TUMU LAVA KARTEEK: AI&DS-DSA-Practice-1

Date: 9/11/2024

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**

import java.util.\*;

class kadane {

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

int n=sc.nextInt();

int[] nums= new int[n];

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

nums[i]=sc.nextInt();

}

if (n<0){

System.out.println(0);

}

int maxi=Integer.MIN\_VALUE;

int s=0;

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

s+=nums[i];

if(s>maxi){

maxi=s;

}

if(s<0){

s=0;

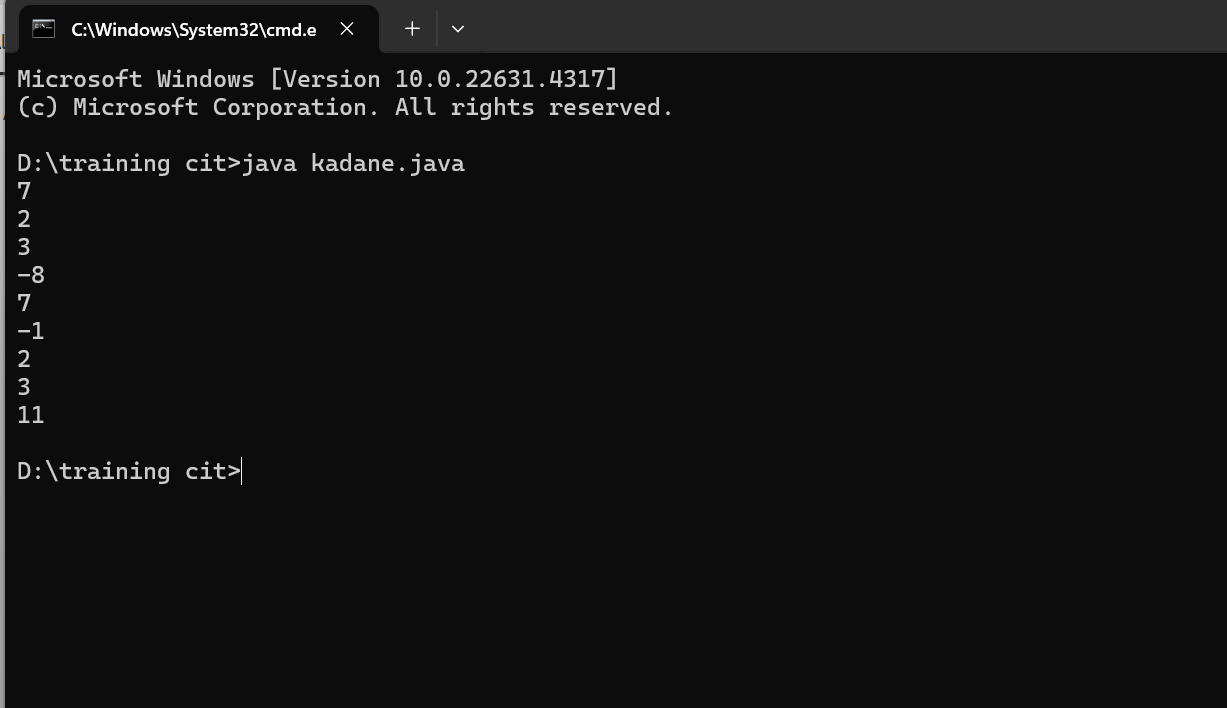
}

}

System.out.println(maxi);

}

}



TIME COMPLEXITY:

O(n)

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**

import java.util.\*;

class maximumprodsubray{

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

int n=sc.nextInt();

int[] nums= new int[n];

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

nums[i]=sc.nextInt();

}

int maxi=Integer.MIN\_VALUE;

int prod=1;

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

prod\*=nums[i];

if(maxi<prod){

maxi=prod;

}

if(prod==0){

prod=1;

}

}

prod=1;

for(int i=nums.length-1;i>=0;i--){

prod\*=nums[i];

if(maxi<prod){

maxi=prod;

}

if(prod==0){

prod=1;

