Problem 1: Array Operations

```
public class ArrayOperations {
    public static void main(String[] args) {
        int[] arr = {10, 0, 5, 20, 0, 8, 15};
        System.out.println("Second largest element: " + findSecondLargest(arr));
        moveZerosToEnd(arr);
        System.out.print("Array after moving zeros: ");
        for (int num : arr) System.out.print(num + " ");
    }

    static int findSecondLargest(int[] arr) {
        int largest = Integer.MIN_VALUE, secondLargest = Integer.MIN_VALUE;
        for (int num : arr) {
            if (num > largest) {
                secondLargest = largest;
                largest = num;
            } else if (num > secondLargest && num != largest) {
                secondLargest = num;
            }
        }
        return secondLargest;
    }

    static void moveZerosToEnd(int[] arr) {
        int index = 0;
        for (int num : arr) {
            if (num != 0) arr[index++] = num;
        }
```

```
        while (index < arr.length) arr[index++] = 0;
    }
}
```

Problem 2: String Operations

```
import java.util.Arrays;

public class StringOperations {
    public static void main(String[] args) {
        String str1 = "listen", str2 = "silent";
        String sentence = "Practice makes a man perfect";

        System.out.println("Are '" + str1 + "' and '" + str2 + "' anagrams? " + areAnagrams(str1, str2));
        System.out.println("Longest word: " + findLongestWord(sentence));
        int[] counts = countVowelsAndConsonants(sentence);
        System.out.println("Vowels: " + counts[0] + ", Consonants: " + counts[1]);
    }

    static boolean areAnagrams(String s1, String s2) {
        char[] arr1 = s1.toCharArray(), arr2 = s2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }

    static String findLongestWord(String sentence) {
        String[] words = sentence.split(" ");
        String longest = "";
```

```
        for (String word : words) if (word.length() > longest.length()) longest = word;
        return longest;
    }

    static int[] countVowelsAndConsonants(String sentence) {
        int vowels = 0, consonants = 0;
        for (char c : sentence.toLowerCase().toCharArray()) {
            if ("aeiou".indexOf(c) != -1) vowels++;
            else if (Character.isLetter(c)) consonants++;
        }
        return new int[]{vowels, consonants};
    }
}
```

Problem 3: Sorted Array Operations

```
import java.util.Arrays;

public class SortedArrayOperations {
    public static void main(String[] args) {
        int[] arr = {1, 3, 3, 3, 5, 6, 8}, peakArr = {1, 2, 18, 4, 5, 0};
        int key = 3;

        System.out.println("Key found at index: " + binarySearch(arr, key));
        System.out.println("First occurrence: " + findFirstOccurrence(arr, key));
        System.out.println("Last occurrence: " + findLastOccurrence(arr, key));
        System.out.println("Total count of key: " + countOccurrences(arr, key));
        System.out.println("Peak element: " + findPeakElement(peakArr));
    }
```

```
static int binarySearch(int[] arr, int key) {
    int low = 0, high = arr.length - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key) return mid;
        if (arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

static int findFirstOccurrence(int[] arr, int key) {
    int low = 0, high = arr.length - 1, result = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key) {
            result = mid;
            high = mid - 1;
        } else if (arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return result;
}

static int findLastOccurrence(int[] arr, int key) {
    int low = 0, high = arr.length - 1, result = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
```

```
            if (arr[mid] == key) {
                result = mid;
                low = mid + 1;
            } else if (arr[mid] < key) low = mid + 1;
            else high = mid - 1;
        }
        return result;
    }

    static int countOccurrences(int[] arr, int key) {
        return findLastOccurrence(arr, key) - findFirstOccurrence(arr, key) + 1;
    }

    static int findPeakElement(int[] arr) {
        for (int i = 1; i < arr.length - 1; i++)
            if (arr[i] > arr[i - 1] && arr[i] > arr[i + 1]) return arr[i];
        return arr[0] > arr[arr.length - 1] ? arr[0] : arr[arr.length - 1];
    }
}
```

