

PRACTICE SET 1 09.11.24

NAME: HARSHITHAA RG

DEPARTMENT: IT

REGISTER NO.: 22IT035

20. Maximum depth of a binary tree(height)

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.

```
import java.util.*;

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

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

public class HelloWorld {
    public static int height(Node root){
        if(root==null){
            return 0;
        }
        int level=0;
```

```

Queue<Node> q = new LinkedList<>();
q.add(root);
while(!q.isEmpty()){
    int size=q.size();
    for(int i=0;i<size;i++){
        Node front=q.poll();
        if(front.left!=null){
            q.add(front.left);
        }
        if(front.right!=null){
            q.add(front.right);
        }
    }
    level++;
}
return level;
}

public static void main(String[] args) {
    Node root=new Node(12);
    root.left= new Node(8);
    root.right= new Node(18);
    root.left.left=new Node(5);
    root.left.right=new Node(11);
    int treeheight=height(root);
    System.out.println("Height of the tree is"+treeheight);
}
}

```

Output:

```
Height of the tree is 3
```

Time complexity: $O(N)$

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.

Code:

```
import java.util.*;

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

public class HelloWorld {

    public static void rightview(Node root,List<Integer> res,int lvl){
        if(root==null){
            return;
        }
        if(res.size()==lvl){
            res.add(root.data);
```

```

    }
    rightview(root.right,res,lvl+1);
    rightview(root.left,res,lvl+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.left.right=new Node(5);
    List<Integer> res=new ArrayList<>();
    rightview(root,res,0);
    for(int i:res){
        System.out.println(i);
    }
}

```

Output:

```

1
3|
4
5

```

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.

Code:

```
import java.util.*;
```

```

class HelloWorld {
    public static void nextgrt(int[] arr, int[] ans){
        Stack<Integer> st=new Stack<>();
        for(int i=arr.length-1;i>=0;i--){
            while(!st.isEmpty() && arr[i]>=st.peek()){
                st.pop();
            }
            if(i<arr.length){
                if(!st.isEmpty()){
                    ans[i]=st.peek();
                }
                else{
                    ans[i]=-1;
                }
            }
            st.push(arr[i]);
        }
    }
    public static void main(String[] args) {
        int arr[]={4,5,2,25};
        int ans[]=new int[arr.length];
        nextgrt(arr,ans);
        for(int i:ans){
            System.out.print(i+" ");
        }
    }
}

```

Output:

```
5 25 25 -1
```

Time Complexity: $O(N)$

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.

Code:

```
import java.util.*;

class HelloWorld {

    public static void dltnmid(Stack<Integer> st, int n, int curr){
        if(st.isEmpty() || curr==n){
            return;
        }
        int x=st.pop();
        dltnmid(st,n,curr+1);
        if(curr!=n/2){
            st.push(x);
        }
    }

    public static void main(String[] args) {
        Stack<Integer> st=new Stack<>();
        st.push(6);
        st.push(5);
        st.push(4);
        st.push(3);
        st.push(2);
```

```

    st.push(1);
    int n=st.size();
    dltmid(st,n,0);
    while(!st.isEmpty()){
        int i=st.pop();
        System.out.print(i+" ");
    }
}
}

```

Output:

```
1 2 4 5
```

Time Complexity: $O(N)$

16. Longest Common Prefix

Code:

```

class HelloWorld {
    public static String longpr(String[] arr){
        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.length()==0){
                    return "";
                }
            }
        }
        return prefix;
    }
}

```

```

public static void main(String[] args) {
    String arr[]={"geeksforgeeks", "geeks", "geek", "geezer"};
    String st=longpr(arr);
    System.out.println(st);

}
}

```

Output:



Time Complexity: $O(N)$

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.

Code:

```

import java.util.*;

class HelloWorld {

    public static String longestpalindrome(String st){
        if(st.length()<=1) return st;
        String subst=st.substring(0,1);
        for(int i=0;i<st.length()-1;i++){
            String odd=expand(st,i,i);
            String even=expand(st,i,i+1);
            if(odd.length()>subst.length()) subst=odd;
            if(even.length()>subst.length()) subst=even;
        }
    }
}

```



```

        return subst;
    }

    public static String expand(String st, int left, int right){
        while(left>=0 && right<st.length() && st.charAt(left)==st.charAt(right)){
            left--;
            right++;
        }
        return st.substring(left+1,right);
    }

    public static void main(String[] args) {
        String st="forgeeksskeegfor";
        System.out.println("Longest Palindrome: "+longestpalindrome(st));
    }
}

```

Output:

```
Longest Palindrome: geeksskeeg
```

Time Complexity: $O(N^2)$

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.

