

**Name : BHARATH V**

**Department: CSBS**

### **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**

**Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.**

**Input: arr[] = {-2, -4}**

**Output: -2**

**Explanation: The subarray {-2} has the largest sum -2.**

**Input: arr[] = {5, 4, 1, 7, 8}**

**Output: 25**

**Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.**

```
import java.util.*;

public class Sol1{

    public static void main(String[] args){

        int a=0;

        int b=Integer.MIN_VALUE;

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the no of elements");

        int n=sc.nextInt();

        int[] arr=new int[n];

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

            arr[i]=sc.nextInt();

        }

    }

}
```

```

System.out.println("Your original array is");
for(int i=0;i<n;i++){
System.out.println(arr[i]);
}
for(int i=0;i<n;i++){
a+=arr[i];
b=Math.max(a,b);
if(a<0){
a=0;
}
}
System.out.println("Maximum sum of the sub array is");
System.out.println(a);
}
} TIME COMPLEXITY : O(n)

```

```

C:\Users\Bharath\Downloads>javac Sol1.java
C:\Users\Bharath\Downloads>java Sol1
Enter the no of elements
5
5
4
1
7
8
Your original array is
5
4
1
7
8
Maximum sum of the sub array is
25
C:\Users\Bharath\Downloads>

```

## 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

**Explanation:** The subarray with maximum product is {6, -3, -10} with product =  $6 * (-3) * (-10) = 180$

**Input:** arr[] = {-1, -3, -10, 0, 60}

**Output:** 60

**Explanation:** The subarray with maximum product is {60}.

```
import java.util.*;

public class Sol2{

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the no of elements");

        int n=sc.nextInt();

        int[] arr=new int[n];

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

            arr[i]=sc.nextInt();

        }

        System.out.println("Your original array is");

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

            System.out.println(arr[i]);

        }

        int a=1;
```

```
int b=arr[0];
for(int i=1;i<n;i++){
a*=arr[i];
b=Math.max(a,b);
if(a==0){
a=1;
}
}
System.out.println("Product is");
System.out.println(b);
}
} TIME COMPLEXITY : O(n)
```

```
C:\Users\Bharath\Downloads>javac Sol2.java
C:\Users\Bharath\Downloads>java Sol2
Enter the no of elements
6
-2
6
-3
-10
0
2
Your original array is
-2
6
-3
-10
0
2
Product is
180
C:\Users\Bharath\Downloads>|
```

### 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

Input : `arr[] = { 4, 5, 6, 7, 0, 1, 2 }`, `key = 3`

Output : -1

Input : `arr[] = {50, 10, 20, 30, 40}`, `key = 10`

Output : 1

```
import java.util.*;

public class Sol3{

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the no of elements");

        int n=sc.nextInt();

        int[] arr=new int[n];

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

            arr[i]=sc.nextInt();

        }

        System.out.println("Your original array is");

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

            System.out.println(arr[i]);

        }

        System.out.println("Enter the key elements");

        int key=sc.nextInt();

        int start = 0;
```

```

int end = arr.length - 1;
while (start <= end) {
int mid = (start + end) / 2;
if (arr[mid] == key) {
System.out.print(mid);
break;
}
if (arr[start] <= arr[mid]) {
if (arr[start] <= key && key < arr[mid]) {
end = mid - 1;
} else {
start = mid + 1;
}
} else {
if (arr[mid] < key && key <= arr[end]) {
start = mid + 1;
} else {
end = mid - 1;
}
}
}
}
}
}
}

```

**TIME COMPLEXITY :  $O(\log n)$**

```
C:\Users\Bharath\Downloads>javac Sol3.java
```

```
C:\Users\Bharath\Downloads>java Sol3
```

```
Enter the no of elements
```

```
5
```

```
50
```

```
10
```

```
20
```

```
30
```

```
40
```

```
Your original array is
```

```
50
```

```
10
```

```
20
```

```
30
```

```
40
```

```
Enter the key elements
```

```
10
```

```
1
```

```
C:\Users\Bharath\Downloads>
```

#### 4. Container with Most Water

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

**Output:** 6

**Explanation:**

5 and 3 are distance 2 apart. So the size of the base = 2.

Height of container =  $\min(5, 3) = 3$ . So total area =  $3 * 2 = 6$

**Input:** arr = [3, 1, 2, 4, 5]

**Output:** 12

**Explanation:**

5 and 3 are distance 4 apart. So the size of the base = 4.

