Name: Abishek S H Dept: CSE

Regno: 22CS009 Date:9/11/24

1**.Maximum sum subarray**

Code:

package sample1;

import java.util.\*;

public class MaxSum {

public static void main(String[] args) {

Scanner sc=new Scanner(System.***in***);

System.***out***.println("Enter in the size of n");

int n=sc.nextInt();

int[] arr=new int[n];

for(int i=0;i<n;i++) {

arr[i]=sc.nextInt();

}

int curSum=arr[0];

int maxSum=arr[0];

for(int j=1;j<n;j++) {

curSum=Math.*max*(arr[j],curSum+arr[j]);

maxSum=Math.*max*(maxSum, curSum);

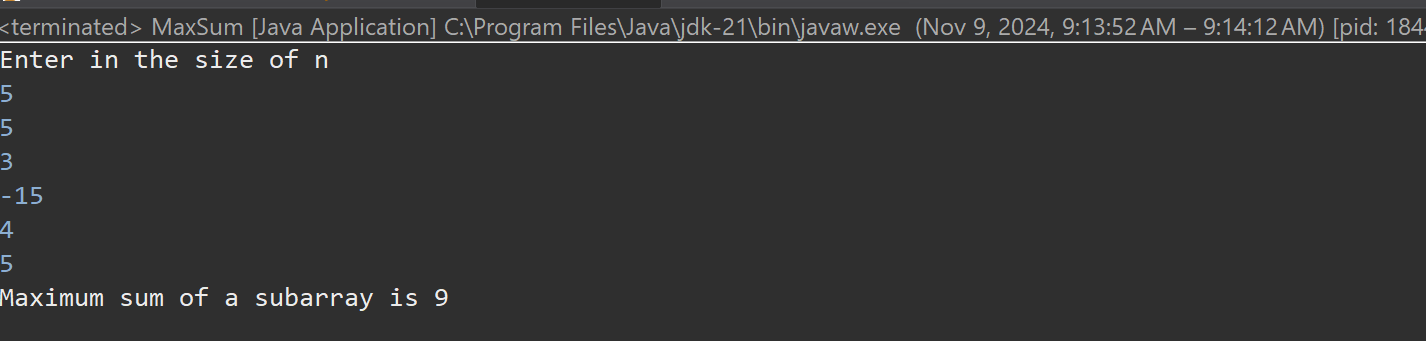
}

System.***out***.println("Maximum sum of a subarray is "+maxSum);

}

}

Output:



Time complexity : O(n)

**2. Maximum product Subarray**

Code:

package sample1;

import java.util.\*;

public class MaxPro {

public static void main(String[] args) {

Scanner sc=new Scanner(System.***in***);

System.***out***.println("Enter the size of n: ");

int n=sc.nextInt();

int[] arr=new int[n];

for(int i=0;i<n;i++) {

arr[i]=sc.nextInt();

}

int pro=1;

int max=Integer.***MIN\_VALUE***;

for(int i=0;i<n;i++) {

pro\*=arr[i];

max=Math.*max*(max, pro);

if(pro==0) pro=1;

}

pro=1;

for(int i=n-1;i>=0;i--) {

pro\*=arr[i];

max=Math.*max*(max, pro);

if(pro==0) pro=1;

}

System.***out***.println("Maximum product of subarray is "+max);

}

}

Output:



Time complexity: O(n)

**3.Search in Rotated Sorted array**

Code:

package sample1;

import java.util.\*;

public class RotatedSearch {

public static int search(int[] arr, int target) {

int left=0;

int right=arr.length-1;

while(left<=right) {

int mid=left+(right-left)/2;

if(arr[mid]==target) {

return mid;

}

if(arr[left]<=arr[mid]) {

if(target>=arr[left] && target<arr[mid]) {

right=mid-1;

}

else {

left=mid+1;

}

}

else {

if(target>arr[mid] && target<=arr[right]) {

left=mid+1;

}

else {

right=mid-1;

}

}

}

return -1;

}

public static void main(String[] args) {

/\*Scanner sc=new Scanner(System.in);

System.out.println("Enter the size of the array:");

int n=sc.nextInt();

int[] arr=new int[n];

System.out.println("Enter the elements in sorted order:");

for(int i=0;i<n;i++) {

arr[i]=sc.nextInt();

}

System.out.println("Enter the target element:");

int target=sc.nextInt();\*/

int[] arr= {4,5,0,1,2,3};

