1. Maximum Subarray Sum – Kadane’s Algorithm:

public class Solution1 {

public int maxSubArray(int[] nums) {

int ans = Integer.MIN\_VALUE;

int add = 0;

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

add += nums[i];

ans = Math.max(ans, add);

if (add < 0) {--

add = 0;

}

}

return ans;

}

public static void main(String[] args) {

Solution1 solution = new Solution();

// Test cases

int[] test1 = {2,3,-8,7,-1,2,3};

int[] test2 = {-2,-4};

int[] test3 = {5, 4, -1, 7, 8};

// Run and print test results

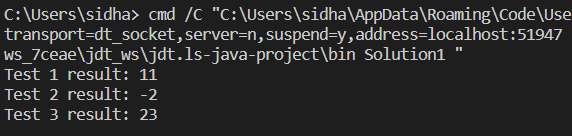
System.out.println("Test 1 result: " + solution.maxSubArray(test1));

System.out.println("Test 2 result: " + solution.maxSubArray(test2));

System.out.println("Test 3 result: " + solution.maxSubArray(test3));

}

}



TIME COMPLEXITY : O(n)

2.Maximum Product Subarray  
class Solution2 {

public int maxProduct(int[] nums) {

int n = nums.length;

int l = 1;5

int r = 1;

int ans = nums[0];

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

l = (l == 0 ? 1 : l) \* nums[i];

r = (r == 0 ? 1 : r) \* nums[n - 1 - i];

int temp = Math.max(l, r);

ans = Math.max(temp, ans);

}

return ans;

}

public static void main(String[] args) {

Solution2 solution = new Solution();

// Test cases

int[] test1 = {-2, 6, -3, -10, 0, 2};

int[] test2 = {-1, 0, 0};

int[] test3 = {1, 2, 3, 4, -1};

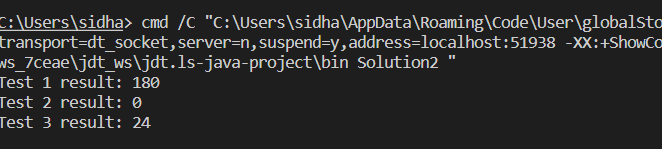
System.out.println("Test 1 result: " + solution.maxProduct(test1));

System.out.println("Test 2 result: " + solution.maxProduct(test2));

System.out.println("Test 3 result: " + solution.maxProduct(test3));

}

}



Time Complexity : O(n)

3.Search in a sorted and rotated array   
public class Solution3 {

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

int left = 0;

int right = nums.length-1;

while(left <= right){

int mid = (left+right)/2;

if(nums[mid] == target){

return mid;

}

else if(nums[left] <= nums[mid]){

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

right = mid-1;

}

else{

left = mid + 1;

}

}

else{

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

left = mid + 1;

}

else{

right = mid - 1;

}

}

}

return -1;

}

public static void main(String[] args) {

Solution3 solution = new Solution();

// Test cases

int[] test1 = { 4, 5, 6, 7, 0, 1, 2} ;

int key1 = 0;

int[] test2 = {4, 5, 6, 7, 0, 1, 2};

int key2 = 3;

int[] test3 = {50, 10, 20, 30, 40};

int key3 = 10;

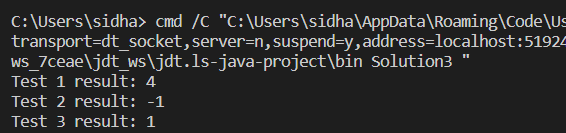
System.out.println("Test 1 result: " + solution.maxprodSubArray(test1,key1));

System.out.println("Test 2 result: " + solution.maxprodSubArray(test2,key2));

System.out.println("Test 3 result: " + solution.maxprodSubArray(test3,key3));

}

}



Time complexity : O(logn)

4.Container with most water

import java.util.\*;

class Solution4 {

public int maxArea(int[] height) {

int left = 0;

int right = height.length - 1;

int dis = 0;

int area = 0;

while (left < right) {

dis = right - left;

int leftside = height[left];

int rightside = height[right];

area = Math.max(area, Math.min(leftside, rightside) \* dis);

if (leftside >= rightside) {

right--;