Height of container =  $\min(5, 3) = 3$ . So total area =  $4 * 3 = 12$

```
import java.util.*;

class Sol4{

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the no of elements");

        int n=sc.nextInt();

        int[] arr=new int[n];

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

            arr[i]=sc.nextInt();

        }

        System.out.println("Your original array is");

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

            System.out.println(arr[i]);

        }

        int start = 0;
```



```
int end = arr.length-1;
int maxarea = 0;
while(start<end){
int currarea = Math.min(arr[start], arr[end])*(end-start);
maxarea = Math.max(currarea,maxarea);
if(arr[start]<arr[end]){
start+=1;
}else{
end-=1;
}}
System.out.println(maxarea);
}} TIME COMPLEXITY : O(n)
```

```
C:\Users\Bharath\Downloads>javac Sol4.java
```

```
C:\Users\Bharath\Downloads>java Sol4
```

```
Enter the no of elements
```

```
4
```

```
1
```

```
5
```

```
4
```

```
3
```

```
Your original array is
```

```
1
```

```
5
```

```
4
```

```
3
```

```
6
```

```
C:\Users\Bharath\Downloads>
```

### 5. Find the Factorial of a large number

**Input: 100**

### Output:

**933262154439441526816992388562667004907159682643816214685929638  
952175999932299156089414639761565182862536979208272237582511852  
10916864000000000000000000000000**

**Input: 50**

**Output:**

304140932017133780436126081660647688443776415689605120000000000  
00

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

```
class Sol5{
```

```
static int mul(int x, int arr[], int size){
```

```
int carry = 0;
```

```
for(int i = 0;i<size;i++){
```

```
int prod = arr[i]*x + carry;
```

```
arr[i] = prod%10;
```

```
carry = prod/10;
```

}

```
while(carry !=0){
```

```
arr[size] = carry%10;
```

```
carry = carry/10;
```

```
size++;
```

}

```
return size;
```

}

```
static void factorial(int n){
```

```
int[] arr = new int[400];
```



**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

**Explanation:** The expected rainwater to be trapped is shown in the above image.

**Input:** `arr[] = {3, 0, 2, 0, 4}`

**Output:** 7

**Explanation:** We trap  $0 + 3 + 1 + 3 + 0 = 7$  units.

**Input:** `arr[] = {1, 2, 3, 4}`

**Output:** 0

**Explanation :** We cannot trap water as there is no height bound on both sides

**Input:** `arr[] = {10, 9, 0, 5}`

**Output:** 5

**Explanation :** We trap  $0 + 0 + 5 + 0 = 5$

```
import java.util.*;

public class Sol6{

    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the no of elements");
        int n=sc.nextInt();
        int[] arr=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.println("Your original array is");
```

```

for(int i=0;i<n;i++){
System.out.println(arr[i]);
}
int i = 0, j = n - 1, ans = 0;
while (i < j) {
if (arr[i] <= arr[j]) {
int tmp = arr[i];
while (tmp >= arr[i]) {
ans += tmp - arr[i++];
}
} else {
int tmp = arr[j];
while (tmp >= arr[j]) {
ans += tmp - arr[j--];
}}}
System.out.println(ans);
}} TIME COMPLEXITY: O(n)

```

```

C:\Users\Bharath\Downloads>javac Sol6.java
C:\Users\Bharath\Downloads>java Sol6
Enter the no of elements
7
3
0
1
0
4
0
2
Your original array is
3
0
1
0
4
0
2
10
C:\Users\Bharath\Downloads>|

```

## 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.

Input: `arr[] = {7, 3, 2, 4, 9, 12, 56}`,  $m = 3$

Output: 2

Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

Input: `arr[] = {7, 3, 2, 4, 9, 12, 56}`,  $m = 5$

Output: 7

Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is  $9 - 2 = 7$ .

```
import java.util.*;

class Sol7{

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the no of elements");

        int n=sc.nextInt();

        int[] arr=new int[n];

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

            arr[i]=sc.nextInt();

        }

        System.out.println("Your original array is");

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

```

System.out.println(arr[i]);
}
System.out.println("Enter the value of m");
int m=sc.nextInt();
int res = Integer.MAX_VALUE;
Arrays.sort(arr);
for(int i =0; i<arr.length-m; i++){
    res = Math.min(res, arr[i+m-1]-arr[i]);
}
System.out.println(res);
}} TIME COMPLEXITY: O(n)

```