Problem 4: Recursive Operations

```
public class RecursiveOperations {
    public static void main(String[] args) {
        int num = 7, fibIndex = 6, a = 2, b = 5;
        String str = "racecar";
        int numForSum = 1234;

        System.out.println("Is prime: " + isPrime(num, 2));
```

```
    System.out.println("Is '" + str + "' a palindrome? " + isPalindrome(str, 0, str.length() - 1));
    System.out.println("Sum of digits of " + numForSum + ": " + sumOfDigits(numForSum));
    System.out.println("Fibonacci(" + fibIndex + "): " + fibonacci(fibIndex));
    System.out.println(a + "^" + b + " = " + power(a, b));
}

static boolean isPrime(int n, int i) {
    if (n <= 2) return n == 2;
    if (n % i == 0) return false;
    if (i * i > n) return true;
    return isPrime(n, i + 1);
}

static boolean isPalindrome(String str, int left, int right) {
    if (left >= right) return true;
    if (str.charAt(left) != str.charAt(right)) return false;
    return isPalindrome(str, left + 1, right - 1);
}

static int sumOfDigits(int n) {
    if (n == 0) return 0;
    return n % 10 + sumOfDigits(n / 10);
}

static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
    static int power(int a, int b) {
        if (b == 0) return 1;
        return a * power(a, b - 1);
    }
}
```

Dry Run and Analysis

```
void printTriangle(int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j <= i; j++)
            System.out.print("*");
}
```

Dry Run for n=4n = 4n=4:

- Iterations:
    - **i = 0** → 1 print (*)
    - **i = 1** → 2 prints (**)
    - **i = 2** → 3 prints (***)
    - **i = 3** → 4 prints (****)
- Total: 1+2+3+4=101 + 2 + 3 + 4 = 101+2+3+4=10.

Time Complexity:

- Outer loop runs nnn times.
- Inner loop runs i+1i + 1i+1 times for each iii.
- Total: 1+2+3+...+n=n(n+1)21 + 2 + 3 + ... + n = \frac{n(n+1)}{2}1+2+3+...+n=2n(n+1).
- **Time Complexity: O(n2)O(n^2)O(n2)**.

Question 2: Pattern Printing

java

CopyEdit

```
void printPattern(int n) {
   for (int i = 1; i <= n; i *= 2)
      for (int j = 0; j < n; j++)
         System.out.println(i + "," + j);
}
```

Dry Run for n=8n = 8n=8:

- Outer loop (i): 1,2,4,81, 2, 4, 81,2,4,8 → runs log⁡2(n)\log_2(n)log2(n) times.
- Inner loop (j): runs nnn times for each iii.

Total Iterations:

- log⁡2(n)×n=4×8=32\log_2(n) \times n = 4 \times 8 = 32log2(n)×n=4×8=32.

Time Complexity:

- **Outer loop:** O(log⁡n)O(\log n)O(logn).
- **Inner loop:** O(n)O(n)O(n).
- Combined: O(nlog⁡n)O(n \log n)O(nlogn).

Question 3: Recursive Half

java

CopyEdit

```
void recHalf(int n) {
   if (n <= 0) return;
   System.out.print(n + " ");
   recHalf(n / 2);
}
```

Dry Run for n=20n = 20n=20:

- n=20→10→5→2→1→0n = 20 → 10 → 5 → 2 → 1 → 0n=20→10→5→2→1→0.
- Recursive calls: 5.
- Printed values: 20 10 5 2 1.

Time Complexity:

- Number of calls proportional to log⁡2(n)\log_2(n)log2(n).

- **Time Complexity: O(log⁡n)O(\log n)O(logn)**.

v**Question 4: Exponential Recursion**

java

CopyEdit

```
void fun(int n) {
   if (n == 0) return;
   fun(n - 1);
   fun(n - 1);
}
```

Dry Run for n=3n = 3n=3:

- **Calls Breakdown:**
  - fun(3)→fun(2),fun(2)fun(3) → fun(2), fun(2)fun(3)→fun(2),fun(2).
  - fun(2)→fun(1),fun(1)fun(2) → fun(1), fun(1)fun(2)→fun(1),fun(1).
  - fun(1)→fun(0),fun(0)fun(1) → fun(0), fun(0)fun(1)→fun(0),fun(0).
- Total calls: 2n−1=23−1=72^n - 1 = 2^3 - 1 = 72n−1=23−1=7.

Time Complexity:

- T(n)=2T(n−1)+1T(n) = 2T(n-1) + 1T(n)=2T(n−1)+1 → O(2n)O(2^n)O(2n).

Question 5: Triple Nested Loops

java

CopyEdit

```
void tripleNested(int n) {
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         for (int k = 0; k < n; k++)
            System.out.println(i + j + k);
}
```

Dry Run for n=3n = 3n=3:

- Total iterations: n3=33=27n^3 = 3^3 = 27n3=33=27.

Time Complexity:

- Each loop runs nnn times.
- Combined: O(n3)O(n^3)O(n3).