

CODING PRACTICE PROBLEMS

(Sowmya A)

DATE: 09/11/24

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

```
import java.util.*;

class maxsumsub{

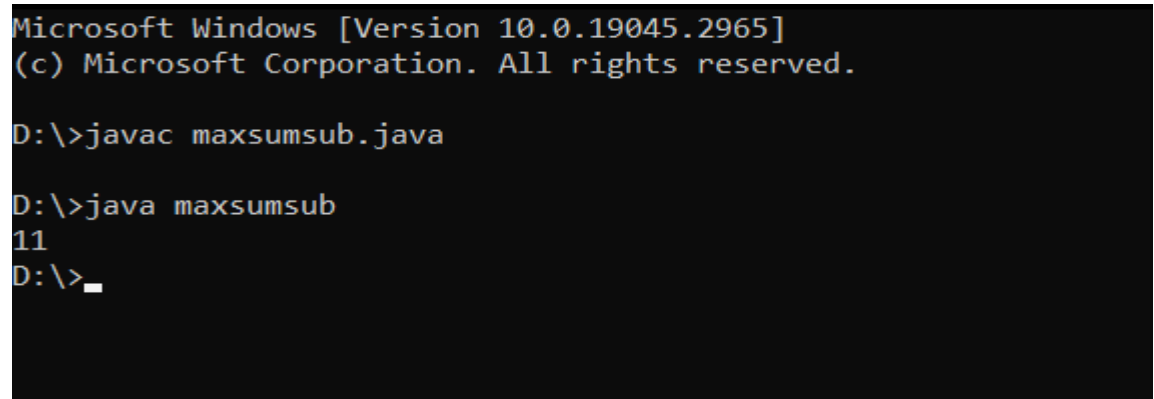
    static int maxsubarray(int[] nums){
        int maxsum=nums[0];
        int currentsum=nums[0];

        for(int i=1;i<nums.length;i++){
            currentsum=Math.max(nums[i],currentsum+nums[i]);

            maxsum=Math.max(maxsum,currentsum);
        }
        return maxsum;
    }

    public static void main(String args[]){
        int[] nums={2, 3, -8, 7, -1, 2, 3};
        System.out.print(maxsubarray(nums));
    }
}
```

O/P:



```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

D:\>javac maxsumsub.java

D:\>java maxsumsub
11
D:\>_
```

Time Complexity: O(n)

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

```
import java.util.*;

class maxpro{
    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[] nums){
        int currmax=nums[0];
        int currmin=nums[0];
        int maxprod=nums[0];

        for(int i=1;i<nums.length;i++){

            int temp=max(nums[i],nums[i]*currmax,nums[i]*currmin);

            currmin=min(nums[i],nums[i]*currmin,nums[i]*currmax);

            currmax=temp;

            maxprod=Math.max(maxprod,currmax);
        }
        return maxprod;
    }
    public static void main(String args[]){
        int[] nums={-2, 6, -3, -10, 0, 2};
        System.out.print(maxproduct(nums));
    }
}
```

O/P:

```
D:\>javac maxpro1.java

D:\>java maxpro
180
D:\>
```

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.

Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0

Output : 4

```
import java.util.*;

class Rotatedsearch{
    static int searchkey(int[] nums,int key){

        int low=0;
        int high=nums.length-1;

        while(low<=high){

            int mid=(low+high)/2;

            if(nums[mid]==key){
                return mid;
            }

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

    public static void main(String args[]){
        int[] nums={4, 5, 6, 7, 0, 1, 2};
        int key=0;
        int result=searchkey(nums,key);
        System.out.print(result);
    }
}
```

O/P:

```
D:\>javac searchsort.java
```

```
D:\>java Rotatedsearch
```

```
4
```

```
D:\>
```

Time Complexity: $O(\log(n))$

4. Container with Most Water

Input: arr = [1, 5, 4, 3]

