Date: 09-11-2024

Practice set 1

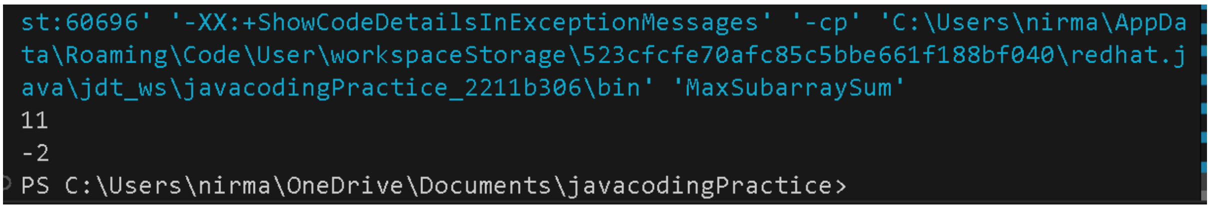
Coding practice Problems:

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

Code:

```
public class MaxSubarraySum {
  public static int sum(int[] arr){
    int s=arr[0];
    for (int i=0;i<arr.length;i++){
      int c=0;
      for (int j=i;j<arr.length;j++){
        c=c+arr[j];
        s=Math.max(s,c);
      }
    }
    return s;
  }
  public static void main(String[] args) {
    int[] arr1={2, 3, -8, 7, -1, 2, 3};
    int[] arr2={-2, -4};
    System.out.println(sum(arr1));
    System.out.println(sum(arr2));
  }
}
```

Output;



```
st:60696' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\nirma\AppDa
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'MaxSubarraySum'
11
-2
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time Complexity**:** O(n^2)

Space Complexity: O(1)

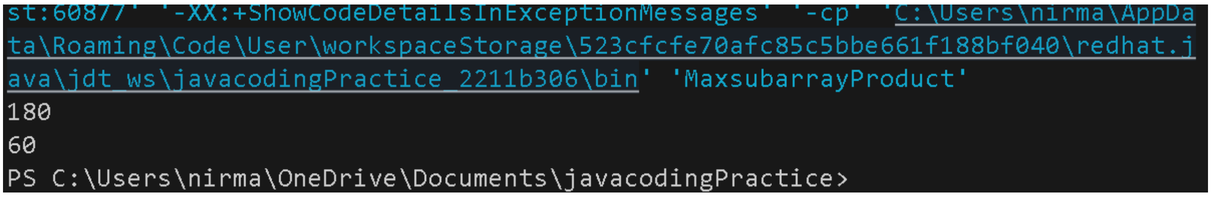
2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray. Input: arr[] = {-2, 6, -3, -10, 0, 2} Output: 180 Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 * (-3) * (-10) = 180 Input: arr[] = {-1, -3, -10, 0, 60} Output: 60

Code:

```
public class MaxsubarrayProduct {
  public static int prod(int[] arr){
    int p=arr[0];
    for (int i=0;i<arr.length;i++){
      int c=1;
      for (int j=i;j<arr.length;j++){
        c=c*arr[j];
        p=Math.max(p,c);
      }
    }
    return p;
  }
  public static void main(String[] args){
    int[] arr1={-2, 6, -3, -10, 0, 2};
    int[] arr2={-1, -3, -10, 0, 60};
    System.out.println(prod(arr1));
    System.out.println(prod(arr2));
  }
```

}

Output:



```
st:60877' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\nirma\AppDa
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'MaxsubarrayProduct'
180
60
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time Complexity: O(n^2)

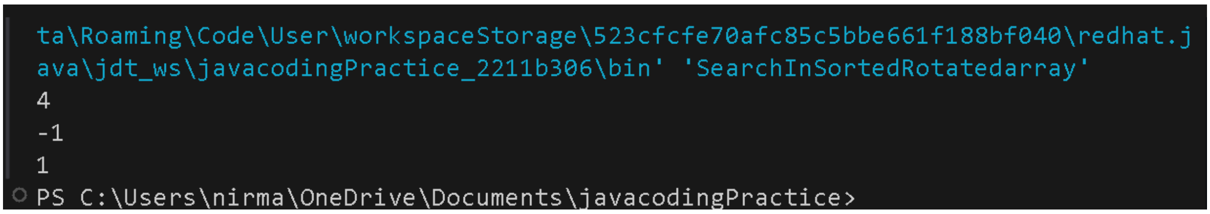
Space complexity: O(1)

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

Code:

```
public class Searchvalue {
  public static int search(int[] arr, int key) {
    int low=0,high=arr.length-1;
    while (low <= high){
      int mid=low+(high-low)/2;
      if (arr[mid]==key){
        return mid;
      }
      if (arr[mid]>=arr[low]) {
        if (arr[low]<=key && key<=arr[mid]) {
          high=mid-1;
        } else {
          low=mid+1;
        }
      }
      else {
        if (arr[mid]<=key && key<=arr[high]) {
          low=mid+1;
        } else {
          high=mid-1;
        }
      }
    }
    return -1;
  }
  public static void main(String[] args) {
    int[] arr1={4, 5, 6, 7, 0, 1, 2};
    int[] arr2={50, 10, 20, 30, 40};
    int key=0;
    int key2=3;
    int key3=10;
    System.out.println(search(arr1, key));
    System.out.println(search(arr1, key2));
    System.out.println(search(arr2, key3));
  }
}
```

Output:

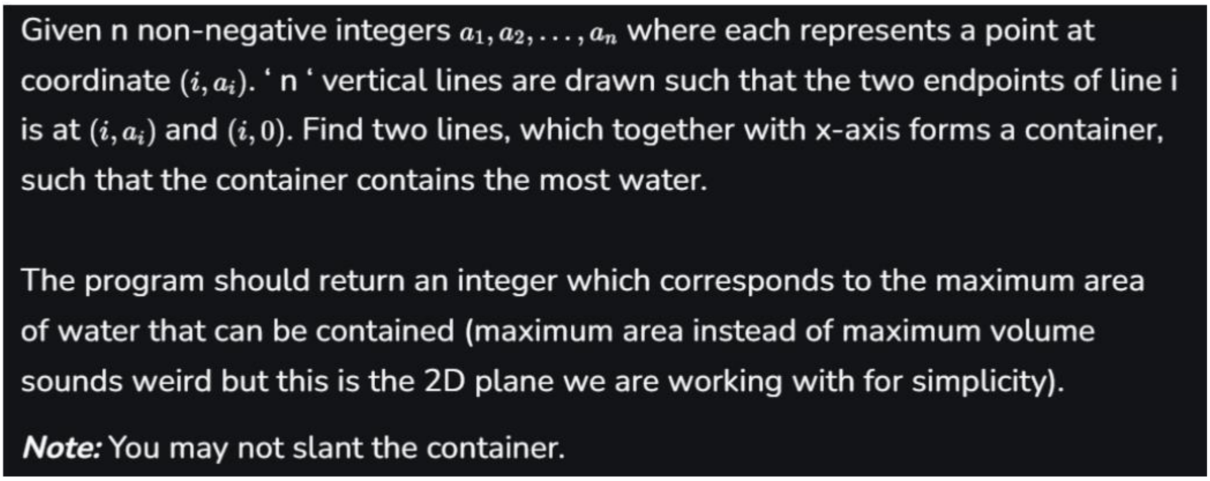


```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'SearchInSortedRotatedarray'
4
-1
1
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(log n)

Space complexity: O(1)

4. Container with Most Water



Given n non-negative integers $a_1, a_2, \ldots, a_n$ 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).

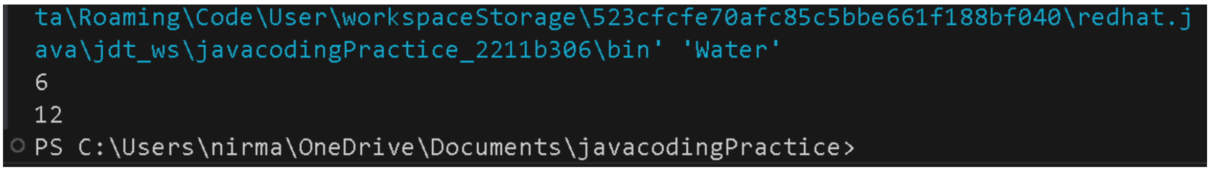
Note: You may not slant the container.

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

Code:

```
public class Water {
  public static int result(int[] arr){
    int low=0,high=arr.length-1;
    int m=0;
    while (low<high){
      int w=high-low;
      int h=Math.min(arr[low], arr[high]);
      m=Math.max(m, h*w);
      if (arr[low]<arr[high]){
        low++;
      }else{
        high--;
      }
    }
    return m;
  }
  public static void main(String[] args) {
    int[] arr1={1,5,4,3};
    int[] arr2={3,1,2,4,5};
    System.out.println(result(arr1));
    System.out.println(result(arr2));

  }
}
```

Output:



```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'Water'
6
12
○ PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n)

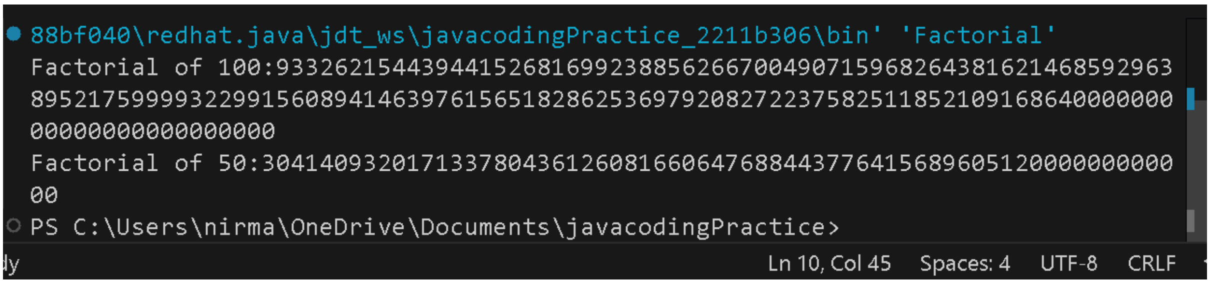
Space complexity: O(1)

5. Find the Factorial of a large number
   Input: 100 Output:
   9332621544394415268169923885626670049071596826438162146859296389
   52175999932299
   1560894146397615651828625369792082722375825118521091686400000000
   00000000000000 00
   Input: 50 Output:
   3041409320171337804361260816606476884437764156896051200000000000
   0

