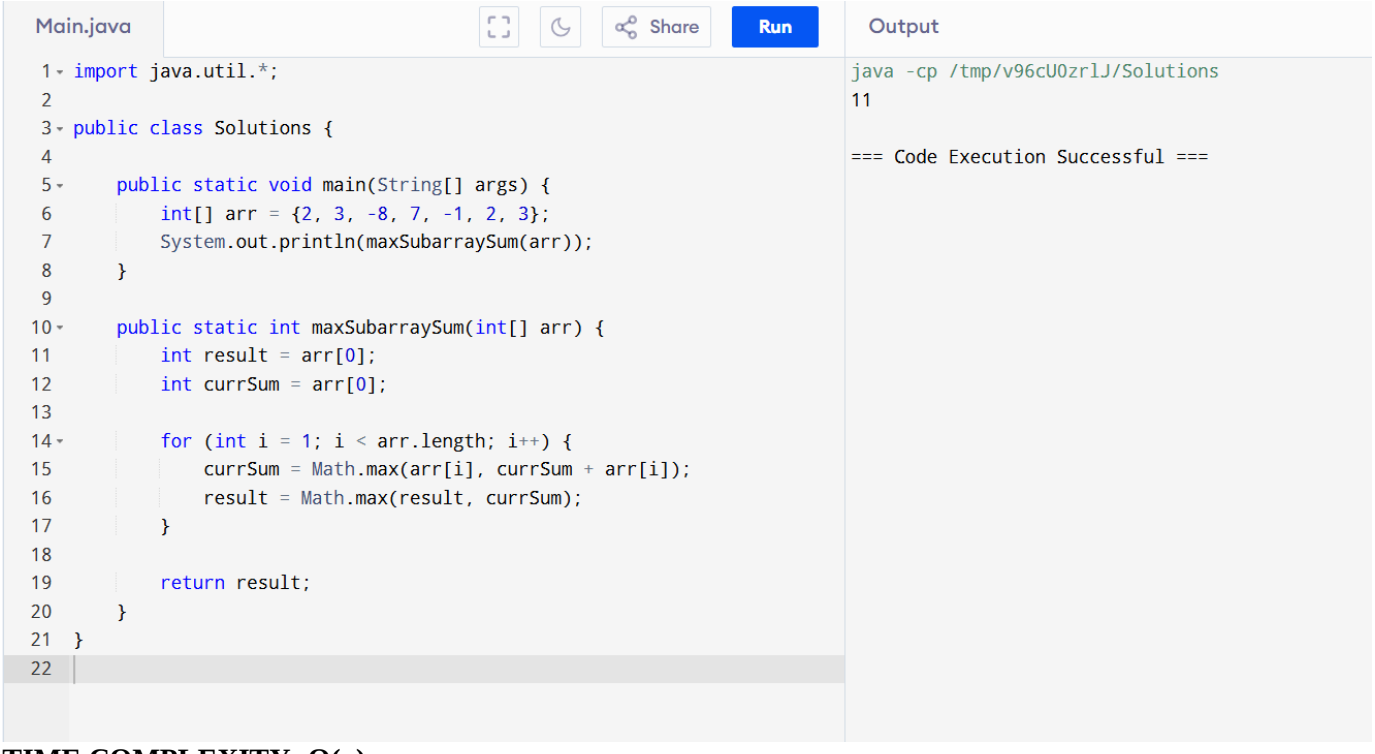
Coding Question Practice test- 01
9/11/2024