Output: 6

```
import java.util.*;

class containerwater {
    static int maxArea(int[] arr) {
        int n = arr.length;
        int area = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {

                // Calculating the max area
                area = Math.max(area, Math.min(arr[j], arr[i]) * (j - i));
            }
        }
        return area;
    }

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



```
Java ▾ 🔒 Auto
1 class Solution {
2     public int maxArea(int[] height) {
3         int area=0;
4         int left=0;
5         int right=height.length-1;
6
7         while(left<right){
8             area=Math.max(area,Math.min(height[left],height[right])*(right-left));
9
10            if(height[left]<height[right]){
11                left++;
12            }else{
13                right--;
14            }
15        }
16        return area;
17    }
18 }
```

```
D:\>javac containerwater.java
```

```
D:\>java containerwater  
6
```

Time Complexity: $O(n^2)$

5. Find the Factorial of a large number

Input: 100

Output:

93326215443944152681699238856266700490715968264381621468592963895217599993229915
6089414639761565182862536979208272237582511852109168640000000000000000000000

```
class factorial {  
    static void factorial(int n)  
    {  
        int res[] = new int[500];  
        res[0] = 1;  
        int res_size = 1;  
        for (int x = 2; x <= n; x++)  
            res_size = multiply(x, res, res_size);  
  
        for (int i = res_size - 1; i >= 0; i--)  
            System.out.print(res[i]);  
    }  
  
    static int multiply(int x, int res[], int res_size)  
    {  
        int carry = 0;  
  
        for (int i = 0; i < res_size; i++) {  
            int prod = res[i] * x + carry;  
            res[i] = prod % 10;  
  
            carry = prod / 10;  
        }  
  
        while (carry != 0) {  
            res[res_size] = carry % 10;  
            carry = carry / 10;  
            res_size++;  
        }  
        return res_size;  
    }  
  
    public static void main(String args[])  
    {  
        factorial(100);  
    }  
}
```

```
D:\>javac factorial.java  
  
D:\>java factorial  
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000  
D:\>
```

Time Complexity: $O(N \log (N!))$

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

```
import java.util.*;
class trappingwater {

    static int maxWater(int[] arr) {
        int res = 0;

        for (int i = 1; i < arr.length - 1; i++) {

            int left = arr[i];
            for (int j = 0; j < i; j++)
                left = Math.max(left, arr[j]);

            int right = arr[i];
            for (int j = i + 1; j < arr.length; j++)
                right = Math.max(right, arr[j]);

            res += Math.min(left, right) - arr[i];
        }

        return res;
    }

    public static void main(String[] args) {
        int[] arr = {3, 0, 1, 0, 4, 0, 2};
        System.out.println(maxWater(arr));
    }
}
```

```
D:\>javac trappingwater.java
```

```
D:\>java trappingwater
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 `i`th packet.

Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

```

import java.util.Arrays;

class chocolate {

    static int findMinDiff(int[] arr, int m) {
        int n = arr.length;

        Arrays.sort(arr);

        int minDiff = Integer.MAX_VALUE;

        for (int i = 0; i + m - 1 < n; i++) {

            int diff = arr[i + m - 1] - arr[i];

            if (diff < minDiff)
                minDiff = diff;
        }
        return minDiff;
    }

    public static void main(String[] args) {
        int[] arr = {7, 3, 2, 4, 9, 12, 56};
        int m = 3;

        System.out.println(findMinDiff(arr, m));
    }
}

```

```

D:\>javac chocolatedistribution.java

D:\>java chocolate
2

```

Time Complexity: $n \cdot \log(n)$

8. Merge Overlapping Intervals

Given an array of time intervals where $\text{arr}[i] = [\text{start}_i, \text{end}_i]$, the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

Input: $\text{arr} = [[1, 3], [2, 4], [6, 8], [9, 10]]$

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

```

import java.util.*;

class mergeoverlap {

    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 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        List<int[]> res = mergeOverlap(arr);

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

```

```
D:\>javac mergeoverlap.java
```

```
D:\>java mergeoverlap
```

```
1 4
```

```
6 8
```

```
9 10
```

Time Complexity: $O(n \cdot \log(n))$

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 `i`th row and `j`th column as 1.

