**CODING PRACTICE PROBLEMS DATE:09/11/24**

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. Input: arr[] = {2, 3, -8, 7, -1, 2, 3} Output: 11 Explanation: The subarray {7, -1, 2, 3} has the largest sum 11. Input: arr[] = {-2, -4} Output: –2 Explanation: The subarray {-2} has the largest sum -2. Input: arr[] = {5, 4, 1, 7, 8} Output: 25 Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

CODE:

package util;

class array {

static int maxSubarraySum(int[] arr) {

int result = arr[0];

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

int curSum = 0;

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

curSum = curSum + arr[j];

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

}

return result;

}

public static void main(String[] args) {

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

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

int[] arr1 = {-2,-4};

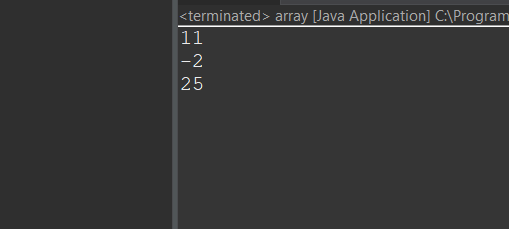
System.***out***.println(*maxSubarraySum*(arr1));

int[] arr2 = {1,4,1,7,8};

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

}

}



Time complexity:O(N^2)

2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray. Input: arr[] = {-2, 6, -3, -10, 0, 2} Output: 180 Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 \* (-3) \* (-10) = 180 Input: arr[] = {-1, -3, -10, 0, 60} Output: 60 Explanation: The subarray with maximum product is {60}.

CODE:

package util;

class product {

static int maxProductSubarray(int[] arr) {

int res = arr[0];

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

int currproduct = 1;

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

currproduct = currproduct \* arr[j];

res = Math.*max*(res, currproduct);

}

}

return res;

}

public static void main(String[] args) {

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

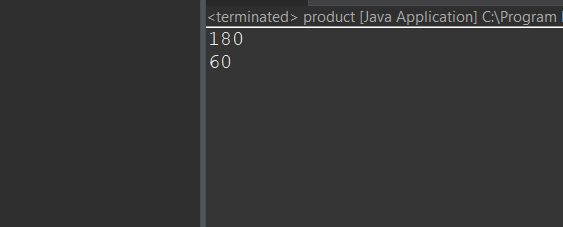
System.***out***.println(*maxProductSubarray*(arr));

int[] arr1 = {-1, -3, -10, 0, 60};

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

}

}



Time Complexity:o(N^2)

3.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. Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0 Output : 4

CODE:

package util;

public class sorted {

public static int searchInRotatedArray(int[] arr, int key) {

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

while (low <= high) {

int mid = (low + high) / 2;

if (arr[mid] == key) {

return mid;

}

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

if (arr[low] <= key && key < arr[mid]) {

high = mid - 1;

} else {

low = mid + 1;

}

}

else {

if (arr[mid] < key && key <= arr[high]) {

low = mid + 1;

} else {

high = mid - 1;

}

}

}

return -1;

}

public static void main(String[] args) {

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

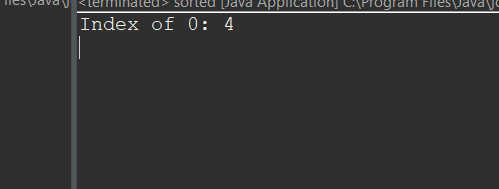
int key = 0;

int result = *searchInRotatedArray*(arr, key);

System.***out***.println("Index of " + key + ": " + result);

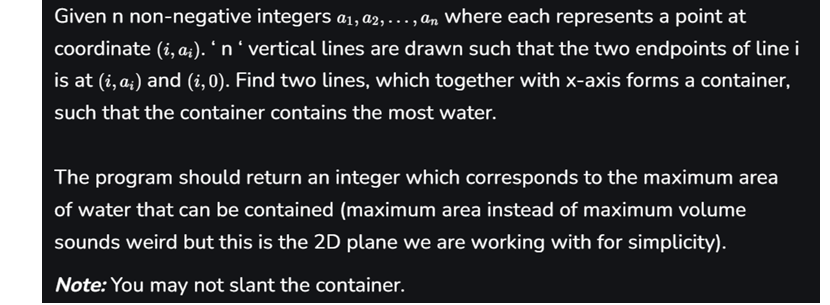
}

}



Time Complexity: O(log n)