```

C:\Users\Bharath\Downloads>javac Sol7.java
C:\Users\Bharath\Downloads>java Sol7
Enter the no of elements
7
7
3
2
4
9
12
56
Your original array is
7
3
2
4
9
12
56
Enter the value of m
3
2
C:\Users\Bharath\Downloads>

```

## 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.

**Input:**  $arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]$

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

**Explanation:** In the given intervals, we have only two overlapping intervals  $[1, 3]$  and  $[2, 4]$ . Therefore, we will merge these two and return  $[[1, 4], [6, 8], [9, 10]]$ .

**Input:**  $arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]]$

**Output:**  $[[1, 6], [7, 8]]$

**Explanation:** We will merge the overlapping intervals  $[[1, 5], [2, 4], [4, 6]]$  into a single interval  $[1, 6]$ .

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

```
public class Sol8{  
    public static void main(String[] args) {  
        int[][] mat = {{7, 8},{1, 5},{2, 4},{4, 6}};  
        Arrays.sort(mat, (a, b) -> Integer.compare(a[0], b[0]));  
        List<int[]> mergedIntervals = new ArrayList<>();  
        int start = mat[0][0];  
        int end = mat[0][1];  
        for (int i = 1; i < mat.length; i++) {  
            if (end >= mat[i][0]) {  
                end = Math.max(end, mat[i][1]);  
            } else {  
                mergedIntervals.add(new int[]{start, end});  
                start = mat[i][0];  
                end = mat[i][1];  
            }  
        }  
        mergedIntervals.add(new int[]{start, end});  
    }  
}
```



```

start = mat[i][0];
end = mat[i][1];
}}
mergedIntervals.add(new int[]{start, end});
int[][] result = new int[mergedIntervals.size()][2];
for (int i = 0; i < mergedIntervals.size(); i++) {
result[i] = mergedIntervals.get(i);
}
System.out.println("Merged intervals:");
for (int[] interval : result) {
System.out.println(Arrays.toString(interval));
}}} TIME COMPLEXITY:  $O(n^2)$ 

```

```

C:\Users\Bharath\Downloads>javac Sol8.java

C:\Users\Bharath\Downloads>java Sol8
Merged intervals:
[1, 6]
[7, 8]

C:\Users\Bharath\Downloads>|

```

## 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}}`

Input: `{{0, 0, 0},`

`{0, 0, 1}}`

Output: `{{0, 0, 1},`

`{1, 1, 1}}`

Input: `{{1, 0, 0, 1},`

`{0, 0, 1, 0},`

`{0, 0, 0, 0}}`

Output: `{{1, 1, 1, 1},`

`{1, 1, 1, 1},`

`{1, 0, 1, 1}}`

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

```
class Sol9{
```

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

```
int[][] matrix = {{0, 0, 0},
```

```
    {0, 0, 1}};
```

```
int row = matrix.length;
```

```
int col = matrix[0].length;
```

```
List<Integer> rowarr = new ArrayList<>();
```

```
List<Integer> colarr = new ArrayList<>();
```

```

for(int i=0; i<row; i++){
for(int j =0; j<col; j++){
if(matrix[i][j]==1){
rowarr.add(i);
colarr.add(j);
}}}
for(int i=0; i<rowarr.size(); i++){
for(int j=0; j<col; j++){
matrix[rowarr.get(i)][j] = 1;
}}
for(int i=0; i<colarr.size(); i++){
for(int j=0; j<row; j++){
matrix[j][colarr.get(i)] = 1;
}}

```

```

for(int[] i: matrix){
for(int j: i){
System.out.print(j+" ");
}
System.out.println();
}

```

**}} TIME COMPLEXITY:  $O(n^2)$**

```

C:\Users\Bharath\Downloads>javac Sol9.java
C:\Users\Bharath\Downloads>java Sol9
0 0 1
1 1 1
C:\Users\Bharath\Downloads>|

```

**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

**Input:** matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}}

**Output:** 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

**Explanation:** The output is matrix in spiral format.

**import java.util.\*;**

**public class Sol10{**

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

**spiral(new int[][]{{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}});**

**spiral(new int[][]{{1,2,3,4,5,6},{7,8,9,10,11,12},{13,14,15,16,17,18}});**

**}**

**public static void spiral(int[][] arr){**

**int top= 0;**

**int bottom = arr.length-1;**

**int left = 0;**

**int right = arr[0].length-1;**

**int size = arr.length\*arr[0].length;**

**List<Integer> l = new ArrayList<>();**

**int i= 0;**

**while(i!= size){**

**for(int n = left; n <= right && (i!= size); n++){**

**l.add(arr[top][n]);**

**i++;**

**}**

