**Bhuvaneswari P AIDS DSA Practice 1**

**1.Maximum Subarray Sum – Kadane‟s Algorithm:**

**MaximumSubarraySum.java**

import java.util.\*;

class MaximumSubarraySum {

public int maxSubArray(int[] arr) {

int maximum\_value = Integer.MIN\_VALUE;

int sum\_value = 0;

boolean negativeArray = true;

for (int num : arr) {

if (num >= 0) {

negativeArray = false;

break;

}

maximum\_value = Math.max(maximum\_value, num);

}

if (negativeArray) {

return maximum\_value;

}

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

sum\_value = sum\_value + arr[i];

if (sum\_value > 0) {

maximum\_value = Math.max(maximum\_value, sum\_value);

}

if (sum\_value < 0) {

sum\_value = 0;

}

}

return maximum\_value;

}

public static void main(String[] args) {

MaximumSubarraySum obj\_name = new MaximumSubarraySum();

int[] arr1 = {2, 3, -8, 7, -1, 2, 3};

int[] arr2 = {-2, -4};

int[] arr3 = {5, 4, 1, 7, 8};

System.out.println("Maximum Subarray Sum of Array1 is " + obj\_name.maxSubArray(arr1));

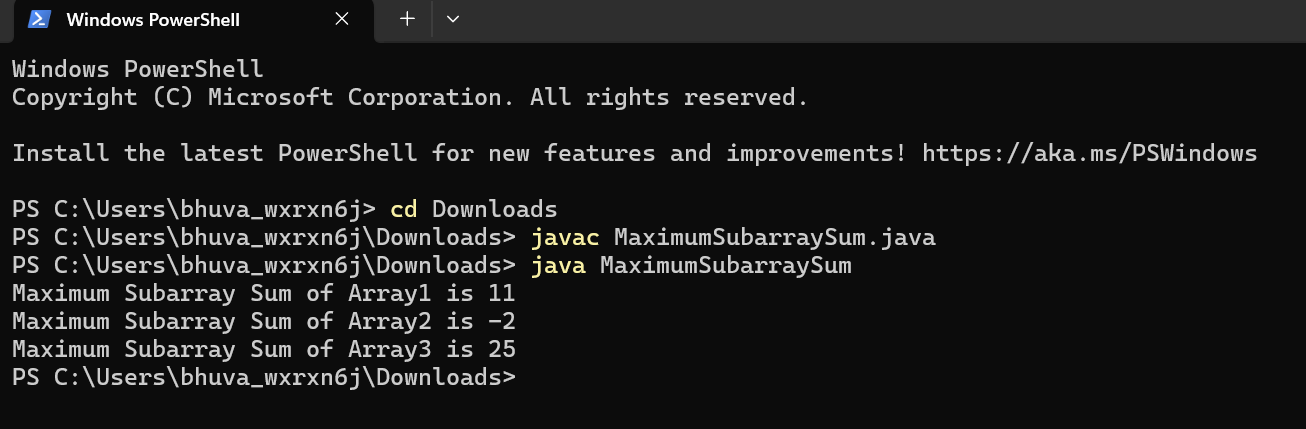
System.out.println("Maximum Subarray Sum of Array2 is " + obj\_name.maxSubArray(arr2));

System.out.println("Maximum Subarray Sum of Array3 is " + obj\_name.maxSubArray(arr3));

}

}

**Output :**



Time Complexity : O(n)

Space Complexity : O(1)

**2. Maximum Product Subarray**

**Maximum Product Subarray.java**

import java.util.\*;

class MaximumProductSubarray {

public int maxProduct(int[] nums) {

int maximum\_value = nums[0];

int sum\_value = 1;

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

sum\_value = sum\_value \* nums[i];

maximum\_value = Math.max(maximum\_value, sum\_value);

if (sum\_value == 0) {

sum\_value = 1;

}

}

sum\_value = 1;

for (int i = nums.length - 1; i >= 0; i--) {

sum\_value = sum\_value \* nums[i];

maximum\_value = Math.max(maximum\_value, sum\_value);

if (sum\_value == 0) {

sum\_value = 1;

}

}

return maximum\_value;

}

public static void main(String[] args) {

MaximumProductSubarray obj = new MaximumProductSubarray();

int[] arr1 = {-2, 6, -3, -10, 0, 2};

int[] arr2 = {-1, -3, -10, 0, 60};

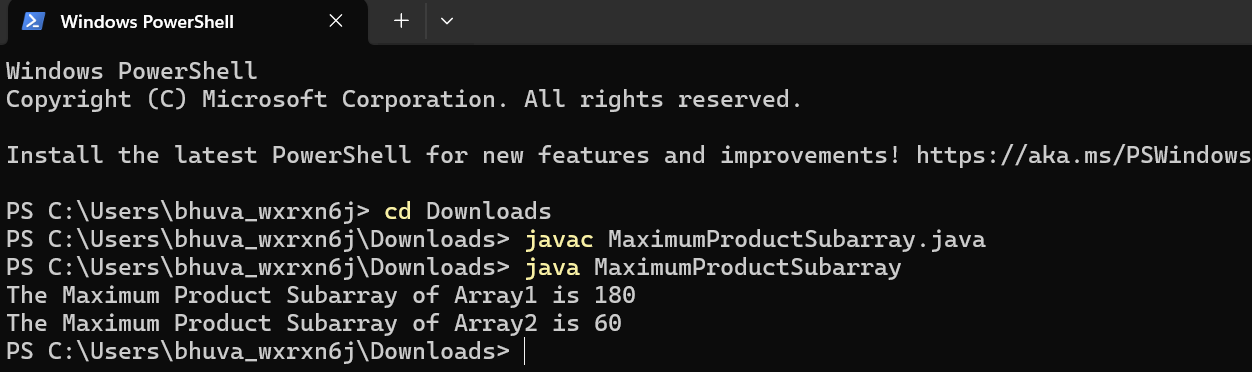
System.out.println("The Maximum Product Subarray of Array1 is " + obj.maxProduct(arr1));

System.out.println("The Maximum Product Subarray of Array2 is " + obj.maxProduct(arr2));

}

}

**Output :**

****

Time Complexity : O(n)

Space Complexity : O(1)

**3.Search in a sorted and rotated Array**

**RotatedSortedArray.java**

class RotatedSortedArray {

public int search(int[] nums, int target) {

int low = 0, high = nums.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (nums[mid] == target) {

return mid;

}

if (nums[low] <= nums[mid]) {

if (nums[low] <= target && target < nums[mid]) {

high = mid - 1;

} else {

low = mid + 1;

}

} else {

if (nums[mid] < target && target <= nums[high]) {

low = mid + 1;

} else {

high = mid - 1;

}

}

}

return -1;