4.Container with Most Water



Input: arr = [1, 5, 4, 3] Output: 6 Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 \* 2 = 6

CODE:

class Mostwater {

public static int container(int[] arr){

int left=0,right=arr.length-1,maxx=0;

while(left<right){

int width=right-left;

int height=Math.min(arr[left],arr[right]);

maxx=Math.max(maxx,width\*height);

if(arr[left]<arr[right]) left++;

else right--;

}

return maxx;

}

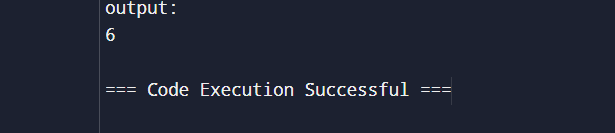
public static void main(String[] args) {

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

System.out.println(container(arr1));

}

}



Time Complexity:O(N)

5. Find the Factorial of a large number Input: 100 Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00

CODE:

package test;

import java.math.BigInteger;

import java.util.Scanner;

public class factorial {

public static void main(String[] args) {

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

System.***out***.print("Enter the value");

int n = sc.nextInt();

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

}

public static BigInteger fact(int n) {

BigInteger f = new BigInteger("1");

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

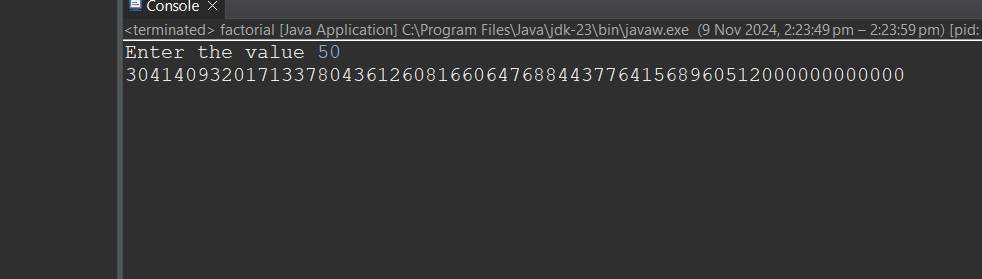
f=f.multiply(BigInteger.*valueOf*(i));

}

return f;

}

}



Time Complexity: O(N)

6. 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. Input: arr[] = {3, 0, 1, 0, 4, 0, 2} Output: 10 Explanation: The expected rainwater to be trapped is shown in the above image.

CODE:

package util;

public class trapping {

public static int trapRainwater(int[] arr) {

int n = arr.length;

if (n < 3) return 0;

int left = 0, right = n - 1;

int leftMax = 0, rightMax = 0;

int waterTrapped = 0;

while (left < right) {

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

if (arr[left] >= leftMax) {

leftMax = arr[left];

} else {

waterTrapped += leftMax - arr[left];

}

left++;

} else {

if (arr[right] >= rightMax) {

rightMax = arr[right];

} else {

waterTrapped += rightMax - arr[right];

}

right--;

}

}

return waterTrapped;

}

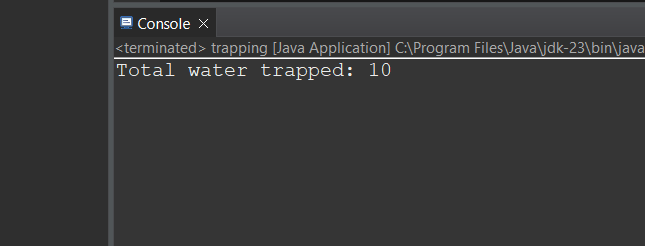
public static void main(String[] args) {

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

System.***out***.println("Total water trapped: " + *trapRainwater*(arr));

}

}



Time Complexity: O(N)

7. 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: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized. Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3 Output: 2 Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

CODE:

import java.util.Arrays;

class chocolate {

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

int n = arr.length;

Arrays.sort(arr);

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) {

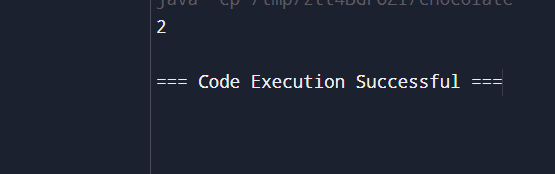
int[] arr = {7, 3, 2, 4, 9, 12, 56};

int m = 3;

System.out.println(findMinDiff(arr, m));

}

}



Time Complexity: n\*log(n)

8. 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. Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]] Output: [[1, 4], [6, 8], [9, 10]] Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].

