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

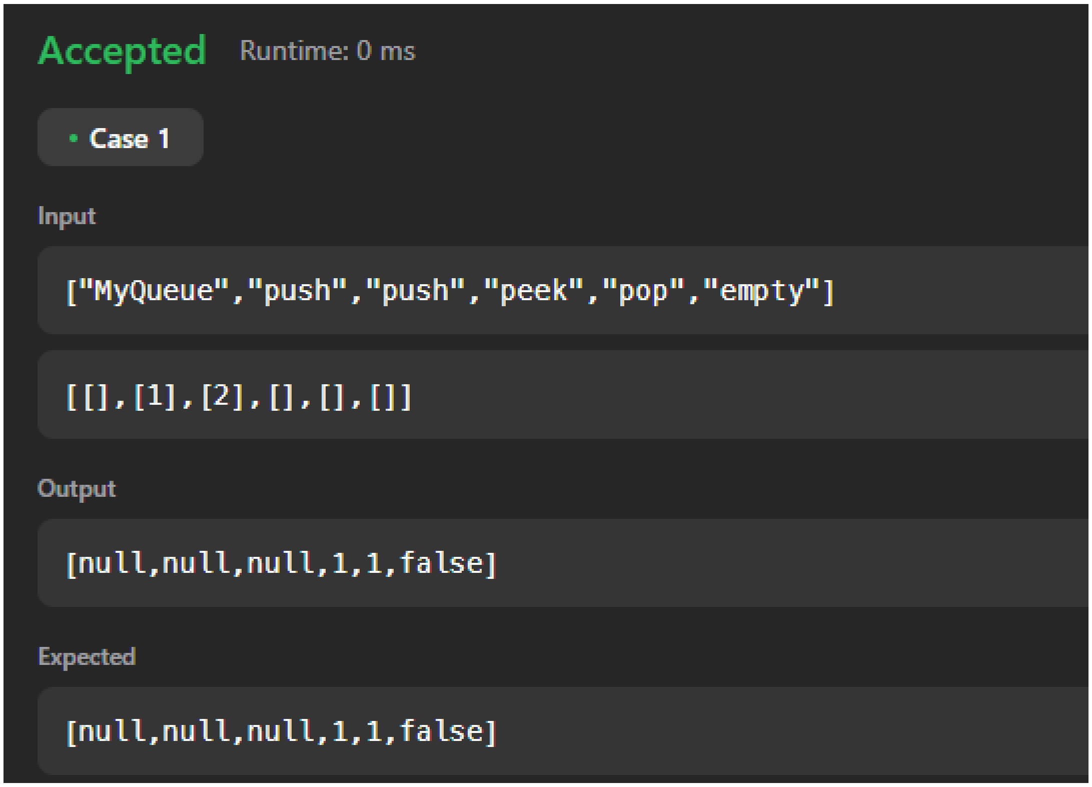
Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

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"]
```

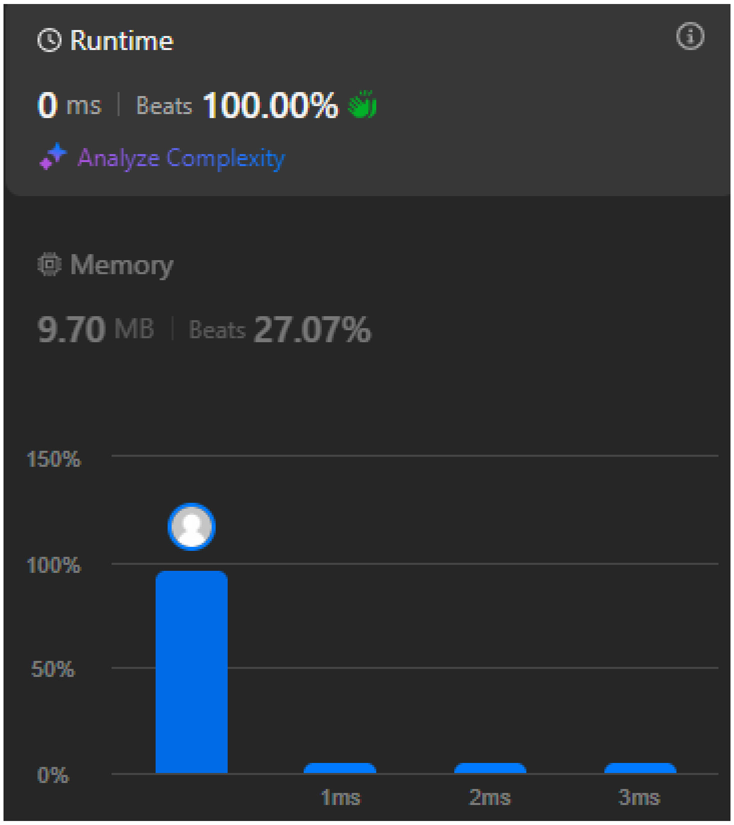
```
[[],[1],[2],[],[],[]]
```

Output

