ig - SYNTAX ERROR

Striver a2z DSA SHEET - "495"

By - SYNTAX ERROR

Step 1 : Learn the basics

The PDF created by *Abhishek Rathor (Instagram:SYNTAX ERROR)*

PROBLEM:1

Problem Statement:

Write a Java program to count the number of digits in a given integer.

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class DigitCounter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();

        int count = countDigits(number);
        System.out.println("Number of digits: " + count);
    }

    public static int countDigits(int num) {
        num = Math.abs(num); // Handle negative numbers
        return String.valueOf(num).length();
    }
}
```

Explanation:

1. The program takes an integer input from the user.
2. It converts the number to its absolute value (to handle negatives).
3. The `countDigits` method converts the number to a string and returns its length.
4. Finally, the program prints the count of digits.

PROBLEM:2

Problem Statement:

Write a Java program to reverse a given integer.

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class NumberReverser {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();

        int reversed = reverseNumber(number);
        System.out.println("Reversed number: " + reversed);
    }

    public static int reverseNumber(int num) {
        boolean isNegative = num < 0;
        num = Math.abs(num);
        int reversed = 0;

        while (num > 0) {
            reversed = reversed * 10 + num % 10;
            num /= 10;
        }

        return isNegative ? -reversed : reversed;
    }
}
```

Explanation:

1. The program takes an integer input from the user.
2. It determines if the number is negative and converts it to its absolute value.
3. The `reverseNumber` method iterates through the digits, constructing the reversed number.
4. If the input was negative, the result is made negative again.
5. Finally, the reversed number is printed.

PROBLEM:3

Problem Statement:
Write a Java program to check whether a given integer is a palindrome.

Java Code:

```
java
CopyEdit
import java.util.Scanner;
```

```
public class PalindromeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();

        boolean isPalindrome = checkPalindrome(number);
        System.out.println(isPalindrome ? "Palindrome" : "Not a
Palindrome");
    }

    public static boolean checkPalindrome(int num) {
        if (num < 0) return false; // Negative numbers are not palindromes

        int original = num, reversed = 0;
        while (num > 0) {
            reversed = reversed * 10 + num % 10;
            num /= 10;
        }

        return original == reversed;
    }
}
```

Explanation:

1. The program takes an integer input from the user.
2. If the number is negative, it immediately returns `false` (negative numbers aren't palindromes).
3. The `checkPalindrome` method reverses the number and compares it with the original.
4. If both are equal, the number is a palindrome; otherwise, it is not.
5. Finally, the program prints whether the number is a palindrome or not.

PROBLEM:3

Problem Statement:
Write a Java program to find the Greatest Common Divisor (GCD) or Highest Common Factor (HCF) of two given numbers.

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class GCDCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter first number: ");
        int num1 = scanner.nextInt();
        System.out.print("Enter second number: ");
        int num2 = scanner.nextInt();
        scanner.close();

        int gcd = findGCD(num1, num2);
        System.out.println("GCD (HCF): " + gcd);
    }

    public static int findGCD(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return Math.abs(a);
    }
}
```

Explanation:

1. The program takes two integer inputs from the user.
2. The `findGCD` method uses the **Euclidean algorithm** to compute the GCD.
   - It repeatedly replaces `a` with `b` and `b` with `a % b` until `b` becomes `0`.
   - The final value of `a` is the GCD.
3. The program prints the computed GCD (HCF).

PROBLEM : 4

Problem Statement:
Write a Java program to check whether a given integer is an **Armstrong number**.
An **Armstrong number** (or **Narcissistic number**) for an `n`-digit number is a number where the sum of its digits raised to the power `n` is equal to the number itself.

Example:

- **153** → 13+53+33=1531^3 + 5^3 + 3^3 = 15313+53+33=153 ☑ (Armstrong Number)
- **9474** → 94+44+74+44=94749^4 + 4^4 + 7^4 + 4^4 = 947494+44+74+44=9474 ☑ (Armstrong Number)
- **123** → 13+23+33=361^3 + 2^3 + 3^3 = 3613+23+33=36 ❌ (Not an Armstrong Number)

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class ArmstrongNumberChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();
        boolean isArmstrong = checkArmstrong(number);
        System.out.println(isArmstrong ? "Armstrong Number" : "Not an
Armstrong Number");
    }

    public static boolean checkArmstrong(int num) {
        int original = num, sum = 0, digits = String.valueOf(num).length();

        while (num > 0) {
            int digit = num % 10;
            sum += Math.pow(digit, digits);
            num /= 10;
        }

        return sum == original;
    }
}
```

Explanation:

1. The program takes an integer input from the user.
2. It calculates the **number of digits** in the given number.
3. The `checkArmstrong` method extracts each digit, raises it to the power of the number of digits, and adds the result to `sum`.
4. If the final `sum` equals the original number, it is an **Armstrong number**; otherwise, it is not.
5. The program prints whether the number is an **Armstrong number** or **not**.

PROBLEM: 5

Problem Statement:
Write a Java program to print all divisors of a given integer.

A **divisor** of a number `n` is an integer that divides `n` completely without leaving a remainder.

Example:

- **Input:** `n = 12`
- **Output:** `1 2 3 4 6 12`
- **Input:** `n = 17`
- **Output:** `1 17` (Since 17 is a prime number)

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class DivisorPrinter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();

        System.out.print("Divisors: ");
        printDivisors(number);
    }

    public static void printDivisors(int num) {
        for (int i = 1; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                System.out.print(i + " ");
                if (i != num / i) {
                    System.out.print((num / i) + " ");
                }
            }
        }
        System.out.println();
    }
}
```

Explanation:

1. The program takes an integer input from the user.
2. The `printDivisors` method iterates from `1` to `√n` to find all divisors efficiently.
3. If `i` is a divisor of `n`, then `n / i` is also a divisor.
4. The divisors are printed in pairs to optimize performance.
5. The program finally prints all divisors in ascending order.

PROBLEM: 6

Problem Statement:
Write a Java program to check whether a given integer is a **prime number**.

A **prime number** is a number greater than `1` that has exactly **two divisors**: `1` and itself.

Example:

- **Input:** `n = 7`
- **Output:** `Prime Number` ☑
- **Input:** `n = 10`
- **Output:** `Not a Prime Number` ✖

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class PrimeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();

        boolean isPrime = checkPrime(number);
        System.out.println(isPrime ? "Prime Number" : "Not a Prime Number");
    }

    public static boolean checkPrime(int num) {
        if (num < 2) return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Explanation:

1. The program takes an integer input from the user.
2. If the number is **less than 2**, it is **not a prime**.
3. The `checkPrime` method iterates from `2` to `√n` to check if `n` is divisible by any number.
4. If a divisor is found, the function returns **false** (not prime).
5. Otherwise, the function returns **true** (prime).
6. The program prints whether the number is **prime or not**.

PROBLEM: 7

Problem Statement:
Write a Java program to print a message **N times** using recursion.

Example:

- **Input:** `N = 5`, Message: `"Hello"`
- **Output:**

  ```
  nginx
  CopyEdit
  Hello
  Hello
  Hello
  Hello
  Hello
  ```

Java Code:

```
java
CopyEdit
import java.util.Scanner;
```

```
public class RecursionPrinter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of times to print: ");
        int n = scanner.nextInt();
        scanner.close();

        printMessage(n);
    }

    public static void printMessage(int n) {
        if (n <= 0) return; // Base case: Stop when N reaches 0
        System.out.println("Hello");
        printMessage(n - 1); // Recursive call
    }
}
```

Explanation:

1. The program takes an integer input `N` from the user.
2. The `printMessage` method prints `"Hello"` and then calls itself with `N - 1`.
3. The **base case** (`if n <= 0 return;`) stops recursion when `N` reaches `0`.
4. The recursion continues until `N` becomes `0`, printing the message `N` times.

PROBLEM: 8

Problem Statement:
Write a Java program to print a given **name** `N` times using recursion.

Example:

- **Input:** `N = 3`, Name: `"Alice"`
- **Output:**

  ```
  nginx
  CopyEdit
  Alice
  Alice
  Alice
  ```

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class NamePrinter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();
        System.out.print("Enter the number of times to print: ");
```

```
        int n = scanner.nextInt();
        scanner.close();

        printName(n, name);
    }

    public static void printName(int n, String name) {
        if (n <= 0) return; // Base case: Stop when N reaches 0
        System.out.println(name);
        printName(n - 1, name); // Recursive call
    }
}
```

Explanation:

1. The program takes a **name** and an integer `N` as input.
2. The `printName` method prints the name and calls itself with `N - 1`.
3. The **base case** (`if n <= 0 return;`) stops recursion when `N` becomes `0`.
4. The recursion continues until `N` times, printing the name on each call.

