```
// 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.

import java.util.Scanner;

public class Problem1{
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the Size of 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();
      }

      int max = Integer.MIN_VALUE, sum = 0;
      for(int i=0; i<n; i++){
         sum += arr[i];
         max = Math.max(sum, max);
         if(sum < 0) sum=0;
      }
      System.out.println("Maximum Subarray sum is"+ " " + max);

      sc.close();
   }
}

// Output:
// arr = {1,2,3,4,5}
```

// Maximum Subarray sum is 15

// Time complexity : O(n);

// Space complexity : O(1);

```
PS C:\Users\subas\OneDrive\Desktop\9112024> javac Problem1.java
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem1.java
Enter the Size of the Array
7
Enter the Elements of the Array
2
3
-8
7
-1
2
3
Maximum Subarray sum is 11
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem1.java
Enter the Size of the Array
2
Enter the Elements of the Array
-2
-4
Maximum Subarray sum is -2
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

// Maximum Product Subarray

// Given an integer array, the task is to find the maximum product of any subarray.

```
import java.util.Scanner;

public class Problem2{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Size of 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();
        }

        int max = Integer.MIN_VALUE, prod = 1;
        for(int i=0; i<n; i++){
            for(int j=i+1; j<n; j++){
                prod *= arr[j];
                max = Math.max(prod, max);
            }
        }

        System.out.println(max);
        sc.close();
    }
}

// Input: arr[] = {-2, 6, -3, -10, 0, 2}
// Output: 180

// Time complexity : O(n**2);
// Space complexity : O(1);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem2.java
Enter the Size of the Array
6
Enter the Elements of the Array
-2
6
-3
-10
0
2
180
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem2.java
Enter the Size of the Array
5
Enter the Elements of the Array
-1
-3
-10
0
60
30
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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.

import java.util.Scanner;

public class Problem3 {
    static int findelement(int[] nums, int target){
        int n = nums.length;
        if(nums[n-1] >= target){
            for(int i = n-1; i>=0; i--){
                if(nums[i] == target)   return i;
            }
        }
        else{
            for(int i=0; i<n-1; i++){
```

```
            if(nums[i] == target)   return i;
        }
    }
    return -1;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Size of the Array");
    int n = sc.nextInt();
    System.out.println("Enter the Target Element");
    int target = sc.nextInt();
    int[] nums = new int[n];
    System.out.println("Enter the Elements of the Array");
    for(int i=0; i<n; i++){
        nums[i] = sc.nextInt();
    }

    System.out.println(findelement(nums, target));
    sc.close();
  }
}

// Time complexity : O(n);
// Space complexity : O(1);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem3.java
Enter the Size of the Array
7
Enter the Target Element
0
Enter the Elements of the Array
4
5
6
7
0
1
2
4
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// container with Most Water

import java.util.Scanner;

