ADS Assignment

Name Akshay Janrao

PRN 250240520006

Problem 1:

Given an array of integers, perform the following operations:

1. Find the second largest element in the array.

2. Move all zeros to the end of the array while maintaining the order of non-zero

elements.

Input:

arr = [10, 0, 5, 20, 0, 8, 15]

Output:

Second largest element: 15

Array after moving zeros: [10, 5, 20, 8, 15, 0, 0]

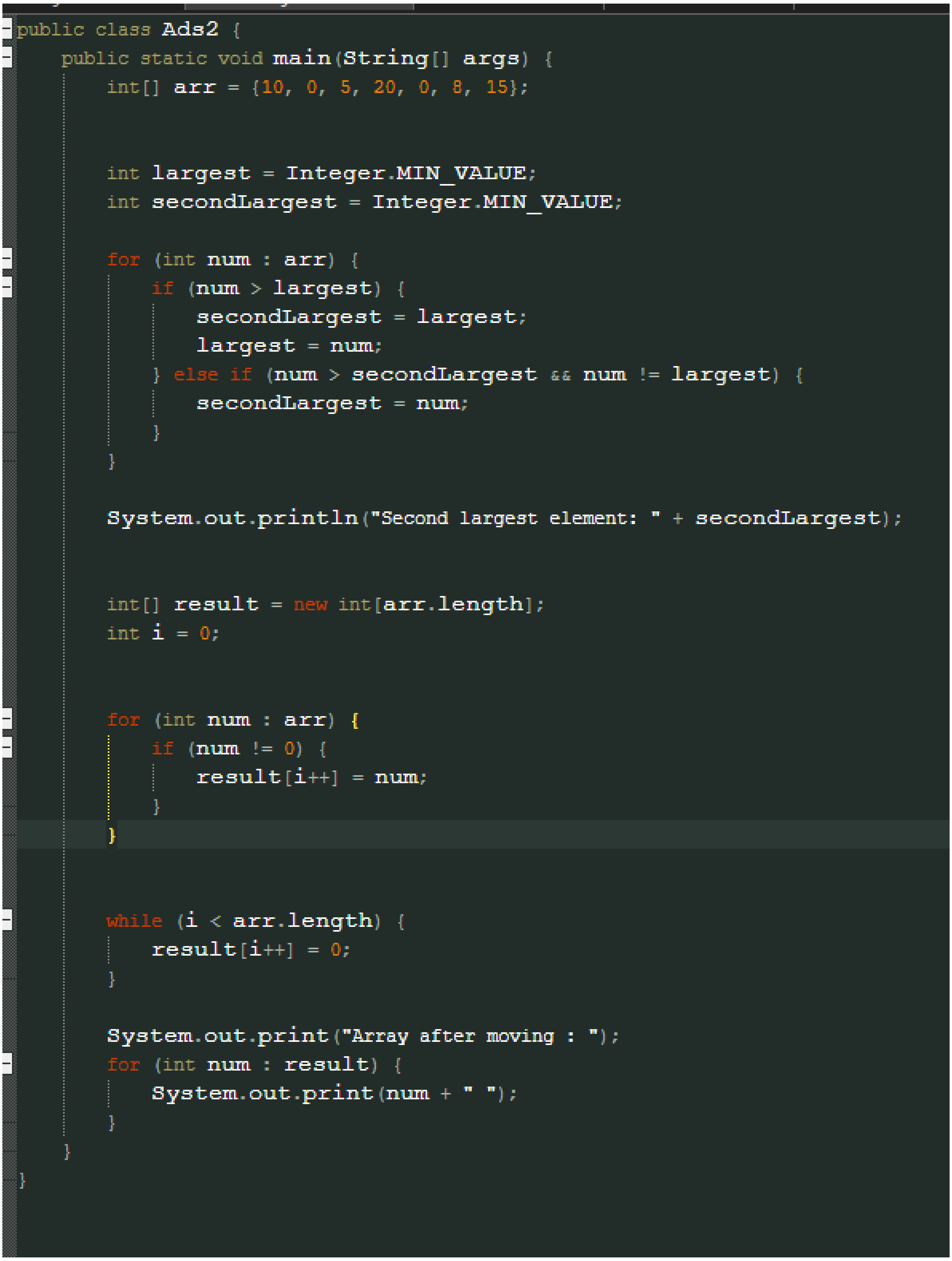
Constraints:

● Do not use built-in sort functions.

● The array may contain duplicate elements or zeros at any position.

● Array length ≥ 2.

