

RANJIT H_IT_DAY-1 PRACTICE

QUESTION 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.

Solution:

CODE:

```
package JavaPractice;

import java.util.*;

public class Array {

    public static int maxSubarraySum(int[] arr) {
        int result=arr[0];
        int maxSum=arr[0];
        for(int i=1;i<arr.length;i++) {
            maxSum=Math.max(arr[i],maxSum+arr[i]);
            result=Math.max(result,maxSum);
        }
        return result;
    }

    public static void main(String[] args) {
        int[] arr= {2, 3, -8, 7, -1, 2, 3};
        System.out.println(maxSubarraySum(arr));
        int[] arr1={-2, -4};
        System.out.println(maxSubarraySum(arr1));
    }
}
```

```
1 package MaximumSubarray;
2 //import java.util.*;
3 public class MaxArray {
4     static int maxSubarraySum(int[] arr) {
5         int result = arr[0];
6         int maxSum = arr[0];
7
8         for (int i = 1; i < arr.length; i++) {
9             maxSum = Math.max(maxSum + arr[i], arr[i]);
10            result = Math.max(result, maxSum);
11        }
12        return result;
13    }
14
15    public static void main(String[] args) {
16        int[] arr = {2, 3, -8, 7, -1, 2, 3};
17        System.out.println(maxSubarraySum(arr));
18        int[] arr2 = {-2, -4};
19        System.out.println(maxSubarraySum(arr2));
20    }
21 }
```

Problems Javadoc Declaration Console ×

<terminated> MaxArray [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (09-Nov-2024, 9:07:5

11
-2

TIME COMPLEXITY: $O(N)$

QUESTION2: 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}

SOLUTION:

```
class Solution {
    public int maxProduct(int[] arr) {
        int n = arr.length;
        int currMax = arr[0];
        int currMin = arr[0];
```

```

int maxProd = arr[0];
for (int i = 1; i < n; i++) {
    int temp = max(arr[i], arr[i] * currMax, arr[i] * currMin);
    currMin = min(arr[i], arr[i] * currMax, arr[i] * currMin);
    currMax = temp;
    maxProd = Math.max(maxProd, currMax);
}
return maxProd;
}

static int max(int a, int b, int c) {
    return Math.max(a, Math.max(b, c));
}

static int min(int a, int b, int c) {
    return Math.min(a, Math.min(b, c));
}

public static void main(String[] args) {
    int[] arr= {-2, 6, -3, -10, 0, 2};
    System.out.println(maxProduct(arr));
    int[] arr1={-1, -3,-10,0,60};
    System.out.println(maxProduct(arr1));
}
}

```

```
1 package MaximumSubarray;
2 //import java.util.*;
3 public class MaxArray {
4     public static int maxProduct(int[] arr) {
5         int n = arr.length;
6         int currMax = arr[0];
7         int currMin = arr[0];
8         int maxProd = arr[0];
9         for (int i = 1; i < n; i++) {
10             int temp = max(arr[i], arr[i] * currMax, arr[i] * currMin);
11             currMin = min(arr[i], arr[i] * currMax, arr[i] * currMin);
12             currMax = temp;
13             maxProd = Math.max(maxProd, currMax);
14         }
15
16         return maxProd;
17     }
18     static int max(int a, int b, int c) {
19         return Math.max(a, Math.max(b, c));
20     }
21
22     static int min(int a, int b, int c) {
23         return Math.min(a, Math.min(b, c));
24     }
25     public static void main(String[] args) {
26         int[] arr = {-2, 6, -3, -10, 0, 2};
27         System.out.println(maxProduct(arr));
28         int[] arr2 = {-1, -3, -10, 0, 60};
29         System.out.println(maxProduct(arr2));
30     }
31 }
```

<

Problems Javadoc Declaration Console ×

<terminated> MaxArray [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (09-Nov-2024, 10:18:00)

180
60

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.