```
[null,null,null,1,1,false]
```

Expected

```
[null,null,null,1,1,false]
```



Runtime

0 ms | Beats 100.00%

Memory

9.70 MB | Beats 27.07%

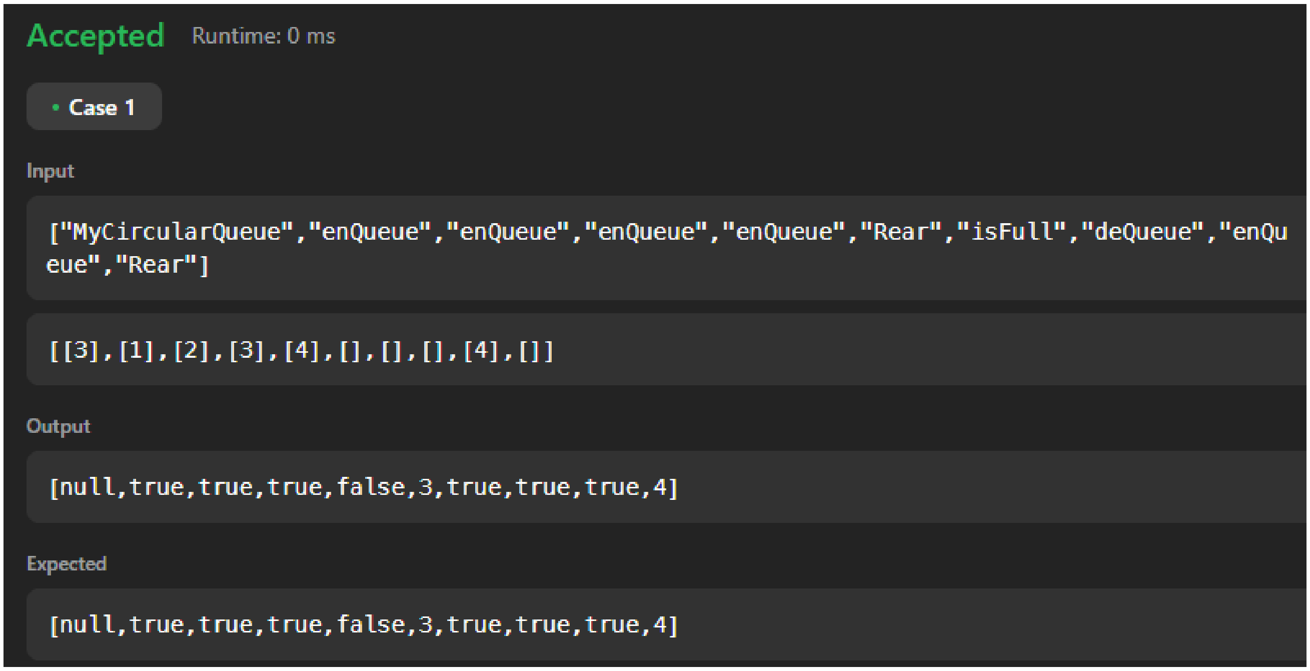
Q.2. Implement Circular Queue using Queue

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:



Accepted Runtime: 0 ms

• Case 1

Input

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

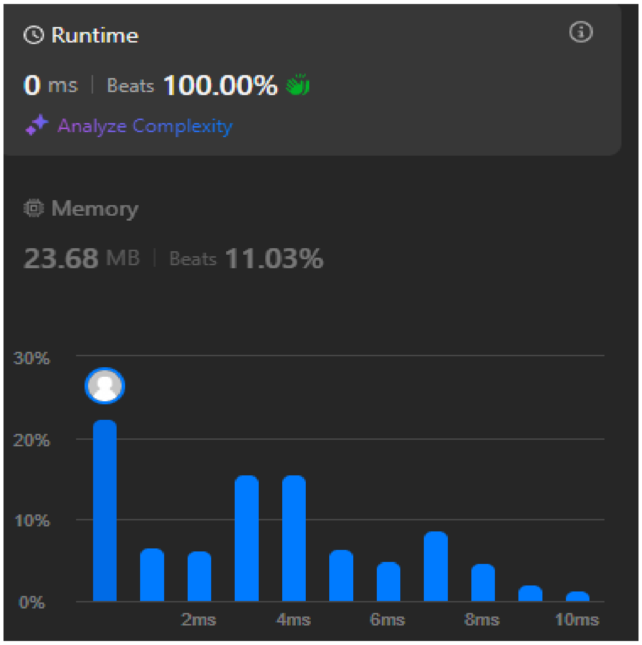
```
[[3],[1],[2],[3],[4],[],[],[],[4],[]]
```

Output

```
[null,true,true,true,false,3,true,true,true,4]
```

Expected

```
[null,true,true,true,false,3,true,true,true,4]
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