}

public static void main(String[] args) {

RotatedSortedArray obj = new RotatedSortedArray();

int[] arr1 = {4, 5, 6, 7, 0, 1, 2};

int key1 = 0;

int[] arr2 = {4, 5, 6, 7, 0, 1, 2};

int key2 = 3;

int[] arr3 = {50, 10, 20, 30, 40};

int key3 = 10;

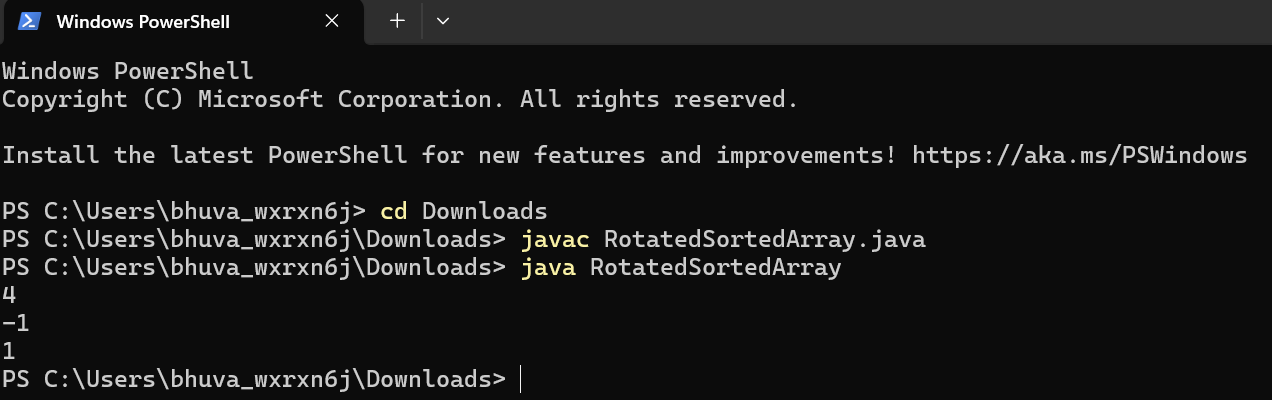
System.out.println(obj.search(arr1, key1));

System.out.println(obj.search(arr2, key2));

System.out.println(obj.search(arr3, key3));

}

}



Time Complexity : O(log n)

Space Complexity : O(1)

**4. Container With Most Water**

**ContainerWithMostWater.java**

class ContainerWithMostWater {

public int maxArea(int[] height) {

int left\_pointer = 0;

int right\_pointer = height.length - 1;

int max\_area = 0;

int area = 0;

int width = 0;

while (left\_pointer < right\_pointer) {

width = right\_pointer - left\_pointer;

area = Math.min(height[right\_pointer], height[left\_pointer]) \* width;

max\_area = Math.max(area, max\_area);

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

left\_pointer++;

} else {

right\_pointer--;

}

}

return max\_area;

}

public static void main(String[] args) {

ContainerWithMostWater obj = new ContainerWithMostWater();

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

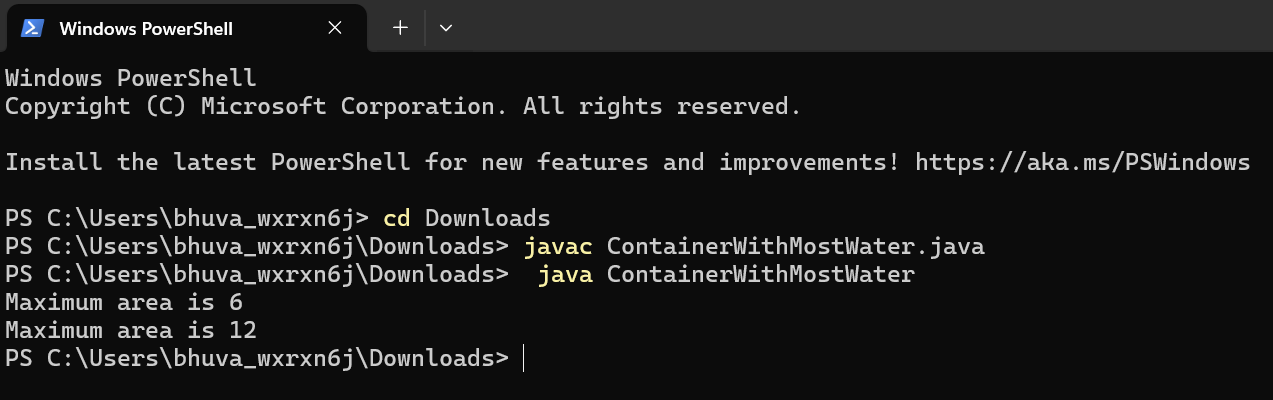
int[] arr2 = {3, 1, 2, 4, 5};

System.out.println("Maximum area is " + obj.maxArea(arr1));

System.out.println("Maximum area is " + obj.maxArea(arr2));

}

}



Time Complexity : O(n)

Space Complexity : O(1)

**5.Factorial of a Large Integer**

**FactorialOfLargeInteger.java**

import java.math.BigInteger;

class FactorialOfLargeInteger {

public static BigInteger factorial(int n) {

BigInteger result = BigInteger.ONE;

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

result = result.multiply(BigInteger.valueOf(i));

}

return result;

}

public static void main(String[] args) {

int num1 = 100;

int num2 = 50;

System.out.println("Factorial of " + num1 + " is:");

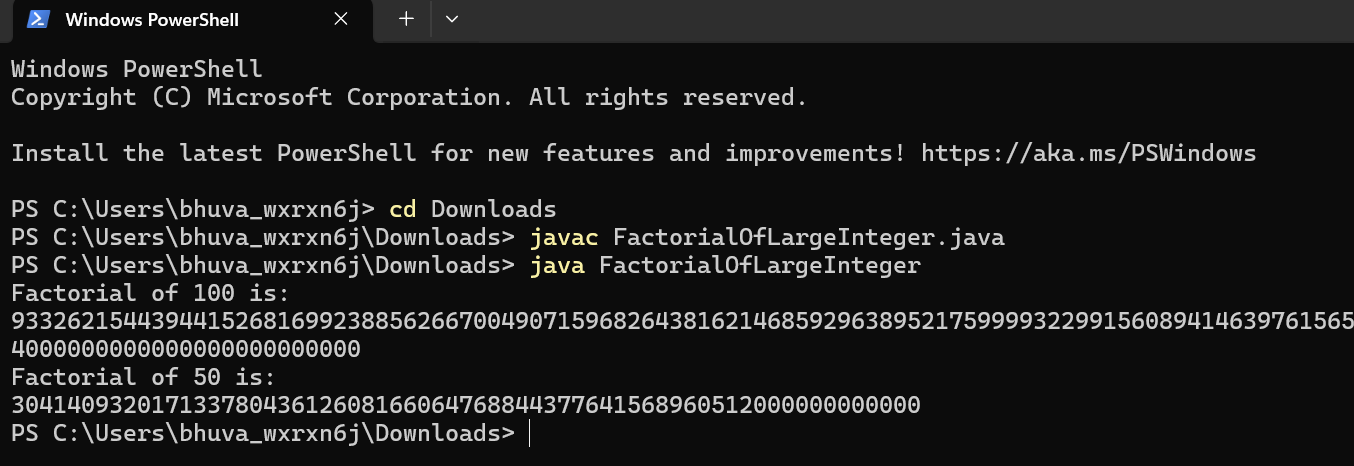
System.out.println(factorial(num1));

