Ezhilarasan R-AI&DS_DSA_PRACTICE-1

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:

```
import java.util.Scanner;
public class MaxSubarraySum {
   public static void main(String[] args) {
     Scanner input = new Scanner(System.in);
     System.out.print("Enter the number of elements: ");
       int n = input.nextInt();
       int[] arr = new int[n];
       System.out.println("Enter the elements:");
       for (int i = 0; i < n; i++) {
           arr[i] = input.nextInt();
       }
       int maxSum = arr[0];
       int currentSum = arr[0];
       for (int i = 1; i < n; i++) {
           currentSum = Math.max(arr[i], currentSum + arr[i]);
           maxSum = Math.max(maxSum, currentSum);
       }
       System.out.println("Maximum Subarray Sum: " + maxSum);
       input.close();
}
```

Output:

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\Desktop\code>javac MaxSubarraySum.java

C:\Users\ASUS\Desktop\code>java MaxSubarraySum
Enter the number of elements: 2
Enter the elements:
-2
-4
Maximum Subarray Sum: -2

C:\Users\ASUS\Desktop\code>
```

Time complexity:O(n)

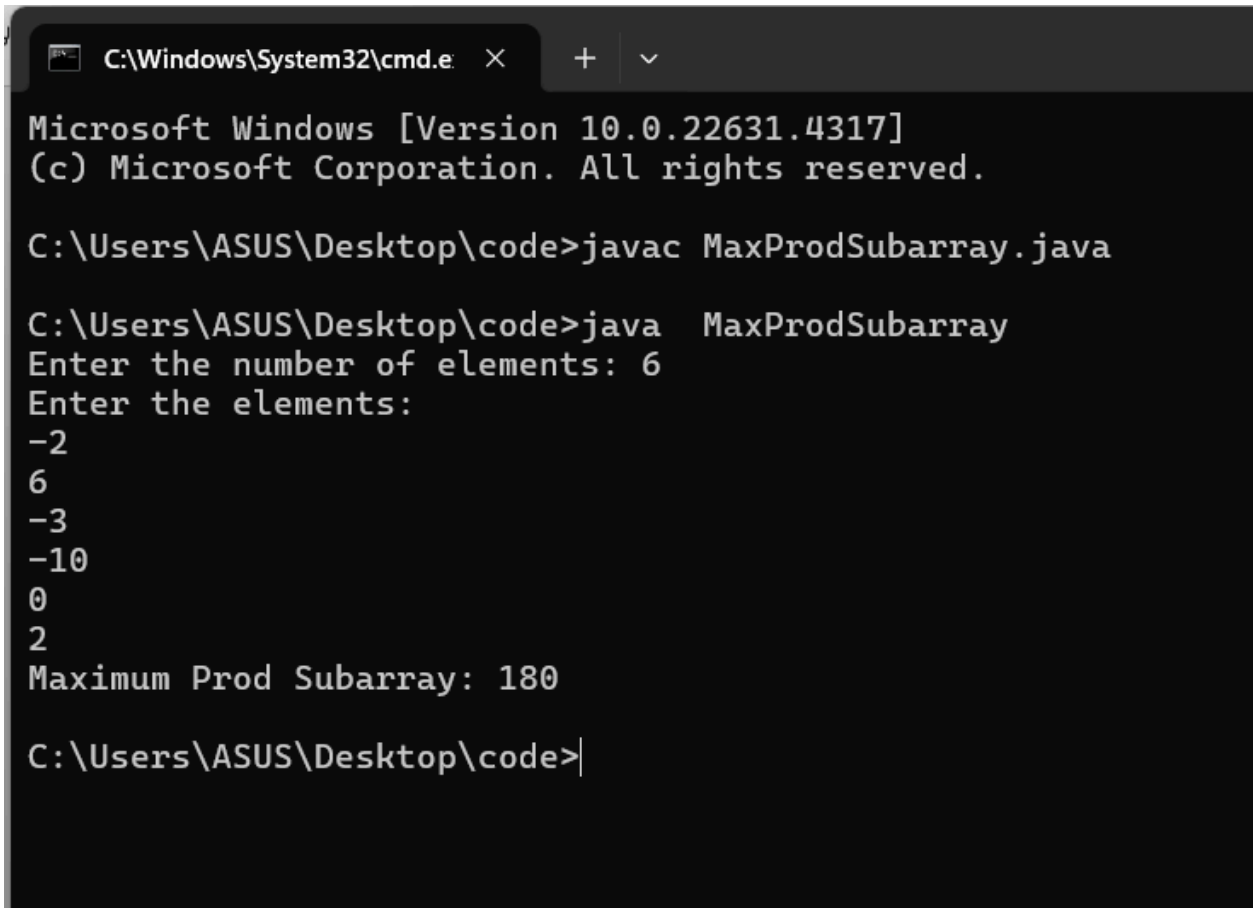
2)Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray. Input: arr[] = {-2, 6, -3, -10, 0, 2} Output: 180 Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 * (-3) * (-10) = 180 Input: arr[] = {-1, -3, -10, 0, 60} Output: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

Code:

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

```
        int maxProduct = arr[0];
        int minProduct = arr[0];
        int result = arr[0];
        for (int i = 1; i < n; i++) {
            int temp = maxProduct;
            maxProduct = Math.max(arr[i], Math.max(maxProduct * arr[i], minProduct *
arr[i]));
            minProduct = Math.min(arr[i], Math.min(temp * arr[i], minProduct * arr[i]));
            result = Math.max(result, maxProduct);
        }
        System.out.println("Maximum Prod Subarray: " + result);
        input.close();
    }
}
```

Output:



Time complexity:O(n)

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 Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3 Output : -1 Input : arr[] = {50, 10, 20, 30, 40}, key = 10 Output : 1

Code:

```
import java.util.Scanner;
public class SearchRotarr {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the no. of elements: ");
        int n = input.nextInt();
        int[] arr = new int[n];

