Experiment 4

Student Name: Rahul Prasad **UID:** 22BET10167

Branch: IT **Section/Group:** 701/A

Semester: 6th **Date :** 15/02/2025

Subject Name: Advanced Programming Lab - 2 **Subject Code:** 22ITP-351

1. Problem 1:

➢ Number of 1 Bits:

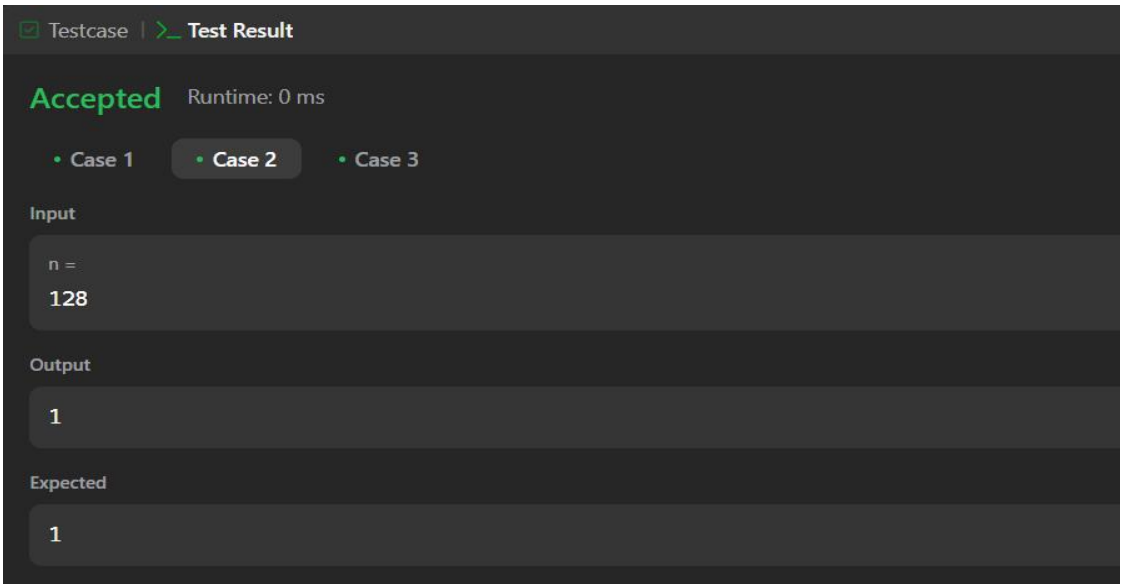
Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

➢ Code:

```
class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while(n != 0){
            if(n%2 == 1){
                count ++;
            }
            n = n/2;
        }
        return count;
```

```
    }
}
```

➢ **Output**



2. **Problem 2:**

➢ **Search a 2D Matrix II:**

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.

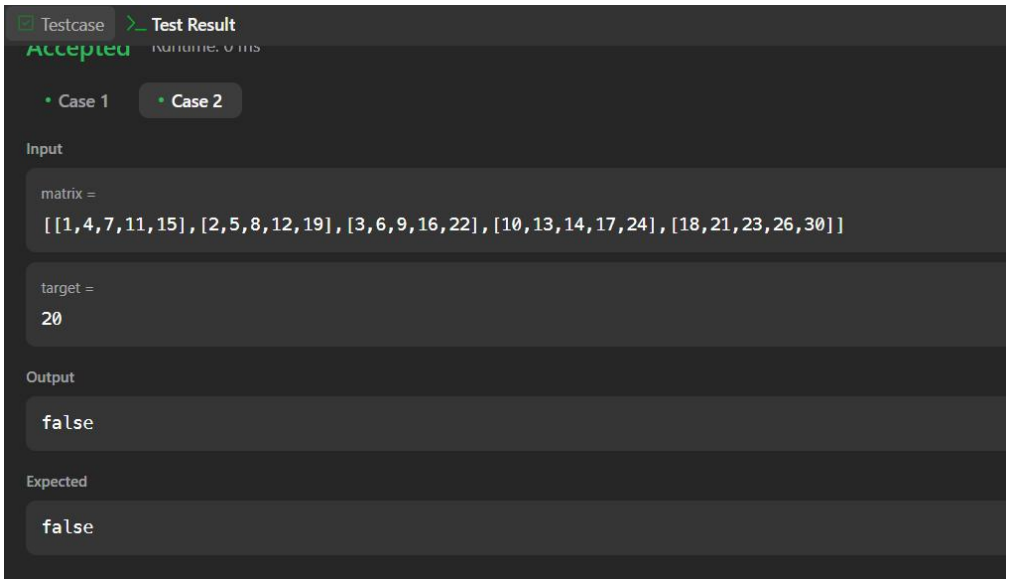
Integers in each column are sorted in ascending from top to bottom.

