窗体顶端

Conference Schedule

A schedule has just been released for an upcoming tech conference. The schedule provides the start and end times of each of the presentations. Once a presentation has begun, no one can enter or leave the room. It takes no time to go from one presentation to another. Determine the maximum number of presentations that can be attended by one person.

**Example  
n** *= 3  
scheduleStart = [1, 1, 2]  
scheduleEnd = [3, 2, 4]*Using 0-based indexing, an attendee could attend any presentation alone, or both presentations *1* and *2*. Presentation *0* ends too late to be able to attend presentation *2* afterwards. The maximum number of presentations one person can attend is *2.***Function Description**Complete the function *maxPresentations* in the editor below.  
  
maxPresentations has the following parameter(s):  
*scheduleStart[n]:* the start times of presentation *I* *scheduleEnd[n]:* the end times of presentation *I***Returns:***I  
  
int:* the maximum number of presentations that can be attended by one person

**Constraints**

* 1 ≤ *n* ≤ 105
* 1 ≤ *scheduleStart[i], scheduleEnd[i]* ≤ 10^9

**Input Format For Custom Testing**

The first line contains an integer, *n*, the number of elements in *scheduleStart[]*.  
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains *an* integer that describes *scheduleStart[i].*The next line contains an integer, *n*, the number of elements in *scheduleEnd[]*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains *an* integer that describes *scheduleEnd[i].*

**Sample Case 0**

**Sample Input**

STDIN Function

----- -----

4 → scheduleStart[] size n = 4

1 → scheduleStart = [1, 1, 2, 3]

1

2

3

4 → scheduleEnd[] size n = 4

2 → scheduleEnd = [2, 3, 3, 4]

3

3

4

**Sample Output**

3

**Explanation**

An attendee can go to presentations *0, 2*, and *3* without conflict. If presentation *1* is chosen, from time *1* to *3*, only two presentations can be attended. The maximum number of presentations a single person can attend is *3.*

**Sample Case 1**

**Sample Input**

STDIN Function

----- -----

4 → scheduleStart[] size n = 4

6 → scheduleStart = [6, 1, 2, 3]

1

2

4

4 → scheduleEnd[] size n = 4

8 → scheduleEnd = [8, 9, 4, 7]

9

4

7

**Sample Output**

2

**Explanation**

An attendee can attend presentation *1* only as it runs the entire day, or they can instead attend meeting *2* from *2* until *4*, then choose to attend either presentation *0* or *3*. *The maximum number of presentations a single person can attend is 2*.

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'maxPresentations' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY scheduleStart

\* 2. INTEGER\_ARRAY scheduleEnd

\*/

public static int maxPresentations(List<Integer> scheduleStart, List<Integer> scheduleEnd) {

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int scheduleStartCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> scheduleStart = IntStream.range(0, scheduleStartCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int scheduleEndCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> scheduleEnd = IntStream.range(0, scheduleEndCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int result = Result.maxPresentations(scheduleStart, scheduleEnd);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();  
 }  
}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*