

Vasanthan_T_DSA_Practice-1

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

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.

Program:

```
import java.util.*;

class Solution {
    public static int maxSubArray(int[] nums) {
        int sum=0;
        int max=Integer.MIN_VALUE;
        for(int i=0;i<nums.length;i++){
            if(sum<0){
                sum=0;
            }
            sum+=nums[i];
            max=Math.max(sum,max);
        }
        return max;
    }
    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();
        }
        System.out.print(maxSubArray(arr));
    }
}
```

Output:

```
C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
7
2 3 -8 7 -1 2 3
11
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

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}

Program:

```
import java.util.*;
class Solution {
    public static int maxProduct(int[] nums) {
        int sufpro=1;
        int prepro=1;
        int ind=nums.length-1;
        int max1=Integer.MIN_VALUE;
        int max2=Integer.MIN_VALUE;
        for(int i=0;i<nums.length;i++){
            prepro*=nums[i];
            max1=Math.max(prepro,max1);
            if(prepro==0){
                prepro=1;
            }
            sufpro*=nums[ind--];
            max2=Math.max(sufpro,max2);
            if(sufpro==0){
                sufpro=1;
            }
        }
        return Math.max(max1,max2);
    }
    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();
}
System.out.print(maxProduct(arr));
}
}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
6
-2 6 -3 -10 0 2
180

```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

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

Program:

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

```

class Solution {
    public static int search(int[] nums, int target) {
        int l = 0;
        int r = nums.length - 1;

        while (l <= r) {
            int mid = (l + r) / 2;
            if (nums[mid] == target) {

```

```

        return mid;
    }
    if (nums[mid] >= nums[l]) {
        if (nums[mid] > target && target >= nums[l]) {
            r = mid - 1;
        } else {
            l = mid + 1;
        }
    } else {
        if (nums[mid] < target && target <= nums[r]) {
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
}
return -1;
}

public static void main (String[] args) {
    Scanner sc=new Scanner(System.in);
    int n=sc.nextInt();
    int t=sc.nextInt();
    int[] arr=new int[n];
    for(int i=0;i<n;i++){
        arr[i]=sc.nextInt();
    }
    System.out.print(search(arr,t));
}
}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
7
0
4 5 6 7 0 1 2
4

```

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

Space Complexity: $O(1)$

4. Container with Most Water

Given n non-negative integers a_1, a_2, \dots, a_n where each represents a point at coordinate (i, a_i) . ' n ' vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is the 2D plane we are working with for simplicity).

Note: You may not slant the container.

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 = $4 * 3 = 12$.

Program:

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

```
class Solution {
    public static int mostWater(int[] nums) {
        int l=0;
        int r=nums.length-1;
        int max=0;
        while(l<r){
            int area=1;
            if(nums[l]<nums[r]){
                area=nums[l]*(r-l);
                l++;
            }else{
                area=nums[r]*(r-l);
                r--;
            }
            max=Math.max(max, area);
        }
        return max;
    }
}
```

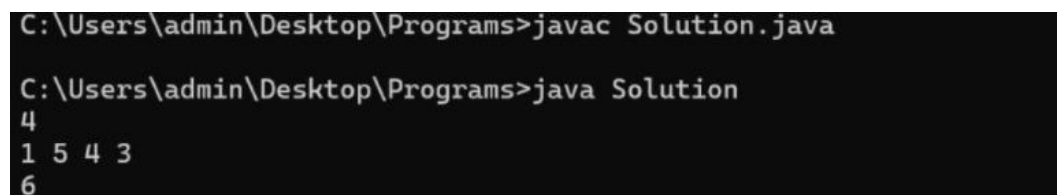
```

    }
    max=Math.max(max,area);

    }
    return max;
}
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();
    }
    System.out.print(mostWater(arr));
}
}

```

Output:



```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
4
1 5 4 3
6

```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

5. Find the Factorial of a large number

Input: 100

Output:

```

93326215443944152681699238856266700490715968264381621468592963895217599993
2299
156089414639761565182862536979208272237582511852109168640000000000000000
0000 00

```

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

Program:

```

import java.util.*;
import java.math.BigInteger;

```

```

class Solution {

```

```
public static BigInteger fun(int n) {
    BigInteger res=BigInteger.ONE;
    for(int i=1;i<=n;i++){
        res = res.multiply(BigInteger.valueOf(i));
    }
    return res;
}

public static void main(String[] args){
    Scanner sc=new Scanner(System.in);
    int n=sc.nextInt();
    BigInteger res=fun(n);
    System.out.print(res);
}
}
```

Output:

```
C:\Users\admin\Desktop\Programs>javac Solution.java  
C:\Users\admin\Desktop\Programs>java Solution  
100  
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582  
51185210916864000000000000000000000000
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

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 $0 + 0 + 5 + 0 = 5$

Program:

```

import java.util.*;

class Solution {
    public static int trap(int[] height) {
        int n = height.length;
        int[] prefix = new int[n];
        int[] suffix = new int[n];

        prefix[0] = height[0];
        for (int i = 1; i < n; i++) {
            prefix[i] = Math.max(height[i], prefix[i - 1]);
        }

        suffix[n - 1] = height[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            suffix[i] = Math.max(height[i], suffix[i + 1]);
        }

        int sum = 0;
        for (int i = 1; i < n - 1; i++) {
            sum += Math.min(prefix[i], suffix[i]) - height[i];
        }
        return sum;
    }

    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();
        }
        System.out.print(trap(arr));
    }
}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
7
3 0 1 0 4 0 2
10

```

Time Complexity: $O(n)$

Space 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.

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 $9 - 2 = 7$.

Program:

```
import java.util.*;

class Solution {
    public static int distribute(int[] arr, int m) {
        Arrays.sort(arr);
        if (arr.length < m) return -1;
        int min = Integer.MAX_VALUE;
        for (int i = m - 1; i < arr.length; i++) {
            min = Math.min(arr[i] - arr[i - m + 1], min);
        }
        return min;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print(distribute(arr, m));
    }
}
```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
7
3
7 3 2 4 9 12 56
2

```

Time Complexity: $O(n \log n)$ (due to sorting)

Space Complexity: $O(1)$

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 $[1, 6]$.

Program:

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

```
class Solution {
```

```
    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 s = arr[i][0];
            int e = arr[i][1];
```

```
            if (!res.isEmpty() && res.get(res.size() - 1)[1] >= e) {
                continue;
            }
```

```
            for (int j = i + 1; j < n; j++) {
                if (arr[j][0] <= e) {
                    e = Math.max(e, arr[j][1]);
                }
```

```

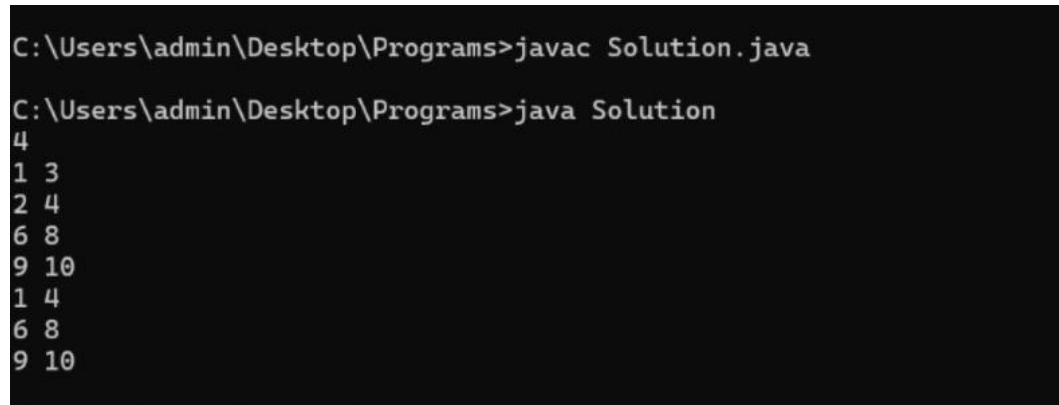
        }
    }
    res.add(new int[]{s, e});
}
return res;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int[][] arr = new int[n][2];
    for (int i = 0; i < n; i++) {
        arr[i][0] = sc.nextInt();
        arr[i][1] = sc.nextInt();
    }
    List<int[]> res = mergeOverlap(arr);