CODE:

package util;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Comparator;

import java.util.List;

public class merge {

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

if (intervals.length <= 1) {

return intervals;

}

Arrays.*sort*(intervals, Comparator.*comparingInt*(a -> a[0]));

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

int[] curr = intervals[0];

merged.add(curr);

for (int[] interval : intervals) {

int currentEnd = curr[1];

int nextStart = interval[0];

int nextEnd = interval[1];

if (currentEnd >= nextStart) {

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

} else {

curr = interval;

merged.add(curr);

}

}

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

}

public static void main(String[] args) {

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

int[][] result = *merge*(intervals);

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

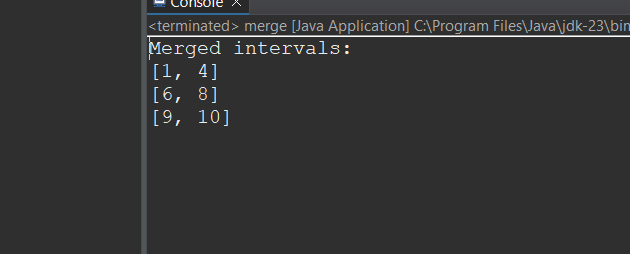
for (int[] interval : result) {

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

}

}

}



Time Complexity: O(nlogn)

9.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. Input: {{1, 0}, {0, 0}}

CODE:

package util;

public class booleanmatrix {

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

int M = mat.length;

int N = mat[0].length;

boolean[] rows = new boolean[M];

boolean[] cols = new boolean[N];

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

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

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

rows[i] = true;

cols[j] = true;

}

}

}

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

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

if (rows[i] || cols[j]) {

mat[i][j] = 1;

}

}

}

}

public static void main(String[] args) {

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

*modifyMatrix*(mat);

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

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

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

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

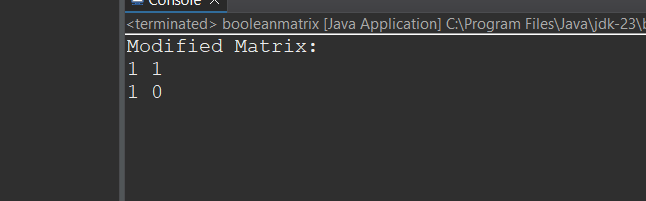
}

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

}

}

}



Time Complexity: O(M×N)

10.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. Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }} Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

CODE:

package util;

public class spiral {

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

int m = matrix.length;

int n = matrix[0].length;

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

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

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

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

}

top++;

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

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

}

right--;

if (top <= bottom) {

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

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

}

bottom--;

}

if (left <= right) {

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

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

}

left++;

}

}

}

public static void main(String[] args) {

int[][] matrix = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

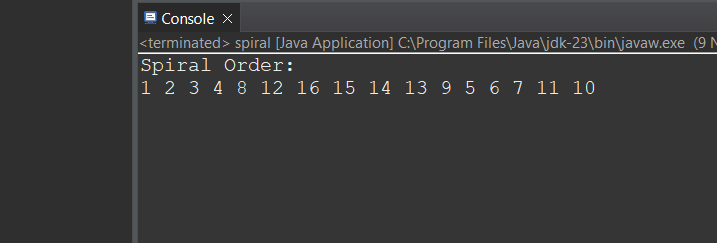
};

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

*printSpiral*(matrix);

}

}



Time Complexity: O(m×n)

13. 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. Input: str = “((()))()()” Output: Balanced Input: str = “())((())

CODE:

package util;

import java.util.Stack;

public class balanced{

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) {

String str = "((()))()()";

if (*isBalanced*(str)) {

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

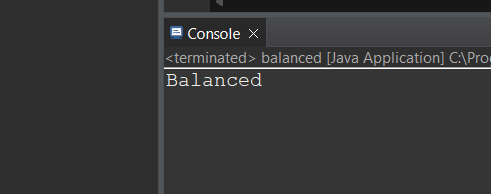
} else {

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

}

}

}



Time Complexity: O(N)

14. 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. Input: s1 = “geeks” s2 = “kseeg” Output: true Explanation: Both the string have same characters with same frequency. So, they are anagrams.

