Coding practice Problems:

1. Maximum Subarray Sum – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum

Code:

import java.util.\*;

public class Solution{

public static void main(String args[]){

Scanner scanner = new Scanner (System.in);

System.out.print("Enter the array size");

int n=scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the array elements");

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

nums[i]=scanner.nextInt();

}

int s=nums[0];

int m=nums[0];

for(int j=1;j<nums.length;j++){

s=Math.max(nums[j],s+nums[j]);

m=Math.max(m,s);

}

System.out.println( m);

}

}

Output:

A computer screen with white text

Description automatically generated

Time complexity: O(n)

1. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.

Code:

import java.util.\*;

public class Solution {

public int maxProduct(int[] nums) {

if (nums.length == 0) return 0;

int maxProduct = nums[0];

int minProduct = nums[0];

int globalMax = nums[0];

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

int current = nums[i];

int tempMax = maxProduct;

maxProduct = Math.max(current, Math.max(maxProduct \* current, minProduct \* current));

minProduct = Math.min(current, Math.min(tempMax \* current, minProduct \* current));

globalMax = Math.max(globalMax, maxProduct);

}

return globalMax;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Solution solution = new Solution();

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

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

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

nums[i] = scanner.nextInt();

}

int result = solution.maxProduct(nums);

System.out.println("Max Subarray Product: " + result);

scanner.close();

}

}

Output:

A black screen with white text

Description automatically generated

Time complexity: O(n)

1. Search in a sorted and rotated Array Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.

Code:

import java.util.\*;

public class Search{

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Search search=new Search();

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

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

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

nums[i] = scanner.nextInt();

}

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

int tar=scanner.nextInt();

int result = search.indexfind(nums,tar);

System.out.println("Output: " + result);

scanner.close();

}

public int indexfind(int[] nums,int tar) {

int l=0;

int h=nums.length-1;

while(l<=h){

int mid=(l+h)/2;

if(nums[mid]==tar){

System.out.println(mid);

}

if(nums[l]<nums[mid]){

if(nums[l]<=tar && tar<nums[mid]){

h=mid-1;

}

else{

l=mid+1;

}

}

else{

if(nums[mid]<=tar && tar<=nums[h]){

l=mid+1;

}

else{

h=mid-1;

}

}

}

return -1;

}

}

Output:

A black screen with white text

Description automatically generated

Time complexity: O(log n)

1. Container with Most Water

Code:

import java.util.Scanner;

class Bool {

public int maxArea(int[] height) {

int left = 0;

int right = height.length - 1;

int maxArea = 0;

while (left < right) {

int currentArea = Math.min(height[left], height[right]) \* (right - left);

maxArea = Math.max(maxArea, currentArea);

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

left++;

} else {

right--;

}

}

return maxArea;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of elements:");

int n = scanner.nextInt();

int[] height = new int[n];

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

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

height[i] = scanner.nextInt();

}

Bool b = new Bool();

int maxArea = b.maxArea(height);

System.out.println("The maximum area is: " + maxArea);

scanner.close();

}

}

Output:

A computer screen shot of a black screen

Description automatically generated

Time complexity: O(n)

1. Find the Factorial of a large number

Code:

import java.util.\*;

import java.math.BigInteger;

public class Fac{

public static void main(String arg[]){

Scanner sc=new Scanner(System.in);

System.out.println("enter:");

int n=sc.nextInt();

BigInteger res=BigInteger.ONE;