Name:Kowsick K
Reg No:22CS083
Dept : CSE

1. Maximum Subarray Sum – Kadane‟s Algorithm:

```
import java.util.*;

public class Solutions {

    public static void main(String[] args) {
        int[] arr = {2, 3, -8, 7, -1, 2, 3};
        System.out.println(maxSubarraySum(arr));
    }

    public static int maxSubarraySum(int[] arr) {
        int result = arr[0];
        int currSum = arr[0];

        for (int i = 1; i < arr.length; i++) {
            currSum = Math.max(arr[i], currSum + arr[i]);
            result = Math.max(result, currSum);
        }

        return result;
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(n)

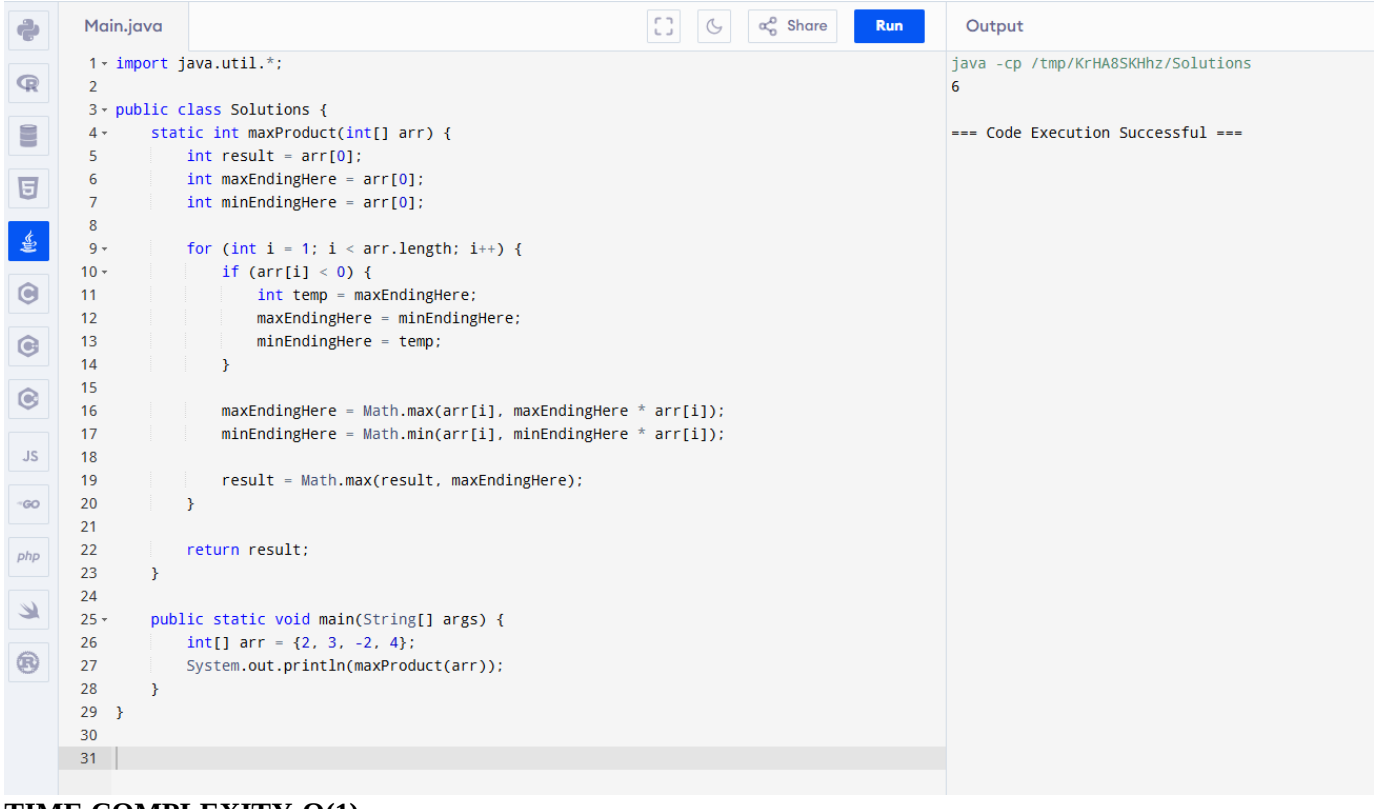
2. Maximum Product Subarray

```
import java.util.*;

public class Solutions {
   static int maxProduct(int[] arr) {
      int result = arr[0];
      int maxEndingHere = arr[0];
      int minEndingHere = arr[0];

      for (int i = 1; i < arr.length; i++) {
         if (arr[i] < 0) {
            int temp = maxEndingHere;
            maxEndingHere = minEndingHere;
            minEndingHere = temp;
         }

         maxEndingHere = Math.max(arr[i], maxEndingHere * arr[i]);
         minEndingHere = Math.min(arr[i], minEndingHere * arr[i]);

         result = Math.max(result, maxEndingHere);
      }

      return result;
```

```
    }

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

`

OUTPUT:



TIME COMPLEXITY:O(1)

