**DSA PRACTICE SET 1**

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.

**CODE:**

class SubSum {

public static int maxSubSum(int[] arr) {

int b = arr[0];

int a = arr[0];

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

a = Math.max(a+ arr[i], arr[i]);

b = Math.max(b, a);

}

return b;

}

public static void main(String[] args) {

int[] arr = {2, 3, -8, 7, -1, 2, 3};

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

Int[]arr = {-2,-4};

System.out.println(maxSubSum)

Int[] arr= {5,4,1,7,8};

}

}

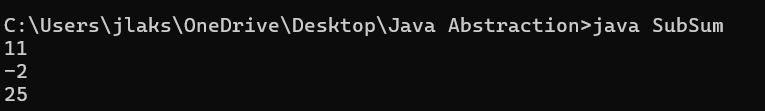
Input:

arr[] = {2, 3, -8, 7, -1, 2, 3}

arr[] = {-2, -4}

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

Output:



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}.

**CODE:**

public class MaxSubProd {

    public static int MaxPro(int[] arr){

        int minsum=arr[0];

        int maxsum=arr[0];

        int ans=arr[0];

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

            if (arr[i]<0){

                int temp = maxsum;

                maxsum=minsum;

                minsum=temp;

            }

            maxsum=Math.max(arr[i],maxsum\*arr[i]);

            minsum=Math.min(arr[i],minsum\*arr[i]);

            ans = Math.max(ans,maxsum);

        }

        return ans;

    }

    public static void main(String args[]){

        int[] arr1={-2, 6, -3, -10, 0, 2};

        System.out.println(MaxPro(arr1));

        int[] arr2={-1, -3, -10, 0, 60};

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

    }

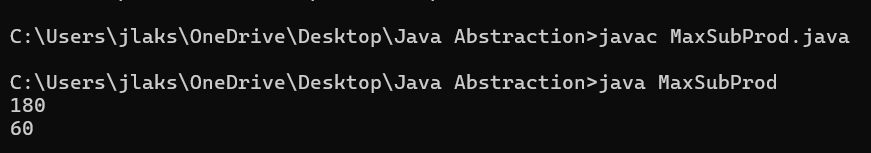
}

Input:

arr[] = {-2, 6, -3, -10, 0, 2}

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

Output:



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

**CODE:**

public class Revsearch {

    public static int RevSearch(int[] arr, int key){

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

            if (arr[i]==key){

                return i;

            }

        }

        return -1;

    }

    public static void main(String args[]){

        int[] arr = {4,5,6,7,0,1,2};

        int k1 = 0;

        System.out.println(RevSearch(arr,k1));

        int[] arr1 = {4,5,6,7,0,1,2};

        int k2 = 3;

        System.out.println(RevSearch(arr1,k2));

    }

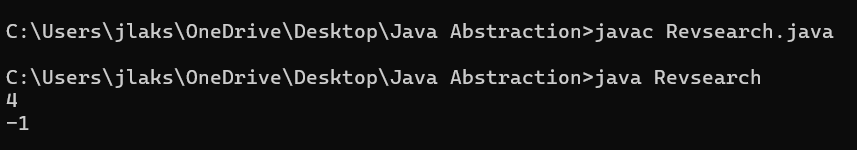
}

Input :

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

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

Output:



Time complexity: O(n)

Space complexity: O(1)

4. Container with Most Water

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

Output: 6

Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 \* 2 = 6

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

Output: 12

Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12

**CODE:**

public class mostwater {

    public static int mostWater(int[] arr){

        int left =0;

        int right =arr.length-1;

        int current=0;

        int maxarea=0;

        while(left<right){

            current=Math.min(arr[left],arr[right])\*(right-left);

            if(arr[left]<arr[right]){

                left++;

            }

            else{

                right--;

            }

            maxarea=Math.max(current,maxarea);

        }

        return maxarea;

    }

    public static void main(String args[]){

        int[] arr = {1,5,4,3};

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

        int[] arr1 = {3,1,2,4,5};

        System.out.println(mostWater(arr1));