Input: `{{1, 0},`

`{0, 0}}`

Output: `{{1, 1}`

`{1, 0}}`

```
import java.util.*;

class booleanmatrix {

    static void setZeroes(int[][] matrix)
    {

        int rows = matrix.length;
        int cols = matrix[0].length;

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {

                if (matrix[i][j] == 1) {

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

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

```

        ind++;
    }
}

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

}

public static void main(String[] args)
{
    int[][] arr = { { 1, 0 },
                    { 0, 0 } };
    setZeroes(arr);
    System.out.println("The Final Matrix is:");
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr[0].length; j++) {
            System.out.print(arr[i][j] + " ");
        }
        System.out.println();
    }
}
}

```

```

D:\>javac booleanmatrix.java

D:\>java booleanmatrix
The Final Matrix is:
1 1
1 0

```

Time Complexity: $O((N*M)*(N + M))$

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

```

import java.util.*;

class SpiralOrderMatrix {

    public static List<Integer> spiralOrder(int[][] matrix) {

        int m = matrix.length;

        int n = matrix[0].length;

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

        if (m == 0)
            return result;

        boolean[][] seen = new boolean[m][n];

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

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

        int r = 0, c = 0;

        int di = 0;

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

            result.add(matrix[r][c]);

            seen[r][c] = true;

            int newR = r + dr[di];
            int newC = c + dc[di];

            if (0 <= newR && newR < m && 0 <= newC && newC < n
                && !seen[newR][newC]) {

                r = newR;

                c = newC;

            } else {
                di = (di + 1) % 4;
            }
        }
    }
}

```

```

        r += dr[di];
        c += dc[di];
    }
}

return result;
}

// Main function
public static void main(String[] args) {

    // Example matrix initialization
    int[][] matrix = {
        { 1, 2, 3, 4 },
        { 5, 6, 7, 8 },
        { 9, 10, 11, 12 },
        { 13, 14, 15, 16 }
    };

    List<Integer> result = spiralOrder(matrix);

    for (int num : result) {
        System.out.print(num + " ");
    }
}
}

```

```

D:\>javac spiralmatrix.java

D:\>java SpiralOrderMatrix
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
D:\>

```

Time Complexity: $O(m*n)$

13. Check if given Parentheses expression is balanced or not

Given a string str of length N, consisting of „(„, „)„, only, the task is to check whether it is balanced or not.

Input: str = "((()))()"

Output: Balanced

```
class parentheses{

public static boolean isBalanced(String exp)
{
    boolean flag = true;
    int count = 0;

    for(int i = 0; i < exp.length(); i++)
    {
        if (exp.charAt(i) == '(')
        {
            count++;
        }
        else
        {
            // It is a closing parenthesis
            count--;

            if (count < 0)
            {
                flag = false;
                break;
            }
        }
    }

    if (count != 0)
    {
        flag = false;
    }
    return flag;
}

public static void main(String[] args)
{
    String exp1 = "((()))()";

    if (isBalanced(exp1))
        System.out.println("Balanced");
    else
        System.out.println("Not Balanced");
}
}
```

```
D:\>javac parentheses.java
```

```
D:\>java parentheses
Balanced
```

Time Complexity: $O(n)$

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

```
import java.util.HashMap;

class anagrams {

    static boolean areAnagrams(String s1, String s2) {

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

        for (char ch : s1.toCharArray())
            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);

        for (char ch : s2.toCharArray())
            charCount.put(ch, charCount.getOrDefault(ch, 0) - 1);

        for (var pair : charCount.entrySet()) {
            if (pair.getValue() != 0) {
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        String s1 = "geeks";
        String s2 = "kseeg";
        System.out.println(areAnagrams(s1, s2) ? "true" : "false");
    }
}
```