PROBLEM: 9

Problem Statement:
Write a Java program to print numbers from **1 to N** using recursion.

Example:

- **Input:** `N = 5`
- **Output:**

  ```
  CopyEdit
  1
  2
  3
  4
  5
  ```

Java Code:

```
import java.util.Scanner;

public class PrintNumbers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number N: ");
        int n = scanner.nextInt();
        scanner.close();

        printNumbers(1, n);
    }

    public static void printNumbers(int start, int n) {
        if (start > n) return; // Base case: Stop when start exceeds N
```

```
        System.out.println(start);
        printNumbers(start + 1, n); // Recursive call with incremented
value
    }
}
```

Explanation:

1. The program takes an integer `N` as input.
2. The `printNumbers` method starts from `1` and recursively prints numbers up to `N`.
3. The **base case** (`if start > N return;`) stops recursion when `start` exceeds `N`.
4. The recursion continues, printing numbers in increasing order.

PROBLEM: 10

Problem Statement:
Write a Java program to print numbers from **N to 1** using recursion.

Example:

- **Input:** `N = 5`
- **Output:**

```
5
4
3
2
1
```

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class PrintReverse {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number N: ");
        int n = scanner.nextInt();
        scanner.close();

        printNumbers(n);
    }

    public static void printNumbers(int n) {
        if (n <= 0) return; // Base case: Stop when N reaches 0
        System.out.println(n);
        printNumbers(n - 1); // Recursive call with decremented value
    }
}
```

Explanation:

1. The program takes an integer `N` as input.
2. The `printNumbers` method prints `N` and recursively calls itself with `N - 1`.
3. The **base case** (`if n <= 0 return;`) stops recursion when `N` reaches `0`.
4. The recursion continues, printing numbers in **decreasing order** from `N` to `1`.

PROBLEM: 11

Problem Statement:
Write a Java program to find the **sum of the first N natural numbers** using recursion.

Example:

- **Input:** `N = 5`
- **Output:** `15`
  (Explanation: `1 + 2 + 3 + 4 + 5 = 15`)

Java Code:

```
import java.util.Scanner;

public class SumOfNumbers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number N: ");
        int n = scanner.nextInt();
        scanner.close();

        int sum = sumOfN(n);
        System.out.println("Sum of first " + n + " numbers: " + sum);
    }

    public static int sumOfN(int n) {
        if (n == 0) return 0; // Base case: sum of 0 is 0
        return n + sumOfN(n - 1); // Recursive call
    }
}
```

Explanation:

1. The program takes an integer `N` as input.
2. The `sumOfN` method recursively computes the sum:
   - **Base case:** If `N == 0`, return `0`.
   - Otherwise, return `N + sumOfN(N-1)`.
3. The recursion continues until `N` reaches `0`, summing all numbers from `N` to `1`.
4. Finally, the sum is printed.

PROBLEM: 12

Problem Statement:
Write a Java program to find the **factorial of N** using recursion.

Factorial Formula:

N!=N×(N−1)×(N−2)×...×1N! = N \times (N-1) \times (N-2) \times ... \times 1N!=N×(N−1)×(N−2)×...×1

- **Example:**
  - **Input:** `N = 5`
  - **Output:** `120`
    (Explanation: `5 × 4 × 3 × 2 × 1 = 120`)

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number N: ");
        int n = scanner.nextInt();
        scanner.close();

        long fact = factorial(n);
        System.out.println("Factorial of " + n + " is: " + fact);
    }

    public static long factorial(int n) {
        if (n == 0 || n == 1) return 1; // Base case: 0! = 1 and 1! = 1
        return n * factorial(n - 1); // Recursive call
    }
}
```

Explanation:

1. The program takes an integer `N` as input.
2. The `factorial` method recursively computes the factorial:
   - **Base case:** If `N == 0` or `N == 1`, return `1`.
   - Otherwise, return `N × factorial(N-1)`.
3. The recursion continues until `N` reaches `1`, multiplying all numbers from `N` down to `1`.
4. Finally, the factorial value is printed.

PROBLEM: 12

Problem Statement:
Write a Java program to reverse an array using recursion.

Example:

- **Input:** `[1, 2, 3, 4, 5]`
- **Output:** `[5, 4, 3, 2, 1]`

Java Code:

```
import java.util.Scanner;

public class ReverseArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        reverseArray(arr, 0, n - 1);

        System.out.println("Reversed array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void reverseArray(int[] arr, int left, int right) {
        if (left >= right) return; // Base case: Stop when left crosses
right
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp; // Swap elements

        reverseArray(arr, left + 1, right - 1); // Recursive call
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into an array.
3. The `reverseArray` method swaps elements from both ends using recursion:
   - **Base case:** If `left >= right`, stop recursion.
   - Swap `arr[left]` and `arr[right]`.
   - Recursively call `reverseArray` with `left + 1` and `right - 1`.
4. The reversed array is printed after recursion completes.

PROBLEM: 13

Problem Statement:
Write a Java program to check if a given string is a **palindrome** using recursion.

Example:

- **Input:** `"madam"`
- **Output:** `Yes, it is a palindrome`
- **Input:** `"hello"`
- **Output:** `No, it is not a palindrome`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class PalindromeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        scanner.close();

        boolean isPalindrome = checkPalindrome(str, 0, str.length() - 1);
        if (isPalindrome) {
            System.out.println("Yes, it is a palindrome");
        } else {
            System.out.println("No, it is not a palindrome");
        }
    }

    public static boolean checkPalindrome(String str, int left, int right)
{
        if (left >= right) return true; // Base case: If pointers cross,
it's a palindrome
        if (str.charAt(left) != str.charAt(right)) return false; //
Mismatch found
        return checkPalindrome(str, left + 1, right - 1); // Recursive call
    }
}
```

Explanation:

1. The program takes a **string** as input.
2. The `checkPalindrome` function checks if the string is a palindrome using recursion:
   - **Base case:** If `left >= right`, return `true`.
   - If characters at `left` and `right` do not match, return `false`.
   - Otherwise, recursively check the next pair of characters.
3. The program prints `"Yes, it is a palindrome"` if the string is a palindrome, otherwise `"No, it is not a palindrome"`.

PROBLEM: 14

Problem Statement:
Write a Java program to find the **Nth Fibonacci number** using recursion.

Fibonacci Sequence:

F(n)=F(n−1)+F(n−2)F(n) = F(n-1) + F(n-2)F(n)=F(n−1)+F(n−2)

where

- F(0)=0F(0) = 0F(0)=0
- F(1)=1F(1) = 1F(1)=1
- F(2)=1F(2) = 1F(2)=1
- F(3)=2F(3) = 2F(3)=2
- F(4)=3F(4) = 3F(4)=3
- F(5)=5F(5) = 5F(5)=5
- and so on...