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

        System.out.print("Enter  key to search: ");
        int key = input.nextInt();
        int index = search(arr, 0, n - 1, key);
        System.out.println("Index of key: " + index);
        input.close();
    }

    public static int search(int[] arr, int left, int right, int key) {
        if (left > right)
return -1;

        int mid = left + (right - left) / 2;

        if (arr[mid] == key) return mid;

        if (arr[left] <= arr[mid]) {
            if (key >= arr[left] && key < arr[mid]) {
                return search(arr, left, mid - 1, key);
            } else {
                return search(arr, mid + 1, right, key);
            }
        } else {
```

```
            if (key > arr[mid] && key <= arr[right]) {
                return search(arr, mid + 1, right, key);
            } else {
                return search(arr, left, mid - 1, key);
            }
        }
    }
}
```

Output:

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\Desktop\code>javac SearchRotarr.java

C:\Users\ASUS\Desktop\code>java  SearchRotarr.java
Enter the no. of elements: 7
Enter the elements:
4
5
6
7
0
1
2
Enter  key to search: 0
Index of key: 4

C:\Users\ASUS\Desktop\code>
```

Time complexity:O(logn)

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 Input: arr = [3, 1, 2, 4, 5] Output: 12 Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 * 3 = 12

Given n non-negative integers $a_1, a_2, \ldots, a_n$ where each represents a point at coordinate (i, a_i). ‘ n ‘ vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

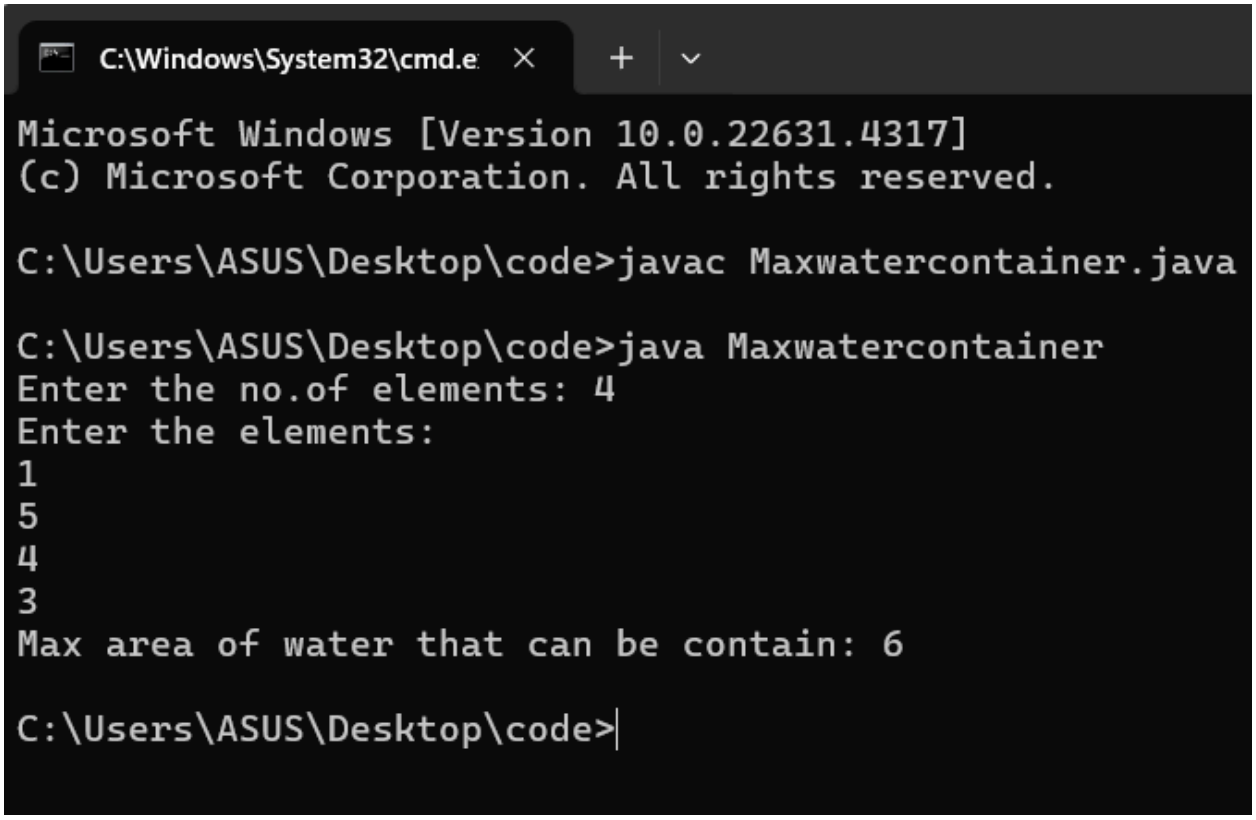
The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is the 2D plane we are working with for simplicity).

Note: You may not slant the container.

Code:

```
import java.util.Scanner;
public class Maxwatercontainer {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the no.of elements: ");
        int n = input.nextInt();
        int[] height = new int[n];

        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            height[i] = input.nextInt();
        }

        int maxArea = maxWaterContainer(height);
        System.out.println("Max area of water that can be contain: " + maxArea);
        input.close();
    }

    public static int maxWaterContainer(int[] height) {
        int left = 0, right = height.length - 1;
        int maxArea = 0;

        while (left < right) {
            int currentArea = Math.min(height[left], height[right]) * (right - left);
            maxArea = Math.max(maxArea, currentArea);

            if (height[left] < height[right]) {
                left++;
```

```
        } else {
            right--;
        }
    }

    return maxArea;
  }
}
```



Time complexity:O(n)

