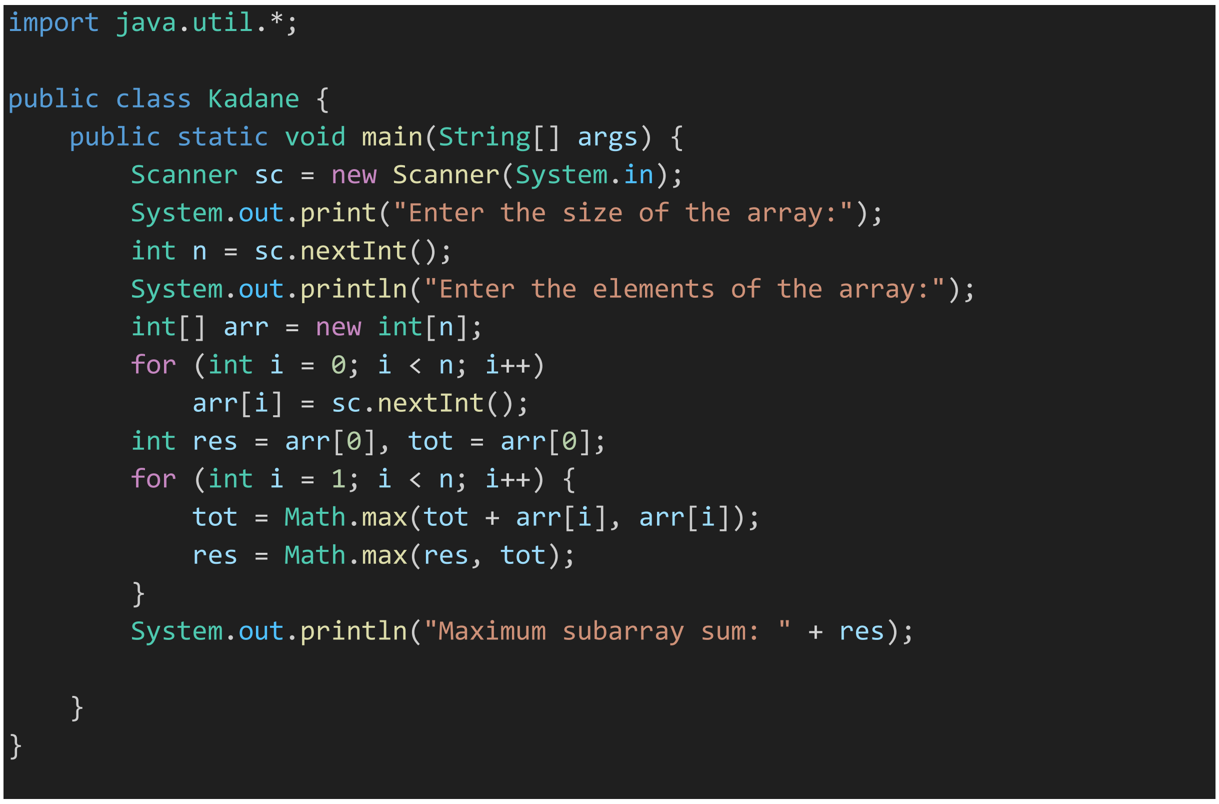
Coding practice Problems:

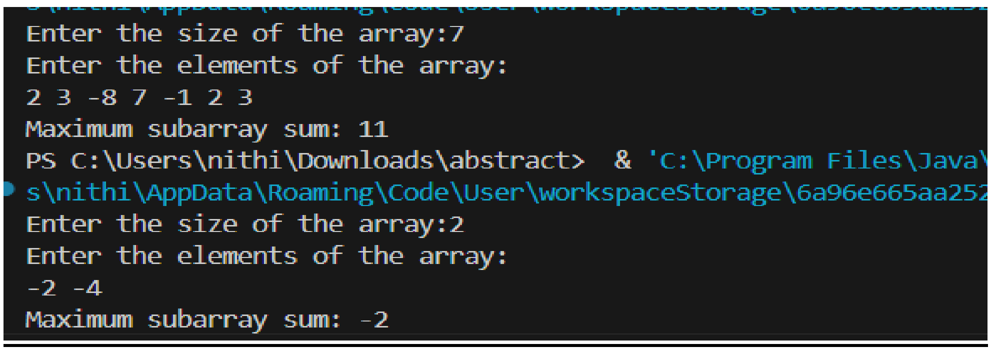
1. Maximum Subarray Sum – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.
   Input: arr[] = {2, 3, -8, 7, -1, 2, 3} Output: 11
   Explanation: The subarray {7, -1, 2, 3} has the largest sum 11. Input: arr[] = {-2, -4}
   Output: –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.

<u>CODE:</u>



```java
import java.util.*;

public class Kadane {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array:");
        int n = sc.nextInt();
        System.out.println("Enter the elements of the array:");
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        int res = arr[0], tot = arr[0];
        for (int i = 1; i < n; i++) {
            tot = Math.max(tot + arr[i], arr[i]);
            res = Math.max(res, tot);
        }
        System.out.println("Maximum subarray sum: " + res);

    }
}
```