Example:

- **Input:** `N = 6`
- **Output:** `8`
  (Explanation: `0, 1, 1, 2, 3, 5, 8 → 6th Fibonacci number is 8`)

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number N: ");
        int n = scanner.nextInt();
        scanner.close();

        int result = fibonacci(n);
        System.out.println("Fibonacci number at position " + n + " is: " +
result);
    }

    public static int fibonacci(int n) {
        if (n == 0) return 0; // Base case: F(0) = 0
        if (n == 1) return 1; // Base case: F(1) = 1
        return fibonacci(n - 1) + fibonacci(n - 2); // Recursive call
    }
}
```

Explanation:

1. The program takes an integer **N** as input.
2. The `fibonacci` method recursively computes the **Nth Fibonacci number**:
   - **Base case:** If `N == 0`, return `0`.
   - If `N == 1`, return `1`.
   - Otherwise, return `fibonacci(N-1) + fibonacci(N-2)`.
3. The recursion continues until reaching the base cases.
4. The final result is printed as the **Nth Fibonacci number**.

PROBLEM: 15

Problem Statement:
Write a Java program to count the frequency of each element in an array.

Example:

- **Input:** `[1, 2, 2, 3, 1, 4, 2, 3, 4, 4]`
- **Output:**

```
bash
CopyEdit
1 → 2 times
2 → 3 times
3 → 2 times
4 → 3 times
```

Java Code:

```
java
CopyEdit
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class FrequencyCounter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        countFrequencies(arr);
    }

    public static void countFrequencies(int[] arr) {
        Map<Integer, Integer> freqMap = new HashMap<>();

        for (int num : arr) {
            freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
        }
        System.out.println("Element Frequencies:");
        for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {
            System.out.println(entry.getKey() + " → " + entry.getValue() +
" times");
        }
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into an array.
3. The `countFrequencies` method uses a **HashMap** to store the frequency of each element:
    - If an element appears for the first time, it is added to the map with a count of `1`.
    - If it already exists, the count is incremented.
4. Finally, the program prints the frequency of each unique element in the array.

PROBLEM:16

Problem Statement:
Write a Java program to count the frequency of each element in an array.

Example:

- **Input:** `[1, 2, 2, 3, 1, 4, 2, 3, 4, 4]`
- **Output:**

  ```
  bash
  CopyEdit
  1 → 2 times
  2 → 3 times
  3 → 2 times
  4 → 3 times
  ```

Java Code:

```
java
CopyEdit
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class FrequencyCounter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
```

```
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        countFrequencies(arr);
    }

    public static void countFrequencies(int[] arr) {
        Map<Integer, Integer> freqMap = new HashMap<>();

        for (int num : arr) {
            freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
        }

        System.out.println("Element Frequencies:");
        for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {
            System.out.println(entry.getKey() + " → " + entry.getValue() +
" times");
        }
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into an array.
3. The `countFrequencies` method uses a **HashMap** to store the frequency of each element:
   - If an element appears for the first time, it is added to the map with a count of `1`.
   - If it already exists, the count is incremented.
4. Finally, the program prints the frequency of each unique element in the array.

PROBLEM:17

Problem Statement:
Write a Java program to find the element with the **highest** and **lowest** frequency in an array.

Example:

- **Input:** `[1, 2, 2, 3, 1, 4, 2, 3, 4, 4]`
- **Output:**

