

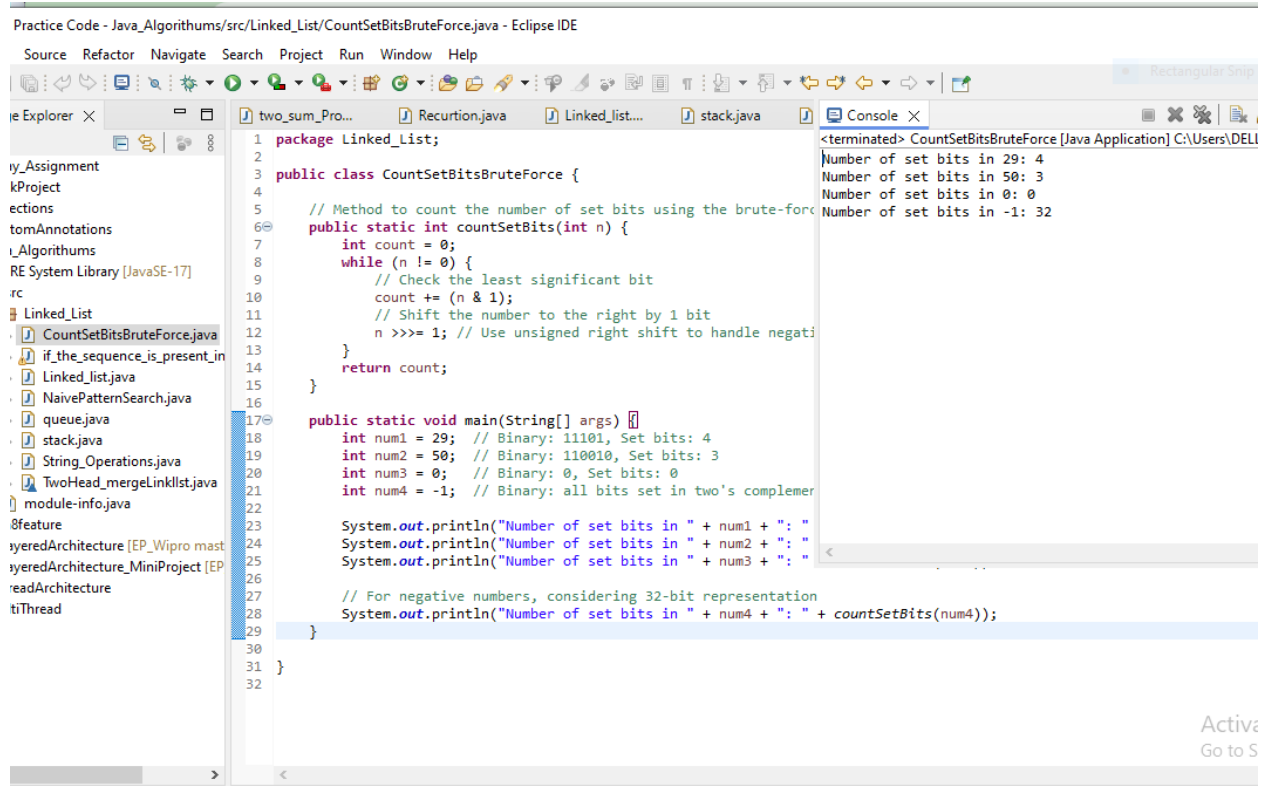
Day 12:

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.

- ☐ **Process:**
 - Initialize a counter count to 0.
 - Use a while loop to iterate as long as n is not zero.
 - In each iteration, check the least significant bit (LSB) of n using $n \& 1$.
 - If the LSB is 1, increment the count.
 - Right shift n by one position using $n \gg= 1$. The \gg operator performs an unsigned right shift, filling the leftmost bit with zero. This is particularly useful for handling negative numbers correctly.
- **Output:** Return the count which represents the number of set bits.

☐



```
Practice Code - Java_Algorithms/src/Linked_List/CountSetBitsBruteForce.java - Eclipse IDE
Source Refactor Navigate Search Project Run Window Help
two_sum_Pro... Recursion.java Linked_list... stack.java Console
1 package Linked_List;
2
3 public class CountSetBitsBruteForce {
4
5     // Method to count the number of set bits using the brute-force
6     public static int countSetBits(int n) {
7         int count = 0;
8         while (n != 0) {
9             // Check the least significant bit
10            count += (n & 1);
11            // Shift the number to the right by 1 bit
12            n >>= 1; // Use unsigned right shift to handle negative numbers
13        }
14        return count;
15    }
16
17    public static void main(String[] args) {
18        int num1 = 29; // Binary: 11101, Set bits: 4
19        int num2 = 50; // Binary: 110010, Set bits: 3
20        int num3 = 0; // Binary: 0, Set bits: 0
21        int num4 = -1; // Binary: all bits set in two's complement
22
23        System.out.println("Number of set bits in " + num1 + ": " + countSetBits(num1));
24        System.out.println("Number of set bits in " + num2 + ": " + countSetBits(num2));
25        System.out.println("Number of set bits in " + num3 + ": " + countSetBits(num3));
26
27        // For negative numbers, considering 32-bit representation
28        System.out.println("Number of set bits in " + num4 + ": " + countSetBits(num4));
29    }
30 }
31
32
<terminated> CountSetBitsBruteForce [Java Application] C:\Users\DELL
Number of set bits in 29: 4
Number of set bits in 50: 3
Number of set bits in 0: 0
Number of set bits in -1: 32
```

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.

To find the two non-repeating elements in an array where every other element appears twice, you can use the properties of the XOR operation. This approach takes advantage of the fact that XOR-ing a number with itself results in 0 and XOR-ing a number with 0 results in the number itself.

Here is the efficient algorithm using bitwise XOR operations:

1. **XOR all elements in the array:** This step will give you the XOR of the two unique numbers because the duplicate numbers will cancel each other out.
2. **Find a set bit in the result:** This bit is set in one of the unique numbers and not in the other.
3. **Divide the array into two groups:** One group where the set bit is present and another where it is not.
4. **XOR all elements within each group:** This will isolate the two unique numbers.

The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Displays a project structure with folders like 'ray_Assignment', 'nkProject', and 'Collections'. A file named 'FindTwoUniqueNumbers.java' is selected under the 'Collections' folder.
- Editor (Center):** Shows the source code of 'FindTwoUniqueNumbers.java'. The code implements a method 'findUniqueNumbers' that uses XOR to find two unique numbers in an array. The 'main' method tests the function with the array {1, 2, 3, 2, 1, 4, 5, 5}.
- Console (Right):** Displays the output of the program: '<terminated> FindTwoUniqueNumbers [Java Application]' and 'The two unique numbers are: 3 and 4'.

```
70 public static int[] findUniqueNumbers(int[] nums) {
71     int xorResult = 0;
72
73     for (int num : nums) {
74         xorResult ^= num;
75     }
76
77     int setBit = xorResult & ~(xorResult - 1);
78
79     int num1 = 0;
80     int num2 = 0;
81
82     for (int num : nums) {
83         if ((num & setBit) != 0) {
84             num1 ^= num;
85         } else {
86             num2 ^= num;
87         }
88     }
89
90     return new int[]{num1, num2};
91 }
92
93 public static void main(String[] args) {
94     int[] nums = {1, 2, 3, 2, 1, 4, 5, 5};
95     int[] result = findUniqueNumbers(nums);
96
97     System.out.println("The two unique numbers are: " + result[0] + " and " + result[1]);
98 }
99 }
```