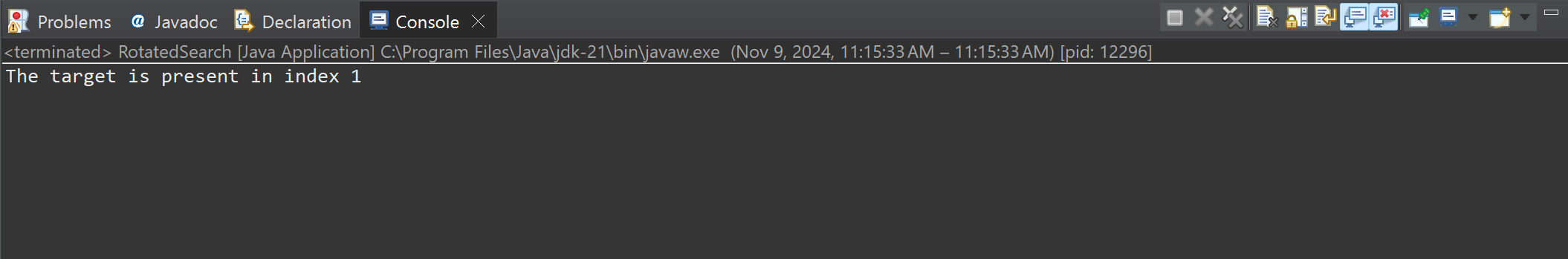
int res=*search*(arr,5);

System.***out***.println(res);

}

}

Output:



Time complexity: O(log n)

**4. Container with most water**

Code :

package sample1;

import java.util.\*;

public class ContainerWater {

static int WaterContain(int[] arr) {

int left=0;

int right=arr.length-1;

int area=0;

int parea=0;

while(left<right) {

parea=Math.*min*(arr[left], arr[right])\*(right-left);

area=Math.*max*(area, parea);

if(arr[left]<arr[right]) {

left+=1;

}

else {

right-=1;

}

}

return area;

}

public static void main(String[] args) {

int[] arr= {1,7,4,1,3,6};

int res=*WaterContain*(arr);

System.***out***.println(res);

int[] arr1= {1,5,4,3};

int res1=*WaterContain*(arr1);

System.***out***.println(res1);

int[] arr2= {3,1,2,4,6};

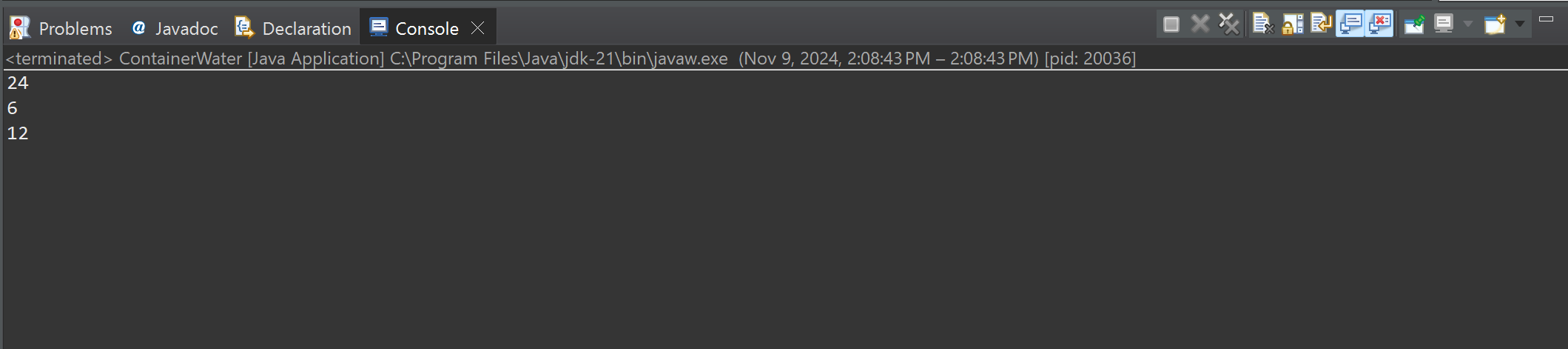
int res2=*WaterContain*(arr2);

System.***out***.println(res2);

}

}

Output:



Time Complexity: O(n)

**5.Factorial of a large number**

Code:

package sample1;

import java.util.\*;

import java.math.BigInteger;

public class Factorial {

public static void main(String[] args) {

Scanner sc=new Scanner(System.***in***);

System.***out***.println("Enter the value of n");

int n=sc.nextInt();

BigInteger fact=BigInteger.***ONE***;

for(int i=2;i<=n;i++) {

fact= fact.multiply(BigInteger.*valueOf*(i));

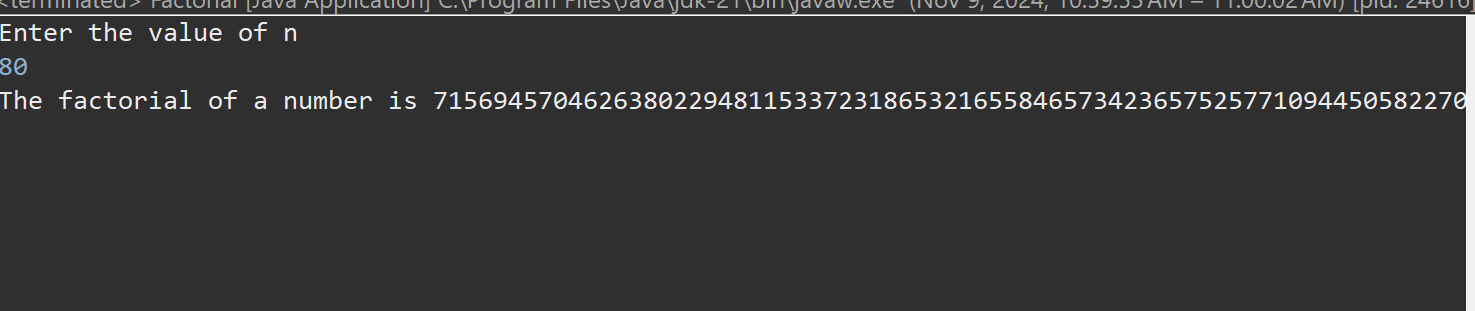
}

System.***out***.println("The factorial of a number is "+fact);

