DSA PROBLEMS(09/11/24)

LYDIA V(22CB026)

B.TECH CSBS

1. Maximum Subarray Sum – Kadane's Algorithm:

```
import java.util.*;
public class tUf {
  public static long maxSubarraySum(int[] arr, int n) {
    long maxi = Long.MIN_VALUE;
    long sum = 0;
    int start = 0;
    int ansStart = -1, ansEnd = -1;
    for (int i = 0; i < n; i++) {
      if (sum == 0) start = i;
      sum += arr[i];
      if (sum > maxi) {
        maxi = sum;
        ansStart = start;
        ansEnd = i;
      }
      if (sum < 0) {
        sum = 0;
      }
    }

    System.out.print("The subarray is: [");
    for (int i = ansStart; i <= ansEnd; i++) {
      System.out.print(arr[i] + " ");
    }
    System.out.print("]n");
```

```
        return maxi;
    }
    public static void main(String args[]) {
        int[] arr = { -2, 1, -3, 4, -1, 2, 1, -5, 4};
        int n = arr.length;
        long maxSum = maxSubarraySum(arr, n);
        System.out.println("The maximum subarray sum is: " + maxSum);
    }
}
```

OUTPUT:

```
The subarray is: [4 -1 2 1 ]The maximum subarray sum is: 6
```

TIME COMPLEXITY :O(N)

SPACE COMPLEXITY: O(1)

2. Maximum Product Subarray :

```
import java.util.*;
public class Kadan{
public static int MaxArr(int arr[]) {
    int l=arr.length;
    int pre=1,post=1;
    int res=Integer.MIN_VALUE;
    for(int i=0;i<l;i++){
        if(pre==0) pre=1;
        if(post==0) post=1;
        pre*=arr[i];
        post*=arr[l-i-1];
        res=Math.max(res, Math.max(pre,post));
    }
```

```
   return res;
  }
  public static void main(String[] args) {
   int arr[]={-2, 6, -3, -10, 0, 2};
   int ans=MaxArr(arr);
   System.out.println("The Maximum product of subarray is"+ans);
  }
}
```

OUTPUT:

```
The Maximum product of subarray is: 180
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

3. Search in a sorted and rotated Array

```
import java.util.*;
public class GFG {
    public static int pivotedSearch(List<Integer> arr, int key) {
        int low = 0, high = arr.size() - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr.get(mid) == key)
                return mid;
            if (arr.get(mid) >= arr.get(low)) {
                if (key >= arr.get(low) && key < arr.get(mid))
                    high = mid - 1;
                else
                    low = mid + 1;
            }
            else {
                if (key > arr.get(mid) && key <= arr.get(high))
                    low = mid + 1;
                else
                    high = mid - 1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        List<Integer> arr1 = Arrays.asList(4, 5, 6, 7, 0, 1, 2);
        int key1 = 0;
        int result1 = pivotedSearch(arr1, key1);
        System.out.println("The index of given key element is:"+ result1); // Output: 4    }
}
```

OUTPUT:

```
The index of given key element is:8
```

TIME COMPLEXITY:O(LOG N)

SPACE COMPLEXITY:O(1)

4. Container with Most Water :

```
import java.util.*;

class Solution {
    public int maxArea(int[] height) {
        int m=0;
        int left=0;
        int right=height.length-1;
        while(left < right){
            m=Math.max(m,(right-left)*Math.min(height[left],height[right]));
        if(height[left]<height[right]){
            left++;
        }
        else{
        right--;}}
        return m;
    }
 public static void main(String[] args) {
        int[] arr1 ={ 1,8,6,2,5,4,8,3,7};
        int result1 = MaxArea(arr1);
        System.out.println("The max is:"+ result1); // Output: 49
}
```

OUTPUT:

```
The max is :6
```

TIME COMPLEXITY:O(log n)

SPACE COMPLEXITY:O(1)

5. Find the Factorial of a large number

```
import java.math.BigInteger;
import java.util.Scanner;

public class Example {
    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 = 100;
        System.out.println("Factorial of given number is:"+factorial(N));
    }
}
```

OUTPUT:

```
Factorial of given number is:
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976
156518286253697920827223758251185210916864000000000000000000000000
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

6. Trapping Rainwater Problem:

```
import java.util.*;
class TUF {
    static int trap(int[] height) {
        int n = height.length;
        int left = 0, right = n - 1;
        int res = 0;
        int maxLeft = 0, maxRight = 0;
        while (left <= right) {
            if (height[left] <= height[right]) {
                if (height[left] >= maxLeft) {
                    maxLeft = height[left];
                } else {
                    res += maxLeft - height[left];
                }
                left++;
            } else {
                if (height[right] >= maxRight) {
                    maxRight = height[right];
                } else {
                    res += maxRight - height[right];
                }
                right--;
            }
        }
        return res;
    }
    public static void main(String args[]) {
        int arr[] =  {3, 0, 2, 0, 4} ;
        System.out.println("The duplicate element is " + trap(arr));
    }
}
```

OUTPUT:

```
The duplicate element is 7
```

TIME COMPLEXITY: O(N)

SPACE COMPLEXITY:O(1)

7. Chocolate Distribution Problem

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

```
public class Main {


  public static int findMinimumDifference(int[] packets, int numStudents) {
    int numPackets = packets.length;

    if (numPackets < numStudents)
      return -1;

    int minDifference = Integer.MAX_VALUE;

    Arrays.sort(packets);

    for (int i = 0; i <= numPackets - numStudents; i++) {

      int maxPacket = packets[i+numStudents-1];
      int minPacket = packets[i];

      int difference = Math.abs(maxPacket - minPacket);
      if (difference < minDifference)
        minDifference = difference;
    }
    return minDifference;
  }

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

```
        int numStudents = 3;


        int minDifference = findMinimumDifference(packets, numStudents);


        if (minDifference == -1)
            System.out.println("Invalid input");
        else
            System.out.println("Minimum difference is " + minDifference);
    }
}
```

Output:

```
Minimum difference is 2
```

TIME COMPLEXITY: O(N log N)

SPACE COMPLEXITY:O(N)

8. Merge Overlapping Intervals

```
import java.util.*;


public class Main {


    public static List<List<Integer>> mergeOverlappingIntervals(int[][] arr) {
        int n = arr.length;
        Arrays.sort(arr, new Comparator<int[]>() {
            public int compare(int[] a, int[] b) {
                return a[0] - b[0];
            }
        });
        List<List<Integer>> ans = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            if (ans.isEmpty() || arr[i][0] > ans.get(ans.size() - 1).get(1)) {
                ans.add(Arrays.asList(arr[i][0], arr[i][1]));
            }
            else {
                ans.get(ans.size() - 1).set(1,
                                            Math.max(ans.get(ans.size() - 1).get(1), arr[i][1]));
            }
        }
        return ans;
    }

    public static void main(String[] args) {
        int[][] arr = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        List<List<Integer>> ans = mergeOverlappingIntervals(arr);
        System.out.print("The merged intervals are: \n");
        for (List<Integer> it : ans) {
            System.out.print("[" + it.get(0) + ", " + it.get(1) + "] ");
        }
        System.out.println();
    }
}
```

OUTPUT:

```
The merged intervals are:
[1, 4] [6, 8] [9, 10]
```

TIME COMPLEXITY: O(N*logN)+O(N)

SPACE COMPLEXITY:O(N)

9. A Boolean Matrix Question

```
import java.io.*;
class Main {
    public static void modifyMatrix(int mat[][])
    {
        boolean row_flag = false;
        boolean col_flag = false;

        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[0].length; j++) {
                if (i == 0 && mat[i][j] == 1)
                    row_flag = true;

                if (j == 0 && mat[i][j] == 1)
                    col_flag = true;

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

                    mat[0][j] = 1;
                    mat[i][0] = 1;
                }
            }
        }

        for (int i = 1; i < mat.length; i++)
            for (int j = 1; j < mat[0].length; j++)
                if (mat[0][j] == 1 || mat[i][0] == 1)
                    mat[i][j] = 1;
        if (row_flag == true)
            for (int i = 0; i < mat[0].length; i++)
                mat[0][i] = 1;

        if (col_flag == true)
            for (int i = 0; i < mat.length; i++)
                mat[i][0] = 1;
    }