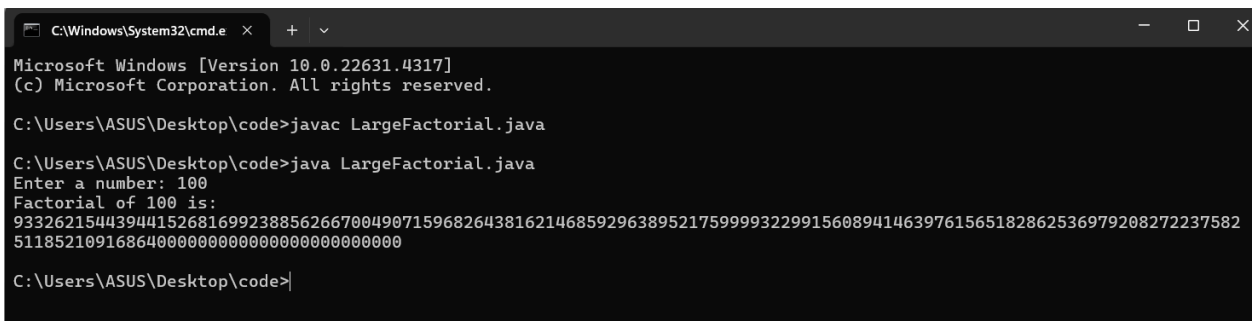
5) Find the Factorial of a large number Input: 100 Output: 93326215443944152681699238856266700490715968264381621468592963895217599932299 15608941463976156518286253697920827223758251185210916864000000000000000000000000 Input: 50 Output: 30414093201713378043612608166064768844377641568960512000000000000

Code:

```
import java.math.BigInteger;
import java.util.Scanner;
public class LargeFactorial {
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = input.nextInt();

        BigInteger factorial = BigInteger.ONE;

        for (int i = 2; i <= n; i++) {
            factorial = factorial.multiply(BigInteger.valueOf(i));
        }

        System.out.println("Factorial of " + n + " is:");
        System.out.println(factorial);
        input.close();
    }
}
```

Output:



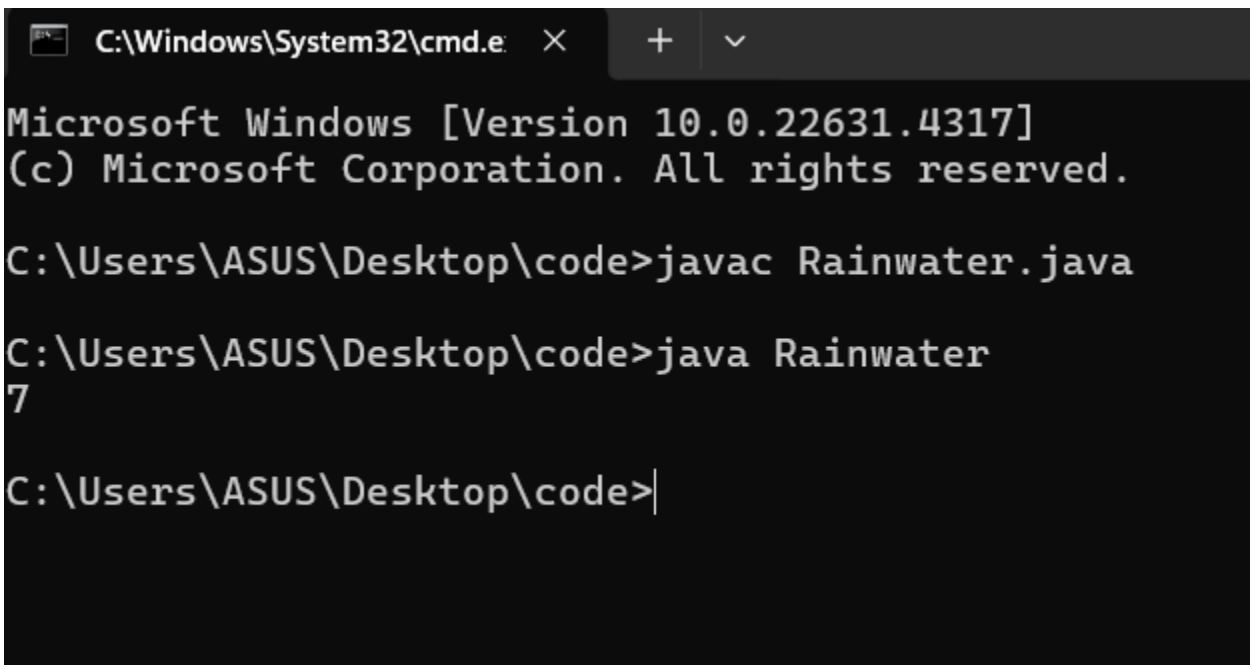
Time complexity:**O(n log n)**.

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. Input: arr[] = {3, 0, 2, 0, 4} Output: 7 Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units. Input: arr[] = {1, 2, 3, 4} Output: 0 Explanation : We cannot trap water as there is no height bound on both sides

Code:

```
import java.util.*;

public class Rainwater {
    public static int trap(int[] arr) {
        int n = arr.length;
        if (n == 0) return 0;
        int left = 0, right = n - 1, leftMax = 0, rightMax = 0, result = 0;
        while (left <= right) {
```

```
        if (arr[left] <= arr[right]) {
            if (arr[left] >= leftMax) leftMax = arr[left];
            else result += leftMax - arr[left];
            left++;
        } else {
            if (arr[right] >= rightMax) rightMax = arr[right];
            else result += rightMax - arr[right];
            right--;
        }
    }
    return result;
  }

  public static void main(String[] args) {
    int[] arr = {3, 0, 2, 0, 4};
    System.out.println(trap(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;

public class ChocolateDistribution {
    public static int distribute(int[] arr, int m) {
        if (m > arr.length) return -1;
        Arrays.sort(arr);
        int minDiff = Integer.MAX_VALUE;
        for (int i = 0; i + m - 1 < arr.length; 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(distribute(arr, m));
    }
}
```

Time complexity:O(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]]. Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]] Output: [[1, 6], [7, 8]] Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval [1, 6].