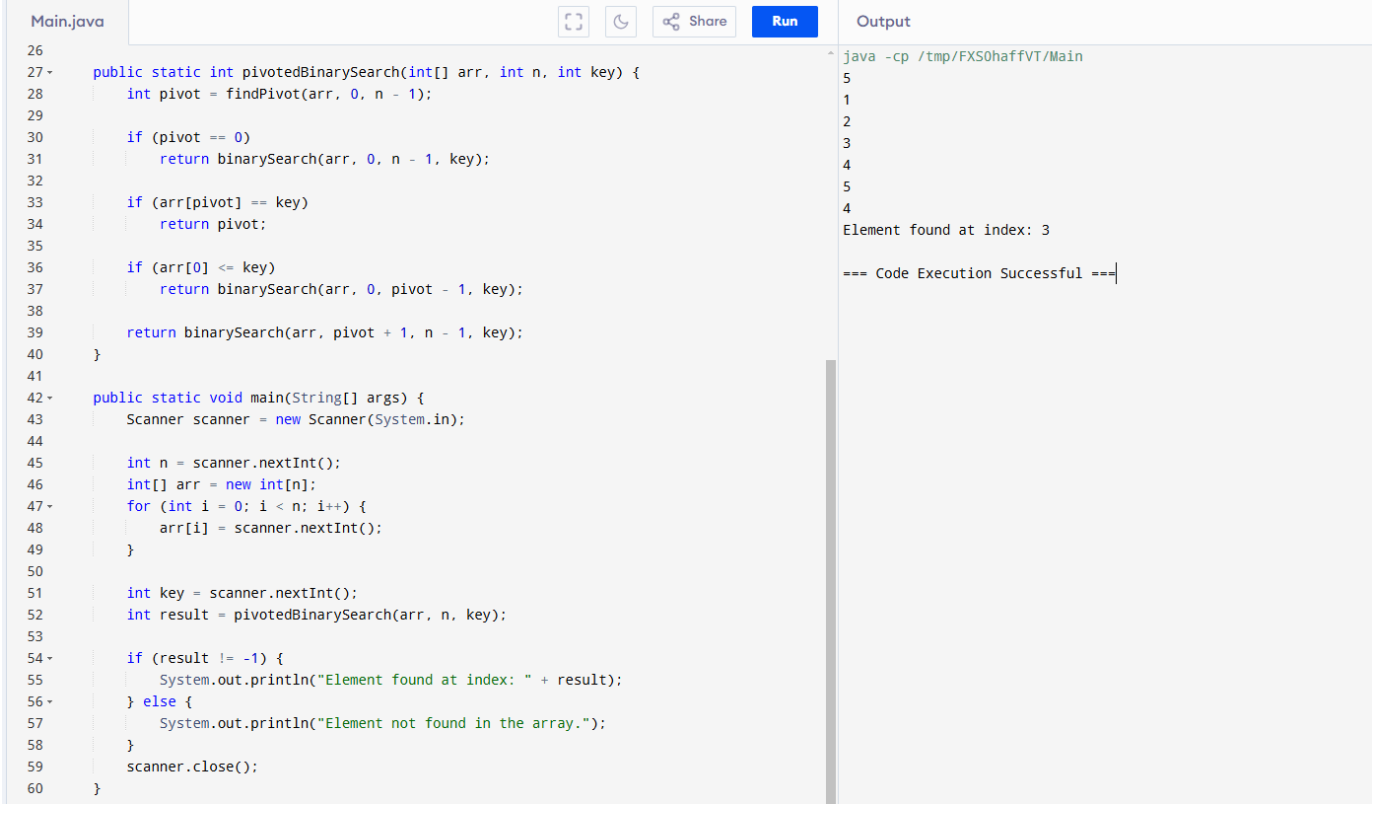
3. Search in a sorted and rotated Array

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

public class Main {
    public static int binarySearch(int[] arr, int low, int high, int x) {
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == x) return mid;
            if (arr[mid] < x) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }
)
    public static int findPivot(int[] arr, int low, int high) {
```

```
        while (low < high) {

            if (arr[low] <= arr[high])
                return low;

            int mid = (low + high) / 2;

            if (arr[mid] > arr[high])
                low = mid + 1;

            else
                high = mid;
        }
        return low;
    }

    public static int pivotedBinarySearch(int[] arr, int n, int key) {
        int pivot = findPivot(arr, 0, n - 1);

        if (pivot == 0)
            return binarySearch(arr, 0, n - 1, key);

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

        if (arr[0] <= key)
            return binarySearch(arr, 0, pivot - 1, key);
        return binarySearch(arr, pivot + 1, n - 1, key);
    }

    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.println("Enter " + n + " elements of the array (sorted and
pivoted):");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.print("Enter the key to search: ");
        int key = scanner.nextInt();

        int result = pivotedBinarySearch(arr, n, key);

        if (result != -1) {
            System.out.println("Element found at index: " + result);
```

