GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026

DATE: 09-11-24 **NAME: JASHVARTHINI R - CSBS**

1. **MAXIMUM SUBARRAY SUM – KADANE'S ALGORITHM:**

```java
import java.util.Scanner;

public class KabaneAlgorithm {
    public static int kabaneAlgorithm(int arr[]) {
        int currSum = 0;
        int maxSum = arr[0];

        for (int n : arr) {
            currSum = Math.max(currSum, 0);
            currSum += n;
            maxSum = Math.max(maxSum, currSum);
        }
        return maxSum;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        int output = kabaneAlgorithm(arr);
        System.out.println("The maximum subarray sum: " + output);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 7
Enter the elements of the array:
2 3 -8 7 -1 2 3
The maximum subarray sum: 11
```

```
Enter the number of elements in the array: 2
Enter the elements of the array:
-2 -4
The maximum subarray sum: -2
```

TIME COMPLEXITY: O(n)

2. MAXIMUM PRODUCT SUBARRAY

```java
import java.util.Scanner;

public class MaxProductSubarray {
    public static int MaxProductSubarray(int arr[]) {
        int res=arr[0];
        int cMin=1, cMax=1;
        for(int n: arr){
            int temp=cMax*n;
            cMax= Math.max((Math.max(cMax*n, (cMin*n))), n);
            cMin= Math.min((Math.min(cMin*n, (temp))), n);
            res= Math.max(res, cMax);
        }

        return res;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        int output = MaxProductSubarray(arr);
        System.out.println("The maximum product subarray: " + output);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 6
Enter the elements of the array:
-2 6 -3 -10 0 2
The maximum product subarray: 180
```

```
Enter the number of elements in the array: 5
Enter the elements of the array:
-1 -3 -10 0 60
The maximum product subarray: 60
```

TIME COMPLEXITY: O(n)

3. SEARCH IN A SORTED AND ROTATED ARRAY

```java
import java.util.Scanner;

public class SearchInRotatedSortedArray {

    public static int searchInRotatedSortedArray(int arr[], int target) {
        int l = 0;
        int r = arr.length - 1;
        while (l <= r) {
            int mid = (l + r) / 2;
            if (arr[mid] == target) {
                return mid;
            }
            if (arr[l] <= arr[mid]) {
                if (arr[l] <= target && target <= arr[mid]) {
                    r = mid - 1;
                } else {
                    l = mid + 1;
                }
            } else {
                if (arr[mid] <= target && target <= arr[r]) {
                    l = mid + 1;
                } else {
                    r = mid - 1;
                }
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        System.out.print("Enter the target element: ");
        int target = scanner.nextInt();
        int output = searchInRotatedSortedArray(arr, target);
        if (output == -1) {
            System.out.println("Element not found.");
        } else {
            System.out.println("Element found at index: " + output);
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 7
Enter the elements of the array:
4 5 6 7 0 1 2
Enter the target element: 0
Element found at index: 4
```

```
Enter the number of elements in the array: 5
Enter the elements of the array:
50 10 20 30 40
Enter the target element: 10
Element found at index: 1
```

TIME COMPLEXITY: O(log n)

4. **CONTAINER WITH MOST WATER**

```java
import java.util.Scanner;

public class MostWater {

    public static int mostWater(int arr[]) {
        int n= arr.length;
        int left=0;
        int right=n-1;
        int area=0;

        while(left<right){
            area=Math.max(area, Math.min(arr[left], arr[right])*(right-left));

            if (arr[left]<arr[right]){
                left+=1;
            }
            else{
                right-=1;
            }
        }
        return area;

    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
```

```
        }

        int output = mostWater(arr);
        System.out.println("The Container area is: "+output);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 4
Enter the elements of the array:
1 5 4 3
The Container area is: 6
```

```
Enter the number of elements in the array: 5
Enter the elements of the array:
3 1 2 4 5
The Container area is: 12
```

TIME COMPLEXITY: O(n)

5. **FIND THE FACTORIAL OF A LARGE NUMBER**

```
import java.util.Scanner;
import java.math.BigInteger;

public class Factorial {

    public static BigInteger factorial(int n) {
        BigInteger fact = BigInteger.ONE;
        for(int i=2; i<=n; i++){
            fact = fact.multiply(BigInteger.valueOf(i));
        }
        return fact;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number: ");
        int n = scanner.nextInt();

        BigInteger output = factorial(n);
        System.out.println("The Factorial of the number is: " + output);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number: 100
The Factorial of the number is: 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
```

```
Enter the number: 50
The Factorial of the number is: 30414093201713378043612608166064768844377641568960512000000000000
```

TIME COMPLEXITY: O(n)