}

}

Output:



Time Complexity: O(n)

**6. Trapping rain water**

Code:

package sample1;

public class RainWater {

static int solve(int[] height) {

if (height == null || height.length == 0) {

return 0;

}

int left = 0;

int right = height.length - 1;

int lMax = 0;

int rMax = 0;

int out = 0;

while (left < right) {

if (height[left] < height[right]) {

if (height[left] >= lMax) {

lMax = height[left];

} else {

out+= lMax - height[left];

}

left++;

} else {

if (height[right] >= rMax) {

rMax = height[right];

} else {

out+= rMax - height[right];

}

right--;

}

}

return out;

}

public static void main(String[] args) {

int[] arr = {3, 0, 1, 0, 4, 0, 2};

int res=*solve*(arr);

System.***out***.println(res);

int[] arr1 = {3, 0, 2,0,4};

int res1=*solve*(arr1);

System.***out***.println(res1);

int[] arr2 = {1,2,3,4};

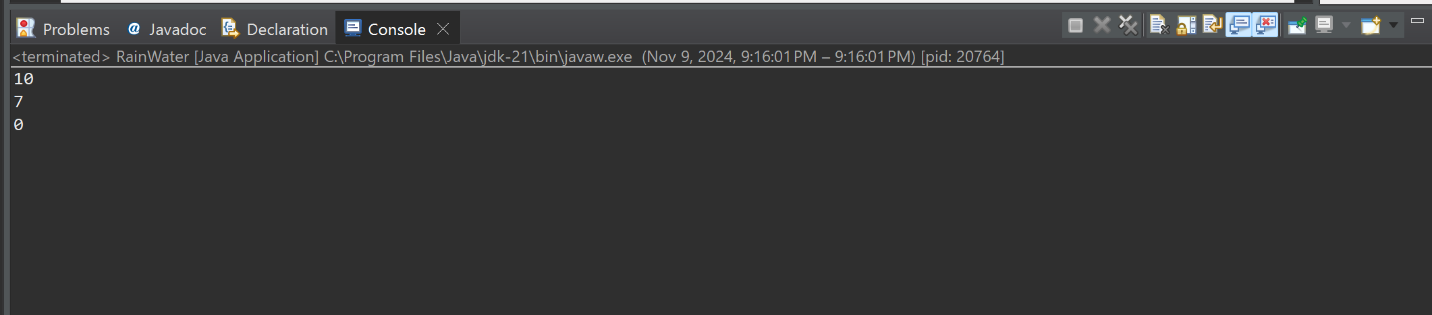
int res2=*solve*(arr2);

System.***out***.println(res2);

}

}

Output:



Time Complexity: O(n)

**7. Chocolate Distribution Problem**

Code:

package sample1;

import java.util.\*;

public class ChocolateDistribution {

static int findMinDiff(int[] arr, int n, int m) {

if (m == 0 || n == 0) {

return 0;

}

Arrays.*sort*(arr);

if (n < m) {

return -1;

}

int minDiff = Integer.***MAX\_VALUE***;

for (int i = 0; i + m - 1 < n; i++) {

int diff = arr[i + m - 1] - arr[i];

minDiff = Math.*min*(minDiff, diff);

}

return minDiff;

}

public static void main(String[] args) {

int arr1[] = {7, 3, 2, 4, 9, 12, 56};

int m1 = 3;

int n1 = arr1.length;

System.***out***.println("Minimum difference (m=3): " + *findMinDiff*(arr1, n1, m1));

int arr2[] = {7, 3, 2, 4, 9, 12, 56};

int m2 = 5;

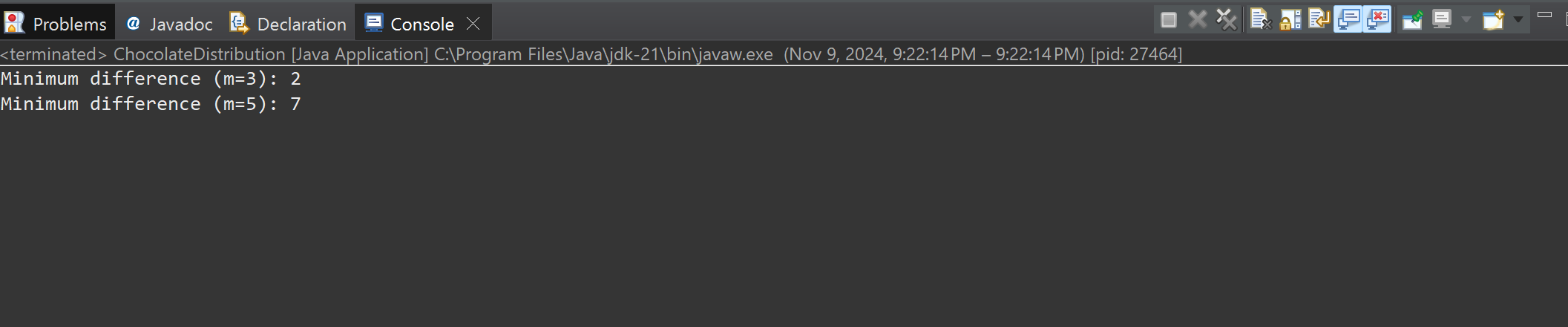
int n2 = arr2.length;