public class Problem4 {
    static int maxwater(int[] arr){
        int left= 0, right = arr.length-1, maxarea =0;
        while(left < right){
            int area = Math.min(arr[left], arr[right]) * (right - left);
            maxarea = Math.max(area, maxarea);
            if(arr[left] < arr[right])  left++;
            else    right--;
        }
        return maxarea;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Size of the Array");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the Elements of the Array");
        for(int i=0; i<n; i++){
```

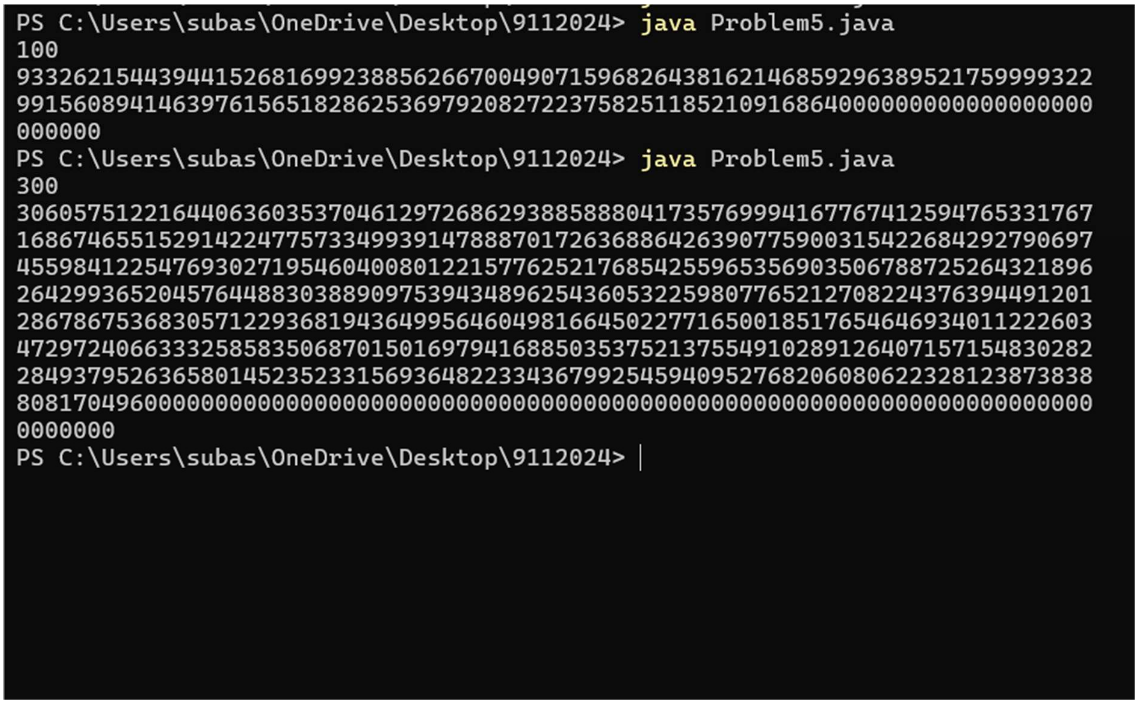
```
            nums[i] = sc.nextInt();
        }
        System.out.println(maxwater(nums));
        sc.close();
    }
}

// Time complexity : O(n)
// Space complexity : O(1)
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem4.java
Enter the Size of the Array
4
Enter the Elements of the Array
1
5
4
3
6
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem4.java
Enter the Size of the Array
5
Enter the Elements of the Array
3
1
2
4
5
12
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// Find the Factorial of a large number

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

public class Problem5 {
    static BigInteger factorial(int n){
```

```
        if(n == 1)  return BigInteger.ONE;
        return BigInteger.valueOf(n).multiply(factorial(n-1));
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println(factorial(sc.nextInt()));
        sc.close();
    }

}
```

```
// Time complexity : O(n**2);
// Space Complexity : O(1);
```



```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem5.java
100
9332621544394415268169923885626670049071596826438162146859296389521759999322
9915608941463976156518286253697920827223758251185210916864000000000000000000
000000
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem5.java
300
3060575122164406360353704612972686293885888041735769994167767412594765331767
1686746551529142247757334993914788870172636886426390775900315422684292790697
4559841225476930271954604008012215776252176854255965356903506788725264321896
2642993652045764488303889097539434896254360532259807765212708224376394491201
2867867536830571229368194364995646049816645022771650018517654646934011222603
4729724066333258583506870150169794168850353752137554910289126407157154830282
2849379526365801452352331569364822334367992545940952768206080622328123873838
8081704960000000000000000000000000000000000000000000000000000000000000000000
0000000
PS C:\Users\subas\OneDrive\Desktop\9112024> |
```

```
// 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.
```

```
import java.util.Scanner;

public class Problem6 {
    public static int trap(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int leftMax = height[left];
        int rightMax = height[right];
        int water = 0;

        while (left < right) {
            if (leftMax < rightMax) {
                left++;
                leftMax = Math.max(leftMax, height[left]);
                water += leftMax - height[left];
            } else {
                right--;
                rightMax = Math.max(rightMax, height[right]);
                water += rightMax - height[right];
            }
        }