Answer



```java
public class Ads2 {
    public static void main(String[] args) {
        int[] arr = {10, 0, 5, 20, 0, 8, 15};


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

        for (int num : arr) {
            if (num > largest) {
                secondLargest = largest;
                largest = num;
            } else if (num > secondLargest && num != largest) {
                secondLargest = num;
            }
        }

        System.out.println("Second largest element: " + secondLargest);


        int[] result = new int[arr.length];
        int i = 0;


        for (int num : arr) {
            if (num != 0) {
                result[i++] = num;
            }
        }


        while (i < arr.length) {
            result[i++] = 0;
        }

        System.out.print("Array after moving : ");
        for (int num : result) {
            System.out.print(num + " ");
        }
    }
}
```

Problem 2:

Write a program that performs the following operations on strings:

1. Check whether two given strings are anagrams of each other.

2. Identify the longest word in a given sentence.

3. Count the number of vowels and consonants in the same sentence.

Input:

String 1: listen

String 2: silent

Sentence: Practice makes a man perfect

Output:

Are 'listen' and 'silent' anagrams? true

Longest word: Practice

Vowels: 9, Consonants: 17

Answer

```
import java.util.Arrays;

public class Ads3 {
  public static void main(String[] args) {


    String a1 = "listen";
    String a2 = "silent";
    String sentence = "Practice makes a man perfect";


    boolean areAnagrams = checkAnagrams(a1, a2);
    System.out.println("Are '" + a1 + "' and '" + a2 + "' anagrams? " + areAnagrams);



    String longestWord = findLongestWord(sentence);
    System.out.println("Longest word: " + longestWord);
    int[] vowelConsonantCount = countVowelsAndConsonants(sentence);

    System.out.println("Vowels: " + vowelConsonantCount[0] + ", Consonants: " + vowelConsonantCount[1]);

}




public static boolean checkAnagrams(String s1, String s2) {

    char[] arr1 = s1.toLowerCase().toCharArray();

    char[] arr2 = s2.toLowerCase().toCharArray();

    Arrays.sort(arr1);

    Arrays.sort(arr2);

    return Arrays.equals(arr1, arr2);

}




public static String findLongestWord(String sentence) {

    String[] words = sentence.split(" ");

    String longest = "";

    for (String word : words) {

        if (word.length() > longest.length()) {

            longest = word;

        }

    }

    return longest;

}




public static int[] countVowelsAndConsonants(String sentence) {

    int vowels = 0, consonants = 0;

    String lowerCaseSentence = sentence.toLowerCase();
```

```
        for (char ch : lowerCaseSentence.toCharArray()) {
            if (ch >= 'a' && ch <= 'z') {
                if ("aeiou".indexOf(ch) >= 0) {
                    vowels++;
                } else {
                    consonants++;
                }
            }
        }
        return new int[]{vowels, consonants};
    }
}
```

Problem 3:

Given a sorted array of integers (which may include duplicates), perform the following

operations:

1. Search for a given key and return its index (if found) with Binary Search.

2. Find the first and last occurrence of the key in the array.

3. Count the total number of times the key appears.

4. Find any peak element in the array (an element greater than its neighbors).

Input:

arr = [1, 3, 3, 3, 5, 6, 8], key = 3

Input for Peak Element:

arr =[1, 2, 18, 4, 5, 0]

Output:

```
Key found at index: 2
First occurrence: 1
Last occurrence: 3
Total count of key: 3
Peak element: 18
```

Answer

```java
public class Ads4 {
    public static void main(String[] args) {


        int[] arr = {1, 3, 3, 3, 5, 6, 8};
        int key = 3;



        int keyIndex = binarySearch(arr, key);
        System.out.println("Key found at index: " + keyIndex);



        int first = findFirst(arr, key);
        int last = findLast(arr, key);
        System.out.println("First occurrence: " + first);
        System.out.println("Last occurrence: " + last);
    int totalCount = count(arr, key);
    System.out.println("Total count of key: " + totalCount);


    int[] peakArr = {1, 2, 18, 4, 5, 0};
    int peakElement = findPeakElement(peakArr);
    System.out.println("Peak element: " + peakElement);
}


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


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



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

public static int count(int[] arr, int key) {
    int first = findFirst(arr, key);
    int last = findLast(arr, key);
    if (first == -1 || last == -1) {
        return 0;
    }
```

```
        return last - first + 1;
    }



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

Problem 4:

Write a recursive program that performs the following operations:

1. Check if a number is prime using recursion.

2. Check whether a given string is a palindrome.

3. Find the sum of digits of a given number.

4. Calculate the nth Fibonacci number.

5. Calculate a raised to the power b

Input:

```
num = 7
str = "racecar"
num = 1234
fibIndex = 6
a = 2, b = 5
```

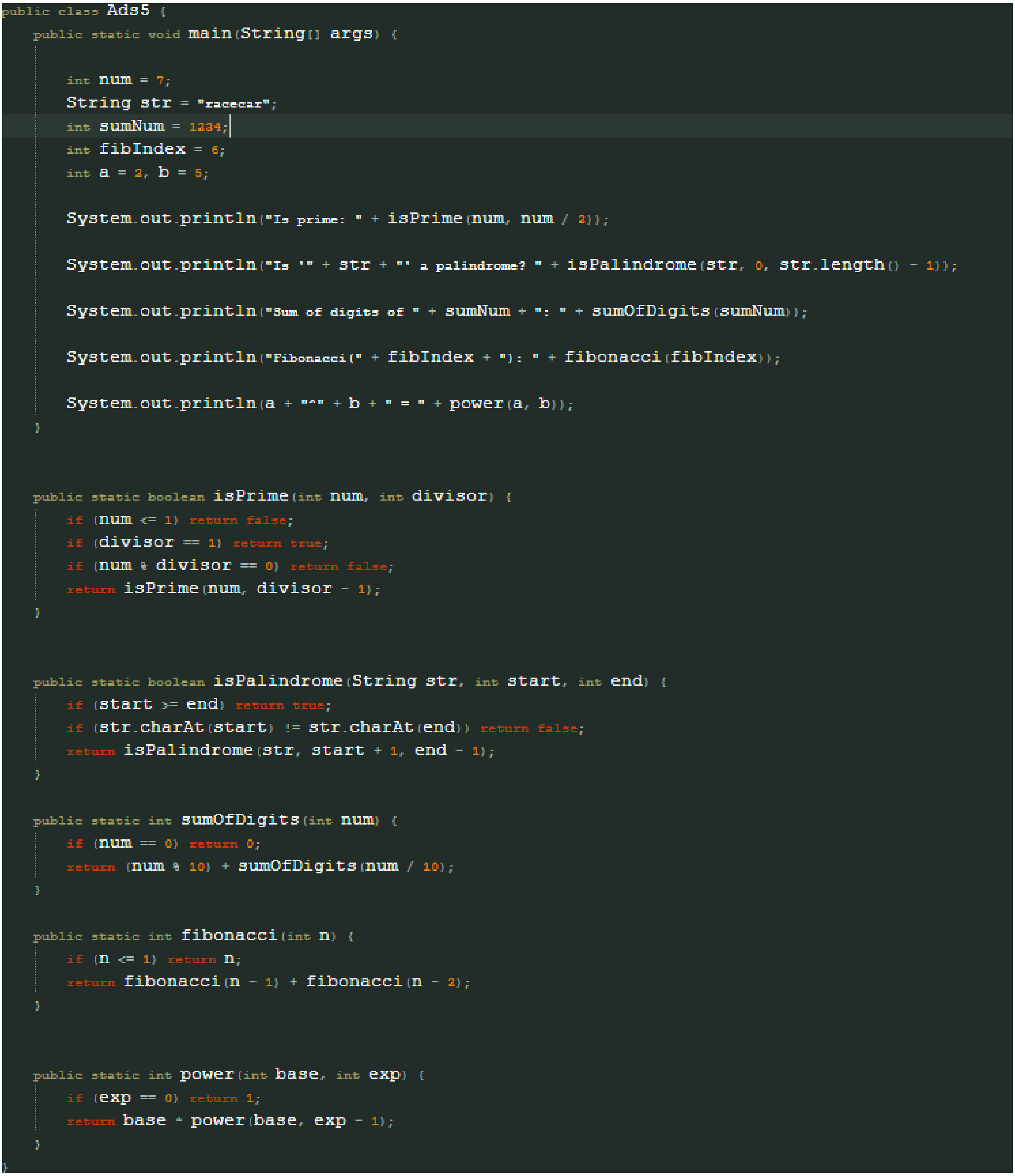
Output:

```
Is prime: true
Is 'racecar' a palindrome? true
Sum of digits of 1234: 10
Fibonacci(6): 8
2^5 = 32
```

Constraints:

- Do not use loops or built-in reverse methods.
- Use charAt() for string access.
- You can assume valid positive integer inputs.

Answer



```
public class Ads5 {
    public static void main(String[] args) {

        int num = 7;
        String str = "racecar";
        int sumNum = 1234;
        int fibIndex = 6;
        int a = 2, b = 5;

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

        System.out.println("Is '" + str + "' a palindrome? " + isPalindrome(str, 0, str.length() - 1));

        System.out.println("Sum of digits of " + sumNum + ": " + sumOfDigits(sumNum));

        System.out.println("Fibonacci(" + fibIndex + "): " + fibonacci(fibIndex));

        System.out.println(a + "^" + b + " = " + power(a, b));
    }


    public static boolean isPrime(int num, int divisor) {
        if (num <= 1) return false;
        if (divisor == 1) return true;
        if (num % divisor == 0) return false;
        return isPrime(num, divisor - 1);
    }


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

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

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


    public static int power(int base, int exp) {
        if (exp == 0) return 1;
        return base * power(base, exp - 1);
    }
}
```

Problem 5:

Dry Run & Analyze: Time and Space Complexity

1. Dry run the code for n = 4. How many times is * printed? What is the time complexity?

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

Answer

The time complexity of the given program is O(n^2),

And the "*" will be printed 10 times.

2. Dry run for n = 8. What's the number of iterations? Time complexity?

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

Answer the loop will run for 32 iterations and the time complexity is O(log n)

3. Dry run for n = 20. How many recursive calls? What values are printed?

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

Answer

The recursive call will be made 6 times with the time complexity of O (log n).

4. Dry run for n = 3. How many total calls are made? What’s the time complexity?

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

Answer   The recursive call will be made 15 times and the time complexity is O (2^n).

5. Dry run for n = 3. How many total iterations? Time complexity?

```
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);
}
```

Answer

The total no of iterations will be 27 times with time complexity of O(n^3).