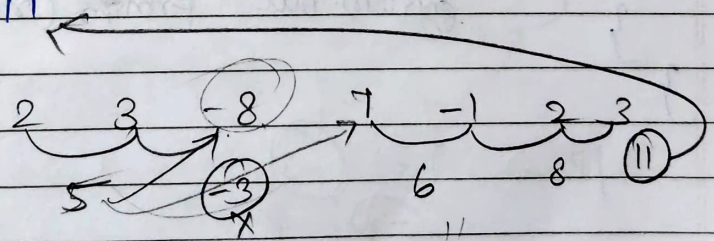
Practice Set - 1

1. Max Sum in Subarray;

```
import java.util.*;
class MaxSum{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("enter size: ");
        int n = sc.nextInt();
        System.out.println("enter elements: ");
        int[] arr = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
        int max = arr[0];
        int result = 0;
        for(int i=1; i<n; i++){
            result = Math.max(max+arr[i], arr[i]);
            max = Math.max(max, result);
        }
        System.out.println(result);
    }
}
```

I/P: 2 3 -8 -1 2 3

O/P: 11



Max = 0
→ 2
→ 5 ✓

(6, -1) ✓ — 6 + 2 = 8 + 3 = 11

max = (7)

Time comp: ⇒ O(n);

2. Maximum Product Subarray :

```
import java.util.*;
class maxProduct {
    public static void main (String [] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] a = new int [n];
        for (int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
        int max = arr[0];
        int min = arr[0];
        int result = arr[0];
        for(int i=1; i<n; i++){
            if(arr[i]<0){
                int temp = max
                max = min;
                min = temp;
            }
            max = Math.max(max*arr[i], arr[i]);
            min = Math.min(min*arr[i], arr[i]);
            result = Math.max(max, result);
        }
        System.out.println(result);
    }
}
```

I/P => {-1, -3, -10, 0, 60}

max: -1 -3 0, -10 60 ← (0, 60)

min: -3

res: -1

O/P => {60}

Time Comp => O(n)

3) Search in Sorted Array!

```
import java.util.*;

class Searchrotate {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
        boolean = false;
        int x = sc.nextInt(); // element to be search;
        for(int i=0; i<n; i++){
            if(arr[i]==x){
                found = true;
                System.out.println(i);
                break;
            }
        }
        if(!found){
            return false; -1;
        }
    }
```

I/P:

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

key = 3

∴ return -1

Time comple ⇒ O(n)

→ Brute comp! $O(n^2)$

4. Container with Most water

```
import java.util.*;

Class Mostwater{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = new sc.nextInt();
        int[] arr = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
        int area;
        int max=0
        for(int i=0; i<n; i++){
            max=0;
            for(int j=i+1; j<n; j++){
                area = (j-i) * Math.min(arr[i], arr[j]);
                if(area > max){
                    max = area;
                }
            }
        }
        System.out.println(area);
    }
}
```

I/P: [1, 5, 4, 3]

max = 0

area = 1

[1,5] 1*4 = 4 ✗ [1,4] 3*1 = 3 ✗ [1,3] 2*1 = 2 ✗

[5,4] 1*4 = 4 ✗

[5,3] = 2*3 = (6)

[4,3] = 1*3 = 3 ✗

∴ Max = 6

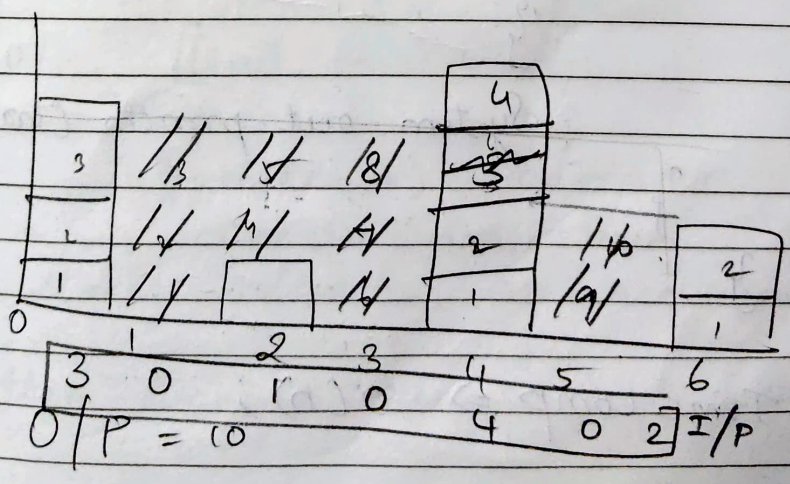
5. Factorial:

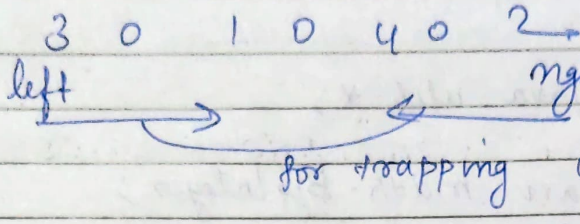
```
import java.util.*;
class
import java.math.BigInteger;
class factorial{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
        BigInteger result = BigInteger.ONE;
        for(int i=0; i<n; i++){
            result = result.multiply
                (BigInteger.valueOf(i));
        }
        System.out.println(result);
    }
}
```

I/P → 5! → 120 ← O/P

Time Comp => O(n)

6. Trapping rain water!






```
import java.util.*;
Class TrappingRain{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for(int i=0; i<n; i++){ arr[i]=sc.nextInt(); }
        int [] left = new int[n];
        int [] right = new int[n];
        left [0] = arr[0]
        for (int i=1, i<n; i++){
            left = Math.max (arr[i], arr[i-1]);
        }
        right [n-1] = arr [n-i]
        for (int i=n-2; i>=0; i--){
            right = Math.max (arr[i], arr[i+1]);
        }
        int trap = 0;
        for (int i=0; i<n; i++){
            trap += Math.min (left[i],
                              right[i])-arr[i];
        }
        System.out.println(trap);
    }
}
```

Time comp ⇒ O(n);

7. Chocolate distribution:

arr = {7, 3, 2, 4, 9, 12, 56} m=3

first 3 elements (smallest) 1. Sort

compare last & first ← take m elements

```
import java.util.*;
class Chocolate{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[];
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
int m=sc.nextInt();
        Arrays.sort(arr); int res;
        for(int i=0; i<m; i++){
            res = arr[m-1] - arr[0];
        }
        System.out.println(res);
    }
}
```

Time comple: O(n log n),

8) I/P ⇒ [[1,3],[2,4],[6,8],[9,10]]

1 2 3 4 | 6 8 | 9 10

O/P: [[1,4],[6,8],[9,10]]

→ create comparator, create merge array to store intervals as required

then compare them as last with 1st digit of interval

```
import java.util.*;
public class MergeIntervals{
    public static int [][] merge(int [][] intervals){
        Arrays.sort(intervals, (a,b) -> Integer.compare(a[0], b[0]));
        LinkedList<int> merged = new LinkedList<>();
        for (int[] interval : intervals){
            if (merged.isEmpty() || merged.getLast()[1] < interval[0]){
                merged.add(interval);
            }else{
                merged.getLast()[1] = Math.max(merged.getLast()[1], interval[1]);
            }
        }
        return merged.toArray(new int[merged.size()][]);
    }
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("enter the size: ");
        int n = sc.nextInt();
        int [][] intervals = [n][2];
        for(int i=0; i<n; i++){
            System.out.println("start: "); intervals[i][0] = sc.nextInt();
            System.out.println("end: "); intervals[i][1] = sc.nextInt();
        }
        int [][] minterval = merge(intervals);
        for (int i=0; i<n; i++){
            System.out.println(" + interval[0] + ", " + interval[1]);
        }
    }
}
```

enter intervals

enter Merged intervals

Time comp : O(n log n)