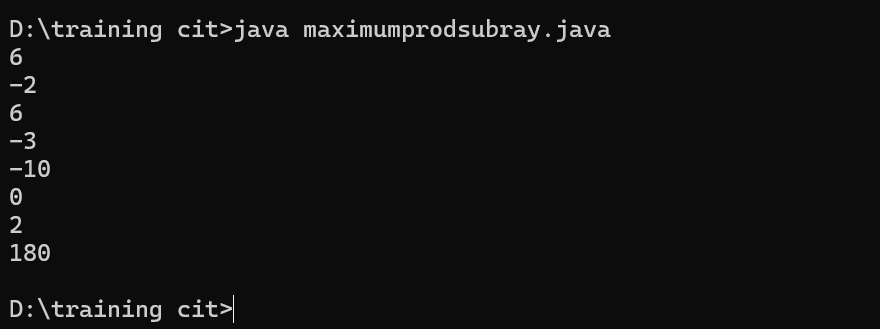
}

}

System.out.println( maxi);

}

}



TIME COMPLEXITY:

O(n)

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

class Solution {

public:

int search(vector<int>& nums, int target) {

int low=0;

int n=nums.size();

int high=n-1;

while(low<=high){

int mid=low+(high-low)/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;

}

};

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**

import java.util.\*;

class rotated{

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

int n=sc.nextInt();

int[] nums= new int[n];

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

nums[i]=sc.nextInt();

}

int target=sc.nextInt();

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

if(nums[i]==target){

System.out.println(i%nums.length);

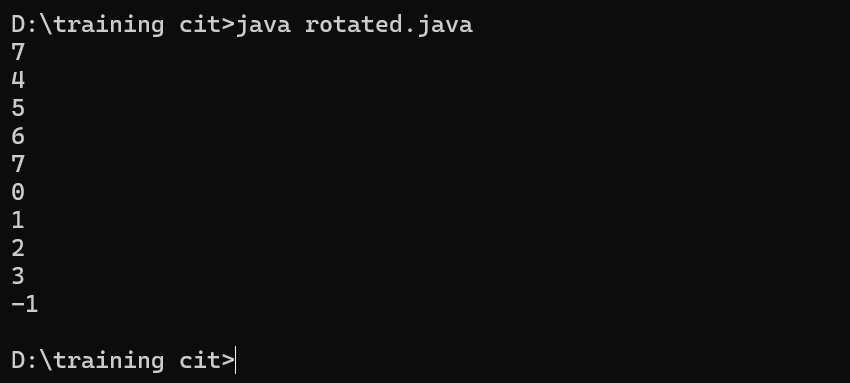
}

}

System.out.println(-1);

}

}



TIME COMPLEXITY:  
O(n)

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**

import java.util.\*;

class contmost{

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

int n=sc.nextInt();

int[] height= new int[n];

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

height[i]=sc.nextInt();

}

int h,w,a;

int m=0;

int i=0;

int j=height.length-1;

while(i<j){

h=Math.min(height[i],height[j]);

w=j-i;

a=h\*w;

m=Math.max(m,a);

if(height[i]<height[j]){

i++;

}else{

j--;

}

}

System.out.println(m);

}

}  


TIME COMPLEXITY  
O(n)

5.

Find the Factorial of a large number

Input: 100

Output: 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**Code**

import java.util.\*;

import java.math.BigInteger;

class bigfact{

public BigInteger factorial (int fact){

BigInteger result=BigInteger.valueOf(fact);

if (fact>1){

result=result.multiply(factorial(fact-1));

}

return result;

}

public static void main(String [] args){

bigfact obj=new bigfact();

int num1=100;

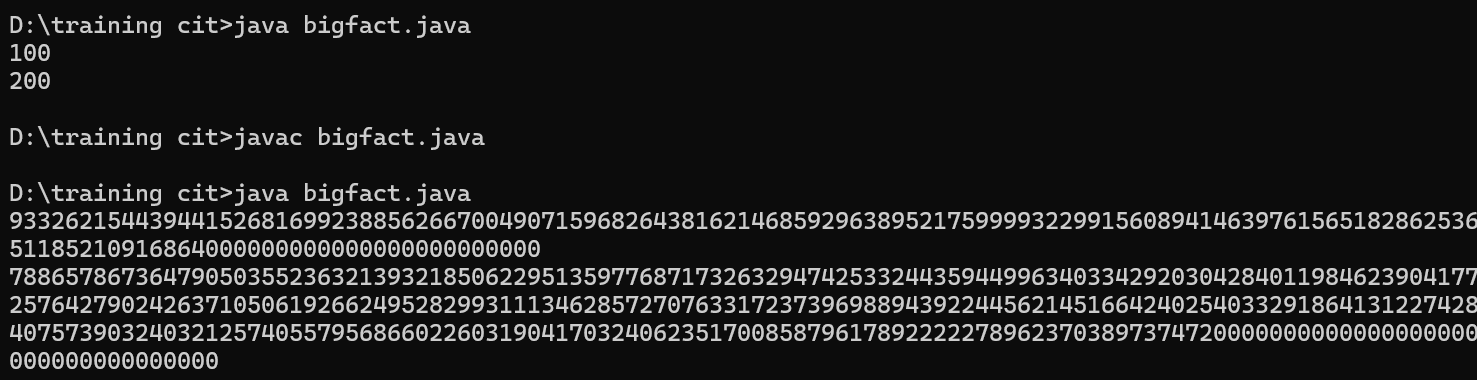
int num2=200;

