Coding practice Problems(9/11/2024)

1. Maximum Subarray Sum – Kadane‟s Algorithm:

```
import java.util.Arrays;

class MaximumSubarraySum {

    static int maxSubarraySum(int[] arr) {

        int res = arr[0];

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

            int currSum = 0;

            for (int j = i; j < arr.length; j++) {

                currSum += arr[j];

                res = Math.max(res, currSum);

            }

        }

        return res;

    }

    public static void main(String[] args) {

        int[] arr = {2, 3, -8, 7, -1, 2, 3};

        System.out.println(maxSubarraySum(arr));

    }

}
```

Time Complexity: O(n)

Output:

```
java -cp /tmp/Rx46631FbB/MaximumSubarray
11
```

2.Maximum Product Subarray

```
import java.io.*;

class ProductSubarray {

    static int maxProduct(int[] array) {

        int maxProductResult = array[0];

        for (int start = 0; start < array.length; start++) {
```

```
            int currentProduct = 1;
            for (int end = start; end < array.length; end++) {
                currentProduct *= array[end];
                maxProductResult = Math.max(maxProductResult, currentProduct);
            }
        }
        return maxProductResult;
    }
    public static void main(String[] args) {
        int[] array = { -2, 6, -3, -10, 0, 2 };
        System.out.println(maxProduct(array));
    }
}
```

Output:

Time Complexity: O(n)

3. Search in a sorted and rotated Array

```
public class SortedArray {
    public static int findIndex(int[] array, int target) {
        for (int index = 0; index < array.length; index++) {
            if (array[index] == target) {
                return index;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
```

```
        int[] array1 = {4, 5, 6, 7, 0, 1, 2};
        int target1 = 0;
        System.out.println(findIndex(array1, target1)); // Output: 4
    }
}
```

Output:

Time Complexity: O(n)

4. Container with Most Water

```
class MaximumArea {
    static int maxArea(int[] array) {
        int leftPointer = 0;
        int rightPointer = array.length - 1;
        int maxAreaResult = 0;
        while (leftPointer < rightPointer) {
            int height = Math.min(array[leftPointer], array[rightPointer]);
            int width = rightPointer - leftPointer;
            int currentArea = height * width;
            maxAreaResult = Math.max(maxAreaResult, currentArea);
            if (array[leftPointer] < array[rightPointer]) {
                leftPointer++;
            } else {
                rightPointer--;
            }
        }
        return maxAreaResult;
```

```
    }
    public static void main(String[] args) {
        int[] array = {3, 1, 2, 4, 5};
        System.out.println(maxArea(array));
    }
}
```

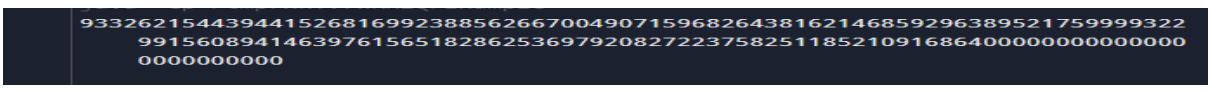
Output:

Time Complexity: O(n)

5. Find the Factorial of a large number

```
import java.math.BigInteger;
public class FactorialNumber {
    static BigInteger calculateFactorial(int number) {
        BigInteger result = BigInteger.ONE;
        for (int multiplier = 2; multiplier <= number; multiplier++) {
            result = result.multiply(BigInteger.valueOf(multiplier));
        }
        return result;
    }
    public static void main(String[] args) {
        int number = 20;
        System.out.println(calculateFactorial(number));
    }
}
```

Output:



Time Complexity: O(n)

6. Trapping Rainwater Problem

```
import java.util.*;
class Water {
	static int calculateTrappedWater(int[] heights) {
		int length = heights.length;
		int[] leftMax = new int[length];
		int[] rightMax = new int[length];

		int trappedWater = 0;
		leftMax[0] = heights[0];
		for (int i = 1; i < length; i++) {
			leftMax[i] = Math.max(leftMax[i - 1], heights[i]);
		}
		rightMax[length - 1] = heights[length - 1];
		for (int i = length - 2; i >= 0; i--) {
			rightMax[i] = Math.max(rightMax[i + 1], heights[i]);
		}
		for (int i = 1; i < length - 1; i++) {
			int waterLevel = Math.min(leftMax[i - 1], rightMax[i + 1]);
			if (waterLevel > heights[i]) {
				trappedWater += waterLevel - heights[i];
			}
		}
		return trappedWater;
	}
	public static void main(String[] args) {
		int[] heights = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
```

```
        System.out.println(calculateTrappedWater(heights));
    }
}
```