<u>OUTPUT :</u>



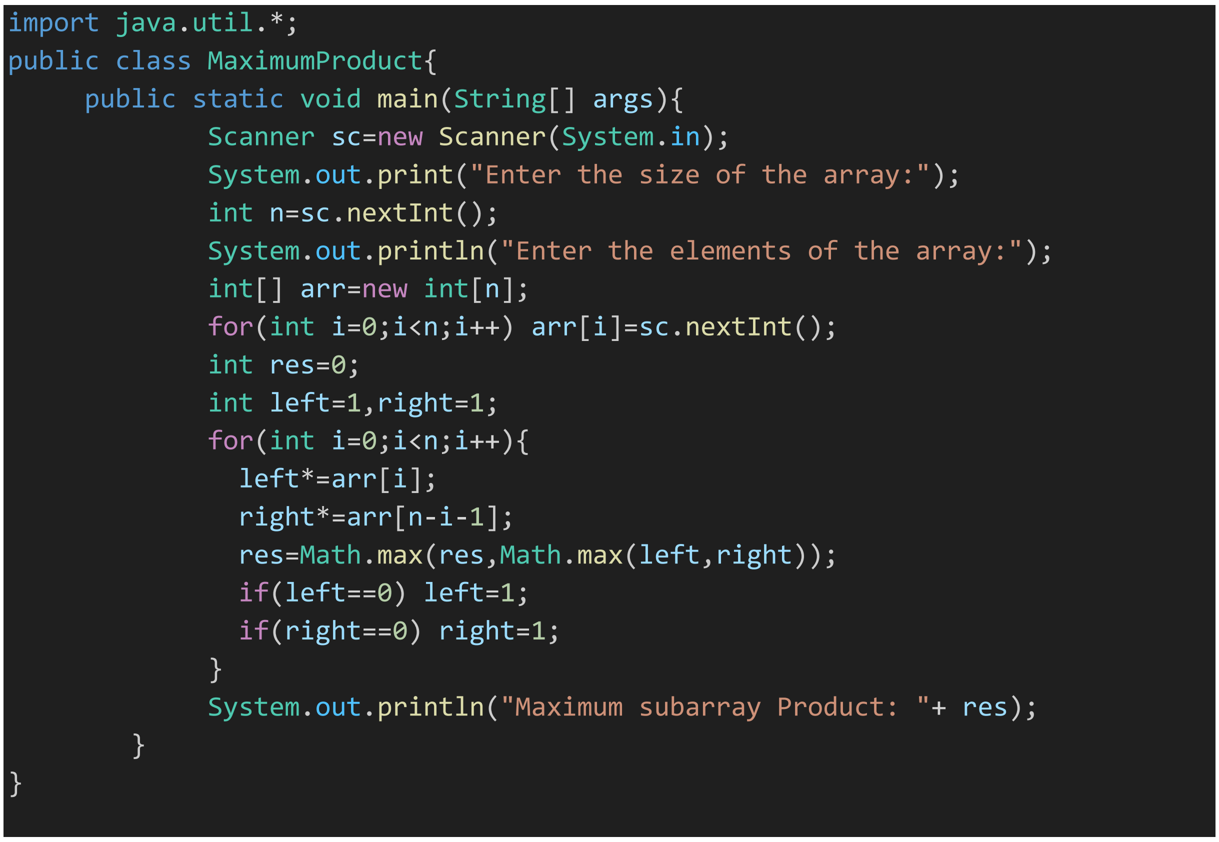
```
Enter the size of the array:7
Enter the elements of the array:
2 3 -8 7 -1 2 3
Maximum subarray sum: 11
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa252
Enter the size of the array:2
Enter the elements of the array:
-2 -4
Maximum subarray sum: -2
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

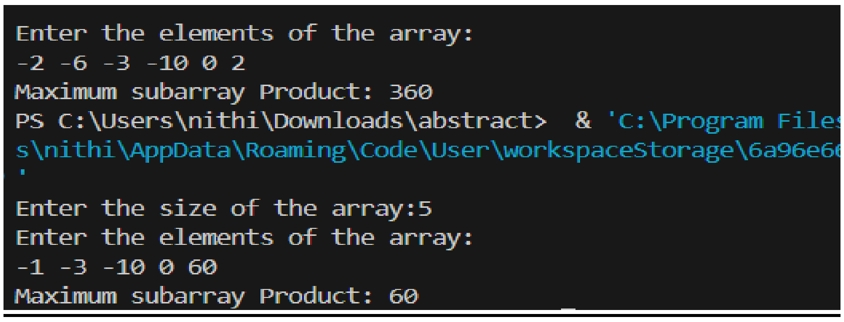
2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.
   Input: arr[] = {-2, 6, -3, -10, 0, 2} Output: 180
   Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 * (-3) * (-10) = 180
   Input: arr[] = {-1, -3, -10, 0,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

<u>CODE :</u>



```java
import java.util.*;
public class MaximumProduct{
     public static void main(String[] args){
             Scanner sc=new Scanner(System.in);
             System.out.print("Enter the size of the array:");
             int n=sc.nextInt();
             System.out.println("Enter the elements of the array:");
             int[] arr=new int[n];
             for(int i=0;i<n;i++) arr[i]=sc.nextInt();
             int res=0;
             int left=1,right=1;
             for(int i=0;i<n;i++){
               left*=arr[i];
               right*=arr[n-i-1];
               res=Math.max(res,Math.max(left,right));
               if(left==0) left=1;
               if(right==0) right=1;
             }
             System.out.println("Maximum subarray Product: "+ res);
        }
}
```

<u>OUTPUT :</u>



```
Enter the elements of the array:
-2 -6 -3 -10 0 2
Maximum subarray Product: 360
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e6
'
Enter the size of the array:5
Enter the elements of the array:
-1 -3 -10 0 60
Maximum subarray Product: 60
```

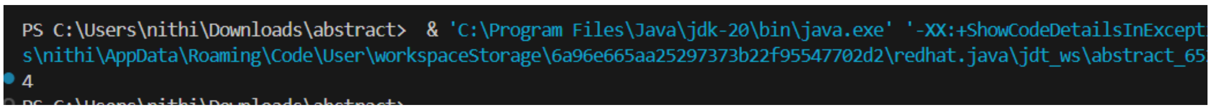
TIME COMPLEXITY : O(n)   SPACE COMPLEXITY : O(1)

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 :

```java
import java.util.*;

public class Search {
    public static void main(String[] args) {
        int[] arr = { 4, 5, 6, 7, 0, 1, 2 };
        int key = 0;
        int start = 0;
        int end = arr.length - 1;

        while (start <= end) {
            int mid = (start + end) / 2;
            if (arr[mid] == key) {
                System.out.print(mid);
                break;
            }

            if (arr[start] <= arr[mid]) {
                if (arr[start] <= key && key < arr[mid]) {
                    end = mid - 1;
                } else {
                    start = mid + 1;
                }
            } else {
                if (arr[mid] < key && key <= arr[end]) {
                    start = mid + 1;
                } else {
                    end = mid - 1;
                }
            }
        }
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\jdk-20\bin\java.exe' '-XX:+ShowCodeDetailsInExcept
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa25297373b22f95547702d2\redhat.java\jdt_ws\abstract_65
4
PS C:\Users\nithi\Downloads\abstract>
```

TIME COMPLEXITY : O(log n)

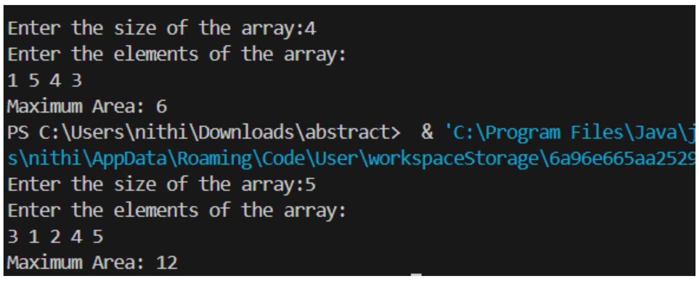
SPACE COMPLEXITY : O(1)

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

CODE :

```java
import java.util.*;

public class Container {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array:");
        int n = sc.nextInt();
        System.out.println("Enter the elements of the array:");
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        int i = 0, j = n - 1;
        int res = 0;
        while (i <= j) {
            int area = Math.min(arr[i], arr[j]) * (j - i);
            res = Math.max(area, res);
            if (arr[i] < arr[j])
                i++;
            else
                j--;
        }
        System.out.println("Maximum Area: " + res);