    }

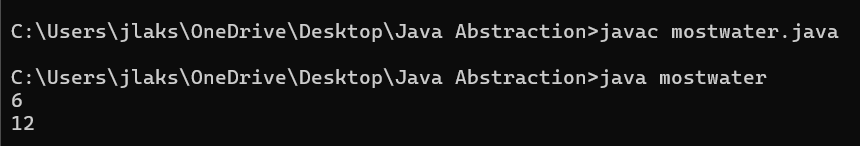
}

Input:

arr = [1, 5, 4, 3]

arr = [3, 1, 2, 4, 5]

Output:



Time complexity: O(n)

Space complexity: O(1)

5. Find the Factorial of a large number

Input: 100

Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00 Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**CODE:**

import java.math.BigInteger;

public class longfact {

    public static BigInteger longfactorial(int a){

        BigInteger fact = BigInteger.ONE;

        for (int i=1;i<=a;i++){

            fact = fact.multiply(BigInteger.valueOf(i));

        }

        return fact;

    }

    public static void main(String args[]){

        int a = 100;

        System.out.println(longfactorial(a));

        int b = 50;

        System.out.println(longfactorial(b));

    }

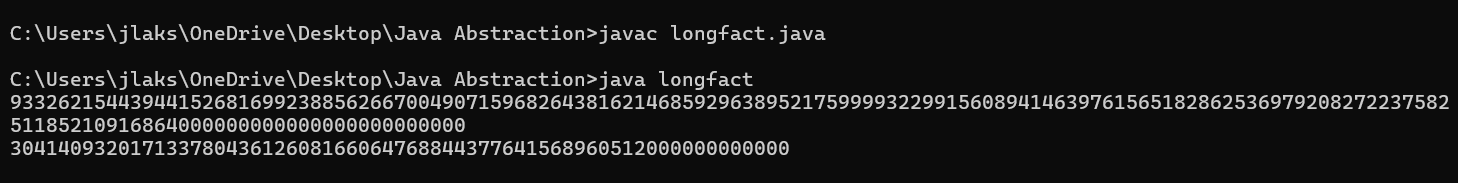
}

Input:

100

50

Output:



Time complexity: O(n^2log(n))

Space complexity: O(nlogn)

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

**CODE:**

class traprainwater {

    public static void main(String[] args) {

        int[] arr = {3, 0, 1, 0, 4, 0, 2};

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

            }

    public static int trap(int[] height) {

        int left = 0;

        int right = height.length - 1;

        int leftMax = height[left];

        int rightMax = height[right];

        int water = 0;

        while (left < right) {

            if (leftMax < rightMax) {

                left++;

                leftMax = Math.max(leftMax, height[left]);

                water += leftMax - height[left];

            } else {

                right--;

                rightMax = Math.max(rightMax, height[right]);

                water += rightMax - height[right];

            }

        }

        return water;

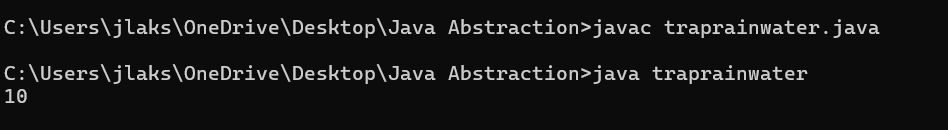
    }

}

Input:

arr[] = {3, 0, 1, 0, 4, 0, 2}

Output:



Time complexity: O(n)  
Space complexity: O(1)

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 chocolatedistribute {

    public static int distribute(int[] arr,int m){

        int[] sub1 = Arrays.copyOfRange(arr, 0, m);

        Arrays.sort(sub1);

        int a = sub1[m-1]-sub1[0];

        for(int i=0; i<arr.length-m;i++){

            int[] sub = Arrays.copyOfRange(arr, i, i+m);

            Arrays.sort(sub);

            a=Math.min(a,sub[sub.length-1]-sub[0]);

        }

        return a;

    }

    public static void main(String args[]){

        int[] arr = {7, 3, 2, 4, 9, 12, 56};

        int m=3;

        System.out.println(distribute(arr,m));

        int[] arr1 = {7, 3, 2, 4, 9, 12, 56};

        int n =5;

        System.out.println(distribute(arr1,n));