```

    for(int n = top; n < bottom && (i!= size); n++){
        l.add(arr[n+1][right]);

        i++;
    }

    for(int n = right-1; n >= left && (i!= size); n--){
        l.add(arr[bottom][n]);

        i++;
    }

    for(int n = bottom-1; n >= top+1 && (i!= size); n--){
        l.add(arr[n][left]);

        i++;
    }

    top++;
    bottom--;
    left++;
    right--;
}

System.out.println(l);
}

```

**} TIME COMPLEXITY:  $O(n^2)$**

```

C:\Users\Bharath\Downloads>javac Sol10.java

C:\Users\Bharath\Downloads>java Sol10
[1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]
[1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11]

C:\Users\Bharath\Downloads>|

```

### 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.

Input: str = “((()))()”

Output: Balanced

Input: str = “()((())”

Output: Not Balanced

```
import java.util.*;

class Sol13{

    public static void main(String[] args){

        String st = "((()))()";

        int c =0;

        for(int i=0; i<st.length(); i++){

            if(st.charAt(i)=='('){

                c+=1;

            }else{

                c-=1;

            }

            if(c<0){

                System.out.println("Not Balanced");

            }

            System.out.println("Balanced");

        }

    }

}

TIME COMPLEXITY: O(n)
```

```
C:\Users\Bharath\Downloads>javac Sol13.java

C:\Users\Bharath\Downloads>java Sol13
Balanced

C:\Users\Bharath\Downloads>|
```

#### 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

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = "allergy" s2 = "allergic"

Output: false

Explanation: Characters in both the strings are not same. s1 has extra character „y" and s2 has extra characters „i" and „c", so they are not anagrams.

Input: s1 = "g", s2 = "g"

Output: true

Explanation: Characters in both the strings are same, so they are anagrams.

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

```
class Sol14{  
    public static void main(String[] args) {  
        String st1 = "allergy";  
        String st2 = "allergic";  
        char[] st1array = st1.toCharArray();  
        char[] st2array = st2.toCharArray();  
        Arrays.sort(st1array);
```

```
Arrays.sort(st2array);  
if (Arrays.equals(st1array, st2array)) {  
System.out.println(true);  
} else {  
System.out.println(false);  
}}}  
TIME COMPLEXITY: O(n)
```

```
C:\Users\Bharath\Downloads>javac Sol14.java  
  
C:\Users\Bharath\Downloads>java Sol14  
false  
  
C:\Users\Bharath\Downloads>|
```



## 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"`

Explanation: There are several possible palindromic substrings like `"kssk"`, `"ss"`, `"eeksske"` etc. But the substring `"geeksskeeg"` is the longest among all.

Input: `str = "Geeks"`

Output: `"ee"`

Input: `str = "abc"`

Output: `"a"`

Input: `str = ""`

Output: `""`

```
import java.util.*;

class Sol15{

    public static void main(String[] args) {

        String s = "Geeks";

        String res = "";

        int resLen = 0;

        for (int i =0; i<s.length(); i++) {

            int l = i, r = i;

            while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {

                if ((r-l+1)> resLen) {
```

```

res = s.substring(l, r + 1);
resLen=r-l+ 1;
}
l--;
r++;
}
l=i;
r=i+1;
while(l>=0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
if ((r - l + 1) > resLen) {
res = s.substring(l, r + 1);
resLen=r-l+1;
}
l--;
r++;
}}
System.out.println("Longest Palindromic Substring: " + res);
}} TIME COMPLEXITY: O(n)

```

```

C:\Users\Bharath\Downloads>javac Sol15.java

C:\Users\Bharath\Downloads>java Sol15
Longest Palindromic Substring: ee

C:\Users\Bharath\Downloads>|

```

## 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`

Explanation: `"gee"` is the longest common prefix in all the given strings.

Input: `arr[] = ["hello", "world"]`

Output: `-1`

Explanation: There's no common prefix in the given strings.

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

```
public class Sol16{
```

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

```
        prefix(new String[]{"geeksforgeeks", "geeks", "geek", "geezer"});
```

