**BCSE103E – Java**

**Exercises**

**B S Naveen – 23BAI1069**

Link:

1. You are given the heights of consecutive buildings. You can move from the roof of a

building to the roof of the next adjacent building. You need to find the maximum number

of consecutive steps you can put forward such that you gain an increase in altitude with

each step.

**Code:**

import java.util.Scanner;

public class Altitude {

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i=0;i<n;i++){

            arr[i] = sc.nextInt();

        }

        int c=0;

        int max=0;

        int i=0;

        while (i<n-1){

            c=0;

            while (i<n-1 && arr[i]<arr[i+1]){

                c++;

                i++;

            }

            if (c > max){

                max=c;

            }

            i++;

        }

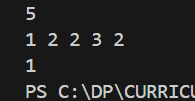
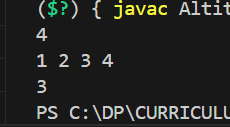
        System.out.println(max);

        sc.close();

     }

}

**Output:**

** **

2. Given a singly linked list of integers. The task is to check if the given linked list is

palindrome or not.

**Code:**

import java.util.Scanner;

class Node{

    int data;

    Node next;

    Node(){

        data=0;

        next=null;

    }

    Node(int x){

        data=x;

        next=null;

    }

}

public class Linkedlist {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        Node head = new Node();

        Node d = head;

        for (int i=0;i<n;i++){

            Node temp = new Node(sc.nextInt());

            d.next = temp;

            d = temp;

        }

        sc.close();

        Node copy = new Node();

        for (Node i=head.next; i!=null; i=i.next){

            Node t = new Node();

            t.data=i.data;

            t.next = copy;

            copy = t;

        }

        Node j=head.next;

        Node k=copy;

        while (j!=null && k!=null){

            if (j.data != k.data){

                System.out.println(false);

                return;

            }

            j=j.next;

            k=k.next;

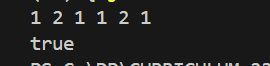
        }

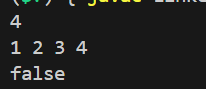
        System.out.println(true);

    }

}

**Output:**

****

****

3. Given two strings **s** and **p**. Find the smallest window in the string **s** consisting of all the

characters(**including duplicates**) of the string **p**. Return "**-1**" in case there is no such

window present. In case there are multiple such windows of same length, return the one

with the **least starting index**.

**Code:**

import java.util.Scanner;

public class Window{

    public static String smallestWindow(String s, String p) {

        if (s.length() < p.length()) {

            return "-1";

        }

        int[] pCnt = new int[26];

        for (char c : p.toCharArray()) {

            pCnt[c - 'a']++;

        }

        int[] windowCnts = new int[26];

        int required = 0;

        for (int count : pCnt) {

            if (count > 0) required++;

        }

        int l = 0, r = 0, formed = 0;

        int minLen = 100001;

        int minLeft = 0;

        while (r < s.length()) {

            char c = s.charAt(r);

            windowCnts[c - 'a']++;

            if (pCnt[c - 'a'] > 0 && windowCnts[c - 'a'] == pCnt[c - 'a']) {

                formed++;

            }

            while (l <= r && formed == required) {

                c = s.charAt(l);

                if (r - l + 1 < minLen) {

                    minLen = r - l + 1;

                    minLeft = l;

                }

                windowCnts[c - 'a']--;

                if (pCnt[c - 'a'] > 0 && windowCnts[c - 'a'] < pCnt[c - 'a']) {

                    formed--;

                }

                l++;

            }

            r++;

        }

        return minLen == Integer.MAX\_VALUE ? "-1" : s.substring(minLeft, minLeft + minLen);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String s1 = sc.next();

        String p1 = sc.next();

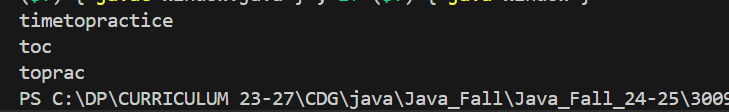
        sc.close();

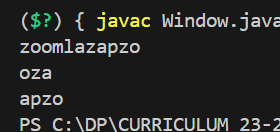
        System.out.println(smallestWindow(s1, p1));

    }

}

**Output:**

****

****

4. Given an unsorted array ***arr*** of positive integers. One number '**A**' from set {1, 2,....,n}

is missing and one number '**B**' occurs twice in array. Find numbers **A** and **B**.

**Note:** The test cases are generated such that there always exists one missing and one repeating

number within the range **[1,n]**.

**Code:**

import java.util.Scanner;

public class Find {

    public static void findMissingAndRepeating(int[] arr) {

        int n = arr.length;

        int repeating = -1;

        int missing = -1;

        for (int i = 0; i < n; i++) {

            int index = Math.abs(arr[i]) - 1;

            if (arr[index] < 0) {

                repeating = Math.abs(arr[i]);

            } else {

                arr[index] = -arr[index];

            }

        }

        for (int i = 0; i < n; i++) {

            if (arr[i] > 0) {

                missing = i + 1;

                break;

            }

        }

        System.out.println(repeating+" "+missing);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

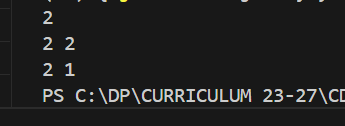
        }

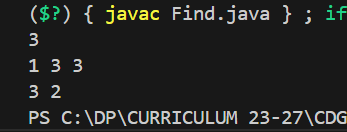
        findMissingAndRepeating(arr);

        scanner.close();

    }}

**Output:**

****

****

5. Given a string of characters, find the length of the longest proper prefix which is also a

proper suffix.

**Code:**

import java.util.Scanner;

public class Longest {

    public static int findLongest(String str) {

        int n = str.length();

        int[] arr = new int[n];

        compute(str, arr);

        return arr[n - 1];

    }

    private static void compute(String str, int[] arr) {

        int n = str.length();

        int i = 1, j = 0;

        while (i < n) {

            if (str.charAt(i) == str.charAt(j)) {

                j++;

                arr[i] = j;

                i++;

            } else {

                if (j == 0) {

                    arr[i] = 0;

                    i++;

                } else {

                    j = arr[j - 1];

                }

            }

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a string: ");

        String str = sc.nextLine();

        int length = findLongest(str);

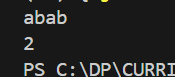
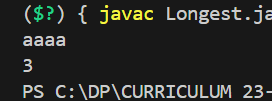
        System.out.println("Length of the longest proper prefix which is also a proper suffix: " + length);

        sc.close();

    }

}

**Output:**

** **