6. TRAPPING RAINWATER PROBLEM

```java
import java.util.Scanner;

public class TrappingWater {

    public static int trappingWater(int heights[]) {
        int left=0;
        int right=heights.length-1;
        int left_max= heights[left];
        int right_max= heights[right];
        int water=0;

        while(left<right){
            if(left_max<right_max){
                left+=1;
                left_max=Math.max(left_max,heights[left]);
                water+=left_max-heights[left];
        }

        else{
            right-=1;
            right_max=Math.max(right_max, heights[right]);
            water+=right_max-heights[right];
        }}

        return water;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        int heights[] = new int[n];
        System.out.print("Enter the element: ");
        for (int i = 0; i < n; i++) {
```

```
            heights[i] = scanner.nextInt();
        }

        int output = trappingWater(heights);
        System.out.println("The height of the water it can hold is: " + output);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 7
Enter the element: 3 0 1 0 4 0 2
The height of the water it can hold is: 10
```

```
Enter the number of elements in the array: 5
Enter the element: 3 0 2 0 4
The height of the water it can hold is: 7
```

```
Enter the number of elements in the array: 4
Enter the element: 1 2 3 4
The height of the water it can hold is: 0
```

TIME COMPLEXITY: O(N)

7. CHOCOLATE DISTRIBUTION PROBLEM

```
import java.util.Arrays;
import java.util.Scanner;

public class ChocolateDistribution {

    public 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];
```

```
            if (diff < minDiff) {
                minDiff = diff;
            }
        }

        return minDiff;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of packets: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the number of chocolates in each packet:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.print("Enter the number of students: ");
        int m = scanner.nextInt();

        int result = findMinDiff(arr, n, m);
        if (result == -1) {
            System.out.println("The number of students is greater than the number of
packets.");
        } else {
            System.out.println("Minimum difference is " + result);
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of packets: 7
Enter the number of chocolates in each packet:
7 3 2 4 9 12 56
Enter the number of students: 3
Minimum difference is 2
```

```
Enter the number of packets: 7
Enter the number of chocolates in each packet:
7 3 2 4 9 12 56
Enter the number of students: 5
Minimum difference is 7
```

TIME COMPLEXITY: O(n log n)

8. **MERGE OVERLAPPING INTERVAL**

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class MergeIntervals {

    public static int[][] merge(int[][] intervals) {
        if (intervals.length <= 1) {
            return intervals;
        }

        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.print("Enter the number of intervals: ");
        int n = scanner.nextInt();
        int[][] intervals = new int[n][2];

        System.out.println("Enter each interval as two integers (start end):");
        for (int i = 0; i < n; i++) {
            System.out.print("Interval " + (i + 1) + ": ");
            intervals[i][0] = scanner.nextInt();
            intervals[i][1] = scanner.nextInt();
        }
```

```
        int[][] result = merge(intervals);
        System.out.println("Merged intervals: " + Arrays.deepToString(result));

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of intervals: 4
Enter each interval as two integers (start end):
Interval 1: 1 3
Interval 2: 2 4
Interval 3: 6 8
Interval 4: 9 10
Merged intervals: [[1, 4], [6, 8], [9, 10]]
```

```
Enter the number of intervals: 4
Enter each interval as two integers (start end):
Interval 1: 7 8
Interval 2: 1 5
Interval 3: 2 4
Interval 4: 4 6
Merged intervals: [[1, 6], [7, 8]]
```

TIME COMPLEXITY: O(n log n)

9. A BOOLEAN MATRIX QUESTION

```
import java.util.Scanner;

public class BooleanMatrix {

    public static 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++) {
            if (row[i]) {
                for (int j = 0; j < n; j++) {
```

```
                    mat[i][j] = 1;
                }
            }
        }

        for (int j = 0; j < n; j++) {
            if (col[j]) {
                for (int i = 0; i < m; i++) {
                    mat[i][j] = 1;
                }
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int m = scanner.nextInt();
        System.out.print("Enter the number of columns: ");
        int n = scanner.nextInt();

        int[][] mat = new int[m][n];
        System.out.println("Enter the elements of the matrix:");
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                mat[i][j] = scanner.nextInt();
            }
        }

        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:

```
Enter the number of rows: 2
Enter the number of columns: 2
Enter the elements of the matrix:
1 0
0 0
Modified Matrix:
1 1
1 0
```

```
Enter the number of rows: 2
Enter the number of columns: 3
Enter the elements of the matrix:
0 0 0
0 0 1
Modified Matrix:
0 0 1
1 1 1
```

```
Enter the number of rows: 3
Enter the number of columns: 4
Enter the elements of the matrix:
1 0 0 1
0 0 1 0
0 0 0 0
Modified Matrix:
1 1 1 1
1 1 1 1
1 0 1 1
```