    }
}
```

OUTPUT :



```
Enter the size of the array:4
Enter the elements of the array:
1 5 4 3
Maximum Area: 6
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\j
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa2529
Enter the size of the array:5
Enter the elements of the array:
3 1 2 4 5
Maximum Area: 12
```

TIME COMPLEXITY : O(n)

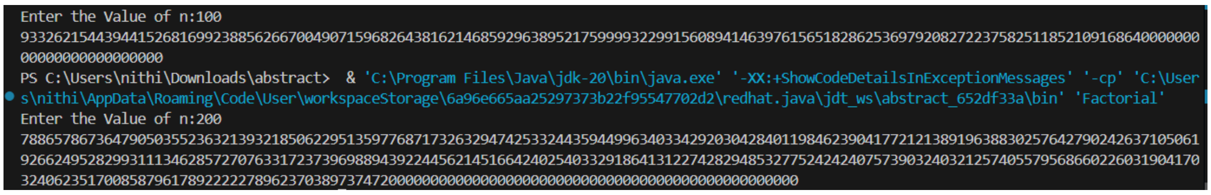
SPACE COMPLEXITY : O(1)

5. Find the Factorial of a large number
   Input: 100 Output:
   93326215443944152681699238856266700490715968264381621468592963895217599993
   22991560894146397615651828625369792082722375825118521091686400000000000000
   00000000 00
   Input: 50 Output:
   30414093201713378043612608166064768844377641568960512000000000000

CODE :

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

public class Factorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Value of n:");
        int n = sc.nextInt();
        BigInteger result = new BigInteger("1");
        for (int i = 2; i <= n; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        System.out.println(result);
    }
}
```

OUTPUT :



```
Enter the Value of n:100
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000
000000000000000000
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\jdk-20\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\User
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa25297373b22f95547702d2\redhat.java\jdt_ws\abstract_652df33a\bin' 'Factorial'
Enter the Value of n:200
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984623904177212138919638830257642790242637105061
926624952829931113462857270763317237396988943922445621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170
3240623517008587961789222227896237038973747200000000000000000000000000000000000000000000000000
```

TIME COMPLEXITY : O(n)

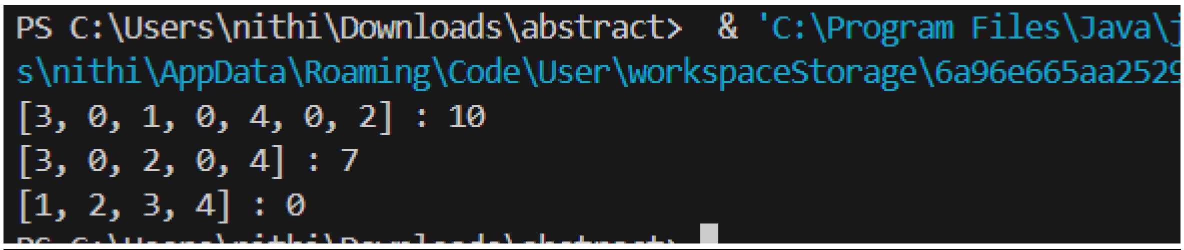
SPACE COMPLEXITY : O(n)

6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.
   Input: arr[] = {3, 0, 1, 0, 4, 0, 2} Output: 10 Explanation: The expected rainwater to be trapped is shown in the above image.
   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 :

```java
import java.util.Arrays;

public class Trapping {
    public static void main(String[] args) {
        trap(new int[] { 3, 0, 1, 0, 4, 0, 2 });
        trap(new int[] { 3, 0, 2, 0, 4 });
        trap(new int[] { 1, 2, 3, 4 });
    }

    public static void trap(int[] arr) {
        System.out.print(Arrays.toString(arr) + " : ");
        int i = 0, j = arr.length - 1;
        int left = arr[i], right = arr[j];
        int ans = 0;
        while (i < j) {
            if (left < right) {
                i++;
                left = Math.max(left, arr[i]);
                ans += left - arr[i];
            } else {
                j--;
                right = Math.max(right, arr[j]);
                ans += right - arr[j];
            }
        }
        System.out.println(ans);
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\j
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa2529
[3, 0, 1, 0, 4, 0, 2] : 10
[3, 0, 2, 0, 4] : 7
[1, 2, 3, 4] : 0
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

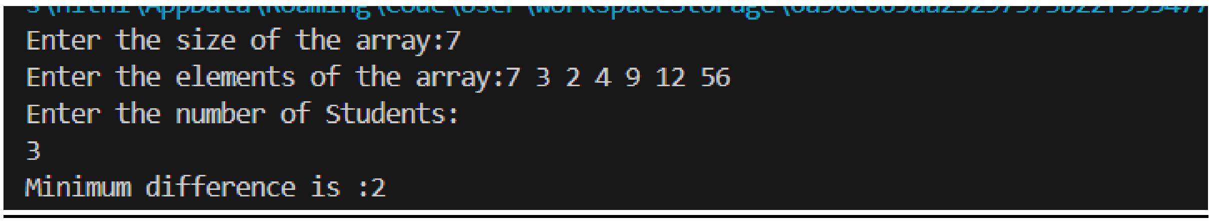
7. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. 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 :

```java
import java.util.*;
public class Chocolate{
  public static void main(String[] args){
            Scanner sc=new Scanner(System.in);
            System.out.print("Enter the size of the array:");
            int n=sc.nextInt();
            System.out.print("Enter the elements of the array:");
            int[] arr=new int[n];
            for(int i=0;i<n;i++) arr[i]=sc.nextInt();
            System.out.println("Enter the number of Students:");
            int m=sc.nextInt();
            Arrays.sort(arr);
            int res=Integer.MAX_VALUE;
            for(int i=0;i<=n-m;i++){
                   res=Math.min(arr[i+m-1]-arr[i],res);
            }
            System.out.println("Minimum difference is :"+res);
```

```java
    }
}
```

OUTPUT :