System.out.println("Factorial of " + num2 + " is:");

System.out.println(factorial(num2));

}

}



Time Complexity : O(n)

Space Complexity : O(1)

**6.Trapping Rain Water**

**TrappingRainWater.java**

import java.util.\*;

public class TrappingRainWater {

public static int trap(int[] arr) {

int n = arr.length;

if (n == 0) return 0;

int[] leftMax = new int[n];

int[] rightMax = new int[n];

leftMax[0] = arr[0];

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

leftMax[i] = Math.max(arr[i], leftMax[i - 1]);

}

rightMax[n - 1] = arr[n - 1];

for (int i = n - 2; i >= 0; i--) {

rightMax[i] = Math.max(arr[i], rightMax[i + 1]);

}

int totalWater = 0;

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

totalWater += Math.min(leftMax[i], rightMax[i]) - arr[i];

}

return totalWater;

}

public static void main(String[] args) {

TrappingRainWater obj=new TrappingRainWater();

int[] arr1 = {3, 0, 1, 0, 4, 0, 2};

int[] arr2 = {3, 0, 2, 0, 4};

int[] arr3 = {1, 2, 3, 4};

int[] arr4 = {10, 9, 0, 5};

System.out.println("Water trapped for arr1: " + obj.trap(arr1));

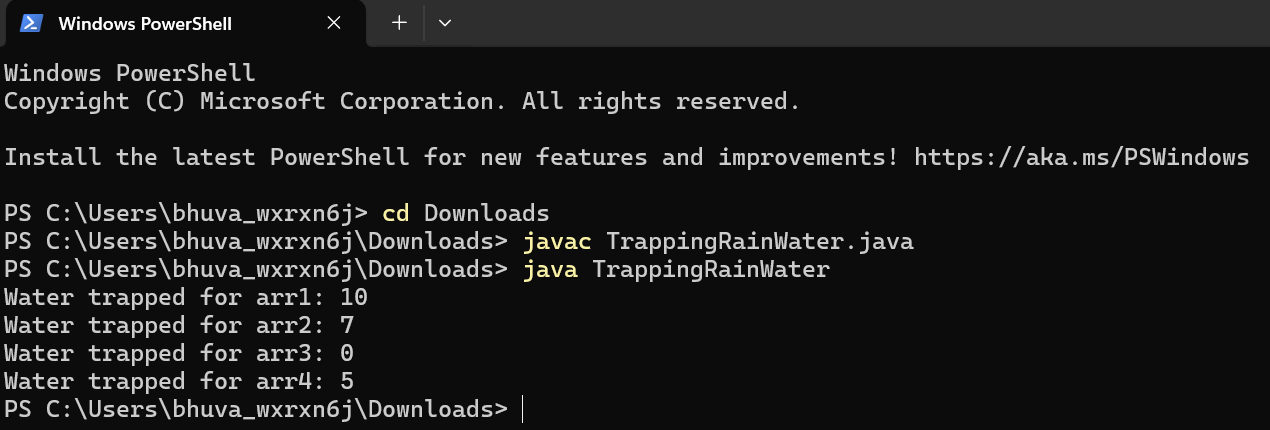
System.out.println("Water trapped for arr2: " + obj.trap(arr2));

System.out.println("Water trapped for arr3: " + obj.trap(arr3));

System.out.println("Water trapped for arr4: " + obj.trap(arr4));

}

}**Output :**



Time Complexity : O(n)

Space Complexity : O(n)

**7. Chocolate Distribution Problem**

**ChocolateDistribution.java**

import java.util.Arrays;

class ChocolateDistribution{

public int chocoval(int[] arr, int m){

int n = arr.length;

int min\_value = Integer.MAX\_VALUE;

int balance = n - m;

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

int[] sliding\_window = Arrays.copyOfRange(arr, i, m + i);

int a = Arrays.stream(sliding\_window).max().getAsInt();

int b = Arrays.stream(sliding\_window).min().getAsInt();

min\_value = Math.min(min\_value, a - b);

}

return min\_value;

}

public static void main (String[] args){

ChocolateDistribution obj = new ChocolateDistribution();

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

int m1 = 3;

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

int m2 = 5;

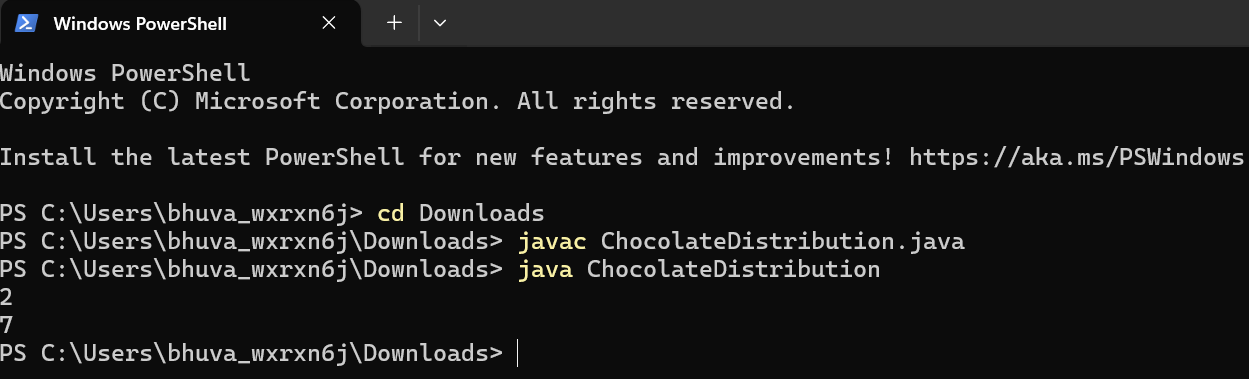
System.out.println(obj.chocoval(arr1, m1));

System.out.println(obj.chocoval(arr2, m2));

}

}

**Output:**



Time Complexity : O(n\*m) and o(n^2)

Space Complexity :O(1)

**8.Merge Overlapping Intervals**

**MergeIntervals.java**

import java.util.\*;

public class MergeIntervals {

public static List<int[]> mergeIntervals(int[][] intervals) {

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

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

for (int[] interval : intervals) {

if (merged.isEmpty() || merged.get(merged.size() - 1)[1] < interval[0]) {

merged.add(interval);