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

    public static void main(String args[])
    {
        int mat[][] =  {{1, 0},
                        {0, 0}};

        System.out.println("Input Matrix :");
        printMatrix(mat);
        modifyMatrix(mat);
        System.out.println("Matrix After Modification :");
        printMatrix(mat);
    }
}
```

OUPUT:

```
Input Matrix :
1 0
0 0
Matrix After Modification :
1 1
1 0
```

TIME COMPLEXITY:O(M*N)

SPACE COMPLEXITY:O(1)

10. Print a given matrix in spiral form

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

public class Main {

    public static List<Integer> printSpiral(int[][] mat) {
        List<Integer> ans = new ArrayList<>();
        int n = mat.length;
        int m = mat[0].length;
        int top = 0, left = 0, bottom = n - 1, right = m - 1;
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++)
                ans.add(mat[top][i]);
            top++;
            for (int i = top; i <= bottom; i++)
            ans.add(mat[i][right]);
        right--;
        if (top <= bottom) {
            for (int i = right; i >= left; i--)
                ans.add(mat[bottom][i]);

            bottom--;
        }
        if (left <= right) {
            for (int i = bottom; i >= top; i--)
                ans.add(mat[i][left]);

            left++;
        }
    }
    return ans;
}

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

    List<Integer> ans = printSpiral(mat);

    for(int i = 0;i<ans.size();i++){
        System.out.print(ans.get(i) + " ");
    }
```

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

OUTPUT:

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

TIME COMPLEXITY:O(M*N)

SPACE COMPLEXITY: O(N)

11. Check if given Parentheses expression is balanced or not:

```
import java.util.*;
class TUF {
public static boolean isValid(String s) {
        Stack<Character> st = new Stack<Character>();
        for (char it : s.toCharArray()) {
            if (it == '(' || it == '[' || it == '{')
                st.push(it);
            else {
                if(st.isEmpty()) return false;
                char ch = st.pop();
                if((it == ')' && ch == '(') ||  (it == ']' && ch == '[') || (it == '}' && ch == '{')) continue;
                else return false;
            }
        }
        return st.isEmpty();
    }

 public static void main (String[] args) {
```

```
              String s="((()))()()";
              if(isValid(s)==true)
              System.out.println("Balanced");
              else
              System.out.println("Not Balanced");
       }
}
```

OUTPUT:

```
Balanced
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(N)

14. Check if two Strings are Anagrams of each other

```
public class Main {
 public static boolean checkAnagrams(String str1, String str2) {
  str1 = str1.toUpperCase();
  str2 = str2.toUpperCase();
  if (str1.length() != str2.length())
   return false;

  int[] freq = new int[26];
  for (int i = 0; i < str1.length(); i++) {
   freq[str1.charAt(i) - 'A']++;
  }
  for (int i = 0; i < str2.length(); i++) {
   freq[str2.charAt(i) - 'A']--;
  }
  for (int i = 0; i < 26; i++) {
   if (freq[i] != 0)
    return false;
  }
  return true;
 }

 public static void main(String args[]) {
  String Str1 = "geeks";
  String Str2 = "kseeg";
  System.out.println(checkAnagrams(Str1, Str2)); // Output: true
 }
}
```

OUTPUT:

```
True
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

12. Longest Palindromic Substring

```
import java.util.*;

class TUF {
  static int lcs(String s1, String s2) {
    int n = s1.length();
    int m = s2.length();
    int[] prev = new int[m + 1];
    int[] cur = new int[m + 1];
    for (int ind1 = 1; ind1 <= n; ind1++) {
      for (int ind2 = 1; ind2 <= m; ind2++) {
        if (s1.charAt(ind1 - 1) == s2.charAt(ind2 - 1))
          cur[ind2] = 1 + prev[ind2 - 1];
```

```
                else
                    cur[ind2] = Math.max(prev[ind2], cur[ind2 - 1]);
            }
            prev = cur.clone();
        }
        return prev[m];
    }
    static int longestPalindromeSubsequence(String s) {
        String reversed = new StringBuilder(s).reverse().toString();
        return lcs(s, reversed);
    }
    public static void main(String args[]) {
        String s = "forgeeksskeegfor";
        System.out.print("The Length of Longest Palindromic Subsequence is ");
        System.out.println(longestPalindromeSubsequence(s));
    }
}
```