    }

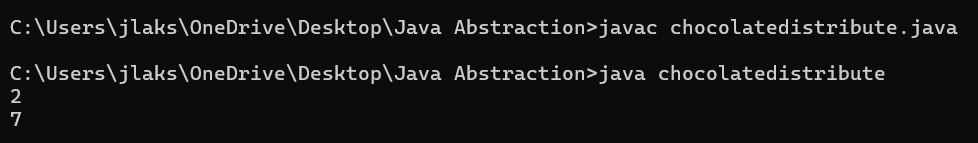
}

Input:

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

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

Output:



Time complexity: O(n\*mlogm)

Space complexity: O(m)

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.\*;

public class IntervalMerger {

    public static List<int[]> mergeIntervals(int[][] intervals) {

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

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

            if (result.isEmpty()) {

                result.add(new int[]{intervals[i][0], intervals[i][1]});

            } else {

                int[] lastInterval = result.get(result.size() - 1);

                if (intervals[i][0] <= lastInterval[1]) {

                    lastInterval[1] = Math.max(lastInterval[1], intervals[i][1]);

                } else {

                    result.add(new int[]{intervals[i][0], intervals[i][1]});

                }

            }

        }

        return result;

    }

    public static void main(String[] args) {

        int[][] intervals = {{1, 3}, {2, 6}, {8, 10}, {15, 18}};

        List<int[]> mergedIntervals = mergeIntervals(intervals);

        for (int[] interval : mergedIntervals) {

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

        }

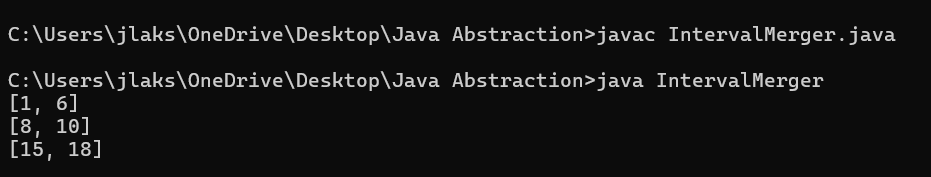
    }

}

Input:

arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]

Output:



Time complexity: O(nlogn)

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.HashSet;

public class booleanmatrix{

    public static void main(String[] args) {

        int[][] a = {{1, 0}, {0, 0}};

        int[][] result = spirals(a);

        for (int[] row : result) {

            for (int val : row) {

                System.out.print(val + " ");

            }

            System.out.println();

        }

    }

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

        HashSet<Integer> rows = new HashSet<>();

        HashSet<Integer> cols = new HashSet<>();

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

            for (int j = 0; j < matrix[0].length; j++) {

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

                    rows.add(i);

                    cols.add(j);

                }

            }

        }

        for (int i : rows) {

            for (int j = 0; j < matrix[0].length; j++) {

                matrix[i][j] = 1;

            }

        }

        for (int j : cols) {

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

                matrix[i][j] = 1;

            }

        }

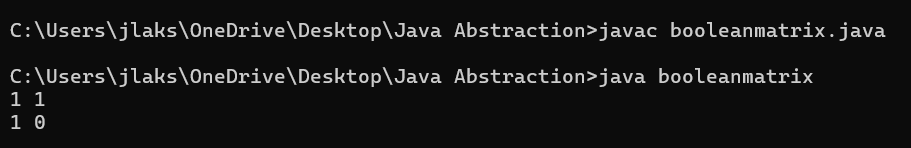
        return matrix;

    }

}

Input: {{1, 0}, {0, 0}} ;

Output:



Time complexity: O(m\*n)

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:**

import java.util.\*;

public class spiralmatrix {

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

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

        if (matrix.length == 0) {

            return result;

        }

        int top = 0;

        int bottom = matrix.length - 1;

        int left = 0;

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

        while (top <= bottom && left <= right) {

            for (int i = left; i <= right; i++) {

                result.add(matrix[top][i]);

            }

            top++;

            for (int i = top; i <= bottom; i++) {

                result.add(matrix[i][right]);

            }

            right--;

            if (top <= bottom) {

                for (int i = right; i >= left; i--) {

                    result.add(matrix[bottom][i]);