23.68 MB | Beats 11.03%

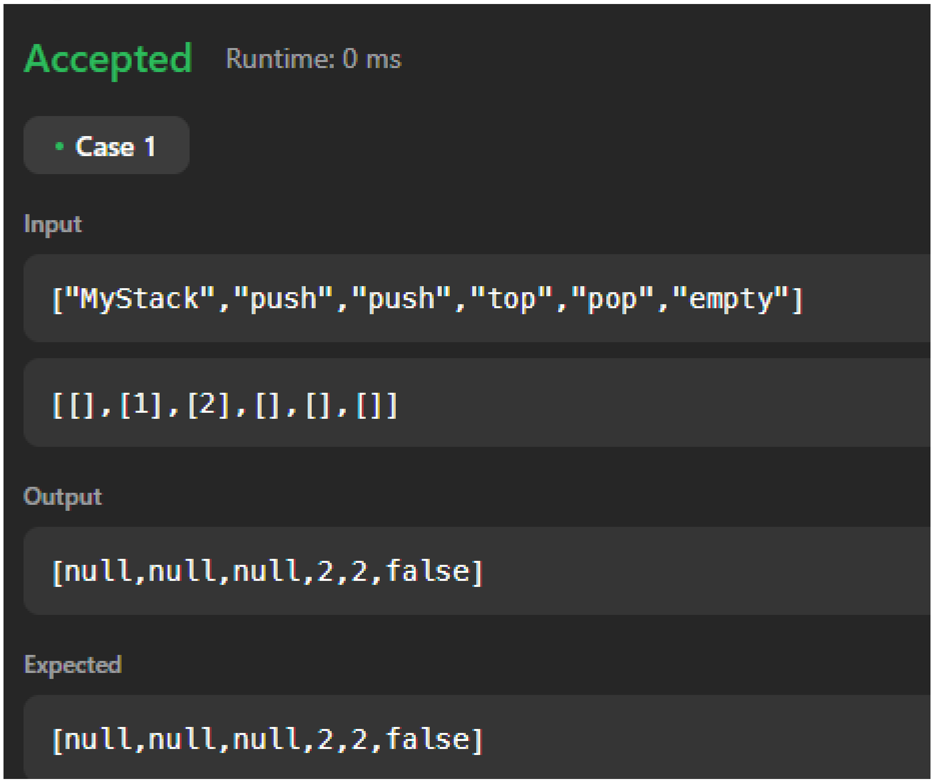
Q. 3. Implement Stack using Queues

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"]
```

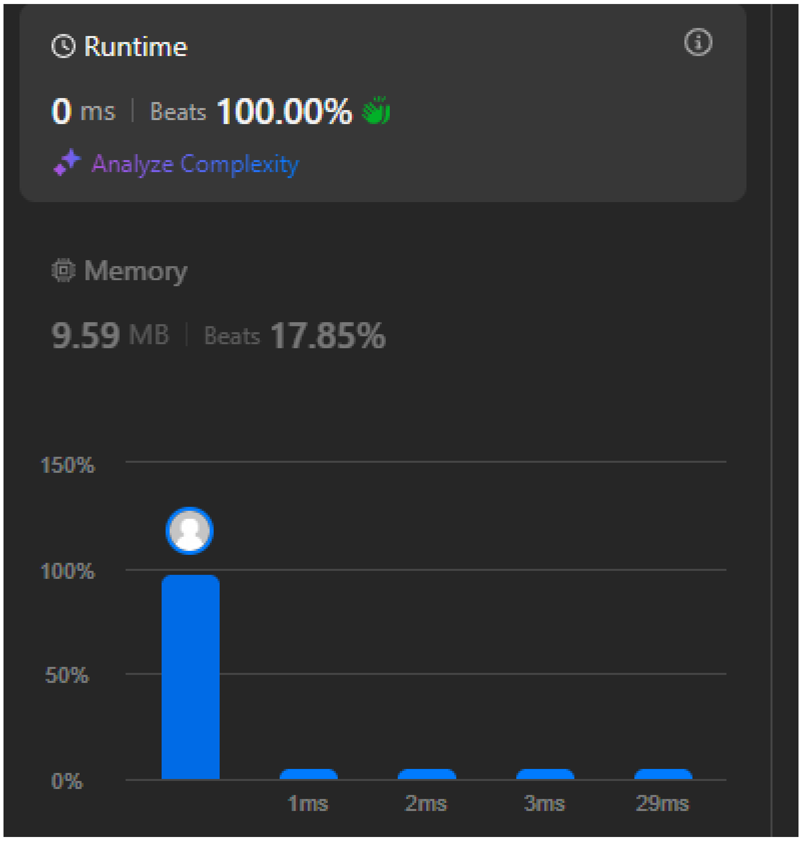
```
[[],[1],[2],[],[],[]]
```

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%

Q.4. Implement Stack using an Array

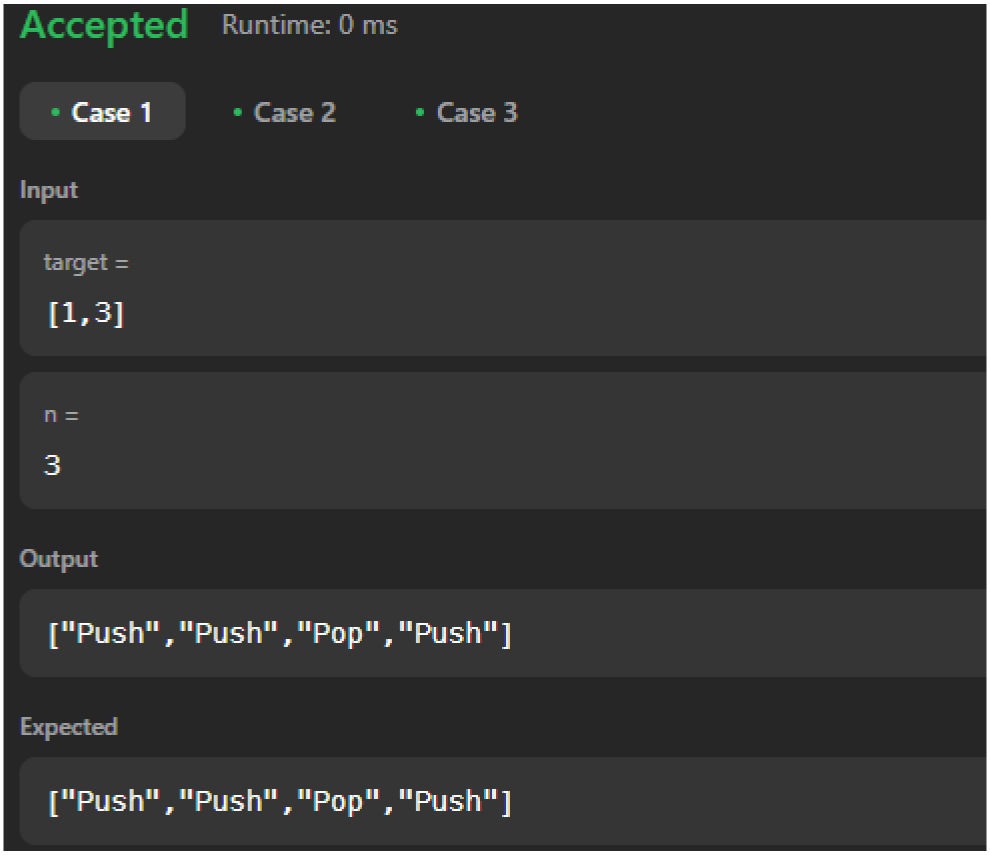
You are given an integer array target and an integer n. You have an empty stack with the two following operations:

"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 =

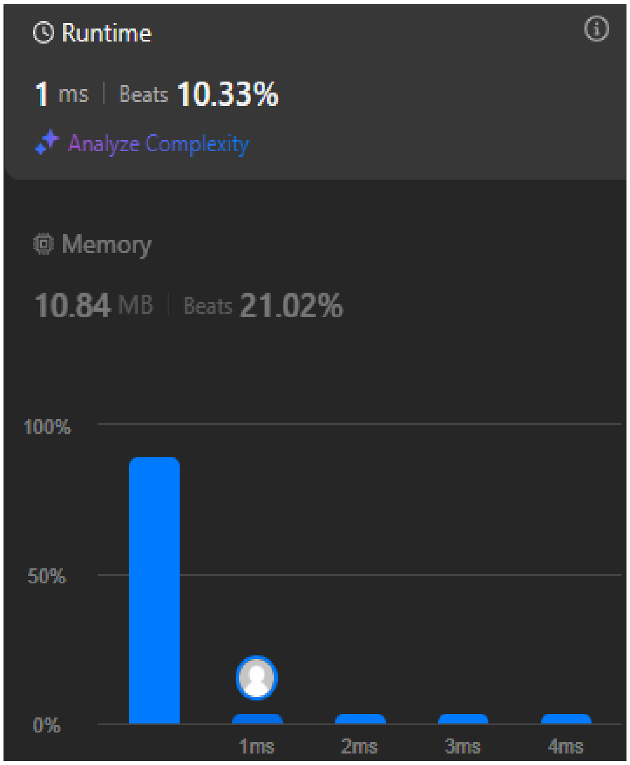
```
3
```

Output