Output:

Time Complexity: O(n)

7. Chocolate Distribution Problem

```
import java.util.Arrays;
class Chocolate {
    static int findMinimumDifference(int[] packets, int students) {
        int packetCount = packets.length;
        Arrays.sort(packets);
        int minimumDifference = Integer.MAX_VALUE;
        for (int i = 0; i + students - 1 < packetCount; i++) {
            int difference = packets[i + students - 1] - packets[i];
            if (difference < minimumDifference) {
                minimumDifference = difference;
            }
        }
        return minimumDifference;
    }
    public static void main(String[] args) {
        int[] packets = {7, 3, 2, 4, 9, 12, 56};
        int students = 3;
        System.out.println(findMinimumDifference(packets,  students));
    }
}
```

Output:

Time Complexity: O(n)

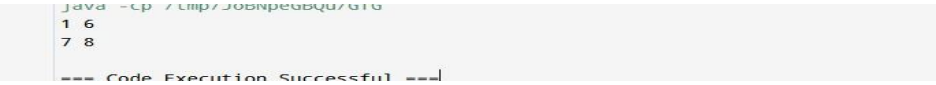
8. Merge Overlapping Intervals

```
import java.util.ArrayList;
import java.util.Arrays;
 import java.util.List;
 class MergeOverlapping {
        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<>();

                for (int i = 0; i < n; i++) {
                        int start = arr[i][0];
                        int end = arr[i][1];
                        if (!res.isEmpty() && res.get(res.size() - 1)[1] >= end) {
                                continue;
                        }
                        for (int j = i + 1; j < n; j++) {
                                if (arr[j][0] <= end) {
                                        end = Math.max(end, arr[j][1]);
                                }
                        }
                        res.add(new int[]{start, end});
                }
```

```
        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:



Time Complexity: O(n^2)

9. A Boolean Matrix Question

```
import java.util.*;
class Main {
    static void updateMatrix(int[][] grid)
    {
        int rowCount = grid.length;
        int colCount = grid[0].length;

        for (int x = 0; x < rowCount; x++) {
            for (int y = 0; y < colCount; y++) {

                if (grid[x][y] == 1) {
```

```
int index = x - 1;
while (index >= 0) {
	if (grid[index][y] != 1) {
		grid[index][y] = -1;
	}
	index--;
}
index = x + 1;
while (index < rowCount) {
	if (grid[index][y] != 1) {
		grid[index][y] = -1;
	}
	index++;
}
index = y - 1;
while (index >= 0) {
	if (grid[x][index] != 1) {
		grid[x][index] = -1;
	}
	index--;
}
index = y + 1;
while (index < colCount) {
	if (grid[x][index] != 1) {
		grid[x][index] = -1;
	}
	index++;
}
```

```
                }
            }
        }
        for (int x = 0; x < rowCount; x++) {
            for (int y = 0; y < colCount; y++) {
                if (grid[x][y] < 0) {
                    grid[x][y] = 1;
                }
            }
        }
    }
    public static void main(String[] args)
    {
        int[][] matrix = { { 1, 0, 0, 1 },
                           { 0, 0, 1, 0 },
                           { 0, 0, 0, 0 } };
        updateMatrix(matrix);
        System.out.println("The Final Matrix is:");
        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();
        }
    }
}
```

Output:

```
The Final Matrix is:
1 1 1 1
1 1 1 1
1 0 1 1
```

Time Complexity: O(n*m)

10. Print a given matrix in spiral form

```
import java.util.*;

public class SpiralTraversal {

    public static List<Integer> getSpiralOrder(int[][] grid) {

        int rowCount = grid.length;

        int colCount = grid[0].length;

