**Date- 09/11/2024** **Coding practice Problems:**

**1.QUESTION**

**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

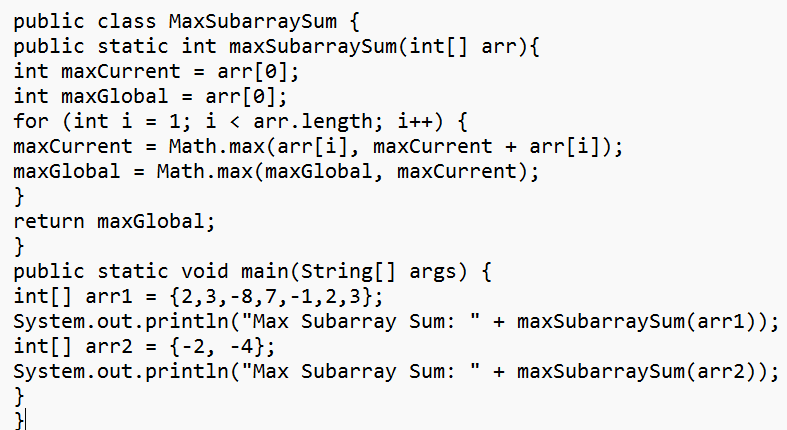
Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

Input: arr[] = {-2, -4}

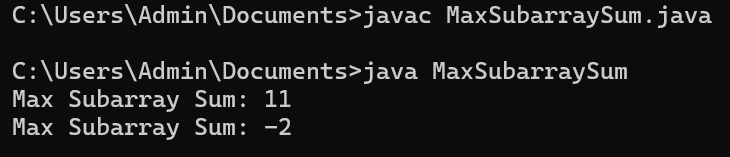
Output: –2

Explanation: The subarray {-2} has the largest sum -2.

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n)** - maxSubarraySum method has a single for loop that iterates through each element of the array exactly once.

**2.QUESTION**

**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)

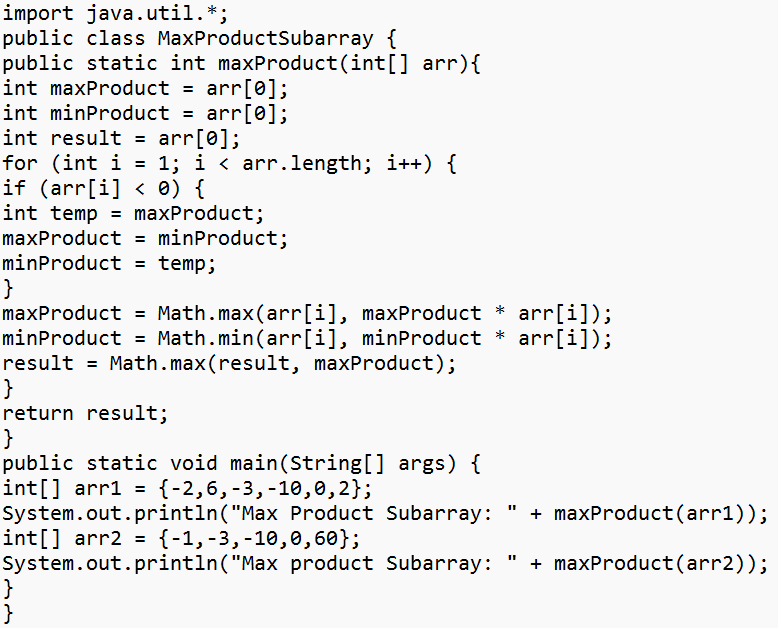
= 180

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

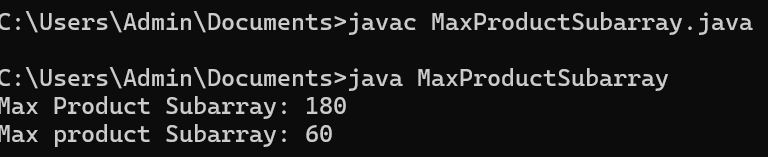
Output: 60

Explanation: The subarray with maximum product is {60}.

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY :**

**O(n) -** The code has a for loop that iterates through each element of the array exactly once, starting from the second element up to the last element.

**3) QUESTION**

**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

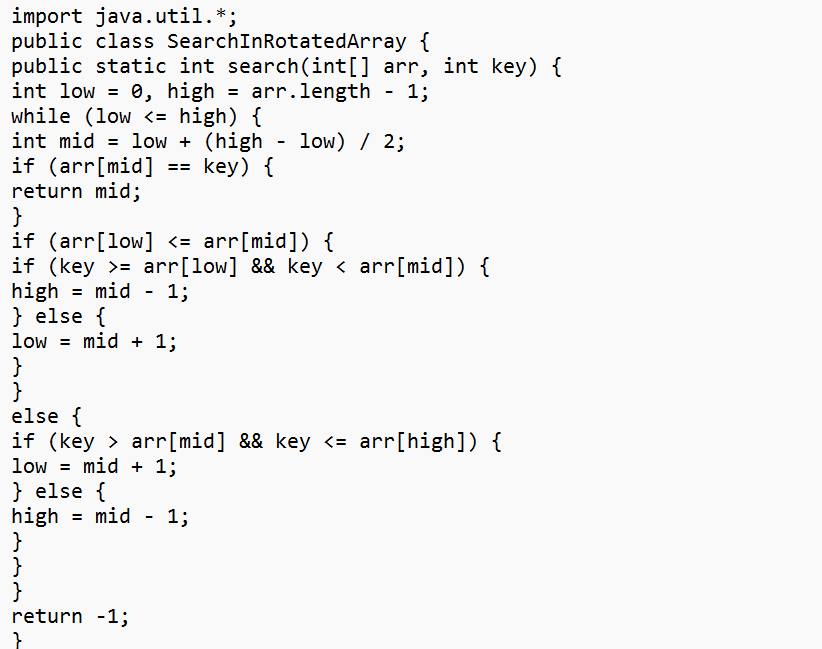
Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3

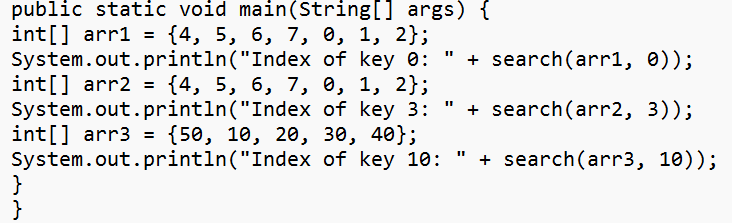
Output : -1

Input : arr[] = {50, 10, 20, 30, 40}, key = 10

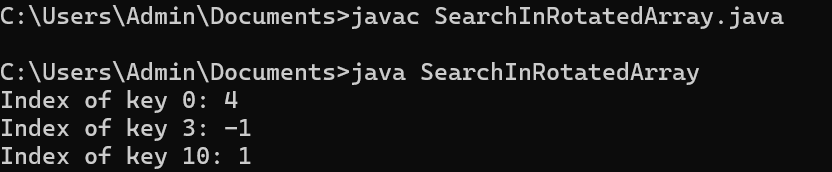
Output : 1

**SOLUTION**

****

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(log n) -** The algorithm uses binary search, dividing the search space by half each time.

