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

Experiment - 2

Student Name: **Aditya chandel** UID: **22BCS10109**

Branch: **BE - CSE** Section/Group: **NTPP 602 b**

Semester: **6** Sub Code: **22CSP-351**

Subject Name: **Advanced Programming Lab - 2** Date: **21-01-2025**

Problem - 1

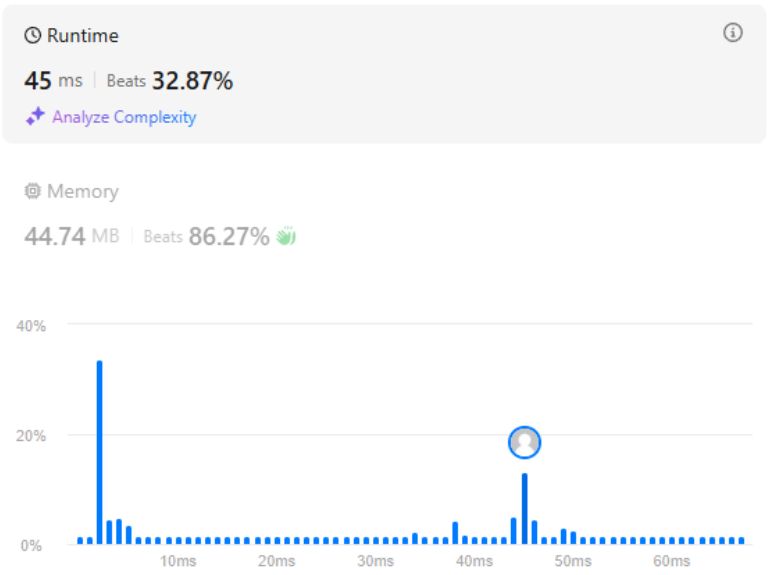
Aim - Given an array of integers nums and an integer target, return the indices of the two numbers such that they add up to target. Each input has exactly one solution, and you cannot use the same element twice.

Objective - The goal of this code is to return the indices of the numbers which add up to the target.

Code –

```
class Solution {
  public int[] twoSum(int[] nums, int target) {
     // Iterate through each pair of numbers
     for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
           // Check if the sum of the pair equals the target
           if (nums[i] + nums[j] == target) {
              return new int[] {i, j};
           }
        }
     }
     // Return an empty array if no solution is found (though problem guarantees one)
     return new int[] {};
  }
}
```

Output -



Time Complexity : O(n) **Space Complexity -** O(1)

Problem - 4

Aim - 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).

Objective - Implement the MyQueue class. void push(int x) Pushes element x to the back of the queue. int pop() Removes the element from the front of the queue and returns it. int peek() Returns the element at the front of the queue. boolean empty() Returns true if the queue is empty, false otherwise.

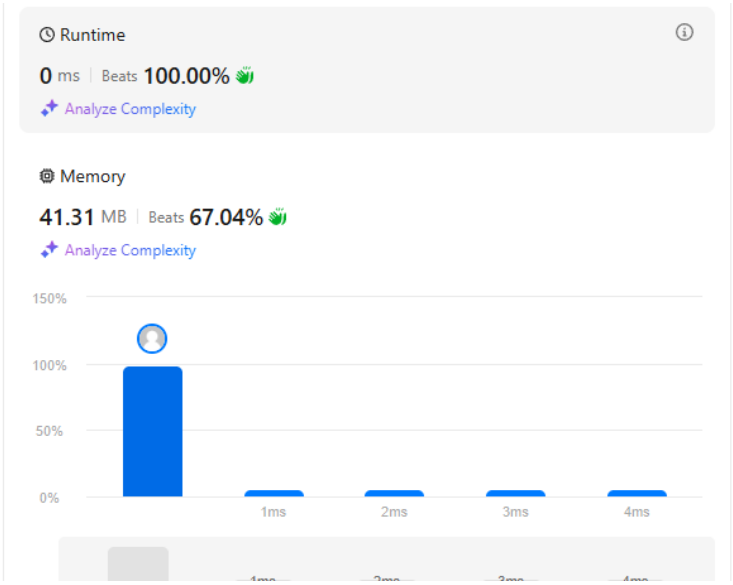
Code –

```
class MyQueue {
    int[] queue;
    int front;
    int rear;
    int size;
```

```
public MyQueue() {
    queue = new int[100];
    front = 0;
    rear = 0;
    size = 0;
}

// Adds an element to the back of the queue
public void push(int x) {
    if (size == queue.length) {
        System.out.println("Queue is full");
        return;
    }
    queue[rear] = x;
    rear = (rear + 1) % queue.length;
    size++;
}
public int pop() {
    if (size == 0) {
        System.out.println("Queue is empty");
        return -1;
    }
    int result = queue[front];
    front = (front + 1) % queue.length;
    size--;
    return result;
}

public int peek() {
    if (size == 0) {
        System.out.println("Queue is empty!");
        return -1;
    }
    return queue[front];
}

public boolean empty() {
```

```
        return size == 0;
    }
}

public class Main {
    public static void main(String[] args) {
        MyQueue obj = new MyQueue();
        obj.push(1);
        obj.push(2);
        System.out.println(obj.peek());
        System.out.println(obj.pop());
        System.out.println(obj.empty());
    }
}
```

Output –



Time Complexity:- O(1).

Space Complexity:- O(1)

Learning Outcomes -

1. Understanding Data Structures: Working on these questions helps solidify your understanding of fundamental data structures. For instance, stacks operate on a Last In, First Out (LIFO) principle, while queues follow a First In, First Out (FIFO) approach.
2. Problem-Solving Skills: Tackling problems involving these data structures enhances your problem-solving abilities.
3. Algorithmic Thinking: This fosters algorithmic thinking, as you must consider time and space complexity, especially when comparing array-based implementations to linked list implementations.
4. Practical Coding Skills: By coding solutions for stack and queue problems, you gain hands-on experience with syntax and programming concepts.