TIME COMPLEXITY: O(n*m)

10. PRINT A GIVEN MATRIX IN SPIRAL FORM

```java
import java.util.Scanner;

public class SpiralMatrix {

    public static void printSpiral(int[][] matrix, int m, int n) {
        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) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int m = scanner.nextInt();
        System.out.print("Enter the number of columns: ");
        int n = scanner.nextInt();

        int[][] matrix = new int[m][n];
        System.out.println("Enter the elements of the matrix:");
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Spiral Order:");
        printSpiral(matrix, m, n);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of rows: 4
Enter the number of columns: 4
Enter the elements of the matrix:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Spiral Order:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

```
Enter the number of rows: 3
Enter the number of columns: 6
Enter the elements of the matrix:
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
Spiral Order:
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

TIME COMPLEXITY: O(n^2)

13. CHECK IF GIVEN PARENTHESES EXPRESSION IS BALANCED OR NOT

```java
import java.util.Scanner;
import java.util.Stack;

public class BalancedParentheses {

    public static boolean isBalanced(String str) {
        Stack<Character> stack = new Stack<>();

        for (char ch : str.toCharArray()) {
            if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                if (stack.isEmpty()) {
                    return false;
                }
                stack.pop();
            }
        }

        return stack.isEmpty();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a parentheses expression: ");
        String str = scanner.nextLine();

        if (isBalanced(str)) {
            System.out.println("Balanced");
        } else {
            System.out.println("Not Balanced");
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter a parentheses expression: ((()))()()
Balanced
```

```
Enter a parentheses expression: ())((())
Not Balanced
```

TIME COMPLEXITY: O(n)

14. CHECK IF TWO STRINGS ARE ANAGRAMS OF EACH OTHER

```java
import java.util.Arrays;
import java.util.Scanner;

public class AnagramCheck {

    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }

        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);

        return Arrays.equals(arr1, arr2);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first string: ");
        String s1 = scanner.nextLine();
        System.out.print("Enter 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.");
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter first string: geeks
Enter second string: kseeg
The strings are anagrams.
```

```
Enter first string: allergy
Enter second string: allergic
The strings are not anagrams.
```

```
Enter first string: racecar
Enter second string: racecar
The strings are anagrams.
```

TIME COMPLEXITY: O(1)

15. LONGEST PALINDROMIC SUBSTRING

```java
import java.util.*;

public class LongestPalindrome {

    static String longestPalSubstr(String s) {
        int n = s.length();
        if (n == 0) return "";

        int start = 0, maxLen = 1;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j <= 1; j++) {
                int low = i;
                int hi = i + j;

                while (low >= 0 && hi < n && s.charAt(low) == s.charAt(hi)) {
                    int currLen = hi - low + 1;
                    if (currLen > maxLen) {
                        start = low;
                        maxLen = currLen;
                    }
                    low--;
                    hi++;
                }
            }
        }

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

    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);
        System.out.print("Enter a string: ");
```

```
        String s = scanner.nextLine();

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

        scanner.close();
    }
}
```

OUTPUT:

```
Enter a string: geeks
Longest Palindromic Substring: ee
```

```
Enter a string: assessment
Longest Palindromic Substring: ssess
```

```
Enter a string: jashvarthini
Longest Palindromic Substring: ini
```

TIME COMPLEXITY: O(n^2)

16. LONGEST COMMON PREFIX USING SORTING

```
import java.util.Scanner;

public class LongestCommonPrefix {
    static String longestCommonPrefix(String[] arr) {
        int n = arr.length;
        if (n == 0) return "-1";

        java.util.Arrays.sort(arr);

        String first = arr[0];
        String last = arr[n - 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) {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Enter the number of strings: ");
        int n = scanner.nextInt();
        scanner.nextLine();

        String[] arr = new String[n];
        System.out.println("Enter the strings:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextLine();
        }

        String result = longestCommonPrefix(arr);
        System.out.println("Longest Common Prefix: " + result);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of strings: 2
Enter the strings:
hello
world
Longest Common Prefix: -1
```

```
Enter the number of strings: 3
Enter the strings:
goodmorning
goodevening
goodnight
Longest Common Prefix: good
```

TIME COMPLEXITY: O(nlogn+m)