**4) QUESTION**

**Container with Most Water**

Given n non-negative integers a1, a2,..., where each represents a point at coordinate (i, a,). 'n' vertical lines are drawn such that the two endpoints of line i is at (i, a) 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]

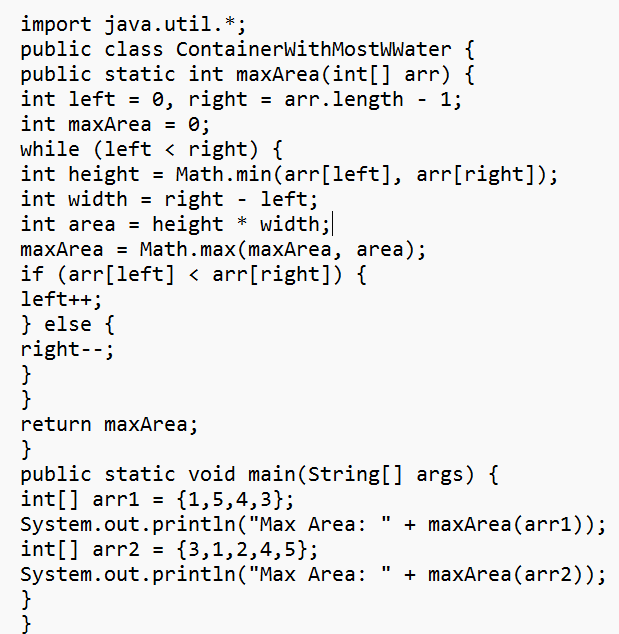
Output: 12

Explanation:

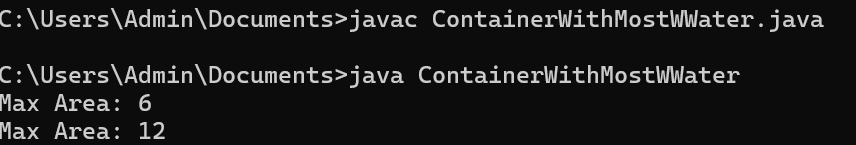
5 and 3 are distance 4 apart. So the size of the base = 4.

Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n)-** The algorithm uses a single loop with two pointers moving towards each other.

**5) QUESTION**

**Find the Factorial of a large number**

Input: 100

Output:

933262154439441526816992388562667004907159682643816214685929638952175999932299

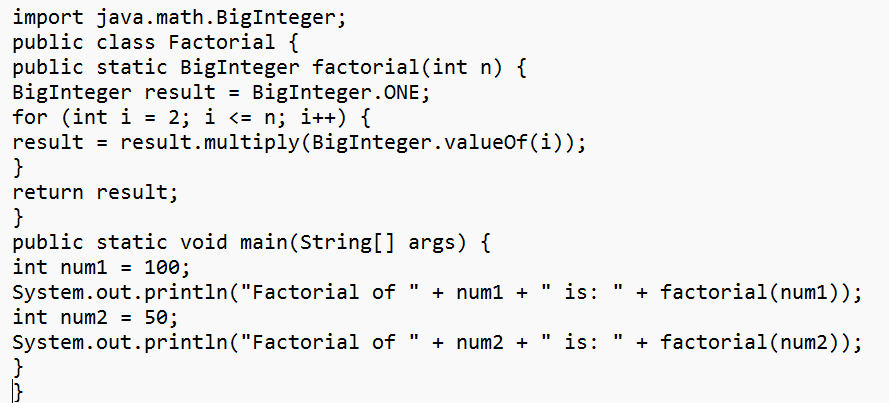
156089414639761565182862536979208272237582511852109168640000000000000000000000

00

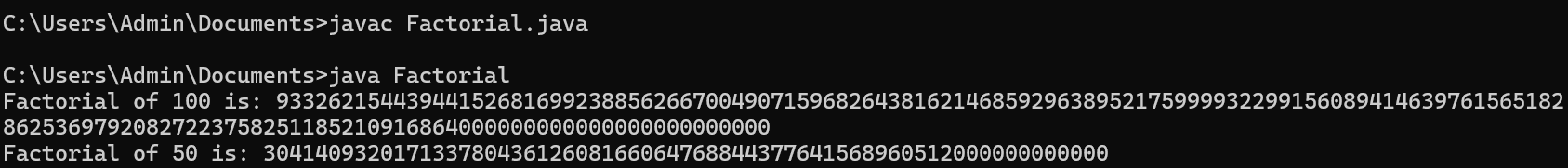
Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n) -** The algorithm performs a simple loop from 2 to n, where each step involves multiplying a BigInteger with a constant number, so the time complexity is linear in terms of the input size.

**6) QUESTION**

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

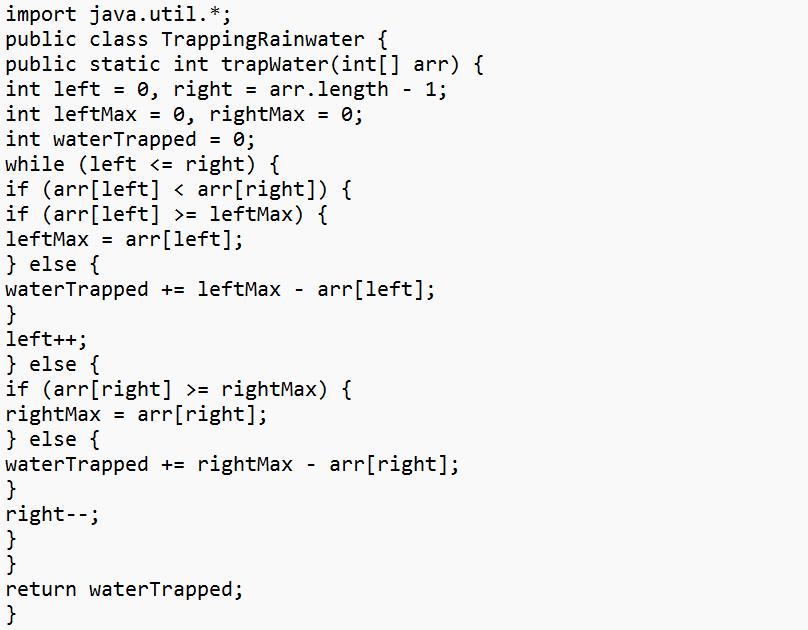
Explanation : We cannot trap water as there is no height bound on both sides

