

09/11/2024

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.

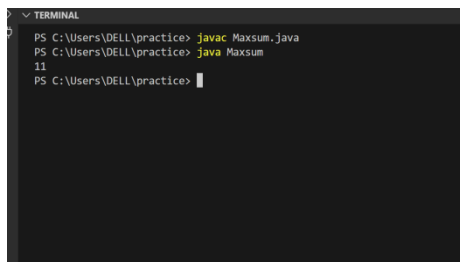
```
import java.util.Arrays;

class Maxsum {
    public static void main(String[] args) {
        int[] arr = {2, 3, -8, 7, -1, 2, 3};
        int res = arr[0];
        int curr = arr[0];

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

            curr = Math.max(arr[i], curr + arr[i]);
            res = Math.max(res, curr);
        }

        System.out.println(res);
    }
}
```



```
PS C:\Users\DELL\practice> javac Maxsum.java
PS C:\Users\DELL\practice> java Maxsum
11
PS C:\Users\DELL\practice>
```

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 = $6 * (-3) * (-10) = 180$

```
public class Maxprod {
    public static void main(String[] args) {
        int arr[] = {-2, 6, -3, -10, 0, 2};
        int maxprod = arr[0];
        int currmin = arr[0];
```

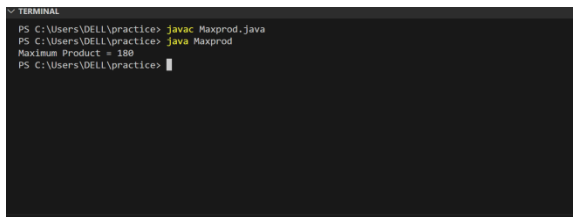
```

int currmax = arr[0];
int n = arr.length;

for (int i = 1; i < n; i++) {
    if (arr[i] < 0) {
        int temp = currmax;
        currmax = currmin;
        currmin = temp;
    }
    currmax = Math.max(arr[i], currmax * arr[i]);
    currmin = Math.min(arr[i], currmin * arr[i]);
    maxprod = Math.max(maxprod, currmax);
}

System.out.println("Maximum Product = " + maxprod);
}
}

```



```

TERMINAL
PS C:\Users\DELL\practice> javac Maxprod.java
PS C:\Users\DELL\practice> java Maxprod
Maximum Product = 180
PS C:\Users\DELL\practice>

```

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

```

import java.util.Scanner;

public class BinarySearch {
    public static int binarySearchRotated(int[] arr, int target) {
        int low = 0, high = arr.length - 1;

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

            if (arr[mid] == target) {
                return mid;
            }

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

```

```

        if (arr[low] <= target && target < arr[mid]) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    } else {
        if (arr[mid] < target && target <= arr[high]) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
}

return -1;
}

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

    System.out.print("Enter n: ");
    int n = sc.nextInt();

    int[] arr = new int[n];
    System.out.println("Enter array elements:");
    for (int i = 0; i < n; i++) {
        arr[i] = sc.nextInt();
    }

    System.out.print("Enter target: ");
    int target = sc.nextInt();

    int result = binarySearchRotated(arr, target);
    if (result != -1) {
        System.out.println("Element found at index: " + result);
    } else {
        System.out.println("Element not found.");
    }
}

```

```

PS C:\Users\DELL\practice> javac BinarySearch.java
PS C:\Users\DELL\practice> java BinarySearch
Enter n: 7
Enter array elements:
4 5 6 7 0 1 2
Enter target: 0
Element found at index: 4
PS C:\Users\DELL\practice>

```

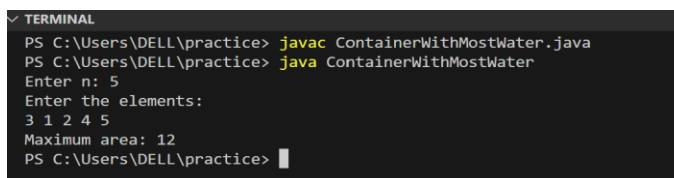
4. Container with Most Water

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 = $3 * 2 = 6$

```
import java.util.Scanner;

class ContainerWithMostWater {
    static int maxArea(int[] arr) {
        int n = arr.length;
        int area = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                area = Math.max(area, Math.min(arr[j], arr[i]) * (j - i));
            }
        }
        return area;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter n: ");
        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("Maximum area: " + maxArea(arr));
    }
}
```



```
✓ TERMINAL
PS C:\Users\DELL\practice> javac ContainerWithMostWater.java
PS C:\Users\DELL\practice> java ContainerWithMostWater
Enter n: 5
Enter the elements:
3 1 2 4 5
Maximum area: 12
PS C:\Users\DELL\practice> █
```