```
bash
CopyEdit
Highest Frequency Element: 2 (3 times)
Lowest Frequency Element: 1 (2 times)
```

Java Code:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
```

```
public class FrequencyAnalyzer {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        findHighestLowestFrequency(arr);
    }

    public static void findHighestLowestFrequency(int[] arr) {
        Map<Integer, Integer> freqMap = new HashMap<>();

        for (int num : arr) {
            freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
        }

        int maxFreq = Integer.MIN_VALUE, minFreq = Integer.MAX_VALUE;
        int maxElement = -1, minElement = -1;

        for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {
            int key = entry.getKey();
            int freq = entry.getValue();

            if (freq > maxFreq) {
                maxFreq = freq;
                maxElement = key;
            }
            if (freq < minFreq) {
                minFreq = freq;
                minElement = key;
            }
        }

        System.out.println("Highest Frequency Element: " + maxElement + "
(" + maxFreq + " times)");
        System.out.println("Lowest Frequency Element: " + minElement + " ("
+ minFreq + " times)");
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into an array.
3. The `findHighestLowestFrequency` method performs the following:
    - Uses a **HashMap** to store the frequency of each element.
    - Iterates through the map to find the **highest** and **lowest** frequency elements.
    - Prints the elements along with their frequency count.

Step 2: SORTING

PROBLEM:17

Problem Statement:
Write a Java program to implement **Selection Sort**. Selection Sort works by repeatedly finding the minimum element from the unsorted part of the array and placing it at the beginning.

Example:

- **Input:** `[64, 25, 12, 22, 11]`
- **Output:** `[11, 12, 22, 25, 64]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class SelectionSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        selectionSort(arr);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void selectionSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            // Swap the found minimum element with the first element
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `selectionSort` method sorts the array using the Selection Sort algorithm:
    - It iterates over the array, and for each position, it finds the minimum element from the unsorted part.
    - Then, it swaps the minimum element with the element at the current position.
4. After sorting, the program prints the sorted array.

PROBLEM:17

Problem Statement:
Write a Java program to implement **Bubble Sort**. Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The process is repeated until the array is sorted.

Example:

- **Input:** `[64, 25, 12, 22, 11]`
- **Output:** `[11, 12, 22, 25, 64]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class BubbleSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        bubbleSort(arr);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            // Flag to check if any swap was made in this iteration
            boolean swapped = false;

            // Last i elements are already in place
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Swap arr[j] and arr[j+1]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }

            // If no two elements were swapped by inner loop, the array is
sorted
            if (!swapped) {
                break;
            }
        }
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `bubbleSort` method sorts the array using the Bubble Sort algorithm:
   - It iterates over the array and compares adjacent elements. If the elements are out of order, they are swapped.
   - This process is repeated for each element in the array, and with each iteration, the largest element "bubbles up" to its correct position.
   - The `swapped` flag helps to optimize the algorithm by stopping early if no swaps were made in a full pass, indicating the array is already sorted.
4. After sorting, the program prints the sorted array.

PROBLEM:17

Problem Statement:
Write a Java program to implement **Insertion Sort**. Insertion Sort works by picking elements one by one and placing them in their correct position in the sorted part of the array.

Example:

- **Input:** `[64, 25, 12, 22, 11]`
- **Output:** `[11, 12, 22, 25, 64]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class InsertionSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        insertionSort(arr);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;

            // Shift elements of arr[0..i-1] that are greater than key to
one position ahead
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `insertionSort` method sorts the array using the Insertion Sort algorithm:
   - It starts from the second element (index `1`), treating the first element as already sorted.
   - For each element, it finds the correct position in the sorted part by shifting larger elements to the right.
   - It inserts the current element (`key`) into its correct position.
4. After sorting, the program prints the sorted array.

PROBLEM:18

Problem Statement:
Write a Java program to implement **Merge Sort**. Merge Sort is a divide-and-conquer algorithm that recursively divides the array into two halves, sorts them, and then merges them back together.

Example:

- **Input:** `[64, 25, 12, 22, 11]`
- **Output:** `[11, 12, 22, 25, 64]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class MergeSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        mergeSort(arr, 0, n - 1);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    public static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];
        // Copy data to temp arrays
        for (int i = 0; i < n1; i++) {
            leftArray[i] = arr[left + i];
        }
        for (int i = 0; i < n2; i++) {
            rightArray[i] = arr[mid + 1 + i];
        }

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                arr[k] = leftArray[i];
                i++;
            } else {
                arr[k] = rightArray[j];
                j++;
            }
            k++;
        }

        // Copy remaining elements of leftArray[]
        while (i < n1) {
            arr[k] = leftArray[i];
            i++;
            k++;
        }

        // Copy remaining elements of rightArray[]
        while (j < n2) {
            arr[k] = rightArray[j];
            j++;
            k++;
        }
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `mergeSort` function:
   - Recursively divides the array into two halves.
   - Calls itself for each half.
   - Merges the two halves using the `merge` function.
4. The `merge` function:
   - Creates two temporary subarrays.
   - Merges them in sorted order back into the original array.
5. Finally, the sorted array is printed.

PROBLEM:19

Problem Statement:

Write a Java program to implement **Recursive Bubble Sort**. Instead of using iterative loops, the sorting is done recursively by repeatedly moving the largest element to the end in each pass.

Example:

- **Input:** `[64, 25, 12, 22, 11]`
- **Output:** `[11, 12, 22, 25, 64]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class RecursiveBubbleSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        bubbleSort(arr, n);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void bubbleSort(int[] arr, int n) {
        if (n == 1) {
            return; // Base case: If only one element is left, array is
sorted
        }

        // Perform one pass of bubble sort, moving the largest element to
the end
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] > arr[i + 1]) {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }

        // Recur for the remaining unsorted part
        bubbleSort(arr, n - 1);
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `bubbleSort` method:
   - Moves the largest element to the end of the array in one pass.
   - Recursively calls itself with a reduced problem size (`n - 1`).
   - Stops when only one element is left (base case).
4. After sorting, the program prints the sorted array.

PROBLEM:20

Problem Statement:

Write a Java program to implement **Recursive Insertion Sort**. Instead of using loops, the sorting is done recursively by placing each element in its correct position in the sorted part of the array.

Example:

- **Input:** `[64, 25, 12, 22, 11]`
- **Output:** `[11, 12, 22, 25, 64]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class RecursiveInsertionSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        insertionSort(arr, n);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void insertionSort(int[] arr, int n) {
        if (n <= 1) {
            return; // Base case: A single-element array is already sorted
        }

        // Sort the first (n-1) elements
        insertionSort(arr, n - 1);

        // Insert the nth element into its correct position
        int key = arr[n - 1];
        int j = n - 2;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `insertionSort` method:
    - Recursively sorts the first `N-1` elements.
    - Places the `Nth` element in its correct position by shifting larger elements.
    - Stops when `N` becomes `1` (base case).
4. After sorting, the program prints the sorted array.

PROBLEM:21

Problem Statement:

Write a Java program to implement **Quick Sort** using the **divide and conquer** approach. The algorithm selects a **pivot**, partitions the array into elements smaller and greater than the pivot, and recursively sorts the partitions.

Example:

- **Input:** `[64, 25, 12, 22, 11]`
- **Output:** `[11, 12, 22, 25, 64]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;
```

```java
public class QuickSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        quickSort(arr, 0, n - 1);

        System.out.println("Sorted array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);
            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high]; // Choosing last element as pivot
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // Swap pivot to its correct position
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1; // Return pivot index
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `quickSort` function:
   - Selects a **pivot** (last element).

   - Calls `partition()` to place the pivot in its correct position.
   - Recursively sorts elements before and after the pivot.
4. The `partition` function:
   - Moves smaller elements to the left of the pivot.
   - Moves larger elements to the right.
   - Returns the pivot index.
5. Finally, the sorted array is printed.

Step 3: Solve Problems on Arrays

PROBLEM:22

Write a Java program to find the **largest element** in a given array.

Example:

- **Input:** `[10, 20, 4, 45, 99, 6]`
- **Output:** `Largest element: 99`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class LargestElement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        int largest = findLargest(arr, n);
        System.out.println("Largest element: " + largest);
    }

    public static int findLargest(int[] arr, int n) {
        int max = arr[0]; // Assume first element is the largest

        for (int i = 1; i < n; i++) {
            if (arr[i] > max) {
                max = arr[i]; // Update max if a larger element is found
            }
        }

        return max;
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `findLargest` method:
    - Assumes the first element is the largest.
    - Iterates through the array, updating `max` when a larger element is found.
4. Finally, the largest element is printed.

PROBLEM:23

Problem Statement:

Write a Java program to find the **second largest element** in a given array **without sorting**.

Example:

- **Input:** `[10, 20, 4, 45, 99, 6]`
- **Output:** `Second largest element: 45`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class SecondLargestElement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        int secondLargest = findSecondLargest(arr, n);
        if (secondLargest == Integer.MIN_VALUE) {
            System.out.println("No second largest element found.");
        } else {
            System.out.println("Second largest element: " + secondLargest);
        }
    }

    public static int findSecondLargest(int[] arr, int n) {
        if (n < 2) {
            return Integer.MIN_VALUE; // If there are fewer than 2 elements,
no second largest
        }

        int largest = Integer.MIN_VALUE, secondLargest = Integer.MIN_VALUE;

        for (int i = 0; i < n; i++) {
            if (arr[i] > largest) {
                secondLargest = largest;
                largest = arr[i];
            } else if (arr[i] > secondLargest && arr[i] != largest) {
                secondLargest = arr[i];
            }
        }

        return secondLargest;
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `findSecondLargest` method:
   - Initializes `largest` and `secondLargest` to `Integer.MIN_VALUE`.
   - Iterates through the array to find the largest and second largest elements.
   - If a new largest number is found, the previous largest becomes second largest.
   - If an element is smaller than the largest but greater than the second largest, it updates `secondLargest`.
4. If no second largest element is found, it returns `Integer.MIN_VALUE`.
5. Finally, the second largest element is printed.

PROBLEM:24

Problem Statement:

Write a Java program to check if a given array is **sorted in non-decreasing order**.

Example 1:

- **Input:** `[1, 2, 3, 4, 5]`
- **Output:** `Array is sorted`

Example 2:

- **Input:** `[1, 3, 2, 4, 5]`
- **Output:** `Array is not sorted`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class CheckSorted {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        if (isSorted(arr, n)) {
            System.out.println("Array is sorted");
        } else {
            System.out.println("Array is not sorted");
        }
    }

    public static boolean isSorted(int[] arr, int n) {
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] > arr[i + 1]) {
                return false; // Found an element out of order
            }
        }
        return true; // Array is sorted
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `isSorted` method:
   - Iterates through the array.
   - If it finds an element that is greater than the next element, the array is **not sorted**.
   - If the loop completes without finding such a case, the array is **sorted**.
4. Finally, the result is printed.

PROBLEM:25

Problem Statement:

Write a Java program to **remove duplicates** from a **sorted array** and return the new length of the array with unique elements.

Example:

- **Input:** `[1, 1, 2, 2, 3, 4, 4, 5]`
- **Output:** `Unique elements: [1, 2, 3, 4, 5]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class RemoveDuplicates {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " sorted elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        int newLength = removeDuplicates(arr, n);
        System.out.print("Unique elements: ");
        for (int i = 0; i < newLength; i++) {
            System.out.print(arr[i] + " ");
        }
    }

    public static int removeDuplicates(int[] arr, int n) {
        if (n == 0 || n == 1) {
            return n; // If array has 0 or 1 element, return its length
        }

        int j = 0; // Pointer for the new array without duplicates
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] != arr[i + 1]) {
                arr[j++] = arr[i]; // Copy unique element
            }
        }
        arr[j++] = arr[n - 1]; // Copy the last element

        return j; // Return new length of array
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` **sorted** elements into the array.
3. The `removeDuplicates` method:
    - Uses a **two-pointer approach** to overwrite duplicate elements.

    - If `arr[i]` is **different** from `arr[i+1]`, it is stored in the array.
    - The last element is always stored.
    - Returns the **new length** of the array with unique elements.
4. The unique elements are printed.

PROBLEM:26

Problem Statement:

Write a Java program to **left rotate** an array by **one place**.

Example:

- **Input:** `[1, 2, 3, 4, 5]`
- **Output:** `[2, 3, 4, 5, 1]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class LeftRotateByOne {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        leftRotateByOne(arr, n);

        System.out.print("Array after left rotation: ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void leftRotateByOne(int[] arr, int n) {
        if (n <= 1) return; // No need to rotate if there's only 1 element

        int first = arr[0]; // Store the first element
        for (int i = 0; i < n - 1; i++) {
            arr[i] = arr[i + 1]; // Shift elements left
        }
        arr[n - 1] = first; // Move the first element to the end
    }
}
```

Explanation:

1. The program takes an integer `N` as input (size of the array).
2. It reads `N` elements into the array.
3. The `leftRotateByOne` method:
   - Stores the **first element**.
   - **Shifts all elements left** by one position.
   - Places the **first element at the last index**.
4. The rotated array is printed.

PROBLEM: 27

Problem Statement:

Write a Java program to **left rotate** an array by **D places**.