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

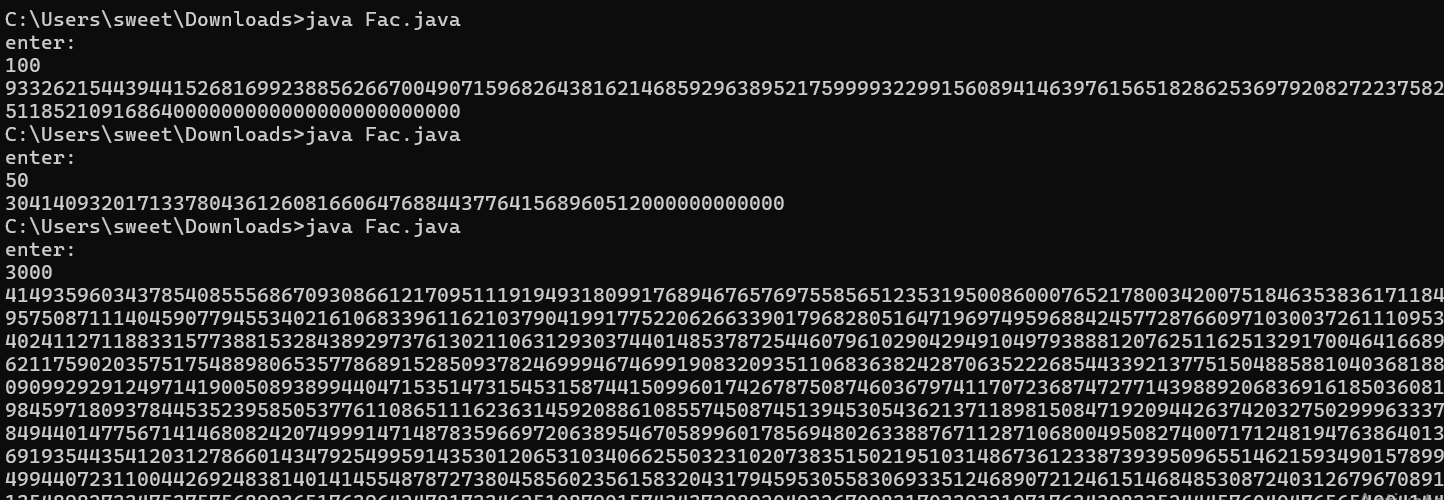
res=res.multiply(BigInteger.valueOf(i));

}

System.out.print(res);

}

}



Time complexity: O(n)

1. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

Code:

import java.util.Scanner;

class Solution {

public int trapRainWater(int[] height) {

int left = 0, right = height.length - 1;

int leftMax = 0, rightMax = 0;

int trappedWater = 0;

while (left < right) {

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

if (height[left] >= leftMax) {

leftMax = height[left];

} else {

trappedWater += leftMax - height[left];

}

left++;

} else {

if (height[right] >= rightMax) {

rightMax = height[right];

} else {

trappedWater += rightMax - height[right];

}

right--;

}

}

return trappedWater;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of elements:");

int n = scanner.nextInt();

int[] heights = new int[n];

System.out.println("Enter the heights separated by spaces:");

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

heights[i] = scanner.nextInt();

}

Solution solution = new Solution();

int result = solution.trapRainWater(heights);

System.out.println("Trapped rainwater: " + result);

scanner.close();

}

}

Output:

A screen shot of a computer program

Description automatically generated

Time complexity: O(n)

1. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

Code:

import java.util.\*;

class Choco{

public static void main(String arg[]){

Scanner sc=new Scanner(System.in);

System.out.print("enter size");

int n=sc.nextInt();

int[] arr=new int[n];

System.out.print("enter the element:");

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

arr[i]=sc.nextInt();

}

System.out.print("enter m:");

int m=sc.nextInt();

Arrays.sort(arr);

int mdiff=Integer.MAX\_VALUE;

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

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

if(diff<mdiff){

mdiff=diff;

}

}

System.out.print(mdiff);

}

}

A computer screen with white text

Description automatically generated

Time complexity: O(nlogn)

1. Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals

Code:

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

class Merge {

public int[][] merge(int[][] intervals) {

// Sort intervals by start time

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

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

int[] currentInterval = intervals[0];

merged.add(currentInterval);

for (int[] interval : intervals) {

int currentEnd = currentInterval[1];

int nextStart = interval[0];

int nextEnd = interval[1];

if (currentEnd >= nextStart) {

currentInterval[1] = Math.max(currentEnd, nextEnd);

} else {

currentInterval = interval;

merged.add(currentInterval);

}

}

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of intervals:");

int n = scanner.nextInt();

int[][] intervals = new int[n][2];