} else {

left++;

}

}

return area;

}

public static void main(String[] args) {

Solution solution = new Solution();

// Test cases

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

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

int [] arr3 = {1,1,1,1};

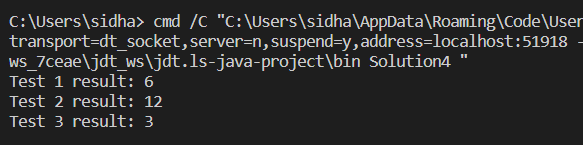
System.out.println("Test 1 result: " + solution.maxArea(arr1));

System.out.println("Test 2 result: " + solution.maxArea(arr2));

System.out.println("Test 3 result: " + solution.maxArea(arr3));

}

}



Time Complexity : O(n)

5.Factorial of a large Number

import java.math.BigInteger;

public class Solution5 {

public BigInteger Factorial(int num) {

BigInteger ans = BigInteger.ONE;

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

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

}

return ans;

}

public static void main(String[] args) {

Solution5 solution = new Solution();

// Test cases

int test1 = 100 ;

int test2 = 50;

int test3 = 90;

// Run and print test results

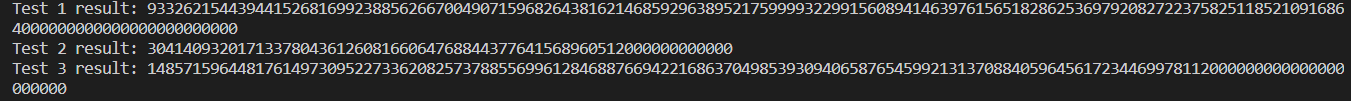
System.out.println("Test 1 result: " + solution.Factorial(test1));

System.out.println("Test 2 result: " + solution.Factorial(test2));

System.out.println("Test 3 result: " + solution.Factorial(test3));

}

}



Time Complexity : O(n)

6.Trapping Rainwater

class Solution6 {

public int TrappingRainwater(int[] height) {

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

return 0;

}

int left =0;

int right = height.length -1;

int maxLeft =0;

int maxRight = 0;

int ans = 0;

while(left < right){

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

if(height[left] >= maxLeft){

maxLeft = height[left];

}

else{

ans += maxLeft-height[left];

}

left ++;

}

else{

if(height[right] > maxRight){

maxRight = height[right];

}

else{

ans += maxRight-height[right];

}

right --;

}

} return ans;

}

public static void main(String[] args) {

Solution6 solution = new Solution();

// Test cases

int [] test1 = {3, 0, 1, 0, 4, 0, 2};

int [] test2 = {3, 0, 2, 0, 4} ;

int [] test3 = {10, 9, 0, 5 } ;

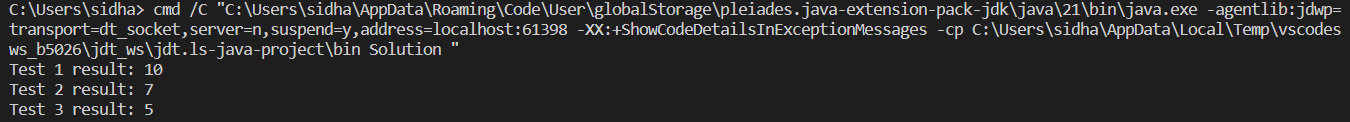
System.out.println("Test 1 result: " + solution.TrappingRainwater(test1));

System.out.println("Test 2 result: " + solution.TrappingRainwater(test2));

System.out.println("Test 3 result: " + solution.TrappingRainwater(test3));

}

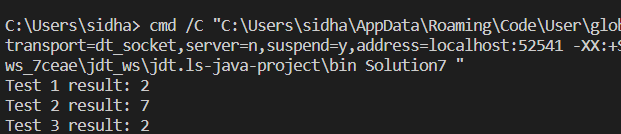
}



Time complexity : O(n)