Code:

```
import java.util.*;

public class MergeIntervals {
    public static List<int[]> merge(int[][] intervals) {
        if (intervals.length == 0) return new ArrayList<>();
        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
        List<int[]> merged = new ArrayList<>();
        merged.add(intervals[0]);
        for (int i = 1; i < intervals.length; i++) {
            int[] last = merged.get(merged.size() - 1);
            if (last[1] >= intervals[i][0]) {
                last[1] = Math.max(last[1], intervals[i][1]);
            } else {
                merged.add(intervals[i]);
            }
        }
        return merged;
    }

    public static void main(String[] args) {
        int[][] intervals = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        List<int[]> result = merge(intervals);
        for (int[] interval : result) {
            System.out.println(Arrays.toString(interval));
        }
    }
}
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\Desktop\code>javac  MergeIntervals.java

C:\Users\ASUS\Desktop\code>java  MergeIntervals
[1, 4]
[6, 8]
[9, 10]

C:\Users\ASUS\Desktop\code>
```

Time complexity:O(n logn)

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}} Output: {{1, 1} {1, 0}} Input: {{0, 0, 0}, {0, 0, 1}} Output: {{0, 0, 1}, {1, 1, 1}} Input: {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}} Output: {{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 0, 1, 1}}

Code:

```
import java.util.*;

public class BooleanMatrix {
    public static void modifyMatrix(int[][] mat) {
        int m = mat.length;
        int n = mat[0].length;
        boolean[] row = new boolean[m];
        boolean[] col = new boolean[n];


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

```
        }


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

    public static void main(String[] args) {
        int[][] mat1 = {{1, 0}, {0, 0}};
        int[][] mat2 = {{0, 0, 0}, {0, 0, 1}};
        int[][] mat3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};

        modifyMatrix(mat1);
        modifyMatrix(mat2);
        modifyMatrix(mat3);

        System.out.println(Arrays.deepToString(mat1));
        System.out.println(Arrays.deepToString(mat2));
        System.out.println(Arrays.deepToString(mat3));
    }
}
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\Desktop\code>javac BooleanMatrix.java

C:\Users\ASUS\Desktop\code>java BooleanMatrix
[[1, 1], [1, 0]]
[[0, 0, 1], [1, 1, 1]]
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 0, 1, 1]]

C:\Users\ASUS\Desktop\code>
```

Time complexity:O(M*N)

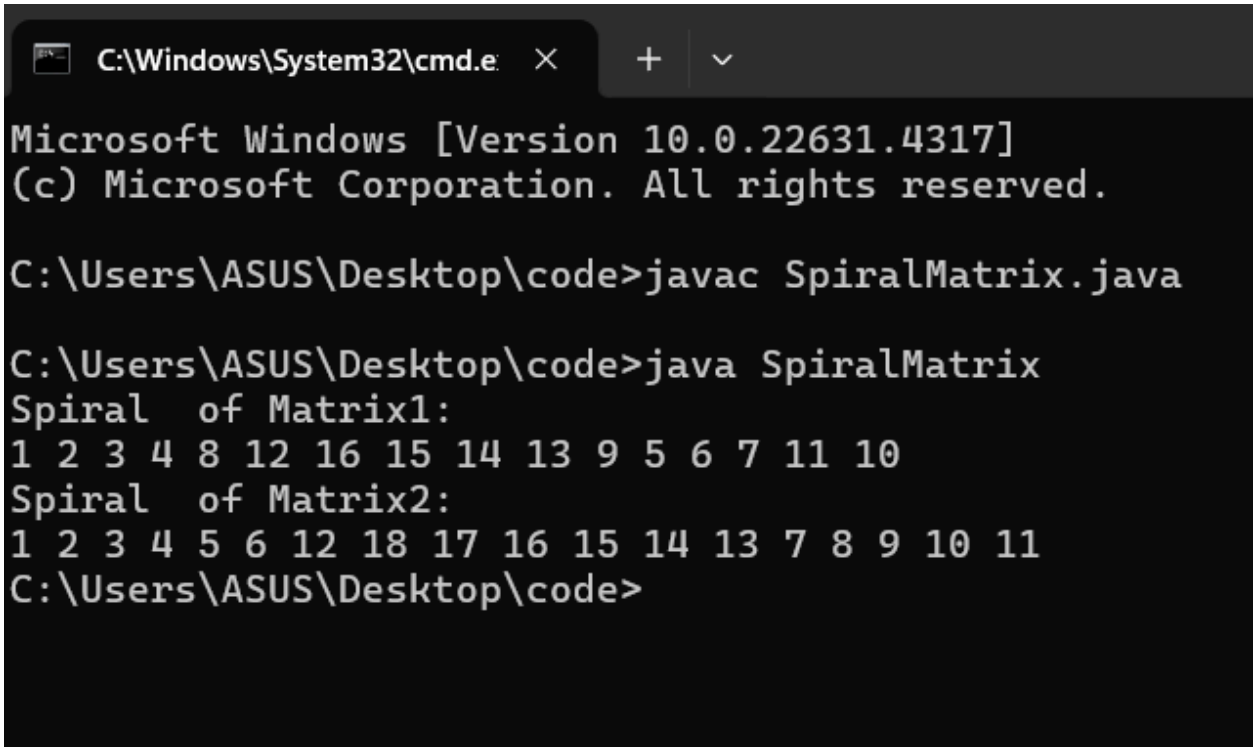
10) Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form. Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }} Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10 Input: matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}} Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11 Explanation: The output is matrix in spiral format.

Code:

```
public class SpiralMatrix {
public static void printSpiral(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        int top = 0, left = 0, bottom = m - 1, 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[][] matrix1 = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };

    int[][] matrix2 = {
        {1, 2, 3, 4, 5, 6},
        {7, 8, 9, 10, 11, 12},
        {13, 14, 15, 16, 17, 18}
    };

    System.out.println("Spiral  of Matrix1:");
    printSpiral(matrix1);
    System.out.println();