System.out.println("Enter each interval as two space-separated integers");

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

intervals[i][0] = scanner.nextInt();

intervals[i][1] = scanner.nextInt();

}

Solution solution = new Solution();

int[][] mergedIntervals = solution.merge(intervals);

System.out.println("Merged intervals:");

for (int[] interval : mergedIntervals) {

System.out.println(Arrays.toString(interval));

}

scanner.close();

}

}

Output:

A computer screen shot of a number of intervals

Description automatically generated

Time complexity: O(n)

1. A Boolean Matrix Question Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

Code:

import java.util.Scanner;

class Bool {

public void modifyMatrix(int[][] mat, int M, int N) {

boolean[] row = new boolean[M];

boolean[] col = new boolean[N];

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

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

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

row[i] = true;

col[j] = true;

}

}

}

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

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

if (row[i] || col[j]) {

mat[i][j] = 1;

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of rows (M):");

int M = scanner.nextInt();

System.out.println("Enter the number of columns (N):");

int N = scanner.nextInt();

int[][] mat = new int[M][N];

System.out.println("Enter the elements of the matrix (0 or 1) row by row:");

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

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

mat[i][j] = scanner.nextInt();

}

}

Bool solution = new Bool();

solution.modifyMatrix(mat, M, N);

System.out.println("Modified matrix:");

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

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

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

}

System.out.println();

}

scanner.close();

}

}

Output:

A screenshot of a computer program

Description automatically generated

Time complexity: O(m+n)

1. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form

Code:

import java.util.\*;

public class Bool {

public static List<Integer> spiralOrder(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

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

if (m == 0) return result;

boolean[][] seen = new boolean[m][n];

int[] dr = {0, 1, 0, -1};

int[] dc = {1, 0, -1, 0};

int r = 0, c = 0, di = 0;

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

result.add(matrix[r][c]);

seen[r][c] = true;

int newR = r + dr[di];

int newC = c + dc[di];

if (0 <= newR && newR < m && 0 <= newC && newC < n && !seen[newR][newC]) {

r = newR;

c = newC;

} else {

di = (di + 1) % 4;

r += dr[di];

c += dc[di];

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of rows: ");

int rows = scanner.nextInt();

System.out.print("Enter the number of columns: ");

int cols = scanner.nextInt();

int[][] matrix = new int[rows][cols];

System.out.println("Enter the matrix elements:");

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

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

matrix[i][j] = scanner.nextInt();

}

}

List<Integer> result = spiralOrder(matrix);

System.out.println("Spiral Order:");

for (int num : result) {

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

}

}

}

Output:

A computer screen shot of a number

Description automatically generated

Time complexity: O(n)

1. Check if given Parentheses expression is balanced or not Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not

Code:

import java.util.Scanner;

class Bool {

public static boolean isBalanced(String exp) {

boolean flag = true;

int count = 0;

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

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

count++;

} else {

count--;

}

if (count < 0) {

flag = false;

break;

}

}

if (count != 0) {

flag = false;

}

return flag;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the parentheses expression: ");

String exp = scanner.nextLine();

if (isBalanced(exp)) {

System.out.println("Balanced");

} else {

System.out.println("Not Balanced");

}

}

}

Output:

A black screen with white text

Description automatically generated

Time complexity: O(n)

1. Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

Code:

import java.util.Arrays;

import java.util.Scanner;

class Bool {

static boolean areAnagrams(String s1, String s2) {

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

return false;

}

char[] s1Array = s1.toCharArray();

char[] s2Array = s2.toCharArray();

Arrays.sort(s1Array);

Arrays.sort(s2Array);

return Arrays.equals(s1Array, s2Array);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the first string: ");

String s1 = scanner.nextLine();

System.out.print("Enter the second string: ");

String s2 = scanner.nextLine();

if (areAnagrams(s1, s2)) {

System.out.println("The strings are anagrams.");

} else {

System.out.println("The strings are not anagrams.");

}

}

}

Output:

A screen shot of a computer

Description automatically generated

Time complexity: O(nlogn)

1. Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Code:

import java.util.Scanner;

public class Bool {

static boolean checkPal(String s, int low, int high) {

while (low < high) {

if (s.charAt(low) != s.charAt(high))

return false;

low++;

high--;

}

return true;

}

static String longestPalSubstr(String s) {

int n = s.length();

int maxLen = 1, start = 0;

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

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

if (checkPal(s, i, j) && (j - i + 1) > maxLen) {

start = i;

maxLen = j - i + 1;

}

}

}

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String s = scanner.nextLine();

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

}

}

Output:

A screen shot of a computer program

Description automatically generated

Time complexity: O(n^3)

1. Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”.

Code:

import java.util.Arrays;

import java.util.Scanner;

class Bool {

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 minLength = Math.min(first.length(), last.length());

int i = 0;

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

i++;

}

if (i == 0)

return "-1";

return first.substring(0, i);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of strings: ");

int n = scanner.nextInt();

scanner.nextLine(); // Consume newline

String[] arr = new String[n];

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

System.out.print("Enter string " + (i + 1) + ": ");

arr[i] = scanner.nextLine();

}

System.out.println("The longest common prefix is: " + longestCommonPrefix(arr));

}

}

Output:

A computer screen shot of a black screen

Description automatically generated

Time complexity: O(nlogn+m)

1. Delete middle element of a stack Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

Code:

import java.util.Scanner;

import java.util.Stack;

import java.util.Vector;

public class Bool {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.print("Enter characters (no spaces): ");

String input = scanner.nextLine();

for (char ch : input.toCharArray()) {

st.push(ch);

}

Vector<Character> v = new Vector<>();

while (!st.empty()) {

v.add(st.pop());

}

int n = v.size();

int target = n / 2; // Middle index

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

if (i == target) continue; // Skip the middle element

st.push(v.get(i));

}

System.out.print("Printing stack after deletion of middle: ");

while (!st.empty()) {

System.out.print(st.pop() + " ");

}

}

}

Output:

A screen shot of a computer program

Description automatically generated

Time complexity: O(n)

1. Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1

Code:

import java.util.Scanner;

public class Bool {

static class Stack {

int top;

int items[] = new int[100];

Stack() {

top = -1;

}

void push(int x) {

if (top == 99) {

System.out.println("Stack full");

} else {

items[++top] = x;

}

}

int pop() {

if (top == -1) {

System.out.println("Underflow error");

return -1;

} else {

return items[top--];

}

}

boolean isEmpty() {

return top == -1;

}

}

static void printNGE(int arr[], int n) {

Stack s = new Stack();

s.push(arr[0]);

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

int next = arr[i];

while (!s.isEmpty() && s.items[s.top] < next) {

int element = s.pop();

System.out.println(element + " --> " + next);

}

s.push(next);

}

while (!s.isEmpty()) {

int element = s.pop();

System.out.println(element + " --> -1");

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements: ");

int n = scanner.nextInt();

int arr[] = new int[n];

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

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

arr[i] = scanner.nextInt();

}

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

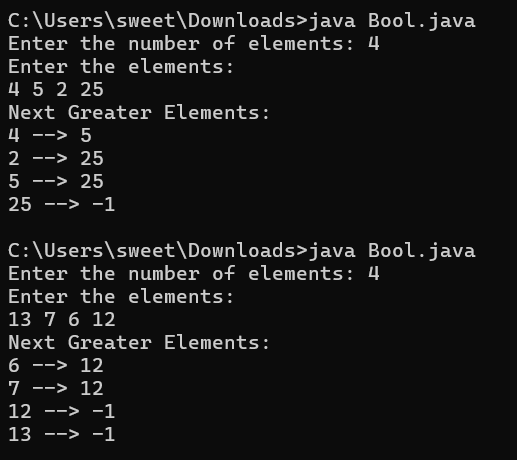
printNGE(arr, n);

scanner.close();

}

}

Output:



Time complexity: O(n)

1. Print Right View of a Binary Tree Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level

Code:

import java.util.ArrayList;

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int x) {

data = x;

left = right = null;