```
        prefix(new String[]{"hello", "world"});
```

```
    }
```

```
    public static void prefix(String[] arr){
```

```
        System.out.print(Arrays.toString(arr) + " : ");
```

```
        int l = arr[0].length();
```

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

```
            l = Math.min(l , arr[i].length());
```

```
            for(int j = 0 ; j<l; j++){
```

```
                char c= arr[0].charAt(j);
```

```
                char d = arr[i].charAt(j);
```

```
                if(c!=d){
```

```

        l = j;
        break;
    }
}
}
String ans = arr[0].substring(0,l);
System.out.println(ans.length() > 0 ? ans : -1 );
}
} TIME COMPLEXITY: O(n)

```

```

C:\Users\Bharath\Downloads>javac Sol16.java

C:\Users\Bharath\Downloads>java Sol16
[geeksforgeeks, geeks, geek, geezer] : gee
[hello, world] : -1

C:\Users\Bharath\Downloads>|

```

## 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]

**Input :** Stack[] = [1, 2, 3, 4, 5, 6]

**Output :** Stack[] = [1, 2, 4, 5, 6]

```
import java.util.*;

public class Sol17{

    public static void main(String[] args) {
        Stack<Integer> st = new Stack<>();
        st.push(1);
        st.push(2);
        st.push(3);
        st.push(4);
        st.push(5);
        st.push(6);
        int n = st.size()/2;
        Stack<Integer> temp = new Stack<>();
        for(int i=0; i<n; i++){
            temp.push(st.pop());
        }
        st.pop();
        while(!temp.isEmpty()){
            st.push(temp.pop());
        }
    }
}
```

```
}  
for(int i: st){  
System.out.println(i);  
}}
```

**TIME COMPLEXITY:  $O(n)$**

```
C:\Users\Bharath\Downloads>javac Sol17.java  
C:\Users\Bharath\Downloads>java Sol17  
1  
2  
4  
5  
6  
C:\Users\Bharath\Downloads>|
```

## 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

**Explanation:** Except 25 every element has an element greater than them present on the right side

**Input:** arr[] = [ 13 , 7, 6 , 12 ]

**Output:** 13 → -1

7 → 12

6 → 12

12 → -1

**Explanation:** 13 and 12 don't have any element greater than them present on the right side

```
import java.util.*;

class Sol18{

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the no of elements");

        int n=sc.nextInt();

        int[] arr=new int[n];
```

```

for(int i=0;i<n;i++){
arr[i]=sc.nextInt();
}
System.out.println("Your original array is");
for(int i=0;i<n;i++){
System.out.println(arr[i]);
}
for (int i = 0; i < arr.length; i++) {
int greatest = -1;
for (int j = i + 1; j < arr.length; j++) {
if (arr[j] > arr[i]) {
greatest = arr[j];
break;
}}
System.out.println(arr[i] + " -> " + greatest);
}}} TIME COMPLEXITY:  $O(n^2)$ 

```

```

C:\Users\Bharath\Downloads>javac Sol18.java
C:\Users\Bharath\Downloads>java Sol18
Enter the no of elements
4
4
5
2
25
Your original array is
4
5
2
25
4 -> 5
5 -> 25
2 -> 25
25 -> -1
C:\Users\Bharath\Downloads>|

```



## 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.*;

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

class Sol19{
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        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.right.right = new TreeNode(5);
        root.right.left = new TreeNode(4);
        Sol19 solution = new Sol19();
        List<Integer> result = solution.rightSideView(root);
        System.out.println(result);
    } TIME COMPLEXITY:  $O(n \log n)$ 

```

```

C:\Users\Bharath\Downloads>javac Sol19.java

C:\Users\Bharath\Downloads>java Sol19
[1, 3, 5]

C:\Users\Bharath\Downloads>|

```

## 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.**

```
import java.util.*;

class TreeNode {

    int val;

    TreeNode left;

    TreeNode right;

    TreeNode(int val) {

        this.val = val;

        this.left = null;

        this.right = null;

    }

}

class Sol20{

    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);  
  
    Sol20 main = new Sol20();  
    int depth = main.maxDepth(root);  
    System.out.println("Maximum Depth of the Tree: " + depth);  
}  
} TIME COMPLEXITY:  $O(n \log n)$ 
```

```
C:\Users\Bharath\Downloads>javac Sol20.java  
  
C:\Users\Bharath\Downloads>java Sol20  
Maximum Depth of the Tree: 3  
  
C:\Users\Bharath\Downloads>|
```