    System.out.println("Spiral  of Matrix2:");
    printSpiral(matrix2);
```

```
    }
}
```



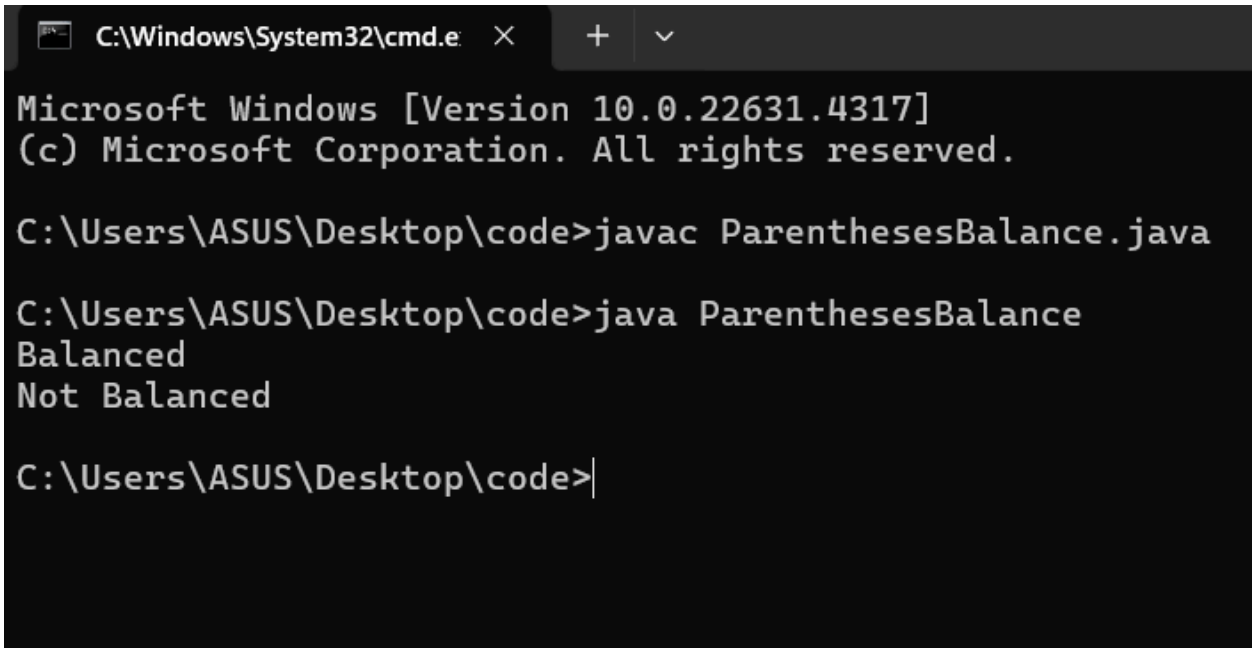
time:O(m*n)

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 = “())((())” Output: Not Balanced

Code:

```
public class ParenthesesBalance {
    public static String checkBalanced(String str) {
        int count = 0;
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == '(') {
                count++;
            } else if (str.charAt(i) == ')') {
                count--;
            }

            if (count < 0) {
                return "Not Balanced";
            }
        }
       return (count == 0) ? "Balanced" : "Not Balanced";
```

```
    }

    public static void main(String[] args) {
        String str1 = "((()))()()";
        String str2 = "())((())";

        System.out.println(checkBalanced(str1));
        System.out.println(checkBalanced(str2));
    }
}
```



tiem: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. Input: s1 = “allergy” s2 = “allergic” Output: false Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has extra characters „i‟ and „c‟, so they are not anagrams. Input: s1 = “g”, s2 = “g” Output: true Explanation: Characters in both the strings are same, so they are anagrams.

Code:

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

```java
public class AnagramChecker {
    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) {
        String s1 = "geeks";
        String s2 = "kseeg";

        String s3 = "allergy";
        String s4 = "allergic";

        String s5 = "g";
        String s6 = "g";

        System.out.println(areAnagrams(s1, s2));
        System.out.println(areAnagrams(s3, s4));
        System.out.println(areAnagrams(s5, s6));
    }
}
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\Desktop\code>javac AnagramChecker.java

C:\Users\ASUS\Desktop\code>java AnagramChecker
true
false
true

C:\Users\ASUS\Desktop\code>
```

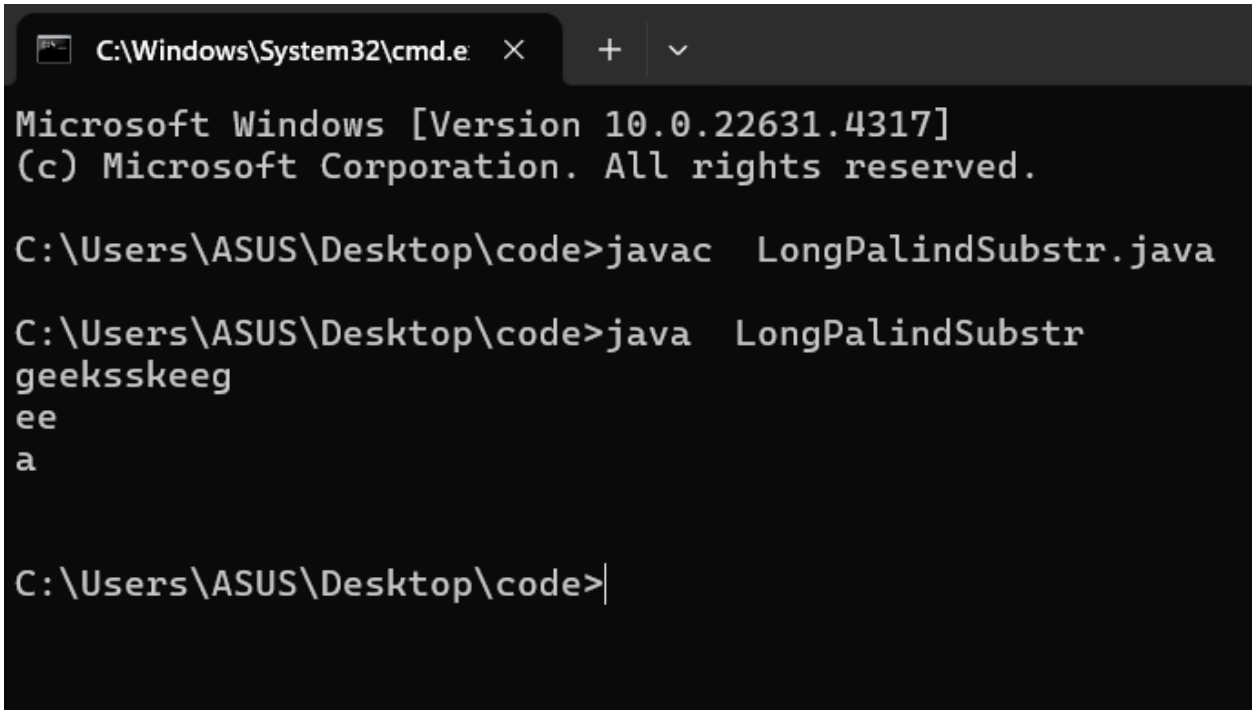
time:O(n logn)

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. Input: str = “Geeks” Output: “ee” Input: str = “abc” Output: “a” Input: str = “” Output: “”

Code:

```
public class LongPalindSubstr {
public static String longestPalindrome(String str) {
    if (str == null || str.length() == 0) {
        return "";
    }

    int start = 0, maxLength = 1;

    for (int i = 0; i < str.length(); i++) {
        for (int j = i; j < str.length(); j++) {
            if (isPalindrome(str, i, j) && j - i + 1 > maxLength) {
                start = i;
                maxLength = j - i + 1;
            }
        }
```

```
        }

        return str.substring(start, start + maxLength);
    }

    public static boolean isPalindrome(String str, int left, int right) {
        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    public static void main(String[] args) {
        String str1 = "forgeeksskeegfor";
        String str2 = "Geeks";
        String str3 = "abc";
        String str4 = "";

        System.out.println(longestPalindrome(str1));
        System.out.println(longestPalindrome(str2));
        System.out.println(longestPalindrome(str3));
        System.out.println(longestPalindrome(str4));
    }
}
```



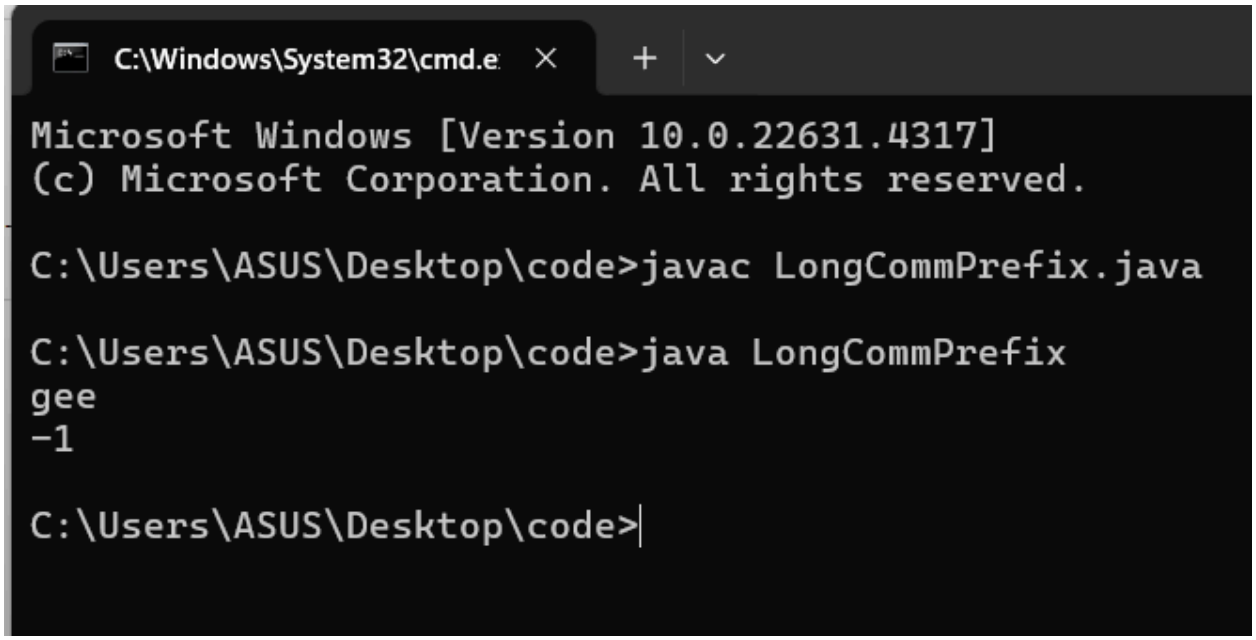
time:O(n^3)

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. Input: arr[] = [“hello”, “world”] Output: -1 Explanation: There‟s no common prefix in the given strings.

Code:

```
import java.util.Arrays;
public class LongCommPrefix {
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 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) {
        String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
        String[] arr2 = {"hello", "world"};