Code:

```
import java.math.BigInteger;
public class Factorial {
  public static BigInteger fact(int n){
    BigInteger r=BigInteger.valueOf(1);
    for (int i=1;i<=n;i++){
      r=r.multiply(BigInteger.valueOf(i));
    }
    return r;
  }
  public static void main(String[] args) {
    int n1=100;
    int n2=50;
    System.out.println("Factorial of "+n1+":"+fact(n1));
    System.out.println("Factorial of "+n2+":"+fact(n2));
  }
}
```

Output:



```
● 88bf040\redhat.java\jdt_ws\javacodingPractice_2211b306\bin' 'Factorial'
  Factorial of 100:9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369792082722375825118521091686400000000000000000000000
  Factorial of 50:3041409320171337804361260816606476884437764156896051200000000000
○ PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

```
ly                                         Ln 10, Col 45   Spaces: 4   UTF-8   CRLF
```

Time complexity: O(n^2 log n)

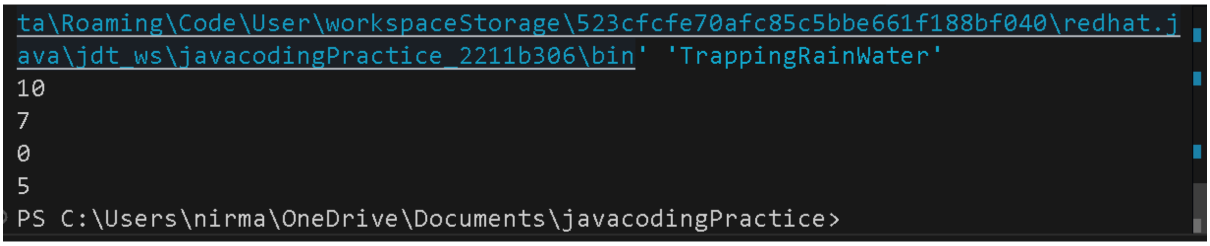
Space complexity: O(n)

6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain. 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

Code:

```
public class TrappingRainWater {
  public static int water(int[] arr){
    int n=arr.length;
    if (n<3){
        return 0;
    }
    int[] left=new int[n];
    int[] right=new int[n];
    int r=0;
    left[0]=arr[0];
    for (int i=1;i<n;i++){
        left[i]=Math.max(left[i-1], arr[i]);
    }
    right[n-1]=arr[n-1];
    for (int i=n-2;i>=0;i--){
        right[i]=Math.max(right[i+1], arr[i]);
    }
    for (int i=0;i<n;i++){
        r+=Math.min(left[i],right[i])-arr[i];
    }
    return r;
}
public static void main(String[] args) {
    int[] arr1={3,0,1,0,4,0,2};
    int[] arr2={3,0,2,0,4};
    int[] arr3={1,2,3,4};
    int[] arr4={10,9,0,5};
    System.out.println(water(arr1));
    System.out.println(water(arr2));
    System.out.println(water(arr3));
    System.out.println(water(arr4));
    }
  }
```

Output:



```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'TrappingRainWater'
10
7
0
5
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n)

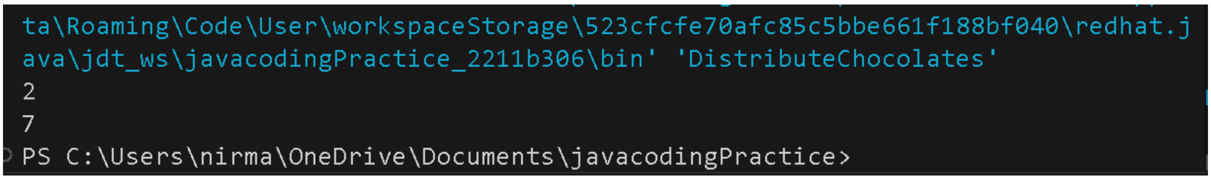
Space complexity: O(n)

7. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith 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.

   Code:

```
import java.util.Arrays;
public class DistributeChocolates {
  public static int md(int[] arr,int m){
    Arrays.sort(arr);
    if (arr.length<m){
      return -1;
    }
    int d=Integer.MAX_VALUE;
    for (int i=0;i<arr.length-m;i++){
      int mi=arr[i+m-1]-arr[i];
      d=Math.min(mi,d);
    }
    return d;
  }
  public static void main(String[] args) {
    int[] arr1={7, 3, 2, 4, 9, 12, 56};
    int[] arr2={7, 3, 2, 4, 9, 12, 56};
    int m1=3,m2=5;
    System.out.println(md(arr1, m1));
    System.out.println(md(arr2, m2));
  }
}
```

Output:



```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'DistributeChocolates'
2
7
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n log n)

Space complexity: O(1)

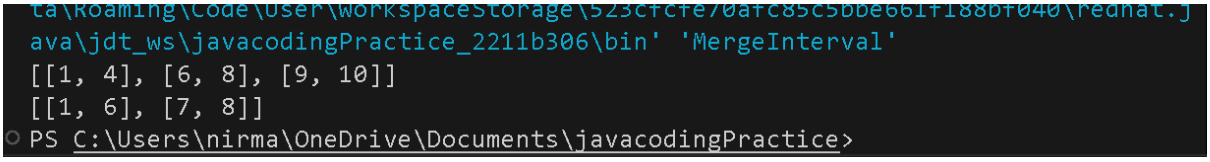
8. Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals. Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]] Output: [[1, 4], [6, 8], [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

Code:

```
import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Stack;

public class MergeInterval {

  public static int[][] merge(int[][] arr){

    if (arr.length<=1){

      return arr;

    }

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

```
        Stack<int[]> st=new Stack<>();
        st.push(arr[0]);
        for (int i=1;i<arr.length;i++){
          int[] top=st.peek();
          int[] current=arr[i];
          if (top[1]>current[0]){
            top[1]=Math.max(top[1],current[1]);
          }else{
            st.push(current);
          }
        }
        List<int[]> r=new ArrayList<>(st);
        return r.toArray(new int[r.size()][]);
      }
      public static void main(String[] args) {
        int[][] arr1={{1,3},{2,4},{6,8},{9,10}};
        int[][] arr2={{7,8},{1,5},{2,4},{4,6}};
        System.out.println(Arrays.deepToString(merge(arr1)));
        System.out.println(Arrays.deepToString(merge(arr2)));
      }
    }
```

Output:



```
  ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
  ava\jdt_ws\javacodingPractice_2211b306\bin' 'MergeInterval'
  [[1, 4], [6, 8], [9, 10]]
  [[1, 6], [7, 8]]
○ PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n log n)

Space complexity: O(n)

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 ith row and jth 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}}

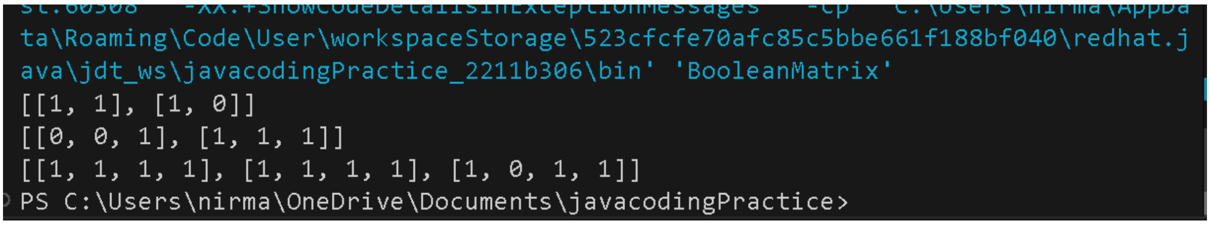
Output: {{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 0, 1, 1}}

Code:

```
import java.util.Arrays;

public class BooleanMatrix {

  public static int[][] matrix(int[][] arr){

    int m=arr.length;

    int n=arr[0].length;

    boolean[] row=new boolean[m];

    boolean[] col=new boolean[n];

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

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

        if (arr[i][j]==1){

          row[i]=true;

          col[j]=true;

        }

      }

    }

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

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

        if (row[i]||col[j]){

          arr[i][j]=1;

        }
```

```
          }
        }
        return arr;
      }
      public static void main(String[] args) {
        int[][] arr1={{1,0},{0,0}};
        int[][] arr2={{0,0,0},{0,0,1}};
        int[][] arr3={{1,0,0,1},{0,0,1,0},{0,0,0,0}};
        matrix(arr1);
        matrix(arr2);
        matrix(arr3);
        System.out.println(Arrays.deepToString(arr1));
        System.out.println(Arrays.deepToString(arr2));
        System.out.println(Arrays.deepToString(arr3));
      }
    }
```

Output:



```
st:60308  -XX:+ShowCodeDetailsInExceptionMessages  -cp  C:\Users\nirma\AppDa
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'BooleanMatrix'
[[1, 1], [1, 0]]
[[0, 0, 1], [1, 1, 1]]
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 0, 1, 1]]
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(MxN)

Space complexity: O(M+N)

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

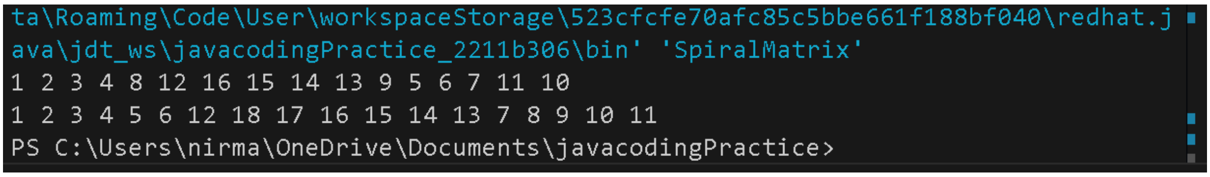
Code:

```
public class SpiralMatrix {
```

```
public static void print(int[][] arr){
   if(arr.length==0){
      return ;
   }
   int top=0,bottom=arr.length-1;
   int left=0,right=arr[0].length-1;
   while (top<=bottom && left<=right){
      for (int i=left;i<=right;i++){
         System.out.print(arr[top][i]+" ");
      }
      top++;
      for (int i=top;i<=bottom;i++){
         System.out.print(arr[i][right]+" ");
      }
      right--;
      if (top<=bottom){
         for (int i=right;i>=left;i--){
            System.out.print(arr[bottom][i]+" ");
         }
         bottom--;
      }
      if (left<=right){
         for (int i=bottom;i>=top;i--){
            System.out.print(arr[i][left]+" ");
         }
         left++;
      }
   }
   System.out.println();
```

```
  }
  public static void main(String[] args) {
    int[][] arr1={
      {1, 2, 3, 4},
      {5, 6, 7, 8},
      {9, 10, 11, 12},
      {13, 14, 15, 16 }
    };
    int[][] arr2={
      {1, 2, 3, 4, 5, 6},
      {7, 8, 9, 10, 11, 12},
      {13, 14, 15, 16, 17, 18}
    };
    print(arr1);
    print(arr2);
  }
}
```