Example:

- **Input:**
  - Array: `[1, 2, 3, 4, 5]`
  - D = `2`
- **Output:**
  - `[3, 4, 5, 1, 2]`

Java Code:

```
java
CopyEdit
import java.util.Scanner;

public class LeftRotateByD {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.print("Enter the number of rotations (D): ");
        int d = scanner.nextInt();
        scanner.close();

        leftRotateByD(arr, n, d);
        System.out.print("Array after " + d + " left rotations: ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void leftRotateByD(int[] arr, int n, int d) {
        // Handle if D is greater than the array length
        d = d % n;

        if (d == 0) {
            return; // No need to rotate if D is 0 or a multiple of n
        }

        // Reverse the first part of the array (from index 0 to d-1)
        reverse(arr, 0, d - 1);

        // Reverse the second part of the array (from index d to n-1)
        reverse(arr, d, n - 1);

        // Reverse the whole array
        reverse(arr, 0, n - 1);
    }

    // Helper method to reverse a portion of the array
    public static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            // Swap the elements at start and end
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;

            // Move the pointers
            start++;
            end--;
        }
    }
}
```

Explanation:

1. **Input:**
   - The program takes the size of the array `N` and the array elements.
   - It also takes the number of rotations `D` from the user.
2. **Rotation Logic:**
   - The `leftRotateByD` method uses the **reverse method** to efficiently rotate the array:
     - First, reverse the portion of the array from `0` to `D-1`.
     - Then reverse the portion from `D` to `N-1`.
     - Finally, reverse the entire array to achieve the left rotation by `D` places.
3. **Reverse Method:**
   - The `reverse` method swaps elements in the array from the `start` index to the `end` index, effectively reversing the portion of the array.
4. **Edge Case Handling:**

    - If `D` is greater than or equal to `N`, the program uses `d = d % n` to ensure the rotations don't exceed the array length.
    - If `D` is `0` or a multiple of `N`, no rotations are needed.
5. **Output:**
    - The program prints the array after performing the left rotation by `D` places.

PROBLEM: 28

Problem Statement:
Write a Java program to move all zeros in an array to the end without changing the order of non-zero elements.

Example:

- **Input:** `[0, 1, 2, 0, 3, 0, 4]`
- **Output:** `[1, 2, 3, 4, 0, 0, 0]`

```
public class MoveZerosToEnd {
    public static void main(String[] args) {
        int[] arr = {0, 1, 2, 0, 3, 0, 4};
        moveZeros(arr);
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void moveZeros(int[] arr) {
        int nonZeroIndex = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] != 0) {
                arr[nonZeroIndex++] = arr[i];
            }
        }
        while (nonZeroIndex < arr.length) {
            arr[nonZeroIndex++] = 0;
        }
    }
}
```

PROBLEM: 29

Problem Statement:
Write a Java program to implement a linear search algorithm to find an element in an array.

Example:

- **Input:** `[3, 5, 7, 9]`, target = `7`
- **Output:** `Element found at index 2`

```
public class LinearSearch {
    public static void main(String[] args) {
        int[] arr = {3, 5, 7, 9};
        int target = 7;
        int index = linearSearch(arr, target);
        if (index != -1) {
            System.out.println("Element found at index " + index);
        } else {
            System.out.println("Element not found");
        }
    }

    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }
}
```

PROBLEM: 30

Problem Statement:
Write a Java program to find the union of two arrays.

Example:

- **Input:** `arr1 = [1, 2, 3],arr2 = [2, 3, 4]`
- **Output:** `Union = [1, 2, 3, 4]`

```
import java.util.HashSet;

public class UnionOfArrays {
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3};
        int[] arr2 = {2, 3, 4};
        int[] union = findUnion(arr1, arr2);
        for (int num : union) {
            System.out.print(num + " ");
        }
    }

    public static int[] findUnion(int[] arr1, int[] arr2) {
        HashSet<Integer> unionSet = new HashSet<>();
        for (int num : arr1) {
            unionSet.add(num);
        }
        for (int num : arr2) {
            unionSet.add(num);
        }
        return unionSet.stream().mapToInt(i -> i).toArray();
    }
}
```

PROBLEM: 31

Problem Statement:
Write a Java program to find the missing number in a given array of integers from 1 to N.

Example:

- **Input:** `[1, 2, 4, 5]`
- **Output:** `3`

```
public class MissingNumber {
    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5};
        int missing = findMissingNumber(arr, 5);
        System.out.println("The missing number is " + missing);
    }

    public static int findMissingNumber(int[] arr, int n) {
        int sum = n * (n + 1) / 2;  // Sum of numbers from 1 to n
        int arrSum = 0;
        for (int num : arr) {
            arrSum += num;
        }
        return sum - arrSum;
    }
}
```

PROBLEM: 32

Problem Statement:
Write a Java program to find the **maximum number of consecutive 1's** in a binary array.

Example:

- **Input:** `[1, 1, 0, 1, 1, 1]`
- **Output:** `3`

```
public class MaxConsecutiveOnes {
    public static void main(String[] args) {
        int[] arr = {1, 1, 0, 1, 1, 1};
        int result = findMaxConsecutiveOnes(arr);
        System.out.println("Maximum consecutive 1's: " + result);
    }

    public static int findMaxConsecutiveOnes(int[] arr) {
        int maxCount = 0, currentCount = 0;
        for (int num : arr) {
            if (num == 1) {
                currentCount++;
                maxCount = Math.max(maxCount, currentCount);
            } else {
                currentCount = 0;
            }
        }
        return maxCount;
    }
}
```

PROBLEM: 33

Problem Statement:
Write a Java program to find the element that appears only once in an array, where all other elements appear twice.

Example:

- **Input:** `[4, 3, 2, 4, 3, 1, 2]`
- **Output:** `1`

```
public class ElementAppearsOnce {
    public static void main(String[] args) {
        int[] arr = {4, 3, 2, 4, 3, 1, 2};
        int result = findElement(arr);
        System.out.println("The element that appears once is " + result);
    }

    public static int findElement(int[] arr) {
        int result = 0;
        for (int num : arr) {
            result ^= num; // XOR all elements
        }
        return result;
    }
}
```

PROBLEM: 34

Problem Statement:
Write a Java program to find the **length of the longest subarray with a sum equal to K**, where all elements are positive integers.

Example:

- **Input:** `[1, 2, 3, 4, 1]`, K = `5`
- **Output:** `2` (subarray `[2, 3]` or `[1, 4]`)

```
import java.util.HashMap;

public class LongestSubarrayWithSumK {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 1};
        int k = 5;
        int result = findLongestSubarray(arr, k);
        System.out.println("Longest subarray with sum " + k + " has length: 
" + result);
    }

    public static int findLongestSubarray(int[] arr, int k) {
        HashMap<Integer, Integer> map = new HashMap<>();
        int sum = 0, maxLength = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
            if (sum == k) {
                maxLength = i + 1;
            }
            if (map.containsKey(sum - k)) {
                maxLength = Math.max(maxLength, i - map.get(sum - k));
            }
            map.putIfAbsent(sum, i);
        }
        return maxLength;
    }
}
```

PROBLEM: 35

Problem Statement:
Write a Java program to find the **length of the longest subarray with sum equal to K**, where the array contains both positive and negative integers.

Example:

- **Input:** `[1, -1, 5, -2, 3]`, K = `3`
- **Output:** `4` (subarray `[1, -1, 5, -2]`)

```
import java.util.HashMap;

public class LongestSubarrayWithSumK {
    public static void main(String[] args) {
        int[] arr = {1, -1, 5, -2, 3};
        int k = 3;
        int result = findLongestSubarray(arr, k);
        System.out.println("Longest subarray with sum " + k + " has length: 
" + result);
    }

    public static int findLongestSubarray(int[] arr, int k) {
        HashMap<Integer, Integer> map = new HashMap<>();
        int sum = 0, maxLength = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
            if (sum == k) {
                maxLength = i + 1;
            }
            if (map.containsKey(sum - k)) {
                maxLength = Math.max(maxLength, i - map.get(sum - k));
            }
            map.putIfAbsent(sum, i);
        }
        return maxLength;
    }
}
```

PROBLEM: 36

Problem Statement:
Write a Java program to find two numbers in an array that add up to a specific target sum.

Example:

- **Input:** `[2, 7, 11, 15]`, target = `9`
- **Output:** `[2, 7]`

```
import java.util.HashMap;

public class TwoSum {
    public static void main(String[] args) {
        int[] arr = {2, 7, 11, 15};
        int target = 9;
        int[] result = twoSum(arr, target);
        if (result != null) {
            System.out.println("Indices: " + result[0] + ", " + result[1]);
        } else {
            System.out.println("No solution found");
        }
    }

    public static int[] twoSum(int[] arr, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < arr.length; i++) {
            int complement = target - arr[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(arr[i], i);
        }
        return null;
    }
}
```

PROBLEM: 37

Problem Statement:
Write a Java program to sort an array consisting of only 0's, 1's, and 2's.