        System.out.println(longestCommonPrefix(arr1));
        System.out.println(longestCommonPrefix(arr2));
    }
}
```



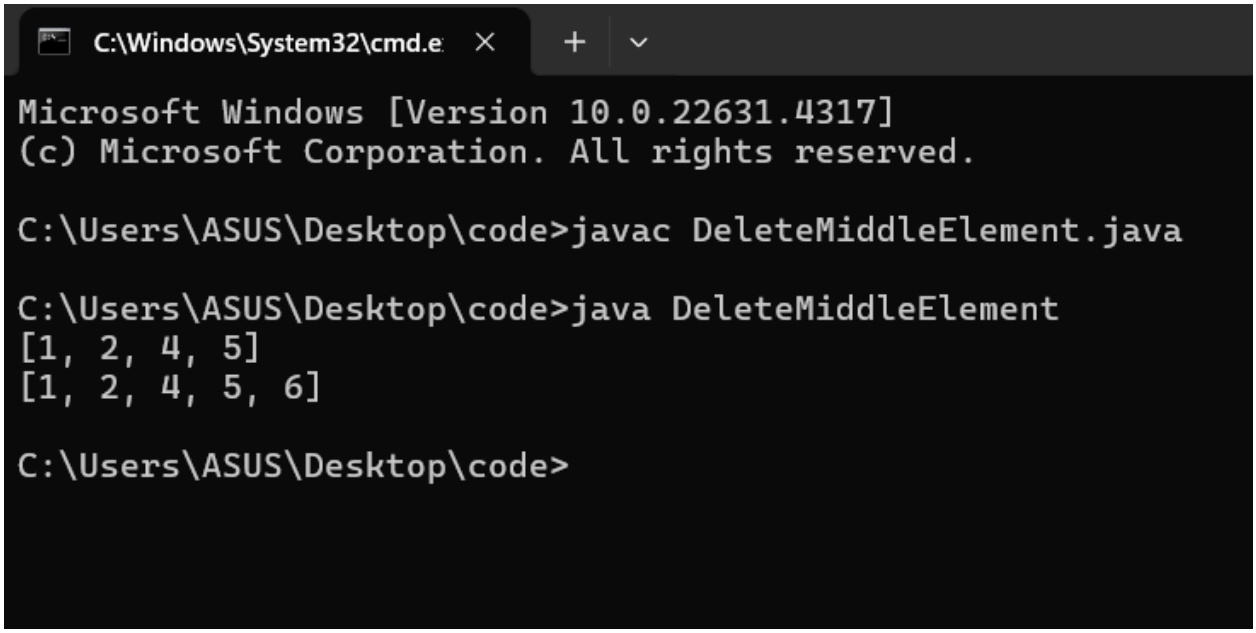
time:O(n logn)

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] Input : Stack[] = [1, 2, 3, 4, 5, 6] Output : Stack[] = [1, 2, 4, 5, 6]

Code:

```java
import java.util.Stack;

public class DeleteMiddleElement {
    public static void deleteMiddle(Stack<Integer> stack, int size, int currentIndex) {
        if (stack.isEmpty() || currentIndex == size) {
            return;
        }

        int middleIndex = size / 2;

        int temp = stack.pop();
        if (currentIndex == middleIndex) {
            return;
        }

        deleteMiddle(stack, size, currentIndex + 1);
        stack.push(temp);
    }

    public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<>();
        stack1.push(1);
        stack1.push(2);
        stack1.push(3);
        stack1.push(4);
        stack1.push(5);

        Stack<Integer> stack2 = new Stack<>();
        stack2.push(1);
        stack2.push(2);
        stack2.push(3);
        stack2.push(4);
        stack2.push(5);
        stack2.push(6);

        deleteMiddle(stack1, stack1.size(), 0);
        deleteMiddle(stack2, stack2.size(), 0);

        System.out.println(stack1);
        System.out.println(stack2);
```

```
    }
}
```



time: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 2 –> 5 –> 25 –> 25 25 –> -1 Explanation: Except 25 every element has an element greater than them present on the right side Input: arr[] = [13 , 7, 6 , 12] Output: 13 –> 7 -1 –> 12 6 12 –> 12 –> -1 Explanation: 13 and 12 don‟t have any element greater than them present on the right side

Code:

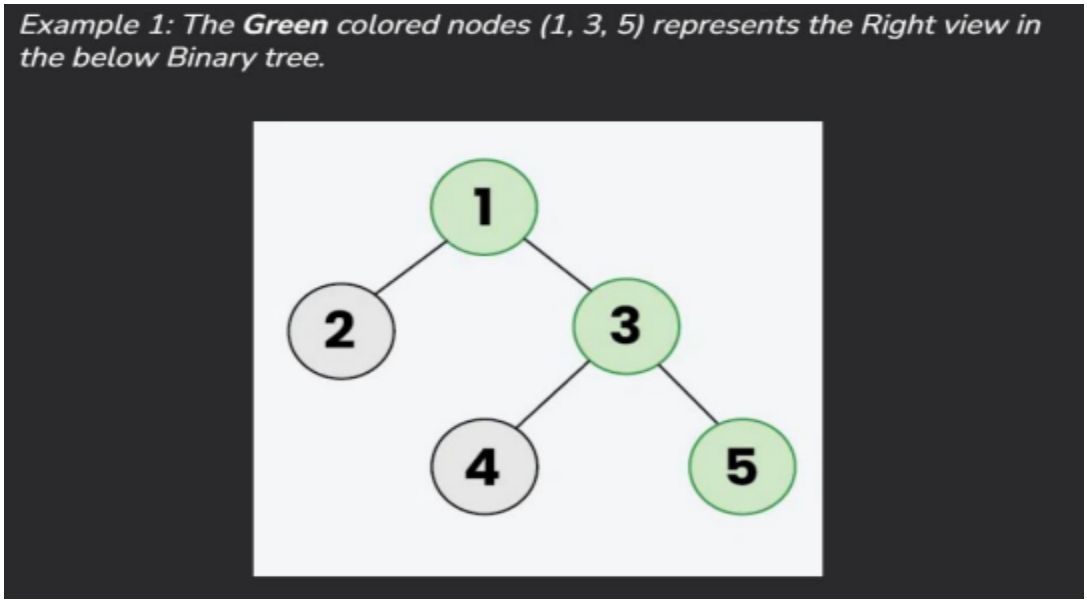
```
import java.util.Stack;

public class NextGreaterElement {
    public static void nextGreater(int[] arr) {
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < arr.length; i++) {
            while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {
                int index = stack.pop();
                System.out.println(arr[index] + " --> " + arr[i]);
```

```
        }
        stack.push(i);
    }

    while (!stack.isEmpty()) {
        int index = stack.pop();
        System.out.println(arr[index] + " --> " + -1);
    }
  }

  public static void main(String[] args) {
    int[] arr1 = {4, 5, 2, 25};
    int[] arr2 = {13, 7, 6, 12};

    nextGreater(arr1);
    nextGreater(arr2);
  }
}
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\Desktop\code>javac NextGreaterElement.java

C:\Users\ASUS\Desktop\code>java NextGreaterElement
4 --> 5
2 --> 25
5 --> 25
25 --> -1
6 --> 12
7 --> 12
12 --> -1
13 --> -1

