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

   Code:
   ```
   import java.util.*;
   import java.util.Arrays;
   public class Kadane {
       static int maxSubarraySum(int[] a) {
           int res=a[0];
           int maxend=a[0];
           for(int i=1;i<a.length;i++){
               maxend=Math.max(maxend+a[i],a[i]);
               res=Math.max(res,maxend);
           }
       return res;
       }
       public static void main(String[] args){
           Kadane ob=new Kadane();
           int[] a={2, 3, -8, 7, -1, 2, 3};
           int r=ob.maxSubarraySum(a);
           System.out.println(r);
       }
   }
   ```
   Output:11

Time complexity: O(n)

2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.

   Code:
   ```
   class MaxProd {
       static int max(int a, int b, int c) {
           return Math.max(a, Math.max(b, c));
       }

       static int min(int a, int b, int c) {
           return Math.min(a, Math.min(b, c));
       }

       static int maxProduct(int[] arr) {
           int n = arr.length;
   ```

```
        int currMax = arr[0];
        int currMin = arr[0];
        int maxProd = arr[0];

        for (int i = 1; i < n; i++) {
            int temp = max(arr[i], arr[i] * currMax, arr[i] * currMin);
            currMin = min(arr[i], arr[i] * currMax, arr[i] * currMin);
            currMax = temp;
            maxProd = Math.max(maxProd, currMax);
        }

        return maxProd;
    }

    public static void main(String[] args) {
        int[] arr = { -2, 6, -3, -10, 0, 2 };
        System.out.println(maxProduct(arr));
    }
}
```

```
Output:180
```

Time complexity: O(n)

3. Search in a sorted and rotated Array Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.

   Code:

```
import java.util.*;
public class IndexK {
    static int keyFind(int[] a, int key){
        for(int i=0;i<a.length;i++){
            if(a[i]==key){
                return i;
            }
        }
        return -1;
    }
        public static void main(String[] args){
            int[] a1= {4, 5, 6, 7, 0, 1, 2};
            int key1=0;
            System.out.println(keyFind(a1,key1));
        }

}
```

Output:4

Time complexity: O(log n)

4. Container with Most Water

   Code:

```
. public class ContainerWater {
    public int maxArea(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int maxA = 0;
        while (left < right) {
            int currA = Math.min(height[left], height[right]) * (right - left);
            maxA = Math.max(maxA, currA);
            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }
        return maxA;
    }

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

   Output:12

   Time complexity: O(n)

5. Find the Factorial of a large number

   Code:

```
. import java.util.*;
import java.math.BigInteger;
public class FactLarge {

    static BigInteger factorial(int N) {
        BigInteger f = new BigInteger("1");
        for (int i = 2; i <= N; i++)
            f = f.multiply(BigInteger.valueOf(i));
        return f;
    }

    public static void main(String args[]) throws Exception {
        int N = 50;
        System.out.println(factorial(N));
    }
}
```

   Output:

```
30414093201713378043612608166064768844377641568960512000000000000
```

   Time 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.

Code:

```
class WaterTrap {
    static int trap(int[] height) {
        int left = 0, right = height.length - 1;
        int left_max = 0, right_max = 0;
        int water = 0;
        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= left_max) {
                    left_max = height[left];
                } else {
                    water += left_max - height[left];
                }
                left++;
            } else {
                if (height[right] >= right_max) {
                    right_max = height[right];
                } else {
                    water += right_max - height[right];
                }
                right--;
            }
        }
        return water;
    }
    public static void main(String[] args){
        int[] height={ 3, 0, 1, 0, 4, 0, 2};
        System.out.println(trap(height));
    }
}
```

Output:10

Time complexity: O(n)

7. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable

number of chocolates. There are m students, the task is to distribute chocolate packets such that:

Code:

```
import java.util.*;
public class Chocolate {
    static int dChocy(int[] arr,int m){
        Arrays.sort(arr);
        int d=0;
        int min=Integer.MAX_VALUE;
        for(int i=0;i<arr.length-m;i++){
            d=arr[i+m-1]-arr[i];
            min=Math.min(min,d);
        }
        return min;
    }
    public static void main(String[] args){
        int [] arr={7, 3, 2, 4, 9, 12, 56};
        int m=3;
        System.out.println(dChocy(arr,m));
    }
}
```

Output:2
Time complexity: O(nlogn)

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
   Code:
```
import java.util.*;

class MergeIntervals {
    static int[][] merge(int[][] intervals) {
        if (intervals.length <= 1) return intervals;