Output:



```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'SpiralMatrix'
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
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(MxN)

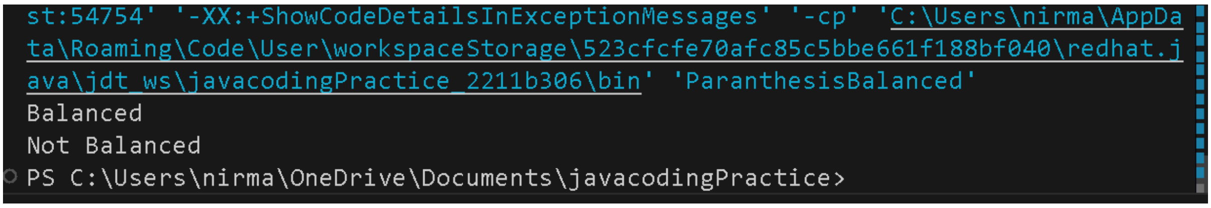
Space complexity: O(1)

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;
public class ParanthesisBalanced {
    public static String check(String str){
        Stack<Character> st=new Stack<>();
        for (int i=0;i<str.length();i++){
            if (str.charAt(i)=='('){
                st.push(str.charAt(i));
            }else{
                if (st.isEmpty()){
                    return "Not Balanced";
                }else{
                    st.pop();
                }
            }
        }
        if (st.isEmpty()){
            return "Balanced";
        }else{
            return "Not Balanced";
        }
    }
    public static void main(String[] args) {
        System.out.println(check("((()))()()"));
        System.out.println(check("())((())"));
    }
}
```

Output:



```
st:54754' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\nirma\AppDa
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'ParanthesisBalanced'
Balanced
Not Balanced
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n)

Space complexity: O(n)

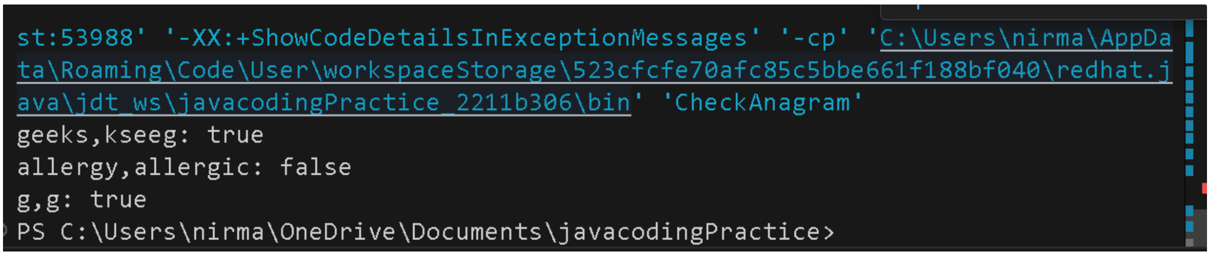
14. Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters 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.

Code:

```
import java.util.HashMap;

public class CheckAnagram {

  public static boolean compare(String s1,String s2) {

    HashMap<String,Integer> map1=new HashMap<>();

    HashMap<String,Integer> map2=new HashMap<>();

    String a="";

    if (s1.length()!=s2.length()){

      return false;

    }

    else{

      for (int i=0;i<s1.length();i++){

        if (!a.contains(Character.toString(s1.charAt(i)))){

          a=a+s1.charAt(i);

        }

      }
      for (int j=0;j<a.length();j++){
        int c=0;
        for (int k=0;k<s1.length();k++){
          if (s1.charAt(k)==a.charAt(j)){
            c++;
          }
        }
        int b=0;
        for (int l=0;l<s2.length();l++){
          if (s2.charAt(l)==a.charAt(j)){
            b++;
          }
        }
        map1.put(Character.toString(a.charAt(j)),c);
        map2.put(Character.toString(a.charAt(j)),b);
      }
      int m=0;
      for (int r=0;r<a.length();r++){
        if
(map1.get(Character.toString(a.charAt(r)))==map2.get(Character.toString(a.charAt(r))))
{
          m++;
        }
      }
      if (m==a.length()){
        return true;
      }else{
        return false;
      }
```

```
        }
    }
    public static void main(String[] args) {
        System.out.println("geeks,kseeg: "+compare("geeks", "kseeg"));
        System.out.println("allergy,allergic: "+compare("allergy", "allergic"));
        System.out.println("g,g: "+compare("g", "g"));
    }
}
```

Output:



```
st:53988' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\nirma\AppDa
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'CheckAnagram'
geeks,kseeg: true
allergy,allergic: false
g,g: true
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n^2)

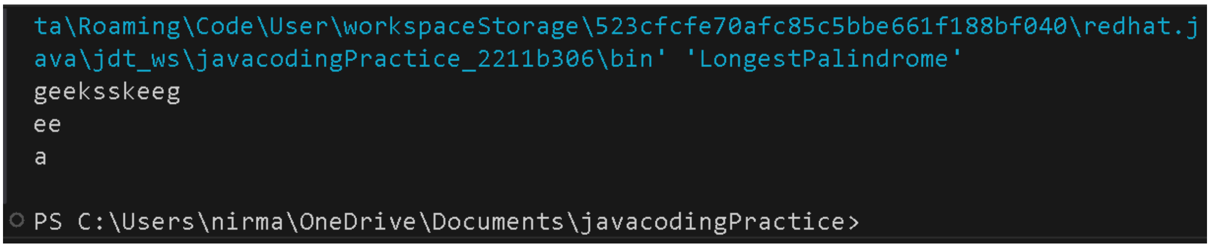
Space complexity: O(n)