CODE:

package util;

public class anagram {

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

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

return false;

}

int[] charCount = new int[26];

for (int i = 0; i < s1.length(); i++) {

charCount[s1.charAt(i) - 'a']++;

charCount[s2.charAt(i) - 'a']--;

}

for (int count : charCount) {

if (count != 0) {

return false;

}

}

return true;

}

public static void main(String[] args) {

String s1 = "geeks";

String s2 = "kseeg";

if (*areAnagrams*(s1, s2)) {

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

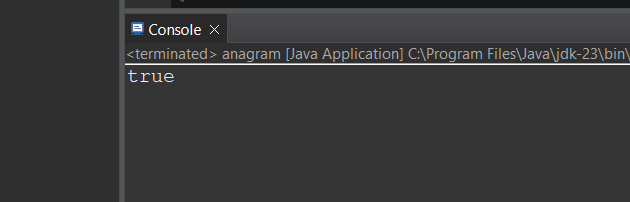
} else {

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

}

}

}



Time Complexity: O(N)

15.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. Input: str = “forgeeksskeegfor” Output: “geeksskeeg” Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

CODE:

package util;

public class palindrome {

public static String longestPalindrome(String str) {

if (str == null || str.length() < 1) {

return "";

}

int start = 0, end = 0;

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

int len1 = *expandAroundCenter*(str, i, i);

int len2 = *expandAroundCenter*(str, i, i + 1);

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

if (len > end - start) {

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

end = i + len / 2;

}

}

return str.substring(start, end + 1);

}

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

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

left--;

right++;

}

return right - left - 1;

}

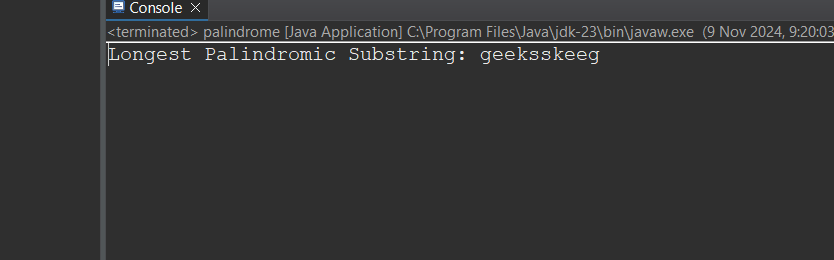
public static void main(String[] args) {

String str = "forgeeksskeegfor";

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

}

}



Time Complexity: O(N^2)

16.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”. Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”] Output: gee Explanation: “gee” is the longest common prefix in all the given strings.

CODE:

package util;

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 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);

}

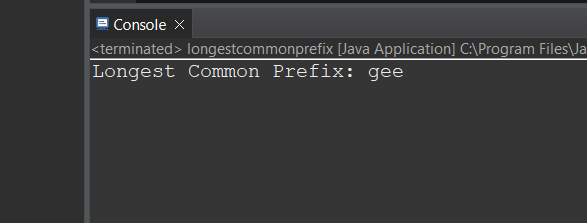
public static void main(String[] args) {

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

System.***out***.println("Longest Common Prefix: " + *longestCommonPrefix*(arr));

}

}



Time Complexity: O(NlogN+M)

17. 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. Input : Stack[] = [1, 2, 3, 4, 5] Output : Stack[] = [1, 2, 4, 5]

CODE:

package util;

import java.util.Stack;

import java.util.Vector;

public class deletemiddleelement {

public static void main(String[] args) {

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

st.push('1');

st.push('2');

st.push('3');

st.push('4');

st.push('5');

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

while (!st.empty()) {

v.add(st.pop());

}

int n = v.size();

if (n % 2 == 0) {

int target = (n / 2);

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

if (i == target) continue;

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

}

} else {

int target = (int) Math.*ceil*(n / 2);

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

if (i == target) continue;

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

}

}

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

while (!st.empty()) {

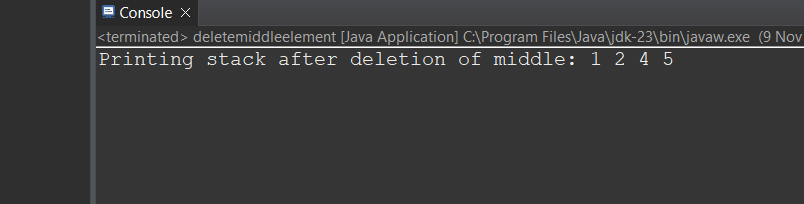
char p = st.pop();

System.***out***.print(p + " ");

}

}

}



Time Complexity: O(N)

18.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. Input: arr[] = [ 4 , 5 , 2 , 25 ] Output: 4 –> 5 5 –> 25 2 –> 25 25 –> -1 Explanation: Except 25 every element has an element greater than them present on the right side