7.Chocolate Distribution Problem

import java.util.Arrays;  
  
class Solution7{  
 public int Chocolate(int[] choco, int m) {  
 int ans = Integer.MAX\_VALUE;  
 Arrays.sort(choco);  
 for(int i = 0; i <= choco.length - m; i++) {  
 int diff = choco[i + m - 1] - choco[i];  
 ans = Math.min(diff, ans);  
 }  
 return ans;  
 }  
  
 public static void main(String[] args) {  
 Solution7 solution = new Solution7();  
  
 // Test cases  
 int[] test1 = {7, 3, 2, 4, 9, 12, 56};  
 int[] test2 = {7 ,3 ,2 , 4, 9, 12, 56};  
 int[] test3 = {1, 5, 100, 101, 102};  
   
 int m1 = 3;  
 int m2 = 5;  
 int m3 = 3;  
  
 System.out.println("Test 1 result: " + solution.Chocolate(test1, m1));  
 System.out.println("Test 2 result: " + solution.Chocolate(test2, m2));  
 System.out.println("Test 3 result: " + solution.Chocolate(test3, m3));  
 }  
}



Time Complexity : O(nlogn)

8.Merge Overlapping Intervals

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

class Solution8 {

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

if (intervals.length <= 1) {

return intervals;

}

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

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

merged\_intervals.add(intervals[0]);

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

int[] current\_interval = intervals[i];

int[] last\_interval = merged\_intervals.get(merged\_intervals.size() - 1);

if (current\_interval[0] <= last\_interval[1]) {

last\_interval[1] = Math.max(last\_interval[1], current\_interval[1]);

} else {

merged\_intervals.add(current\_interval);

}

}

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

}

public static void main(String[] args) {

Solution8 solution = new Solution8();

// Test cases

int[][] arr1 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

int[][] arr2 = {{-1, 4}, {1, 5}, {2, 4}, {8, 10}};

int[][] arr3 = {{1,6},{7,8}};

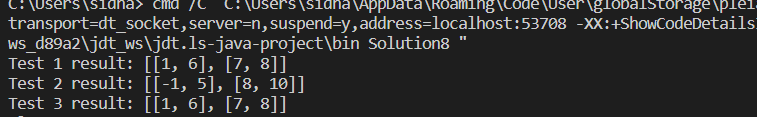
System.out.println("Test 1 result: " + Arrays.deepToString(solution.merge(arr1)));

System.out.println("Test 2 result: " + Arrays.deepToString(solution.merge(arr2)));

System.out.println("Test 3 result: " + Arrays.deepToString(solution.merge(arr3)));

}

}



Time Complexity : O(nlogn)

9.A Boolean Matrix Question

import java.util.\*;

class Solution9 {

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

ArrayList<int[]> change = new ArrayList<>();

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

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

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

change.add(new int[]{i, j});

}

}

}

for (int[] e : change) {

int r = e[0];

int c = e[1];

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

matrix[r][i] = 1;

}

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

matrix[j][c] = 1;

}

}

return matrix;

}

public static void main(String[] args) {

Solution9 solution = new Solution9();

// Test cases

int[][] arr1 = {{1, 0}, {0, 0}};

int[][] arr2 = {{0, 0, 0}, {0, 0, 1}};

int[][] arr3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};

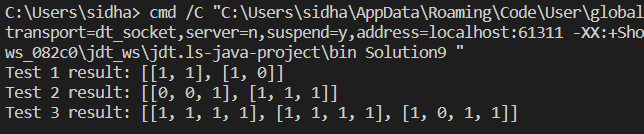
System.out.println("Test 1 result: " + Arrays.deepToString(solution.merge(arr1)));

System.out.println("Test 2 result: " + Arrays.deepToString(solution.merge(arr2)));

System.out.println("Test 3 result: " + Arrays.deepToString(solution.merge(arr3)));

}

}



Time Complexity : O(n\*m)