Q. Boolean Matrix:

first traverse the matrix once & Set row[i] & cols[j] = true if mat[i][j] is 1

then update the Matrix:

if rows[i] or cols[j] is true set mat[i][j]=1

Printmatrix helps us to print o/p matrix

```
import java.util.*;
class BooleanMatrix{
    public static void ModifyMatrix(int mat[][]){
        int M = mat.length;
        int N = mat[0].length;
// to create array to mark the rows, cols updated
        boolean[] rows = new boolean[M];
        boolean[] cols = new boolean[N];
// Traverse
        for(int i=0; i<M; i++){
            for(int j=0; j<N; j++){
                if(mat[i][j]==1){
                    rows[i]=true;
                    cols[j]=true;
                }
            }
        }
// update the Matrix
        for(int i=0; i<M; i++){
            for(int j=0; j<N; j++){
                if(rows[i] || cols[j]){
                    mat[i][j]=1;
                }
            }
        }
    }
```

~~T.C = O(n²)~~

```
    public static void main(String[] args){
        Scanner Sc = new Scanner(System.in);
        Print -> no. of rows:
        int M = Scanner.nextInt();
        Print -> no. of cols:
        int N = Scanner.nextInt();
```

```
int [][] mat = new int [m] [N];
Print → enter matrix elements!
for (int i=0; i<M; i++){
    for (int j=0; j<N; j++){
        System.out.println(i+j);
        mat [i] [j] = sc.nextInt();
    }
}
```

a fn

```
print → original Matrix
printmatrix (mat);
modifymatrix (mat);
print → (modified matrix);
printMatrix (mat);
}
public static void printMatrix (int[][] mat){
    for (int[] row: mat){
        System.out.println(Arrays.
            toString (row));
    }
    System.out.println();
}
```

I/P: → I/P {{1,0}, {0,0}}

O/P:- modified Matrix:

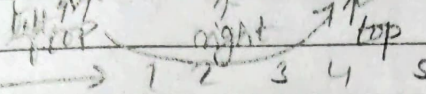
→ [0,1], [1,0]

we can also use curly braces

Time complexity [O(MXN)]

10. Matrix in Spiral form.

I/P: {{1,2,3,4},{5,6,7,8},{9,10,11,12}}

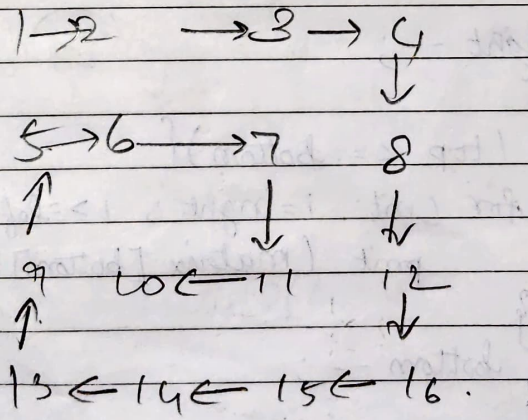


O/P: 1 2 3 4 5 6 7 8 9 10 11 12

```
class SpiralMatrix{
    public static void printSpiral(int[][] matrix){
        if (matrix == null || matrix.length == 0 ||
                matrix[0].length == 0){
        } return; // empty matrix

    int top = 0,  bottom = matrix.length - 1;
    int left = 0;  right = matrix[0].length - 1;

    while (top <= bottom && left <= right){
        // to print top row from left to right
        for (int i=left; i<= right; i++){
            print (matrix[top][i] + " ");
        }
        top++; // move top down (to print next row)
    }
        // to print right col top to bottom
        for (int i = top; i <= bottom; i++){
            print (matrix[i][right] + " ");
        }
        right--; // move left ← right

        if (top <= bottom){
            // print bottom row
            for (int i = right; i >= left; i--){
                print (matrix[bottom][i] + " ");
            }
            bottom--;
        }
```

```
if (left <= right) {
    for (int i = bottom; i >= top; i--) {
        print(matrix[i][left] + " ");
    }
    left++;
}
}
}
```

```
Public static void main (String[] args) {
    Scanner sc = new Scanner(System.in);
    Print("no rows: ");
    int m = sc.nextInt();
    Print("no-of cols: ");
    int n = sc.nextInt();
    int[][] matrix = new matrix[m][n];
<--> Enter elem:
    for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++) {
            print("[" + i + "]" + "[" + j + "]");
            matrix[i][j] = sc.nextInt();
        }
    }
    System.out.println("Spiral matrix");
    printSpiral(matrix);
```