} else {

merged.get(merged.size() - 1)[1] = Math.max(merged.get(merged.size() - 1)[1], interval[1]);

}

}

return merged;

}

public static void main(String[] args) {

MergeIntervals obj=new MergeIntervals();

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

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

List<int[]> result1 = mergeIntervals(intervals1);

List<int[]> result2 = mergeIntervals(intervals2);

System.out.println("Merged Intervals 1: " + obj.formatIntervals(result1));

System.out.println("Merged Intervals 2: " + obj.formatIntervals(result2));

}

public static String formatIntervals(List<int[]> intervals) {

StringBuilder sb = new StringBuilder();

for (int[] interval : intervals) {

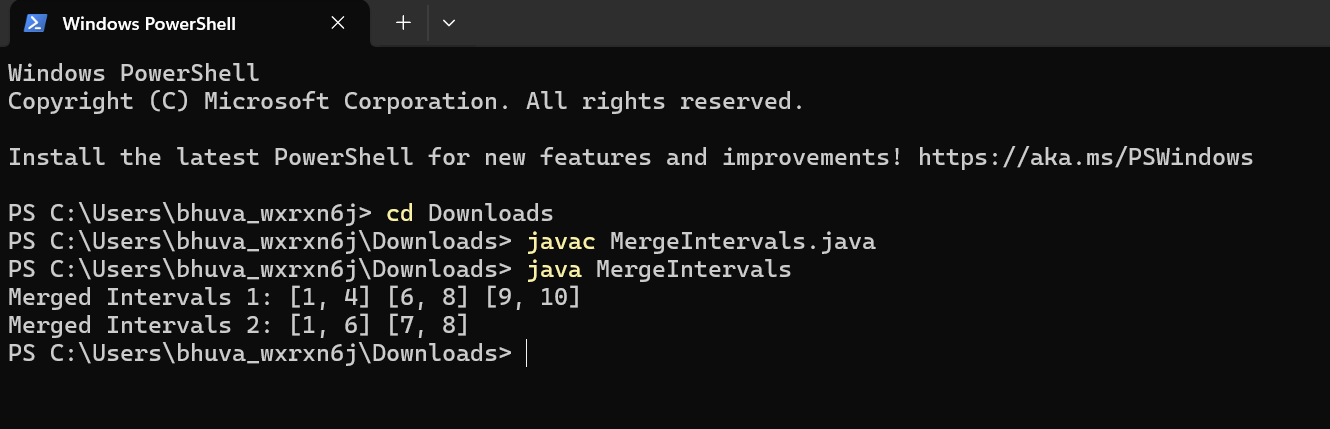
sb.append(Arrays.toString(interval)).append(" ");

}

return sb.toString().trim();

}

}**Output:**



Time Complexity: O(n log n)

Space Complexity : O(n)

**9. Boolean Matrix**

**BooleanMatrix.java**

import java.util.\*;

public class BooleanMatrix {

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

int M = mat.length;

int N = mat[0].length;

boolean[] rowFlags = new boolean[M];

boolean[] colFlags = new boolean[N];

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

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

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

rowFlags[i] = true;

colFlags[j] = true;

}

}

}

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

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

if (rowFlags[i] || colFlags[j]) {

mat[i][j] = 1;

}

}

}

}

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

for (int[] row : mat) {

for (int element : row) {

System.out.print(element + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

BooleanMatrix obj=new BooleanMatrix();

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

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

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

System.out.println("Original Matrix 1:");

printMatrix(mat1);

modifyMatrix(mat1);

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

obj.printMatrix(mat1);

System.out.println("Original Matrix 2:");

printMatrix(mat2);

modifyMatrix(mat2);

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

obj.printMatrix(mat2);

System.out.println("Original Matrix 3:");

printMatrix(mat3);

modifyMatrix(mat3);

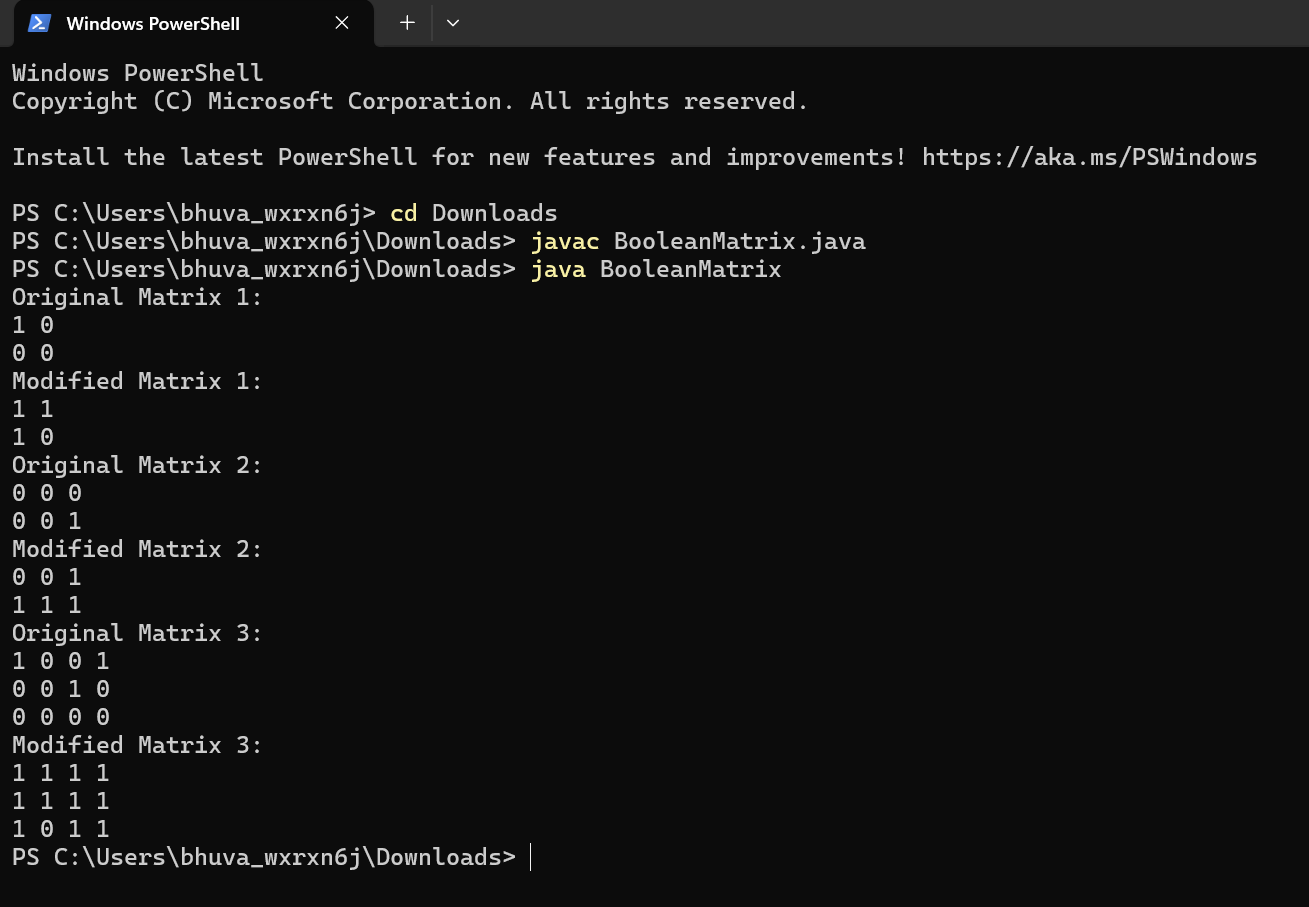
System.out.println("Modified Matrix 3:");

obj.printMatrix(mat3);

