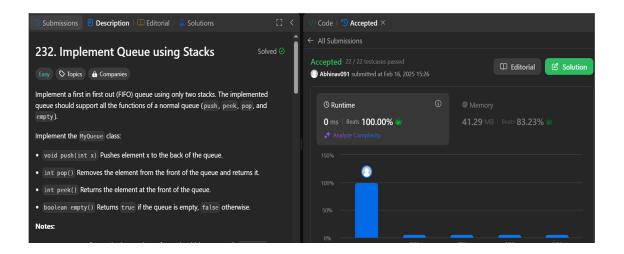
1. Implement Queue using Stacks

```
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();
  }}
```

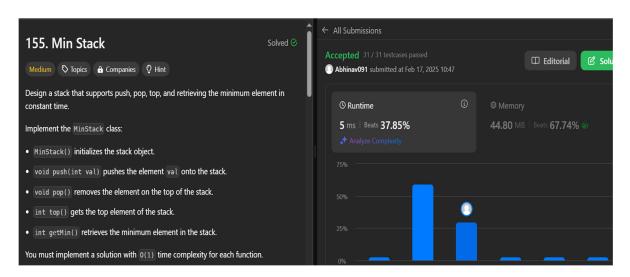


2. Implement Min Stack using Two Stacks Code:

```
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()){</pre>
    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();
  }
 Submissions  

Description  

Editorial  

Solutions
                                                                            Code | 🧐 Accepted ×
225. Implement Stack using Queues
                                                                                                                            Easy ♦ Topics ♠ Companies
                                                                          Abhinav091 submitted at Feb 16, 2025 15:30
Implement a last-in-first-out (LIFO) stack using only two queues. The implemented
stack should support all the functions of a normal stack (push, top, pop, and empty).
                                                                             0 ms | Beats 100.00% 🞳
Implement the MyStack class:
• void push(int x) Pushes element x to the top of the stack.
• int pop() Removes the element on the top of the stack and returns it.
• int top() Returns the element on the top of the stack.
• boolean empty() Returns true if the stack is empty, false otherwise.
Notes:
• You must use only standard operations of a queue, which means that only push
```

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");
    }
}</pre>
```

```
C++;
        }
        return res;
   }
1441. Build an Array With Stack
                                                                                                                                            Operations
                                                                                    Abhinav091 submitted at Mar 19, 2025 15:25
 Medium ♥ Topics ♠ Companies ♥ Hint
You are given an integer array target and an integer n.
                                                                                       0 ms | Beats 100.00% 🎳
You have an empty stack with the two following operations:
 "Push": pushes an integer to the top of the stack.
• "Pop": removes the integer on the top of the stack.
You also have a stream of the integers in the range [1, n].
Use the two stack operations to make the numbers in the stack (from the bottom to the
top) equal to target. You should follow the following rules:
\bullet\, If the stream of the integers is not empty, pick the next integer from the stream
```

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 Iru = oldest.next;
    remove(Iru);
    cache.remove(Iru.key);
    }
}
```