10. Spiral Matrix

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

class Solution10 {

public int[] spiral(int[][] matrix) {

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

if (matrix.length == 0 || matrix[0].length == 0) return new int[0];

int top = 0;

int bottom = matrix.length - 1;

int left = 0;

int right = matrix[0].length - 1;

while (true) {

for (int i = left; i <= right; i++) res.add(matrix[top][i]);

top++;

if (left > right || top > bottom) break;

for (int i = top; i <= bottom; i++) res.add(matrix[i][right]);

right--;

if (left > right || top > bottom) break;

for (int i = right; i >= left; i--) res.add(matrix[bottom][i]);

bottom--;

if (left > right || top > bottom) break;

for (int i = bottom; i >= top; i--) res.add(matrix[i][left]);

left++;

if (left > right || top > bottom) break;

}

int[] result = new int[res.size()];

for (int i = 0; i < res.size(); i++) {

result[i] = res.get(i);

}

return result;

}

public static void main(String[] args) {

Solution10 solution = new Solution10();

int[][] arr1 = {{1,2,3,4}, {5,6,7,8},{9,10,11,12}, {13,14,15,16}};

int[][] arr2 = {{1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12},{13, 14, 15, 16, 17, 18}};

int[][] arr3 = {{1, 6, 7, 1}, {0, 2, 1, 10}, {2, -1, 9, 4}};

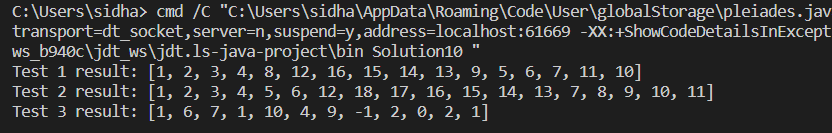
System.out.println("Test 1 result: " + Arrays.toString(solution.spiral(arr1)));

System.out.println("Test 2 result: " + Arrays.toString(solution.spiral(arr2)));

System.out.println("Test 3 result: " + Arrays.toString(solution.spiral(arr3)));

}

}



Time Complexity : O(m\*n)

13. Check Valid Parenthesis

import java.util.Stack;

class Solution13 {

public Boolean validPara(String parenthesis) {

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

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

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

st.push('(');

}

else{

if(st.isEmpty()){

return false;

}

st.pop();

}

}

return st.isEmpty();

}

public static void main(String[] args) {

Solution13 solution = new Solution();

// Test cases

String test1 = "(()";

String test2 = "()()()";

String test3 = "()()(())";

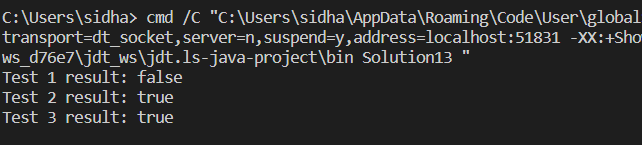
System.out.println("Test 1 result: " + solution.validPara(test1));

System.out.println("Test 2 result: " + solution.validPara(test2));

System.out.println("Test 3 result: " + solution.validPara(test3));

}

}



Time Complexity : O(n)

14.Check if two String are anagrams

import java.util.HashMap;

class Solution14 {

public boolean isAnagram(String s, String t) {

if(s.length() != t.length()){

return false;

}

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

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

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

}

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

if(!map.containsKey(c)){

return false;

}

map.put(c, map.get(c) - 1);

if(map.get(c) == 0){

map.remove(c);

}

}

return map.isEmpty();

}

public static void main(String[] args) {

Solution14 solution = new Solution();

// Test cases

String s1 = "g", t1 = "g";

String s2 = "allergy", t2 = "allergic";

String s3 = "geeks", t3 = "kseeg";

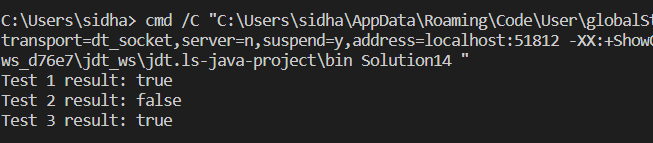
System.out.println("Test 1 result: " + solution.isAnagram(s1, t1));