```
["Push","Push","Pop","Push"]
```

Expected

```
["Push","Push","Pop","Push"]
```



Runtime

1 ms | Beats 10.33%

Analyze Complexity

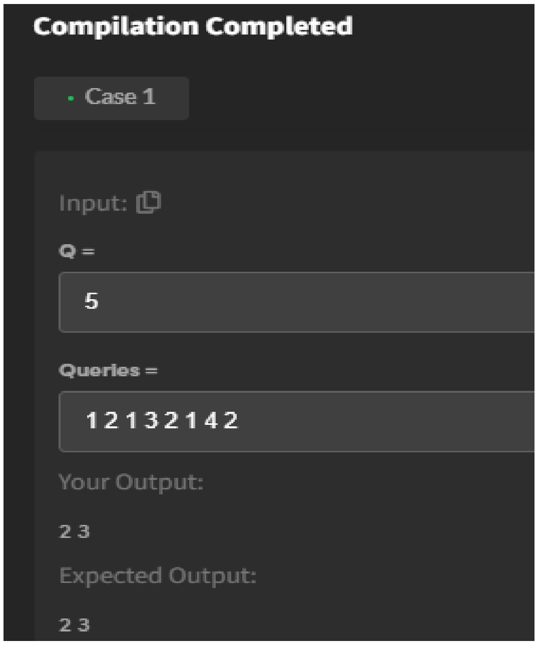
Memory

10.84 MB | Beats 21.02%

Q.5. Implement Queue Using Array

```
class MyQueue {
private:
    int arr[100005];
    int front;
    int rear;

public :
    MyQueue(){front=0;rear=0;}
    void push(int);
    int pop();
};
 */

// Function to push an element x in a queue.
void MyQueue ::push(int x) {
    arr[rear++] = x;
}

// Function to pop an element from queue and return that element.
int MyQueue ::pop() {
    if (front == rear) return -1; // Queue is empty
        return arr[front++];
}
```

Output:



Compilation Completed

Case 1

Input:

Q =

```
5
```

Queries =

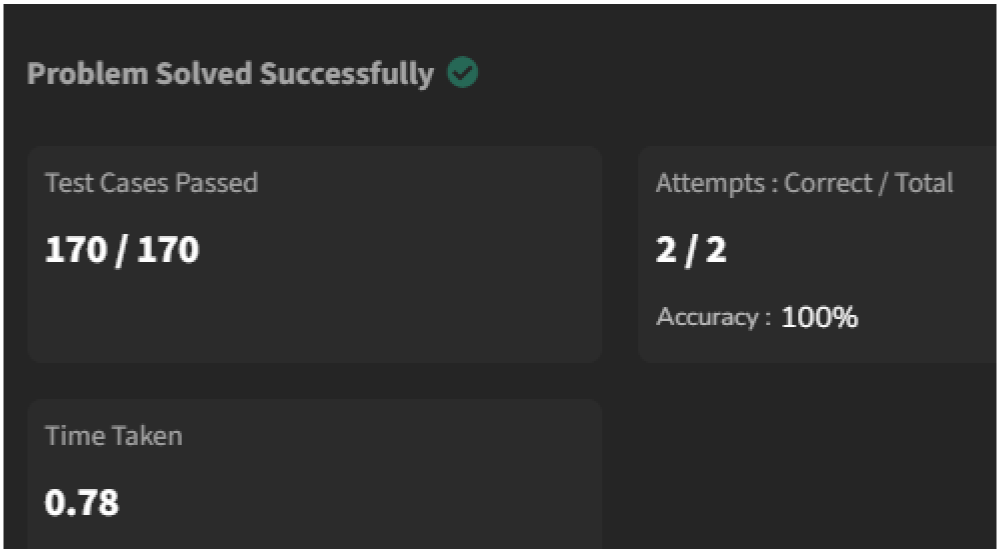
```
1 2 1 3 2 1 4 2
```

Your Output:

```
2 3
```

Expected Output:

```
2 3
```



Problem Solved Successfully ✓

Test Cases Passed

170 / 170

Attempts : Correct / Total

2 / 2

Accuracy : 100%

Time Taken

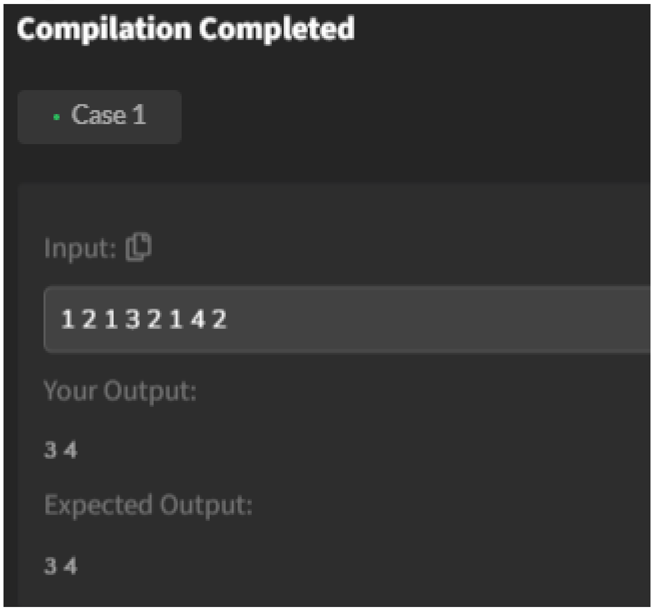
0.78

Q.6. Implement Stack using Linked List

Code:

```
class MyStack {
 private:
   StackNode *top;

 public:
   void push(int x) {
       StackNode* newNode = new StackNode(x);
       newNode->next = top;
       top = newNode;
   }