System.***out***.println("Minimum difference (m=5): " + *findMinDiff*(arr2, n2, m2));

}

}

Output:



Time complexity: O(n log n)

**8. Merge overlapping interval**

Code:

package sample1;

import java.util.\*;

public class MergeIntervals {

public static int[][] mergeIntervals(int[][] intervals) {

if (intervals == null || intervals.length == 0) {

return new int[0][0];

}

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

List<int[]> merged = new ArrayList<>();

merged.add(intervals[0]);

for (int i = 1; i < intervals.length; i++) {

int[] current = intervals[i];

int[] lastMerged = merged.get(merged.size() - 1);

if (current[0] <= lastMerged[1]) {

lastMerged[1] = Math.*max*(lastMerged[1], current[1]);

} else {

merged.add(current);

}

}

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

}

public static void main(String[] args) {

int[][] intervals1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};

int[][] result1 = *mergeIntervals*(intervals1);

System.***out***.println("Merged Intervals 1: " + Arrays.*deepToString*(result1));

int[][] intervals2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

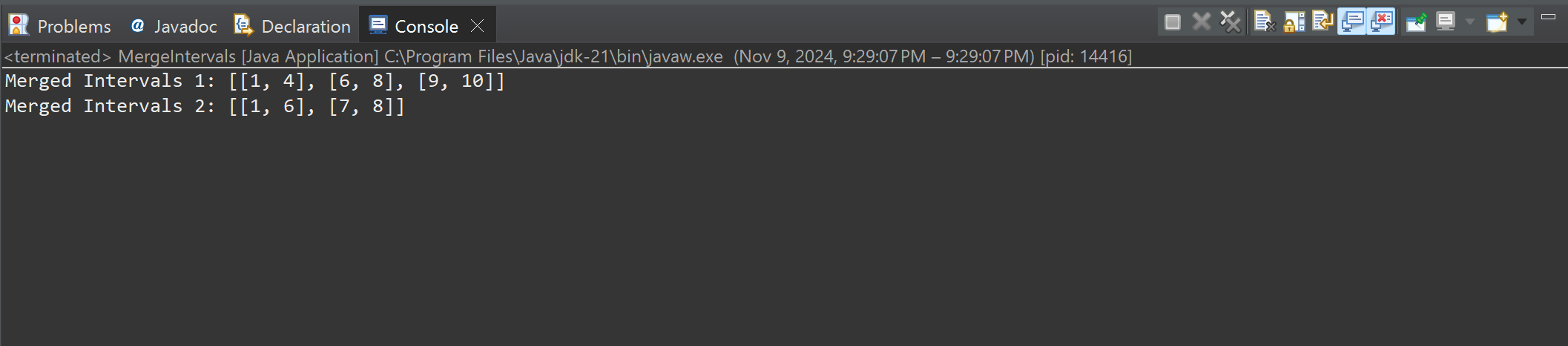
int[][] result2 = *mergeIntervals*(intervals2);

System.***out***.println("Merged Intervals 2: " + Arrays.*deepToString*(result2));

}

}

Output:



Time Complexity: O(n log n)

**9.BooleanMatrix**

Code:

package sample1;

import java.util.\*;

public class BooleanMatrix {

public static void modify(int[][] mat) {

int M = mat.length;

int N = mat[0].length;

boolean[] rowFlag = new boolean[M];

boolean[] colFlag = new boolean[N];

for (int i = 0; i < M; i++) {

for (int j = 0; j < N; j++) {

if (mat[i][j] == 1) {

rowFlag[i] = true;

colFlag[j] = true;

}

}

}

for (int i = 0; i < M; i++) {

for (int j = 0; j < N; j++) {

if (rowFlag[i] || colFlag[j]) {

mat[i][j] = 1;

}

}

}

}

public static void print(int[][] mat) {

for (int i = 0; i < mat.length; i++) {

for (int j = 0; j < mat[i].length; j++) {

System.***out***.print(mat[i][j] + " ");

}

System.***out***.println();

}

}

public static void main(String[] args) {

int[][] mat1 = {{1, 0}, {0, 0}};

System.***out***.println("Modified Matrix 1:");

*modify*(mat1);

*print*(mat1);

int[][] mat2 = {{0, 0, 0}, {0, 0, 1}};

System.***out***.println("Modified Matrix 2:");

*modify*(mat2);

*print*(mat2);

int[][] mat3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};

System.***out***.println("Modified Matrix 3:");