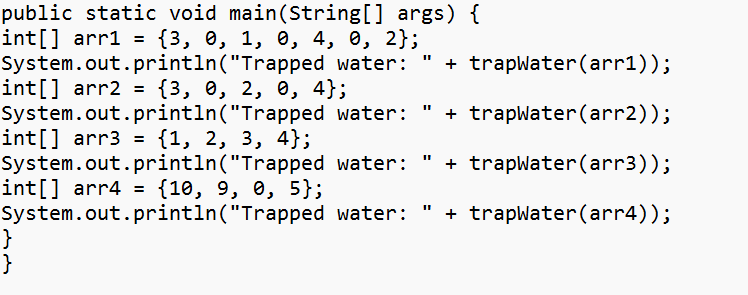
Input: arr[] = {10, 9, 0, 5}

Output: 5

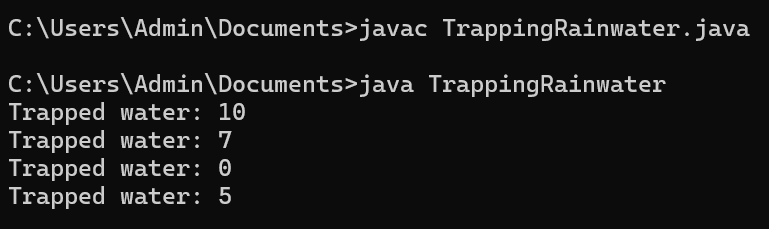
Explanation : We trap 0 + 0 + 5 + 0 = 5

**SOLUTION**

****

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n)-** We traverse the array with two pointers, each moving from opposite ends towards the center, resulting in linear time complexity.

**7) QUESTION**

**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.

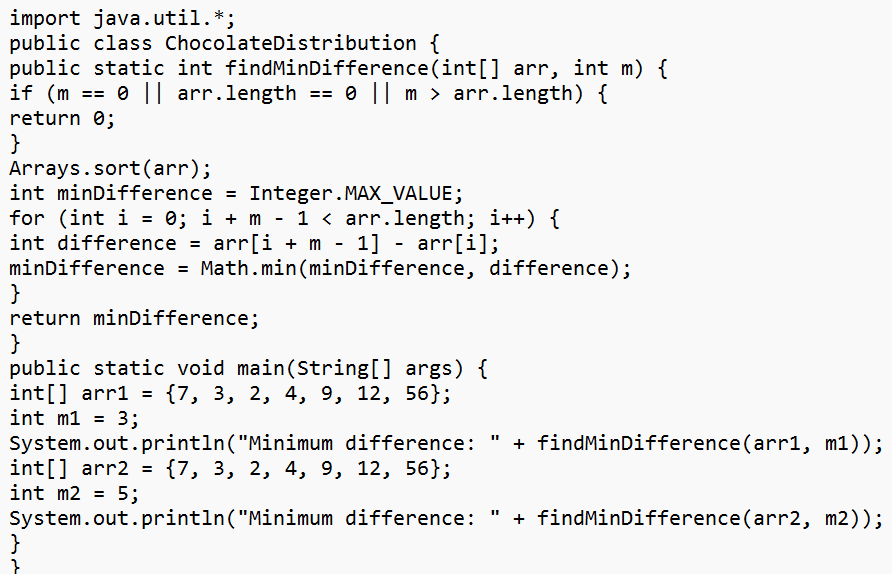
Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5

Output: 7

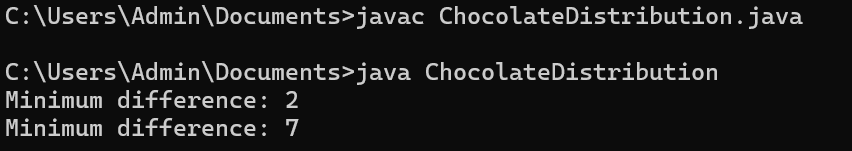
Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum

difference, that is 9 – 2 = 7.

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**Sorting: O(nlogn),** where n is the number of packets.

**Sliding Window: O(n−m),** which is approximately O(n) for large n.

**8) QUESTION**

**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]].

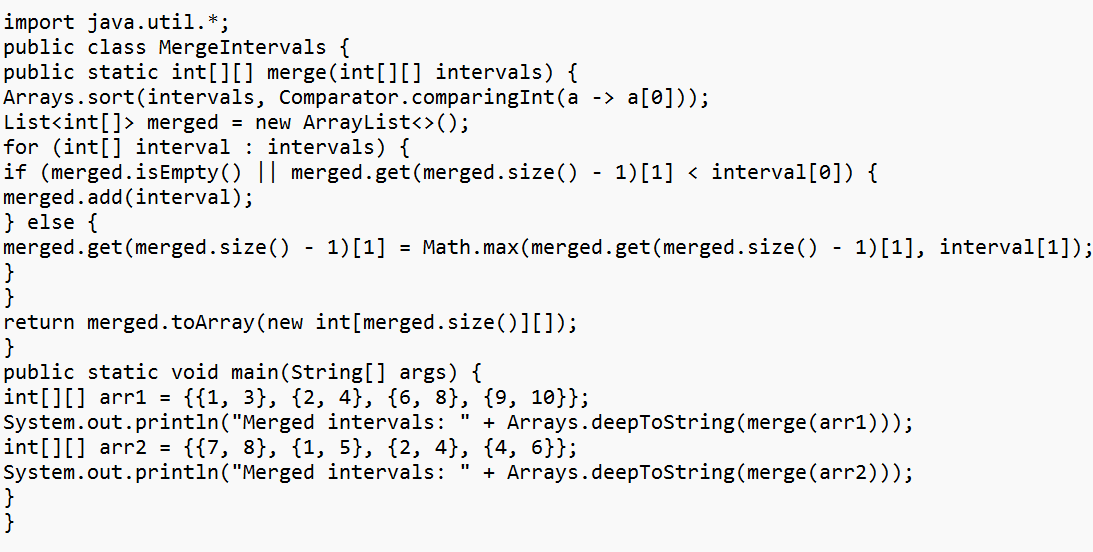
Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]]

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

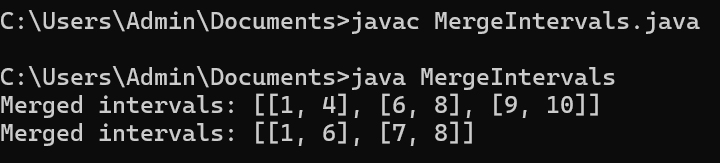
Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval

[1, 6].