System.out.println("Test 2 result: " + solution.isAnagram(s2, t2));

System.out.println("Test 3 result: " + solution.isAnagram(s3, t3));

}

}



Time Complexity : O(n)

15.Longest Palindrome Substring

class Solution15 {

public String longestpalindrome(String s) {

if(s==""){

return "''";

}

String ans = "";

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

String temp = pal(s, i, i);

if (temp.length() > ans.length()) {

ans = temp;

}

String temp2 = pal(s, i, i + 1);

if (temp2.length() > ans.length()) {

ans = temp2;

}

}

return ans;

}

public String pal(String s, int l, int r) {

while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {

l--;

r++;

}

return s.substring(l + 1, r);

}

public static void main(String[] args) {

Solution15 solution = new Solution15();

String arr1 = "";

String arr2 = "geeks";

String arr3 = "abc";

System.out.println("Test 1 result: " + solution.longestpalindrome(arr1));

System.out.println("Test 2 result: " + solution.longestpalindrome(arr2));

System.out.println("Test 3 result: " + solution.longestpalindrome(arr3));

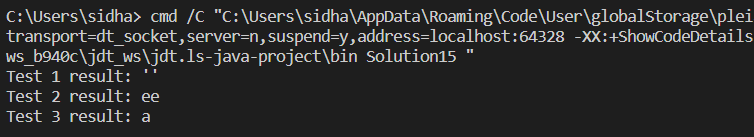
}

}

longestpalindrome(arr3));

}

}

/  
  
Time Complexity : O(n^2)

16.Longest Substring Prefix  
  
import java.util.Arrays;

class Solution16 {

public String Commonprefix(String[] strs) {

Arrays.sort(strs);

int n = strs.length;

String ans = "";

String first = strs[0];

String last = strs[n-1];

for(int i = 0; i < Math.min(first.length(),last.length()); i++){

if(first.charAt(i) != last.charAt(i)){

break;

}

ans += first.charAt(i);

}

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

}

public static void main(String[] args) {

Solution16 solution = new Solution16();

// Test cases

String [] test1 = {"geekforgeeks","geeks", "geek", "geezer"};

String [] test2 = {"hello" , "world"};

String [] test3 = {"madam","madman"};

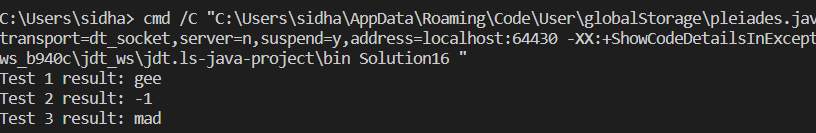
System.out.println("Test 1 result: " + solution.Commonprefix(test1));

System.out.println("Test 2 result: " + solution.Commonprefix(test2));

System.out.println("Test 3 result: " + solution.Commonprefix(test3));

}

}



Time Complexity : O(nlogn)

17.Delete middle element of a stack

import java.util.Stack;

class Solution17 {

public void deleteMiddle(Stack<Integer> stack, int size, int current) {

if (current == size / 2) {

stack.pop();

return;

}

int temp = stack.pop();

deleteMiddle(stack, size, current + 1);

stack.push(temp);

}

public void deleteMiddle(int[] arr) {

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

for (int i : arr) {

stack.push(i);

}

int size = stack.size();

deleteMiddle(stack, size, 0);

int[] result = new int[stack.size()];

int i = 0;

while (!stack.isEmpty()) {

result[i++] = stack.pop();

}

for (int j = result.length - 1; j >= 0; j--) {

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

}

}

public static void main(String[] args) {

Solution17 solution = new Solution17();

int[] arr1 = {1, 2, 3, 4, 5};

int[] arr2 = {1, 2, 3, 4, 5, 6};

int[] arr3 = {0};

solution.deleteMiddle(arr1);

System.out.println();