}

}

Output :



Time Complexity : O(m\*n)

Space Complexity : O(m+n)

**10. Spiral Matrix**

**SpiralMatrix.java**

import java.util.\*;

public class SpiralMatrix {

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

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

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

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

}

System.out.println();

}

}

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

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

return;

}

int top = 0, bottom = matrix.length - 1;

int left = 0, right = matrix[0].length - 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) {

SpiralMatrix obj = new SpiralMatrix();

int[][] matrix1 = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

int[][] matrix2 = {

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

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

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

};

System.out.println("Original Matrix 1:");

printMatrix(matrix1);

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

obj.printSpiral(matrix1);

System.out.println();

System.out.println("Original Matrix 2:");

printMatrix(matrix2);

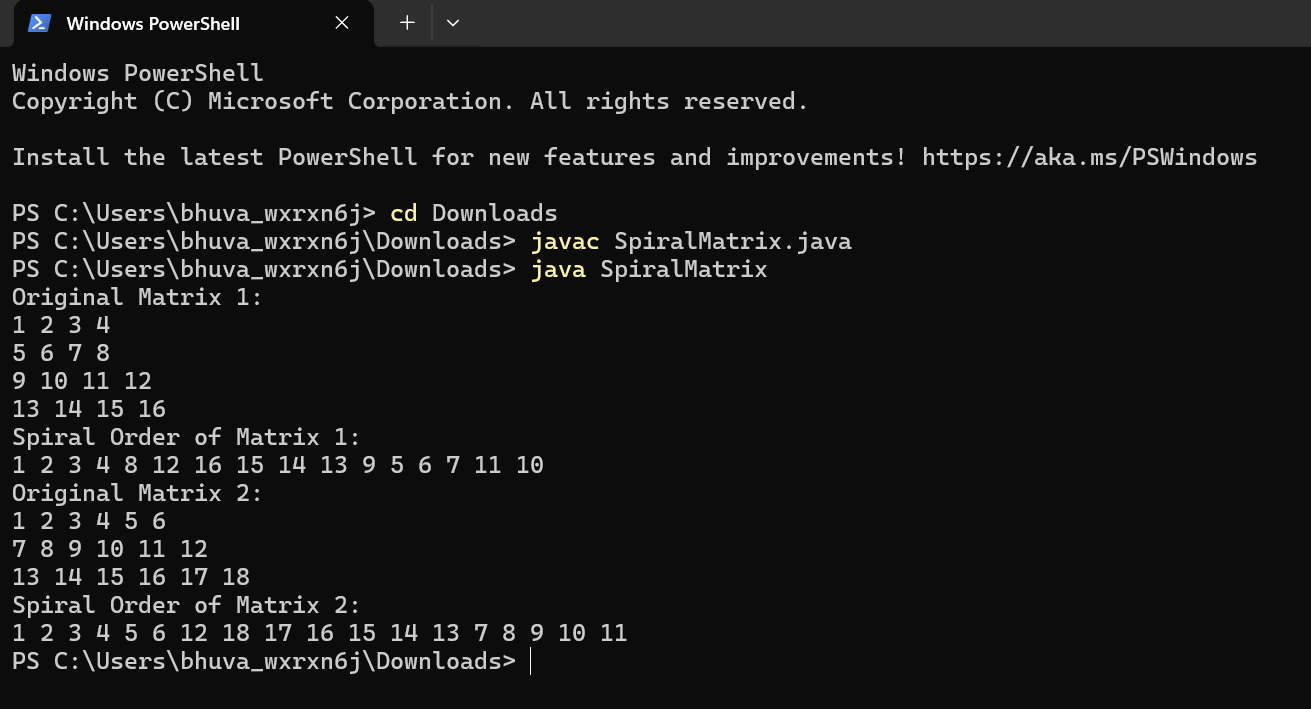
System.out.println("Spiral Order of Matrix 2:");

obj.printSpiral(matrix2);

}

}

**Output:**



Time Complexity : O(m\*n)

Space Complexity: O(1)

**13. Check if given Parentheses expression is balanced or not**

**Parentheses.java**

import java.util.\*;

class Parentheses{

public String stacker(String stack){

int left=0;

int right=0;

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

if (stack.charAt(i)=='('){

left+=1;

}

else{

right+=1;

}

if (left<right){

return "NotBalanced";

}

}

if (left==right){

return "Balanced";

}

return "NotBalanced";

}

public static void main(String[] args){

Parentheses obj = new Parentheses();

String arr1="((()))()()";

System.out.println(obj.stacker(arr1));

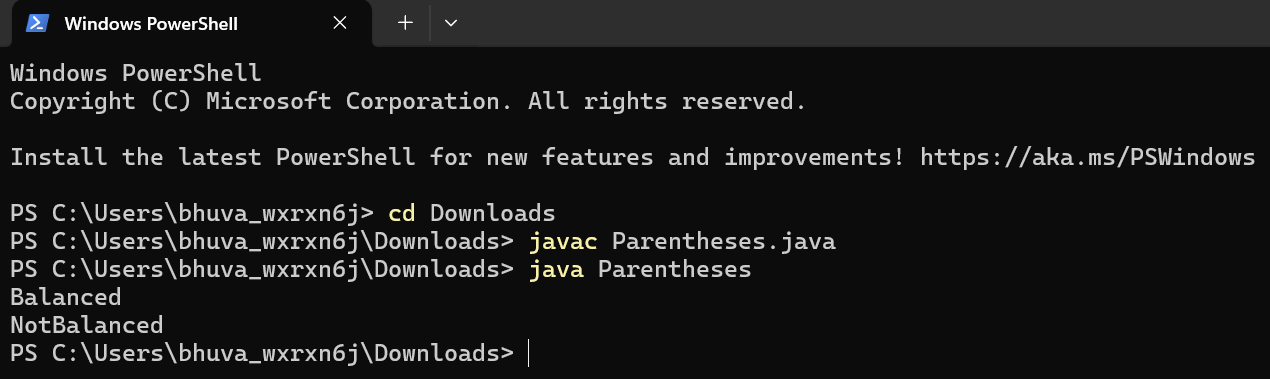
String arr2="())((())";