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

```

Output:



```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
4
1 3
2 4
6 8
9 10
1 4
6 8
9 10

```

Time Complexity: $O(n \log n)$ (due to sorting)

Space 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.
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}}

Program:

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

```
class Solution {
    static void setOnes(int[][] mat) {
        int r = mat.length;
        int c = mat[0].length;

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                if (mat[i][j] == 1) {
                    int k = i - 1;
                    while (k >= 0) {
                        if (mat[k][j] != 1) mat[k][j] = -1;
                        k--;
                    }
                    k = i + 1;
                    while (k < r) {
                        if (mat[k][j] != 1) mat[k][j] = -1;
                        k++;
                    }
                    k = j - 1;
                    while (k >= 0) {
                        if (mat[i][k] != 1) mat[i][k] = -1;
                        k--;
                    }
                    k = j + 1;
                    while (k < c) {
                        if (mat[i][k] != 1) mat[i][k] = -1;
                        k++;
                    }
                }
            }
        }

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                if (mat[i][j] < 0) mat[i][j] = 1;
            }
        }
    }
}
```

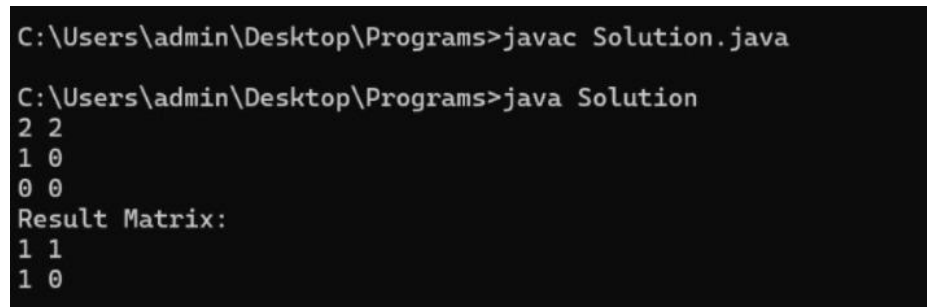
```

    }
}

public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    int n=sc.nextInt();
    int m=sc.nextInt();
    int[][] arr=new int[n][m];
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            arr[i][j]=sc.nextInt();
        }
    }
    setOnes(arr);
    System.out.println("Result Matrix:");
    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();
    }
}
}

```

Output:



```

C:\Users\admin\Desktop\Programs>javac Solution.java

C:\Users\admin\Desktop\Programs>java Solution
2 2
1 0
0 0
Result Matrix:
1 1
1 0

```

Time Complexity: $O(r \times c)$

Space Complexity: $O(1)$

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.

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.

Program:

```
import java.util.*;
class Solution {
    public static List<Integer> spiralOrder(int[][] matrix) {
        int n = matrix.length;
        int m = matrix[0].length;
        int right = m - 1;
        int left = 0;
        int top = 0;
        int bottom = n - 1;
        List<Integer> res = new ArrayList<>();
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) {
                res.add(matrix[top][i]);
            }
            top++;
            for (int i = top; i <= bottom; i++) {
                res.add(matrix[i][right]);
            }
            right--;
            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
                    res.add(matrix[bottom][i]);
                }
                bottom--;
            }
            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    res.add(matrix[i][left]);
                }
                left++;
            }
        }
        return res;
    }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int m=sc.nextInt();
        int[][] arr=new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
```

```

        arr[i][j]=sc.nextInt();
    }
}
System.out.print(spiralOrder(arr));
}
}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java

C:\Users\admin\Desktop\Programs>java Solution
4 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
[1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]

```

Time Complexity: $O(r \times c)$

Space Complexity: $O(r \times c)$

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

Program:

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

```

public class Solution {

    public static boolean ispar(String str) {

        Stack<Character> st = new Stack<>();