        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> merged = new ArrayList<>();

        int[] currentInterval = intervals[0];
        merged.add(currentInterval);

        for (int[] interval : intervals) {
            int currentEnd = currentInterval[1];
            if (interval[0] <= currentEnd) {
                currentInterval[1] = Math.max(currentEnd, interval[1]);
            } else {
                currentInterval = interval;
                merged.add(currentInterval);
            }
        }

        return merged.toArray(new int[merged.size()][]);
    }

    public static void main(String[] args) {
        int[][] intervals = {{1,3}, {2,6}, {8,10}, {15,18}};
        int[][] result = merge(intervals);
        System.out.println("Merged intervals: " + Arrays.deepToString(result));
    }
}
```

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

Time complexity: O(nlogn)

9. A Boolean Matrix Question Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

   Code:

```
import java.util.Scanner;

class Bool {
    public void modifyMatrix(int[][] mat, int M, int N) {
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (mat[i][j] == 1) {
                    for (int k = 0; k < N; k++) {
                        if (mat[i][k] == 0) mat[i][k] = -1;  // Mark columns
                    }
                    for (int k = 0; k < M; k++) {
                        if (mat[k][j] == 0) mat[k][j] = -1;  // Mark rows
                    }
                }
            }
        }

        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (mat[i][j] == -1) {
                    mat[i][j] = 1;
                }
            }
        }
    }

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

        System.out.println("Enter the number of rows (M):");
        int M = scanner.nextInt();
        System.out.println("Enter the number of columns (N):");
        int N = scanner.nextInt();

        int[][] mat = new int[M][N];

        System.out.println("Enter the elements of the matrix (0 or 1) row by row:");
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                mat[i][j] = scanner.nextInt();
            }
        }

        Bool solution = new Bool();
        solution.modifyMatrix(mat, M, N);

        System.out.println("Modified matrix:");
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }

        scanner.close();
    }
}
```
Output:
```
Enter the number of rows (M):
3
Enter the number of columns (N):
3
Enter the elements of the matrix (0 or 1) row by row:
1 0 0
0 0 0
0 0 1
```

Time complexity: O(M×N)

10. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form

    Code:

```
public class SpiralMat {
    static void spir(int[][] arr){
        int m=arr.length;
        int n=arr[0].length;
        int top=0, bot=m-1, l=0, r=n-1;
        while(top<=bot&&l<=r){
            for(int i=l;i<=r;i++){
                System.out.print(arr[top][i] + " ");
            }
            top++;
            for(int i=top;i<=bot;i++){
                System.out.print(arr[i][r] + " ");
            }
            r--;
            if(top <= bot){
                for(int i=r;i>=l;i--){
                    System.out.print(arr[bot][i] + " ");
                }
                bot--;
            }
            if(l<=r){
                for(int i=bot;i>=top;i--){
                    System.out.print(arr[i][l] + " ");
                }
                l++;
            }

        }
    }
    public static void main(String ar[]){
        int[][] arr={
            {1,2,3,4},
            {5,6,7,8},
            {9,10,11,12},
            {13,14,15,16}
        };
        spir(arr);
    }
}
```

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

Time complexity: O(n)

11.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

Code:

```
public class Parentheses {
   static void balanced(String s) {
      int counter = 0;
      for (int i = 0; i < s.length(); i++) {
         if (s.charAt(i) == '(') {
            counter++;
         } else if (s.charAt(i) == ')') {
            if (counter == 0) {
               System.out.println("Not Balanced");
               return;
            }
            counter--;
         }
      }

      System.out.println(counter == 0 ? "Balanced" : "Not Balanced");
   }

   public static void main(String[] args) {
      String s = "((()))()()";
      balanced(s);
   }
}
```

Output:Balanced

Time complexity: O(n)

12.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.

Code:

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

public class Anagram {
   public static boolean isAnagram(String s1, String s2) {
      if (s1.length() != s2.length()) {
         return false;
```

```
        }
        HashMap<Character, Integer> charCountMap = new HashMap<>();
        for (char c : s1.toCharArray()) {
            charCountMap.put(c, charCountMap.getOrDefault(c, 0) + 1);
        }
        for (char c : s2.toCharArray()) {
            if (!charCountMap.containsKey(c) || charCountMap.get(c) == 0) {
                return false;
            }
            charCountMap.put(c, charCountMap.get(c) - 1);
        }
        return true;
    }

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

        System.out.print("Enter first string: ");
        String s1 = scanner.nextLine();

        System.out.print("Enter second string: ");
        String s2 = scanner.nextLine();

        System.out.println(isAnagram(s1, s2));

        scanner.close();
    }
}
```

Output:
```
geeks
kseeg
true
```

Time complexity: O(nlogn)

13. 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.
    Code:

```
public class LongestPalindrome {
    static boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }

        return true;
    }

    static String longestPalindrome(String s) {
        if (s.length() <= 1) {
            return s;
        }

        int maxLen = 1;
        String maxStr = s.substring(0, 1);

        for (int i = 0; i < s.length(); i++) {
            for (int j = i + maxLen; j <= s.length(); j++) {
                if (j - i > maxLen && isPalindrome(s.substring(i, j))) {
                    maxLen = j - i;
                    maxStr = s.substring(i, j);
                }
            }
        }

        return maxStr;
    }
    public static void main(String ar[]){
        String str="";
        System.out.print(longestPalindrome(str));
    }
}
```