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**Sorting: O(nlogn),** where n is the number of intervals.

**Merging: O(n),** since we iterate over the sorted intervals once to merge them**.**

**9) QUESTION**

**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},

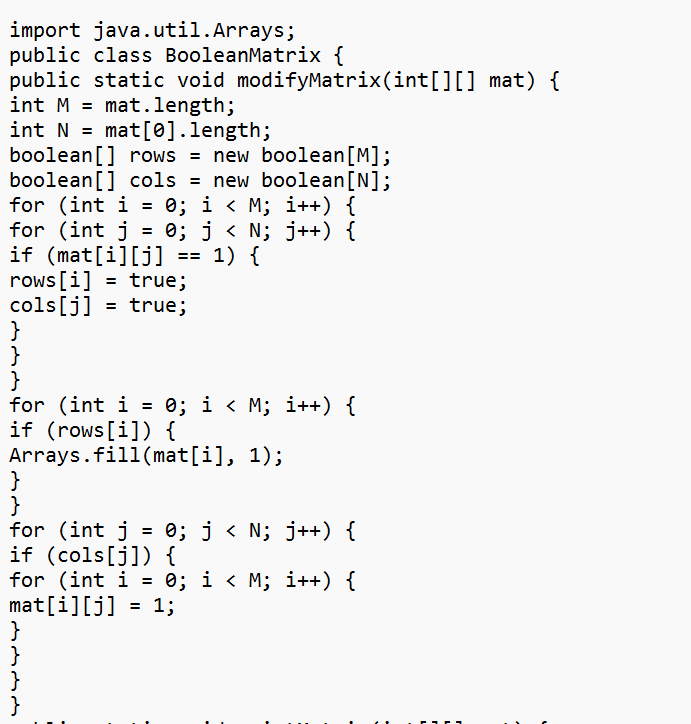
{0, 0, 0, 0}}

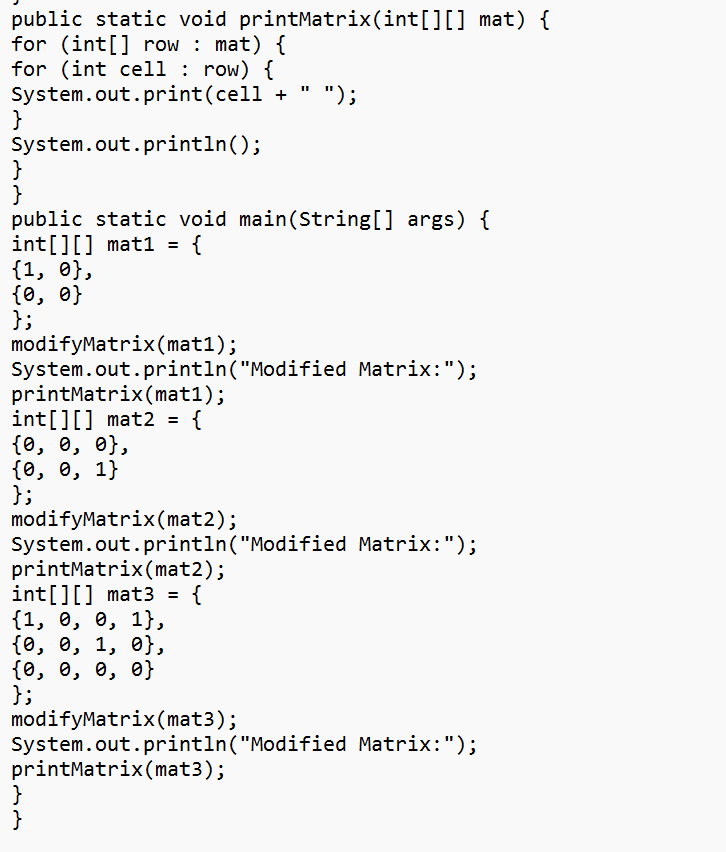
Output: {{1, 1, 1, 1},

{1, 1, 1, 1},

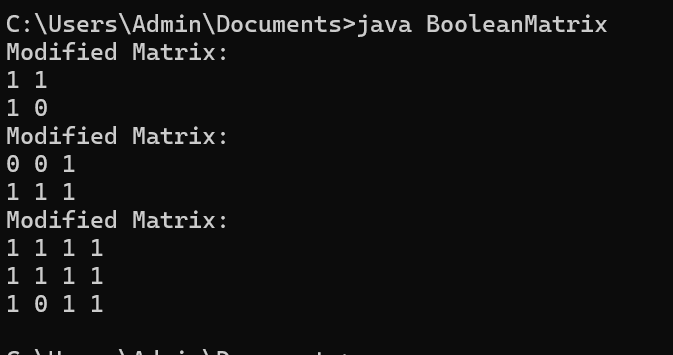
{1, 0, 1, 1}}

**SOLUTION**

****

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(M \* N)-** for scanning and modifying the matrix, where Mand N are the dimensions of the matrix.

