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.*;

class SubarrayAdd {

    public static int add(int[] arr1) {

        int maxc = arr1[0];

        int maxg = arr1[0];

        for (int i = 1; i < arr1.length; i++) {

            maxc = Math.max(arr1[i], maxc + arr1[i]);

            if (maxc > maxg) {

                maxg = maxc;

            }

        }

        return maxg;

    }

    public static void main(String[] args) {

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

```
        System.out.print("Enter the size of the first array: ");
        int size1 = sc.nextInt();
        int[] arr1 = new int[size1];
        System.out.println("Enter the elements of the first array:");
        for (int i = 0; i < size1; i++) {
            arr1[i] = sc.nextInt();
        }
        int result1 = add(arr1);
        System.out.println("Maximum sum of subarray for the first array: " + result1);
        sc.close();
    }
}
```

OUTPUT:

```
G:\java programs\hw>java SubarrayAdd
Enter the size of the array: 5
Enter the elements of the array:
-5 -4 3 -3 7
Maximum sum of subarray for the first array: 7
```

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.*;


class SubarrayMul {
    public static int mul(int[] arr) {
        int far = arr[0];
        int maxc = arr[0];
        int minc = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] <= 0) {
                int temp = maxc;
                maxc = minc;
                minc = temp;
            }
            maxc = Math.max(arr[i], maxc * arr[i]);
            minc = Math.min(arr[i], minc * arr[i]);
            far = Math.max(far, maxc);
        }
        return far;
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int size = sc.nextInt();
        int[] arr = new int[size];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        int result = mul(arr);
        System.out.println("Maximum product of subarray: " + result);
        sc.close();
    }
}
```

OUTPUT:

```
G:\java programs\hw>javac SubarrayMul.java
G:\java programs\hw>java SubarrayMul
Enter the size of the array: 5
Enter the elements of the array:
5 -4 3 0 2
Maximum product of subarray: 5
```

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.*;

class SearchArray {
   public static int find(int[] arr, int f) {
      for (int i = 0; i < arr.length; i++) {
         if (arr[i] == f) {
            return i;
         }
      }
      return -1;
   }

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

      System.out.print("Enter the size of the array: ");
      int size = sc.nextInt();

      int[] arr = new int[size];

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

      System.out.print("Enter the element to search for: ");
      int k = sc.nextInt();

      int result = find(arr, k);

      if (result != -1) {
         System.out.println("Element found at index: " + result);
      } else {
         System.out.println("Element not found in the array.");
```

```
        }

        sc.close();
    }
}
```

OUTPUT:

```
G:\java programs\hw>java SearchArray
Enter the size of the array: 9
Enter the elements of the array:
5 6 7 8 9 1 2 3 4
Enter the element to search for: 4
Element found at index: 8
```

TIME COMPLEXITY:
O(n)

4. Container with Most Water

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.

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

CODE

```
import java.util.*;

public class MaxWaterContainer {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of lines: ");

      int n = scanner.nextInt();

      int[] height = new int[n];

      System.out.print("Enter the heights of the lines: ");

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

         height[i] = scanner.nextInt();

      }

      System.out.println("Maximum area of water that can be contained: " + maxArea(height));
```

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

OUTPUT

```
G:\java programs\hw>java MaxWaterContainer
Enter the number of lines: 5
Enter the heights of the lines: 3 1 2 4 5
Maximum area of water that can be contained: 12
```

TIME COMPLEXITY

O(n)

5.
Find the Factorial of a large number

Input: 100
Output:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

Input: 50
Output: 30414093201713378043612608166064768844377641568960512000000000000

CODE:

```
import java.math.BigInteger;

import java.util.*;

public class FactorialCalculator {

   public static BigInteger factorial(int n) {

      BigInteger result = BigInteger.ONE;

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

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

      }

      return result;

   }

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter a number to find its factorial: ");

      try {

         int n = scanner.nextInt();

         if (n < 0) {

            System.out.println("Factorial is not defined for negative numbers.");

         } else {
```

```
            System.out.println("Factorial of " + n + " is: " + factorial(n));
        }
    } catch (Exception e) {
        System.out.println("Please enter a valid integer.");
    } finally {
        scanner.close();
    }
  }
}
```