Input:"Geeks"
Output:"ee"

Time complexity: O(n^2)

14. 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".
Code:

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

class Bool {
    static String longestCommonPrefix(String[] arr) {
        if (arr == null || arr.length == 0)
```

```
            return "-1";
        Arrays.sort(arr);
        String first = arr[0];
        String last = arr[arr.length - 1];
        int minLength = Math.min(first.length(), last.length());

        int i = 0;
        while (i < minLength && first.charAt(i) == last.charAt(i)) {
            i++;
        }
        if (i == 0)
            return "-1";
        return first.substring(0, i);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of strings: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String[] arr = new String[n];
        for (int i = 0; i < n; i++) {
            System.out.print("Enter string " + (i + 1) + ": ");
            arr[i] = scanner.nextLine();
        }
        System.out.println("The longest common prefix is: " +
longestCommonPrefix(arr));
    }
}
```
Input: arr[] = [“hello”, “world”]
Output: -1
Time complexity: O(nlogn+m)

15. 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.
Code:
```
import java.util.Scanner;
import java.util.Stack;
import java.util.Vector;

public class Bool {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Stack<Character> st = new Stack<>();
        System.out.print("Enter characters (no spaces): ");
        String input = scanner.nextLine();
        for (char ch : input.toCharArray()) {
            st.push(ch);
        }
        Vector<Character> v = new Vector<>();
        while (!st.empty()) {
            v.add(st.pop());
        }

        int n = v.size();
        int target = n / 2;  // Middle index
        for (int i = 0; i < n; i++) {
            if (i == target) continue; // Skip the middle element
            st.push(v.get(i));
        }
        System.out.print("Printing stack after deletion of middle: ");
        while (!st.empty()) {
            System.out.print(st.pop() + " ");
        }
    }
}
```

Input : Stack[] = [1, 2, 3, 4, 5]
Output : Stack[] = [1, 2, 4, 5]
Time complexity: O(n)

16. 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
Code:

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

public class NextGreaterElement {

    public static void printNextGreaterElement(int[] arr) {
        Stack<Integer> stack = new Stack<>();
        HashMap<Integer, Integer> ngeMap = new HashMap<>();

        for (int i = arr.length - 1; i >= 0; i--) {
            int current = arr[i];
            while (!stack.isEmpty() && stack.peek() <= current) {
                stack.pop();
            }
            if (stack.isEmpty()) {
                ngeMap.put(current, -1);
            } else {
                ngeMap.put(current, stack.peek());
            }
            stack.push(current);
        }
        for (int num : arr) {
            System.out.println(num + " -> " + ngeMap.get(num));
        }
    }

    public static void main(String[] args) {
        int[] arr1 = {4, 5, 2, 25};
        System.out.println("Output for arr1:");
        printNextGreaterElement(arr1);

    }
}
```

```
Output: 4 –> 5
 5 –> 25
 2 –> 25
 25 –> -1
```

Time complexity: O(n)

17. 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:

```
class TreeNode {
    int val;
    TreeNode left, right;

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

public class RightViewBinaryTree {
    public static void rightViewDFS(TreeNode node, int level, int[] maxLevel) {
        if (node == null) {
            return;
        }
        if (level > maxLevel[0]) {
            System.out.print(node.val + " ");
            maxLevel[0] = level;
        }
        rightViewDFS(node.right, level + 1, maxLevel);
        rightViewDFS(node.left, level + 1, maxLevel);
    }
    public static void printRightView(TreeNode root) {
        int[] maxLevel = new int[1];
        maxLevel[0] = -1;
        rightViewDFS(root, 0, maxLevel);
    }

    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.right.right = new TreeNode(5);


        printRightView(root);
    }
}
```
Output:1 3 5

Time complexity: O(n)

18. Maximum Depth or Height of Binary Tree Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node
Code:
```
import java.util.LinkedList;
import java.util.Queue;

class TreeNode {
    int val;
    TreeNode left, right;

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

public class BinaryTreeHeightIterative {

    public static int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        int height = 0;

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            height++;
            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll();
                if (currentNode.left != null) {
                    queue.add(currentNode.left);
                }
                if (currentNode.right != null) {
                    queue.add(currentNode.right);
                }
            }
        }

        return height;
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(12);
        root.left = new TreeNode(8);
        root.right = new TreeNode(18);
        root.left.left = new TreeNode(5);
        root.left.right = new TreeNode(11);

        System.out.println( maxDepth(root));
    }
}
```
Output:3

Time Complexity:O(n)