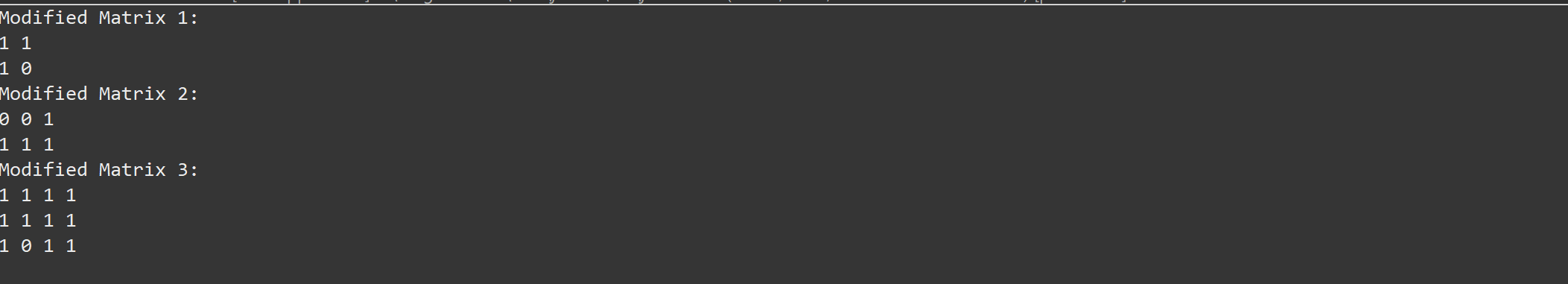
*modify*(mat3);

*print*(mat3);

}

}

Output:



Time Complexity: O(M\*N)

**10. Spiral Matrix**

Code:

package sample1;

public class SpiralMatrix {

public static void printSpiral(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

int top = 0, bottom = m - 1, left = 0, right = n - 1;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++) {

System.***out***.print(matrix[top][i] + " ");

}

top++;

for (int i = top; i <= bottom; i++) {

System.***out***.print(matrix[i][right] + " ");

}

right--;

if (top <= bottom) {

for (int i = right; i >= left; i--) {

System.***out***.print(matrix[bottom][i] + " ");

}

bottom--;

}

if (left <= right) {

for (int i = bottom; i >= top; i--) {

System.***out***.print(matrix[i][left] + " ");

}

left++;

}

}

}

public static void main(String[] args) {

int[][] matrix1 = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

System.***out***.println("Spiral Form of Matrix 1:");

*printSpiral*(matrix1);

System.***out***.println();

int[][] matrix2 = {

{1, 2, 3, 4, 5, 6},

{7, 8, 9, 10, 11, 12},

{13, 14, 15, 16, 17, 18}

};

System.***out***.println("Spiral Form of Matrix 2:");

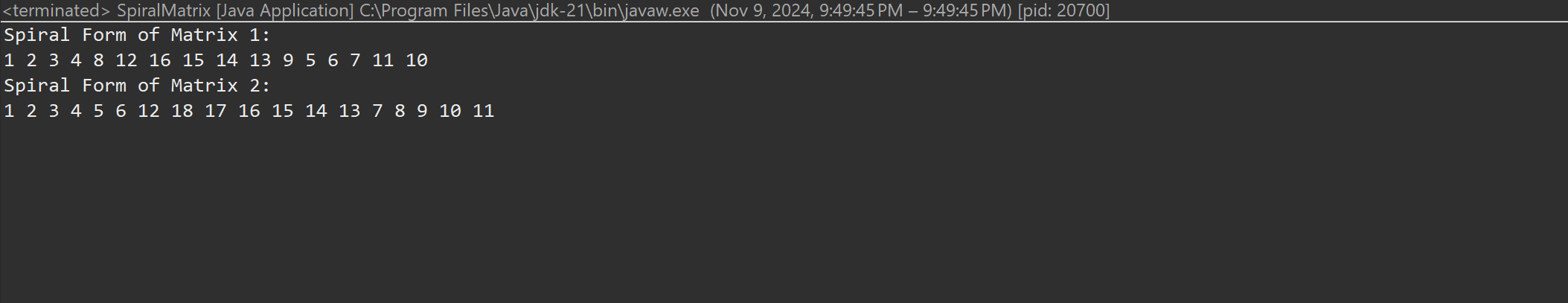
*printSpiral*(matrix2);

System.***out***.println();

}

}

Output:



Time Complexity: O(m\*n)

**13. Check if the given parenthesis string is balanced or not**

Code:

package sample1;

import java.util.\*;

public class Paranthesis {

static String check(String s) {

Stack<Character> st=new Stack<Character>();

for(char i:s.toCharArray()) {

if(i=='(') {

st.push(i);

}

else if(i==')'){

if(!st.isEmpty()) {

st.pop();

}

else {

return "not Balanced";

}

}}

return st.isEmpty()?"Balanced":"not Balanced";

}

public static void main(String[] args) {

String res=*check*("(())");

System.***out***.println(res);

String res1=*check*("“((()))()()");

System.***out***.println(res1);