Example:

- **Input:** `[0, 1, 2, 1, 0, 2, 1]`
- **Output:** `[0, 0, 1, 1, 1, 2, 2]`

```
public class Sort012 {
    public static void main(String[] args) {
        int[] arr = {0, 1, 2, 1, 0, 2, 1};
        sort012(arr);
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void sort012(int[] arr) {
        int low = 0, mid = 0, high = arr.length - 1;
        while (mid <= high) {
            switch (arr[mid]) {
                case 0:
                    swap(arr, low++, mid++);
                    break;
                case 1:
                    mid++;
                    break;
                case 2:
                    swap(arr, mid, high--);
                    break;
            }
        }
    }

    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

PROBLEM: 38

Problem Statement:
Write a Java program to find the majority element in an array (element that appears more than `n/2` times).

Example:

- **Input:** `[3, 3, 4, 2, 4, 4, 2, 4, 4]`
- **Output:** `4`

```
public class MajorityElement {
    public static void main(String[] args) {
        int[] arr = {3, 3, 4, 2, 4, 4, 2, 4, 4};
        int result = findMajorityElement(arr);
        System.out.println("Majority Element: " + result);
    }

    public static int findMajorityElement(int[] arr) {
        int candidate = findCandidate(arr);
        return isMajority(arr, candidate) ? candidate : -1;
    }

    private static int findCandidate(int[] arr) {
        int count = 0, candidate = -1;
        for (int num : arr) {
            if (count == 0) {
                candidate = num;
            }
            count += (num == candidate) ? 1 : -1;
        }
        return candidate;
    }

    private static boolean isMajority(int[] arr, int candidate) {
        int count = 0;
        for (int num : arr) {
            if (num == candidate) {
                count++;
            }
        }
        return count > arr.length / 2;
    }
}
```

PROBLEM: 39

Problem Statement:
Write a Java program to print the subarray with the maximum sum using Kadane's algorithm.

Example:

- **Input:** `[-2, 1, -3, 4, -1, 2, 1, -5, 4]`
- **Output:** `[4, -1, 2, 1]`

```
public class MaxSubArray {
    public static void main(String[] args) {
        int[] arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        int[] result = maxSubArray(arr);
        System.out.print("Subarray with Maximum Sum: ");
        for (int num : result) {
            System.out.print(num + " ");
        }
    }

    public static int[] maxSubArray(int[] arr) {
        int maxSum = arr[0], currentSum = arr[0];
        int start = 0, end = 0, tempStart = 0;

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > currentSum + arr[i]) {
                currentSum = arr[i];
                tempStart = i;
            } else {
                currentSum += arr[i];
            }

            if (currentSum > maxSum) {
                maxSum = currentSum;
                start = tempStart;
                end = i;
            }
        }

        int[] subarray = new int[end - start + 1];
        System.arraycopy(arr, start, subarray, 0, subarray.length);
        return subarray;
    }
}
```

PROBLEM: 40

Problem Statement:
Write a Java program to calculate the maximum profit from stock trading (Buy and Sell). You can only buy and sell once.

Example:

- **Input:** `[7, 1, 5, 3, 6, 4]`
- **Output:** `5` (Buy at 1 and sell at 6)

```
public class StockBuySell {
    public static void main(String[] args) {
        int[] prices = {7, 1, 5, 3, 6, 4};
        int result = maxProfit(prices);
        System.out.println("Maximum Profit: " + result);
    }

    public static int maxProfit(int[] prices) {
        int minPrice = Integer.MAX_VALUE;
        int maxProfit = 0;
        for (int price : prices) {
            minPrice = Math.min(minPrice, price);
            maxProfit = Math.max(maxProfit, price - minPrice);
        }
        return maxProfit;
    }
}
```

PROBLEM: 41

Problem Statement:
Write a Java program to rearrange the array in alternating positive and negative items. The positive numbers should come first.

Example:

- **Input:** `[1, -2, 3, -4, 5, -6]`
- **Output:** `[1, -2, 3, -4, 5, -6]` (or other valid alternating arrangements)

```
public class RearrangeArray {
    public static void main(String[] args) {
        int[] arr = {1, -2, 3, -4, 5, -6};
        rearrange(arr);
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }

    public static void rearrange(int[] arr) {
        int positive = 0, negative = 1;
        while (positive < arr.length && negative < arr.length) {
            if (arr[positive] > 0 && arr[negative] < 0) {
                positive += 2;
                negative += 2;
            } else {
                if (arr[positive] < 0) positive++;
                if (arr[negative] > 0) negative++;
            }
        }
    }
}
```

PROBLEM: 42

Problem Statement:
Write a Java program to generate Pascal's Triangle up to the `n`-th row.

Example:

- **Input:** `n = 5`
- **Output:**
- `1`
- `1 1`
- `1 2 1`
- `1 3 3 1`
- `1 4 6 4 1`

```
import java.util.*;

public class PascalsTriangle {
    public static void main(String[] args) {
        int n = 5;
        List<List<Integer>> triangle = generate(n);
        for (List<Integer> row : triangle) {
            for (int num : row) {
                System.out.print(num + " ");
            }
            System.out.println();
        }
    }

    public static List<List<Integer>> generate(int numRows) {
        List<List<Integer>> triangle = new ArrayList<>();
        for (int rowNum = 0; rowNum < numRows; rowNum++) {
            List<Integer> row = new ArrayList<>();
            row.add(1);
            for (int j = 1; j < rowNum; j++) {
                row.add(triangle.get(rowNum - 1).get(j - 1) +
triangle.get(rowNum - 1).get(j));
            }
            if (rowNum > 0) {
                row.add(1);
            }
            triangle.add(row);
        }
        return triangle;
    }
}
```

PROBLEM: 43

Problem Statement:
Write a Java program to find the majority element in an array (an element that appears more than `n/3` times).

Example:

- **Input:** `[3, 3, 4, 2, 4, 4, 2, 4, 4]`
- **Output:** `4`

```
import java.util.*;

public class MajorityElement {
    public static void main(String[] args) {
        int[] arr = {3, 3, 4, 2, 4, 4, 2, 4, 4};
        List<Integer> result = majorityElement(arr);
        System.out.println("Majority Elements: " + result);
    }

    public static List<Integer> majorityElement(int[] nums) {
        List<Integer> result = new ArrayList<>();
        if (nums == null || nums.length == 0) return result;

        int candidate1 = Integer.MIN_VALUE, candidate2 = Integer.MIN_VALUE;
        int count1 = 0, count2 = 0;

        for (int num : nums) {
            if (num == candidate1) {
                count1++;
            } else if (num == candidate2) {
                count2++;
            } else if (count1 == 0) {
                candidate1 = num;
                count1 = 1;
            } else if (count2 == 0) {
                candidate2 = num;
                count2 = 1;
            } else {
                count1--;
                count2--;
            }
        }

        count1 = count2 = 0;
        for (int num : nums) {
            if (num == candidate1) count1++;
            if (num == candidate2) count2++;
        }
        if (count1 > nums.length / 3) result.add(candidate1);
        if (count2 > nums.length / 3) result.add(candidate2);

        return result;
    }
}
```

PROBLEM: 44

Problem Statement:
Write a Java program to find all unique triplets in an array that sum to zero.

Example:

- **Input:** `[-1, 0, 1, 2, -1, -4]`
- **Output:** `[[-1, 0, 1], [-1, -1, 2]]`

```
import java.util.*;

public class ThreeSum {
    public static void main(String[] args) {
        int[] nums = {-1, 0, 1, 2, -1, -4};
        List<List<Integer>> result = threeSum(nums);
        System.out.println(result);
    }

    public static List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        if (nums == null || nums.length < 3) return result;
        Arrays.sort(nums);

        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue; // Skip
duplicates
            int left = i + 1, right = nums.length - 1;
            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum == 0) {
                    result.add(Arrays.asList(nums[i], nums[left],
nums[right]));
                    while (left < right && nums[left] == nums[left + 1])
left++;
                    while (left < right && nums[right] == nums[right - 1])
right--;
                    left++;
                    right--;
                } else if (sum < 0) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return result;
    }
}
```

PROBLEM: 45

Problem Statement:
Write a Java program to find all unique quadruplets in an array that sum to a target.

Example:

- **Input:** `[1, 0, -1, 0, -2, 2]`, target = `0`
- **Output:** `[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]`