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

Solution:

CODE:

```
package MaximumSubarray;

//import java.util.*;

public class MaxArray {

    static int search(int[] nums, int target) {
        int high=nums.length-1;
        int low=0;
        while(high>=low){
            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 k=0;  
    System.out.println(search(arr,k));  
    int[] arr2= {4, 5, 6, 7, 0, 1, 2};  
    int k2=3;  
    System.out.println(search(arr2,k2));  
    int[] arr3= {50,10,20,30,40};  
    int k3=10;  
    System.out.println(search(arr3,k3));  
}  
}
```

```
1 package MaximumSubarray;
2 //import java.util.*;
3 public class MaxArray {
4     static int search(int[] nums, int target) {
5         int high=nums.length-1;
6         int low=0;
7         while(high>=low){
8             int mid=(low+high)/2;
9             if(nums[mid]==target){
10                 return mid;
11             }
12             if(nums[low]<=nums[mid]){
13                 if(nums[low]<=target && target<nums[mid]){
14                     high=mid-1;
15                 }else{
16                     low=mid+1;
17                 }
18             }
19             else{
20                 if(nums[mid]<target && target<=nums[high]){
21                     low=mid+1;
22                 }
23                 else{
24                     high=mid-1;
25                 }
26             }
27         }
28         return -1;
29     }
30     public static void main(String[] args) {
31         int[] arr = {4, 5, 6, 7, 0, 1, 2} ;
32         int k=0;
33         System.out.println(search(arr,k));
34         int[] arr2= {4, 5, 6, 7, 0, 1, 2};
35         int k2=3;
36         System.out.println(search(arr2,k2));
37     }
38 }
```

Problems Javadoc Declaration Console X

<terminated> MaxArray [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe

4
-1
1

TIMECOMPLEXITY: $O(\log N)$

4. Container with Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i^{th} line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store.*

Notice that you may not slant the container.

CODE:

```
class maxArea {  
    public static int maxArea(int[] height) {
```

```

int ans=0;
int left=0;
int right=height.length-1;
while(right>left){
    int hei=Math.min(height[right],height[left]);
    int bre=right-left;
    int area=hei*bre;
    ans=Math.max(area,ans);
    if(height[right]>height[left]){
        left++;
    }else{
        right--;
    }
}
return ans;
}

public static void main(String[] args) {
    int[] arr ={1,5,4,3} ;
    System.out.println(maxArea(arr));
    int[] arr2= {3,1,2,4,5};
    System.out.println(maxArea(arr2));
}
}
}

```

OUTPUT:

```

D:\javaprograms\Practice ques\4>javac maxArea.java
D:\javaprograms\Practice ques\4>java maxArea
6
12

```

Time Complexity:O(N)

5. Factorial

Find the Factorial of a large number

Input: 100

Output:

```
9332621544394415268169923885626670049071596826438162146859296389521
75999932299
1560894146397615651828625369792082722375825118521091686400000000000
00000000000 00 Input: 50
```

Output:

30414093201713378043612608166064768844377641568960512000000000000

```
import java.math.BigInteger;
```

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

```
public class Fact {
```

```
public static BigInteger Fact(int n) {
```

```
BigInteger ans=BigInteger.valueOf(1);
```

```
for(int i=2;i<=n;i++) {
```

```
ans=ans.multiply(BigInteger.valueOf(i));
```

}

```
return ans;
```

}

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

```
System.out.println(Fact(50));
```

```
System.out.println(Fact(100));
```

}

}

OUTPUT: TIME COMPLEXITY: $O(N)$

```
D:\javaprograms\Practice ques\5>javac Fact.java
```

```
D:\javaprograms\Practice ques\5>java Fact
```

30414093201713378043612608166064768844377641568960512000000000000

03326215443944157681699238856266700490715968764381621468592963895217599993229915608941463976156518286753697920872723758251185210916864000000000000000000000000

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

Code:

```
class Trap {
    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){
        int [] arr = {3, 0, 1, 0, 4, 0, 2};
        System.out.println(trap(arr));
        int [] arr2= {3, 0, 2, 0, 4};
```

```

        System.out.println(trap(arr2));

        int [] arr3={1,2,3,4};

        System.out.println(trap(arr3));
    }
}

```

OUTPUT:

```

D:\javaprograms\Practice ques\6>javac Trap.java
D:\javaprograms\Practice ques\6>java Trap
10
7
0

```

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.\

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. |

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

Output: 7 E

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

CODE:

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

```
class Chocolate {
```

```
    public static int findMinDifference(int[] arr, int m) {
```

```
        int n = arr.length;
```

```

    if (n < m) {
        return -1;
    }

    Arrays.sort(arr);

    int minDiff = Integer.MAX_VALUE;

    for (int i = 0; i + m - 1 < n; i++) {
        int diff = arr[i + m - 1] - arr[i];
        minDiff = Math.min(minDiff, diff);
    }

    return minDiff;
}

public static void main(String[] args) {
    int[] arr1 = {7, 3, 2, 4, 9, 12, 56};
    int m1 = 3;
    System.out.println("Minimum difference: " + findMinDifference(arr1, m1));

    int[] arr2 = {7, 3, 2, 4, 9, 12, 56};
    int m2 = 5;
    System.out.println("Minimum difference: " + findMinDifference(arr2, m2));
}
}

```

OUPTUT:

```
D:\javaprograms\Practice ques\7>javac Chocolate.java
D:\javaprograms\Practice ques\7>java Chocolate
Minimum difference: 2
Minimum difference: 7
```

Time complexity: $O(n \log n)$

8.MERGE INTERVAL

Given an array of intervals where $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

Input: $\text{arr} = [[1, 3], [2, 4], [6, 8], [9, 10]]$

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: $\text{arr} = [[7, 8], [1, 5], [2, 4], [4, 6]]$

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

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

CODE:

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

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

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

```
public class Merge {
```

```
    public int[][] merge(int[][] intervals) {
```

```
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
```

```
        List<int[]> merged = new ArrayList<>();
```

```
        int[] prev = intervals[0];
```

```

for (int i = 1; i < intervals.length; i++) {
    int[] interval = intervals[i];
    if (interval[0] <= prev[1]) {
        prev[1] = Math.max(prev[1], interval[1]);
    } else {
        merged.add(prev);
        prev = interval;
    }
}
merged.add(prev);

// Convert list to array and return
return merged.toArray(new int[merged.size()][]);
}

```

```

public static void main(String[] args) {

    int[][] intervals1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
    int[][] intervals2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

    Merge solution = new Merge();

    System.out.println("Input: [[1, 3], [2, 4], [6, 8], [9, 10]]");
    int[][] mergedIntervals1 = solution.merge(intervals1);
    System.out.println("Merged Intervals:");
    for (int[] interval : mergedIntervals1) {
        System.out.println(Arrays.toString(interval));
    }
}

```

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

```
System.out.println("Input: [[7, 8], [1, 5], [2, 4], [4, 6]]");
```

```
int[][] mergedIntervals2 = solution.merge(intervals2);
```

```
System.out.println("Merged Intervals:");
```

```
for (int[] interval : mergedIntervals2) {
```

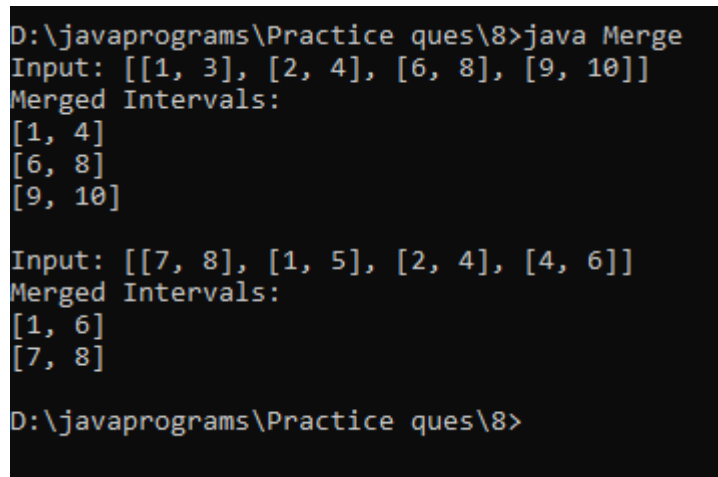
```
    System.out.println(Arrays.toString(interval));
```

```
}
```

```
}
```

```
}
```

OUPUT:



```
D:\javaprograms\Practice ques\8>java Merge
Input: [[1, 3], [2, 4], [6, 8], [9, 10]]
Merged Intervals:
[1, 4]
[6, 8]
[9, 10]

Input: [[7, 8], [1, 5], [2, 4], [4, 6]]
Merged Intervals:
[1, 6]
[7, 8]

D:\javaprograms\Practice ques\8>
```

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

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: $\{\{0, 0, 0\}, \{0, 0, 1\}\}$ Output: $\{\{0, 0, 1\}, \{1, 1, 1\}\}$

Code:

```
public class Setone {
```

```
    public void setOnes(int[][] matrix) {
```

```
        int m = matrix.length;
```