```
        } else {
            System.out.println("Element not found in the array.");
        }
        scanner.close();
    }
}
```

OUTPUT:



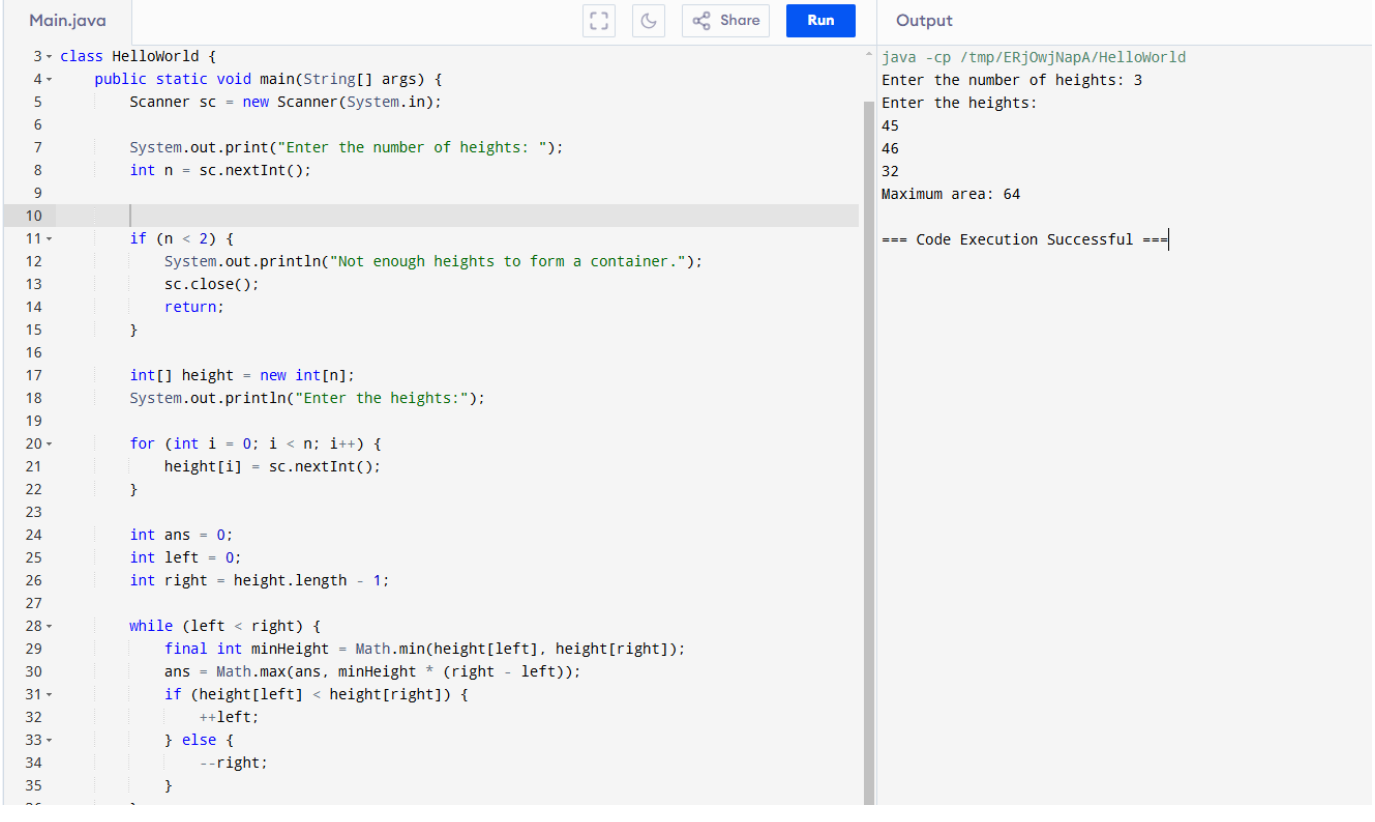
TIME COMPLEXITY: O(log n)

4. Container with Most Water

```
import java.util.Scanner;

class HelloWorld {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of heights: ");
        int n = sc.nextInt();
        int[] height = new int[n];
        System.out.println("Enter the heights:");
        for (int i = 0; i < n; i++) {
            height[i] = sc.nextInt();
        }
        int ans = 0;
        int left = 0;
        int right = height.length - 1;
```

```
        while (left < right) {
            final int minHeight = Math.min(height[l], height[r]);
            ans = Math.max(ans, minHeight * (right - left));
            if (height[left] < height[right])
                ++left;
            else
                --right;
        }

        System.out.println("Maximum area: " + ans);
        sc.close();
    }
}
```

OUTPUT:



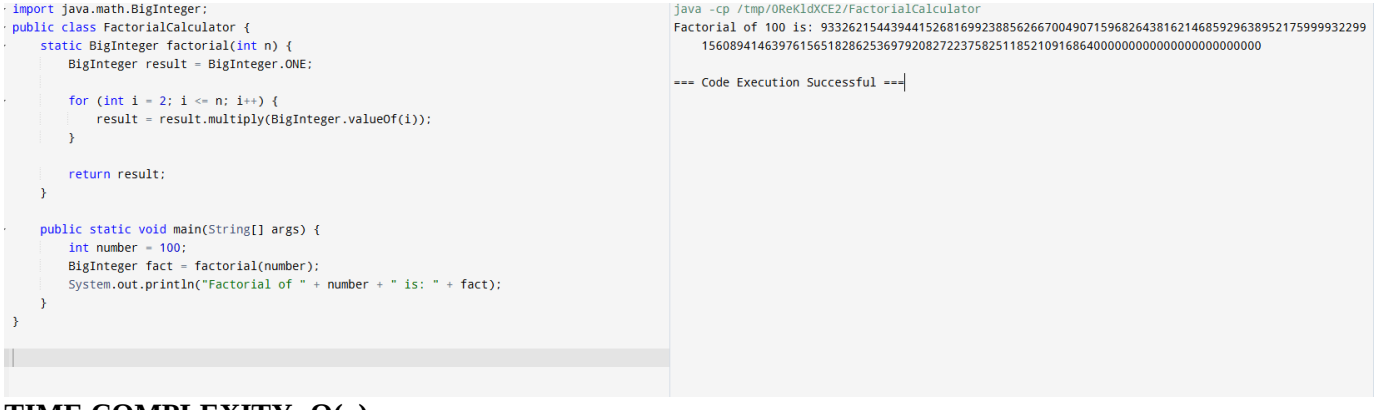
TIME COMPLEXITY: O(n)

5. Find the Factorial of a large number