**10) QUESTION**

**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},

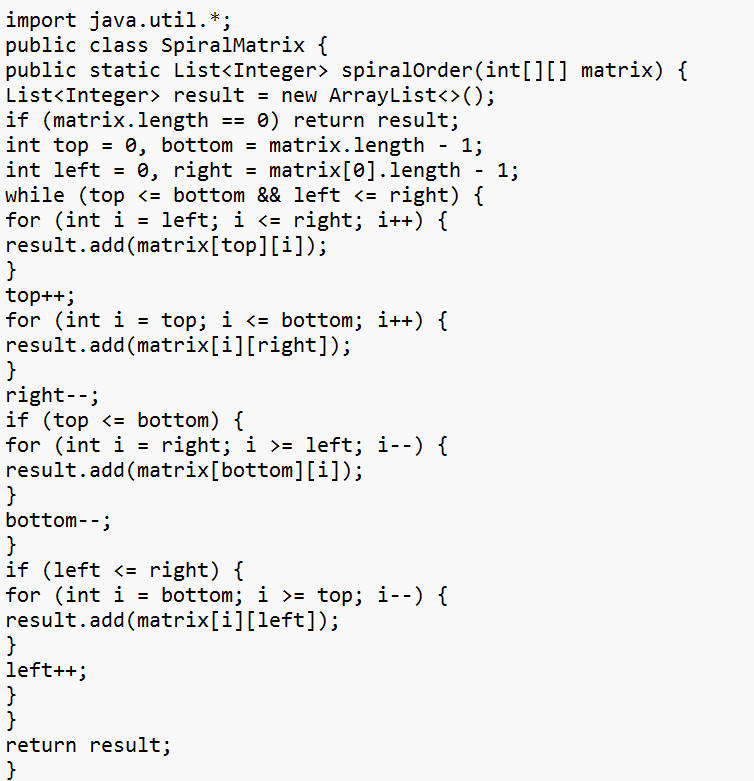
{7, 8, 9, 10, 11, 12},

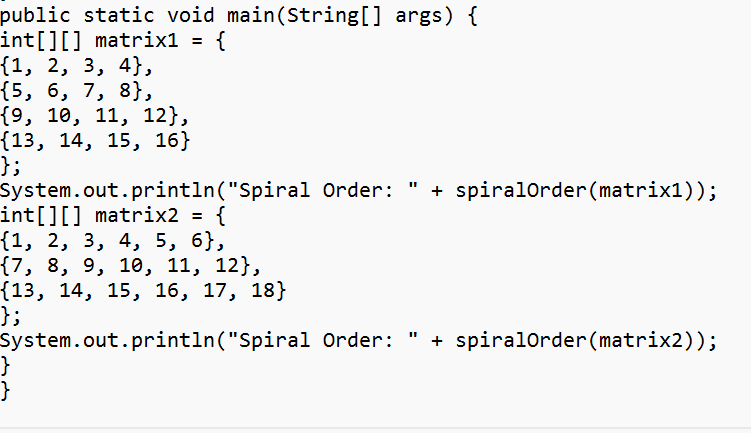
{13, 14, 15, 16, 17, 18}}

Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

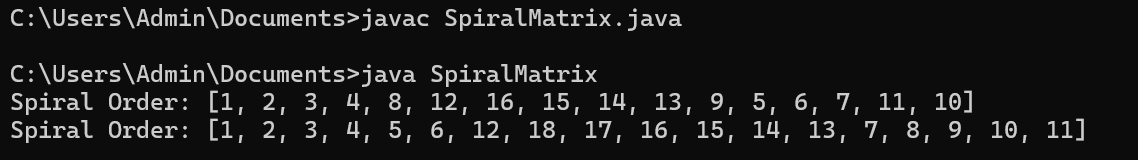
Explanation: The output is matrixin spiral format.

**SOLUTION**

****

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(m \* n) -**where m is the number of rows and n is the number of columns, as each element is visited once.

**13) QUESTION**

**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.

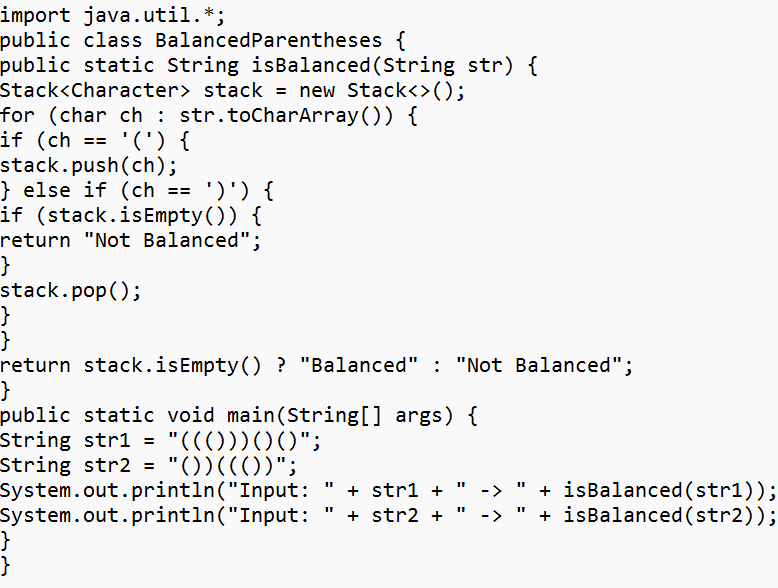
Input: str = “((()))()()”

Output: Balanced

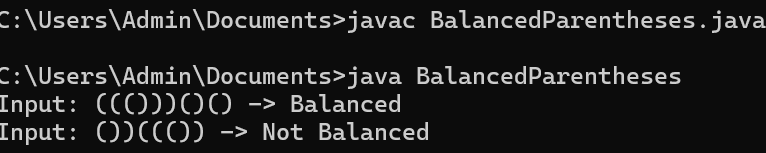
Input: str = “())((())”

Output: Not Balanced

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(N) -** where N is the length of the string, as each character is processed once.

**14) QUESTION**

**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

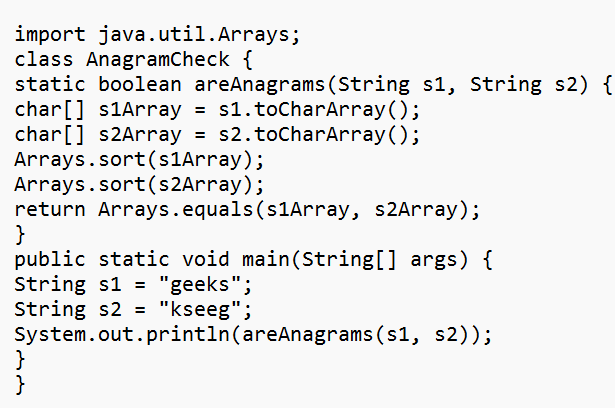
extra characters „i‟ and „c‟, so they are not anagrams.

Input: s1 = “g”, s2 = “g”

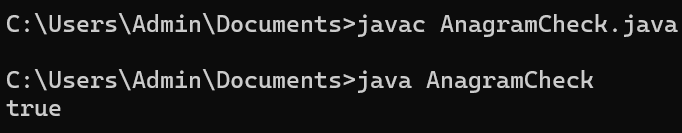
Output: true

Explanation: Characters in both the strings are same, so they are anagrams.

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(N) -** where N is the length of the strings, as we only perform a single pass to count frequencies.