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

```
        boolean[] rows = new boolean[m];
```

```
boolean[] cols = new boolean[n];
```

```
for (int i = 0; i < m; i++) {  
    for (int j = 0; j < n; j++) {  
        if (matrix[i][j] == 1) {  
            rows[i] = true;  
            cols[j] = true;  
        }  
    }  
}
```

```
for (int i = 0; i < m; i++) {  
    for (int j = 0; j < n; j++) {  
        if (rows[i] || cols[j]) {  
            matrix[i][j] = 1;  
        }  
    }  
}
```

```
public static void main(String[] args) {  
    Setone solution = new Setone();
```

```
int[][] matrix1 = {{1, 0}, {0, 0}};
```

```
int[][] matrix2 = {{0, 0, 0}, {0, 0, 1}};
```

```
int[][] matrix3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};
```



```
System.out.println("Original matrix:");  
printMatrix(matrix1);  
solution.setOnes(matrix1);  
System.out.println("Modified matrix:");  
printMatrix(matrix1);
```

```
System.out.println("\nOriginal matrix:");  
printMatrix(matrix2);  
solution.setOnes(matrix2);  
System.out.println("Modified matrix:");  
printMatrix(matrix2);
```

```
System.out.println("\nOriginal matrix:");  
printMatrix(matrix3);  
solution.setOnes(matrix3);  
System.out.println("Modified matrix:");  
printMatrix(matrix3);
```

```
}
```

```
private static void printMatrix(int[][] matrix) {  
    for (int[] row : matrix) {  
        for (int val : row) {  
            System.out.print(val + " ");  
        }  
        System.out.println();  
    }  
}
```

}

```
D:\javaprograms\Practice ques\9>javac Setone.jav
D:\javaprograms\Practice ques\9>java Setone
Original matrix:
1 0
0 0
Modified matrix:
1 1
1 0

Original matrix:
0 0 0
0 0 1
Modified matrix:
0 0 1
1 1 1

Original matrix:
1 0 0 1
0 0 1 0
0 0 0 0
Modified matrix:
1 1 1 1
1 1 1 1
1 0 1 1
```

OUTPUT:

TIME COMPLEXITY: $O(M*N)$

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.

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

Code:

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

```
public class SpiralMatrix {
```

```
    public static List<Integer> spiralOrder(int[][] matrix) {
```

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

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

```
        int top = 0, bottom = matrix.length - 1;
```

```

while (left <= right && top <= bottom) {
    for (int i = left; i <= right; i++) {
        result.add(matrix[top][i]);
    }
    top += 1;
    for (int i = top; i <= bottom; i++) {
        result.add(matrix[i][right]);
    }
    right -= 1;
    if (top <= bottom) {
        for (int i = right; i >= left; i--) {
            result.add(matrix[bottom][i]);
        }
        bottom -= 1;
    }
    if (left <= right) {
        for (int i = bottom; i >= top; i--) {
            result.add(matrix[i][left]);
        }
        left += 1;
    }
}

return result;
}

public static void main(String[] args) {
    int[][] matrix1 = {

```

```

        {1, 2, 3, 4, 5, 6},
        {7, 8, 9, 10, 11, 12},
        {13, 14, 15, 16, 17, 18}
    };

    int[][] matrix2 = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };

    System.out.println("Spiral Order of Matrix 1: " + spiralOrder(matrix1));
    System.out.println("Spiral Order of Matrix 2: " + spiralOrder(matrix2));
}
}

```

OUTPUT:

```

D:\javaprograms\Practice ques\10>javac SpiralMatrix.java
D:\javaprograms\Practice ques\10>java SpiralMatrix
Spiral Order of Matrix 1: [1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11]
Spiral Order of Matrix 2: [1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]

```

TIME COMPLEXITY: $O(M*N)$

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

CODE:

```
import java.util.Stack;
```

```
class Balanced {
```

```
    public static String checkBalancedParentheses(String str) {
```

```
        Stack<Character> stack = new Stack<>();
```

```

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

    if (ch == '(') {
        stack.push(ch);
    } else if (ch == ')') {
        if (stack.isEmpty()) {
            return "Not Balanced";
        }
        stack.pop();
    }
}

return stack.isEmpty() ? "Balanced" : "Not Balanced";
}

public static void main(String[] args) {
    String str1 = "((()))()()";
    System.out.println(checkBalancedParentheses(str1));

    String str2 = "()()()()";
    System.out.println(checkBalancedParentheses(str2));
}
}

