Aladdin and his Carpet

Aladdin wants to travel around the world and will choose a circular path to fly on his magical carpet. The carpet needs enough magic to take him from one place to another. He knows that after traveling some distance, he can find a magic source that will enable the carpet to travel a further distance.

There are *n* magical sources along the circular path numbered from *0* to *n-1*. Initially, the carpet has no magic and Aladdin can use a portal to jump to any magical source and start his journey. The carpet consumes units of magic equal to the units of distance travelled. He needs to choose a point to start his journey that will allow him to complete his journey. Determine the lowest index of the starting points from which Aladdin can start his journey and visit all of the places in the circular path in order. If there is no solution, return -1.

For example, there are *n = 4* sources of magic along his route: *magic = [3, 2, 5, 4]* and *dist = [2, 3, 4, 2].*  The first attempt is starting at the first source*, magic[0] = 3.*  He transports there without cost and collects *3* units of magic*.*  The distance to the next point is *dist[0] = 2.*  It takes *2* units of magic to get there and he collects *magic[1] = 2* units upon arrival, so he has *3 - 2 + 2 = 3* units of magic after making his first carpet ride. Continuing along the journey:

* *3 - dist[1] + magic[2] = 3 - 3 + 5 = 5*
* *5 - dist[2] + magic[3] = 5 - 4 + 4 = 5*
* *5 - dist[3] = 5 - 2 = 3*

At this point, he is back to the first source. Because he can complete his journey starting at source *magic[0]*, there is no reason to continue with the analysis so its index, *0,* is returned. To illustrate a point from the same example, if he starts at position *2*, where *magic[1] = 2* and *dist[1] = 3*, he will not be able to proceed to the next point because the distance is greater than his magic units. Note that the list is circular, so from *magic[3]* in this example, the next source on the path is *magic[0].*

**Function Description**

Complete the function *optimalPoint* in the editor below. The function must return an integer that denotes the minimum index of *magic* from which he can start a successful journey. If no such starting point exists, return *-1.*

*optimalPoint* has the following parameter(s):

*magic[magic[0],...magic[n-1]]:*  an array of integers where *magic[i]* denotes the amount of magic in the *ith* source.

*dist[dist[0],...dist[n-1]]*:  an array of integers where *dist[i]* denotes the distance to the next magical source.

**Constraints**

* *1 ≤ n ≤ 100000*
* *0 ≤ magic[i] ≤ 10000*
* *0 ≤ dist[i] ≤ 10000*

Input Format For Custom Testing

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

The next line again contains the integer, *n*, that denotes the number of elements in *dist*.

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

Sample Case 0

**Sample Input For Custom Testing**

4

2

4

5

2

4

4

3

1

3

**Sample Output**

1

**Explanation**

Here *magic = [2, 4, 5, 2]* and *dist = [4, 3, 1, 3].*  If Aladdin starts at the second magical source, his magic levels are:

* *magic[1] = 4*
* *4 - dist[1] + magic[2] = 4 - 3 + 5 = 6*
* *6 - dist[2] + magic[3] = 6 - 1 + 2 = 7*
* *7 - dist[3] + magic[0] = 7 - 3 + 2 = 6*
* *6 - dist[0] = 6 - 4 = 2.*

The first point from where Aladdin can start his journey is the *2nd* magical source. The output should be *1,* the index of the *2nd* location.

Sample Case 1

**Sample Input For Custom Testing**

4

8

4

1

9

4

10

9

3

5

**Sample Output**

-1

**Explanation**

Here *magic = [8, 4, 1, 9]* and *dist = [10, 9, 3, 5].*  In each case except *magic[3] = 9*, the distance to the next source is greater than the amount of magic at the current source.

*magic[3] = 9*

*9 - dist[3] + magic[0] = 9 - 5 + 8 = 12*

*12 - dist[0] + magic[1] = 12 - 10 + 4 = 6*

*6 - dist[1] + magic[2] = 6 - 9 + 1 = -2*

No matter where Aladdin starts, he will not be able to finish his travel.

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 'optimalPoint' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY magic

\* 2. INTEGER\_ARRAY dist

\*/

public static int optimalPoint(List<Integer> magic, List<Integer> dist) {

// Write your code here

}

}

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 magicCount = Integer.parseInt(bufferedReader.readLine().trim());

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

try {

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

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

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

List<Integer> dist = IntStream.range(0, distCount).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.optimalPoint(magic, dist);

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

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

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