        List<Integer> output = new ArrayList<>();

        if (rowCount == 0)

            return output;

        boolean[][] visited = new boolean[rowCount][colCount];

        int[] rowDir = {0, 1, 0, -1};

        int[] colDir = {1, 0, -1, 0};

        int row = 0, col = 0;

        int direction = 0;

        for (int i = 0; i < rowCount * colCount; ++i) {

            output.add(grid[row][col]);

            visited[row][col] = true;

            int nextRow = row + rowDir[direction];

            int nextCol = col + colDir[direction];

            if (0 <= nextRow && nextRow < rowCount && 0 <= nextCol && nextCol < colCount &&
!visited[nextRow][nextCol]) {

                row = nextRow;

                col = nextCol;
```

```
            } else {
                direction = (direction + 1) % 4;
                row += rowDir[direction];
                col += colDir[direction];
            }
        }
        return output;
    }
    public static void main(String[] args) {
        int[][] grid = {
                { 1, 2, 3, 4 },
                { 5, 6, 7, 8 },
                { 9, 10, 11, 12 },
                { 13, 14, 15, 16 }
        };
        List<Integer> result = getSpiralOrder(grid);
        for (int value : result) {
            System.out.print(value + " ");
        }
    }
}
```

Output:

```
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

Time Complexity: O(n*m)

11. Check if given Parentheses expression is balanced or not

```
import java.util.Stack;
```

```
public class BracketChecker {
	public static boolean checkBalance(String input) {
		Stack<Character> stack = new Stack<>();
		for (int i = 0; i < input.length(); i++) {
			if (input.charAt(i) == '(' || input.charAt(i) == '{' || input.charAt(i) == '[') {
				stack.push(input.charAt(i));
			}
			else {
				if (!stack.empty() &&
					((stack.peek() == '(' && input.charAt(i) == ')') ||
					 (stack.peek() == '{' && input.charAt(i) == '}') ||
					 (stack.peek() == '[' && input.charAt(i) == ']'))) {
					stack.pop();
				}
				else {
					return false;
				}
			}
		}
		return stack.empty();
	}
	public static void main(String[] args) {
		String expression = "{()}[]";
		if (checkBalance(expression))
			System.out.println("true");
		else
			System.out.println("false");
	}
```

}

Output:

```
java -cp /tmp/EeAp2wPgyX/Anagrams
true
```

Time Complexity: O(n)

12. Check if two Strings are Anagrams of each other

```
import java.util.Arrays;

class Anagrams {

    static boolean checkAnagram(String str1, String str2) {

        char[] arr1 = str1.toCharArray();

        char[] arr2 = str2.toCharArray();

        Arrays.sort(arr1);

        Arrays.sort(arr2);

        return Arrays.equals(arr1, arr2);

    }

    public static void main(String[] args) {

        String str1 = "geeks";

        String str2 = "kseeg";

        System.out.println(checkAnagram(str1, str2));

    }

}
```

Output:

```
java -cp /tmp/EeAp2wPgyX/Anagrams
true
```

Time Complexity: O(n)

13. Longest Palindromic Substring

```
public class Palindrome {
    static boolean isPalindrome(String str, int begin, int end) {
        while (begin < end) {
            if (str.charAt(begin) != str.charAt(end))
                return false;
            begin++;
            end--;
        }
        return true;
    }
    static String findLongestPalindrome(String str) {
        int length = str.length();
        int maxLength = 1, startIndex = 0;
        for (int i = 0; i < length; i++) {
            for (int j = i; j < length; j++) {
                if (isPalindrome(str, i, j) && (j - i + 1) > maxLength) {
                    startIndex = i;
                    maxLength = j - i + 1;
                }
            }
        }
        return str.substring(startIndex, startIndex + maxLength);
    }
    public static void main(String[] args) {
        String input = "Geeks";
        System.out.println(findLongestPalindrome(input));
    }
```

}

Output:

```
ee
```

Time Complexity: O(n^2)

14. Longest Common Prefix using Sorting

```
import java.util.Arrays;
class PrefixFinder {
	static String findLongestPrefix(String[] words) {
		if (words == null || words.length == 0)
			return "-1";
		Arrays.sort(words);
		String firstWord = words[0];
		String lastWord = words[words.length - 1];
		int minLength = Math.min(firstWord.length(), lastWord.length());
		int index = 0;
		while (index < minLength && firstWord.charAt(index) == lastWord.charAt(index)) {
			index++;
		}
		if (index == 0)
			return "-1";
		return firstWord.substring(0, index);
	}
	public static void main(String[] args) {
		String[] words = { "geeksforgeeks", "geeks", "geek", "geezer" };
		System.out.println("The longest common prefix is: " + findLongestPrefix(words));
	}
```