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

93326215443944152681699238856266700490715968264381621468592963895217599993
2299
15608941463976156518286253697920827223758251185210916864000000000000000000
0000 00

```
import java.math.BigInteger;
```

```

public class Fact {
    static BigInteger calculateFact(int n, BigInteger[] memo) {
        if (memo[n] != null) {
            return memo[n];
        }
        if (n <= 1) {
            return BigInteger.ONE;
        }
        BigInteger result = BigInteger.valueOf(n).multiply(calculateFact(n - 1, memo));
        memo[n] = result;
        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();
        BigInteger[] memo = new BigInteger[n + 1];
        for (int i = 0; i <= n; i++) {
            memo[i] = null;
        }
        BigInteger result = calculateFact(n, memo);
        System.out.println("Factorial of " + n + " is: ");
        System.out.println(result);
    }
}

```

```
PS C:\Users\DELL\practice> javac Fact.java
PS C:\Users\DELL\practice> java Fact
Enter a number: 100
Factorial of 100 is:
9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369
79208272237582511852109168640000000000000000000000000000
PS C:\Users\DELL\practice>
```

- ```
import java.util.Scanner;
```

```
public class TrappingRainwater {
 public static int trap(int[] arr) {
 int n = arr.length;
```

```

 if (n == 0) return 0;

 int[] left_max = new int[n];
 int[] right_max = new int[n];
 int waterTrapped = 0;

 left_max[0] = arr[0];
 for (int i = 1; i < n; i++) {
 left_max[i] = Math.max(arr[i], left_max[i - 1]);
 }

 right_max[n - 1] = arr[n - 1];
 for (int i = n - 2; i >= 0; i--) {
 right_max[i] = Math.max(arr[i], right_max[i + 1]);
 }

 for (int i = 0; i < n; i++) {
 waterTrapped += Math.min(left_max[i], right_max[i]) - arr[i];
 }

 return waterTrapped;
}

public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.print("Enter n: ");
 int n = sc.nextInt();
 int[] arr = new int[n];
 System.out.println("Enter elements:");
 for (int i = 0; i < n; i++) {
 arr[i] = sc.nextInt();
 }
 System.out.println("Water trapped: " + trap(arr));
}
}

```

```

✓ TERMINAL
PS C:\Users\DELL\practice> java TrappingRainwater
Enter n: 7
Enter elements:
3 0 1 0 4 0 2
Water trapped: 10
PS C:\Users\DELL\practice>

```

**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.

```
import java.util.Arrays;

public class Problem7 {
 static int chocolateDistribution(int[] arr,int m){
 int n = arr.length;
 if(n<m){
 return -1;
 }
 if(n==0 || m==0){
 return 0;
 }
 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];
 minDiff = Math.min(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(chocolateDistribution(arr,m));
 }
}
```

#### ✓ TERMINAL

```
PS C:\Users\DELL\practice> java Problem7
2
PS C:\Users\DELL\practice>
```

**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]]`

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

public class MergeIntervals {
 public static int[][] merge(int[][] intervals) {
 Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
 List<int[]> merged = new ArrayList<>();
 for (int[] interval : intervals) {
 if (merged.isEmpty() || merged.get(merged.size() - 1)[1] < interval[0]) {
 merged.add(interval);
 } else {
 merged.get(merged.size() - 1)[1] = Math.max(merged.get(merged.size() - 1)[1],
interval[1]);
 }
 }
 return merged.toArray(new int[merged.size()][]);
 }

 public static void main(String[] args) {
 int[][] intervals1 = { {1, 3}, {2, 4}, {6, 8}, {9, 10} };
 int[][] result1 = merge(intervals1);
 System.out.println("Merged intervals: " + Arrays.deepToString(result1));

 int[][] intervals2 = { {7, 8}, {1, 5}, {2, 4}, {4, 6} };
 int[][] result2 = merge(intervals2);
 System.out.println("Merged intervals: " + Arrays.deepToString(result2));
 }
}
```