```
import java.math.BigInteger;
public class FactorialCalculator {
    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) {
        int number = 100;
        BigInteger fact = factorial(number);
        System.out.println("Factorial of " + number + " is: " + fact);
    }
}
```

OUTPUT:



```
import java.math.BigInteger;
public class FactorialCalculator {
    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) {
        int number = 100;
        BigInteger fact = factorial(number);
        System.out.println("Factorial of " + number + " is: " + fact);
    }
}
```

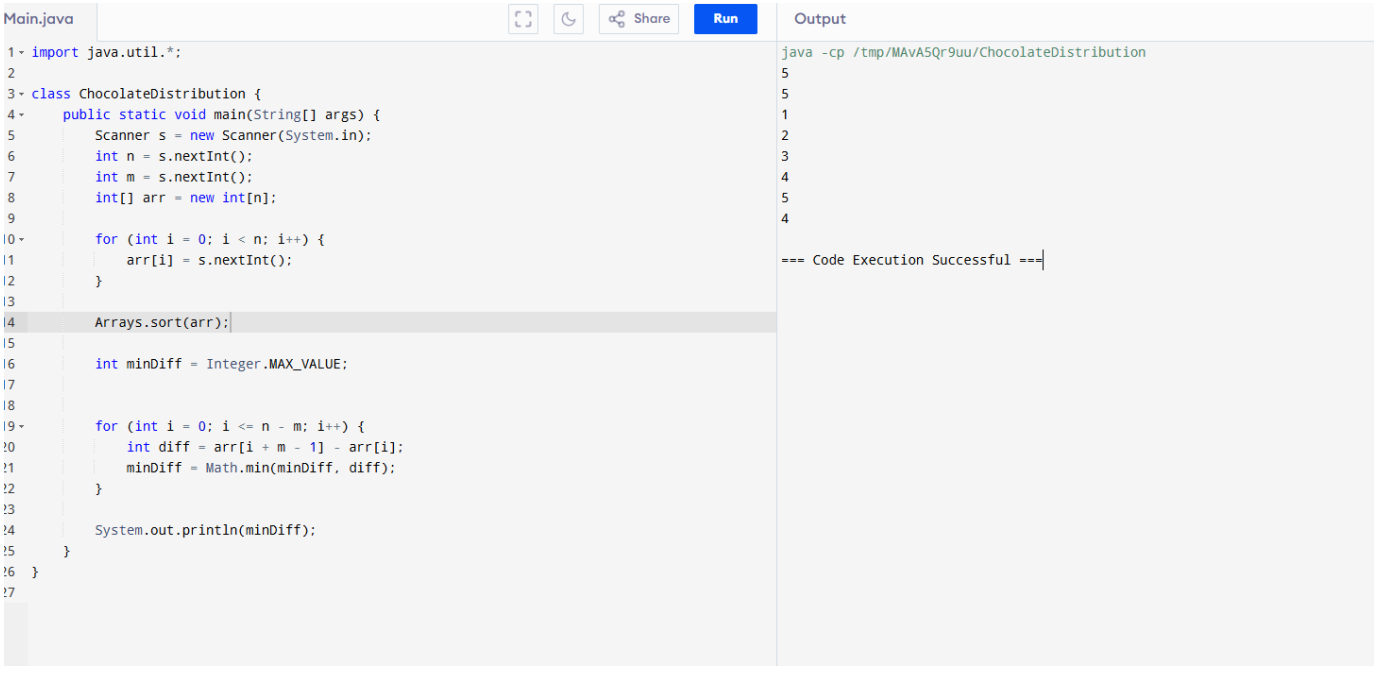
```
java -cp /tmp/OReKldXCE2/FactorialCalculator
Factorial of 100 is: 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(n)

6. TrappingRainwater Problem

```
import java.util.*;

class ChocolateDistribution {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();
        int m = s.nextInt();
        int[] arr = new int[n];
        int[] subarr = new int[m];
        for (int i = 0; i < n; i++) {
            arr[i] = s.nextInt();
        }
        arr.sort();
        for (int i = 0; i < m - 1; i++) {
            subarr.append(arr[i]);
        }
        int lar = Math.max(subarr);
        int sma = Math.min(subarr);
        int diff = lar - sma;
        System.out.println(diff);
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(n log n)

7. ROTATED ARRAY

```
public class searchinrotatedarray {
public static int search(int[] arr, int key) {
int low = 0, high = arr.length - 1;
while (low <= high) {
int mid = low + (high - low) / 2;
if (arr[mid] == key) {
return mid;
}
if (arr[low] <= arr[mid]) {
if (arr[low] <= key && key < arr[mid]) {
high = mid - 1;
} else {
low = mid + 1;
}
} else {
if (arr[mid] < key && key <= arr[high]) {
low = mid + 1;
} else {
high = mid - 1;
}
}
}
```

return -1;
}
public static void main(String[] args) {
int[] arr = {4, 5, 6, 7, 0, 1, 2};
System.out.println(search(arr, 0));
System.out.println(search(arr, 3));
}
}

OUTPUT:

```
public class searchinrotatedarray {
public static int search(int[] arr, int key) {
int low = 0, high = arr.length - 1;
while (low <= high) {
int mid = low + (high - low) / 2;
if (arr[mid] == key) {
return mid;
}

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

return -1;
}
public static void main(String[] args) {
int[] arr = {4, 5, 6, 7, 0, 1, 2};
System.out.println(search(arr, 0));
System.out.println(search(arr, 3));
}
}
```

```
java -cp /tmp/43aViAHLiV/searchinrotatedarray
4
-1

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(n log n)

8. Merge Overlapping Intervals

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

class IntervalMerger {

    static List<int[]> mergeOverlap(int[][] arr) {
        int n = arr.length;
        Arrays.sort(arr, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> res = new ArrayList<>();
        int[] currentInterval = arr[0];

        for (int i = 1; i < n; i++) {
            if (currentInterval[1] >= arr[i][0]) {
                currentInterval[1] = Math.max(currentInterval[1], arr[i][1]);
            } else {
                res.add(currentInterval);
```

```
            currentInterval = arr[i];
        }
    }
    res.add(currentInterval);

    return res;
}

public static void main(String[] args) {
    int[][] arr = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};
    List<int[]> res = mergeOverlap(arr);

    for (int[] interval : res) {
        System.out.println(interval[0] + " " + interval[1]);
    }
}
}
```

OUTPUT:

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

class IntervalMerger {

    static List<int[]> mergeOverlap(int[][] arr) {
        int n = arr.length;
        Arrays.sort(arr, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> res = new ArrayList<>();
        int[] currentInterval = arr[0];

        for (int i = 1; i < n; i++) {
            if (currentInterval[1] >= arr[i][0]) {
                currentInterval[1] = Math.max(currentInterval[1], arr[i][1]);
            } else {
                res.add(currentInterval);
                currentInterval = arr[i];
            }
        }
        res.add(currentInterval);

        return res;
    }

    public static void main(String[] args) {
        int[][] arr = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};
        List<int[]> res = mergeOverlap(arr);

        for (int[] interval : res) {
            System.out.println(interval[0] + " " + interval[1]);
        }
    }
}
```

```
java -cp /tmp/bc1MuZgG46/IntervalMerger
1 6
7 8

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(n log n)

9. BooleanMatrix Question

```
public class BooleanMatrix {
    public static void modifyMatrix(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        boolean[] rowFlags = new boolean[rows];  // Flags for rows
        boolean[] colFlags = new boolean[cols];  // Flags for columns
```

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

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

    public static void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] matrix1 = {{1, 0}, {0, 0}};
        System.out.println("Original Matrix 1:");
        printMatrix(matrix1);
        modifyMatrix(matrix1);
        System.out.println("Modified Matrix 1:");
        printMatrix(matrix1);

        int[][] matrix2 = {{0, 0, 0}, {0, 0, 1}};
        System.out.println("\nOriginal Matrix 2:");
        printMatrix(matrix2);
        modifyMatrix(matrix2);
        System.out.println("Modified Matrix 2:");
        printMatrix(matrix2);
    }
}
```

OUTPUT:

```java
public class BooleanMatrix {
    public static void modifyMatrix(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        boolean[] rowFlags = new boolean[rows];  // Flags for rows
        boolean[] colFlags = new boolean[cols];  // Flags for columns


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

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

    public static void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
```

```
java -cp /tmp/ggnMtu1SO6/BooleanMatrix
Original Matrix 1:
1 0
0 0
Modified Matrix 1:
1 1
1 0

Original Matrix 2:
0 0 0
0 0 1
Modified Matrix 2:
0 0 1
1 1 1

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(rows * cols)

10.Print a givenmatrix in spiral form

```
public class Spiralmatrix {

    public static void printSpiral(int[][] matrix) {
        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) {
    int[][] matrix1 = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };

    System.out.println("Spiral Order of Matrix:");
    printSpiral(matrix1);
    System.out.println();
}
}
```

OUTPUT:

```
public class Spiralmatrix {

    public static void printSpiral(int[][] matrix) {
        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) {
        int[][] matrix1 = {
```

```
java -cp /tmp/6272nkA8Oj/Spiralmatrix
Spiral Order of Matrix:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(rows * cols)

13.Check if given Parentheses expression is balanced or not

```
import java.util.Stack;
class Balanced {
public static String checkBalancedParentheses(String str) {
Stack<Character> stack = new Stack<>();
for (int i = 0; i < str.length(); i++) {
char ch = str.charAt(i);
if (ch == '(') {
stack.push(ch);
} else if (ch == ')') {
if (stack.isEmpty()) {
return "Not Balanced";
}
stack.pop();
}
}
return stack.isEmpty() ? "Balanced" : "Not Balanced";
}
public static void main(String[] args) {
String str1 = "((()))()()";
System.out.println(checkBalancedParentheses(str1));
String str2 = "())((())";
System.out.println(checkBalancedParentheses(str2));
}
}
```