**15)QUESTION**

**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”

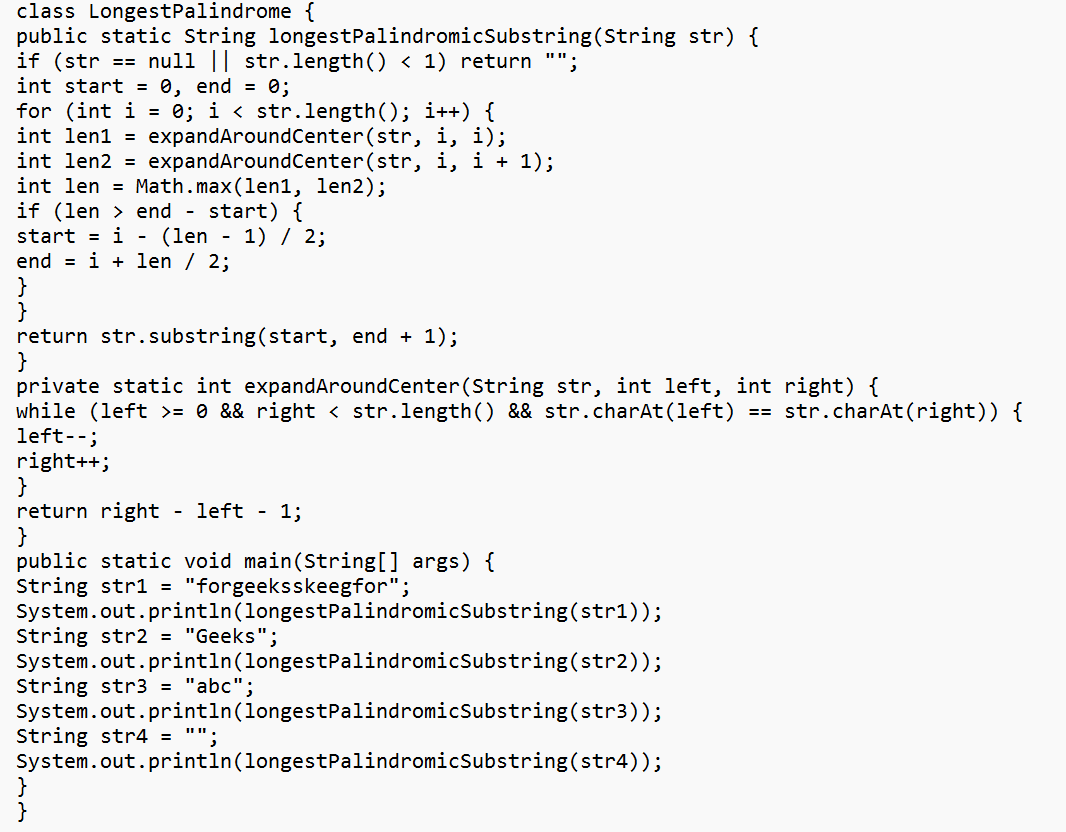
Input: str = “abc”

Output: “a”

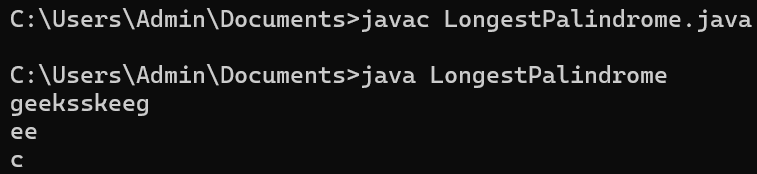
Input: str = “”

Output: “”

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n^2)-**where n is the length of the string. We check each character as a potential center and expand around it.

**16)QUESTION**

**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

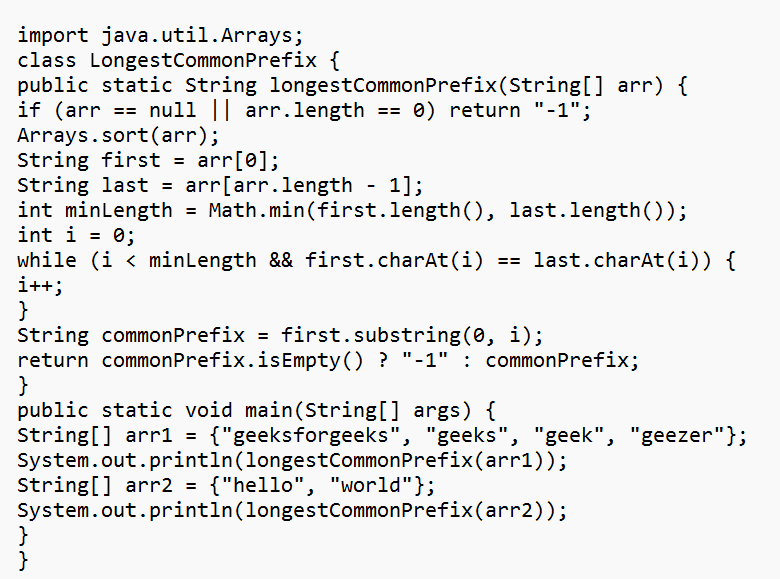
Explanation: “gee” is the longest common prefix in all the given strings.

Input: arr[] = [“hello”, “world”]

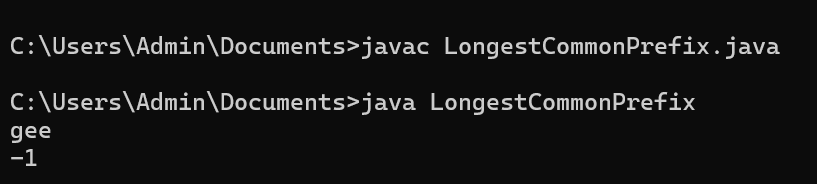
Output: -1

Explanation: There‟s no common prefix in the given strings.

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n log n)-** for sorting, where n is the number of strings.