```
▼ TERMINAL
PS C:\Users\DELL\practice> javac MergeIntervals.java
PS C:\Users\DELL\practice> java MergeIntervals
Merged intervals: [[1, 4], [6, 8], [9, 10]]
Merged intervals: [[1, 6], [7, 8]]
PS C:\Users\DELL\practice> █
```



**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 `i`th row and `j`th column as 1.  
Input: `{{1, 0}, {0, 0}}` Output: `{{1, 1} {1, 0}}` Input: `{{0, 0, 0}, {0, 0, 1}}` Output: `{{0, 0, 1}, {1, 1, 1}}`

```
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);

 printMatrix(mat);

 }

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

 for (int[] row : mat) {

 for (int val : row) {

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

 }

 System.out.println();

 }

 System.out.println();

 }

}

```

```

TERMINAL

PS C:\Users\DELL\practice> java BooleanMatrix
1 1
1 0

PS C:\Users\DELL\practice>

```

**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 }}

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

```

public class SpiralOrderMatrix {

 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 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 matrix elements:");

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

 List<Integer> result = spiralOrder(matrix);

 for (int num : result) {
 System.out.print(num + " ");
 }
}

```

```
}

}
```

```
✓ TERMINAL
PS C:\Users\DELL\practice> java SpiralOrderMatrix
Enter the number of rows: 4
Enter the number of columns: 4
Enter the matrix elements:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
PS C:\Users\DELL\practice> █
```

**11.** 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 = “()()((())” Output: Not Balanced

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

```
class BalanceParenthesis{
```

```
 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;
}

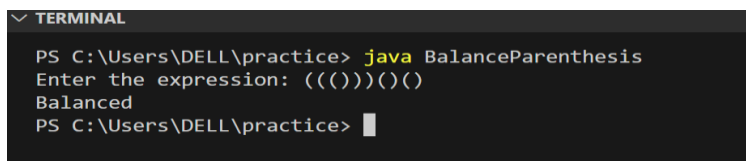
return flag;
}

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

 System.out.print("Enter the expression: ");
 String exp = scanner.nextLine();

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

```



```

▼ TERMINAL
PS C:\Users\DELL\practice> java BalanceParenthesis
Enter the expression: ((())) () ()
Balanced
PS C:\Users\DELL\practice>

```

**12.** 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.

```

import java.util.Arrays;

import java.util.Scanner;

class Anagram {

 static boolean areAnagrams(String s1, String s2) {

 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 first string: ");

 String s1 = scanner.nextLine();

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

 String s2 = scanner.nextLine();

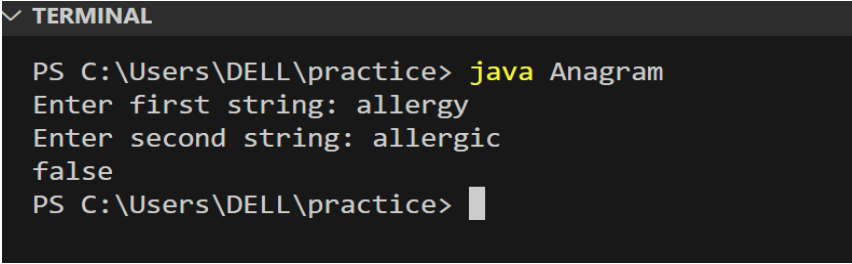
 System.out.println(areAnagrams(s1, s2));

 scanner.close();

 }

}

```



```

✓ TERMINAL
PS C:\Users\DELL\practice> java Anagram
Enter first string: allergy
Enter second string: allergic
false
PS C:\Users\DELL\practice>

```

**13. 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", "eeksskeeg" etc. But the substring "geeksskeeg" is the longest among all

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

```
public class LongestPalindromicSubstring {

 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 a string: ");

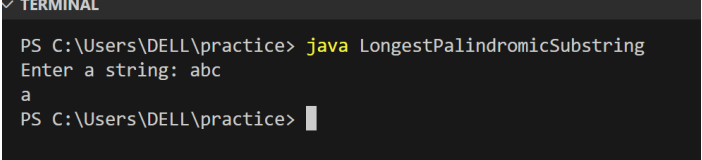
 String s = scanner.nextLine();

 System.out.println(longestPalSubstr(s));

 scanner.close();