System.out.println(obj.stacker(arr2));

}

}

**Output:**



Time Complexity : O(n)

Space Complexity : O(1)

**14.** **Check if two Strings are Anagrams of each other**

**Anagram.java**

import java.util.HashMap;

public class Anagram {

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

if (s1.length() != s2.length()) {

return false;

}

HashMap<Character, Integer> map1 = new HashMap<>();

HashMap<Character, Integer> map2 = new HashMap<>();

for (char c : s1.toCharArray()) {

map1.put(c, map1.getOrDefault(c, 0) + 1);

}

for (char c : s2.toCharArray()) {

map2.put(c, map2.getOrDefault(c, 0) + 1);

}

return map1.equals(map2);

}

public static void main(String[] args) {

Anagram obj=new Anagram();

String s1 = "geeks";

String s2 = "kseeg";

System.out.println(obj.validAnagrams(s1, s2));

s1 = "allergy";

s2 = "allergic";

System.out.println(obj.validAnagrams(s1, s2));

s1 = "g";

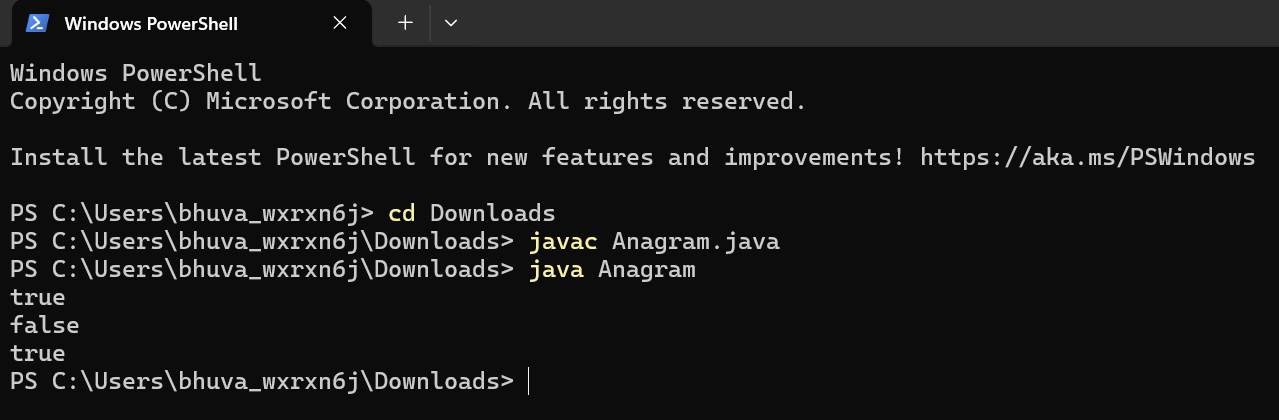
s2 = "g";

System.out.println(obj.validAnagrams(s1, s2));

}

}

**Output:**

****

Time Complexity:O(n)

Space Complexity : O(1)

**15. Longest Palindromic Substring**

**LongestPalindrome.java**

import java.util.\*;

public class LongestPalindrome {

public static String longestPalindrome(String s) {

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

return "";

}

int start = 0, maxLength = 1;

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

int len1 = expandAroundCenter(s, i, i);

int len2 = expandAroundCenter(s, i, i + 1);

int len = Math.max(len1, len2);

if (len > maxLength) {

maxLength = len;

start = i - (maxLength - 1) / 2;

}

}

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

}

private static int expandAroundCenter(String s, int left, int right) {

while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {

left--;

right++;

}

return right - left - 1;

}

public static void main(String[] args) {

LongestPalindrome obj=new LongestPalindrome();

String input = "forgeeksskeegfor";

String result = obj.longestPalindrome(input);

System.out.println("Original String: " + input);

System.out.println("Longest Palindromic Substring: " + result);

input = "Geeks";

result = obj.longestPalindrome(input);

System.out.println("Original String: " + input);

System.out.println("Longest Palindromic Substring: " + result);

input = "abc";

result = obj.longestPalindrome(input);

System.out.println("Original String: " + input);

System.out.println("Longest Palindromic Substring: " + result);

input = "";

result = obj.longestPalindrome(input);

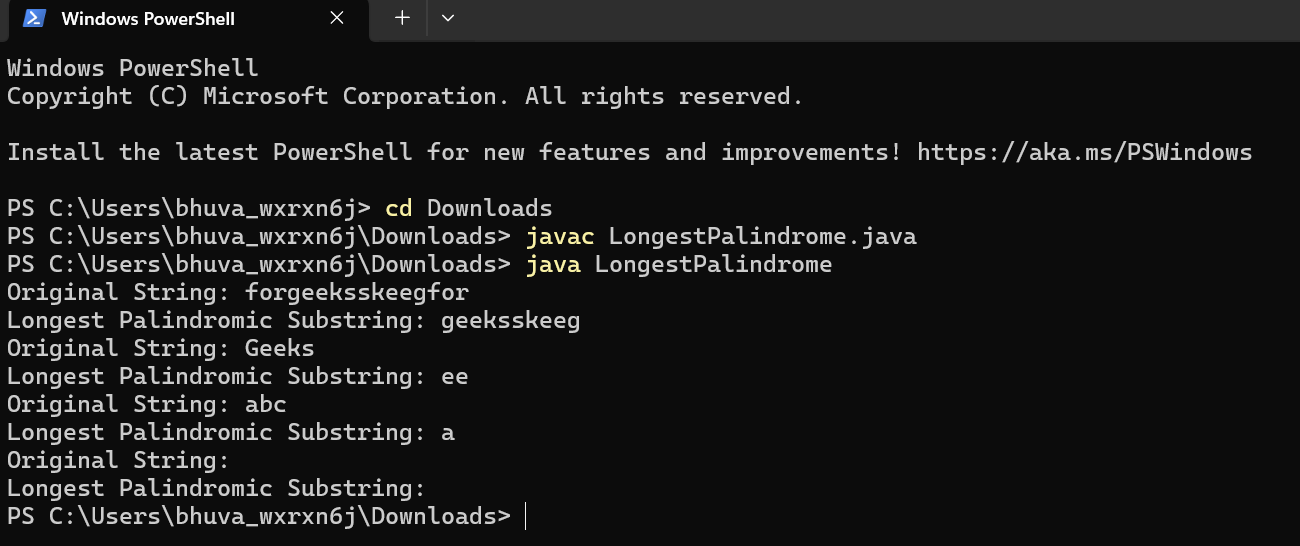
System.out.println("Original String: " + input);

System.out.println("Longest Palindromic Substring: " + result);