                }

                bottom--;

            }

            if (left <= right) {

                for (int i = bottom; i >= top; i--) {

                    result.add(matrix[i][left]);

                }

                left++;

            }

        }

        return result;

    }

    public static void main(String[] args) {

        spiralmatrix sol = new spiralmatrix();

        int[][] matrix = {

            {1, 2, 3},

            {4, 5, 6},

            {7, 8, 9}

        };

        System.out.println(sol.spiralOrder(matrix));

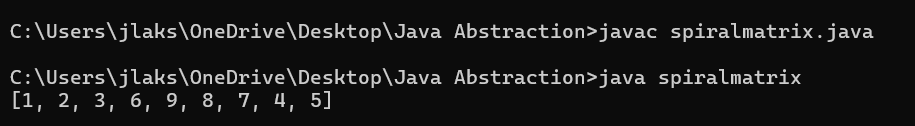
    }

}

Input:

matrix = { {1, 2, 3},{4, 5, 6},{7, 8, 9}};

Output:



Time complexity: O(m\*n)

Space 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:**

public class parantheses {

    public static void main(String[] args) {

        String a = "((()))()()";

        paran(a);

        String b = "“())((())";

        paran(b);

    }

    public static void paran(String s){

        int a=0;

        boolean flag = false;

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

            if(s.charAt(i)=='('){

                a++;

            }

            else{

                a--;

                if (a<0){

                    flag=true;

                }

            }

        }

        System.out.println(flag?"Not Balanced":"Balanced");

    }

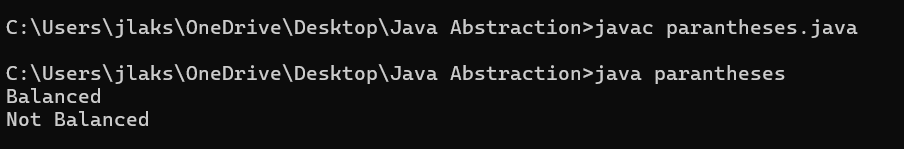
}

Input:

str = “((()))()()”

str = “())((())”

Output:



Time complexity: O(n)

Space complexity: O(1)

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:**

public class anagram {

    public static void main(String[] args) {

        Anagram("geeks","skeeg");

        Anagram("allergy","allergic");

    }

    public static void Anagram(String a,String b){

        if (a.length()!=b.length()){

            System.out.println("Not an anagram");

        }

        else{

            int[] arr = new int[26];

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

                arr[a.charAt(i)-'a']++;

                arr[b.charAt(i)-'a']--;

            }

            boolean flag = false;

            for(int i=0;i<26;i++){

                if (arr[i]!=0){

                    flag=true;

                    break;

                }

            }

            System.out.println(flag?"Not an anagram":"Anagram");

        }

    }

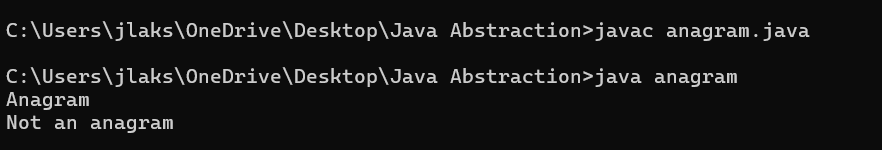
}

Input:

s1 = “geeks” ,s2 = “kseeg” ;

s1 = “allergy” ,s2 = “allergic”;

Output:



Time complexity: O(n)

Space complexity: O(1)

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”

**CODE:**

public class longestsubstrpalindrome {

    static boolean longestpalsubstr(String s, int left, int right) {

        while (left < right) {

            if (s.charAt(left) != s.charAt(right))

                return false;

            left++;

            right--;

        }

        return true;

    }

    static String LongestPalSubstr(String s) {

        int n = s.length();

        int maximumlength = 1, start = 0;

        for (int i = 0; i < n; i++) {

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

                if (longestpalsubstr(s, i, j) && (j - i + 1) > maximumlength) {

                    start = i;

                    maximumlength = j - i + 1;

                }

            }

        }

        return s.substring(start, start + maximumlength);

