Program for Sum of number and its Reverser

```java
package DsProgramTree;

public class SumOfNumberAndReverse {

        public static boolean sumOfNumberAndReverse(int num) {
            // Iterate from num/2 to num (inclusive) to find a pair of numbers
            for (int i = num / 2; i <= num; i++) {
                // Calculate the reverse of i using the reverse method
                int j = reverse(i);

                // Check if the sum of i and its reverse j equals num
                if (i + j == num) {
                    return true;
                }
            }
            return false;
        }

        // Helper method to reverse an integer
        static int reverse(int n) {
            int rev = 0;
            while (n != 0) {
                int rem = n % 10;
                rev = rev * 10 + rem;
                n /= 10;
            }
            return rev;
        }

        public static void main(String[] args) {

            int numToCheck = 443;
            boolean result = sumOfNumberAndReverse(numToCheck);

            if (result) {
                System.out.println(numToCheck + " the sum of a number is reverse. : 
"+ result);
            } else {
                System.out.println(numToCheck + " the sum of a number is reverse. : 
"+ result);
            }
        }
    }
```

OUTPUT

443 the sum of a number is reverse. : true

Program for Minimum Number of operations to Make All Array Elements Equal to 1

```
package DsProgramTree;

public class MinOperationsToMakeAllOnes {

        // Helper function to calculate the greatest common divisor (GCD)
        int getGcd(int value1, int value2) {
            if (value1 == 0) {
                return value2;
            }
            return getGcd(value2 % value1, value1);
        }

        // Main function to find the minimum operations
        public int minOperations(int[] arr) {
            int onesCount = 0; // Count of elements that are already 1
            int n = arr.length; // Length of the array
            int min = Integer.MAX_VALUE; // Initialize the minimum operations to a
large value

            // Count the number of elements that are already 1
            for (int i = 0; i < n; i++) {
                if (arr[i] == 1) {
                    onesCount++;
                }
            }

            // If there are elements that are already 1, return the count of non-1
elements
            if (onesCount != 0) {
                return n - onesCount;
            }

            // Iterate through the array to find the minimum operations
            for (int i = 0; i < n; i++) {
                int gcd = arr[i]; // Initialize GCD with the current element
                for (int j = i + 1; j < n; j++) {
                    gcd = getGcd(arr[j], gcd); // Calculate GCD with the next element
                    if (gcd == 1) {
                        min = Math.min(min, j - i); // Update the minimum operations
                        break; // Break if GCD is 1
                    }
                }
            }

            // If no GCD of 1 is found, return -1
            if (min == Integer.MAX_VALUE) {
                return -1;
            }

            // Return the total operations needed (original length + minimum
operations - 1)
```

```
            return n + min - 1;
        }

    public static void main(String[] args) {
        MinOperationsToMakeAllOnes opt =new MinOperationsToMakeAllOnes();
        int[] nums = {2, 6, 3, 4};

        // Calculate the minimum operations
        int result = opt.minOperations(nums);

        // Display the result
        System.out.println("Result  : " + result);
    }
    }
```

OUTPUT

Result : 4