OUTPUT:

```
The Length of Longest Palindromic Subsequence is 12
```

TIME COMPLEXITY:O(N*N)

SPACE COMPLEXITY:O(N)

14. Longest Common Prefix using Sorting

```
import java.util.Arrays;

class Main {
    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("The longest common prefix is: "

                 + longestCommonPrefix(arr));

    }

}
```

OUTPUT:

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

TIME COMPLEXITY:O(N LOG N+M)

SPACE COMPLEXITY:O(1)

15. Delete middle element of a stack:

```
import java.util.Stack;

public class Main {
```

```
public static void deleteMiddle(Stack<Integer> stack, int currentIndex, int size) {
    if (currentIndex == size / 2) {
        stack.pop();
        return;
    }
    int topElement = stack.pop();
    deleteMiddle(stack, currentIndex + 1, size);
    stack.push(topElement);
}
public static void deleteMiddle(Stack<Integer> stack) {
    int size = stack.size();
    deleteMiddle(stack, 0, size);
}
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);
    deleteMiddle(stack);
    System.out.println("Stack after deleting middle element: " + stack);
    Stack<Integer> stack2 = new Stack<>();
    stack2.push(1);
    stack2.push(2);
    stack2.push(3);
    stack2.push(4);
    stack2.push(5);
```

```
        stack2.push(6);
        System.out.println("Original Stack: " + stack2);
        deleteMiddle(stack2);
        System.out.println("Stack after deleting middle element: " + stack2);
    }
}
```

OUTPUT:

```
Original Stack: [1, 2, 3, 4, 5]
Stack after deleting middle element: [1, 2, 4, 5]
Original Stack: [1, 2, 3, 4, 5, 6]
Stack after deleting middle element: [1, 2, 4, 5, 6]
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

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

```
import java.util.Stack;
public class Main {
    public static void printNextGreater(int[] arr) {
        int n = arr.length;
        Stack<Integer> stack = new Stack<>();
        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            if (!stack.isEmpty()) {
                System.out.println(arr[i] + " --> " + stack.peek());
            } else {
                System.out.println(arr[i] + " --> -1");
            }
            stack.push(arr[i]);
        }
    }
    public static void main(String[] args) {
        int[] arr1 = {4, 5, 2, 25};
        System.out.println("Next Greater Elements for arr1:");
        printNextGreater(arr1);
    }
}
```

OUTPUT:

```
Next Greater Elements for arr1:
25 --> -1
2 --> 25
5 --> 25
4 --> 5
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(N)

17. Print Right View of a Binary Tree

```
import java.util.ArrayList;
import java.util.List;
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) {
        val = x;
    }
```

```java
}
public class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        rightView(root, result, 0);
        return result;
    }
    public void rightView(TreeNode curr, List<Integer> result, int currDepth) {
        if (curr == null) {
            return;
        }
        if (currDepth == result.size()) {
            result.add(curr.val);
        }
        rightView(curr.right, result, currDepth + 1);
        rightView(curr.left, result, currDepth + 1);
    }
    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);
        root.right.right = new TreeNode(6);
        root.left.left.left = new TreeNode(7);
        Solution solution = new Solution();
        List<Integer> rightView = solution.rightSideView(root);
        System.out.println("Right side view of the binary tree: " + rightView);
    }
}
```

OUTPUT:

```
Right side view of the binary tree: [1, 3, 6, 7]
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(H)

18. Maximum Depth or Height of Binary Tree

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) {
        val = x;
    }
}
public class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) return 0;
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        return Math.max(left, right) + 1;
    }
    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);
        root.right.right = new TreeNode(6);
        root.left.left.left = new TreeNode(7);
        Solution solution = new Solution();

        int depth = solution.maxDepth(root);

        System.out.println("Maximum depth of the binary tree: " + depth);

    }

}
```

OUTPUT:

```
Maximum depth of the binary tree: 4
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(N)