DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

ASSIGNMENT - 6

Student Name: Sameer **UID:** 22BCS15631
Branch: Computer Science & Engineering **Section/Group:** IOT-614/B
Semester: 6th **Date of Performance:** 12/03/2025
Subject Name: Advanced Programming Lab-2 **Subject Code:** 22CSP-351

Q.1. Implement Queue using Stacks

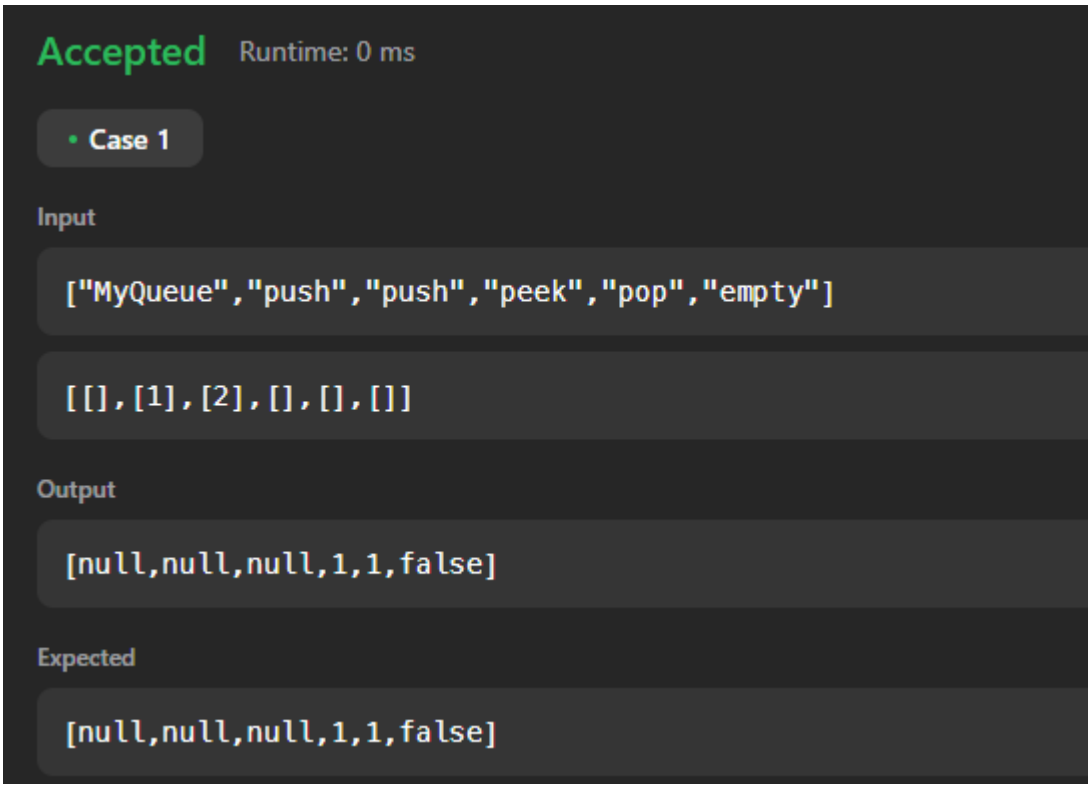
Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