OUTPUT

G:\java programs\hw>java FactorialCalculator

Enter a number to find its factorial: 100

Factorial of 100 is:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

TIME COMPLEXITY

O(n2logn)

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

Input: arr[] = {10, 9, 0, 5}
Output: 5
Explanation : We trap 0 + 0 + 5 + 0 = 5

CODE

```
import java.util.*;

class TrappingRainwater {
    public static int trap(int[] arr) {
        int n = arr.length;
        if (n == 0) return 0;

        int[] leftMax = new int[n];
        int[] rightMax = new int[n];
        leftMax[0] = arr[0];
        rightMax[n - 1] = arr[n - 1];

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

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

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

        return waterTrapped;
    }
```

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

        System.out.print("Enter the number of elements in the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

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

        System.out.println("Water trapped: " + trap(arr));
        sc.close();
    }
}
```

OUTPUT

```
G:\java programs\hw>java TrappingRainwater
Enter the number of elements in the array: 4
Enter the elements of the array:
1 2 3 4
Water trapped: 0
```

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.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5
Output: 7
Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

CODE

```
import java.util.*;

public class ChocolateDistribution {
    public static int findMinDifference(int[] arr, int n, int m) {
        if (m == 0 || n == 0) return 0;
        if (n < m) return -1; // Not enough packets for the students
        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) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of packets: ");
        int n = sc.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] = sc.nextInt();
        }
        System.out.print("Enter the number of students: ");
        int m = sc.nextInt();
        int result = findMinDifference(arr, n, m);
        if (result == -1) {
            System.out.println("Not enough packets for the students.");
        } else {
            System.out.println("Minimum difference is: " + result);
        }
```

```
        sc.close();
    }
}
```

OUTPUT

```
G:\java programs\hw>java ChocolateDistribution
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
```

TIME COMPLEXITY

O (n longn)

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 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 sc = new Scanner(System.in);
    System.out.print("Enter the number of intervals: ");
    int n = sc.nextInt();
    int[][] intervals = new int[n][2];
    System.out.println("Enter the intervals (start and end):");
    for (int i = 0; i < n; i++) {
      intervals[i][0] = sc.nextInt();
      intervals[i][1] = sc.nextInt();
    }
    int[][] result = merge(intervals);
    System.out.println("Merged Intervals:");
    for (int[] interval : result) {
      System.out.print("[" + interval[0] + ", " + interval[1] + "]");
    }
    sc.close();
  }
}
```

OUTPUT

G:\java programs\hw>java MergeIntervals

Enter the number of intervals: 4

Enter the intervals (start and end):

1 3

2 4

6 8

9 10

Merged Intervals:

[1, 4][6, 8][9, 10]

TIME COMPLEXITY OF

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 BooleanMatrixModifier {

   public static void modifyMatrix(int[][] mat, int M, int N) {

      boolean[] rowFlag = new boolean[M];

      boolean[] colFlag = new boolean[N];

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

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

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

               rowFlag[i] = true;

               colFlag[j] = true;

            }

         }
```

```
    }
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (rowFlag[i] || colFlag[j]) {
                mat[i][j] = 1;
            }
        }
    }
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of rows (M): ");
    int M = scanner.nextInt();
    System.out.print("Enter the number of columns (N): ");
    int N = scanner.nextInt();
    int[][] mat = new int[M][N];
    System.out.println("Enter the matrix elements (0 or 1):");
    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

```
G:\java programs\hw>java BooleanMatrixModifier
Enter the number of rows (M): 3
Enter the number of columns (N): 4
Enter the matrix elements (0 or 1):
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(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

```
import java.util.*r;

public class SpiralMatrix {

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

      if (matrix == null || matrix.length == 0) return;

      int top = 0, bottom = matrix.length - 1;

      int left = 0, right = matrix[0].length - 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 number of rows: ");
    int m = scanner.nextInt();
    System.out.print("Enter 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.print("Spiral Order: ");
        printSpiral(matrix);
        scanner.close();
    }
}
```

OUTPUT