```

OUTPUT:

```

D:\javaprograms\Practice ques\13>javac Balanced.java
D:\javaprograms\Practice ques\13>java Balanced
Balanced
Not Balanced

```

Time 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

CODE:

```
class Anagram {
```

```
    public static boolean isAnagram(String s, String t) {
```

```
        if (s.length() != t.length()) {
```

```
            return false;
```

```
        }
```

```
        int[] x = new int[256];
```

```
        for (char c : s.toCharArray()) {
```

```
            x[c]++;
```

```
        }
```

```
        for (char c : t.toCharArray()) {
```

```
            x[c]--;
```

```
        }
```

```
for (int i : x) {  
    if (i != 0) {  
        return false;  
    }  
}  
  
return true;  
}
```

```
public static void main(String[] args) {  
    String s1 = "geeks";  
    String s2 = "kseeg";  
    System.out.println("Are the strings anagrams? " + isAnagram(s1, s2));  
  
    String s3 = "allergy";  
    String s4 = "allergic";  
    System.out.println("Are the strings anagrams? " + isAnagram(s3, s4));  
  
    String s5 = "g";  
    String s6 = "g";  
    System.out.println("Are the strings anagrams? " + isAnagram(s5, s6));  
}
```

```
}
```

```
D:\javaprograms\Practice ques\14>javac Anagram.java
```

```
D:\javaprograms\Practice ques\14>java Anagram
```

```
Are the strings anagrams? true
```

```
Are the strings anagrams? false
```

```
Are the strings anagrams? true
```

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.

Input: str = "forgeeksskeegfor"

Output: "geeksskeeg"

Explanation: There are several possible palindromic substrings like "kssk", "ss", "eeksskeeg" etc. But the substring "geeksskeeg" is the longest among all. Input: str = "Geeks" Output: "ee" Input: str = "abc" Output: "a" Input: str = "" Output: ""

CODE:

```
public class Palindrome {
```

```
    public String longestPalindrome(String s) {
```

```
        if (s.length() <= 1) {
```

```
            return s;
```

```
        }
```

```
        String maxStr = s.substring(0, 1);
```

```
        for (int i = 0; i < s.length() - 1; i++) {
```

```
            String odd = expandFromCenter(s, i, i);
```

```
            String even = expandFromCenter(s, i, i + 1);
```

```
            if (odd.length() > maxStr.length()) {
```

```
                maxStr = odd;
```

```
            }
```



```

        if (even.length() > maxStr.length()) {
            maxStr = even;
        }
    }

    return maxStr;
}

```

```

private String expandFromCenter(String s, int left, int right) {
    while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
        left--;
        right++;
    }
    return s.substring(left + 1, right);
}

```

```

public static void main(String[] args) {
    Palindrome solution = new Palindrome();

    String str1 = "forgeeksskeegfor";

    System.out.println("Longest palindromic substring: " +
        solution.longestPalindrome(str1));
}

```

```

    String str2 = "Geeks";

    System.out.println("Longest palindromic substring: " +
        solution.longestPalindrome(str2));
}

```

```

    String str3 = "abc";

    System.out.println("Longest palindromic substring: " +
        solution.longestPalindrome(str3));
}

```

```

    String str4 = "";
}

```

```
System.out.println("Longest palindromic substring: " +  
solution.longestPalindrome(str4));
```

```
String str5 = "babad";
```

```
System.out.println("Longest palindromic substring: " +  
solution.longestPalindrome(str5));
```

```
}
```

```
}
```

OUTPUT:

```
D:\javaprograms\Practice ques\15>javac Palindrome.java  
  
D:\javaprograms\Practice ques\15>java Palindrome  
Error: Could not find or load main class Palindrome  
Caused by: java.lang.ClassNotFoundException: Palindrome  
  
D:\javaprograms\Practice ques\15>java Palindrome  
Longest palindromic substring: geeksskeeg  
Longest palindromic substring: ee  
Longest palindromic substring: a  
Longest palindromic substring:  
Longest palindromic substring: bab
```

TIME COMPLEXITY: $O(N^2)$

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.