```
D:\>javac anagrams.java
```

```
D:\>java anagrams
true
```

Time Complexity: $O(m \cdot \log(m) + n \cdot \log(n))$

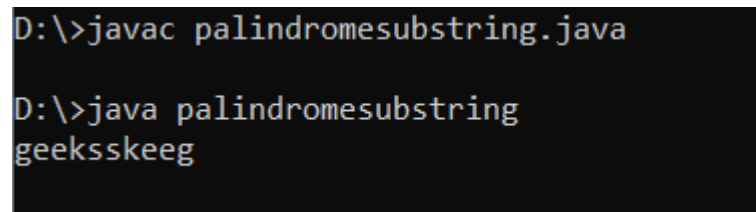
15. Longest Palindromic Substring

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"

```
class palindromesubstring {  
    static boolean checkPal(String s, int low, int high) {  
        while (low < high) {  
            if (s.charAt(low) != s.charAt(high))  
                return false;  
            low++;  
            high--;  
        }  
        return true;  
    }  
  
    static String longestPalSubstr(String s) {  
        int n = s.length();  
  
        int maxLen = 1, start = 0;  
  
        for (int i = 0; i < n; i++) {  
            for (int j = i; j < n; j++) {  
                if (checkPal(s, i, j) && (j - i + 1) > maxLen) {  
                    start = i;  
                    maxLen = j - i + 1;  
                }  
            }  
        }  
  
        return s.substring(start, start + maxLen);  
    }  
  
    public static void main(String[] args) {  
        String s = "forgeeksskeegfor";  
        System.out.println(longestPalSubstr(s));  
    }  
}
```



```
D:\>javac palindromesubstring.java  
  
D:\>java palindromesubstring  
geeksskeeg
```

Time complexity: $O(N^3)$

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

```

import java.util.Arrays;

class commonprefix {
    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){
        String[] arr = { "geeksforgeeks", "geeks", "geek",
            "geezer" };
        System.out.println(longestCommonPrefix(arr));
    }
}

```

```
D:\>javac commonprefix.java
```

```
D:\>java commonprefix
gee
```

Time Complexity: $O(n \log n + m)$

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]


```

import java.util.Stack;
import java.util.Vector;

class deletemain {

    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));
            }
        }
        while (!st.empty()) {
            char p = st.pop();
            System.out.print(p + " ");
        }
    }
}

```

```
D:\>javac deletemiddle.java
```

```
D:\>java deletemain
```

```
1 2 3 5 6 7
```

Time 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

5 -> 25

2 -> 25

25 -> -1

```
class nextgreat {  
  
    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[] = { 4, 5, 2, 25 };  
        int n = arr.length;  
        printNGE(arr, n);  
    }  
}
```

```
D:\>javac nextgreat.java
```

```
D:\>java nextgreat
```

```
4 -- 5
```

```
5 -- 25
```

```
2 -- 25
```

```
25 -- -1
```

Time Complexity: $O(n^2)$

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.

```

import java.util.ArrayList;

class Node {
    int data;
    Node left, right;

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

class righttree {
    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};

        RecursiveRightView(root, 0, maxLevel, result);

        return result;
    }

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

```

```
D:\>javac righttree.java
```

```
D:\>java righttree
```

```
1 3 5
```

Time 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.

```
class Node {
    int data;
    Node left, right;

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

class maxtree {
    static int maxDepth(Node node) {
        if (node == null)
            return 0;

        int lDepth = maxDepth(node.left);
        int rDepth = maxDepth(node.right);

        return Math.max(lDepth, rDepth) + 1;
    }

    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);

        System.out.println(maxDepth(root));
    }
}
```

```
D:\>javac maxtree.java
```

```
D:\>java maxtree
```

```
3
```

Time Complexity: $O(n)$