```
Enter the size of the array:7
Enter the elements of the array:7 3 2 4 9 12 56
Enter the number of Students:
3
Minimum difference is :2
```

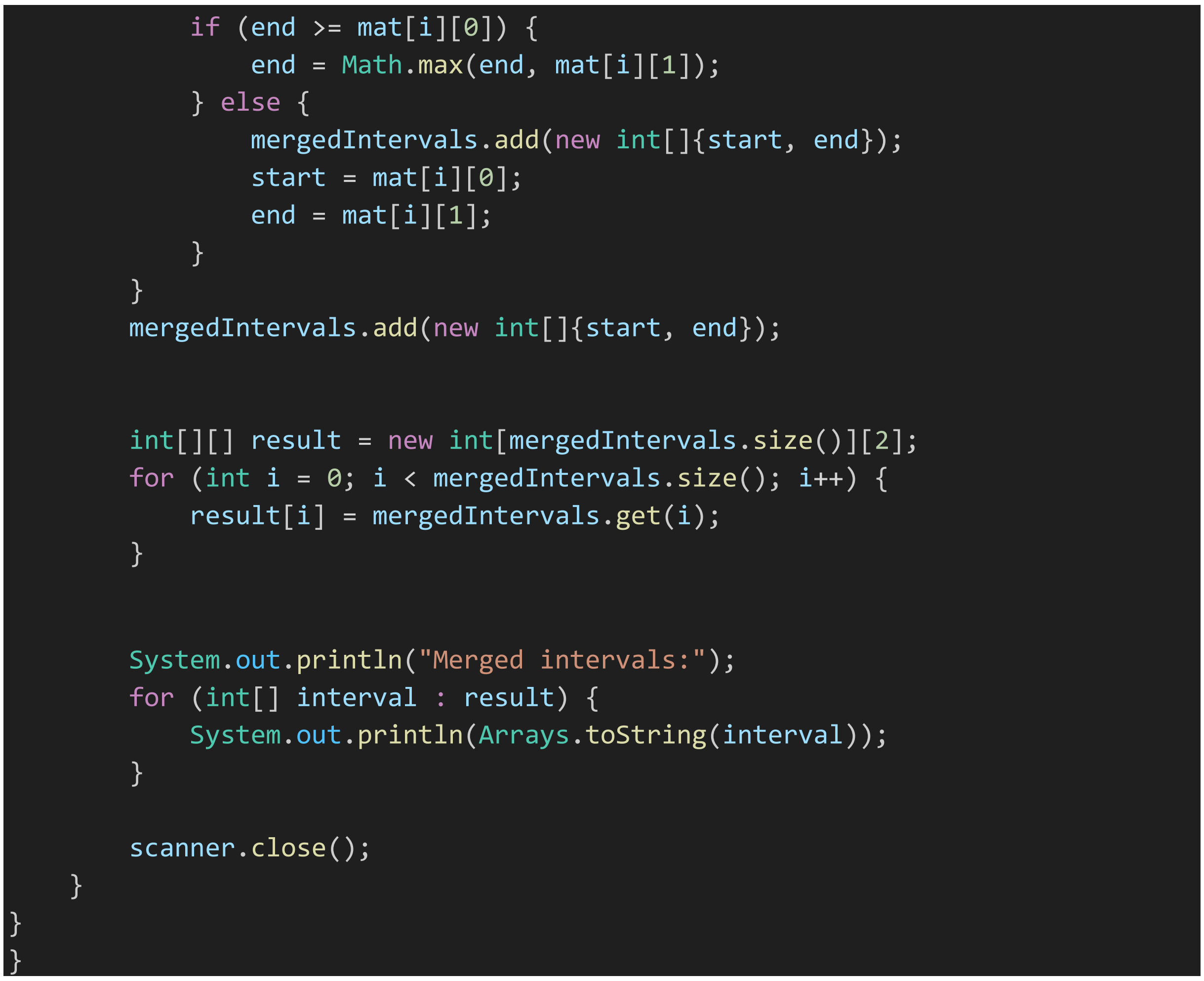
TIME COMPLEXITY : O(n log n)

SPACE COMPLEXITY : O(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].

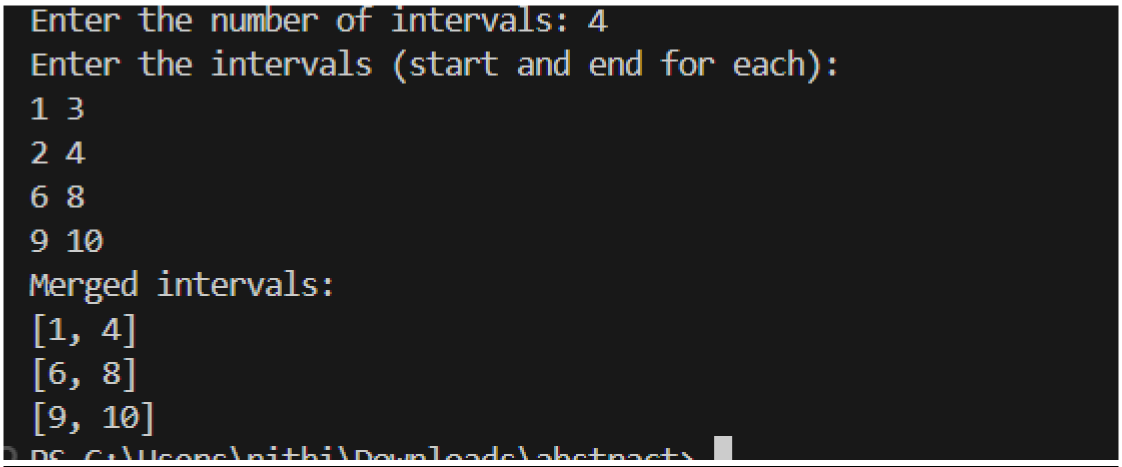
CODE :

```java
import java.util.*;

public class Merge {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of intervals: ");
        int n = scanner.nextInt();
        int[][] mat = new int[n][2];
        System.out.println("Enter the intervals (start and end for each):");
        for (int i = 0; i < n; i++) {
            mat[i][0] = scanner.nextInt();
            mat[i][1] = scanner.nextInt();
        }
        Arrays.sort(mat, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> mergedIntervals = new ArrayList<>();
        int start = mat[0][0];
        int end = mat[0][1];

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



```java
            if (end >= mat[i][0]) {
                end = Math.max(end, mat[i][1]);
            } else {
                mergedIntervals.add(new int[]{start, end});
                start = mat[i][0];
                end = mat[i][1];
            }
        }
        mergedIntervals.add(new int[]{start, end});


        int[][] result = new int[mergedIntervals.size()][2];
        for (int i = 0; i < mergedIntervals.size(); i++) {
            result[i] = mergedIntervals.get(i);
        }


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

        scanner.close();
    }
}
}
```

OUTPUT :



```
Enter the number of intervals: 4
Enter the intervals (start and end for each):
1 3
2 4
6 8
9 10
Merged intervals:
[1, 4]
[6, 8]
[9, 10]
PS C:\Users\nithi\Downloads\abstract>
```

TIME COMPLEXITY : O(n log n)