```
G:\java programs\hw>java SpiralMartix
Enter number of rows: 8
Enter 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 19 20
21 22 23 24
25 26 27 28
29 30 31 32
Spiral order of the matrix:
1 2 3 4 8 12 16 20 24 28 32 31 30 29 25 21 17 13 9 5 6 7 11 15 19 23 27 26 22 18 14 10
```

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 = “())((())”
Output: Not Balanced

CODE

```
import java.util.*;

public class ParenthesesBalance {

   public static String isBalanced(String str) {

      int count = 0;

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

         char ch = str.charAt(i);

         if (ch == '(') {

            count++;

         } else if (ch == ')') {

            count--;

            if (count < 0) {

               return "Not Balanced";

            }

         }

      }

      return count == 0 ? "Balanced" : "Not Balanced";

   }

   public static void main(String[] args) {

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

```
        System.out.print("Enter a parentheses expression: ");
        String str = scanner.nextLine();
       String result = isBalanced(str);
       System.out.println("Expression: " + str + " => " + result);
       scanner.close();
    }
}
```

OUTPUT

```
G:\java programs\hw>java ParenthesesBalance
Enter a parentheses expression: ({}{})}[]({]][[)}
Expression: ({}{})}[]({]][[)} => 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.

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.*;
class Anagram {
   public static boolean equalstring(String str1, String str2) {
      str1 = str1.toLowerCase();
      str2 = str2.toLowerCase();
      char[] s1 = str1.toCharArray();
      char[] s2 = str2.toCharArray();
      Arrays.sort(s1);
      Arrays.sort(s2);
      return Arrays.equals(s1, s2);
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter the number of strings: ");
      int n = sc.nextInt();
      sc.nextLine();  // Consume the newline character after the integer input
      String[] arr = new String[n];
      System.out.println("Enter the strings:");
      for (int i = 0; i < n; i++) {
         arr[i] = sc.nextLine();
      }
      boolean allAnagrams = true;
      for (int i = 0; i < n - 1; i++) {
         if (!equalstring(arr[i], arr[i + 1])) {
            allAnagrams = false;
            break;
         }
      }
      if (allAnagrams) {
         System.out.println("All strings are anagrams.");
```

```
        } else {
            System.out.println("Not all strings are anagrams.");
        }
        sc.close();
    }
}
```

OUTPUT:

```
G:\java programs\hw>java Anagram
Enter the number of strings: 3
Enter the strings:
sanga
ansag
ganas
All strings are anagrams.

G:\java programs\hw>java Anagram
Enter the number of strings: 3
Enter the strings:
daya
dayi
dayal
Not all strings are anagrams.
```

TIME COMPLEXITY:

O(log(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.

Input: str = “Geeks”
Output: “ee”

Input: str = “abc”
Output: “a”

Input: str = “”
Output: “”

CODE

```
import java.util.*;

public class LongestPalindromeSubstring {

   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++) {

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

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

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

        if (len > maxLength) {

            maxLength = len;

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

         }
```

```
        }
        return str.substring(start, start + maxLength);
    }
    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) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        String result = longestPalindrome(str);
        System.out.println("Longest Palindromic Substring: " + result);
        scanner.close();
    }
}
```

OUPUT

```
G:\java programs\hw>java LongestPalindromeSubstring
Enter a string: pammans
Longest Palindromic Substring: amma
```

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.

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

CODE

```
import java.util.*;

public class LongestCommonPrefix {

public static String longestCommonPrefix(String[] arr) {

      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++;

      }

          return i > 0 ? first.substring(0, i) : "-1";

   }

   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

```
G:\java programs\hw>java LongestCommonPrefix
Enter the number of strings: 4
Enter the strings:
geeksfor
geeks
geek
geeze
Longest Common Prefix: gee
```

TIME COMPLEXITY

O(n*log 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]

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

CODE

```
import java.util.*;

public class DeleteMiddleElements {

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

      if (stack.isEmpty() || currentIndex == size) {

         return;

      }

      int element = stack.pop();

     deleteMiddle(stack, size, currentIndex + 1);

      if (currentIndex != size / 2) {

         stack.push(element);

      }

   }

   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++) {
```

```
            stack.push(scanner.nextInt());
        }
        int size = stack.size();
        deleteMiddle(stack, size, 0);
        System.out.println("Stack after deleting the middle element: " + stack);
        scanner.close();
    }
}
```

OUTPUT