OUTPUT:

```
D:\javaprograms\Practice ques\13>javac Balanced.java

D:\javaprograms\Practice ques\13>java Balanced
Balanced
Not Balanced
```

TIME COMPLEXITY: O(n) (single traversal)

14.Check if two Strings areAnagrams of each other

```
 import java.util.HashMap;

public class AnagramCheck {

    public static boolean checkAnagrams(String str1, String str2) {
        if (str1.length() != str2.length()) {
            return false;
        }

        HashMap<Character, Integer> charCount = new HashMap<>();

        for (char ch : str1.toCharArray()) {
            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);
        }

        for (char ch : str2.toCharArray()) {
```

```
            if (!charCount.containsKey(ch)) {
                return false;  // If a character is not in the map, return false
            }
            charCount.put(ch, charCount.get(ch) - 1);
            if (charCount.get(ch) == 0) {
                charCount.remove(ch);  // Remove the character if count becomes 0
            }
        }

        return charCount.isEmpty();
    }

    public static void main(String[] args) {
        String str1 = "geeks";
        String str2 = "kseeg";
        System.out.println(checkAnagrams(str1, str2)); // Output: true

        str1 = "allergy";
        str2 = "allergic";
        System.out.println(checkAnagrams(str1, str2)); // Output: false
    }
}
```

OUTPUT:

```
D:\javaprograms\Practice ques\14>javac Anagram.java

D:\javaprograms\Practice ques\14>java Anagram
Are the strings anagrams? true
Are the strings anagrams? false
Are the strings anagrams? true
```

TIME COMPLEXITY: O(n)

15.Longest Palindromic Substring

```
public class Solution {

    public static String longPalindrome(String str) {

        if (str == null || str.length() < 1) {
            return "";
        }

        int start = 0, end = 0;

        for (int i = 0; i < str.length(); i++) {
            // Expand around the center for odd-length and even-length palindromes
            int len1 = expandAroundCenter(str, i, i);       // Odd-length palindrome
            int len2 = expandAroundCenter(str, i, i + 1);   // Even-length palindrome


            int len = Math.max(len1, len2);
            if (len > (end - start)) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
```

```
        }

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

    private static int expandAroundCenter(String str, int left, int right) {
        // Expand while characters at left and right are equal
        while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;  // Return length of the palindrome
    }

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

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

OUTPUT:

```
D:\javaprograms\Practice ques\15>javac Palindrome.java

D:\javaprograms\Practice ques\15>java Palindromee
Error: Could not find or load main class Palindromee
Caused by: java.lang.ClassNotFoundException: Palindromee

D:\javaprograms\Practice ques\15>java Palindrome
Longest palindromic substring: geeksskeeg
Longest palindromic substring: ee
Longest palindromic substring: a
Longest palindromic substring:
Longest palindromic substring: bab
```

TIME COMPLEXITY: $O(n^2)$

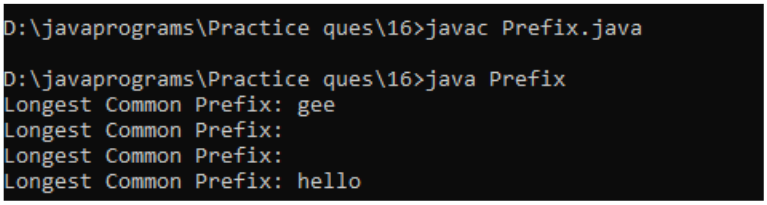
16.Longest Common Prefix using Sorting

```
public class Solution {
    public static String longestCommonPrefix(String[] arr) {
        if (arr == null || arr.length == 0) {
            return "-1";
        }

        String prefix = arr[0];
        for (int i = 1; i < arr.length; i++) {
            while (arr[i].indexOf(prefix) != 0) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) {
                    return "-1";
                }
```

```
            }
        }
        return prefix;
    }

    public static void main(String[] args) {
        String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
        String[] arr2 = {"hello", "world"};

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

import java.util.Stack;

public class DeleteMiddleElement {
    // Recursive function to delete the middle element from the stack
    public static void deleteMiddle(Stack<Integer> stack, int size, int current) {
        if (stack.isEmpty() || current == size / 2) {
            stack.pop();
            return;
        }

        int temp = stack.pop(); // Pop the top element
        deleteMiddle(stack, size, current + 1);

        stack.push(temp); // Push the element back to the stack
    }

    public static void deleteMiddleElement(Stack<Integer> stack) {
        int size = stack.size();
        if (size == 0) {
            return;
        }

        deleteMiddle(stack, size, 0);
    }

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

        deleteMiddleElement(stack1);
        System.out.println(stack1);

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

```
        stack2.push(5);
        stack2.push(6);

        deleteMiddleElement(stack2);
        System.out.println(stack2);
    }
}
```

OUTPUT:



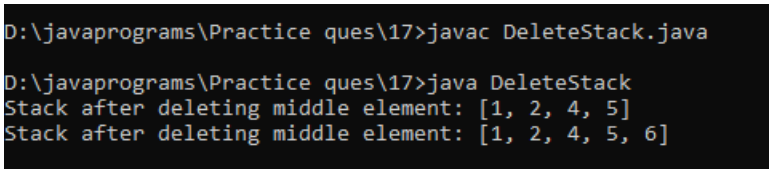
TIME COMPLEXITY: O(n * m)

17. Deletemiddle element of a stack

```
import java.util.Stack;

class Solution {
    public void deleteMiddle(Stack<Integer> s) {
        int m = s.size() / 2;
        delete(s, m);
    }

    private void delete(Stack<Integer> s, int m) {
        if (m == 0) {
            s.pop();
            return;
        }

        int top = s.pop();
        delete(s, m - 1);

        s.push(top);
    }

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

        System.out.println("Original stack: " + stack);

        Solution solution = new Solution();
        solution.deleteMiddle(stack);

        System.out.println("Stack after deleting middle element: " + stack);
```

```
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(n)

18. Next Greater Element (NGE) for every element in given Array

```
class Main {
    static void printNGE(int arr[], int n) {
        int next, i, j;
        for (i = 0; i < n; i++) {
            next = -1;
            for (j = i + 1; j < n; j++) {
                if (arr[i] < arr[j]) {
                    next = arr[j];
                    break;
                }
            }
            System.out.println(arr[i] + " -- " + next);
        }
    }

    public static void main(String args[]) {
        int arr[] = { 11, 13, 21, 3 };
        int n = arr.length;
        printNGE(arr, n);
    }
}

class Node {
    int data;
    Node left, right;

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

public class BinaryTreeRightView {
    // Function to print the right view of a binary tree
    public static void rightView(Node root) {
        if (root == null) {
            return;
        }
```

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

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

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

                // Print the last node at each level
                if (i == nodeCount) {
                    System.out.print(node.data + " ");
                }

                // Enqueue left and right children of the node
                if (node.left != null) {
                    queue.add(node.left);
                }
                if (node.right != null) {
                    queue.add(node.right);
                }
            }
        }
    }

    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.right = new Node(6);
        root.left.left.left = new Node(7);

        System.out.print("Right View: ");
        rightView(root);
    }
}
```

OUTPUT:

```
D:\javaprograms\Practice ques\18>javac Greater.java

D:\javaprograms\Practice ques\18>java Greater
Next Greater Element for arr1:
4 --> 5
5 --> 25
2 --> 25
25 --> -1

Next Greater Element for arr2:
13 --> -1
7 --> 12
6 --> 12
12 --> -1
```

TIME COMPLEXITY: $O(n^2)$

19.Print Right Viewof a Binary Tree

```
import java.util.ArrayList;

class Node {
    int data;
    Node left, right;

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

public class BinaryTree {

    static void RecursiveRightView(Node root, int level, int[] maxLevel, ArrayList<Integer> result) {
        if (root == null) return;

        if (level > maxLevel[0]) {
            result.add(root.data);
            maxLevel[0] = level;
        }

        RecursiveRightView(root.right, level + 1, maxLevel, result);
        RecursiveRightView(root.left, level + 1, maxLevel, result);
    }

    static ArrayList<Integer> rightView(Node root) {
        ArrayList<Integer> result = new ArrayList<>();
        int[] maxLevel = new int[] {-1}; // Array to keep track of the max level reached
        RecursiveRightView(root, 0, maxLevel, result);
        return result;
    }

    static void printArray(ArrayList<Integer> arr) {
        for (int val : arr) {
            System.out.print(val + " ");
        }
        System.out.println();
    }

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

        ArrayList<Integer> result = rightView(root);
        printArray(result);
    }
}
```

OUTPUT:

```
D:\javaprograms\Practice ques\19>javac Right.java

D:\javaprograms\Practice ques\19>java Right
Right side view of tree 1: [1, 3, 5, 6, 7]
```

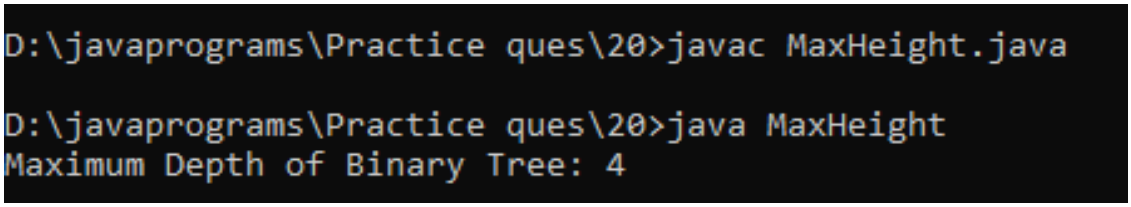
TIME COMPLEXITY: O(n)

20. Maximum Depth or Height of Binary Tree

```
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();
    }
  }
}
```

OUTPUT:



TIME COMPLEXITY: O(n)