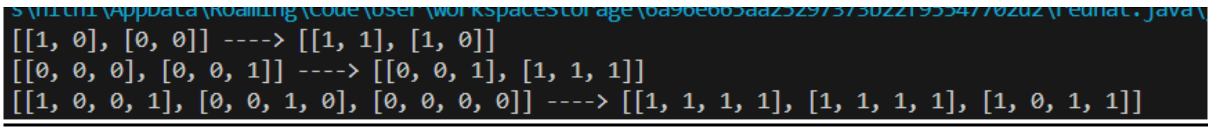
SPACE COMPLEXITY : O(n)

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

CODE :

```
import java.util.*;

public class Boolean {
    public static void main(String[] args) {
        updateMatrix(new int[][] { { 1, 0 }, { 0, 0 } });
        updateMatrix(new int[][] { { 0, 0, 0 }, { 0, 0, 1 } });
        updateMatrix(new int[][] { { 1, 0, 0, 1 }, { 0, 0, 1, 0 }, { 0, 0, 0, 
0 } });
    }

    public static void updateMatrix(int[][] matrix) {
        System.out.print(Arrays.deepToString(matrix) + " ----> ");

        int rowCount = matrix.length;
        int colCount = matrix[0].length;

        Set<Integer> rowsWithOne = new HashSet<>();
        Set<Integer> colsWithOne = new HashSet<>();
        for (int row = 0; row < rowCount; row++) {
            for (int col = 0; col < colCount; col++) {
                if (matrix[row][col] == 1) {
                    rowsWithOne.add(row);
                    colsWithOne.add(col);
                }
            }
        }
        for (int row = 0; row < rowCount; row++) {
            for (int col = 0; col < colCount; col++) {
                if (rowsWithOne.contains(row) || colsWithOne.contains(col)) {
                    matrix[row][col] = 1;
                } else {
                    matrix[row][col] = 0;
                }
            }
        }

        System.out.println(Arrays.deepToString(matrix));

    }
}
```

OUTPUT :



```
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e663aa23297373b22193547702d2\redhat.java\
[[1, 0], [0, 0]] ----> [[1, 1], [1, 0]]
[[0, 0, 0], [0, 0, 1]] ----> [[0, 0, 1], [1, 1, 1]]
[[1, 0, 0, 1], [0, 0, 1, 0], [0, 0, 0, 0]] ----> [[1, 1, 1, 1], [1, 1, 1, 1], [1, 0, 1, 1]]
```

TIME COMPLEXITY : O(n*m)

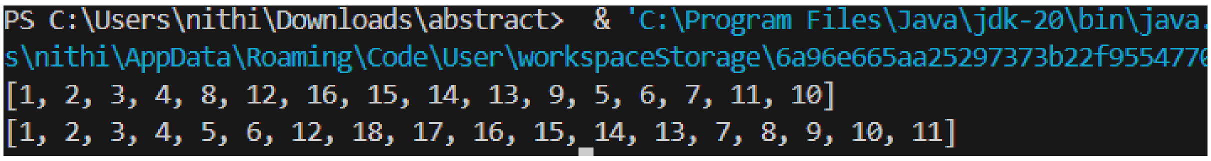
SPACE COMPLEXITY : O(n+m)

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 :

```java
import java.util.*;

public class Spiral {
    public static void main(String[] args) {
        spiral(new int[][] { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12
}, { 13, 14, 15, 16 } });
        spiral(new int[][] { { 1, 2, 3, 4, 5, 6 }, { 7, 8, 9, 10, 11, 12 }, {
13, 14, 15, 16, 17, 18 } });
    }

    public static void spiral(int[][] arr) {
        int top = 0;
        int bottom = arr.length - 1;
        int left = 0;
        int right = arr[0].length - 1;
        int size = arr.length * arr[0].length;
        List<Integer> l = new ArrayList<>();
        int i = 0;
        while (i != size) {
            for (int n = left; n <= right && (i != size); n++) {
                l.add(arr[top][n]);
                i++;
            }
            for (int n = top; n < bottom && (i != size); n++) {
                l.add(arr[n + 1][right]);
                i++;
            }
            for (int n = right - 1; n >= left && (i != size); n--) {
                l.add(arr[bottom][n]);
                i++;
            }
            for (int n = bottom - 1; n >= top + 1 && (i != size); n--) {
                l.add(arr[n][left]);
                i++;
            }
            top++;
            bottom--;
            left++;
            right--;
        }
        System.out.println(l);
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\jdk-20\bin\java.
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa25297373b22f955477
[1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]
[1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11]
```

TIME COMPLEXITY : O(n)

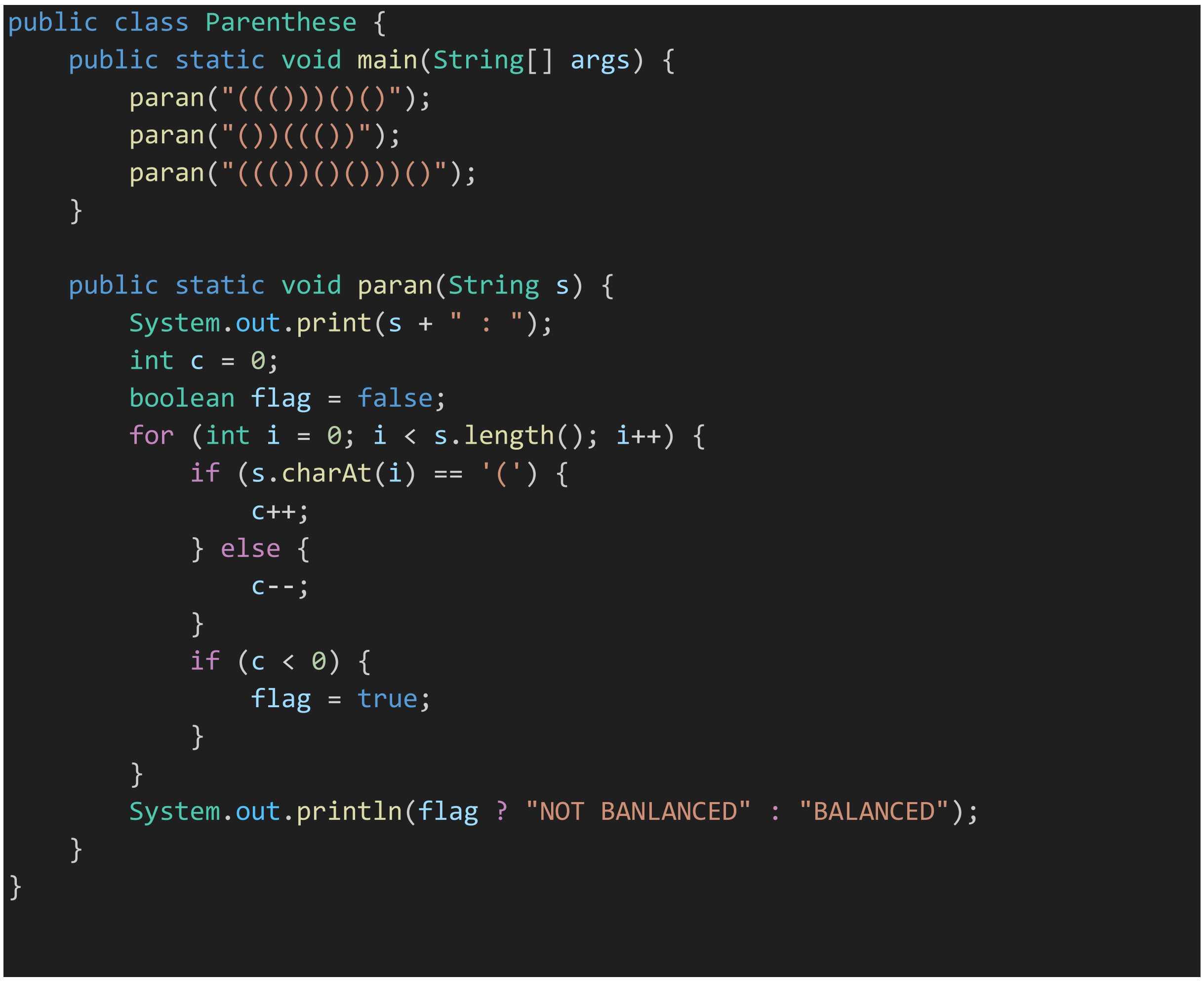
SPACE COMPLEXITY : O(n)