}

}

**Output:**



Time Complexity:O(n^2)

Space Complexity : O(1)

**16. Longest Common Prefix using Sorting**

import java.util.Arrays;

public class LongestCommonPrefix {

public static String longestCommonPrefix(String[] arr) {

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

return "-1";

}

Arrays.sort(arr);

String first = arr[0];

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

int i = 0;

while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {

i++;

}

if (i == 0) {

return "-1";

}

return first.substring(0, i);

}

public static void main(String[] args) {

LongestCommonPrefix obj = new LongestCommonPrefix();

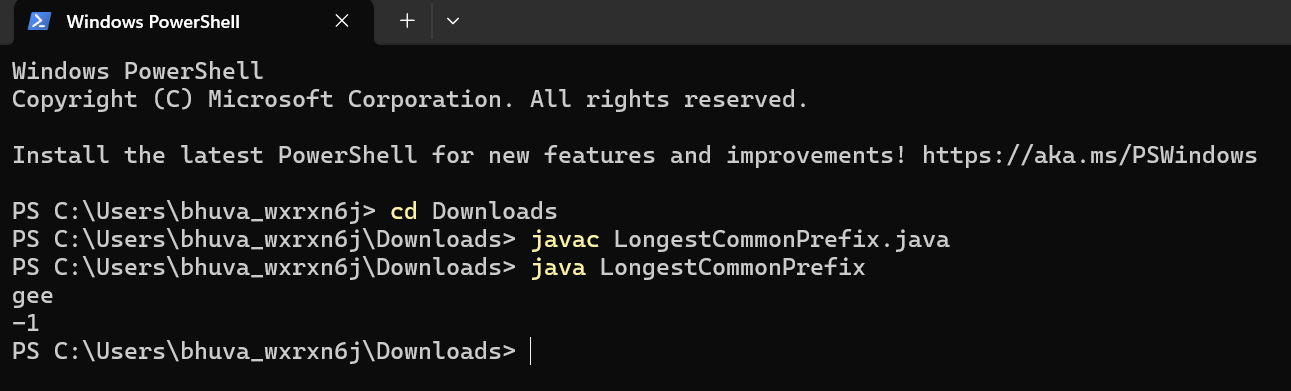
String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};

System.out.println(obj.longestCommonPrefix(arr1));

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

System.out.println(obj.longestCommonPrefix(arr2));

}

}

Time Complexity: O(n \* m \* log n)

Space Complexity : O(n)

**17. Delete middle element of a stack**

**DeleteElement.java**

import java.util.Stack;

public class DeleteMiddleElement {

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

if (stack.isEmpty() || n <= 0) {

return;

}

int mid = n / 2;

deleteMiddleHelper(stack, mid);

}

private static void deleteMiddleHelper(Stack<Integer> stack, int mid) {

if (mid == 0) {

stack.pop();

return;

}

int top = stack.pop();

deleteMiddleHelper(stack, mid - 1);

stack.push(top);

}

public static void main(String[] args) {

Stack<Integer> stack1 = new Stack<>();

stack1.push(1);

stack1.push(2);

stack1.push(3);

stack1.push(4);

stack1.push(5);

System.out.println("Original Stack 1: " + stack1);

deleteMiddle(stack1, stack1.size());

System.out.println("Modified Stack 1: " + stack1);

Stack<Integer> stack2 = new Stack<>();

stack2.push(1);

stack2.push(2);

stack2.push(3);

stack2.push(4);

stack2.push(5);

stack2.push(6);

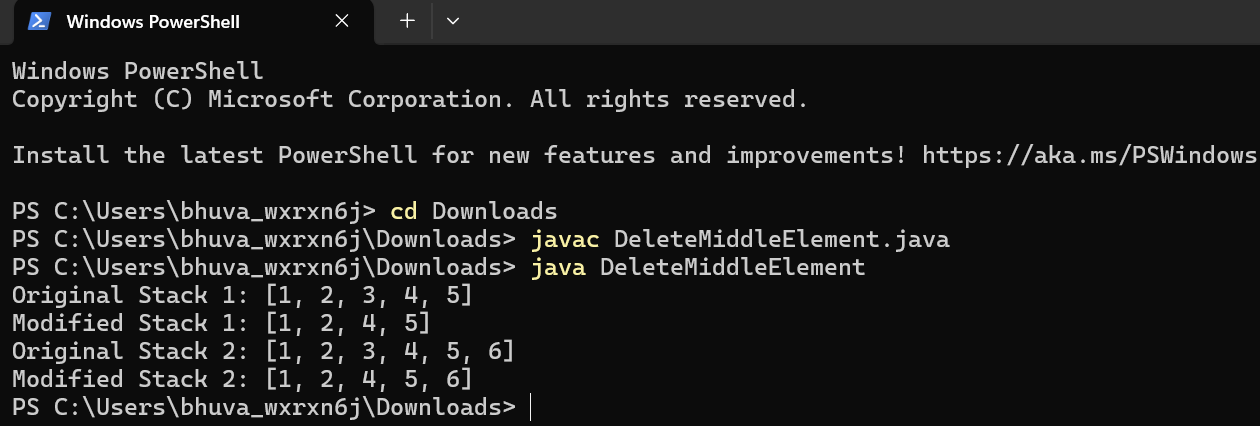
System.out.println("Original Stack 2: " + stack2);

deleteMiddle(stack2, stack2.size());

System.out.println("Modified Stack 2: " + stack2);

}

}

**Output:** ****

Time Complexity :O(n)

Space Complexity : O(n)

**18. Next Greater Element (NGE)**

**NextGreaterElement.java**

class NextGreaterElement {

public void printNextGreaterElement(int[] arr) {

int n = arr.length;

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

int nextGreater = -1;

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

if (arr[j] > arr[i]) {

nextGreater = arr[j];

break;

}

}

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

}

}

public static void main(String[] args) {

NextGreaterElement obj = new NextGreaterElement();

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

System.out.println("Next Greater Elements for arr1:");

obj.printNextGreaterElement(arr1);

