Experiment: -4

Student Nam e Sahil **UID** 22BCS15101
Branch: BE-CSE **Section/ Group:** 22BCS_IOT_641 A
Semester: 6th **Date of Performance:** 20/2/2025
Subject Name: Advanced Programming Lab-2 **Subject Code:** 22CSP-351

Problem – 1

1. **Aim:** Reverse bits of a given 32 bits unsigned integer.
2. **Objective:**
   - **Understanding Bit Manipulation**:- The goal is to learn how to work with bits by reversing the order of 32 bits in a given number. This helps in understanding how data is stored and processed at the binary level.
   - **Improving Logical Thinking**:- Reversing bits requires breaking the problem into small steps, such as shifting and masking bits. This improves problem-solving skills and logical thinking in programming.
   - **Optimizing Performance**:_-The task helps in learning efficient ways to reverse bits using bitwise operations instead of using loops or strings. Faster solutions are useful in time-sensitive applications.
   - **Enhancing Knowledge of Binary Representation**:- By reversing bits, we get a new number with a different value, which helps in understanding how numbers change in binary form and how computers process them.
   - **Practical Use in Real Applications**:- Reversing bits is used in cryptography, data compression, and network communication. Learning this concept helps in understanding its real-world importance in computer science.
3. **CODE:**

```
class Solution
{
public:
   uint32_t reverseBits(uint32_t n)
   {
      uint32_t reversed = 0;
      for (int i = 0; i < 32; i++)
      {
         reversed <<= 1;
         reversed |= (n & 1);
         n >>= 1;
      }
      return reversed;
   }
};
```

4. Output:

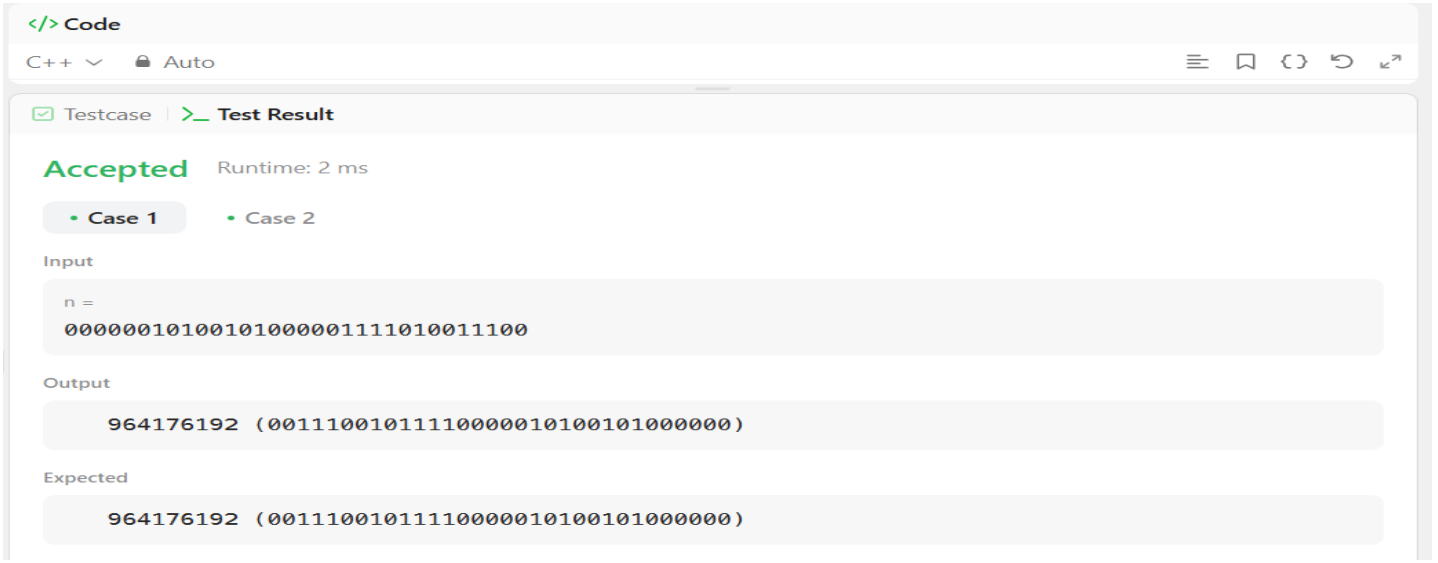

Figure 1: Sample Output

5. Learning Outcomes:

- **Understanding How Data is Stored in Computers**:- You will learn how computers represent numbers using bits and how changing their order affects the final value.
- **Gaining Hands-On Experience with Bitwise Operations**:- You will practice using bitwise operations like shifting and masking to manipulate individual bits in a number.
- **Developing a Step-by-Step Approach to Problem Solving**:- You will improve your ability to break down complex problems into smaller steps, making it easier to find an efficient solution.
- **Writing Faster and More Memory-Efficient Code**:- You will understand how to perform operations on bits without using extra memory, leading to better-performing programs.
- **Exploring Real-World Uses of Bit Manipulation**:- You will discover how bit manipulation is applied in various fields like data compression, cryptography, and computer graphics.