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

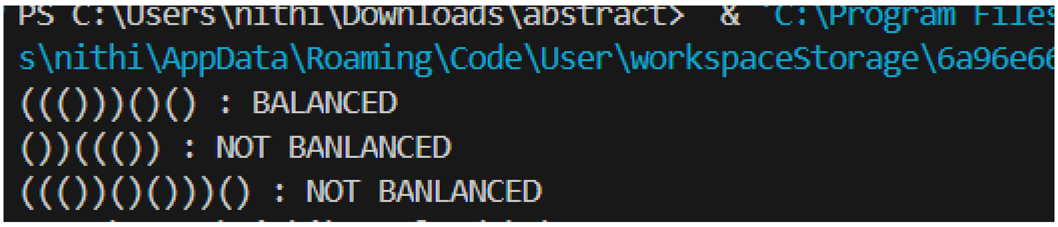
Input: str = “())((())” Output: Not Balanced

CODE :



```java
public class Parenthese {
    public static void main(String[] args) {
        paran("((()))()()");
        paran("())((())");
        paran("((())()()))()");
    }

    public static void paran(String s) {
        System.out.print(s + " : ");
        int c = 0;
        boolean flag = false;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '(') {
                c++;
            } else {
                c--;
            }
            if (c < 0) {
                flag = true;
            }
        }
        System.out.println(flag ? "NOT BANLANCED" : "BALANCED");
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e6
((()))()() : BALANCED
())((()) : NOT BANLANCED
((())()()))() : NOT BANLANCED
```

TIME COMPLEXITY : O(n)

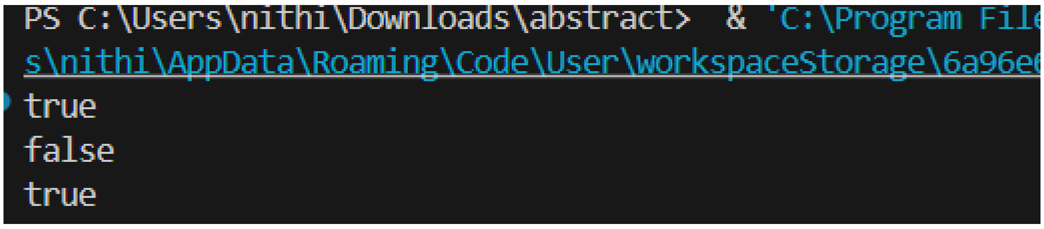
SPACE COMPLEXITY : O(1)

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.

<u>CODE :</u>

```java
import java.util.*;

public class Anagram {
    public static void main(String[] args) {
        checkAnagram("geeks", "kseeg");
        checkAnagram("allergy", "allergic");
        checkAnagram("g", "g");
    }

    public static void checkAnagram(String str1, String str2) {
        if (str1.length() != str2.length()) {
            System.out.println(false);
            return;
        }


        char[] arr1 = str1.toCharArray();
        char[] arr2 = str2.toCharArray();

        Arrays.sort(arr1);
        Arrays.sort(arr2);
        boolean isAnagram = Arrays.equals(arr1, arr2);
        System.out.println(isAnagram);
    }
}
```

<u>OUTPUT :</u>


```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program File
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e
true
false
true
```

TIME COMPLEXITY : O(n log n)

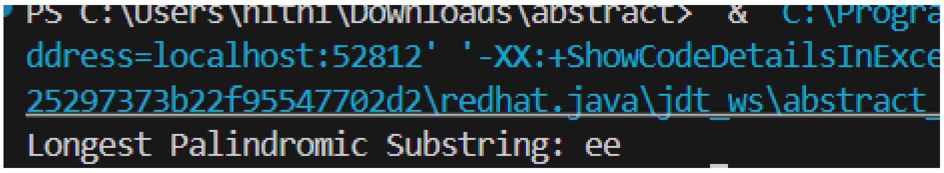
SPACE COMPLEXITY : O(n)

15. Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.
Input: str = “forgeeksskeegfor” Output: “geeksskeeg” Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

CODE :

```java
// Longest palindromic substring
import java.util.*;

public class LongestPaliondrome {
    public static void main(String[] args) {
        String s = "Geeks";
        String res = "";
        int resLen = 0;

        for (int i = 0; i < s.length(); i++) {
            int l = i, r = i;
            while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
                if ((r - l + 1) > resLen) {
                    res = s.substring(l, r + 1);
                    resLen = r - l + 1;
                }
                l--;
                r++;
            }

            l = i;
            r = i + 1;
            while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
                if ((r - l + 1) > resLen) {
                    res = s.substring(l, r + 1);
                    resLen = r - l + 1;
                }
                l--;
                r++;
            }
        }

        System.out.println("Longest Palindromic Substring: " + res);
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & C:\Progra
ddress=localhost:52812' '-XX:+ShowCodeDetailsInExce
25297373b22f95547702d2\redhat.java\jdt_ws\abstract_
Longest Palindromic Substring: ee
```

TIME COMPLEXITY : O(n^2)

SPACE COMPLEXITY : O(1)

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 CommonPrefix {
    public static void main(String[] args) {
        findCommonPrefix(new String[] { "geeksforgeeks", "geeks", "geek",
"geezer" });
        findCommonPrefix(new String[] { "hello", "world" });
    }

    public static void findCommonPrefix(String[] words) {
        System.out.print(Arrays.toString(words) + " : ");

        if (words == null || words.length == 0) {
            System.out.println(-1);
            return;
        }