 }
}

```



```

✓ TERMINAL
PS C:\Users\DELL\practice> java LongestPalindromicSubstring
Enter a string: abc
a
PS C:\Users\DELL\practice>

```

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

```

import java.util.Arrays;

import java.util.Scanner;

class LongestCommonPrefix {

 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();

 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(longestCommonPrefix(arr));

 scanner.close();
}
}

```

```
✓ TERMINAL
PS C:\Users\DELL\practice> java LongestCommonPrefix
Enter the number of strings: 2
Enter string 1: hello
Enter string 2: world
-1
PS C:\Users\DELL\practice> █
```

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]

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

```
import java.util.Vector;
```

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

```
public class DeleteMidOfStack {
```

```
 public static void main(String[] args) {
```

```
 Scanner scanner = new Scanner(System.in);
```

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

```
 System.out.print("Enter elements for stack (no spaces): ");
```

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

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

```
 st.push(ch);
```

```
 }
```

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

```
 while (!st.empty()) {
```

```
 v.add(st.pop());
```

```
 }
```

```
 int n = v.size();
```

```
 int target = (n % 2 == 0) ? (n / 2) : (int) Math.ceil(n / 2.0) - 1;
```

```

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

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

 while (!st.empty()) {
 System.out.print(st.pop() + " ");
 }

 scanner.close();
}
}

```

```

PS C:\Users\DELL\practice> javac DeleteMidOfStack.java
PS C:\Users\DELL\practice> java DeleteMidOfStack
Enter elements for stack (no spaces): 12345
1 2 4 5
PS C:\Users\DELL\practice>

```

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

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

```

class NextGreaterElement {

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

 int next;

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

 next = -1;

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

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

```

```

 next = arr[j];

 break;
 }
}

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

}

}

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();

 }

 printNGE(arr, n);

 scanner.close();

}

}

```

```

PS C:\Users\DELL\practice> javac NextGreaterElement.java
PS C:\Users\DELL\practice> java NextGreaterElement
Enter the number of elements: 4
Enter the elements:
4 5 2 25
4 -- 5
5 -- 25
2 -- 25
25 -- -1
PS C:\Users\DELL\practice>

```

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

```
import java.util.ArrayList;
```

```
class Node {
```

```
 int data;
```

```
 Node left, right;
```

```
 Node(int x) {
```

```
 data = x;
```

```
 left = right = null;
```

```
 }
```

```
}
```

```
class PrintRightView{
```

```
 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};

RecursiveRightView(root, 0, maxLevel, result);

return result;
}

```

```

static void printArray(ArrayList<Integer> arr) {

 for (int val : arr) {

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

 }

 System.out.println();

}

```

```

public static void main(String[] args) {

 Node root = new Node(1);

 root.left = new Node(2);

 root.right = new Node(3);

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

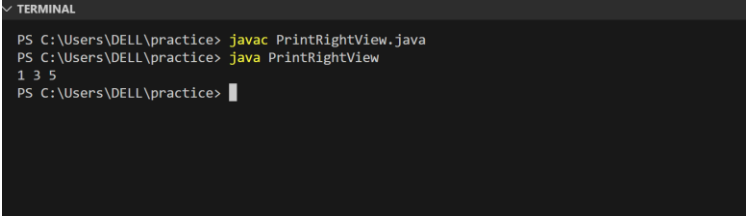
 root.right.right = new Node(5);

 ArrayList<Integer> result = rightView(root);

 printArray(result);

}
}

```



```

TERMINAL
PS C:\Users\DELL\practice> javac PrintRightView.java
PS C:\Users\DELL\practice> java PrintRightView
1 3 5
PS C:\Users\DELL\practice>

```

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

```
class Node {

 int data;

 Node left, right;

 Node(int val) {
 data = val;
 left = null;
 right = null;
 }
}
```

```
class MaximumDepth {

 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;
 }
}
```

```
public static void main(String[] args) {
 Node root = new Node(1);
 root.left = new Node(2);
 root.right = new Node(3);
 root.left.left = new Node(4);
 root.left.right = new Node(5);
}
```



```
 System.out.println(maxDepth(root));
 }
}
```

```
PS C:\Users\DELL\practice> javac MaximumDepth.java
PS C:\Users\DELL\practice> java MaximumDepth
3
PS C:\Users\DELL\practice> |
```