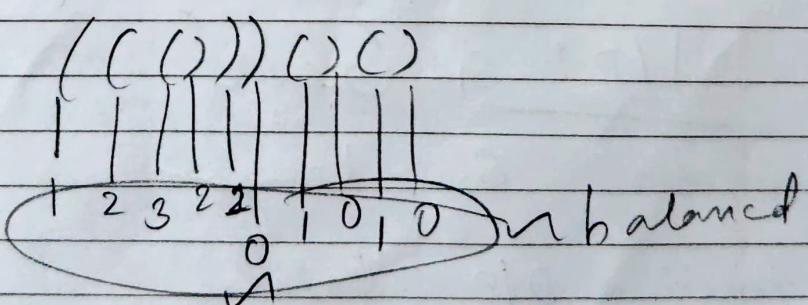


Time comp! => O(n * m)

Paranthesis Expression:

```
import java.util.*;
class Paranthesis{
    public static void main(String [] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("enter exp: ");
        String s = sc.next();
        int count = 0;
        String result = "Balanced";
        for (char ch : s.toCharArray()){
            if (ch == '('){
                count++;
            }
            else if (ch == ')'){
                count--;
                if (count < 0){
                    result = "not balanced";
                } } break;
        }
        if (count != 0){
            result = "not balanced";
        }
        System.out.println(result);
    }
}
```

I/P:



O/P:→ balanced. T.C => O(n)

14) Anagrams: T.C => O(n)log n)

```
import java.util.*;

Class Anagram{
    public static void main(String[] args){
        Scanner sc = new Scanner(
                                System.in);
        System.out.println("enter string 1:");
        String s1 = sc.next();
        print ("String 2");
        String s2 = sc.next();
        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArrays;
        if (s1.length() != s2.length()){
            print (false);
        }
        Arrays.sort(arr1);
        Arrays.sort(arr2);

        if (Arrays.equals(arr1,arr2)){
            print (true);
        }
        else{
            print (false);
        }
```

I/P: S1: Greek

S2: keeys

O/P: false.

T.C → $O(n^2)$

15, Palindrome Substring: I/P: Geeksforgeeks
O/P: True

```
Import java.util.*;
class Palindrome{
    public static void main (String[] args){
        Scanner sc = new Scanner
                    (System.in);
        System.out.println("enter string: ");
        String s = sc.next();
        int n = s.length();
int maxLen = 1;
start=0;
        if (n==0) System.out.println("");
        for(int i=0; i<n; i++){
            for(int j=i; j<n; j++){
                int left=i, right=j;
                boolean isPalindrome = true;
                while (left < right){
                    if (s.charAt(left) != s.charAt
                                (right)){
                        isPalindrome = false;
                        break;
                    }
                    left++; right--;
                if (isPalindrome && (j-i+1)>
                                maxLen){
                    start = i;
                    maxLen = j-i+1;
            }}}
        String result = s.substring
                    (start, start+maxLen);
        if (result.equals(s)){
            System.out.println(true);
        }else print(false);
    }}
```

T.C => O(n*m log n)

16) Longest Common Prefix using Substring :

I/P arr[] = ["geeks", "geeksforgeeks", "geek", "geezer"]

O/P => Gee

```
import java.util.*;

Class LongestPrefix{
    public static void main (String [] args){
        Scanner sc = new Scanner (System.in);
        print => no. of strings
        int n = sc.nextInt();
        Print => Strings :
        String [] arr = new String [n];
        for (int i=0 ; i<n; i++){
            arr [i] = sc.next();
        }
        if (arr.length == 0 || arr == null){
            print (-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}
            Print -1;
        else
            Print (first.substring(0,i));
    }
```

17. Middle element of Stack !

I/P : ⇒ [1,2,3,4,5] T.C => O(n)

O/P ⇒ [1, 2, 3, 4, 5]

```
import java.util.*;
class MiddleElement{
    public class
    Scanner SC = new Scanner(System.in);
    System.out.println("no. of elem in stack :");
    int n = SC.nextInt();
    Stack<Integer> S = new Stack<>();
    Print("elements: ");
    for(int i=0; i<n; i++){
        int elem = SC.nextInt();
    }   Stack.push(elem);
    int size = Stack.size();
    int count = 0;
    while(! Stack.isEmpty() &&
                count != size/2){
        int temp = Stack.pop();
    }   count++;
    if(! Stack.isEmpty()){
        Stack.pop();
    }
    System.out.println(stack);
}
}
```