        return water;
    }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Size of 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(trap(arr));
        sc.close();
    }
}
```

// Time Complexity : O(n);

// Space Complexity : O(1);

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem6.java
Enter the Size of the Array
7
Enter the Elements of the Array
3
0
1
0
4
0
2
10
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem6.java
Enter the Size of the Array
5
Enter the Elements of the Array
3
0
2
0
4
7
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

// 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.

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

public class Problem7 {
    static int findMinDiff(int[] arr, int m) {
        int n = arr.length;
        Arrays.sort(arr);
        int minDiff = Integer.MAX_VALUE;

        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            if (diff < minDiff)
                minDiff = diff;
        }
        return minDiff;
    }
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Size of the Array");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the Elements of the Array");
        for(int i=0; i<n; i++){
            nums[i] = sc.nextInt();
        }
        System.out.println("Enter the Childeren");
        int m = sc.nextInt();
        System.out.println(findMinDiff(nums, m));
        sc.close();
```

```
    }
}


// Time Complexity : O(n);
// Space Complexity : O(1);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem7.java
Enter the Size of the Array
7
Enter the Elements of the Array
7
3
2
4
9
12
56
Enter the Childeren
3
2
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

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


public class Problem8 {
    public static int[][] merge(int[][] intervals) {
        if (intervals.length <= 1) return intervals;
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        ArrayList<int[]> merge = new ArrayList<>();
        int[] cur = intervals[0];
```

```
    merge.add(cur);

    for (int i = 1; i < intervals.length; i++) {
        int currEnd = cur[1];
        int nextStart = intervals[i][0];
        int nextEnd = intervals[i][1];

        if (currEnd >= nextStart) {
            cur[1] = Math.max(currEnd, nextEnd);
        } else {
            cur = intervals[i];
            merge.add(cur);
        }
    }
    return merge.toArray(new int[merge.size()][]);
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the size of the array");
    int n = sc.nextInt();
    int[][] matrix = new int[n][2];
    System.out.println("Enter the elements of the array");

    for (int i = 0; i < n; i++) {
        System.out.println("Enter start and end for interval " + (i + 1));
        matrix[i][0] = sc.nextInt();
        matrix[i][1] = sc.nextInt();
    }

    int[][] result = merge(matrix);
```

```
        System.out.println("Merged intervals:");
        for (int[] interval : result) {
            System.out.println(Arrays.toString(interval));
        }

        sc.close();
    }
}
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem8.java
Enter the size of the array
4
Enter the elements of the array
Enter start and end for interval 1
1
3
Enter start and end for interval 2
2
4
Enter start and end for interval 3
6
8
Enter start and end for interval 4
9
10
Merged intervals:
[1, 4]
[6, 8]
[9, 10]
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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.

import java.util.Scanner;

public class Problem9 {

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

```
Scanner sc = new Scanner(System.in);
System.out.println("Enter the size of the matrix (rows and columns):");
int rows = sc.nextInt();
int cols = sc.nextInt();
int[][] matrix = new int[rows][cols];

System.out.println("Enter the elements of the matrix:");
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        matrix[i][j] = sc.nextInt();
    }
}

for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        if (matrix[i][j] == 1) {
            int ind = i - 1;
            while (ind >= 0) {
                if (matrix[ind][j] != 1) {
                    matrix[ind][j] = -1;
                }
                ind--;
            }
            ind = i + 1;
            while (ind < rows) {
                if (matrix[ind][j] != 1) {
                    matrix[ind][j] = -1;
                }
                ind++;
            }
            ind = j - 1;
```

```
            while (ind >= 0) {
                if (matrix[i][ind] != 1) {
                    matrix[i][ind] = -1;
                }
                ind--;
            }
            ind = j + 1;
            while (ind < cols) {
                if (matrix[i][ind] != 1) {
                    matrix[i][ind] = -1;
                }
                ind++;
            }
        }
    }
}

for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        if (matrix[i][j] < 0) {
            matrix[i][j] = 1;
        }
    }
}

System.out.println("Modified Matrix:");
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
```

```
        }


        sc.close();
    }
}


// Time complexity : O(N^2);
// Space Complexity : O(1);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem9.java
Enter the size of the matrix (rows and columns):
2
2
Enter the elements of the matrix:
1
0
0
0
Modified Matrix:
1 1
1 0
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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.


