Prajwalhonashetti143@gmail.com

Day 12:

Task 1: Bit Manipulation Basics

Create a function that counts the number of set bits (1s) in the binary representation of an integer. Extend this to count the total number of set bits in all integers from 1 to n.

```
public class TotalSetBits {
    public static int totalSetBits(int n) {
        int totalBits = 0;
        for (int i = 1; i <= n; i++) {
            totalBits += countSetBits(i);
        }
        return totalBits;
    }

    private static int countSetBits(int n) {
        return Integer.bitCount(n);
    }

    public static void main(String[] args) {
        int n = 10;
        System.out.println(totalSetBits(n));

    }
}
```

Explanation:

1 **totalSetBits method:**

- **totalSetBits**(int n) iterates through numbers from 1 to n.
- For each number i, it calls the **countSetBits**(i) method to count the set bits in i and accumulates this count in **totalBits**.
- Finally, it returns **totalBits**, which is the total number of set bits in the binary representations of the integers from 1 to n.

countSetBits method:

- **countSetBits**(int n) simply calls **Integer.bitCount**(n), which counts the set bits in the integer n.

main method:

- In the main method, we set n = 10 and print the result of **totalSetBits**(n).
- The expected output for n = 10 is 17, which represents the total number of set bits in the binary representations of the integers from 1 to 10.

Example Output

When we run the main method with n = 10, the output will be:

17

This output means that the total number of set bits in the binary representations of the integers from 1 to 10 is 17.

Binary representation and set bits for numbers from 1 to 10:

1 (binary **0001**) has 1 set bit.

2 (binary **0010**) has 1 set bit.

3 (binary **0011**) has 2 set bits.

4 (binary **0100**) has 1 set bit.

5 (binary **0101**) has 2 set bits.

6 (binary **0110**) has 2 set bits.

7 (binary **0111**) has 3 set bits.

8 (binary **1000**) has 1 set bit.

9 (binary **1001**) has 2 set bits.

10 (binary **1010**) has 2 set bits.

Task 2: Unique Elements Identification

Given an array of integers where every element appears twice except for two, write a function that efficiently finds these two non-repeating elements using bitwise XOR operations

Explanation

Understanding XOR properties:

- XOR of a number with itself is 0: a ^ a = 0.
- XOR of a number with 0 is the number itself: a ^ 0 = a.
- XOR is both commutative and associative: a ^ b ^ a = b ^ (a ^ a) = b.

```
public class TwoNonRepeatingElements {

  public static void findNonRepeating(int[] nums) {

    int xor = 0;


    // Step 1: Get the XOR of all elements

    for (int num : nums) {

      xor ^= num;

    }


    // Step 2: Find any set bit in xor (any bit where x and y differ)

    int bitMask = 1;

    while ((bitMask & xor) == 0) {

      bitMask <<= 1;

    }


    int x = 0;

    int y = 0;
```

```
        // Step 3: Partition the numbers into two groups and find the non-repeating elements
        for (int num : nums) {
            if ((num & bitMask) != 0) {
                x ^= num;
            } else {
                y ^= num;
            }
        }

        System.out.println("Non-repeating elements are: " + x + " and " + y);
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 2, 1, 4};
        findNonRepeating(nums);  // Output: Non-repeating elements are: 3 and 4
    }
}
```

Output:

The output of the program will be

Non-repeating elements are: 3 and 4

Let's continue with the example array nums = {1, 2, 3, 2, 1, 4} and the xor value 7 (0111 in binary).

Initialization:

Start with bitMask = 1 (0001 in binary).

Loop to find the set bit:

First iteration:

(bitMask & 0111) = (0001 & 0111) = 0001

Since this is not zero, continue.

Second iteration:

bitMask is shifted left: 0010

(0010 & 0111) = 0010

This is still not zero, continue.

Third iteration:

bitMask is shifted left: 0100

(0100 & 0111) = 0100

This is still not zero, continue.

Fourth iteration:

bitMask is shifted left: 1000

(1000 & 0111) = 0000

This is zero, exit the loop.