}

Output:

```
The longest common prefix is: gee
```

Time Complexity: O(n*m*log n)

15. Delete middle element of a stack

```
import java.util.Stack;
import java.util.Vector;
public class Main {
	public static void main(String[] args) {
		Stack<Character> st = new Stack<Character>();
		st.push('1');
		st.push('2');
		st.push('3');
		st.push('4');
		st.push('5');
		st.push('6');
		st.push('7');
		Vector<Character> v = new Vector<Character>();
		while (!st.empty()) {
			v.add(st.pop());
		}
		int n = v.size();
		if (n % 2 == 0) {
			int target = (n / 2);
			for (int i = 0; i < n; i++) {
				if (i == target) continue;
```

```
                    st.push(v.get(i));
                }
            } else {
                int target = (int) Math.ceil(n / 2);
                for (int i = 0; i < n; i++) {
                    if (i == target) continue;
                    st.push(v.get(i));
                }
            }
            System.out.print("Printing stack after deletion of middle: ");
            while (!st.empty()) {
                char p = st.pop();
                System.out.print(p + " ");
            }
        }
}
```

Output:

```
Printing stack after deletion of middle: 1 2 3 5 6 7
```

Time Complexity: O(n)

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

```
class GreaterElementFinder {
    static void findNextGreater(int numbers[], int size) {
        int nextGreater, i, j;
        for (i = 0; i < size; i++) {
            nextGreater = -1;
            for (j = i + 1; j < size; j++) {
```

```
                    if (numbers[i] < numbers[j]) {
                        nextGreater = numbers[j];
                        break;
                    }
                }
                System.out.println(numbers[i] + " -- " + nextGreater);
            }
        }
        public static void main(String args[]) {
            int numbers[] = { 11, 13, 21, 3 };
            int size = numbers.length;
            findNextGreater(numbers, size);
        }
}
```

Output:

```
11 -- 13
13 -- 21
21 -- -1
3 -- -1
```

Time Complexity: O(n)

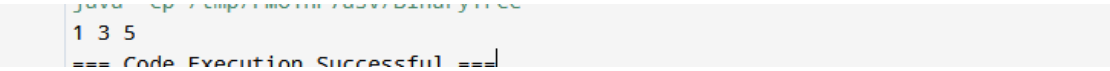
17. Print Right View of a Binary Tree

```
import java.util.*;
 class TreeNode {
        int val;
        TreeNode left, right;
        TreeNode() {
            val = 0;
            left = right = null;
```

```
    }
    TreeNode(int x) {
        val = x;
        left = right = null;
    }
    TreeNode(int x, TreeNode left, TreeNode right) {
        val = x;
        this.left = left;
        this.right = right;
    }
}
public class BinaryTree {
    public static List<Integer> rightSideView(TreeNode root) {
        List<Integer> ans = new ArrayList<>();
        if (root == null) return ans;
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        while (!q.isEmpty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = q.poll();
                if (i == size - 1) {
                    ans.add(node.val);
                }
                if (node.left != null) q.add(node.left);
                if (node.right != null) q.add(node.right);
            }
        }
```

```
        return ans;
    }
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.right = new TreeNode(4);
        root.right.right = new TreeNode(5);
        List<Integer> result = rightSideView(root);
        for (int val : result) {
            System.out.print(val + " ");
        }
    }
}
```

Output:



Time Complexity: O(n)

18. Maximum Depth or Height of Binary Tree

```
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode() {
        val = 0;
        left = right = null;
    }
    TreeNode(int x) {
        val = x;
```

```
        left = right = null;
    }
    TreeNode(int x, TreeNode left, TreeNode right) {
        val = x;
        this.left = left;
        this.right = right;
    }
}
public class BinaryTree {
    public static int maxDepth(TreeNode root) {
        if (root == null) return 0;
        int lh = maxDepth(root.left);
        int rh = maxDepth(root.right);
        return 1 + Math.max(lh, rh);
    }
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        System.out.println("Maximum Depth of Binary Tree: " + maxDepth(root));
    }}
```

Output:

```
Maximum Depth of Binary Tree: 3
```

Time Complexity: O(n)