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


public class Problem10 {
    public static List<Integer> spiralOrder(int[][] matrix) {
        ArrayList<Integer> ans=new ArrayList<>();
        int row=0 , rowls=matrix.length-1;
        int col=0 , colls=matrix[0].length-1;
```

```
    while(row<=rowls && col<=colls){
        for(int i=col; i<=colls; i++){
            ans.add(matrix[row][i]);
        }
        row++;
        for(int i=row; i<=rowls; i++){
            ans.add(matrix[i][colls]);
        }
        colls--;
        if(rowls>=row){
            for(int i=colls; i>=col; i--){
                ans.add(matrix[rowls][i]);
            }

        }
        rowls--;
        if(col<=colls){
            for(int i=rowls; i>=row; i--){
                ans.add(matrix[i][col]);

            }

        }
        col++;
    }
    return ans;
}
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the size of the matrix (rows and columns):");
    int rows = sc.nextInt();
```

```
        int cols = sc.nextInt();
        int[][] matrix = new int[rows][cols];

        System.out.println("Enter the elements of the matrix:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        System.out.println(spiralOrder(matrix));
      sc.close();
    }
}
```

// Time Complexity : O(n);

// Space Complexity : O(n);

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem10.java
Enter the size of the matrix (rows and columns):
4
4
Enter the elements of the matrix:
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
[1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

// 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.

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

public class Problem13 {
    public static boolean ispar(String s) {
        Stack<Character> stk = new Stack<>();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[') {
                stk.push(s.charAt(i));
            }
            else {
                if (!stk.empty() &&
                    ((stk.peek() == '(' && s.charAt(i) == ')') ||
                     (stk.peek() == '{' && s.charAt(i) == '}') ||
                     (stk.peek() == '[' && s.charAt(i) == ']'))) {
                    stk.pop();
                }
                else {
                    return false;
                }
            }
        }
        return stk.empty();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();
        System.out.println(ispar(str)? "Balanced" : "Unbalanced");
```

```
        sc.close();
    }
}

// Time Complexity : O(n);
// Space Complexity : O(n);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem13.java
((()))()()
Balanced
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem13.java
(((()))))
Unbalanced
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem13.java
()()()))))))
Unbalanced
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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.

import java.util.HashMap;
import java.util.Scanner;

public class Problem14 {
    public static boolean isAnagram(String s, String t) {

        if(s.length()!= t.length()) return false;
        HashMap<Character, Integer> map1= new HashMap<>();
        HashMap<Character, Integer> map2= new HashMap<>();
         for(int i=0; i<s.length();i++){
```

```
            if(map1.containsKey(s.charAt(i))) map1.put(s.charAt(i),map1.get(s.charAt(i))+1);
            else map1.put(s.charAt(i),1);
        }
         for(int i=0; i<t.length();i++){
            if(map2.containsKey(t.charAt(i))) map2.put(t.charAt(i),map2.get(t.charAt(i))+1);
            else map2.put(t.charAt(i),1);
        }

        for(char k: map1.keySet()){
           if(!map1.get(k).equals(map2.get(k))) return false;
        }

        return true;

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

        Scanner sc = new Scanner(System.in);
        String str1 = sc.nextLine();
        String str2 = sc.nextLine();
        System.out.println(isAnagram(str1, str2));
        sc.close();

    }
}

// Time Complexity : O(n);
// Space Complexity : O(n + n);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> javac Problem14.java
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem14.java
geeks
kseeg
true
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem14.java
allergy
allergic
false
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem14.java
subash
rajitha
false
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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.


import java.util.Scanner;


public class Problem15 {

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

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

            left--;

            right++;

        }

        return right - left - 1;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println(longestPalindrome(sc.nextLine()));

        sc.close();

    }

    public static String longestPalindrome(String s) {

        if (s == null || s.length() == 0) {
```

```
        return "";
    }

    int start = 0;
    int end = 0;

    for (int i = 0; i < s.length(); i++) {
        int odd = expandAroundCenter(s, i, i);
        int even = expandAroundCenter(s, i, i + 1);
        int max_len = Math.max(odd, even);

        if (max_len > end - start) {
            start = i - (max_len - 1) / 2;
            end = i + max_len / 2;
        }
    }

    return s.substring(start, end + 1);
  }
}

// Time Complexity : O(n + n);
// Space Complexity : O(1);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> javac Problem15.java
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem15.java
forgeeksskeegfor
geeksskeeg
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem15.java
geeks
ee
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem15.java
subash
h
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem15.java
rajithaaa
aaa
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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”.

import java.util.Scanner;

public class Problem16 {
    public static String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) return "";
        String prefix = strs[0];
        for (int i = 1; i < strs.length; i++) {
            while (!strs[i].startsWith(prefix)) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) return "";
            }
        }
        return prefix;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = sc.nextInt();
        sc.nextLine();  // Consume the leftover newline
        String[] nums = new String[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = sc.nextLine();
        }
        System.out.println(longestCommonPrefix(nums));
        sc.close();
```

```
    }
}

// Time Complexity : O(n**2);
// Space Complexity : O(1);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> javac Problem16.java
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem16.java
Enter the size of the array:
4
Enter the elements of the array:
geeksforgeeks
geeks
geek
geezer
gee
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem16.java
Enter the size of the array:
4
Enter the elements of the array:
subash
rajitha
arutselvi
prasanna
```

```
// 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.

import java.util.*;

public class Problem17 {
    public static void del(Stack<Integer> stack) {
        int size = stack.size();
        int mid = size / 2;
        delete(stack, mid);
    }

    private static void delete(Stack<Integer> stack, int middleIndex) {
```

```
        if (middleIndex == 0) {
            stack.pop();
            return;
        }

        int top = stack.pop();
        delete(stack, middleIndex - 1);
        stack.push(top);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Stack<Integer> stk = new Stack<>();
        System.out.println("Enter the size of the stack:");
        int n = sc.nextInt();

        System.out.println("Enter the elements of the stack:");
        for (int i = 0; i < n; i++) {
            stk.push(sc.nextInt());
        }

        System.out.println("Stack before deleting middle element: " + stk);

        del(stk);

        System.out.println("Stack after deleting middle element: " + stk);
        sc.close();
    }
}

// Time Complexity : O(n);
```