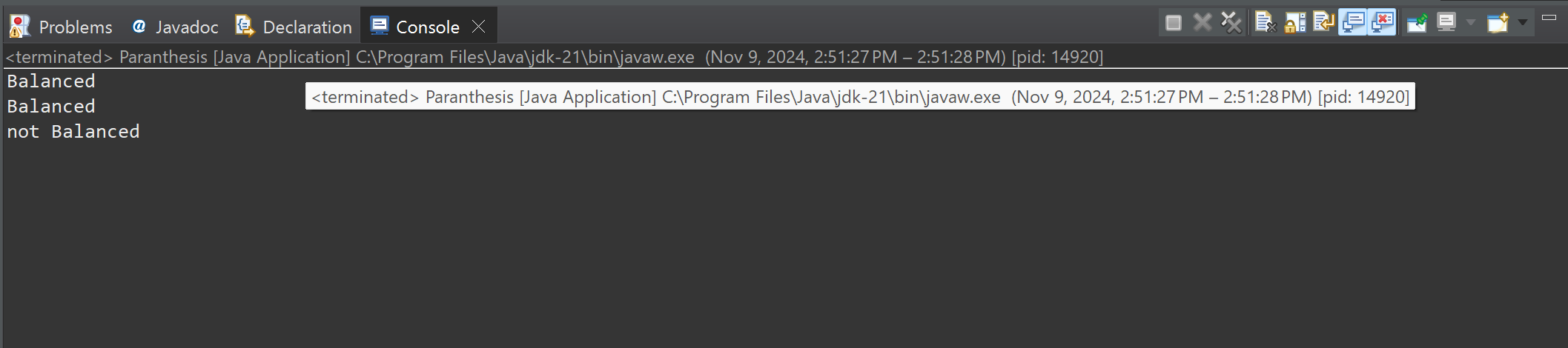
String res2=*check*("())((())”");

System.***out***.println(res2);

}

}

Output:



Time Complexity: O(n)

**14. Two Strings are anagram or not**

Code:

package sample1;

import java.util.\*;

public class ValidAnagram {

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

char[] lst1=s1.toCharArray();

char[] lst2=s2.toCharArray();

if(lst1.length!=lst2.length) {

return false;

}

Arrays.*sort*(lst1);

Arrays.*sort*(lst2);

return Arrays.*equals*(lst1,lst2);

}

public static void main(String[] args) {

String s = "geeks";

String t = "skeeg";

System.***out***.println(*valid*(s,t));

String s1 = "allergy";

String t1 = "allergic";

System.***out***.println(*valid*(s1,t1));

String s2 = "g";

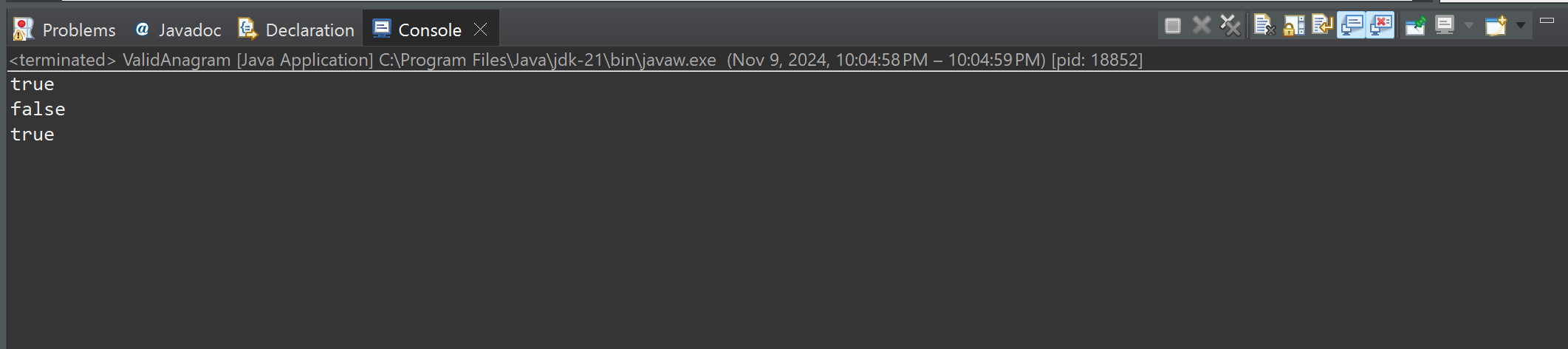
String t2 = "g";

System.***out***.println(*valid*(s2,t2));

}

}

Output:



Time Complexity: O(n log n)

**15. Longest Palindromic substring**

Code:

package sample1;

import java.util.\*;

public class longPalindrome {

public static String longestPalindrome(String str) {

if (str == null || str.length() == 0) {

return "";

}

int n = str.length();

int start = 0;

int maxLength = 1;

boolean[][] dp = new boolean[n][n];

for (int i = 0; i < n; i++) {

dp[i][i] = true;

}

for (int length = 2; length <= n; length++) {

for (int i = 0; i < n - length + 1; i++) {

int j = i + length - 1;

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

if (length == 2 || dp[i + 1][j - 1]) {

dp[i][j] = true;

if (length > maxLength) {

maxLength = length;

start = i;