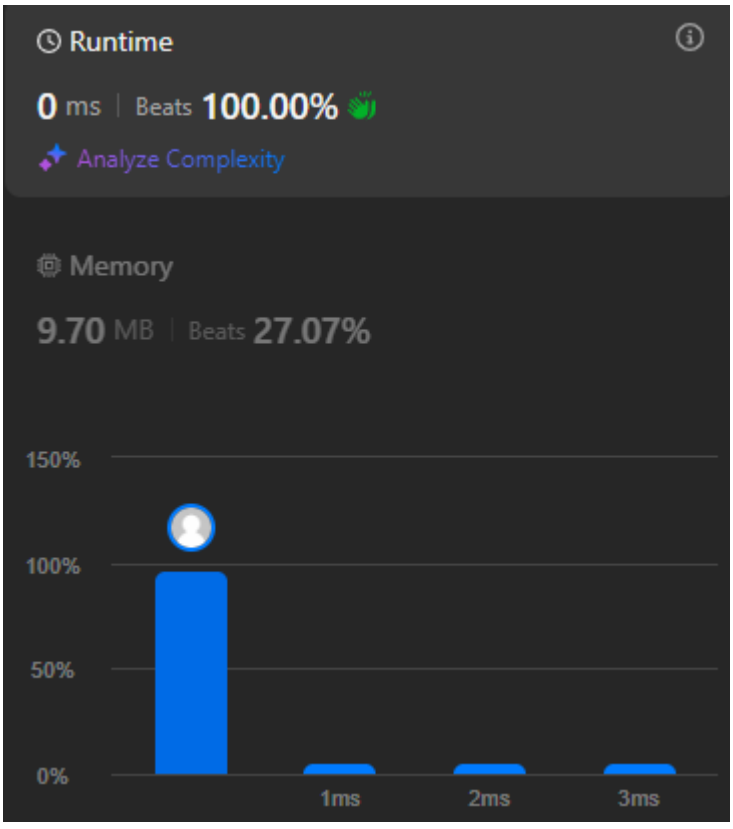
Code:

```
class MyQueue
{
  public:

  stack<int> s1 ;    // original queue
  stack<int> s2 ;

  void push(int x)
  {
    s1.push(x) ;
  }

  int pop()
  {
    while (!s1.empty())
    {
      s2.push(s1.top()) ;
      s1.pop() ;
    }

    int removed = s2.top() ;
    s2.pop() ;

    while (!s2.empty())
    {
      s1.push(s2.top()) ;
      s2.pop() ;
    }
```

```
        return removed ;
    }

    int peek()
    {
        while (!s1.empty())
        {
            s2.push(s1.top()) ;
            s1.pop() ;
        }

        int element = s2.top() ;

        while (!s2.empty())
        {
            s1.push(s2.top()) ;
            s2.pop() ;
        }

        return element ;
    }

    bool empty()
    {
        if (s1.empty())
        {
            return true ;
        }

        return false ;
    }
};
```

Output:


Accepted
Runtime: 0 ms
Case 1
Input
["MyQueue","push","push","peek","pop","empty"]
[[],[1],[2],[],[],[]]
Output
[null,null,null,1,1,false]
Expected
[null,null,null,1,1,false]


Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
9.70 MB | Beats 27.07%
150%
100%
50%
0%
1ms
2ms
3ms

Q.2. Design Circular Queue

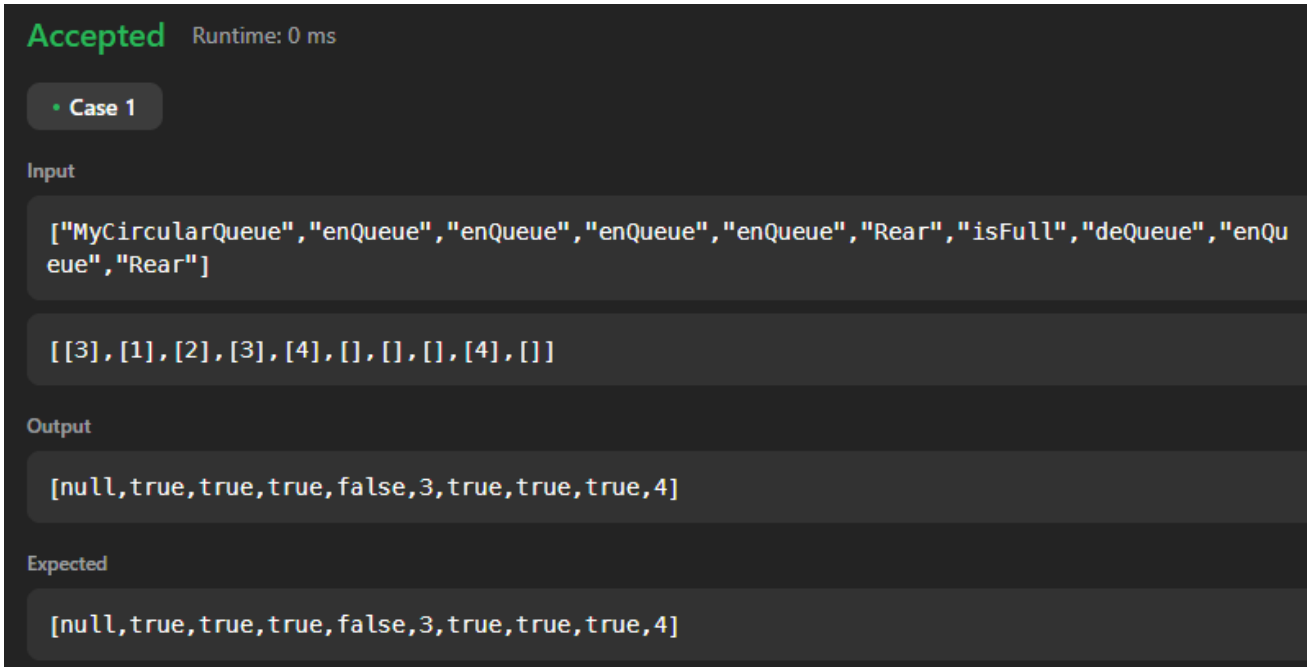
Design your implementation of the circular queue. The circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle, and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