Code:

```

import java.util.*;

class HelloWorld {
    public static boolean anagram(String s1, String s2){

```

```

HashMap<Character, Integer> map=new HashMap<>();
for(int i=0;i<s1.length();i++){
    char x=s1.charAt(i);
    map.put(x, map.getOrDefault(x,0)+1);
}
for(int j=0;j<s2.length();j++){
    char x=s2.charAt(j);
    map.put(x, map.getOrDefault(x,0)-1);
}
for(int u:map.values()){
    if(u!=0){
        return false;
    }
}
return true;
}

public static void main(String[] args) {
    String s1="allergy";
    String s2="allergic";
    String s3="geeks";
    String s4="keegs";
    boolean ans= anagram(s1,s2);
    boolean ans1= anagram(s3,s4);
    System.out.println(ans);
    System.out.println(ans1);

}

```

```
}
```

Output:

```
false  
true
```

Time Complexity: $O(N+M)$

13. 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:

```
import java.util.*;  
  
class HelloWorld {  
    public static boolean validpar(Stack<Character> st, String str){  
        for(int i=0;i<str.length();i++){  
            if(str.charAt(i)=='('){  
                st.push(str.charAt(i));  
            }  
            else{  
                if(!st.isEmpty() && str.charAt(i)==')' && st.peek()=='('){  
                    st.pop();  
                }  
  
                else{  
                    return false;  
                }  
            }  
        }  
    }  
}
```

```

        if(st.isEmpty()){
            return true;
        }
        return false;

    }

    public static void main(String[] args) {
        Stack<Character> st=new Stack<>();
        String str="((()))()";
        boolean ans=validpar(st,str);
        System.out.println(ans);

    }
}

```

Output:

```

true

```

Time Complexity: $O(N)$

10. Print a given matrix in spiral form

Given an $m \times n$ matrix, the task is to print all elements of the matrix in spiral form.

Code:

```

import java.util.*;

class HelloWorld {

    public static ArrayList<Integer> spiral(int[][] arr){
        int rows=0;
        int rowe=arr.length-1;

```

```

int cols=0;
int cole=arr[0].length-1;
ArrayList<Integer> fin=new ArrayList<>();
while(rows<=rowe && cols<=cole){
    for(int i=cols;i<=cole;i++){
        fin.add(arr[rows][i]);
    }
    rows++;
    for(int j=rows;j<=rowe;j++){
        fin.add(arr[j][cole]);
    }
    cole--;
    if(rows<=rowe){
        for(int y=cole;y>=cols;y--){
            fin.add(arr[rowe][y]);
        }
        rowe--;
    }
    if(cols<=cole){
        for(int t=rowe;t>=rows;t--){
            fin.add(arr[t][cols]);
        }
        cols++;
    }
}
return fin;
}

```

```

public static void main(String[] args) {
    int[][] arr={{1,2,3,4,5,6},{7,8,9,10,11,12},{13,14,15,16,17,18}};
    ArrayList<Integer> res=spiral(arr);
    System.out.println(res);
}
}

```

Output:

```
[1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11]
```

Time Complexity: $O(N)$

9. A Boolean Matrix Question

Given a boolean matrix $mat[M][N]$ of size $M \times N$, 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.

Code:

```

import java.util.*;

class HelloWorld {
    public static void boolmat(int mat[][])
    {
        boolean row= false;
        boolean col= 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 = true;
                }
                if (j == 0 && mat[i][j] == 1){
                    col= 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 == true){
    for (int i = 0; i < mat[0].length; i++){
        mat[0][i] = 1;
    }
}

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

public static void printMatrix(int mat[][]){

```

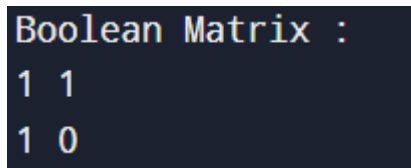
```

        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 } };
        boolmat(mat);
        System.out.println("Boolean Matrix :");
        printMatrix(mat);
    }
}

```

Output:



```

Boolean Matrix :
1 1
1 0

```

Time Complexity: $O(m*n)$

8. Merge Overlapping Intervals

Given an array of time intervals where $arr[i] = [start_i, end_i]$, 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 HelloWorld {
    public static List<List<Integer>> merge(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=merge(arr);
    for(List<Integer> i: ans){
        System.out.print "["+i.get(0)+"," +i.get(1)+"]");
    }
}
}

```

Output:

```
[1,4][6,8][9,10]
```

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: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.

Code:

```
import java.util.*;

class HelloWorld {

    public static int Mindiff(int[] arr,int m){
        int n=arr.length;
        Arrays.sort(arr);
        int mindiff=Integer.MAX_VALUE;
        for(int i=0;i<=n-m;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("Minimum difference is: "+Mindiff(arr, m));
}
}

```

Output:

```

Minimum difference is: 2

```

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.

Code:

```

class Hello {
    public static int water(int[] height){
        int[] prefixmax=new int[height.length];
        prefixmax[0]=height[0];
        for(int i=1;i<height.length;i++){
            prefixmax[i]=Math.max(prefixmax[i-1],height[i]);
        }
        int[] suffixmax=new int[height.length];
        suffixmax[height.length-1]=height[height.length-1];
        for(int i=height.length-2;i>=0;i--){
            suffixmax[i]=Math.max(suffixmax[i+1],height[i]);
        }
    }
}

```

```

int water=0;
for(int i=0;i<height.length;i++){
    water+=Math.min(prefixmax[i],suffixmax[i])-height[i];
}
return water;
}

public static void main(String[] args) {
    int[] height= {3, 0, 1, 0, 4, 0, 2};
    int water=water(height);
    System.out.println("Trapped water level is "+water);

}
}

```

Output:

```
Trapped water level is 10
```

Time Complexity: $O(3n)$

5. Find the Factorial of a large number

Code:

```

import java.math.BigInteger;

class HelloWorld {
    public static BigInteger fact(int n){
        BigInteger big=new BigInteger("1");
        for(int i=2;i<=n;i++){
            big=big.multiply(BigInteger.valueOf(i));
        }
        return big;
    }
}

```

```

public static void main(String[] args) {
    int n=100;
    System.out.println(fact(100));
}
}

```

Output:

```

9332621544394415268169923885626670049071596826438162146859296389521759999322
9915608941463976156518286253697920827223758251185210916864000000000000000
0000000000

```

Time Complexity: $O(N)$

4. Container with Most Water

Code:

```

class HelloWorld {
    public static int area(int[] height){
        int left=0;
        int right=height.length-1;
        int area=0;
        int maxarea=0;
        while(left<right){
            area=Math.min(height[left],height[right])*(right-left);
            maxarea=Math.max(area,maxarea);
            if(height[left]<height[right]){
                left++;
            }else{
                right--;
            }
        }
        return maxarea;
    }
}

```

```

    }

    public static void main(String[] args) {
        int height[]={1, 5, 4, 3};
        int maxarea=area(height);
        System.out.println("The maximum amount of water is "+maxarea);

    }
}

```

Output:

```
The maximum amount of water is 6
```

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:

```

class HelloWorld {
    public static int search(int[] nums, int target){
        int low=0;
        int high=nums.length-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(nums[mid]==target){
                return mid;
            }
            if(nums[low]<=nums[mid]){
                if(nums[low]<=target && target<=nums[mid]){

```

```

        high=mid-1;
    }
    else{
        low=mid+1;
    }
}
else{
    if(nums[mid]<=target && target<=nums[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};
    int target=0;
    int ans=search(arr,target);
    System.out.println("The target is found in "+ans);

}
}

```

Output:

```
The target is found in 4
```

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

2. Maximum Product Subarray

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

Code:

```
import java.util.*;

class HelloWorld {

    public static int maxproduct(int[] nums){

        int n=nums.length;

        int leftprod=1;

        int rightprod=1;

        int ans=nums[0];

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

            leftprod=leftprod==0?1:leftprod;

            rightprod=rightprod==0?1:rightprod;

            leftprod*=nums[i];

            rightprod*=nums[n-1-i];

            ans=Math.max(ans,Math.max(leftprod,rightprod));

        }

        return ans;

    }

    public static void main(String[] args) {

        int[] arr={-2,6,-3,-10,0,2};

        int ans=maxproduct(arr);

        System.out.println("Maximum Product Subarray: "+ans);

    }

}
```


Output:

```
Maximum Product Subarray: 180
```

Time Complexity: $O(N)$

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.*;

class HelloWorld {

    public static long maxSubarraySum(int[] arr, int n) {

        long maxsum = Long.MIN_VALUE;
        long sum = 0;

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

            sum += arr[i];

            if (sum > maxsum) {

                maxsum = sum;

            }

            if (sum < 0) {

                sum = 0;

            }

        }

        return maxsum;

    }

    public static void main(String args[]) {

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

```
int n = arr.length;
long maxSum = maxSubarraySum(arr, n);
System.out.println("Maximum subarray sum is: " + maxSum);
}
}
```

Output:

```
Maximum subarray sum is: 11
```

Time Complexity: $O(N)$