CODE:

package util;

import java.util.Stack;

import java.util.HashMap;

public class greaterele {

public static void printNextGreaterElements(int[] arr) {

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

HashMap<Integer, Integer> ngeMap = new HashMap<>();

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

int currentElement = arr[i];

while (!stack.isEmpty() && stack.peek() <= currentElement) {

stack.pop();

}

int nextGreater = stack.isEmpty() ? -1 : stack.peek();

ngeMap.put(currentElement, nextGreater);

stack.push(currentElement);

}

for (int element : arr) {

System.***out***.println(element + " -> " + ngeMap.get(element));

}

}

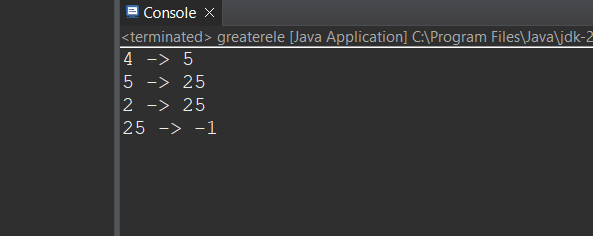
public static void main(String[] args) {

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

*printNextGreaterElements*(arr);

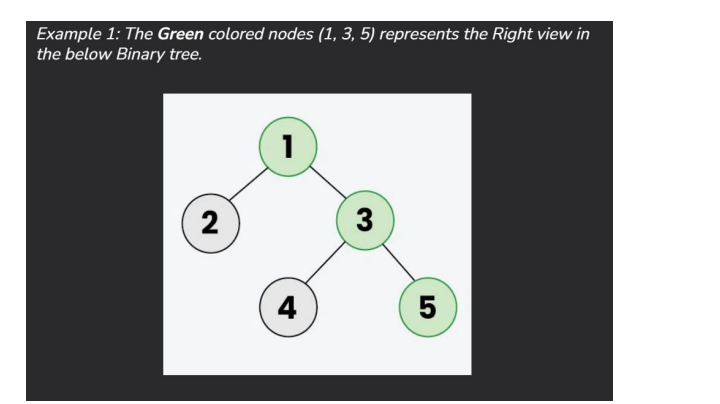
}

}



Time Complexity: O(N)

19.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.List;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

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

}

class Solution {

int maxlevel = 0;

public List<Integer> rightSideView(TreeNode root) {

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

right(root, 1, list);

return list;

}

void right(TreeNode root, int level, List<Integer> list) {

if (root == null) {

return;

}

if (maxlevel < level) {

list.add(root.val);

maxlevel = level;

}

right(root.right, level + 1, list);

right(root.left, level + 1, list);

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

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

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

Solution solution = new Solution();

List<Integer> result = solution.rightSideView(root);

System.out.println(result);

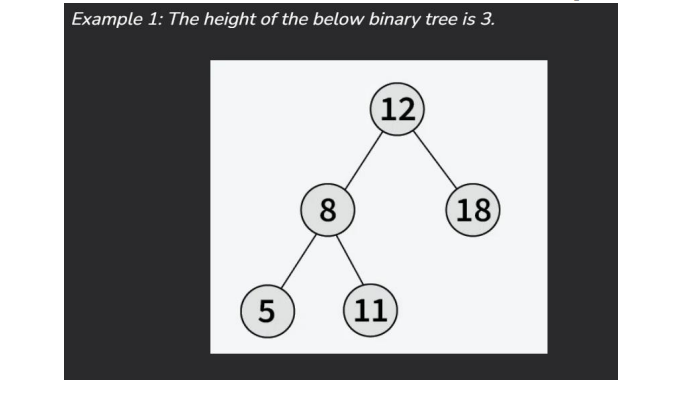
}

}



Time Complexity:O(N)

20. 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:

class TreeNode {

int val;

TreeNode left;

TreeNode right;

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

}

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) return 0;

int left = maxDepth(root.left);

int right = maxDepth(root.right);

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

}

public static void main(String[] args) {

TreeNode root = new TreeNode(3);

root.left = new TreeNode(9);

root.right = new TreeNode(20);

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

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

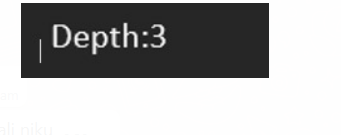
Solution solution = new Solution();

int result = solution.maxDepth(root);

System.out.println(result);

}

}



TimeComplexity: O(n)