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

```

```

        if (str.charAt(i) == '(' || str.charAt(i) == '{' || str.charAt(i) == '[') {
            st.push(str.charAt(i));
        } else {
            if (!st.empty() &&
                ((st.peek() == '(' && str.charAt(i) == ')') ||
                 (st.peek() == '{' && str.charAt(i) == '}') ||
                 (st.peek() == '[' && str.charAt(i) == ']'))) {
                st.pop();
            } else {
                return false;
            }
        }
    }
    return st.empty();
}

```

```

public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    String str=sc.next();
    System.out.println(ispar(str) ? "Balanced" : "Unbalanced");
}
}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
"((()))()()"
Unbalanced

```


Time Complexity: $O(n)$

Space 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

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.

Program:

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

```
class Solution {
```

```
    static boolean check(String str1, String str2) {
```

```
        char[] arr1 = str1.toCharArray();
```

```
        char[] arr2 = str2.toCharArray();
```

```
        Arrays.sort(arr1);
```

```
        Arrays.sort(arr2);
```

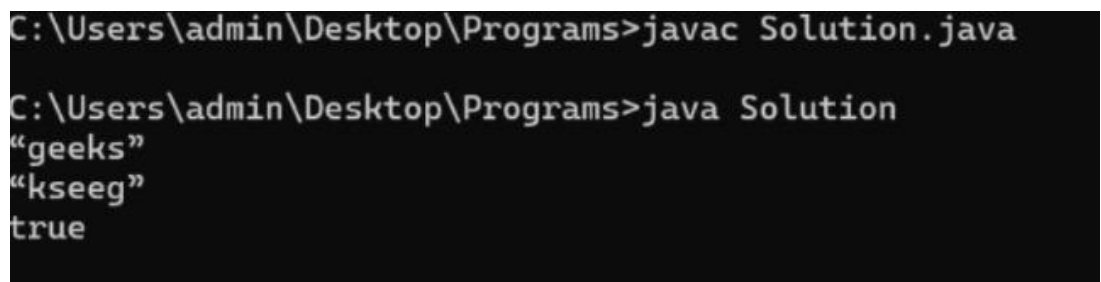
```
        return Arrays.equals(arr1, arr2);
```

```
}
```

```
public static void main(String[] args) {  
    Scanner sc=new Scanner(System.in);  
    String str1 = sc.next();  
    String str2 = sc.next();  
    System.out.println(check(str1, str2));  
}
```

```
}
```

Output:



```
C:\Users\admin\Desktop\Programs>javac Solution.java  
C:\Users\admin\Desktop\Programs>java Solution  
"geeks"  
"kseeg"  
true
```

Time Complexity: $O(n \log n)$ (due to sorting)

Space 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.

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: ""

Program:

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

```
public class Solution {
```

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

```
        Scanner sc = new Scanner(System.in);
```

```
        String s = sc.next();
```

```
        System.out.println(longSub(s));
```

```
    }
```

```
    static boolean check(String s, int l, int h) {
```

```
        while (l < h) {
```

```
            if (s.charAt(l) != s.charAt(h))
```

```
                return false;
```

```
            l++;
```

```
            h--;
```

```
        }
```

```
        return true;
```

```
    }
```

```
    static String longSub(String s) {
```

```
        int len = s.length();
```

```
        int maxL = 1, st = 0;
```

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

```
            for (int j = i; j < len; j++) {
```

```
                if (check(s, i, j) && (j - i + 1) > maxL) {
```

```

        st = i;

        maxL = j - i + 1;

    }

}

}

return s.substring(st, st + maxL);

}

}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java

C:\Users\admin\Desktop\Programs>java Solution
"forgeeksskeegfor"
geeksskeeg

```

Time Complexity: $O(n^3)$

Space Complexity: $O(1)$

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.

Program:

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

```
class Solution {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        String[] arr = new String[n];

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

            arr[i] = sc.next();

        }

        sc.close();

        System.out.println(fun(arr));

    }

    static String fun(String[] arr) {

        if (arr == null || arr.length == 0)

            return "-1";

        Arrays.sort(arr);

        String f = arr[0];

        String l = arr[arr.length - 1];

        int m = Math.min(f.length(), l.length());

        int i = 0;

        while (i < m && f.charAt(i) == l.charAt(i)) {

            i++;