17. DELETE MIDDLE ELEMENT OF A STACK

```
import java.util.*;

public class DeleteMiddleElement {
    public static void deleteMiddleElement(Stack<Integer> stack, int currentIndex,
int size) {
        if (currentIndex == size / 2) {
            stack.pop();
            return;
        }

        int topElement = stack.pop();
        deleteMiddleElement(stack, currentIndex + 1, size);
```

```
        stack.push(topElement);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements in the stack: ");
        int n = scanner.nextInt();

        Stack<Integer> stack = new Stack<>();
        System.out.println("Enter the elements of the stack:");
        for (int i = 0; i < n; i++) {
            int element = scanner.nextInt();
            stack.push(element);
        }

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

        deleteMiddleElement(stack, 0, stack.size());

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

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the stack: 5
Enter the elements of the stack:
1 2 3 4 5
Original Stack: [1, 2, 3, 4, 5]
Stack after deleting middle element: [1, 2, 4, 5]
```

```
Enter the number of elements in the stack: 1
Enter the elements of the stack:
0
Original Stack: [0]
Stack after deleting middle element: []
```

TIME COMPLEXITY: O(N)

18. NEXT GREATER ELEMENT (NGE) FOR EVERY ELEMENT IN GIVEN ARRAY

```
import java.util.*;

public class NextGreaterElement {

    public static void printNextGreaterElement(int[] arr) {
        Stack<Integer> stack = new Stack<>();
```

```
        for (int i = arr.length - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }

            if (stack.isEmpty()) {
                System.out.println(arr[i] + " --> -1");
            } else {
                System.out.println(arr[i] + " --> " + stack.peek());
            }

            stack.push(arr[i]);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");

        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        printNextGreaterElement(arr);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 4
Enter the elements of the array:
13 7 6 12
12 --> -1
6 --> 12
7 --> 12
13 --> -1
```

TIME COMPLEXITY: O(n)

19. PRINT RIGHT VIEW OF A BINARY TREE

```
import java.util.*;
```

```java
class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
        left = right = null;
    }
}

public class Solution {

    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        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 TreeNode buildTree(List<Integer> nodes) {
        if (nodes.isEmpty() || nodes.get(0) == -1)
            return null;

        TreeNode root = new TreeNode(nodes.get(0));
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        int i = 1;
        while (i < nodes.size()) {
            TreeNode current = queue.poll();

            if (nodes.get(i) != -1) {
                current.left = new TreeNode(nodes.get(i));
                queue.offer(current.left);
            }
            i++;

            if (i < nodes.size() && nodes.get(i) != -1) {
                current.right = new TreeNode(nodes.get(i));
                queue.offer(current.right);
```

```
            }
            i++;
        }
        return root;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter tree nodes in level order (-1 for null nodes): ");
        String[] input = sc.nextLine().split(" ");
        List<Integer> nodes = new ArrayList<>();

        for (String s : input) {
            nodes.add(Integer.parseInt(s));
        }

        TreeNode root = buildTree(nodes);

        Solution solution = new Solution();
        List<Integer> rightViewNodes = solution.rightSideView(root);
        System.out.println("Right View: " + rightViewNodes);

        sc.close();
    }
}
```

OUTPUT:

```
Enter tree nodes in level order (-1 for null nodes): 1 2 3 4 5
Right View: [1, 3, 5]
```

TIME COMPLEXITY: O(n)

20. MAXIMUM DEPTH OR HEIGHT OF BINARY TREE

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int x) { val = x; }
}

public class BinaryTreeHeight {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter values in level order (use -1 for null nodes):");
```

```
        String[] values = sc.nextLine().split(" ");

        TreeNode root = buildTree(values);
        System.out.println("The height of the tree is: " + maxDepth(root));

        sc.close();
    }

    public static TreeNode buildTree(String[] values) {
        if (values.length == 0 || values[0].equals("-1")) return null;

        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        int i = 1;

        while (i < values.length) {
            TreeNode current = queue.poll();
            if (!values[i].equals("-1")) {
                current.left = new TreeNode(Integer.parseInt(values[i]));
                queue.add(current.left);
            }
            System.out.println("Node " + current.val + " left child: " +
(current.left != null ? current.left.val : "null"));
            i++;
            if (i < values.length && !values[i].equals("-1")) {
                current.right = new TreeNode(Integer.parseInt(values[i]));
                queue.add(current.right);
            }
            System.out.println("Node " + current.val + " right child: " +
(current.right != null ? current.right.val : "null"));
            i++;
        }

        return root;
    }

    public static int maxDepth(TreeNode root) {
        if (root == null) return 0;
        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);

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

OUTPUT:

```
Enter values in level order (use -1 for null nodes):
1 2 3 4 -1 -1 5 -1 -1 6 7
Node 1 left child: 2
Node 1 right child: 3
Node 2 left child: 4
Node 2 right child: null
Node 3 left child: null
Node 3 right child: 5
Node 4 left child: null
Node 4 right child: null
Node 5 left child: 6
Node 5 right child: 7
The height of the tree is: 4
```

TIME COMPLEXITY: O(N)