// Space Complexity : O(1);

```
Enter the size of the stack:
5
Enter the elements of the stack:
1
2
3
4
5
Stack before deleting middle element: [1, 2, 3, 4, 5]
Stack after deleting middle element: [1, 2, 4, 5]
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem17.java
Enter the size of the stack:
6
Enter the elements of the stack:
1
2
3
4
5
6
Stack before deleting middle element: [1, 2, 3, 4, 5, 6]
Stack after deleting middle element: [1, 2, 4, 5, 6]
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
import java.util.*;


public class Problem18 {

    public static int[] nextGreaterElements(int[] nums) {

        int n = nums.length;

        int[] ans = new int[n];

        Arrays.fill(ans, -1);  // Default value for each element is -1


        // Using a stack to find the next greater element in circular manner

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


        for (int i = 0; i < 2 * n; i++) {

            while (!stack.isEmpty() && nums[stack.peek()] < nums[i % n]) {

                int idx = stack.pop();

                ans[idx] = nums[i % n];

            }
```

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

    return ans;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Size of the Array");
    int n = sc.nextInt();
    int[] nums = new int[n];
    System.out.println("Enter the Elements of the Array");
    for (int i = 0; i < n; i++) {
        nums[i] = sc.nextInt();
    }

    int[] result = nextGreaterElements(nums);
    System.out.println("Next Greater Elements:");
    for (int i : result) {
        System.out.print(i + " ");
    }
    sc.close();
}
```

}

```
PS C:\Users\subas\OneDrive\Desktop\9112024> javac Problem18.java
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem18.java
Enter the Size of the Array
4
Enter the Elements of the Array
4
5
2
25
Next Greater Elements:
5 25 25 -1
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem18.java
Enter the Size of the Array
4
Enter the Elements of the Array
13
7
6
12
Next Greater Elements:
-1 12 12 13
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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;

import java.util.LinkedList;

import java.util.List;

import java.util.Queue;

import java.util.Scanner;


class TreeNode {

    int val;