        }

    }

}
```

```

        if (i == 0)

            return "-1";

        return f.substring(0, i);

    }

}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
4
"geeksforgeeks"
"geeks"
"geek"
"geezer"
"gee

```

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

Space Complexity: $O(1)$

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]

Program:

```

import java.util.*;

class JavaSample {

    public static void fun(Stack<Integer> st, int indx , int sz) {

        if (st.isEmpty()) return;

        if (indx == sz / 2) {

```

```

        st.pop();

        return;
    }

    int temp = st.pop();

    fun(st, indx + 1,sz);

    st.push(temp);
}

public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    Stack<Integer> st = new Stack<>();

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

        st.push(sc.nextInt());

    }

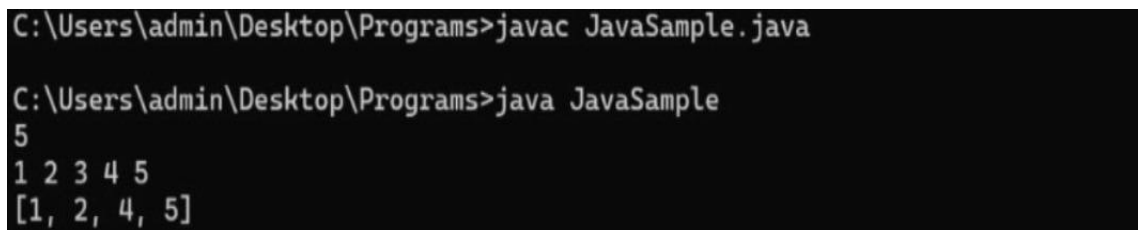
    fun(st, 0 , n);

    System.out.println(st);

}
}

```

Output:



```

C:\Users\admin\Desktop\Programs>javac JavaSample.java

C:\Users\admin\Desktop\Programs>java JavaSample
5
1 2 3 4 5
[1, 2, 4, 5]

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$ (recursive stack space)

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

Program:

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

```
class Solution {
```

```
    static void nextGreat(int[] arr, int n) {
```

```
        int nxt;
```

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

```
            nxt = -1;
```

```
            for (int j = i + 1; j < n; j++) {
```

```
                if (arr[i] < arr[j]) {
```

```
                    nxt = arr[j];
```

```
                    break;
```

```
                }
```

```
            }
```

```
            System.out.println(arr[i] + " -- " + nxt);
```

```
        }
```

```
    }
```

```
    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();

}

nextGreat(arr, n);

}

```

Output:

```

C:\Users\admin\Desktop\Programs>javac Solution.java

C:\Users\admin\Desktop\Programs>java Solution
4
4 5 2 25
4 -- 5
5 -- 25
2 -- 25
25 -- -1

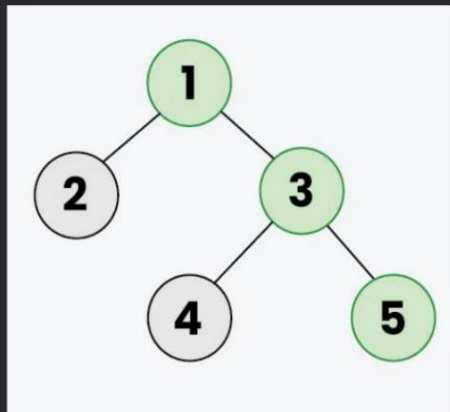
```

Time Complexity: $O(n^2)$

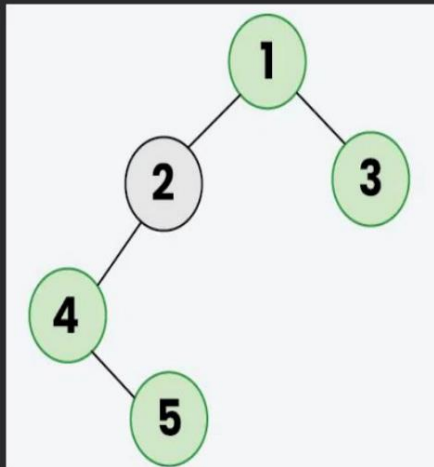
Space Complexity: $O(1)$

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.