    }

    public static void main(String[] args) {

        String s = "forgeeksskeegfor";

        System.out.println(LongestPalSubstr(s));

        String s1 = "Geeks";

        System.out.println(LongestPalSubstr(s1));

        String s2 = "abc";

        System.out.println(LongestPalSubstr(s2));

    }

}

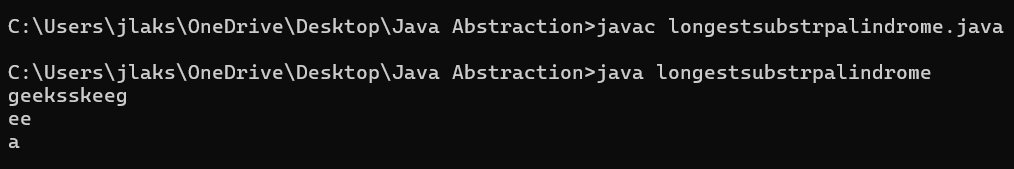
Input:

str = “forgeeksskeegfor”

str = “Geeks”

Input: str = “abc”

Output:



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.\*;

public class longestsubstring {

    public static void main(String[] args) {

        String[] a = {"geeksforgeeks", "geeks", "geek", "geezer"};

        substr(a);

        String[] b = {"hello","world"};

        substr(b);

    }

    public static void substr(String[] s){

        if(s.length==0){

            System.out.println(-1);

        }

        else{

            Arrays.sort(s);

            String start = s[0];

            String end = s[s.length-1];

            int length = Math.min(start.length(),end.length());

            int i = 0;

            while(i<length && start.charAt(i)==end.charAt(i)){

                i++;

            }

            if(i==0){

                System.out.println(-1);

            }

            else{

                System.out.println(start.substring(0,i));

            }

        }

    }

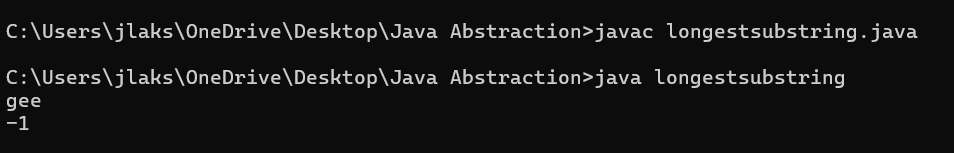
}

Input:

arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

: arr[] = [“hello”, “world”]

Output:



Time complexity: O(nlogn\*m)

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] I

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

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

**CODE:**

import java.util.\*;

public class deletestackmid {

    public static void main(String[] args) {

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

        st.push('1');

        st.push('2');

        st.push('3');

        st.push('4');

        st.push('5');

        st.push('6');

        st.push('7');

        delmid(st);

    }

    public static void delmid(Stack<Character> st){

            Vector<Character> v = new Vector<Character>();

            while(!st.empty()){

                v.add(st.pop());

            }

            int a = v.size();

            if(a%2==0){

                int b = a/2;

                for(int i=0;i<a;i++){

                    if(i!=b){

                        st.push(v.get(i));

                    }

                }

            }

            else{

                int b = (int) Math.ceil(a/2);

                for(int i=0;i<a;i++){

                    if(i!=b){

                        st.push(v.get(i));

                    }

                }

            }

            while (!st.empty()) {

                char p = st.pop();

            System.out.print(p + " ");

        }

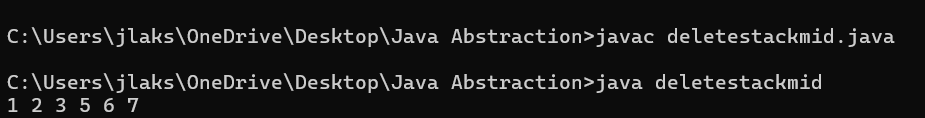
    }

}

Input :

Stack[] = [1, 2, 3, 4, 5, 6,7]

Output:



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, 2 –> 5 –> 25 –> 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 –> 7, -1 –> 12, 6 –> 12,12 -> -1