}

}

class Gfg {

static void RecursiveRightView(Node root, int level,

int[] maxLevel, ArrayList<Integer> result) {

if (root == null) return;

if (level > maxLevel[0]) {

result.add(root.data);

maxLevel[0] = level;

}

RecursiveRightView(root.right, level + 1,

maxLevel, result);

RecursiveRightView(root.left, level + 1,

maxLevel, result);

}

static ArrayList<Integer> rightView(Node root) {

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

int[] maxLevel = new int[] {-1};

// Start recursion with root at level 0

RecursiveRightView(root, 0, maxLevel, result);

return result;

}

static void printArray(ArrayList<Integer> arr) {

for (int val : arr) {

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

}

System.out.println();

}

static Node buildTree() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the root node value: ");

int rootValue = scanner.nextInt();

Node root = new Node(rootValue);

System.out.println("Enter the left and right children of each node. Use -1 for no child.");

buildTreeHelper(root, scanner);

return root;

}

static void buildTreeHelper(Node node, Scanner scanner) {

if (node == null) return;

System.out.print("Enter left child of " + node.data + ": ");

int leftValue = scanner.nextInt();

if (leftValue != -1) {

node.left = new Node(leftValue);

buildTreeHelper(node.left, scanner);

}

System.out.print("Enter right child of " + node.data + ": ");

int rightValue = scanner.nextInt();

if (rightValue != -1) {

node.right = new Node(rightValue);

buildTreeHelper(node.right, scanner);

}

}

public static void main(String[] args) {

Node root = buildTree();

ArrayList<Integer> result = rightView(root);

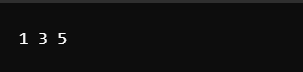
System.out.println("Right view of the tree:");

printArray(result);

}

}

Output:



Time complexity: O(n)

1. Maximum Depth or Height of Binary Tree Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node

Code:

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int val) {

data = val;

left = null;

right = null;

}

}

class GfG {

static int maxDepth(Node node) {

if (node == null)

return 0;

int lDepth = maxDepth(node.left);

int rDepth = maxDepth(node.right);

return Math.max(lDepth, rDepth) + 1;

}

static Node buildTree() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the root node value: ");

int rootValue = scanner.nextInt();

Node root = new Node(rootValue);

System.out.println("Enter the left and right children of each node. Use -1 for no child.");

buildTreeHelper(root, scanner);

return root;

}

static void buildTreeHelper(Node node, Scanner scanner) {

if (node == null) return;

System.out.print("Enter left child of " + node.data + ": ");

int leftValue = scanner.nextInt();

if (leftValue != -1) {

node.left = new Node(leftValue);

buildTreeHelper(node.left, scanner);

}

System.out.print("Enter right child of " + node.data + ": ");

int rightValue = scanner.nextInt();

if (rightValue != -1) {

node.right = new Node(rightValue);

buildTreeHelper(node.right, scanner);

}

}

public static void main(String[] args) {

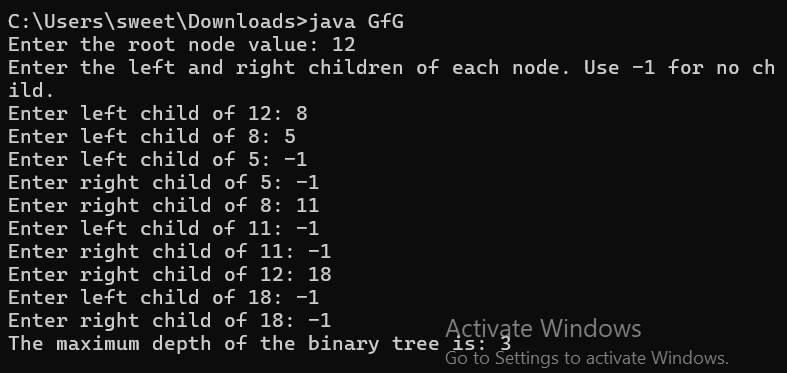
Node root = buildTree();

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

}

}

Output:



Time complexity: O(n)