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”, “eeksskee” 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 LongestPalindrome {
    public static String result(String s){
        if (s == null || s.length()==0){
            return "";
        }
        String l="";
        for (int i=0;i<s.length();i++){
            String o=expand(s,i,i);
            String e=expand(s,i,i+1);
            if (o.length()>l.length()){
                l=o;
            }
            if (e.length()>l.length()){
                l=e;
            }
        }
        return l;
    }
    public static String expand(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) {
        String s1="forgeeksskeegfor";
        String s2="Geeks";
        String s3="abc";
        String s4="";
        System.out.println(result(s1).toString());
        System.out.println(result(s2).toString());
        System.out.println(result(s3).toString());
        System.out.println(result(s4).toString());
    }
}
```

Output:



```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'LongestPalindrome'
geeksskeeg
ee
a

○ PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n^2)

Space complexity: O(n)

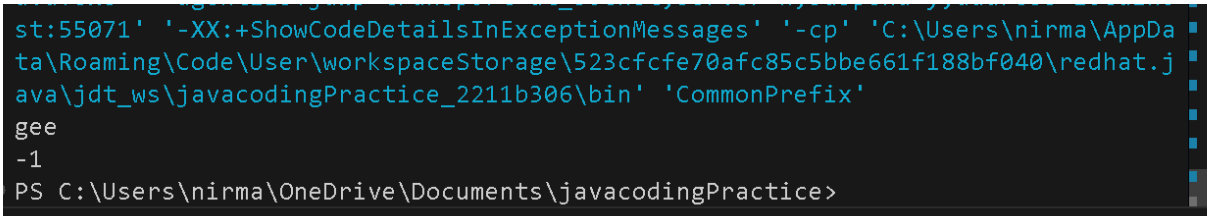
16. Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”. Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”] Output: 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.

Code:

```
import java.util.Arrays;

import java.util.Comparator;

public class CommonPrefix {

  public static String prefix(String[] arr){

    Arrays.sort(arr,Comparator.comparingInt(String::length));

    String r="";

    String s=arr[0];

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

      int c=0;

      for (int j=0;j<arr.length;j++){

        if (s.charAt(i)!=arr[j].charAt(i)){

          break;

        }else{

          c++;

        }
```

```
            }
            if (c==arr.length){
                r=r+s.charAt(i);
            }else{
                break;
            }
        }
        return (r.length()>0)?r:"-1";
    }
    public static void main(String[] args) {
        String[] arr1={"geeksforgeeks", "geeks", "geek", "geezer"};
        String[] arr2={"hello","world"};
        System.out.println(prefix(arr1));
        System.out.println(prefix(arr2));
    }
}
```

Output:



```
st:55071' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\nirma\AppDa
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'CommonPrefix'
gee
-1
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n^2)

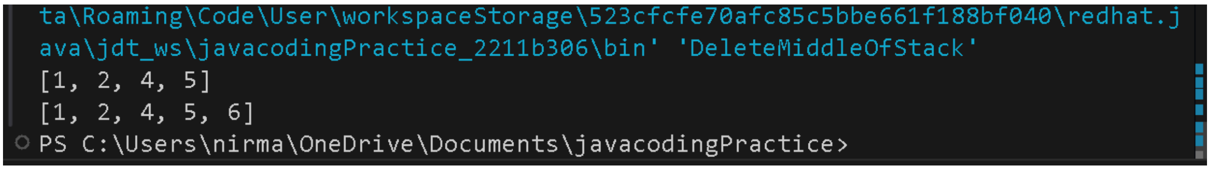
Space complexity: O(n)

17. Delete middle element of a stack Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure. 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:

```java
import java.util.Stack;
import java.util.Arrays;
public class DeleteMiddleOfStack {
    public static int[] delete(int[] arr){
        Stack<Integer> st1=new Stack<>();
        Stack<Integer> st2=new Stack<>();
        for (int i=0;i<arr.length;i++){
            st1.push(arr[i]);
        }
        int m=-1;
        if (arr.length%2==1){
            m=arr.length/2;
        }else{
            m=(arr.length/2)-1;
        }
        while (st1.peek()!=arr[m]){
            st2.push(st1.pop());
        }
        st1.pop();
        while (!st2.isEmpty()){
            st1.push(st2.pop());
        }
        int[] r=new int[arr.length-1];
        for (int j=r.length-1;j>=0;j--){
            r[j]=st1.pop();
        }
        return r;
    }
    public static void main(String[] args) {
        int[] arr1={1,2,3,4,5};
        int[] arr2={1,2,3,4,5,6};
        System.out.println(Arrays.toString(delete(arr1)));
        System.err.println(Arrays.toString(delete(arr2)));
    }
}
```

Output:



```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'DeleteMiddleOfStack'
[1, 2, 4, 5]
[1, 2, 4, 5, 6]
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n)

Space complexity: O(n)