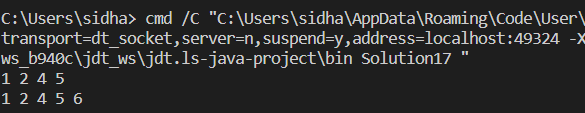
solution.deleteMiddle(arr2);

System.out.println();

solution.deleteMiddle(arr3);

}

}



Time Complexity : O(n)

18. Next Greater Element (NGE) for every element in given Array

import java.util.Stack;

class Solution18 {

public void nextGreaterElement(int[] arr) {

int n = arr.length;

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

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

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

stack.pop();

}

if (stack.isEmpty()) {

System.out.print(arr[i] + " -> -1\n");

} else {

System.out.print(arr[i] + " -> " + stack.peek() + "\n");

}

stack.push(arr[i]);

}

}

public static void main(String[] args) {

Solution18 solution = new Solution18();

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

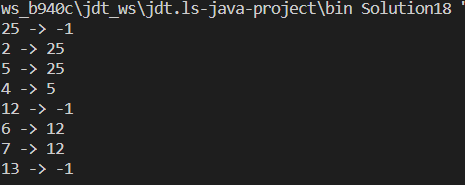
int [] arr2 = {13, 7, 6, 12};

solution.nextGreaterElement(arr1);

solution.nextGreaterElement(arr2);

}

}



Time Complexity : O(n)

19.Print Right View of a Binary Tree

import java.util.ArrayList;

import java.util.List;

public class Solution19 {

public List<Integer> rightSideView(TreeNode root) {

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

rightView(root, result, 0);

return result;

}

public void rightView(TreeNode curr, List<Integer> result, int currDepth) {

if (curr == null) {

return;

}

if (currDepth == result.size()) {

result.add(curr.val);

}

rightView(curr.right, result, currDepth + 1);

rightView(curr.left, result, currDepth + 1);

}

public static void main(String[] args) {

Solution19 solution = new Solution19();

TreeNode root1 = null;

TreeNode root2 = new TreeNode(1, null, new TreeNode(3));

TreeNode root3 = new TreeNode(1, new TreeNode(2), new TreeNode(3, new TreeNode(5), new TreeNode(4))); // [1, 2, 3, null, 5, null, 4]

System.out.println(solution.rightSideView(root1));

System.out.println(solution.rightSideView(root2));

System.out.println(solution.rightSideView(root3));

}

}

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode() {}

TreeNode(int val) { this.val = val; }

TreeNode(int val, TreeNode left, TreeNode right) {

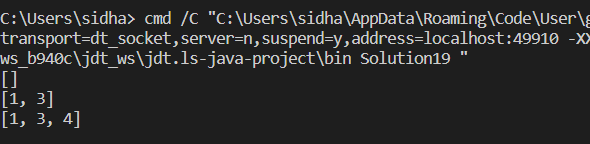
this.val = val;

this.left = left;

this.right = right;

}

}

  
Time Complexity : O(n)

20. Maximum Depth or Height of Binary Tree

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) {

return 0;

}

return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));

}

public static void main(String[] args) {

Solution solution = new Solution();

TreeNode root1 = new TreeNode(3);

root1.left = new TreeNode(9);

root1.right = new TreeNode(20);

root1.right.left = new TreeNode(15);

root1.right.right = new TreeNode(7);

TreeNode root2 = new TreeNode(1);

root2.right = new TreeNode(2);

System.out.println("Max depth of tree 1: " + solution.maxDepth(root1));

System.out.println("Max depth of tree 2: " + solution.maxDepth(root2));

}

}

public class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode() {}

TreeNode(int val) { this.val = val; }

TreeNode(int val, TreeNode left, TreeNode right) {

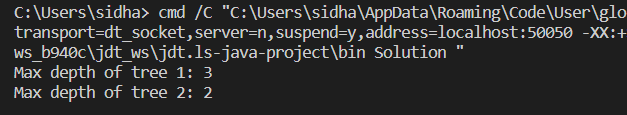
this.val = val;

this.left = left;

this.right = right;

}

}



Time Complexity : O(n)