

Problem List < > >>

Description | Editorial | Solutions | Accepted x | Submissions

All Submissions

Accepted 18 / 18 testcases passed  
yash mittal submitted at Mar 19, 2025 15:09

Editorial Solution

Runtime 0 ms Beats 100.00%  
Memory 40.77 MB Beats 99.66%

Analyze Complexity

Code: Java

```
public class MyStack {
    private Queue<Integer> q;

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

Code

```
1 public class MyStack {
2     private Queue<Integer> q;
3
4     public MyStack() {
5         q = new LinkedList<>();
6     }
7
8     public void push(int x) {
9         q.add(x);
10        for (int i = 1; i < q.size(); i++) {
11            q.add(q.remove());
12        }
13    }
14}
```

Run Submit

Testcase Test Result

Case 1 +

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

[[], [1], [2], [], [], []]

</> Source

Implement stack using queues

Problem List < > >>

Description | Editorial | Solutions | Accepted x | Submissions

All Submissions

Accepted 31 / 31 testcases passed  
yash mittal submitted at Mar 19, 2025 15:15

Editorial Solution

Runtime 4 ms Beats 97.14%  
Memory 44.89 MB Beats 67.74%

Analyze Complexity

Code: Java

```
class MinStack {
    private List<int[]> st;

    public MinStack() {
        st = new ArrayList<>();
    }

    public void push(int val) {
        int[] top = st.isEmpty() ? new int[]{val, val} : st.get(st.size() - 1);
        int min_val = top[1];
        if (min_val > val) {
            min_val = val;
        }
        st.add(new int[]{val, min_val});
    }
}
```

Code

```
1 class MinStack {
2     private List<int[]> st;
3
4     public MinStack() {
5         st = new ArrayList<>();
6     }
7
8     public void push(int val) {
9         int[] top = st.isEmpty() ? new int[]{val, val} : st.get(st.size() - 1);
10        int min_val = top[1];
11        if (min_val > val) {
12            min_val = val;
13        }
14        st.add(new int[]{val, min_val});
15    }
16}
```

Run Submit

Testcase Test Result

Case 1 +

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

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

</> Source

32°C Haze Search 3:15 PM 3/19/2025

Min stack

**Accepted** 22 / 22 testcases passed  
yash mittal submitted at Mar 19, 2025 15:16

**Runtime** 0 ms | Beats 100.00%  
**Memory** 41.87 MB | Beats 10.05%

**Code** Java

```
class MyQueue {
    private Stack<Integer> input;
    private Stack<Integer> output;

    public MyQueue() {
        input = new Stack<>();
        output = new Stack<>();
    }

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

    public int pop() {
        if (output.isEmpty()) {
            while (!input.isEmpty()) {
                output.push(input.pop());
            }
        }
        return output.pop();
    }
}
```

**Testcase** Case 1

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

[[], [1], [2], [1], [1], []]

## Queues using stack

**Accepted** 59 / 59 testcases passed  
yash mittal submitted at Mar 19, 2025 15:18

**Runtime** 4 ms | Beats 100.00%  
**Memory** 44.80 MB | Beats 48.65%

**Code** Java

```
class MyCircularQueue {
    int[] data;
    int front = 0; // Points to the first element
    int rear = -1; // Points to the last element
    int size = 0; // Tracks the number of elements

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

    public boolean enqueue(int value) {
        if (isFull()) return false;
        if (rear == -1) rear = 0;
        else rear = (rear + 1) % data.length;
        data[rear] = value;
        size++;
        return true;
    }

    public boolean dequeue() {
        if (isEmpty()) return false;
        front = (front + 1) % data.length;
        size--;
        return true;
    }

    public int front() {
        return data[front];
    }

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

    public boolean isFull() {
        return size == data.length;
    }

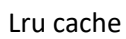
    public int rear() {
        return data[rear];
    }
}
```

**Testcase** Case 1

["MyCircularQueue", "enqueue", "enqueue", "enqueue", "enqueue", "Rear", "isFull", "dequeue", "enqueue", "Rear"]

[[3], [1], [2], [3], [4], [1], [1], [4], [1]]

## Circular queue



## Lru cache