Problem – 2

1. **Aim:** 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).

2. Objective:

- **Understand the Task**:- The goal is to count how many times 1 appears in the binary form of a given number n. This count is called the Hamming weight**.**
- **Check Each Bit**:- The number is already stored in binary**,** so we will check each digit

(bit) one by one to see if it is 1 or 0**.**

- **Use Simple Bitwise Trick**:- We check the last digit (bit) of **n** using **n & 1**. If it is **1, we** increase the count**.** Then, we move to the next bit by shifting **n** to the right.
- **Work for Any Number**:- The function will work for all numbers**,** no matter how big or small, and correctly count the number of 1s in their binary form.
- **Fast and Efficient**:- The function uses no extra memory and runs very fast because it checks only the necessary bits**.** This makes it a quick and smart way to count 1s.

3. CODE:-

```
class Solution
{
public:
   int hammingWeight(int n)
   {
      int bits = 0;
      while (n != 0)
      {
         bits += (n & 1);
         n >>= 1;
      }
      return bits;
   }
};
```

4. OUTPUT

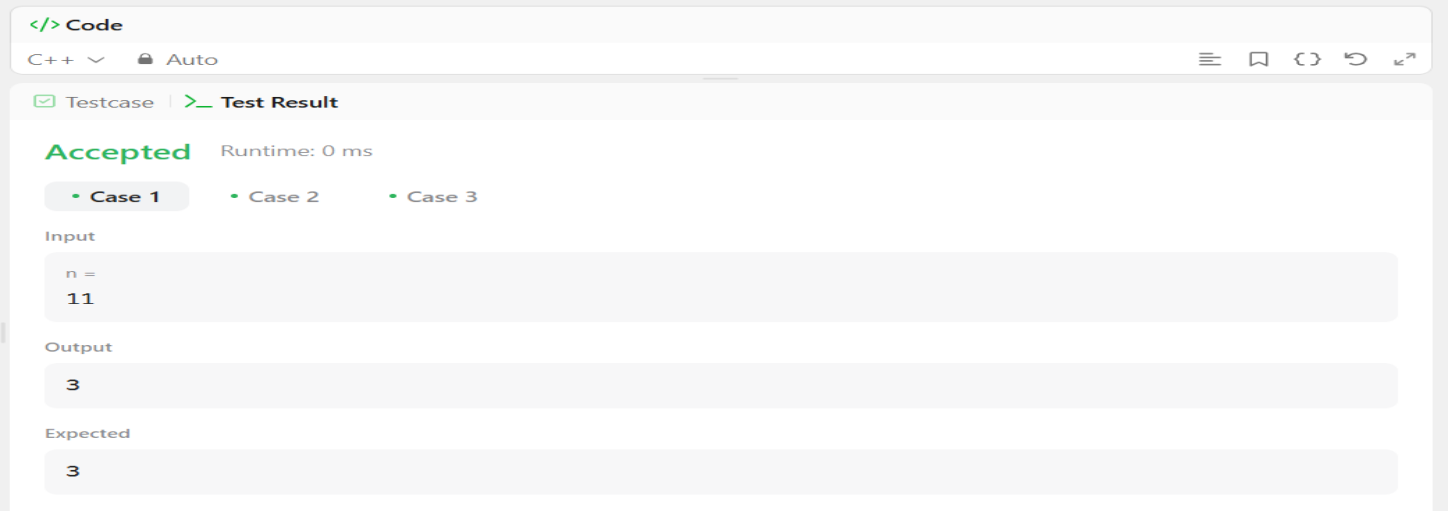

Figure 2: Sample Output

5. Learning Outcomes:

- **Understanding Binary Numbers**:- You will learn how numbers are represented in binary form**,** which consists of only 0s and 1s. This is important in computer science and digital systems.
- **Using Bitwise Operators**:- You will understand how the bitwise AND (**&**) **and** right shift (>>) operators work. These help check and move bits efficiently in a number.
- **Counting Set Bits (1s) Efficiently**:- You will learn how to count the number of **1s** in binary form without converting the number into a string, making it fast and memory-efficient**.**
- **Looping Until All Bits Are Processed**:- You will see how a while loop helps process a number bit by bit until all digits are checked. This ensures we count all 1s correctly.
- **Writing Optimized Code**:- You will learn how to write simple and efficient code that performs well for **large numbers**, using only basic operations without extra memory.