**O(m)-**for prefix comparison, where m is the length of the shortest string.

**17)QUESTION**

**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.

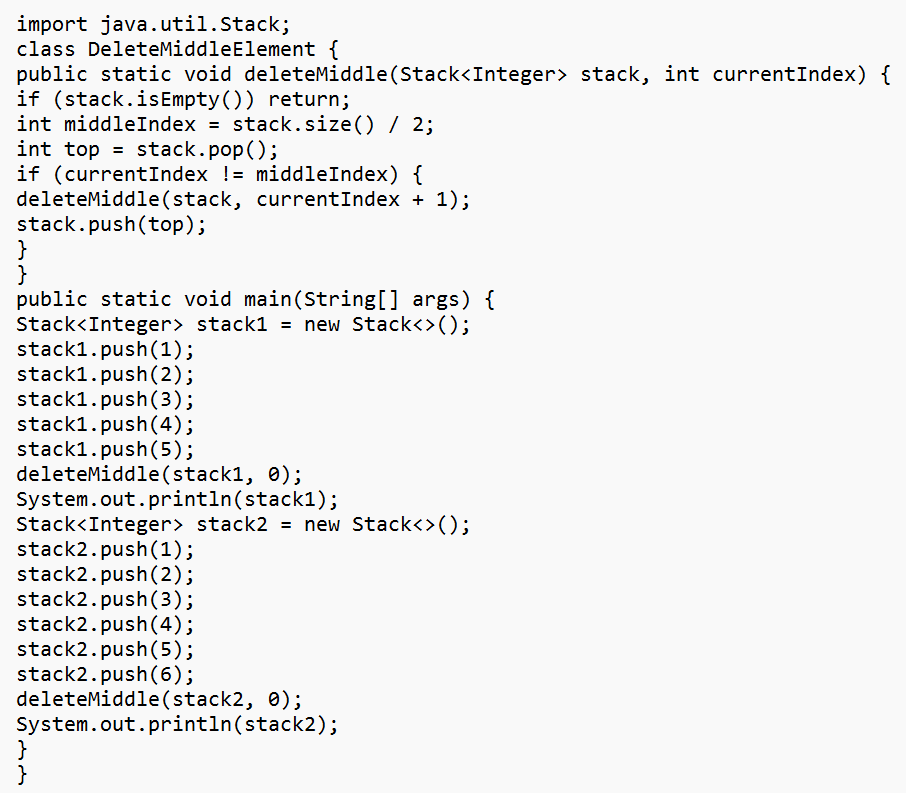
Input : Stack[] = [1, 2, 3, 4, 5]

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

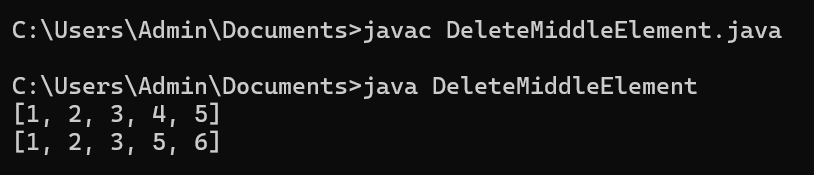
Input : Stack[] = [1, 2, 3, 4, 5, 6]

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

**SOLUTION**

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n**)-where n is the number of elements in the stack, due to recursion depth.

**18)QUESTION**

**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

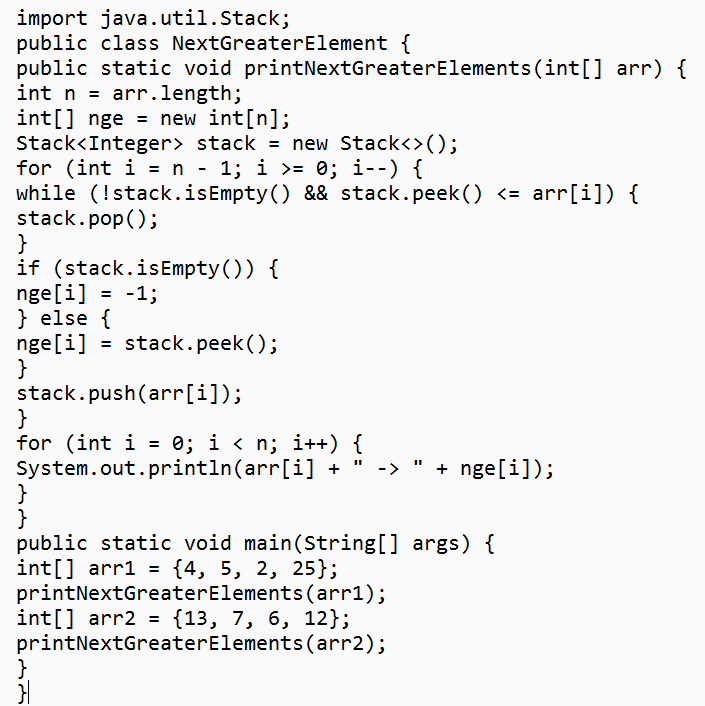
7 –> 12

6 –> 12

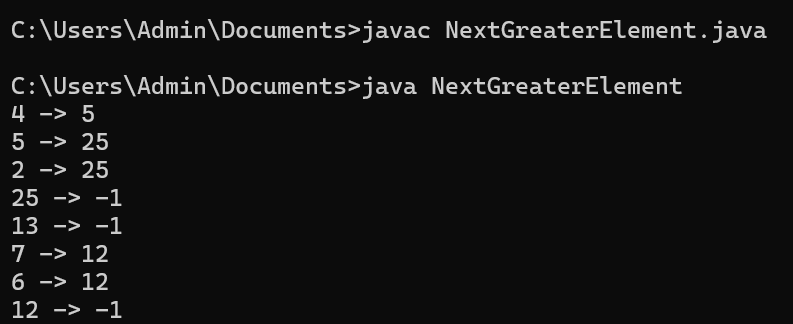
12 –> -1

Explanation: 13 and 12 don‟t have any element greater than them present on the right side