```
G:\java programs\hw>java DeleteMiddleElements
Enter the number of elements in the stack: 14
Enter the elements of the stack:
3 73 38 83 9 83 87 7  9
8 5 87 3  76
Stack after deleting the middle element: [3, 73, 38, 83, 9, 83, 7, 9, 8, 5, 87, 3, 76]
```

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

Input: arr[] = [13 , 7, 6 , 12]
Output: 13 –> -1
7 –> 12
6 –> 12
12 –> -1
Explanation: 13 and 12 don‟t have any element greater than them present on the right side **CODE**

CODE

```
import java.util.*;

public class NextGreaterElement {

   public static void printNGE(int[] arr) {

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

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

      for (int num : arr) {

         while (!stack.isEmpty() && stack.peek() < num) {

            ngeMap.put(stack.pop(), num);

         }

         stack.push(num);

      }

      while (!stack.isEmpty()) {

         ngeMap.put(stack.pop(), -1);

      }
```

```
        for (int num : arr) {
            System.out.println(num + " -> " + ngeMap.get(num));
        }    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements for the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt()      }
        System.out.println("Next Greater Element for the array:");
        printNGE(arr);
        scanner.close();    }  }
```

OUTPUT

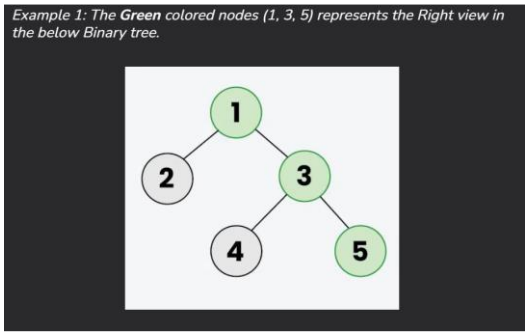
```
G:\java programs\hw>java NextGreaterElement
Enter the size of the array: 4
Enter the elements for the array: 2 4 5 2
Next Greater Element for the array:
2 -> -1
4 -> 5
5 -> -1
2 -> -1
```

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.*;

class Node {
   int data;
   Node left, right;
   public Node(int data) {
      this.data = data;
      left = right = null;
   }
}
public class BinaryTree {
   Node root;
   public void printRightView() {
      if (root == null) return;
      Queue<Node> queue = new LinkedList<>();
      queue.add(root);
      while (!queue.isEmpty()) {
```

```
        int n = queue.size();
        for (int i = 1; i <= n; i++) {
            Node node = queue.poll();
            if (i == n) {
                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) {
    Scanner scanner = new Scanner(System.in);
    BinaryTree tree = new BinaryTree();
    System.out.println("Enter the number of nodes in the binary tree:");
    int n = scanner.nextInt();
    System.out.println("Enter the tree nodes in level order (use -1 for null nodes):");
    tree.root = inputTree(scanner, n);
    System.out.print("Right view of the binary tree: ");
    tree.printRightView();
}
public static Node inputTree(Scanner scanner, int n) {
    if (n == 0) return null;
    System.out.println("Enter root node:");
    int rootData = scanner.nextInt();
```

```
        Node root = new Node(rootData);
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            Node currentNode = queue.poll();
            System.out.println("Enter left child of " + currentNode.data + " (-1 if no left child):");
            int leftData = scanner.nextInt();
            if (leftData != -1) {
                currentNode.left = new Node(leftData);
                queue.add(currentNode.left);
            }
            System.out.println("Enter right child of " + currentNode.data + " (-1 if no right child):");
            int rightData = scanner.nextInt();
            if (rightData != -1) {
                currentNode.right = new Node(rightData);
                queue.add(currentNode.right);
            }
        }
        return root;
    }
}
```

OUTPUT

```
G:\java programs\hw>java BinaryTree
Enter the number of nodes in the binary tree: 5
Enter the tree nodes in level order (use -1 for null nodes):
```