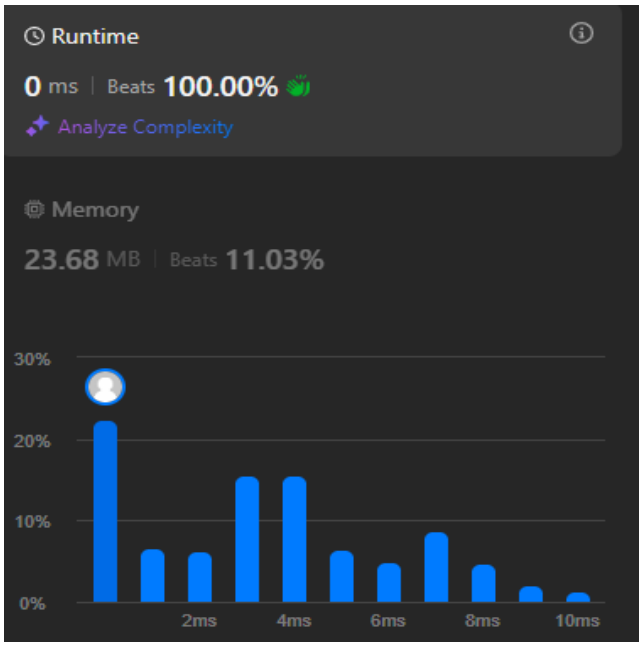
Code:

```
class MyCircularQueue {
private:
    vector<int> queue;
    int front, rear, size, capacity;

public:
    MyCircularQueue(int k) {
        queue.resize(k);
        capacity = k;
        size = 0;
        front = 0;
        rear = -1;
    }

    bool enQueue(int value) {
        if (isFull()) return false;
        rear = (rear + 1) % capacity;
        queue[rear] = value;
        size++;
        return true;
    }

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

    int Front() {
        return isEmpty() ? -1 : queue[front];
    }

    int Rear() {
        return isEmpty() ? -1 : queue[rear];
    }
```

```
    bool isEmpty() {
        return size == 0;
    }

    bool isFull() {
        return size == capacity;
    }
};
```

Output:





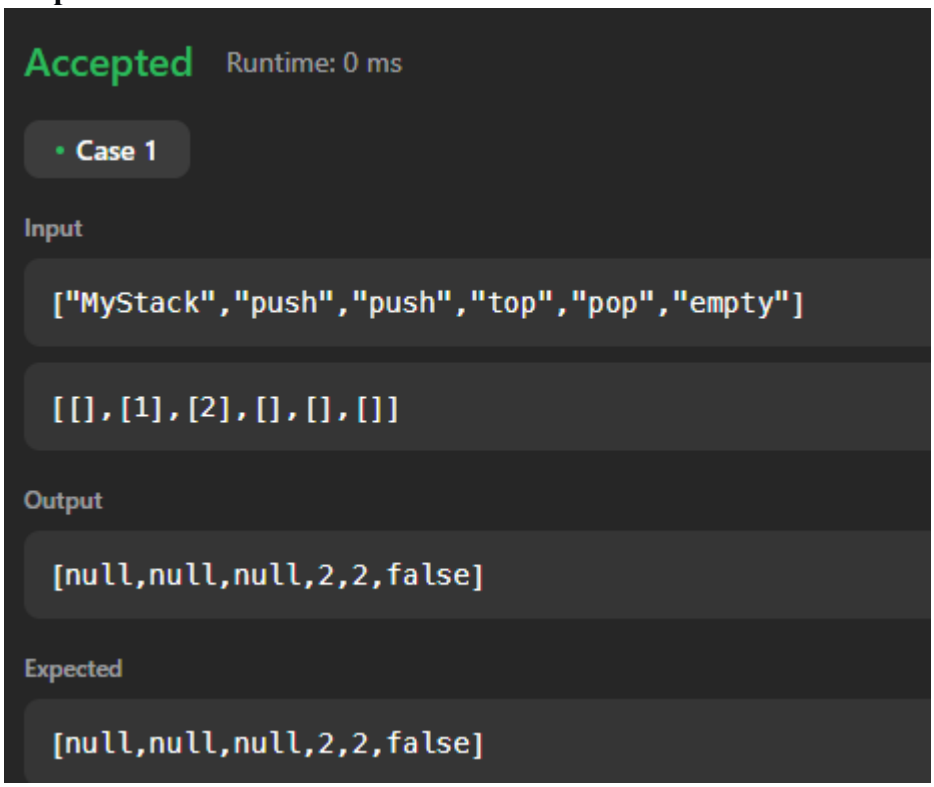
Q. 3. Implement Stack using Queues

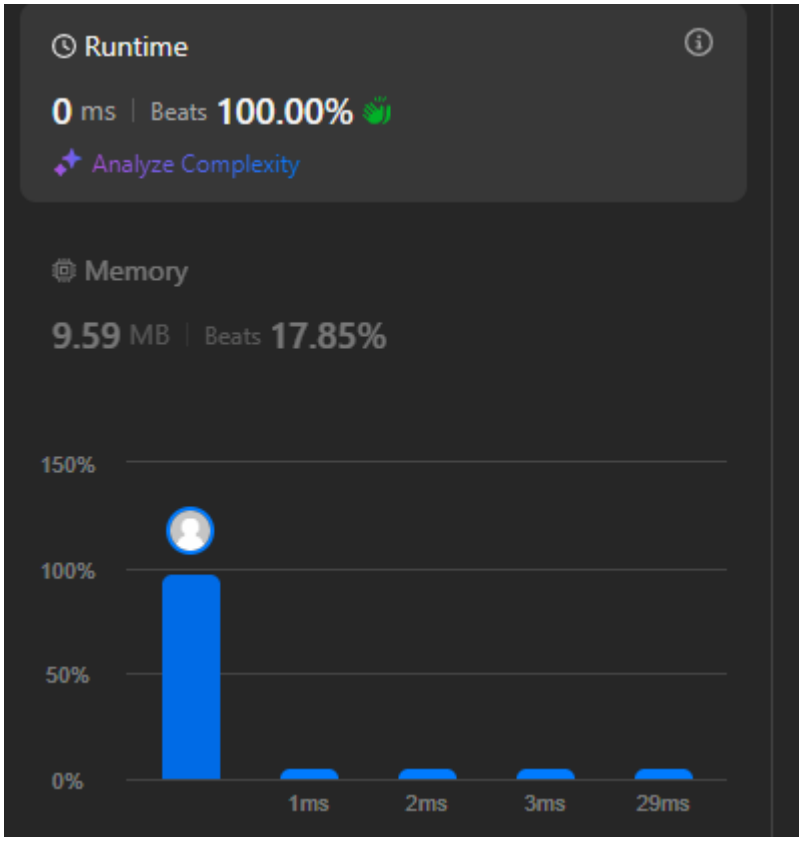
Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

Code:

```
class MyStack {
private:
   queue<int> q1, q2;
public:
   MyStack() {}

   void push(int x) {
      q2.push(x);
      while (!q1.empty()) {
         q2.push(q1.front());
         q1.pop();
      }
      swap(q1, q2);
   }

   int pop() {
      if (q1.empty()) return -1;
      int topElement = q1.front();
      q1.pop();
      return topElement;
   }

   int top() {
      return q1.empty() ? -1 : q1.front();
   }

   bool empty() {
      return q1.empty();
   }
};
```

Output:


Accepted
Runtime: 0 ms
Case 1
Input
["MyStack","push","push","top","pop","empty"]
[[],[1],[2],[],[],[]]
Output
[null,null,null,2,2,false]
Expected
[null,null,null,2,2,false]


Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
9.59 MB | Beats 17.85%
150%
100%
50%
0%
1ms
2ms
3ms
29ms