CODE:

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

```
public class Prefix {
```

```
    public static String longestCommonPrefix(String[] strs) {
```

```
        StringBuilder ans=new StringBuilder();
```

```
        Arrays.sort(strs);
```

```
        String first=strs[0];
```

```
        String last =strs[strs.length-1];
```

```
        for(int i=0;i<Math.min(first.length(),last.length());i++){
```

```
            if(first.charAt(i)!=last.charAt(i)){
```

```
                return ans.toString();
```

```

    }
    ans.append(first.charAt(i));
}
return ans.toString();
}

public static void main(String[] args) {
    String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
    System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr1));
    String[] arr2 = {"apple", "banana", "cherry"};
    System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr2));
    String[] arr3 = {"dog", "racecar", "car"};
    System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr3));
    String[] arr5 = {"hello"};
    System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr5));
}
}

```

OUTPUT:

```

D:\javaprograms\Practice ques\16>javac Prefix.java
D:\javaprograms\Practice ques\16>java Prefix
Longest Common Prefix: gee
Longest Common Prefix:
Longest Common Prefix:
Longest Common Prefix: hello

```

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

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]

CODE: import java.util.Stack;

```
class DeleteStack {
```

```

    public static void deleteMiddle(Stack<Integer> stack, int count, int size) {
        if (stack.isEmpty() || count == size / 2) {

```

```

        stack.pop();
        return;
    }

    int temp = stack.pop();
    deleteMiddle(stack, count + 1, size);
    stack.push(temp);
}

public static void main(String[] args) {
    Stack<Integer> stack1 = new Stack<>();
    stack1.push(1);
    stack1.push(2);
    stack1.push(3);
    stack1.push(4);
    stack1.push(5);
    int size1 = stack1.size();
    deleteMiddle(stack1, 0, size1);
    System.out.println("Stack after deleting middle element: " + stack1);
    Stack<Integer> stack2 = new Stack<>();
    stack2.push(1);
    stack2.push(2);
    stack2.push(3);
    stack2.push(4);
    stack2.push(5);
    stack2.push(6);
    int size2 = stack2.size();
    deleteMiddle(stack2, 0, size2);
    System.out.println("Stack after deleting middle element: " + stack2);
}

```

}

```
D:\javaprograms\Practice ques\17>javac DeleteStack.java
D:\javaprograms\Practice ques\17>java DeleteStack
Stack after deleting middle element: [1, 2, 4, 5]
Stack after deleting middle element: [1, 2, 4, 5, 6]
```

OUTPUT:

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. 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

CODE:

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

```
class Greater {
```

```
    public static void nextGreaterElement(int[] arr) {
```

```
        int n = arr.length;
```

```
        Stack<Integer> stack = new Stack<>();
```

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

```
        Arrays.fill(result, -1);
```

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

```
            while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {
```

```
                result[stack.pop()] = arr[i];
```

```
            }
```

```

        stack.push(i);
    }

    for (int i = 0; i < n; i++) {
        System.out.println(arr[i] + " --> " + result[i]);
    }
}

public static void main(String[] args) {
    int[] arr1 = { 4, 5, 2, 25 };
    System.out.println("Next Greater Element for arr1:");
    nextGreaterElement(arr1);

    int[] arr2 = { 13, 7, 6, 12 };
    System.out.println("\nNext Greater Element for arr2:");
    nextGreaterElement(arr2);
}
}

```

OUTPUT:

```

D:\javaprograms\Practice ques\18>javac Greater.java

D:\javaprograms\Practice ques\18>java Greater
Next Greater Element for arr1:
4 --> 5
5 --> 25
2 --> 25
25 --> -1

Next Greater Element for arr2:
13 --> -1
7 --> 12
6 --> 12
12 --> -1

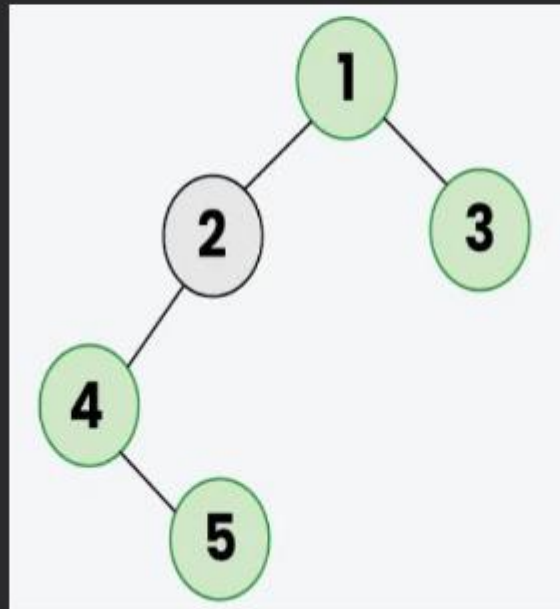
```

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.