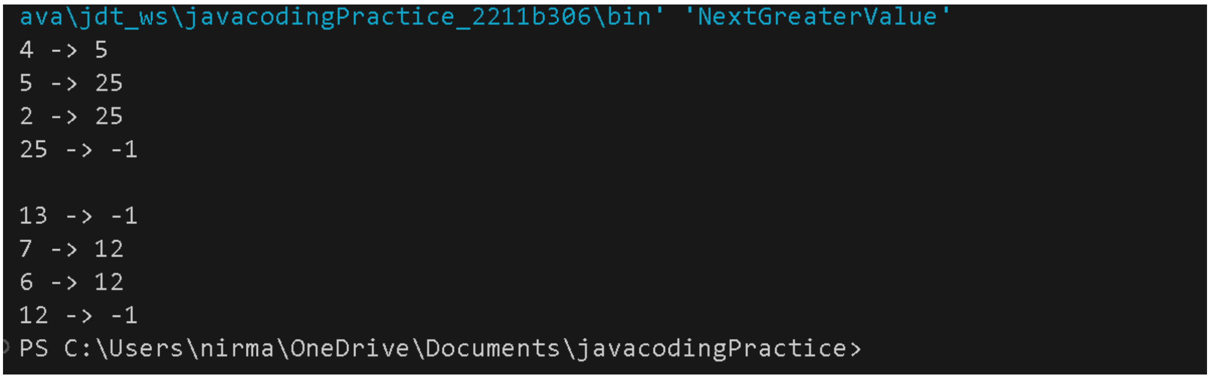
18. Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider 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:

```
public class NextGreaterValue {
    public static void next(int[] arr){
        for (int i=0;i<arr.length;i++){
            if (i==(arr.length-1)){
                System.out.println(arr[i]+" -> "+-1);
            }else{
                int c=-1;
                for (int j=i+1;j<arr.length;j++){
```

```
          if (arr[j]>arr[i]){
            c=j;
            break;
          }
        }
        if (c!=-1){
          System.out.println(arr[i]+" -> "+arr[c]);
        }else{
          System.out.println(arr[i]+" -> "+c);
        }
      }
    }
  }
  public static void main(String[] args) {
    int[] arr1={4,5,2,25};
    int[] arr2={13,7,6,12};
    next(arr1);
    System.out.println();
    next(arr2);
  }
}
```

Output:



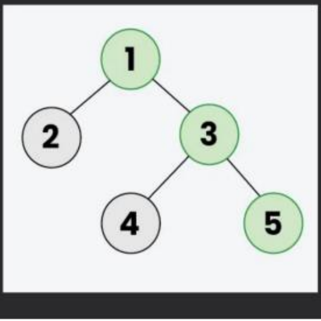
```
ava\jdt_ws\javacodingPractice_2211b306\bin' 'NextGreaterValue'
4 -> 5
5 -> 25
2 -> 25
25 -> -1

13 -> -1
7 -> 12
6 -> 12
12 -> -1
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n^2)

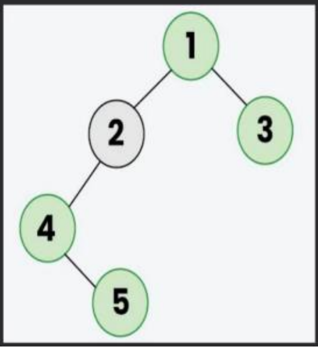
Space complexity: O(1)

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.*

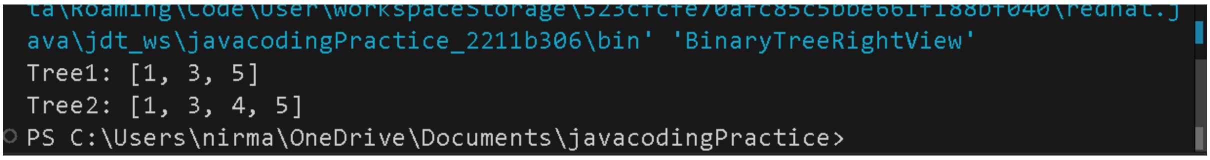


Code:

```
import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

import java.util.Queue;

class TreeNode {

   int val;

   TreeNode left,right;