C:\Users\ASUS\Desktop\code>
```

time: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.LinkedList;
import java.util.Queue;

class Node {
    int data;
    Node left, right;

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

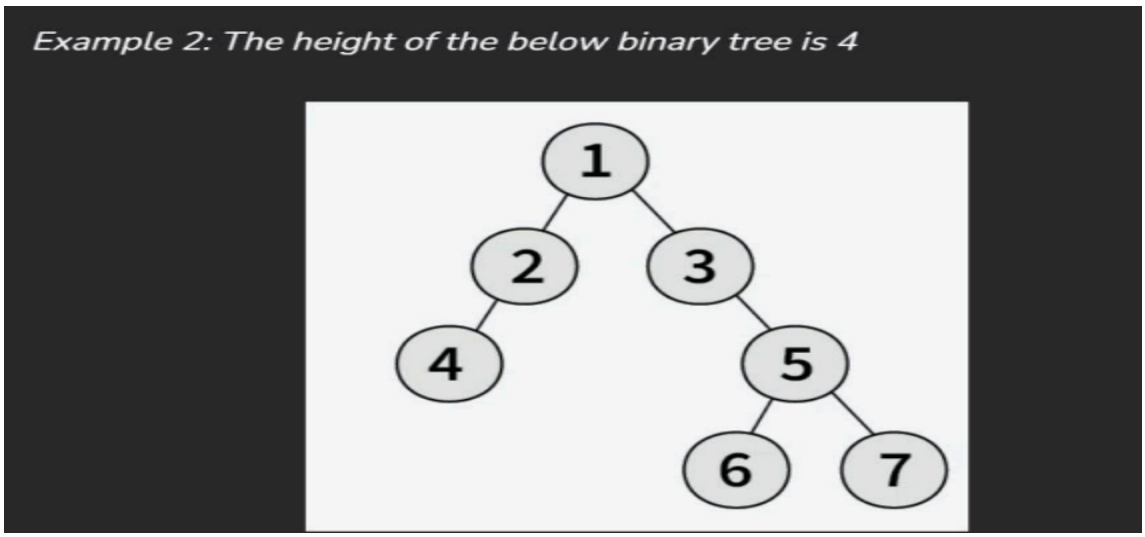
```
public class RightViewOfBinaryTree {
    public static void printRightView(Node root) {
        if (root == null) {
            return;
        }

        Queue<Node> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int size = queue.size();

            for (int i = 1; i <= size; i++) {
                Node node = queue.poll();

                if (i == size) {
                    System.out.print(node.data + " ");
                }

                if (node.left != null) {
                    queue.add(node.left);
                }

                if (node.right != null) {
```

```
                queue.add(node.right);
            }
        }
    }
}

    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);
        root.right.right = new Node(6);

        printRightView(root);
    }
}
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\Desktop\code>javac RightViewOfBinaryTree.java

C:\Users\ASUS\Desktop\code>java RightViewOfBinaryTree
1 3 6
C:\Users\ASUS\Desktop\code>
```

time: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.

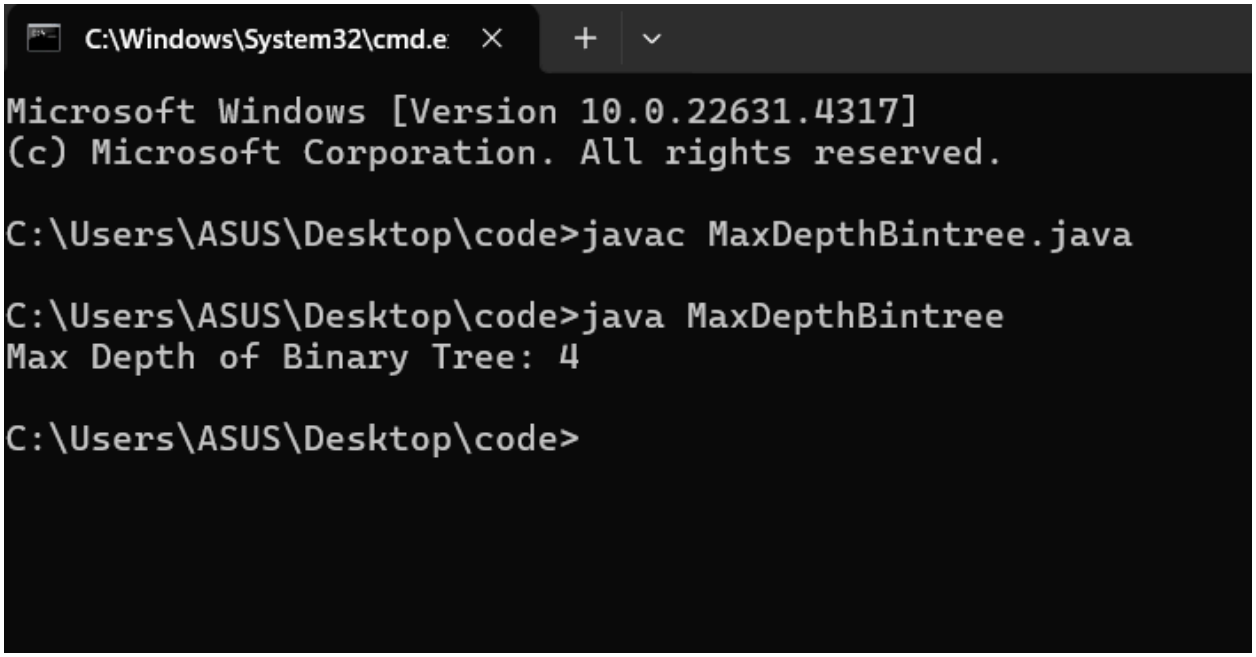


Code:

class Node {

```
    int data;
    Node left, right;

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

public class MaxDepthBintree {
    public static int maxDepth(Node root) {
        if (root == null) {
            return 0;
        }

        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);

        return Math.max(leftDepth, rightDepth) + 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.right.right = new Node(5);
        root.right.right.left = new Node(6);
        root.right.right.right = new Node(7);

        System.out.println("Max Depth of Binary Tree: " + maxDepth(root));
    }
}
```



time:O(n).