        String commonPrefix = words[0];
        for (int i = 1; i < words.length; i++) {
            while (words[i].indexOf(commonPrefix) != 0) {
                commonPrefix = commonPrefix.substring(0, commonPrefix.length()
- 1);
```

```java
                if (commonPrefix.isEmpty()) {
                    System.out.println(-1);
                    return;
                }
            }
        }

        System.out.println(commonPrefix.length() > 0 ? commonPrefix : -1);
    }
}
```

<u>OUTPUT :</u>

```
[geeksforgeeks, geeks, geek, geezer] : gee
[hello, world] : -1
```

TIME COMPLEXITY : O(n*m)

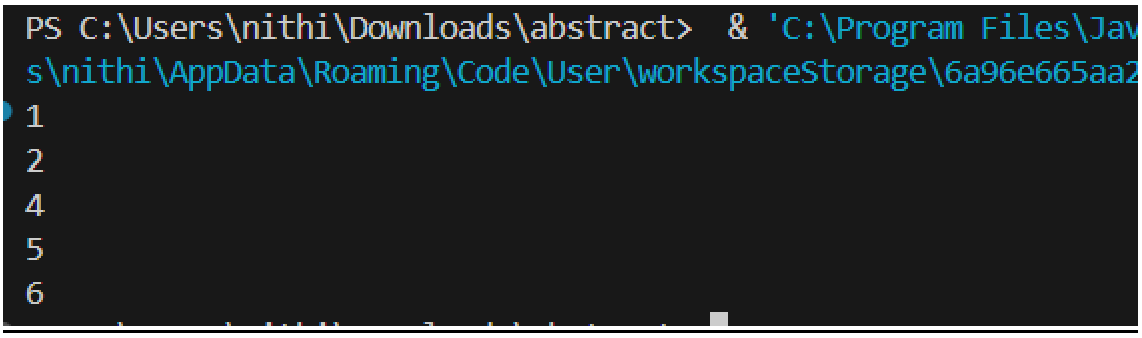
SPACE COMPLEXITY : O(1)

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]

<u>CODE :</u>

```java
import java.util.*;

public class StackMiddle {

    public static void main(String[] args) {
        Stack<Integer> st = new Stack<>();
        st.push(1);
        st.push(2);
        st.push(3);
        st.push(4);
        st.push(5);
        st.push(6);
        int n = st.size() / 2;
        Stack<Integer> temp = new Stack<>();

        for (int i = 0; i < n; i++) {
            temp.push(st.pop());
        }
        st.pop();

        while (!temp.isEmpty()) {
            st.push(temp.pop());
        }

        for (int i : st) {
            System.out.println(i);
        }
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Jav
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa2
1
2
4
5
6
```

TIME COMPLEXITY : O(n)

SPACE 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.

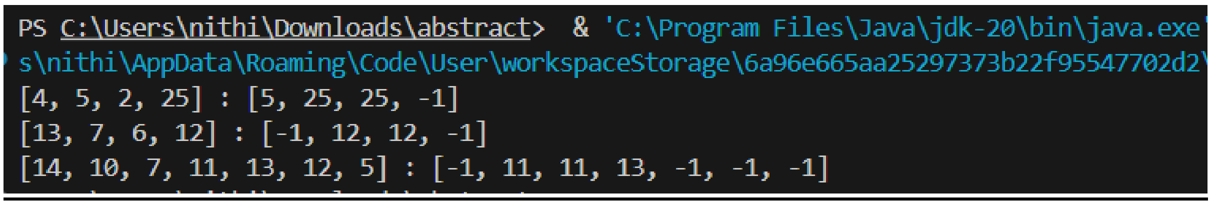
Input: arr[] = [4 , 5 , 2 , 25] Output: 4 5 2 –> 5 –> 25 –> 25 25 –> -1 Explanation: Except 25 every element has an element greater than them present on the right side

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.*;

public class NextGreater {
    public static void main(String[] args) {
        findNextGreater(new int[] { 4, 5, 2, 25 });
        findNextGreater(new int[] { 13, 7, 6, 12 });
        findNextGreater(new int[] { 14, 10, 7, 11, 13, 12, 5 });
    }

    public static void findNextGreater(int[] arr) {
        System.out.print(Arrays.toString(arr) + " : ");
        int[] result = new int[arr.length];
        Stack<Integer> stack = new Stack<>();

        Arrays.fill(result, -1);

        for (int i = arr.length - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }

            if (!stack.isEmpty()) {
                result[i] = stack.peek();
            }
            stack.push(arr[i]);
        }
        System.out.println(Arrays.toString(result));
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\jdk-20\bin\java.exe'
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa25297373b22f95547702d2\
[4, 5, 2, 25] : [5, 25, 25, -1]
[13, 7, 6, 12] : [-1, 12, 12, -1]
[14, 10, 7, 11, 13, 12, 5] : [-1, 11, 11, 13, -1, -1, -1]
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

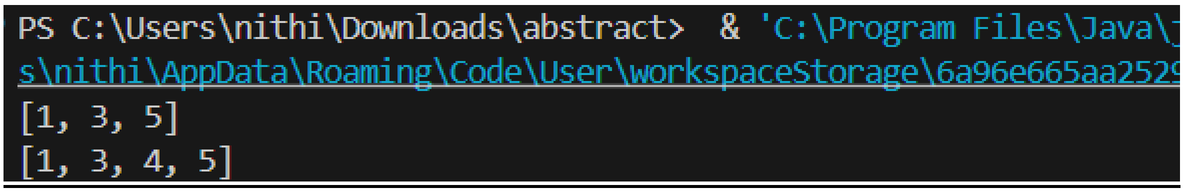
19. Print Right View of a Binary Tree Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

CODE :

```
import java.util.*;

class TreeNode{
    int data;
    TreeNode left ;
```

```java
    TreeNode right ;

    TreeNode(int data){
        this.data = data;
        left = null;
        right = null;
    }
}

public class rightview {

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

        TreeNode root1 = new TreeNode(1);
        root1.left = new TreeNode(2);
        root1.right = new TreeNode(3);
        root1.left.left = new TreeNode(4);
        root1.left.left.right = new TreeNode(5);
        view(root1);
    }

    public static void view(TreeNode root){
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        List<Integer> l = new ArrayList<>();
        while(!q.isEmpty()){
            TreeNode curr = q.poll();
            if(curr.left != null){
                q.add(curr.left);
            }
            if(curr.right != null){
                if(!l.contains(curr.data)) l.add(curr.data);
                l.add(curr.right.data);
                q.add(curr.right);
            }
        }
        System.out.println(l);
    }
}
```

OUTPUT :



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program Files\Java\j
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a96e665aa2529
[1, 3, 5]
[1, 3, 4, 5]
```