   // Function to remove and return the top element of the stack.
   int pop() {
       if (top == NULL) return -1; // Stack is empty

       int poppedData = top->data;
       StackNode* temp = top;
       top = top->next;
       delete temp; // Free memory
       return poppedData;
   }
};
```

Output:



Compilation Completed

Input:

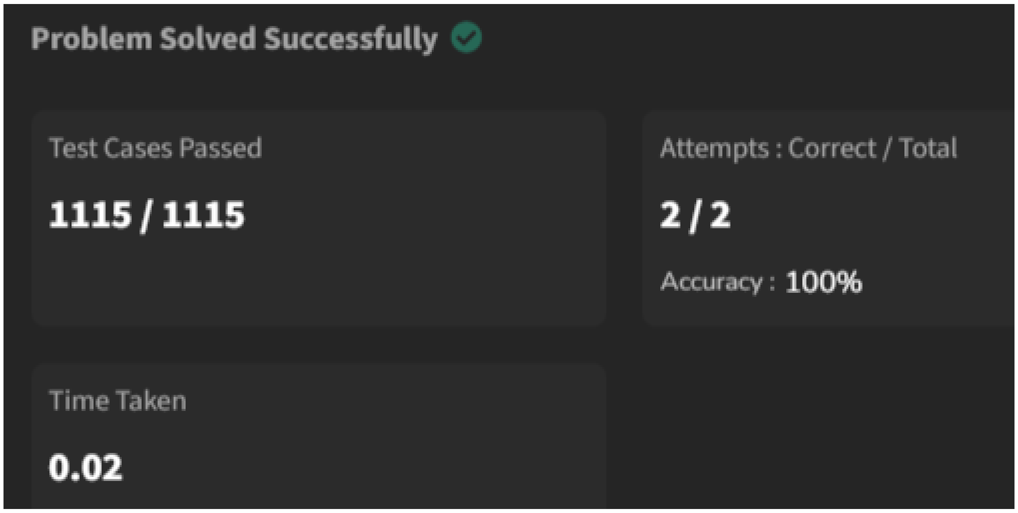
```
1 2 1 3 2 1 4 2
```

Your Output:

```
3 4
```

Expected Output:

```
3 4
```



Problem Solved Successfully ✓

Test Cases Passed

1115 / 1115

Attempts : Correct / Total

2 / 2

Accuracy : 100%

Time Taken

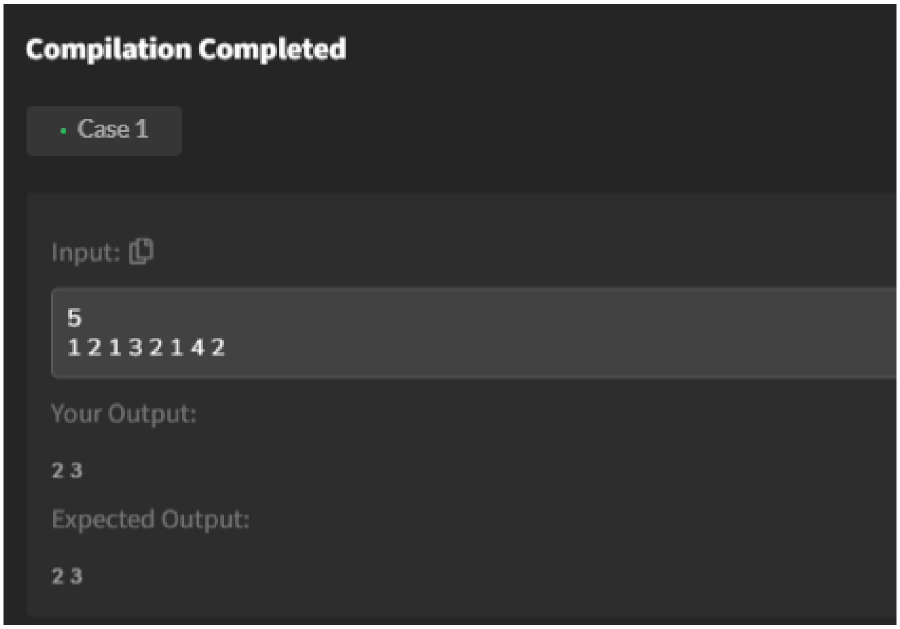
0.02

Q.7. Implement Queue using Linked List

Code:

```
void MyQueue::push(int x) {
    QueueNode* newNode = new QueueNode(x);

    if (rear == NULL) {
        front = rear = newNode;
    }
    else {
        rear->next = newNode;
        rear = newNode;
    }
}
   int MyQueue::pop() {
    if (front == NULL) return -1;

    int poppedData = front->data;
    QueueNode* temp = front;
    front = front->next;

    if (front == NULL) rear = NULL;

    delete temp;
    return poppedData;
}
```

Output:



Compilation Completed

Input:

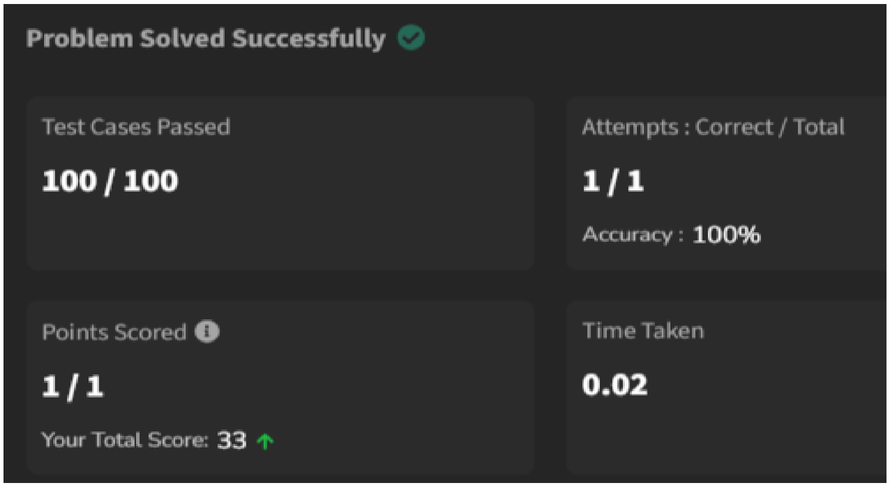
```
5
1 2 1 3 2 1 4 2
```

Your Output:

```
2 3
```

Expected Output:

```
2 3
```



Problem Solved Successfully

Test Cases Passed

100 / 100

Attempts : Correct / Total

1 / 1

Accuracy : 100%

Points Scored

1 / 1

Your Total Score: 33

Time Taken

0.02

Q.8. Implement BST using Linked List (Flattened Representation)

Given the root of a binary tree, flatten the tree into a "linked list":

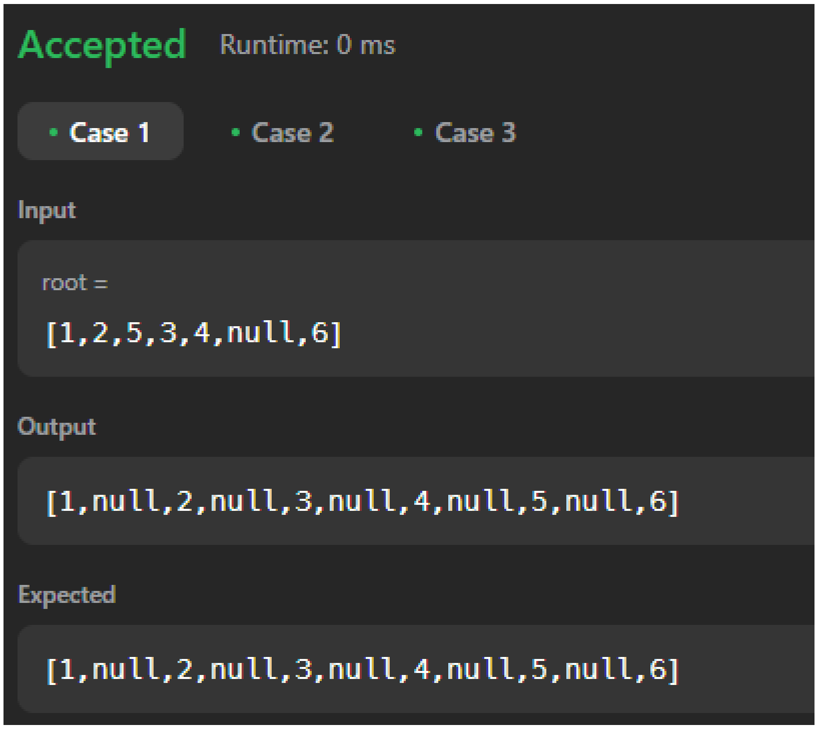
The "linked list" should use the same TreeNode class where the right child pointer points to the next node in the list and the left child pointer is always null.

The "linked list" should be in the same order as a pre-order traversal of the binary tree.

Code:

```
class Solution {
public:
    void flatten(TreeNode* root) {
        if (!root) return;

        TreeNode* curr = root;
        while (curr) {
            if (curr->left) {
                TreeNode* prev = curr->left;
                while (prev->right) {
                    prev = prev->right;
                }
                prev->right = curr->right;
                curr->right = curr->left;
                curr->left = nullptr;
            }
            curr = curr->right;
        }
    }
};
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