Enter root node: 1

Enter left child of 1 (-1 if no left child): 2

Enter right child of 1 (-1 if no right child): 3

Enter left child of 2 (-1 if no left child): 4

Enter right child of 2 (-1 if no right child): 5

Enter left child of 3 (-1 if no left child): -1

Enter right child of 3 (-1 if no right child): 5

Enter left child of 4 (-1 if no left child): -1

Enter right child of 4 (-1 if no right child): 9

Enter left child of 5 (-1 if no left child): -1

Enter right child of 5 (-1 if no right child): -1

Enter left child of 5 (-1 if no left child): -1

Enter right child of 5 (-1 if no right child): -1

Enter left child of 9 (-1 if no left child): -1

Enter right child of 9 (-1 if no right child): -1

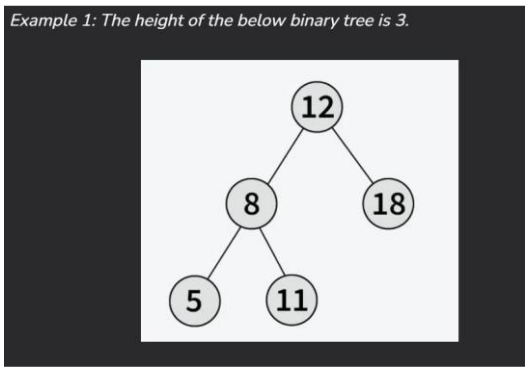
Right view of the binary tree: 1 3 5 9

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

```
import java.util.*;

class Node {
   int data;
   Node left, right;
   public Node(int data) {
      this.data = data;
      left = right = null;
   }
}
public class BinaryTreeExample {
   Node root;
   public int maxDepth(Node node) {
      if (node == null) {
         return 0;
      }
      int leftDepth = maxDepth(node.left);
```

```
    int rightDepth = maxDepth(node.right);
    return Math.max(leftDepth, rightDepth) + 1;
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    BinaryTreeExample tree = new BinaryTreeExample();
    System.out.println("Enter the number of nodes in the binary tree:");
    int n = scanner.nextInt();
    System.out.println("Enter the tree nodes in level order (use -1 for null nodes):");
    tree.root = inputTree(scanner, n);
    int height = tree.maxDepth(tree.root);
    System.out.println("Maximum depth (height) of the binary tree: " + height);
}
public static Node inputTree(Scanner scanner, int n) {
    if (n == 0) return null;
    System.out.println("Enter root node:");
    int rootData = scanner.nextInt();
    Node root = new Node(rootData);
    Queue<Node> queue = new LinkedList<>();
    queue.add(root);
    while (!queue.isEmpty()) {
        Node currentNode = queue.poll();
        System.out.println("Enter left child of " + currentNode.data + " (-1 if no left child):");
        int leftData = scanner.nextInt();
        if (leftData != -1) {
```

```
            currentNode.left = new Node(leftData);
            queue.add(currentNode.left);
        }
        System.out.println("Enter right child of " + currentNode.data + " (-1 if no right child):");
        int rightData = scanner.nextInt();
        if (rightData != -1) {
            currentNode.right = new Node(rightData);
            queue.add(currentNode.right);
        }
      }
      return root;
   }
}
```

OUTPUT

```
G:\java programs\hw>java BinaryTreeExample
Enter the number of nodes in the binary tree: 5
Enter the tree nodes in level order (use -1 for null nodes):
Enter root node: 6
Enter left child of 6 (-1 if no left child): 3
Enter right child of 6 (-1 if no right child): 8
Enter left child of 3 (-1 if no left child): 3
Enter right child of 3 (-1 if no right child): 7
Enter left child of 8 (-1 if no left child): 23
Enter right child of 8 (-1 if no right child): 87
Enter left child of 3 (-1 if no left child): 4
```

Enter right child of 3 (-1 if no right child): -1

Enter left child of 7 (-1 if no left child): -1

Enter right child of 7 (-1 if no right child): -1

Enter left child of 23 (-1 if no left child): 5

Enter right child of 23 (-1 if no right child): -1

Enter left child of 87 (-1 if no left child): -1

Enter right child of 87 (-1 if no right child): -1

Enter left child of 4 (-1 if no left child): -1

Enter right child of 4 (-1 if no right child): -1

Enter left child of 5 (-1 if no left child): -1

Enter right child of 5 (-1 if no right child): -1

Maximum depth (height) of the binary tree: 4

TIME COMPLEXITY

O(n)