**SOLUTION**

****

**OUTPUT**

****

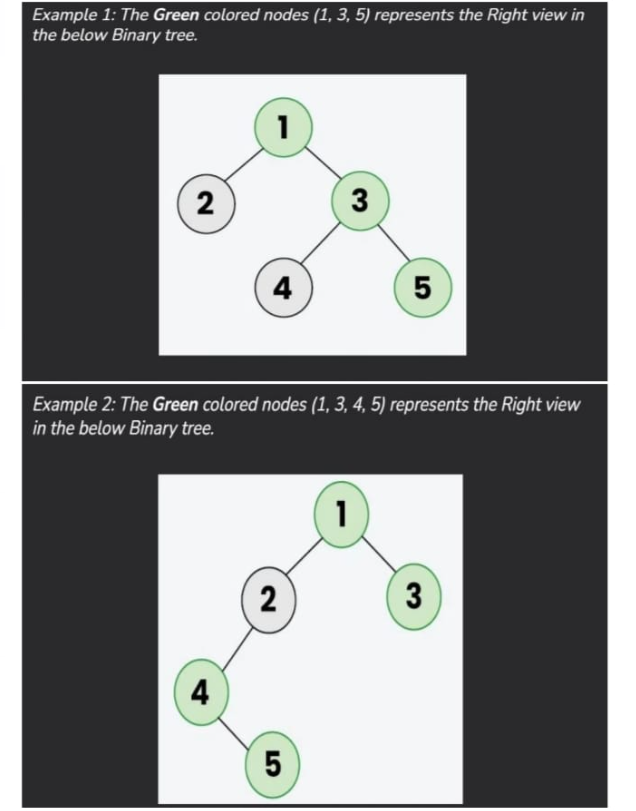
**TIME COMPLEXITY**

**O(n)-**where n is the number of elements in the array, as each element is pushed and popped from the stack once.

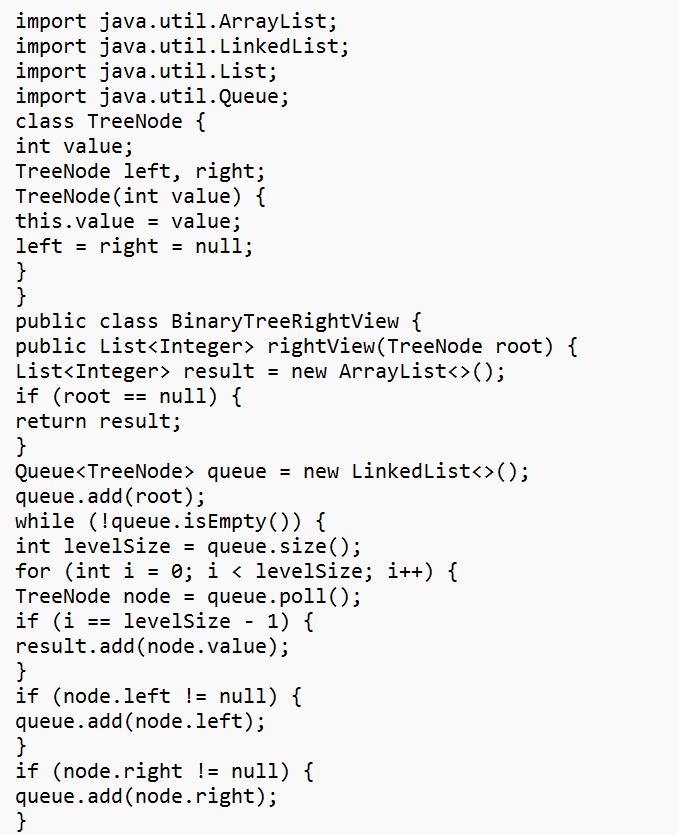
**19) QUESTION**

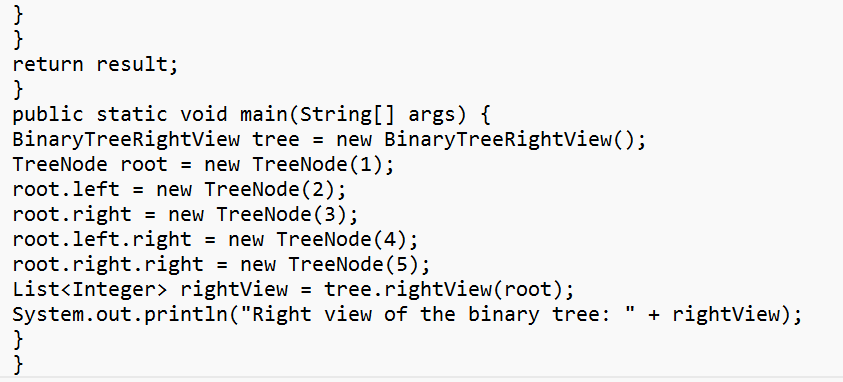
**Print Right View of a Binary Tree**

Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a

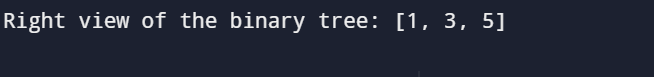
set of rightmost nodes for every level.

**SOLUTION**





**OUTPUT**

****

**TIME COMPLEXITY**

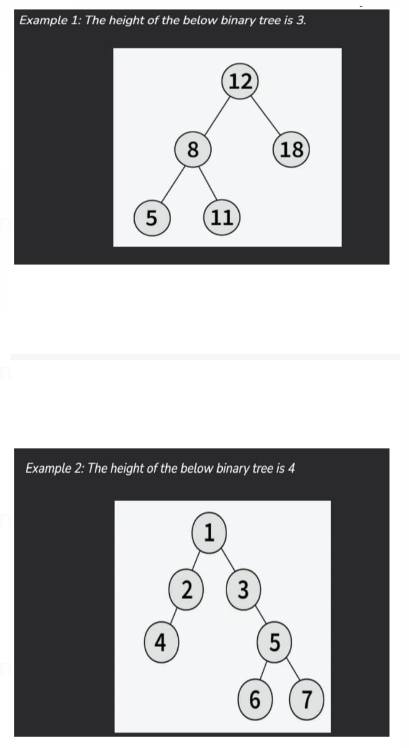
**O(N) -** because each of the N nodes is visited exactly once during the level-order traversal.

**20) QUESTION**

**Maximum Depth or Height of Binary Tree**

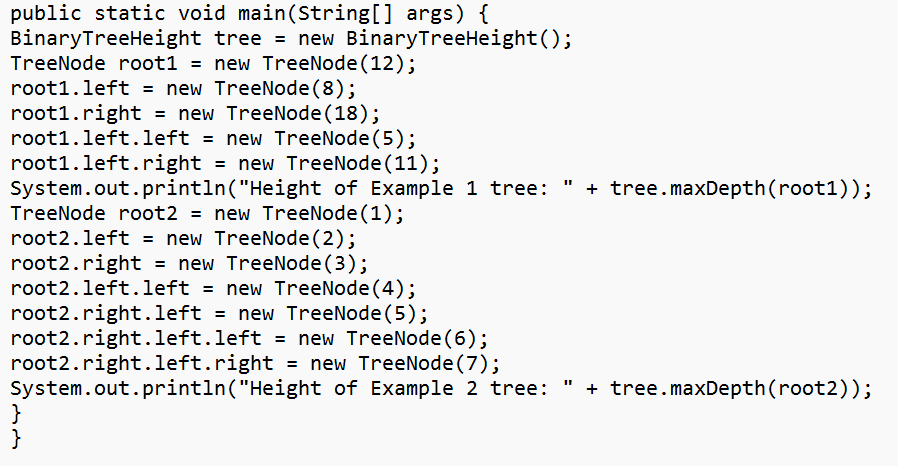
Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the

tree is the number of vertices in the tree from the root to the deepest node.

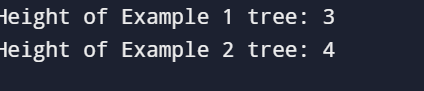


**SOLUTION**

****

****

**OUTPUT**

****

**TIME COMPLEXITY**

**O(N)-**where N is the number of nodes, because each node is visited once.