**CODE:**

import java.util.Stack;

import java.util.Arrays;

public class nextgreatestelement{

    public static void main(String[] args) {

        int[] arr = {4, 5, 2, 25};

        System.out.println(Arrays.toString(great(arr)));

        int[] arr1 = {13, 7, 6, 12};

        System.out.println(Arrays.toString(great(arr1)));

    }

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

        int n = arr.length;

        int[] arr1 = new int[n];

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

        for (int i = n - 1; i >= 0; i--) {

            while (!st.isEmpty() && arr[i] >= st.peek()) {

                st.pop();

            }

            if (st.isEmpty()) {

                arr1[i] = -1;

            } else {

                arr1[i] = st.peek();

            }

            st.push(arr[i]);

        }

        return arr1;

    }

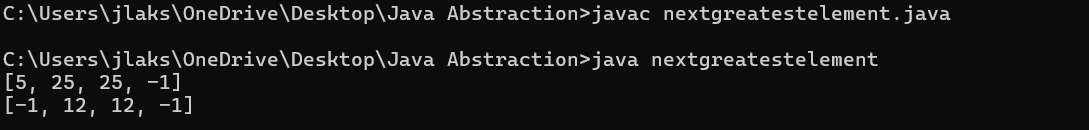
}

Input:

arr[] = [ 4 , 5 , 2 , 25 ]

arr1[] = [13, 7, 6, 12]

Output:



Time complexity: O(n)

Space 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.

**CODE:**

import java.util.\*;

class Node{

    int data;

    Node left,right;

    Node(int x){

        data=x;

        left=right=null;

    }

}

public class rightview{

    public static void main(String[] args) {

        Node root = new Node(1);

        root.left= new Node(2);

        root.right= new Node(3);

        root.right.left= new Node(4);

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

        ArrayList<Integer> result=height(root);

        printarray(result);

    }

    public static void RecursiveRightView(Node root,int level,int[] maxlevel, ArrayList<Integer> result){

        if (root==null){

            return;

        }

        if (level>maxlevel[0]){

            result.add(root.data);

            maxlevel[0]=level;

        }

        RecursiveRightView(root.right,level+1,maxlevel,result);

        RecursiveRightView(root.left,level+1,maxlevel,result);

    }

    public static ArrayList<Integer> height(Node root) {

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

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

        RecursiveRightView(root,0,maxlevel,result);

        return result;

    }

    public static void printarray(ArrayList<Integer> arr){

        for(int i:arr){

            System.out.println(i+" ");

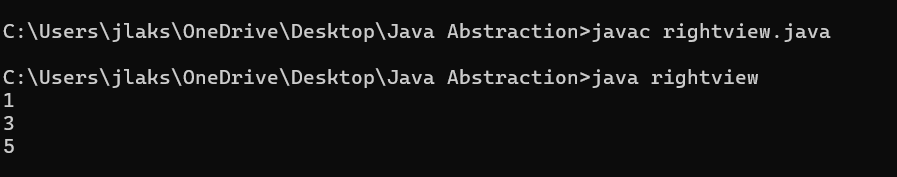
        }

        System.out.println();

    }

}

Output:



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.

**CODE:**

class Node{

    int data;

    Node left,right;

    Node(int x){

        data = x;

        left=right=null;

    }

}

public class heightoftree {

    public static void main(String[] args) {

        Node root = new Node(1);

        root.left= new Node(2);

        root.right= new Node(3);

        root.right.left= new Node(4);

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

        System.out.println(depth(root));

    }

    public static int depth(Node node){

        if (node==null){

            return 0;

        }

        int leftdepth = depth(node.left);

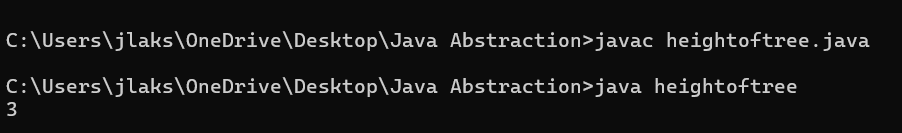
        int rightdepth = depth(node.right);

        return Math.max(leftdepth,rightdepth)+1;

    }

}

Output:



Time complexity: O(n)

Space complexity: O(n)