}

}

}

}

}

return str.substring(start, start + maxLength);

}

public static void main(String[] args) {

String str1 = "forgeeksskeegfor";

System.***out***.println("Longest Palindromic Substring: " + *longestPalindrome*(str1));

String str2 = "Geeks";

System.***out***.println("Longest Palindromic Substring: " + *longestPalindrome*(str2));

String str3 = "abc";

System.***out***.println("Longest Palindromic Substring: " + *longestPalindrome*(str3));

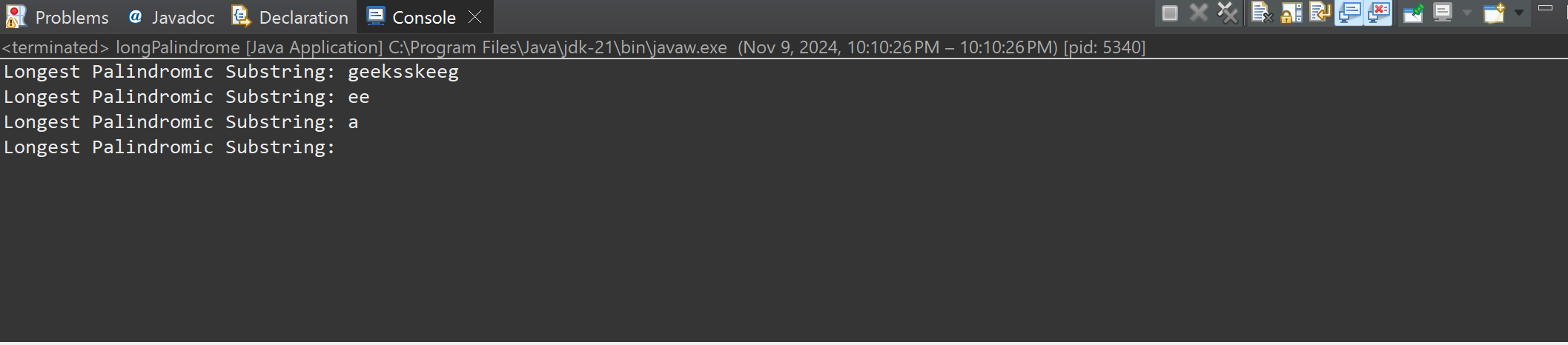
String str4 = "";

System.***out***.println("Longest Palindromic Substring: " + *longestPalindrome*(str4));

}

}

Output:

  
Time Complexity: O(n^2)

**16. Longest Common prefix**

Code:

package sample1;

import java.util.\*;

public class CommonPrefix {

static String Common(String[] arr) {

Arrays.*sort*(arr);

String f=arr[0];

String l=arr[arr.length-1];

int n=Math.*min*(f.length(),l.length());

String res="";

for(int i=0;i<n;i++){

if(f.charAt(i)==l.charAt(i)){

res+=f.charAt(i);

}

else{

break;

}

}

return res.isEmpty()? "-1":res;

}

public static void main(String[] args) {

String[] arr1= {"geeksforgeeks", "geeks", "geek", "geezer"};

System.***out***.println(*Common*(arr1));

String[] arr2= {"hello","world"};

System.***out***.println(*Common*(arr2));

}

}

Output:



Time Complexity: O(n)

**17.Delete Middle element of the stack**

Code:

package sample1;

import java.util.\*;

public class MidDel {

static void deleteMiddle(Stack<Integer> stack, int n, int curr) {

// If the stack is empty or all items are traversed

if (stack.isEmpty() || curr == n) {

return;

}

// Remove the current item

int x = stack.pop();

// Recursively reach the middle element

*deleteMiddle*(stack, n, curr + 1);

// Only push the element back if it's not the middle one

if (curr != n / 2) {

stack.push(x);

}

}

// Wrapper function to start recursion

static void deleteMiddle(Stack<Integer> stack) {

int n = stack.size();

*deleteMiddle*(stack, n, 0);

}

public static void main(String[] args) {

Stack<Integer> stack = new Stack<>();

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

stack.push(5);

System.***out***.println("Original stack: " + stack);

*deleteMiddle*(stack);

System.***out***.println("Stack after deleting middle element: " + stack);

Stack<Integer> stackEven = new Stack<>();

stackEven.push(1);

stackEven.push(2);

stackEven.push(3);

stackEven.push(4);

stackEven.push(5);

stackEven.push(6);

System.***out***.println("Original stack: " + stackEven);

