ADS

Assignment - 1

Algorithm and Data Structure (ADS)

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-functions
- the arrays may contain duplicate elements or zeros at any position
- Array length ≥ 2.

```
public class ArrayOperations {

    public static int findSecondLargest(int[] arr) {
        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;
            }
        }
        return secondLargest;
    }
```

```
Public 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
    }
}
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.println("Array after moving
                    Zeros: " + Arrays.toString
                    (arr));
}
}
```

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.

Input:

String 1: Listen

Ans-

```
Public class StringOperations {
    public static boolean areAnagrams(String str1,
                                       String str2) {
        char[] arr1 = str1.replaceAll("\\s", "").
                      toLowerCase().toCharArray();
        char[] arr2 = str2.replaceAll("\\s", "").
                      toLowerCase().toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }

    public static String findLongestWord(String sentence) {
        String[] words = sentence.split(" ");
        String longestWord = "";
        for (String word : words) {
            if (word.length() > longestWord.length()) {
                longestWord = word;
            }
        }
        return longestWord;
    }
```

```
Public static void countVowelsAndConsonants(String
        Sentence) {
    int vowels = 0, consonants = 0;
    Sentence = Sentence.toLowerCase();
    for(char ch : sentence.toCharArray()){
    {
      if (character.isLetter(ch)) {
          if("aeiou".indexOf(ch) != -1) {
              vowels ++;
          } else {
              consonants ++;
          }
      }
  }
  System.out.println("Vowels:" + vowels +
          ", Consonants:" + consonants);
}
public static void main(String[]args) {
    String str1 = "listen", str2 = "silent";
    String Sentence = "Practise makes a man perfect";
    System.out.println("Are 'listen' and 'silent' anagrams? " + areAnagrams
                                                    (str1, str2));
    System.out.println("Longest word:" + findLongest
            word(Sentence));
    countVowelsAndConsonants(Sentence);
  }
}
```

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

(1)

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

// first occurrence

(2) Public static int findFirstOccurence(int[] arr, int key) {
int index = binarySearch(arr, key);
if (index == -1)
return -1;
while (index > 0 && arr[index-1] == key) {
index--;
}
return index;
}

// last occurrence
public static int findLastOccurence(int[] arr, int key) {
int index = binarySearch(arr, key);
if (index == -1)
return -1;
while (index < arr.length-1 && arr[index+1]
== key) {
index++;
}
return index;
}

(3) Public static int countOccurences(int[] arr, int key) {
int first = findfirstOccurences(arr, key);
int last = findLastOccurences(arr, key);
if (first == -1 || last == -1)
return 0;
return last - first + 1;
}

(4) Public 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 -1;
}

Public static void main(String[] args) {
    int[] arr1 = {1,3,3,3,5,6,8};
    int key=3;

System.out.println("Key found at index :" + binarySearch(
System.out.println("First occurence:" + findFirst(
                                        arr1, key));

System.out.println("Last occurence :" + findLast(
                                        arr1, key));

System.out.println("Total count of key :" + count(
                                        arr1, key));


int[] arr2 = {1,2,18,4,5,0};
    System.out.println("Peak element :" + findPeak
                                        Element(arr2));

    }
}
```

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

constraints:
- DO'nt use loops or built-in reverse method.
- Use charAt() for string access
- YOU can assume valid positive integer inputs

Ans - (1) Public static boolean isPrime(int num, int divisor) {
```
if(num<=2)
return (num==2);
if(num % divisor ==0)
return false;
if (divisor * divisor > num)
    return true;
return isPrime(num, divisor+1);
}
```

(2) 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);
}
```

(3) Public static int SumofDigits(int num) {
```
if (num == 0)
```

```
return 0;
return (num % 10) + sumofDigits(num / 10);
}
```

④
```
Public static int fibonacci(int n){
    if (n<=1)
    return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

⑤
```
Public static int power(int a, int b){
    if (b==0)
      return 1;
    return a * power(a, b-1);
}
```

```
Public static void main(String[] args){
    System.out.println("IS prime:" + isprime(7,2));
    System.out.println("IS 'racecar' a palindrome?
                        + isPalindrome('racecar', 0,6));
    System.out.println("sum of digits of 1234:" +
                        sumofDigits(1234));
    System.out.println("fibonacci(6):" + fibonacci(6));
    System.out.println("2^5=" + power(2,5));
    }
}
```

Problem 5:

Dry Run & Analyze: Time and Space complexity

1. Dry run the code for n=4. How many times is it 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("*");
}
```

Number of * printed for n=4;
1+2+3+4 = 10 times

Time complexity: $O(n^2)$

(2) Dry run for n=8, what's the number of iterations? Time

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

Number of iterations for n=8
i values are 1, 2, 4, 8 → Total iterations = 4x8 = 32

Time complexity: $O(n \log n)$

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

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

Values printed for n=20;
20 10 5 2 1
Number of Recursive calls: 5
Time complexity: $O(\log n)$

(4) Dry run for n=3. How many total calls are made? What time complexity

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

Ans- Number of calls for n=3;

$2^3 - 1 = 7$ calls

Time complexity : $O(2^n)$

(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++) → S.O.P(i+j+k)
}
```

Ans- Number of iterations for n=3;

$n^3 = 27$ iterations

Time complexity : $O(n^3)$