root =

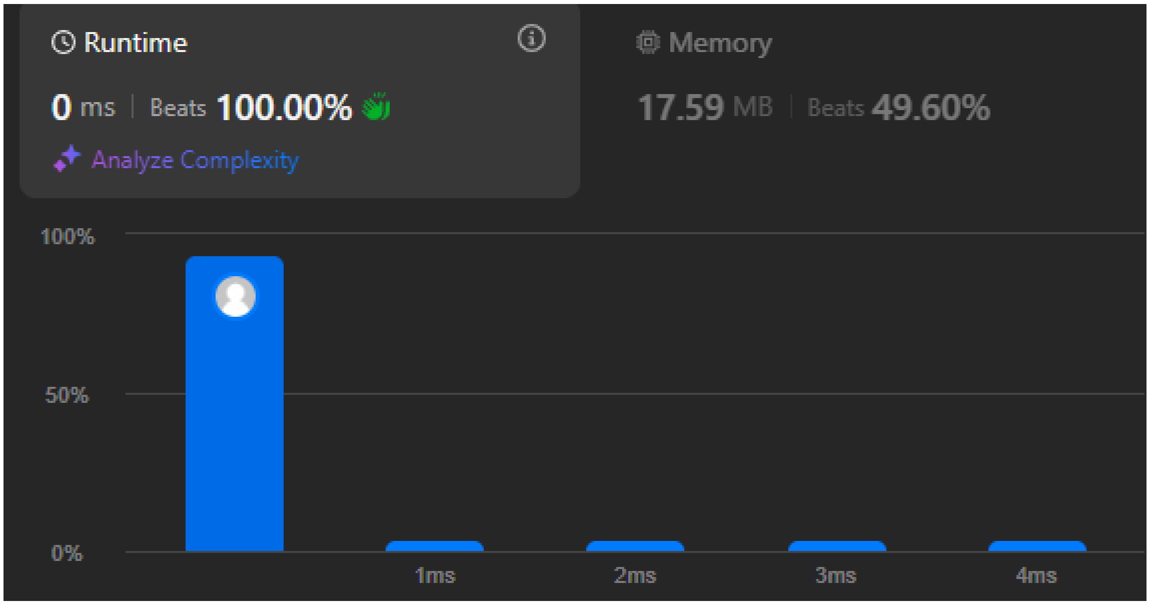
```
[1,2,5,3,4,null,6]
```

Output

```
[1,null,2,null,3,null,4,null,5,null,6]
```

Expected

```
[1,null,2,null,3,null,4,null,5,null,6]
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

17.59 MB | Beats 49.60%

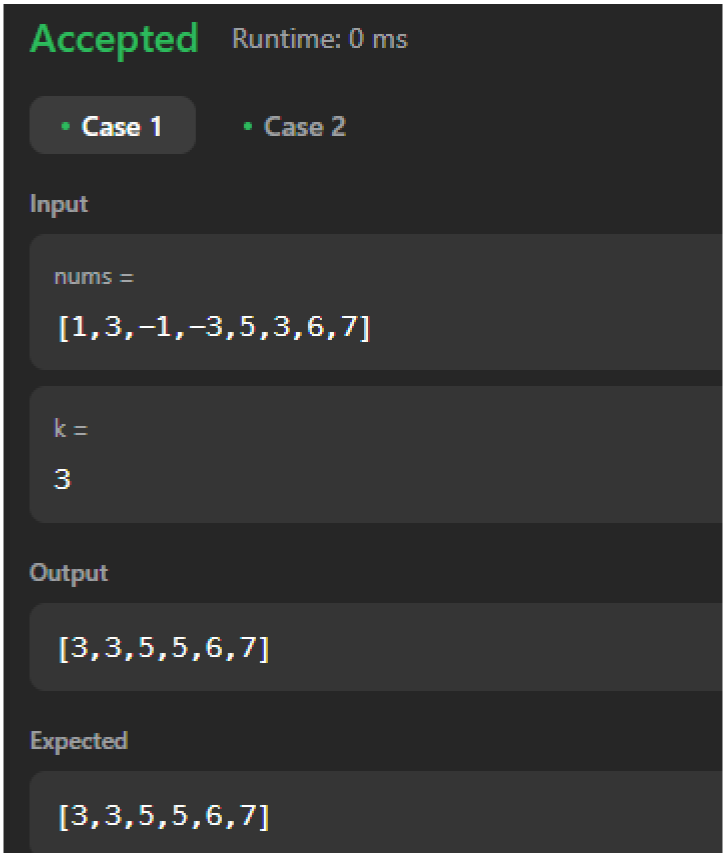
Q.9. Implement Sliding Window Maximum using Deque

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.

Code:

```
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        vector<int> result;
        deque<int> dq;

        for (int i = 0; i < nums.size(); i++) {
            if (!dq.empty() && dq.front() == i - k)
                dq.pop_front();

            while (!dq.empty() && nums[dq.back()] <= nums[i])
                dq.pop_back();

            dq.push_back(i);

            if (i >= k - 1)
                result.push_back(nums[dq.front()]);
        }

        return result;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

nums =
[1,3,-1,-3,5,3,6,7]

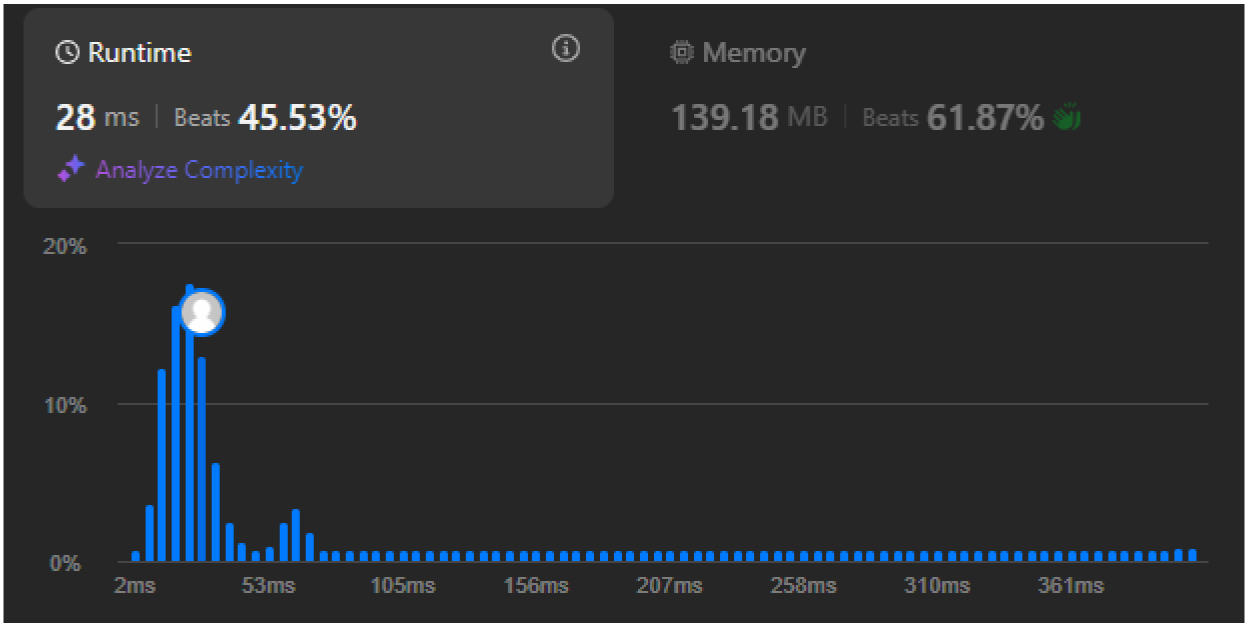
k =
3

Output

[3,3,5,5,6,7]

Expected

[3,3,5,5,6,7]



Runtime
28 ms | Beats 45.53%

Memory
139.18 MB | Beats 61.87%

Q.10. Implement LRU Cache using Hash Table + Doubly Linked List

Design a data structure that follows the constraints of a Least Recently Used (LRU) cache.

Implement the LRUCache class:

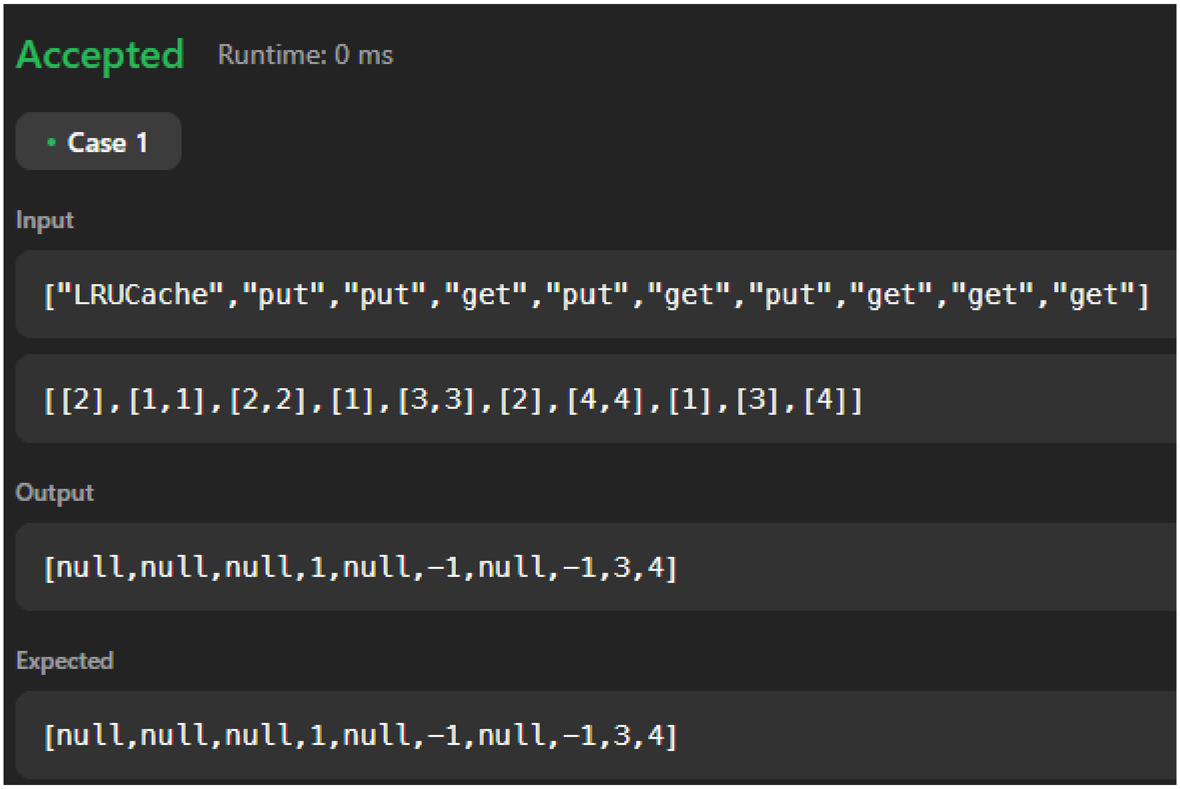
- LRUCache(int capacity) Initialize the LRU cache with positive size capacity.
- int get(int key) Return the value of the key if the key exists, otherwise return -1.
- void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, evict the least recently used key.

Code:

```
class LRUCache {
private:
    int cap;
    list<pair<int, int>> dll;
    unordered_map<int, list<pair<int, int>>::iterator> cache;

public:
    LRUCache(int capacity) : cap(capacity) {}

    int get(int key) {
        if (cache.find(key) == cache.end())
            return -1;
        dll.splice(dll.begin(), dll, cache[key]);
        return cache[key]->second;
    }

    void put(int key, int value) {
        if (cache.find(key) != cache.end()) {
            dll.splice(dll.begin(), dll, cache[key]);
            cache[key]->second = value;
            return;
        }
```

```
            if (dll.size() == cap) {
              cache.erase(dll.back().first);
              dll.pop_back();
            }
            dll.push_front({key, value});
            cache[key] = dll.begin();
        }
    };
```

Output:



Accepted Runtime: 0 ms

• Case 1

Input

```
["LRUCache","put","put","get","put","get","put","get","get","get"]
```

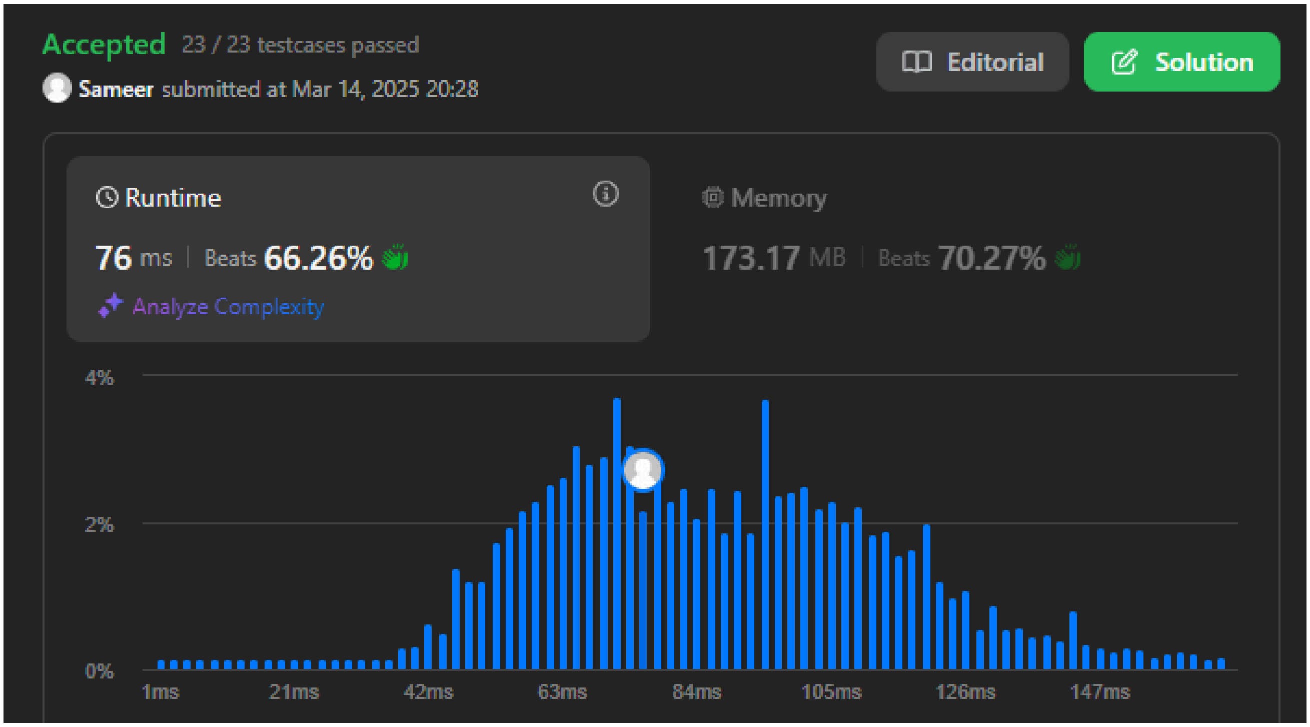
```
[[2],[1,1],[2,2],[1],[3,3],[2],[4,4],[1],[3],[4]]
```

Output

```
[null,null,null,1,null,-1,null,-1,3,4]
```

Expected

```
[null,null,null,1,null,-1,null,-1,3,4]
```



Accepted 23 / 23 testcases passed

Sameer submitted at Mar 14, 2025 20:28

Runtime

76 ms | Beats 66.26%

Memory

173.17 MB | Beats 70.27%