*Example 1: The **Green** colored nodes (1, 3, 5) represents the Right view in the below Binary tree.*



Example 2: The **Green** colored nodes (1, 3, 4, 5) represents the *Right view* in the below Binary tree.



Program:

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

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

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

```

Node n = new Node(1);

n.left = new Node(2);

n.right = new Node(3);

n.right.left = new Node(4);

n.right.right = new Node(5);


ArrayList<Integer> res = rightView(n);


printArr(res);
}


static void right(Node n, int lvl, int[] maxL, ArrayList<Integer> res) {

    if (n == null) return;


    if (lvl > maxL[0]) {

        res.add(n.data);

        maxL[0] = lvl;

    }


    right(n.right, lvl + 1, maxL, res);

    right(n.left, lvl + 1, maxL, res);

}


static ArrayList<Integer> rightView(Node n) {

    ArrayList<Integer> res = new ArrayList<>();

    int[] maxL = new int[] {-1};

```

```

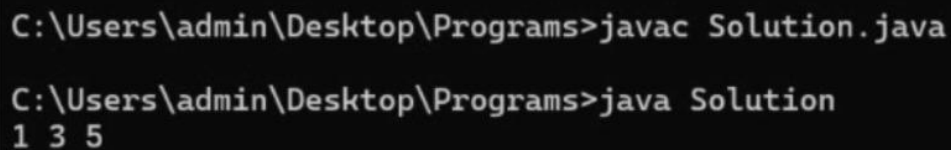
        right(n, 0, maxL, res);

    return res;
}

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

```

Output:



```

C:\Users\admin\Desktop\Programs>javac Solution.java
C:\Users\admin\Desktop\Programs>java Solution
1 3 5

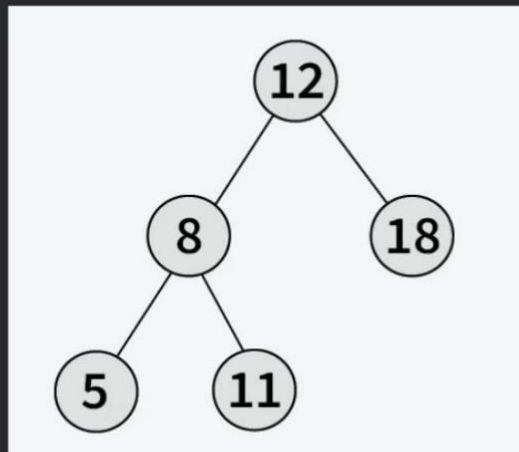
```

Time Complexity: $O(n)$

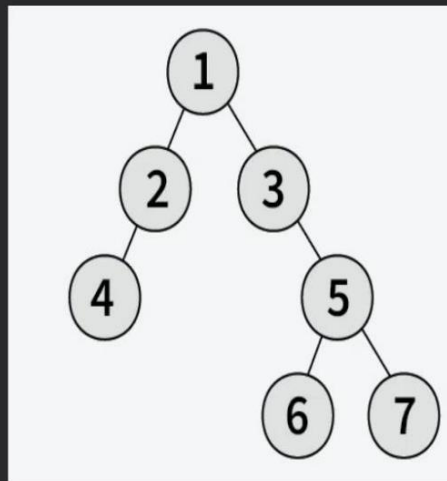
Space Complexity: $O(h)$ (h- height of the tree)

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.

Example 1: The height of the below binary tree is 3.



Example 2: The height of the below binary tree is 4



Program:

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

```
class Node {
```

```
    int data;
```

```
    Node left, right;
```

```
    Node(int val) {
```

```
        data = val;
```

```
        left = right = null;
    }
}

class Solution {
    static int depth(Node n) {
        if (n == null) return 0;

        int l = depth(n.left);
        int r = depth(n.right);

        return Math.max(l, r) + 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(depth(root));
}
}
```

Output:

```
C:\Users\admin\Desktop\Programs>javac Solution.java  
C:\Users\admin\Desktop\Programs>java Solution  
3
```

Time Complexity: $O(n)$

Space Complexity: $O(h)$