Q.4. Build an Array With Stack Operations

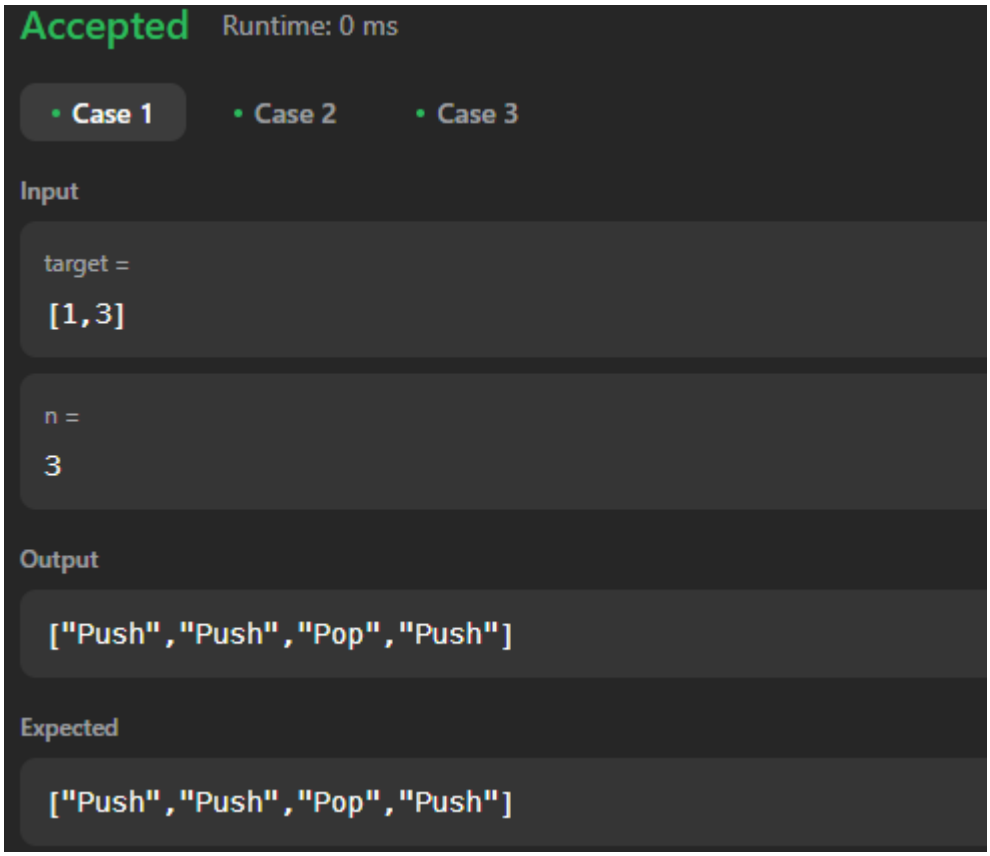
You are given an integer array target and an integer n. You have an empty stack with the two following operations:

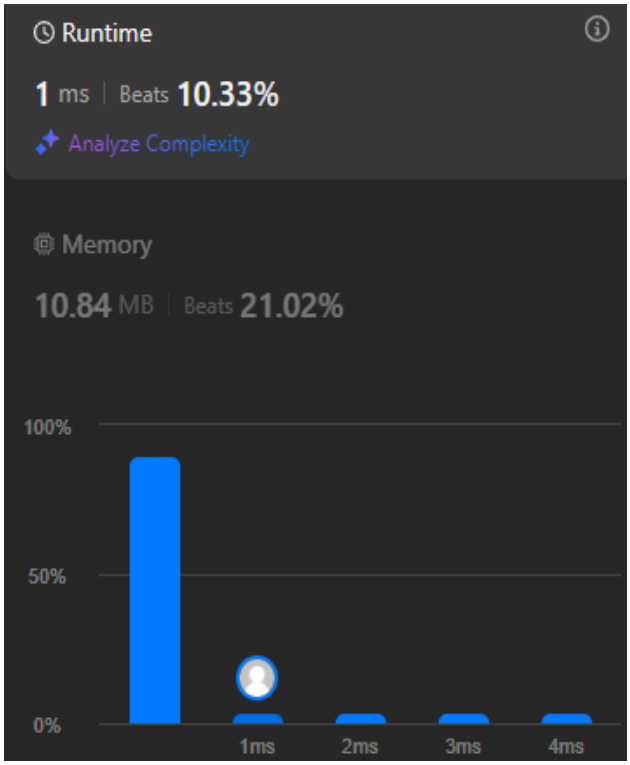
"Push": pushes an integer to the top of the stack.
"Pop": removes the integer on the top of the stack.

Code:

```
class Solution {
public:
    vector<string> buildArray(vector<int>& target, int n) {
        vector<string> operations;
        int index = 0; // Pointer for target array

        for (int i = 1; i <= n; i++) {
            if (index >= target.size()) break; // Stop when we have built the target array

            operations.push_back("Push"); // Always push

            if (target[index] == i) {
                index++; // Move to the next element in target
            } else {
                operations.push_back("Pop"); // Pop if the number is not in target
            }
        }

        return operations;
    }
};
```

Output:


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
target =
[1,3]
n =
3
Output
["Push","Push","Pop","Push"]
Expected
["Push","Push","Pop","Push"]


Runtime
1 ms | Beats 10.33%
Analyze Complexity
Memory
10.84 MB | Beats 21.02%
100%
50%
0%
1ms
2ms
3ms
4ms