System.out.println(obj.factorial(num1));

System.out.println(obj.factorial(num2));

}

}



TIME COMPLEXITY  
time complexity is dominated by the recursion depth, which is O(n)O(n)O(n). However, because of the BigInteger multiplication overhead, the effective time complexity could be considered closer to O(n⋅M(n))O(n \cdot M(n))O(n⋅M(n)), where M(n)M(n)M(n) represents the cost of multiplying large numbers as they grow in size.

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 six {

public static int trap(int[] arr) {

int n = arr.length;

if (n < 3) {

return 0;

}

int[] leftMax = new int[n];

int[] rightMax = new int[n];

leftMax[0] = arr[0];

rightMax[n - 1] = arr[n - 1];

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

leftMax[i] = Math.*max*(leftMax[i - 1], arr[i]);

}

for (int i = n - 2; i >= 0; i--) {

rightMax[i] = Math.*max*(rightMax[i + 1], arr[i]);

}

int water = 0;

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

water += Math.*min*(leftMax[i], rightMax[i]) - arr[i];

}

return water;

}

public static void main(String[] args) {

int[] arr1 = {3, 0, 1, 0, 4, 0, 2};

System.***out***.println(*trap*(arr1));

int[] arr2 = {3, 0, 2, 0, 4};

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

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

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

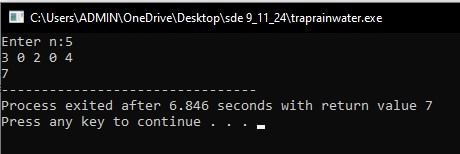
int[] arr4 = {10, 9, 0, 5};

System.***out***.println(*trap*(arr4));

}

}

Output:



TIME 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.\*;

public class seven {

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

int n = arr.length;

if (m == 0 || n == 0) {

return 0;

}

Arrays.*sort*(arr);

if (n < m) {

return -1;

}

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(*findMinDiff*(arr1, m1));

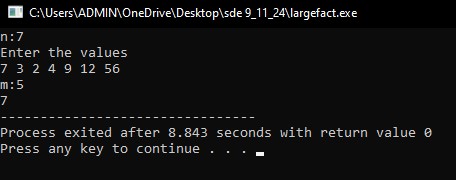
int[] arr2 = {7, 3, 2, 4, 9, 12, 56};

int m2 = 5;

System.*out*.println(*findMinDiff*(arr2, m2));

}

}

OUTPUT  


TIME COMPLEXITY  
O(N log(N))

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 eight {

public static int[][] mergeIntervals(int[][] intervals) {

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

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

for (int[] interval : intervals) {

if (merged.isEmpty() || merged.get(merged.size() - 1)[1] < interval[0]) {

merged.add(interval);

} else {

merged.get(merged.size() - 1)[1] = Math.*max*(merged.get(merged.size() - 1)[1], interval[1]);

}

}

return merged.toArray(new int[merged.size()][]);

}

public static void printIntervals(int[][] intervals) {

for (int[] interval : intervals) {

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

}

System.*out*.println();

}

public static void main(String[] args) {

int[][] arr1 = {

{1, 3}, {2, 4}, {6, 8}, {9, 10}

};

int[][] result1 = *mergeIntervals*(arr1);

*printIntervals*(result1);

int[][] arr2 = {

{7, 8}, {1, 5}, {2, 4}, {4, 6}

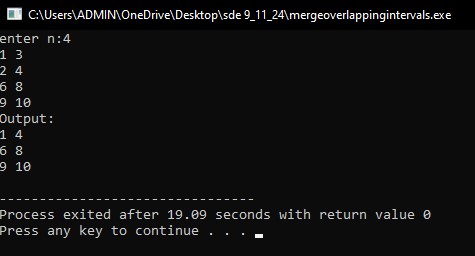
};

int[][] result2 = *mergeIntervals*(arr2);

*printIntervals*(result2);