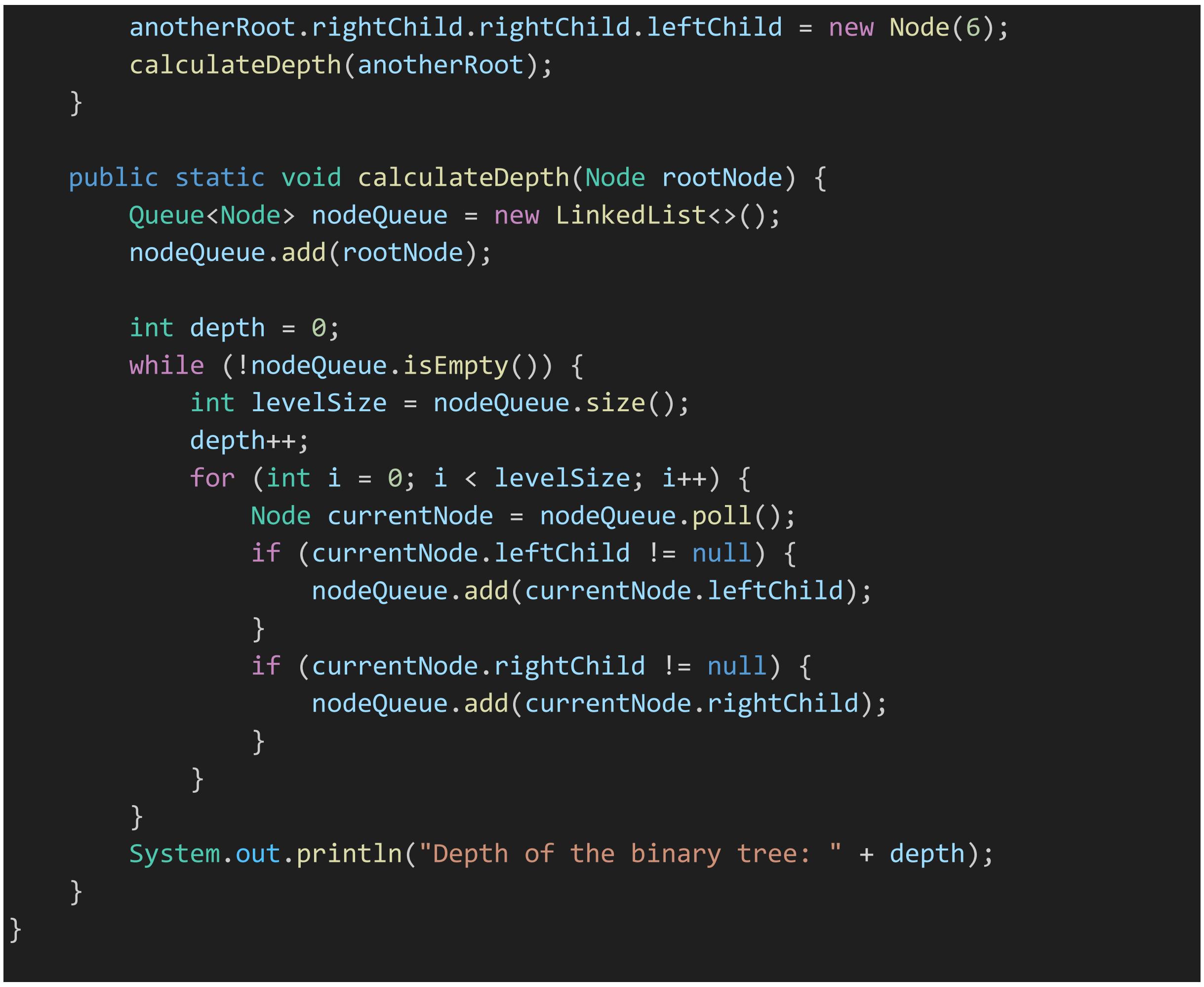
TIME COMPLEXITY : O(n)

SPACE 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.

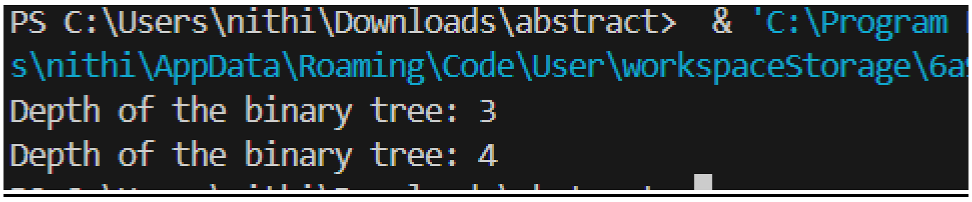
CODE :

```java
import java.util.LinkedList;
import java.util.Queue;

class Node {
    int value;
    Node leftChild;
    Node rightChild;

    Node(int value) {
        this.value = value;
        leftChild = null;
        rightChild = null;
    }
}

public class Depth {
    public static void main(String[] args) {
        Node rootNode = new Node(12);
        rootNode.rightChild = new Node(18);
        rootNode.leftChild = new Node(8);
        rootNode.leftChild.leftChild = new Node(5);
        rootNode.leftChild.rightChild = new Node(11);
        calculateDepth(rootNode);

        Node anotherRoot = new Node(1);
        anotherRoot.leftChild = new Node(2);
        anotherRoot.leftChild.leftChild = new Node(4);
        anotherRoot.rightChild = new Node(3);
        anotherRoot.rightChild.rightChild = new Node(5);
        anotherRoot.rightChild.rightChild.rightChild = new Node(7);
```



```java
        anotherRoot.rightChild.rightChild.leftChild = new Node(6);
        calculateDepth(anotherRoot);
    }

    public static void calculateDepth(Node rootNode) {
        Queue<Node> nodeQueue = new LinkedList<>();
        nodeQueue.add(rootNode);

        int depth = 0;
        while (!nodeQueue.isEmpty()) {
            int levelSize = nodeQueue.size();
            depth++;
            for (int i = 0; i < levelSize; i++) {
                Node currentNode = nodeQueue.poll();
                if (currentNode.leftChild != null) {
                    nodeQueue.add(currentNode.leftChild);
                }
                if (currentNode.rightChild != null) {
                    nodeQueue.add(currentNode.rightChild);
                }
            }
        }
        System.out.println("Depth of the binary tree: " + depth);
    }
}
```

<u>OUTPUT :</u>



```
PS C:\Users\nithi\Downloads\abstract>  & 'C:\Program
s\nithi\AppData\Roaming\Code\User\workspaceStorage\6a
Depth of the binary tree: 3
Depth of the binary tree: 4
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)