➢ **Code:**

```
class Solution {

    public boolean searchMatrix(int[][] matrix, int target) {
```

```
        int n = matrix.length;
        int m = matrix[0].length;
        int row = n-1;
        int col = 0;

        while(row>=0 && col <m){
            if(matrix[row][col] == target){
                return true;
            }
            else if(matrix[row][col]>target){
                row--;
            }
            else{
                col++;
            }
        }
        return false;
    }
}
```

➢ **Output:**



3. Problem - 3:

➢ **Max Subarray:**

To find the contiguous subarray within a one dimensional array of numbers that has the largest sum.

➢ **Code:**

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }

        return maxSum;
    }
}
```

➢ **Output:**

4. **Problem - 4:**

➢ **Super Pow:**

To compute a large power of a number efficiently, given the base and an array of exponents.

➢ **Code:**

```
class Solution {
    private static final int MOD = 1337;

    private int pow(int a, int b) {
        int result = 1;
        a %= MOD;

        for (int i = 0; i < b; i++) {
            result = (result * a) % MOD;
        }

        return result;
    }

    public int superPow(int a, int[] b) {
        int result = 1;
```

```
        for (int i = b.length - 1; i >= 0; i--) {
            result = (result * pow(a, b[i])) % MOD;
            a = pow(a, 10);
        }

        return result;
    }
}
```

➢ **Output:**

5. **Problem - 5:**

➢ **Beautiful Array:**

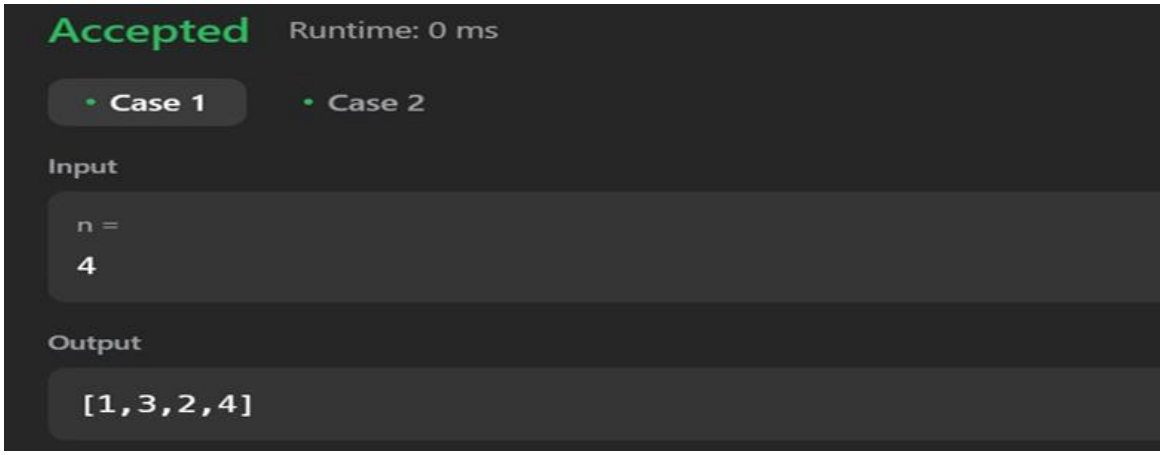
To generate an array of integers that satisfies specific conditions regarding the arrangement of odd and even numbers.

➢ **Code:**

```
import java.util.*;

class Solution {
    public int[] beautifulArray(int N) {
        return helper(N).stream().mapToInt(i -> i).toArray();
    }
```

```
    private List<Integer> helper(int n) {
        if (n == 1) {
            return Arrays.asList(1);
        }

        List<Integer> odd = helper((n + 1) / 2);
        List<Integer> even = helper(n / 2);