}

}

OUTPUT  


TIME COMPLEXITY  
O(N log(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 nine {

public static void modifyMatrix(int[][] mat) {

int M = mat.length;

int N = mat[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 (mat[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]) {

mat[i][j] = 1;

}

}

}

}

public static void printMatrix(int[][] mat) {

for (int[] row : mat) {

System.***out***.println(Arrays.*toString*(row));

}

System.***out***.println();

}

public static void main(String[] args) {

int[][] mat1 = {

{1, 0},

{0, 0}

};

*modifyMatrix*(mat1);

*printMatrix*(mat1);

int[][] mat2 = {

{0, 0, 0},

{0, 0, 1}

};

*modifyMatrix*(mat2);

*printMatrix*(mat2);

int[][] mat3 = {

{1, 0, 0, 1},

{0, 0, 1, 0},

{0, 0, 0, 0}

};

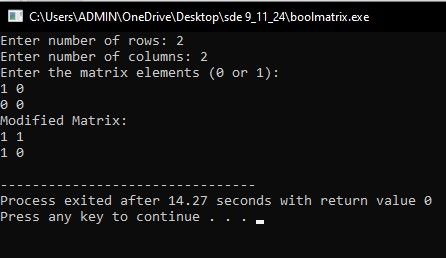
*modifyMatrix*(mat3);

*printMatrix*(mat3);

}

}

OUTPUT



Time 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 ten {

public static void printSpiral(int[][] matrix) {

if (matrix.length == 0) {

return;

}

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++) {

System.*out*.print(matrix[top][i] + " ");

}

top++;

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

System.*out*.print(matrix[i][right] + " ");

}

right--;

if (top <= bottom) {

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

System.*out*.print(matrix[bottom][i] + " ");

}

bottom--;

}

if (left <= right) {

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

System.*out*.print(matrix[i][left] + " ");

}

left++;

}

}

}

public static void main(String[] args) {

int[][] matrix1 = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

System.*out*.println("Spiral Order of matrix1:");

*printSpiral*(matrix1);

System.*out*.println();

int[][] matrix2 = {

{1, 2, 3, 4, 5, 6},

{7, 8, 9, 10, 11, 12},

{13, 14, 15, 16, 17, 18}

};

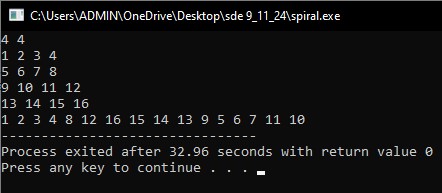
System.*out*.println("Spiral Order of matrix2:");

*printSpiral*(matrix2);

}

}

OUTPUT



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.\*;

public class thirteen {

public boolean check(String s) {

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

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

if (c == '(') {

st.push(c);

} else {

if (st.isEmpty()) {

return false;

}

if ((st.peek() == '(')) {

st.pop();

} else {

return false;

}

}

}

return st.isEmpty();

}

public static void main(String[] args) {

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

String s;

s = sc.nextLine();

thirteen o = new thirteen();

Boolean answer = o.check(s);

if(answer) {

System.***out***.println("balanced");

}else {

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

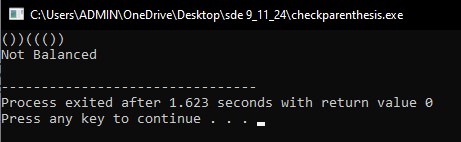
}

sc.close();

}

}

OUTPUT

****

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**

import java.util.\*;

public class fourteen {

public boolean check(String s, String t) {

char[] a = s.toCharArray();

char[] b = t.toCharArray();

Arrays.*sort*(a);

Arrays.*sort*(b);

return Arrays.*equals*(a, b);

}

public static void main(String[] args) {

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

String s, t;

s = sc.nextLine();

t = sc.nextLine();

fourteen o = new fourteen();

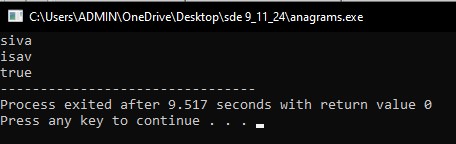
System.***out***.println(o.check(s, t));

sc.close();

}

}

OUTPUT

****

Time complexity

O(n log(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 fifteen {

public static String longestPalindromicSubstring(String str) {

if (str == null || str.length() < 1) {

return "";

}

int start = 0, end = 0;

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

int len1 = *expandAroundCenter*(str, i, i);

int len2 = *expandAroundCenter*(str, i, i + 1);

int len = Math.*max*(len1, len2);

if (len > end - start) {

start = i - (len - 1) / 2;

end = i + len / 2;