```
import java.util.*;

public class FourSum {
    public static void main(String[] args) {
        int[] nums = {1, 0, -1, 0, -2, 2};
        int target = 0;
        List<List<Integer>> result = fourSum(nums, target);
        System.out.println(result);
    }

    public static List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> result = new ArrayList<>();
        if (nums == null || nums.length < 4) return result;
        Arrays.sort(nums);

        for (int i = 0; i < nums.length - 3; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue; // Skip
duplicates
            for (int j = i + 1; j < nums.length - 2; j++) {
                if (j > i + 1 && nums[j] == nums[j - 1]) continue; // Skip
duplicates
                int left = j + 1, right = nums.length - 1;
                while (left < right) {
                    int sum = nums[i] + nums[j] + nums[left] + nums[right];
                    if (sum == target) {
                        result.add(Arrays.asList(nums[i], nums[j],
nums[left], nums[right]));
                        while (left < right && nums[left] == nums[left + 1])
left++;
                        while (left < right && nums[right] == nums[right -
1]) right--;
                        left++;
                        right--;
                    } else if (sum < target) {
                        left++;
                    } else {
                        right--;
                    }
                }
            }
        }
        return result;
    }
}
```

PROBLEM: 46

Problem Statement:
Write a Java program to find the largest subarray with a sum of 0.

Example:

- **Input:** `[15, -2, 2, -8, 1, 7, 10, 23]`
- **Output:** `5` (The largest subarray is `[-2, 2, -8, 1, 7]`)

```
import java.util.*;

public class LargestSubarrayWithZeroSum {
    public static void main(String[] args) {
        int[] arr = {15, -2, 2, -8, 1, 7, 10, 23};
        int result = maxLen(arr);
        System.out.println("Length of largest subarray with 0 sum: " +
result);
    }

    public static int maxLen(int[] arr) {
        Map<Integer, Integer> map = new HashMap<>();
        int maxLength = 0, sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
            if (sum == 0) {
                maxLength = i + 1;
            } else if (map.containsKey(sum)) {
                maxLength = Math.max(maxLength, i - map.get(sum));
            } else {
                map.put(sum, i);
            }
        }
        return maxLength;
    }
}
```

PROBLEM: 47

Problem Statement:
Write a Java program to count the number of subarrays that have a given XOR `K`.

Example:

- **Input:** `arr = [4, 2, 2, 6, 4], K = 6`
- **Output:** `4`

```
import java.util.*;

public class SubarraysWithGivenXor {
    public static void main(String[] args) {
        int[] arr = {4,
```

2, 2, 6, 4}; int k = 6; int result = subarrayXor(arr, k); System.out.println("Number of subarrays with XOR " + k + ": " + result); }

```
public static int subarrayXor(int[] arr, int k) {
    Map<Integer, Integer> map = new HashMap<>();
    map.put(0, 1);  // XOR of 0 appears once (base case)
    int xor = 0, count = 0;

    for (int num : arr) {
        xor ^= num;
        if (map.containsKey(xor ^ k)) {
            count += map.get(xor ^ k);
        }
        map.put(xor, map.getOrDefault(xor, 0) + 1);
    }

    return count;
}
```

}

Here are the solutions for the problems you've mentioned:

PROBLEM: 48

Problem Statement:

Next Permutation

```
import java.util.*;

public class NextPermutation {
    public static void main(String[] args) {
        int[] nums = {1, 2, 3};
        nextPermutation(nums);
        System.out.println(Arrays.toString(nums));
    }

    public static void nextPermutation(int[] nums) {
        int i = nums.length - 2;
        while (i >= 0 && nums[i] >= nums[i + 1]) i--;
        if (i >= 0) {
            int j = nums.length - 1;
            while (nums[j] <= nums[i]) j--;
            swap(nums, i, j);
        }
        reverse(nums, i + 1);
    }

    private static void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

    private static void reverse(int[] nums, int start) {
        int end = nums.length - 1;
        while (start < end) {
            swap(nums, start++, end--);
        }
    }
}
```

PROBLEM: 49

Problem Statement:

Leaders in an Array

```
import java.util.*;

public class LeadersInArray {
    public static void main(String[] args) {
        int[] arr = {16, 17, 4, 3, 5, 2};
        System.out.println("Leaders: " + findLeaders(arr));
    }

    public static List<Integer> findLeaders(int[] arr) {
        List<Integer> leaders = new ArrayList<>();
        int maxRight = arr[arr.length - 1];
        leaders.add(maxRight);

        for (int i = arr.length - 2; i >= 0; i--) {
            if (arr[i] > maxRight) {
                leaders.add(arr[i]);
                maxRight = arr[i];
            }
        }

        Collections.reverse(leaders);
        return leaders;
    }
}
```

PROBLEM: 50

Problem Statement:

Longest Consecutive Sequence in an Array (Medium)

```
import java.util.*;

public class LongestConsecutiveSequence {
    public static void main(String[] args) {
        int[] arr = {100, 4, 200, 1, 3, 2};
        System.out.println("Longest Consecutive Sequence Length: " +
longestConsecutive(arr));
    }

    public static int longestConsecutive(int[] nums) {
        Set<Integer> numSet = new HashSet<>();
        for (int num : nums) {
            numSet.add(num);
        }

        int longest = 0;
        for (int num : nums) {
            if (!numSet.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;

                while (numSet.contains(currentNum + 1)) {
                    currentNum++;
                    currentStreak++;
                }
                longest = Math.max(longest, currentStreak);
            }
        }
        return longest;
    }
}
```

PROBLEM: 51

Problem Statement: S

Set Matrix Zeros

```
public class SetMatrixZeros {
    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3}, {4, 0, 6}, {7, 8, 9}};
        setZeroes(matrix);
        System.out.println(Arrays.deepToString(matrix));
    }

    public static void setZeroes(int[][] matrix) {
        boolean firstRowZero = false, firstColZero = false;

        for (int i = 0; i < matrix.length; i++) {
            if (matrix[i][0] == 0) firstColZero = true;
        }

        for (int j = 0; j < matrix[0].length; j++) {
            if (matrix[0][j] == 0) firstRowZero = true;
        }
        for (int i = 1; i < matrix.length; i++) {
            for (int j = 1; j < matrix[i].length; j++) {
                if (matrix[i][j] == 0) {
                    matrix[i][0] = 0;
                    matrix[0][j] = 0;
                }
            }
        }

        for (int i = 1; i < matrix.length; i++) {
            for (int j = 1; j < matrix[i].length; j++) {
                if (matrix[i][0] == 0 || matrix[0][j] == 0) {
                    matrix[i][j] = 0;
                }
            }
        }

        if (firstRowZero) {
            for (int j = 0; j < matrix[0].length; j++) {
                matrix[0][j] = 0;
            }
        }

        if (firstColZero) {
            for (int i = 0; i < matrix.length; i++) {
                matrix[i][0] = 0;
            }
        }
    }
}
```

PROBLEM: 52

Problem Statement:

Rotate Matrix by 90 degrees

```
public class RotateMatrix {
    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        rotate(matrix);
        System.out.println(Arrays.deepToString(matrix));
    }

    public static void rotate(int[][] matrix) {
        int n = matrix.length;

        for (int i = 0; i < n / 2; i++) {
            for (int j = i; j < n - i - 1; j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[n - j - 1][i];
                matrix[n - j - 1][i] = matrix[n - i - 1][n - j - 1];
                matrix[n - i - 1][n - j - 1] = matrix[j][n - i - 1];
                matrix[j][n - i - 1] = temp;
            }
        }
    }
}
```

PROBLEM: 53

Problem Statement:

Print the Matrix in Spiral Manner

```
import java.util.*;

public class SpiralOrder {
    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        List<Integer> result = printSpiral(matrix);
        System.out.println(result);
    }

    public static List<Integer> printSpiral(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        if (matrix == null || matrix.length == 0) return result;

        int top = 0, bottom = matrix.length - 1;
        int left = 0, right = matrix[0].length - 1;

        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) result.add(matrix[top][i]);
            top++;

            for (int i = top; i <= bottom; i++)
result.add(matrix[i][right]);
            right--;

            if (top <= bottom) {
                for (int i = right; i >= left; i--)
result.add(matrix[bottom][i]);
                bottom--;
            }

            if (left <= right) {
                for (int i = bottom; i >= top; i--)
result.add(matrix[i][left]);
                left++;
            }
        }
        return result;
    }
}
```

PROBLEM: 54

Problem Statement:

Count Subarrays with Given Sum (Medium)