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



CODE:

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

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

```
public class Right {
```

```
    public static List<Integer> rightSideView(TreeNode root) {
```

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

```
        rightView(root, result, 0);
```

```
        return result;
```

```
    }
```

```
    public static 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 root1 = new TreeNode(1);
        root1.left = new TreeNode(2);
        root1.right = new TreeNode(3);
        root1.right.left = new TreeNode(4);
        root1.right.right = new TreeNode(5);
        root1.right.left.left=new TreeNode(6);
        root1.right.left.left.right=new TreeNode(7);
        System.out.println("Right side view of tree 1: " + rightSideView(root1));
    }
}

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

```


OUTPUT:

```
D:\javaprograms\Practice ques\19>javac Right.java
```

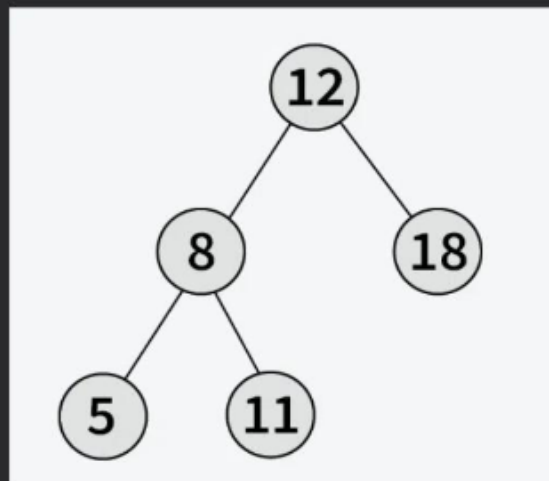
```
D:\javaprograms\Practice ques\19>java Right  
Right side view of tree 1: [1, 3, 5, 6, 7]
```

TIME COMPLEXITY: $O(n)$

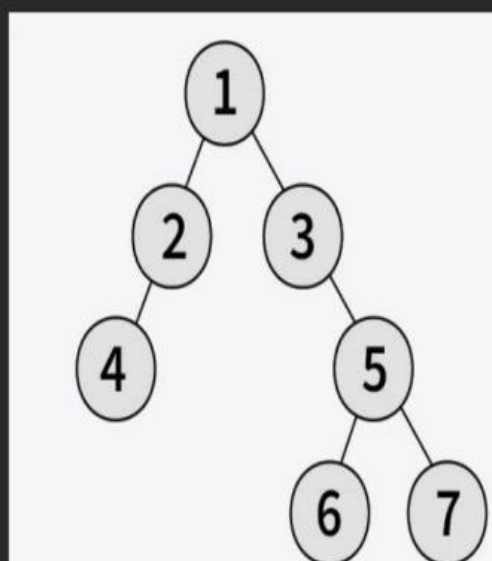
20: Maximum Depth or Height of Binary Tree

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



CODE:

```
public class MaxHeight {  
    public static int maxDepth(TreeNode root) {  
        if (root == null) {  
            return 0;  
        }  
  
        int leftDepth = maxDepth(root.left);  
        int rightDepth = maxDepth(root.right);  
  
        return Math.max(leftDepth, rightDepth) + 1;  
    }  
  
    public static void main(String[] args) {  
        // Create the binary tree  
        TreeNode root = new TreeNode(1);  
        root.left = new TreeNode(2);  
        root.right = new TreeNode(3);  
        root.left.left = new TreeNode(4);  
        root.left.right = new TreeNode(5);  
        root.right.right = new TreeNode(6);  
        root.left.left.left = new TreeNode(7);  
  
        System.out.println("Maximum Depth of Binary Tree: " + maxDepth(root)); //  
Output: 4  
    }  
}  
  
class TreeNode {  
    int val;
```

```
TreeNode left;  
TreeNode right;
```

```
TreeNode(int x) {  
    val = x;  
    left = null;  
    right = null;  
}
```

```
}
```

OUTPUT:

```
D:\javaprograms\Practice ques\20>javac MaxHeight.java  
D:\javaprograms\Practice ques\20>java MaxHeight  
Maximum Depth of Binary Tree: 4
```

Time Complexity: $O(n)$