}

}

return str.substring(start, end + 1);

}

public static int expandAroundCenter(String str, int left, int right) {

while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {

left--;

right++;

}

return right - left - 1;

}

public static void main(String[] args) {

String str1 = "forgeeksskeegfor";

System.***out***.println(str1);

System.***out***.println("Longest Palindromic Substring: " + *longestPalindromicSubstring*(str1));

System.***out***.println();

String str2 = "Geeks";

System.***out***.println(str2);

System.***out***.println("Longest Palindromic Substring: " + *longestPalindromicSubstring*(str2));

System.***out***.println();

String str3 = "abc";

System.***out***.println(str3);

System.***out***.println("Longest Palindromic Substring: " + *longestPalindromicSubstring*(str3));

System.***out***.println();

String str4 = "";

System.***out***.println(str4);

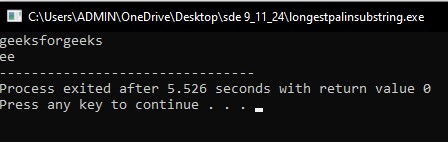
System.***out***.println("Longest Palindromic Substring: " + *longestPalindromicSubstring*(str4));

System.***out***.println();

}

}

OUTPUT

****

Time complexity

O(N2)

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 sixteen {

public static String result(String[] arr) {

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

return "-1";

}

Arrays.*sort*(arr);

String first = arr[0];

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

int i = 0;

while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {

i++;

}

String common = first.substring(0, i);

return common.isEmpty() ? "-1" : common;

}

public static void main(String[] args) {

String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};

for(String i: arr1) {

System.***out***.printf("%s ", i);

}

System.***out***.println();

System.***out***.println("Longest Common Prefix: " + *result*(arr1));

System.***out***.println();

String[] arr2 = {"hello", "world"};

for(String i: arr2) {

System.***out***.printf("%s ", i);

}

System.***out***.println();

System.***out***.println("Longest Common Prefix: " + *result*(arr2));

System.***out***.println();

String[] arr3 = {"saravanan", "sara", "sarav", "saro"};

for(String i: arr3) {

System.***out***.printf("%s ", i);

}

System.***out***.println();

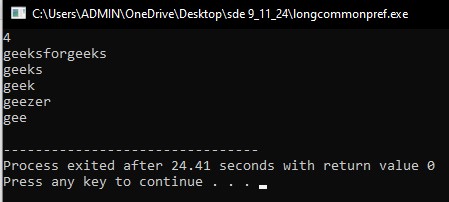
System.***out***.println("Longest Common Prefix: " + *result*(arr3));

System.***out***.println();

}

}

OUTPUT



Time complexity

O(N log(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**

import java.util.\*;

public class seventeen {

public static void result(Stack<Integer> stack) {

int m = stack.size() / 2;

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

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

temp.push(stack.pop());

}

stack.pop();

while (!temp.isEmpty()) {

stack.push(temp.pop());

}

}

public static void main(String[] args) {

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

int n;

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

System.***out***.println("ENTER THE STACK SIZE");

n = sc.nextInt();

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

st.push(sc.nextInt());

}

sc.close();

System.***out***.println("stack before deletion:");

System.***out***.println(st);

*result*(st);

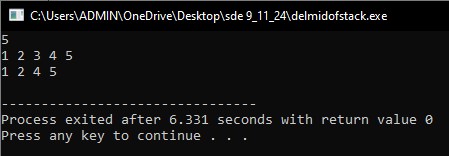
System.***out***.println("stack after deletion:");

System.***out***.println(st);

}

}

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

**Code**

package binary;

import java.util.\*;

public class eighteen {

public static void main(String[] args) {

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

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

int n;

System.***out***.println("enetr size");

n = sc.nextInt();

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

array.add(sc.nextInt());

}

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

boolean flag = true;

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

if(array.get(j) > array.get(i)) {

flag = false;

System.***out***.printf("%s -> %s%n", array.get(i), array.get(j));

break;

}

}

if(flag) {

System.***out***.printf("%s -> %s%n", array.get(i), -1);

}

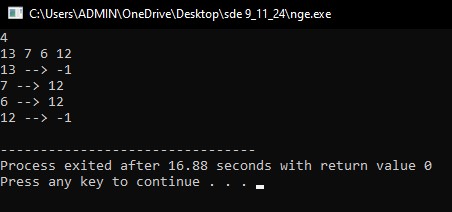
}

sc.close();