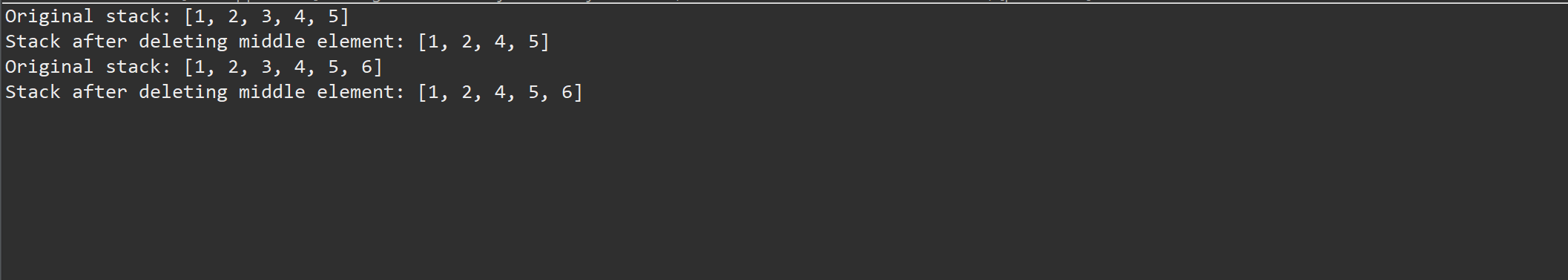
*deleteMiddle*(stackEven);

System.***out***.println("Stack after deleting middle element: " + stackEven);

}

}

Output:



Time Complexity: O(n)

**18. Next Greater Element**

Code:

package sample1;

import java.util.\*;

public class NextGreat {

static void Greater(int[] arr) {

int n = arr.length;

int[] nge = new int[n];

Stack<Integer> stack = new Stack<>();

Arrays.*fill*(nge, -1);

for (int i = n - 1; i >= 0; i--) {

while (!stack.isEmpty() && stack.peek() <= arr[i]) {

stack.pop();

}

if (!stack.isEmpty()) {

nge[i] = stack.peek();

}

stack.push(arr[i]);

}

for (int i = 0; i < n; i++) {

System.***out***.println(arr[i] + " --> " + nge[i]);

}

}

public static void main(String[] args) {

int[] arr1 = {4, 5, 2, 25};

System.***out***.println("Next Greater Elements for the array " + Arrays.*toString*(arr1) + ":");

*Greater*(arr1);

System.***out***.println();

int[] arr2 = {13, 7, 6, 12};

System.***out***.println("Next Greater Elements for the array " + Arrays.*toString*(arr2) + ":");

*Greater*(arr2);

}

}

Output:



Time Complexity: O(n)

**19. Right view of the binary tree**

Code:

package sample1;

import java.util.\*;

class Node {

int val;

Node left, right;

Node(int val) {

this.val = val;

left = right = null;

}

}

public class BinaryTreeRight {

public static List<Integer> rightView(Node root) {

List<Integer> result = new ArrayList<>();

if (root == null) {

return result;

}

Queue<Node> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

int cap = queue.size();

for (int i = 0; i < cap; i++) {

Node curr = queue.poll();

if (i == cap - 1) {

result.add(curr.val);

}

if (curr.left != null) {

queue.add(curr.left);

}

if (curr.right != null) {

queue.add(curr.right);

}

}

}

return result;

}

public static void main(String[] args) {

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

root.left.right = new Node(5);

root.right.right = new Node(4);

List<Integer> rightView = *rightView*(root);

System.***out***.println("Right view of the binary tree: " + rightView);

}

}

Output:



Time complexity: O(n)

**20.Maximum height or depth of the binary tree**

Code:

package sample1;

import java.util.\*;

class Node {

int val;

Node left, right;

Node(int val) {

this.val = val;

this.left = this.right = null;

}

}

public class TreeHeight {

static int maxDepth(Node root) {

if (root == null) {

return 0;

}

int leftDepth = *maxDepth*(root.left);

int rightDepth = *maxDepth*(root.right);

return Math.*max*(leftDepth, rightDepth) + 1;

}

public static void main(String[] args) {

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

root.left.left = new Node(4);

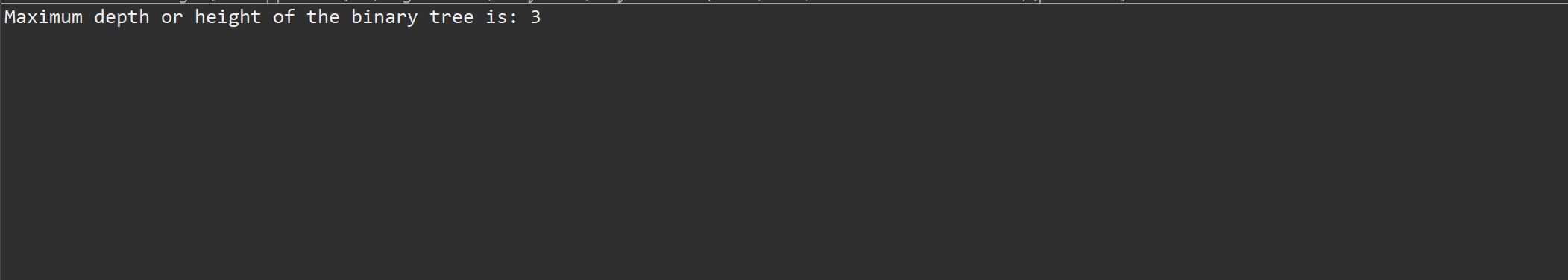
root.left.right = new Node(5);

System.***out***.println("Maximum depth or height of the binary tree is: " + *maxDepth*(root));

}

}

Output:



Time Complexity: O(n)