove());
        }
    }

    public int pop() {
        return queue.remove();
    }
```

```
    public int top() {
        return queue.peek();
    }

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

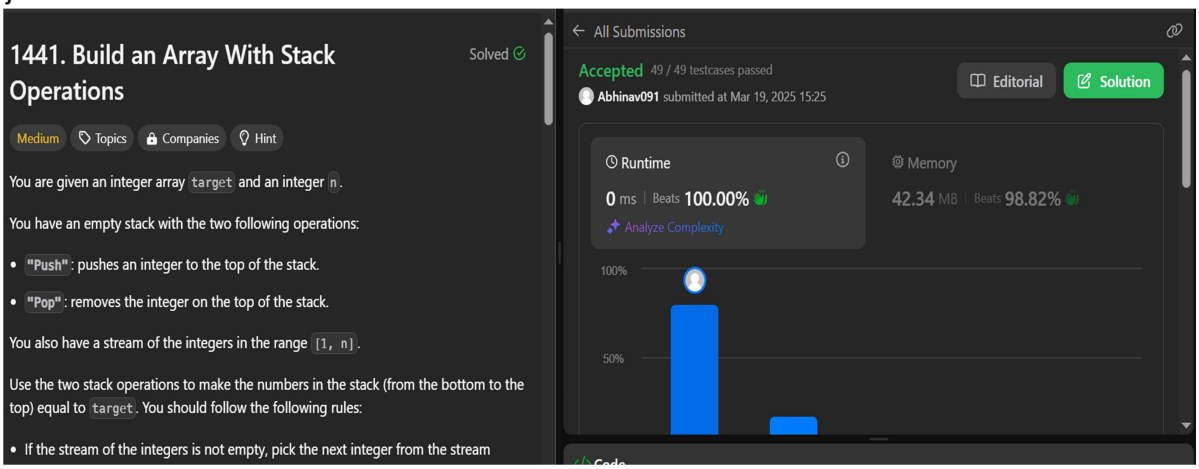


4. Implement stack using array

Code:

```
class Solution {
    public List<String> buildArray(int[] target, int n) {
        int[] array = new int[n];
        List<String> res = new ArrayList<>();
        int i = 0, c = 1;

        while (i < target.length) {
            array[c - 1] = c;
            res.add("Push");

            if (array[c - 1] == target[i]) {
                i++;
            } else {
                res.add("Pop");
            }
```

```
            c++;
        }
        return res;
    }
}
```



5. Implement LRU Cache using Hash Table + Doubly Linked List

Code:

```
class Node {
    int key;
    int val;
    Node prev;
    Node next;

    public Node(int key, int val) {
        this.key = key;
        this.val = val;
        this.prev = null;
        this.next = null;
    }
}
class LRUCache {

    private int cap;
    private Map<Integer, Node> cache;
    private Node oldest;
    private Node latest;

    public LRUCache(int capacity) {
```

```
    this.cap = capacity;
    this.cache = new HashMap<>();
    this.oldest = new Node(0, 0);
    this.latest = new Node(0, 0);
    this.oldest.next = this.latest;
    this.latest.prev = this.oldest;
}

public int get(int key) {
    if (cache.containsKey(key)) {
        Node node = cache.get(key);
        remove(node);
        insert(node);
        return node.val;
    }
    return -1;
}

private void remove(Node node) {
    Node prev = node.prev;
    Node next = node.next;
    prev.next = next;
    next.prev = prev;
}

private void insert(Node node) {
    Node prev = latest.prev;
    Node next = latest;
    prev.next = next.prev = node;
    node.next = next;
    node.prev = prev;
}

public void put(int key, int value) {
    if (cache.containsKey(key)) {
        remove(cache.get(key));
    }
    Node newNode = new Node(key, value);
    cache.put(key, newNode);
    insert(newNode);
```

```
        if (cache.size() > cap) {
            Node lru = oldest.next;
            remove(lru);
            cache.remove(lru.key);
        }
    }
}
```