}

}

OUTPUT



Time complexity

O(N2)

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**

class Node{

int data;

Node left;

Node right;

public Node(int d){

this.data = d;

this.left = null;

this.right = null;

}

}

public class nineteen{

public void Traverse(Node root, boolean isRight){

if(root == null){

return;

}

if(isRight) { System.***out***.println(root.data); }

else if(root.right != null) { System.***out***.println(root.data); }

Traverse(root.left, false);

Traverse(root.right, true);

}

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);

Node root1 = new Node(1);

root1.left = new Node(2);

root1.right = new Node(3);

root1.left.left = new Node(4);

root1.left.left.right = new Node(5);

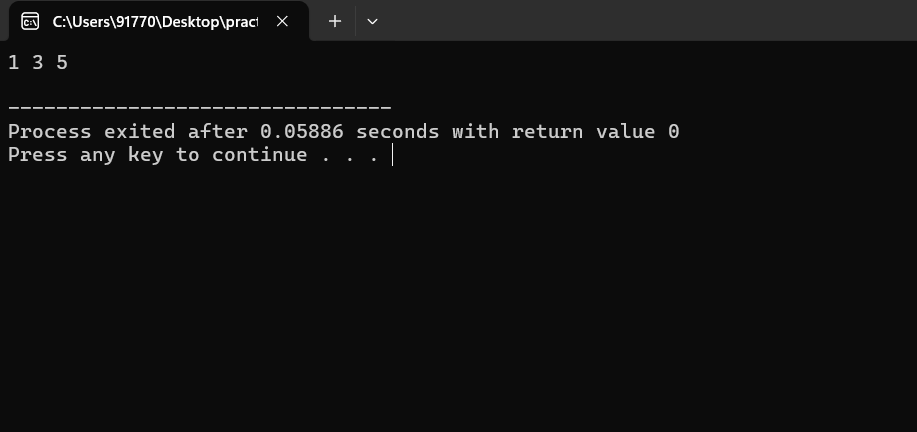
binaryTree nineteen = new binaryTree();

nineteen.Traverse(root, true);

}

}

OUTPUT



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.

**Code**

class Node{

int data;

Node left;

Node right;

Node(int d){

this.data = d;

this.left = null;

this.right = null;

}

}

public class twenty {

public int traverse(Node root) {

if(root == null) {

return 0;

}

int left = traverse(root.left);

int right = traverse(root.right);

return Math.*max*(left, right) + 1;

}

public static void main(String[] args) {

Node root = new Node(12);

root.left = new Node(8);

root.right = new Node(18);

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

root.left.right = new Node(11);

Node root1 = new Node(1);

root1.left = new Node(2);

root1.left.left = new Node(4);

root1.right = new Node(3);

root1.right.right = new Node(5);

root1.right.right.right = new Node(5);

root1.right.right.left = new Node(5);

twenty o = new twenty();

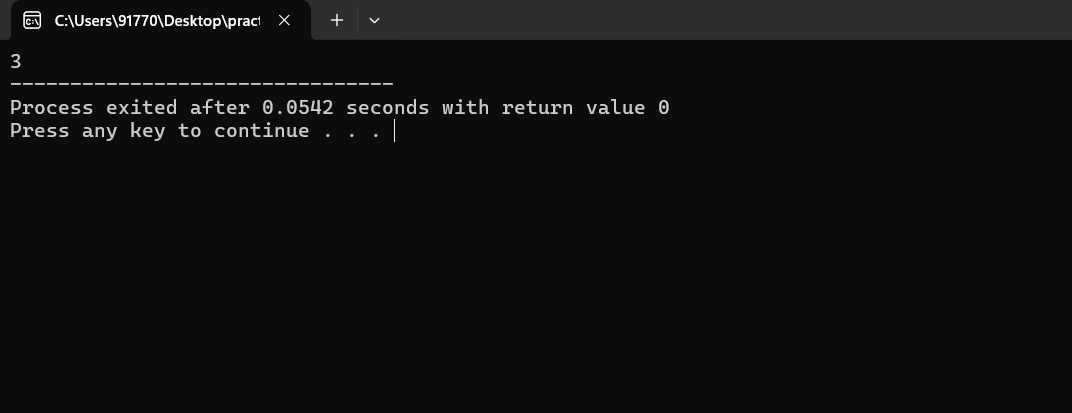
System.***out***.println(o.traverse(root));

System.***out***.println(o.traverse(root1));

}

}

OUTPUT



Time complexity

O(N)