Problem – 3

1. Aim:- Given an integer array nums, find the subarray with the larges sum, and return its sum.

2. Objectives:-

- **Find the Largest Sum:-**The program finds a continuous part of the list that has the biggest total when added. It ensures that the chosen subarray gives the highest possible sum.
- **Check Subarrays Efficiently:**- Instead of checking every possible subarray, it uses a smart approach to find the answer quickly. This saves time and avoids unnecessary calculations.
- **Use Kadane's Algorithm:**-The program updates the sum as it moves through the list, keeping track of the highest sum. This method ensures an optimized and efficient solution.
- **Handle Negative Numbers:**- If all numbers are negative, it picks the least negative number as the result. This ensures the program works correctly in all cases.
- **Return the Final Sum:**- After processing the list, it returns the highest sum found. This helps in identifying the most valuable subarray.

3. CODE:

```
class Solution

{

public:

   int maxSubArray(vector<int> &nums)
```

```
    {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.size(); ++i)
        {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};
```

4. OUTPUT:-

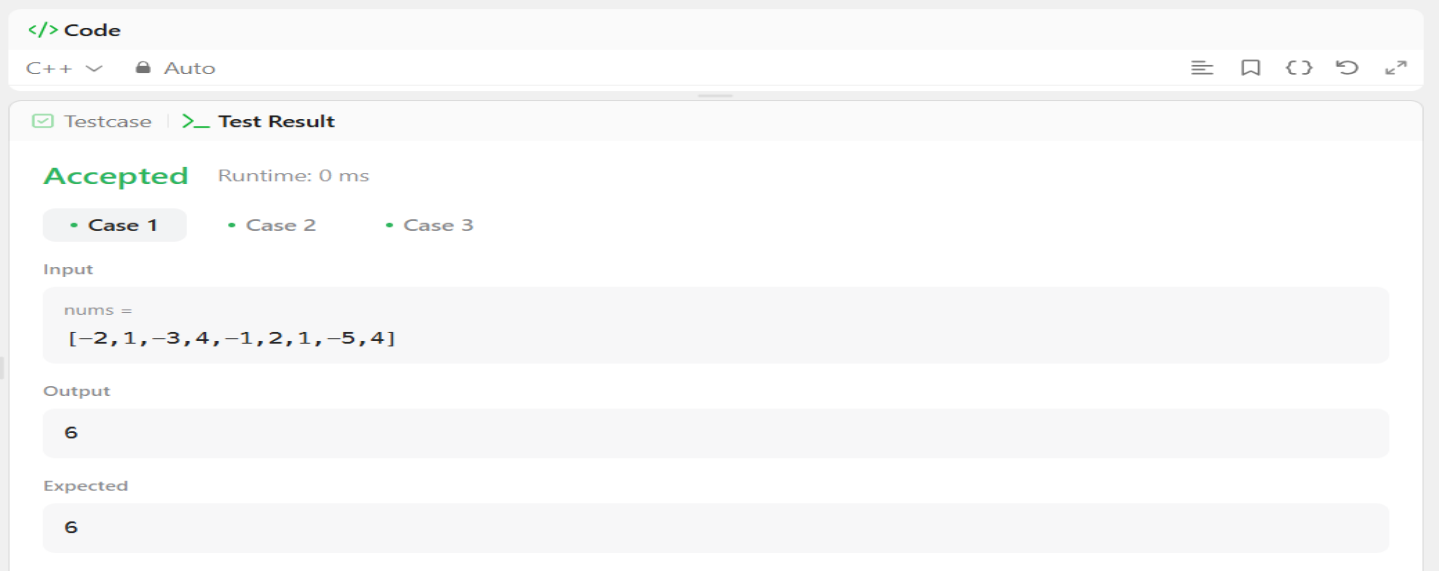

Figure 1: Sample Output

5. Learning Outcomes:-

- **Understanding Subarrays:** You will learn how to find a continuous part of an array that gives the highest sum. This helps in solving many real-world problems.
- **Efficient Problem Solving:** You will understand how to solve problems faster by avoiding unnecessary calculations. This improves coding efficiency.
- **Kadane's Algorithm Concept:** You will learn a powerful technique that keeps track of the highest sum while scanning the array. This method is widely used in competitive programming.

- **Handling Edge Cases:** You will understand how to deal with arrays that contain negative numbers. This makes the program more reliable and accurate.
- **Optimized Coding Skills:** You will improve your ability to write optimized and efficient code. This is important for solving complex problems quickly.