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

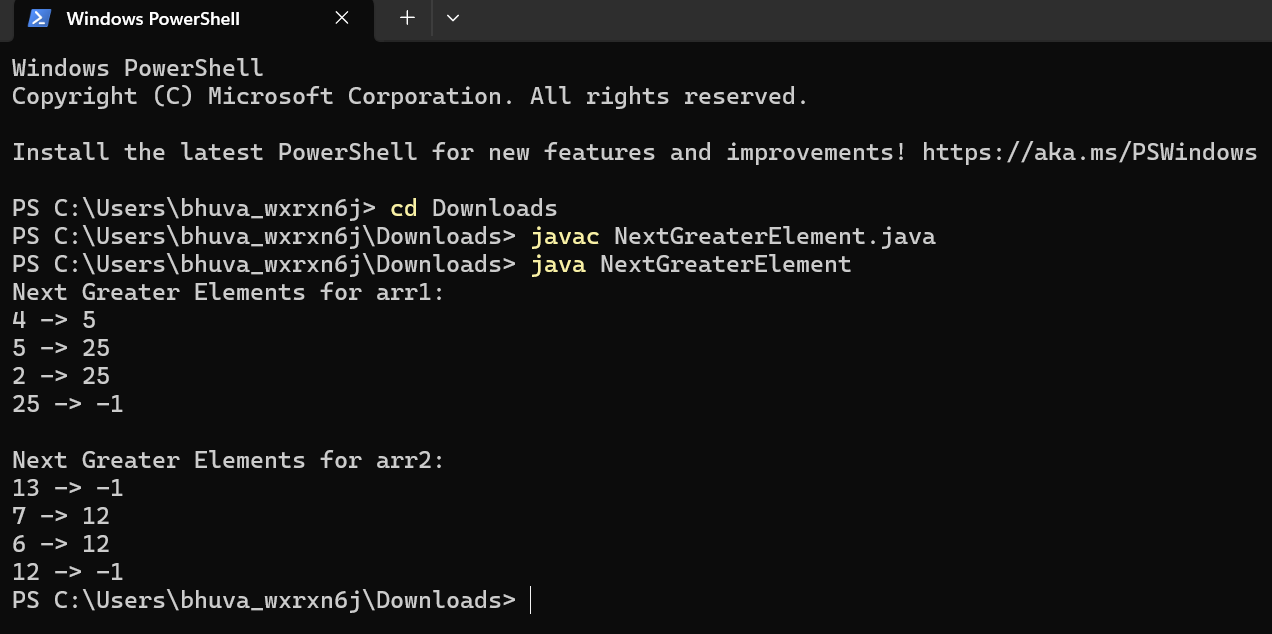
System.out.println("\nNext Greater Elements for arr2:");

obj.printNextGreaterElement(arr2);

}

}

**Output:**

****

Time Complexity : O(n^2)

Space Complexity: O(1)

**19.Right View Of Binary Tree**

import java.util.\*;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

public class RightViewOfBinaryTree {

Node root;

void rightView(Node root) {

if (root == null) return;

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

queue.add(root);

while (!queue.isEmpty()) {

int n = queue.size();

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

Node node = queue.poll();

if (i == n)

System.out.print(node.data + " ");

if (node.left != null)

queue.add(node.left);

if (node.right != null)

queue.add(node.right);

}

}

}

public static void main(String[] args) {

RightViewOfBinaryTree tree = new RightViewOfBinaryTree();

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

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

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

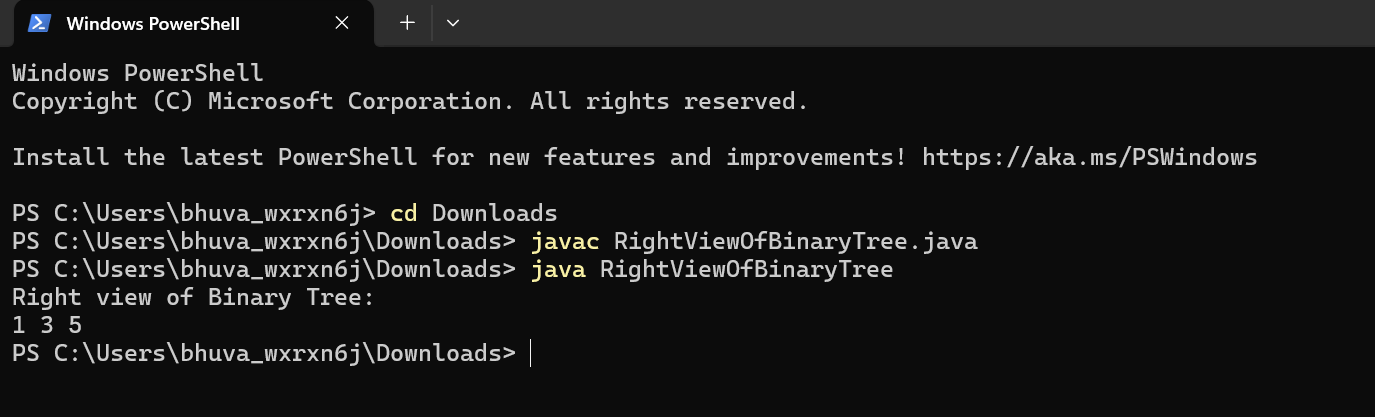
System.out.println("Right view of Binary Tree: ");

tree.rightView(tree.root);

}

}

**Output :**



Time Complexity :O(n)

Space complexity :O(n)

**20.Depth Of a Binary Tree**

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

public class DepthOfBinaryTree {

Node root;

int maxDepth(Node node) {

if (node == null)

return 0;

int leftDepth = maxDepth(node.left);

int rightDepth = maxDepth(node.right);

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

}

public static void main(String[] args) {

DepthOfBinaryTree tree = new DepthOfBinaryTree();

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

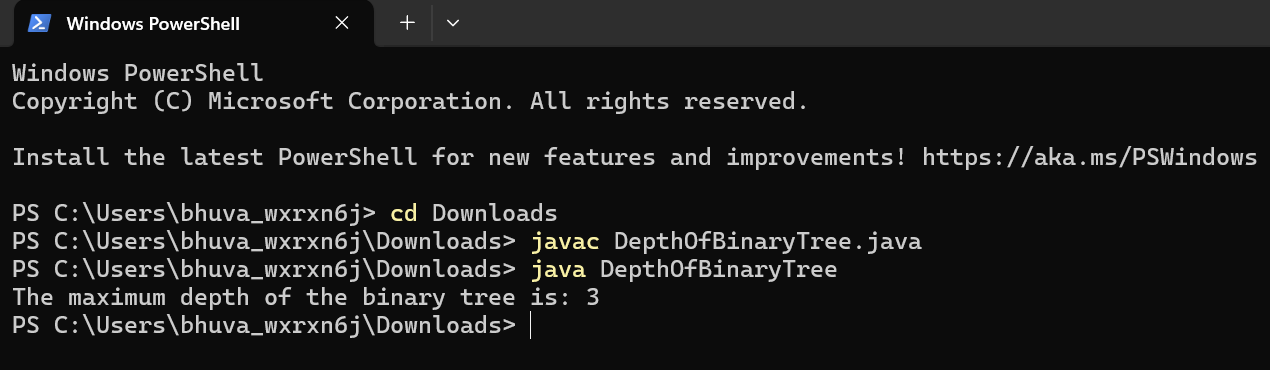
tree.root.left.left = new Node(4);

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

System.out.println("The maximum depth of the binary tree is: " + tree.maxDepth(tree.root));

}

}

**Output:** 

Time Complexity :O(n)

Space Complexity :O(n)