    TreeNode left, right;


    TreeNode(int val) {

        this.val = val;

        left = right = null;
```

```
    }
}

public class Problem19 {
    public static TreeNode createTree() {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.println("Enter the root value: ");
            int rootval = sc.nextInt();

            if (rootval == -1) return null;
            TreeNode root = new TreeNode(rootval);
            Queue<TreeNode> queue = new LinkedList<>();
            queue.add(root);

            while (!queue.isEmpty()) {
                TreeNode curr = queue.poll();
                System.out.println("Enter the left child of " + curr.val + ": ");
                int leftval = sc.nextInt();
                if (leftval != -1) {
                    curr.left = new TreeNode(leftval);
                    queue.add(curr.left);
                }
                System.out.println("Enter the right child of " + curr.val + ": ");
                int rightval = sc.nextInt();
                if (rightval != -1) {
                    curr.right = new TreeNode(rightval);
                    queue.add(curr.right);
                }
            }
            return root;
```

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

    public static void main(String[] args) {
        TreeNode root = createTree();
        System.out.println(rightSideView(root));
    }

    public static List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        rightView(root, result, 0);
        return result;
    }

    public static void rightView(TreeNode curr, List<Integer> result, int currDepth) {
        if (curr == null) {
            return;
        }
        if (currDepth == result.size()) {
            result.add(curr.val);
        }

        rightView(curr.right, result, currDepth + 1);
        rightView(curr.left, result, currDepth + 1);
    }
}

// Time compexity : O(N);
// space compexity : O(H);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> javac Problem19.java
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem19.java
Enter the root value:
1
Enter the left child of 1:
2
Enter the right child of 1:
3
Enter the left child of 2:
-1
Enter the right child of 2:
-1
Enter the left child of 3:
4
Enter the right child of 3:
5
Enter the left child of 4:
-1
Enter the right child of 4:
-1
Enter the left child of 5:
-1
Enter the right child of 5:
-1
[1, 3, 5]
PS C:\Users\subas\OneDrive\Desktop\9112024>
```

```
// 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.


import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;


class TreeNode {

    int val;

    TreeNode left, right;


    TreeNode(int val) {

        this.val = val;

        left = right = null;

    }

}
```

```
public class Problem20 {
    public static int maxDepth(TreeNode root) {
        if (root == null) return 0;
        int lh = maxDepth(root.left);
        int rh = maxDepth(root.right);
        return Math.max(lh, rh) + 1;
    }

    public static void main(String[] args) {
        TreeNode root = createTree();
        System.out.println(maxDepth(root));
    }

    public static TreeNode createTree() {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.println("Enter the root value: ");
            int rootval = sc.nextInt();

            if (rootval == -1) return null;
            TreeNode root = new TreeNode(rootval);
            Queue<TreeNode> queue = new LinkedList<>();
            queue.add(root);

            while (!queue.isEmpty()) {
                TreeNode curr = queue.poll();
                System.out.println("Enter the left child of " + curr.val + ": ");
                int leftval = sc.nextInt();
                if (leftval != -1) {
                    curr.left = new TreeNode(leftval);
```

```
                queue.add(curr.left);
            }
            System.out.println("Enter the right child of " + curr.val + ": ");
            int rightval = sc.nextInt();
            if (rightval != -1) {
                curr.right = new TreeNode(rightval);
                queue.add(curr.right);
            }
        }
        return root;
    } finally {
        sc.close();
    }
  }
}

// Time compexity : O(N);
// space compexity : O(H);
```

```
PS C:\Users\subas\OneDrive\Desktop\9112024> javac Problem20.java
PS C:\Users\subas\OneDrive\Desktop\9112024> java Problem20.java
Enter the root value:
12
Enter the left child of 12:
8
Enter the right child of 12:
18
Enter the left child of 8:
5
Enter the right child of 8:
11
Enter the left child of 18:
-1
Enter the right child of 18:
-1
Enter the left child of 5:
-1
Enter the right child of 5:
-1
Enter the left child of 11:
-1
Enter the right child of 11:
-1
3
PS C:\Users\subas\OneDrive\Desktop\9112024>
```