   public TreeNode(int val) {

      this.val=val;
```

```
        this.left=this.right=null;
    }
}
public class BinaryTreeRightView {
    public List<Integer> rightView(TreeNode root) {
        List<Integer> rightViewList=new ArrayList<>();
        if (root==null){
            return rightViewList;
        }
        Queue<TreeNode> queue=new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            int levelSize=queue.size();
            for (int i=0;i<levelSize;i++) {
                TreeNode node=queue.poll();
                if (i==levelSize-1) {
                    rightViewList.add(node.val);
                }
                if (node.left!=null) {
                    queue.add(node.left);
                }
                if (node.right!=null) {
                    queue.add(node.right);
                }
            }
        }
        return rightViewList;
    }
    public static void main(String[] args) {
        BinaryTreeRightView tree=new BinaryTreeRightView();
        TreeNode root1=new TreeNode(1);
        root1.left=new TreeNode(2);
        root1.right=new TreeNode(3);
        root1.left.right=new TreeNode(4);
        root1.right.right=new TreeNode(5);
        System.out.println("Tree1: "+tree.rightView(root1));
        TreeNode root2 = new TreeNode(1);
        root2.left = new TreeNode(2);
        root2.right = new TreeNode(3);
        root2.left.left = new TreeNode(4);
        root2.left.left.right = new TreeNode(5);
        System.out.println("Tree2: "+tree.rightView(root2));
    }
}
```

Output:



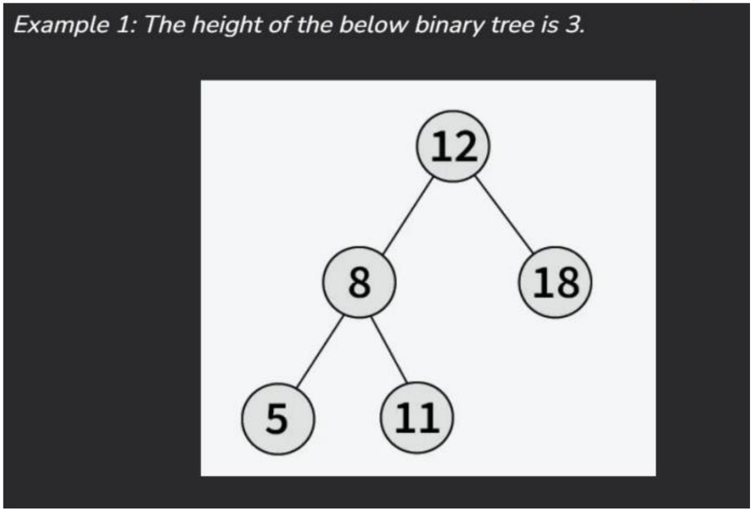
```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'BinaryTreeRightView'
Tree1: [1, 3, 5]
Tree2: [1, 3, 4, 5]
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n)

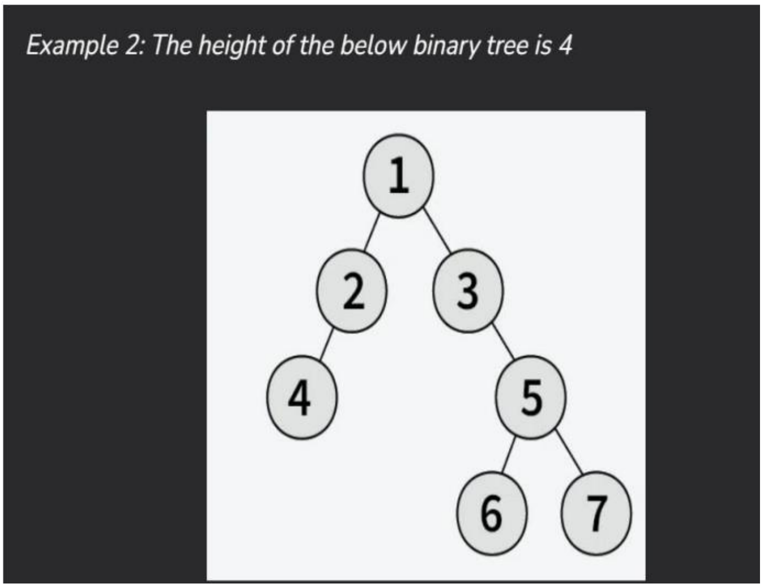
Space complexity: O(n)

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.



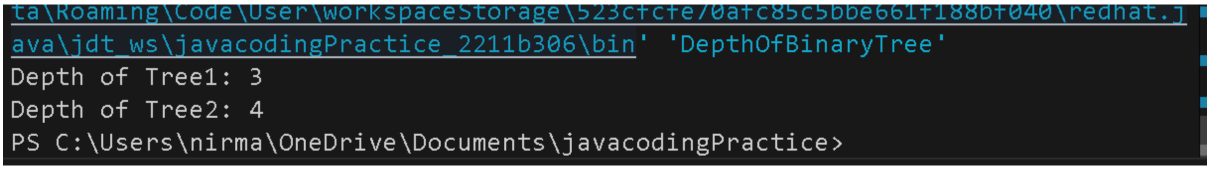
Example 2: The height of the below binary tree is 4

Code:

```
public class DepthOfBinaryTree {
  static class Node {
    int data;
    Node left,right;
    Node(int item){
      data=item;
      left=null;
      right=null;
    }
  }
```

```
    public static int depth(Node root){
        if (root==null){
            return 0;
        }
        int leftdepth=depth(root.left);
        int rightdepth=depth(root.right);
        return Math.max(leftdepth, rightdepth)+1;
    }
    public static void main(String[] args) {
        Node tree1=new Node(1);
        tree1.left=new Node(2);
        tree1.right=new Node(3);
        tree1.left.left=new Node(4);
        tree1.right.right=new Node(5);
        tree1.right.right.left=new Node(6);
        tree1.right.right.right=new Node(7);
        Node tree2=new Node(12);
        tree2.left=new Node(8);
        tree2.right=new Node(18);
        tree2.left.left=new Node(5);
        tree2.left.right=new Node(11);
        System.out.println("Depth of Tree1: "+depth(tree2));
        System.out.println("Depth of Tree2: "+depth(tree1));


    }
}
```

Output:



```
ta\Roaming\Code\User\workspaceStorage\523cfcfe70afc85c5bbe661f188bf040\redhat.j
ava\jdt_ws\javacodingPractice_2211b306\bin' 'DepthOfBinaryTree'
Depth of Tree1: 3
Depth of Tree2: 4
PS C:\Users\nirma\OneDrive\Documents\javacodingPractice>
```

Time complexity: O(n)

Space complexity: O(n)