Time complexity ⇒ O(n)

18) Next Greater Element; T.C => O(n)

```
import java.util.*;
class NextGreater{
    Public Static void main(String[]args){
        Scanner sc= new Scanner(System.in);
        print("no. of elements");
        int n= sc.nextInt();
        int[] arr= new int[n];
        print("elements:");
        for(int i=0; i<n; i++){
            arr[i]= sc.nextInt();
        }
        Stack<Integer> stack=
                new Stack<>();
to store result => int[] a= new int[n];
default val => -1 => Arrays.fill(a,-1)
        for(int i=0; i<n; i++){
            while(!stack.isEmpty() &
arr[i]>arr[stack.peek()]){
                a[stack.pop()]=arr[i];
            }
            stack.push(i);
        }
        for(int i=0; i<i; i++){
            print(arr[i]+"->"+a[i]);
        }
    }
}
```

I/P: 4 5 2 25

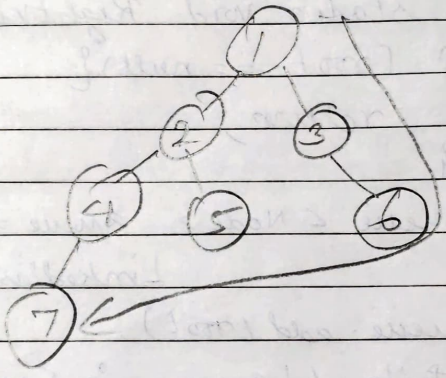
O/P:
4 → 5
5 → 25
2 → 25
25 → -1

19) Right Left view of the binary tree:

T.C => O(n)

```
import java.util.*;
class Node{
    int data;
    Node left, right;
    Node(int item){
        data = item;
        left = right = null;
    }
}

class RightLeftview {
    public static void RightView(Node root){
        if (root == null){
            return;
        }
        Queue<Node> queue = new
                        LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()){
            int levelSize = queue.size();
            for (int i=1; i<=levelsize; i++){
                Node node = queue.poll();
                if(i== levelsize){
                    print(node.data);
                }
                if (node.left != null){
                    queue.add(node.left);
                }
                if (node.right != null){
                    queue.add(node.right);
                }
            }
        }
    }
}
```

```
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.left.left = new node(7)
    root.left.right = new Node(5);
    root.right.right = (6);
    PrintRightView(root);
```



O/P: [1 3 6 7]

Time complexity => O(n)

20) Max Depth or Height of Binary Tree:

T.C => O(n log n)

```
import java.util.*;

Class BinaryTree {
    Static class Node {
        int data;
        Node left, right;
        Node(int item) {
            data = item;
            left = right = null;
        }
    }
    static int maxDepth(Node root) {
        if (root == null) {
            return 0;
        }

        int leftDepth = maxDepth
                        (root.left);
        n
        int rightDepth = maxDepth
                         (root.right);
        return Math.max(leftDepth,
                        rightDepth) + 1;
    }

    Public static Node insert
                    (Node root, int data) {
        if (root == NULL) {
            return new node(data);
        }
        if (data < root.data) {
            root.left = insert(root.
                               left, data);
        }
        else if (data > root.data) {
            root.right = insert(root.
                                right, data);
        }
```

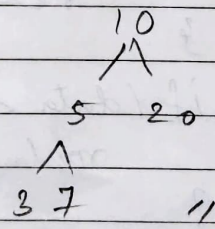
```
    }
    return root;

public static void main(
                String[] args) {
    Scanner sc = new Scanner
                (System.in);
    print("enter no. of nodes:");
    int n = sc.nextInt();
    Node root = null;
    print("enter the nodes:");
    for(int i=0; i<n; i++){
        int data = sc.nextInt();
        root = insert(root, data);
    }
    print(maxDepth(root));
}
}
```

I/P: enter no. of nodes: 5

enter the nodes:

10 5 20 3 7

Max: 3.



Time comp } = O(n)