```
import java.util.*;

public class SubarraysWithGivenSum {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 0, 3};
        int sum = 3;
        System.out.println("Count of subarrays with sum " + sum + ": " +
countSubarrays(arr, sum));
    }

    public static int countSubarrays(int[] arr, int sum) {
        int count = 0;
        int currentSum = 0;
        Map<Integer, Integer> sumMap = new HashMap<>();
        sumMap.put(0, 1);

        for (int num : arr) {
            currentSum += num;
            if (sumMap.containsKey(currentSum - sum)) {
                count += sumMap.get(currentSum - sum);
            }
            sumMap.put(currentSum, sumMap.getOrDefault(currentSum, 0) + 1);
        }

        return count;
    }
}
```

PROBLEM: 55

Problem Statement: Write a Java program to merge overlapping subintervals in a list of intervals. Each interval is represented by a pair of integers `[start, end]`. The intervals may have overlaps, and your task is to merge them into non-overlapping intervals.

Java Code:

```
import java.util.*;

public class MergeIntervals {
    public static void main(String[] args) {
        int[][] intervals = {{1, 3}, {2, 4}, {5, 7}, {6, 8}};
        int[][] mergedIntervals = merge(intervals);
        System.out.println(Arrays.deepToString(mergedIntervals));
    }

    public static int[][] merge(int[][] intervals) {
        if (intervals.length == 0) return new int[0][0];

        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);

        List<int[]> merged = new ArrayList<>();
        merged.add(intervals[0]);
        for (int i = 1; i < intervals.length; i++) {
            int[] lastMerged = merged.get(merged.size() - 1);
            if (lastMerged[1] >= intervals[i][0]) {
                lastMerged[1] = Math.max(lastMerged[1], intervals[i][1]);
            } else {
                merged.add(intervals[i]);
            }
        }

        return merged.toArray(new int[merged.size()][]);
    }
}
```

PROBLEM: 56

Problem Statement: Write a Java program to merge two sorted arrays without using extra space.

Java Code:

```
import java.util.*;

public class MergeSortedArrays {
    public static void main(String[] args) {
        int[] arr1 = {1, 5, 9};
        int[] arr2 = {2, 6, 8};
        merge(arr1, arr2);
        System.out.println(Arrays.toString(arr1));
    }

    public static void merge(int[] arr1, int[] arr2) {
        int m = arr1.length;
        int n = arr2.length;

        int i = m - 1;
        int j = 0;

        while (i >= 0 && j < n) {
            if (arr1[i] > arr2[j]) {
                int temp = arr1[i];
                arr1[i] = arr2[j];
                arr2[j] = temp;
            }
            i--;
            j++;
        }

        Arrays.sort(arr1);
        Arrays.sort(arr2);
    }
}
```

PROBLEM: 57

Problem Statement: Given an array containing `n` elements where each element is in the range from 1 to `n`, find the repeating and missing number in the array.

Java Code:

```
public class RepeatingAndMissing {
    public static void main(String[] args) {
        int[] arr = {4, 3, 2, 7, 8, 2, 1, 3};
        int[] result = findRepeatingAndMissing(arr);
        System.out.println("Repeating: " + result[0] + ", Missing: " +
result[1]);
    }

    public static int[] findRepeatingAndMissing(int[] arr) {
        int[] result = new int[2];
        int n = arr.length;

        // Step 1: Compute sum and sum of squares of elements
        int sum = 0, sumOfSquares = 0;
        for (int num : arr) {
            sum += num;
            sumOfSquares += num * num;
        }

        // Step 2: Expected sum and sum of squares
        int expectedSum = n * (n + 1) / 2;
        int expectedSumOfSquares = n * (n + 1) * (2 * n + 1) / 6;

        // Step 3: Calculate differences
        int diff = expectedSum - sum; // Missing - Repeating
        int diffSquares = expectedSumOfSquares - sumOfSquares; // Missing^2
- Repeating^2

        // Step 4: Calculate missing and repeating
        result[0] = (diff + diffSquares / diff) / 2; // Repeating
        result[1] = result[0] - diff; // Missing

        return result;
    }
}
```

PROBLEM: 58

Problem Statement: Write a Java program to count the number of inversions in an array. An inversion is a pair of indices `(i, j)` such that `i < j` and `arr[i] > arr[j]`.

Java Code:

```
public class CountInversions {
    public static void main(String[] args) {
        int[] arr = {2, 4, 1, 3, 5};
        System.out.println("Inversions: " + countInversions(arr));
    }

    public static int countInversions(int[] arr) {
        return mergeSort(arr, new int[arr.length], 0, arr.length - 1);
    }

    private static int mergeSort(int[] arr, int[] temp, int left, int right)
{
        int invCount = 0;
        if (left < right) {
            int mid = (left + right) / 2;
            invCount += mergeSort(arr, temp, left, mid);
            invCount += mergeSort(arr, temp, mid + 1, right);
            invCount += merge(arr, temp, left, mid, right);
        }
        return invCount;
    }

    private static int merge(int[] arr, int[] temp, int left, int mid, int
right) {
        int i = left, j = mid + 1, k = left, invCount = 0;

        while (i <= mid && j <= right) {
            if (arr[i] <= arr[j]) {
                temp[k++] = arr[i++];
            } else {
                temp[k++] = arr[j++];
                invCount += (mid - i + 1); // All elements from i to mid
are inversions
            }
        }

        while (i <= mid) {
            temp[k++] = arr[i++];
        }

        while (j <= right) {
            temp[k++] = arr[j++];
        }

        for (i = left; i <= right; i++) {
            arr[i] = temp[i];
        }

        return invCount;
    }
}
```

PROBLEM: 58

Problem Statement: Given an array of integers, write a Java program to count the number of reverse pairs. A reverse pair is a pair `(i, j)` where `i < j` and `arr[i] > 2 * arr[j]`.

Java Code:

```
public class ReversePairs {
    public static void main(String[] args) {
        int[] arr = {1, 3, 2, 3, 1};
        System.out.println("Reverse pairs: " + reversePairs(arr));
    }

    public static int reversePairs(int[] nums) {
        return mergeSort(nums, 0, nums.length - 1);
    }

    private static int mergeSort(int[] nums, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1,
right);
        count += merge(nums, left, mid, right);
        return count;
    }

    private static int merge(int[] nums, int left, int mid, int right) {
        int count = 0;
        int j = mid + 1;

        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2L * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }

        int[] temp = new int[right - left + 1];
        int k = 0;
        int i = left, j1 = mid + 1;

        while (i <= mid && j1 <= right) {
            if (nums[i] <= nums[j1]) {
                temp[k++] = nums[i++];
            } else {
                temp[k++] = nums[j1++];
            }
        }

        while (i <= mid) temp[k++] = nums[i++];
        while (j1 <= right) temp[k++] = nums[j1++];

        for (i = 0; i < temp.length; i++) {
            nums[left + i] = temp[i];
        }

        return count;
    }
}
```

PROBLEM: 59

Problem Statement: Given an integer array, write a Java program to find the contiguous subarray within an array which has the largest product.

Java Code:

```
public class MaximumProductSubarray {
    public static void main(String[] args) {
        int[] arr = {2, 3, -2, 4};
        System.out.println("Maximum product subarray: " + maxProduct(arr));
    }

    public static int maxProduct(int[] nums) {
        if (nums.length == 0) return 0;

        int maxProd = nums[0], minProd = nums[0], result = maxProd;
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] < 0) {
                int temp = maxProd;
                maxProd = minProd;
                minProd = temp;
            }

            maxProd = Math.max(nums[i], maxProd * nums[i]);
            minProd = Math.min(nums[i], minProd * nums[i]);

            result = Math.max(result, maxProd);
        }

        return result;
    }
}
```

Congratulations on completing the PDF created by *SYNTAX ERROR****, also known as Abhishek Rathor! Your dedication and perseverance are truly commendable. For more insights and updates, feel free to connect on Instagram at*** *SYNTAX ERROR****. Keep up the great work!***