        List<Integer> result = new ArrayList<>();
        for (int x : odd) {
            result.add(x * 2 - 1);
        }
        for (int x : even) {
            result.add(x * 2);
        }

        return result;
    }
}
```

- **Output:**



6. Problem - 6:

➢ The Skyline Problem:

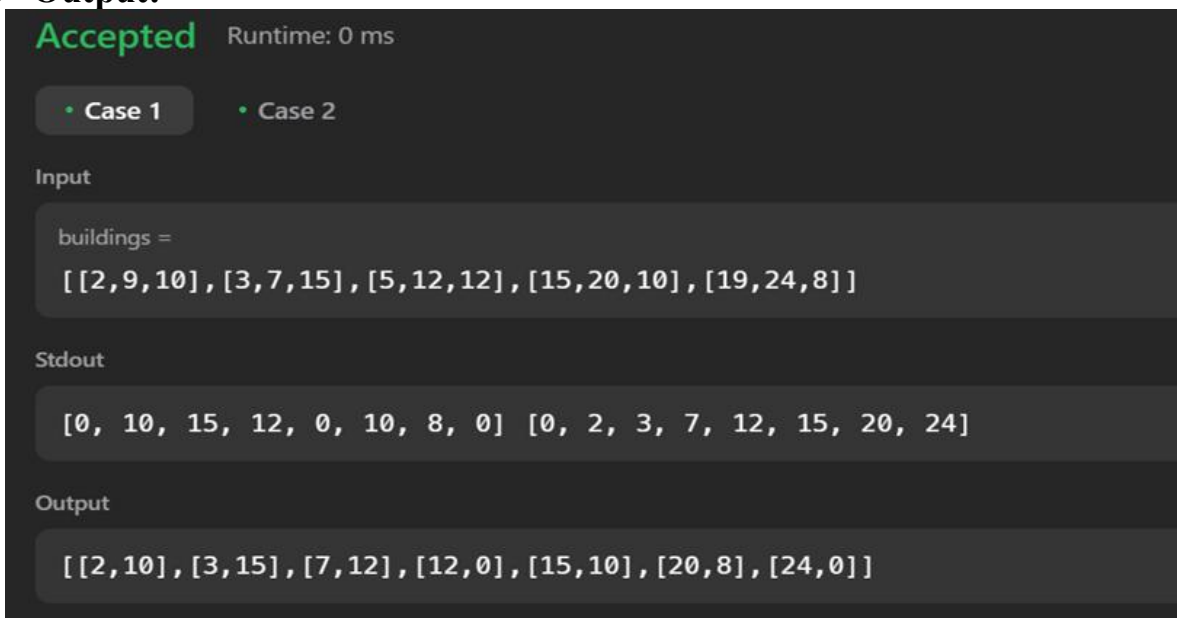
To determine the outline of a city skyline formed by a collection of rectangular buildings.

➢ Code:

```
import java.util.*;

class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> events = new ArrayList<>();
        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2]}); // Start of building
            events.add(new int[]{b[1], b[2]});  // End of building
        }

        // Sort events: first by position, then height (start before end)
        events.sort((a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) :
                Integer.compare(a[0], b[0]));

        List<List<Integer>> res = new ArrayList<>();
        TreeMap<Integer, Integer> heights = new
TreeMap<>(Collections.reverseOrder());
        heights.put(0, 1); // Sentinel value
        int prevHeight = 0;

        for (int[] e : events) {
            if (e[1] < 0) {
                // Start of a building
                heights.put(-e[1], heights.getOrDefault(-e[1], 0) + 1);
            } else {
                // End of a building
                heights.put(e[1], heights.get(e[1]) - 1);
                if (heights.get(e[1]) == 0) {
                    heights.remove(e[1]);
                }
            }

            int currentHeight = heights.firstKey();
            if (currentHeight != prevHeight) {
```

```
                res.add(Arrays.asList(e[0], currentHeight));
                prevHeight = currentHeight;
            }
        }

        return res;
    }
}
```

➢ **Output:**



❖ **Leraning Outcomes:**

- ➢ Understand the use of data structures like TreeMap for maintaining ordered values dynamically.
- ➢ Improve problem-solving skills by implementing mathematical logic and optimization techniques.
- ➢ Enhance understanding of space and time complexity analysis in algorithms.
- ➢ Strengthen knowledge of handling large numbers and preventing integer overflow.
- ➢ Gain experience in converting Python implementations to Java while maintaining logic efficiency.