Dam Design

Your company is designing a dam to be built across a stream to create a small lake. To reduce materials cost, it will be made of one or more concrete walls with mud packed in between them. Determine the maximum height of the mud segments in the dam with the following restrictions:

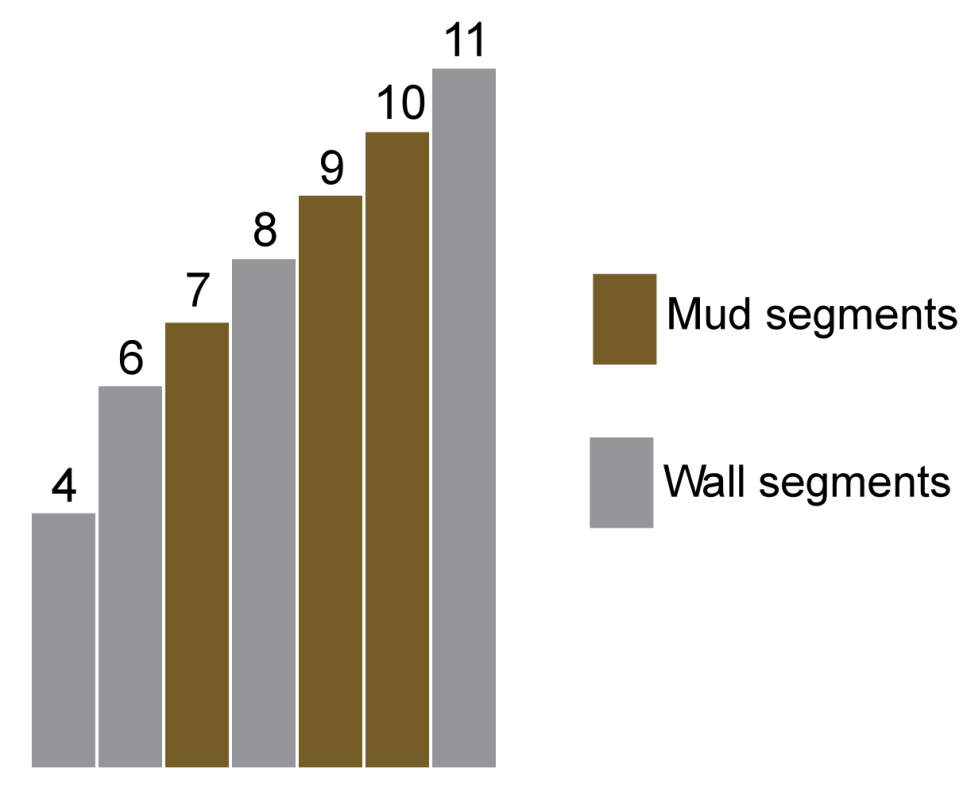
* One unit width of the gap between walls will contain one *segment* of packed mud
* The height of mud in a segment cannot exceed *1* unit more than an adjacent wall or mud segment.

Given the placement of a number of walls and their heights, determine the maximum height of a mud segment that can be built. If no mud segment can be built, return *0.*

****Example****

*wallPositions = [1, 2, 4, 7]*

*wallHeights = [4, 6, 8, 11]*



* There is no space between the first two walls.
* Between positions *2* and *4*, there is one unit open for mud. Heights of the surrounding walls are *6* and *8*, so the maximum height of mud is *6 + 1 = 7.*
* Between positions *4* and *7*there are two units. The heights of surrounding walls are *8* and *11.*
  + The maximum height mud segment next to the wall of height *8* is *9.*
  + The maximum height mud next to a mud segment of height *9* is *10.*
* Overall, mud segment heights are *7, 9* and *10*, and the maximum height is *10.*

**Function Description**

Complete the function *maxHeight* in the editor below.

maxHeight has the following parameter(s):

    int *wallPositions[n]:*  an array of integers

    int *wallHeights[n]:*  an array of integers

Returns:

    int: the maximum height mud segment that can be build

**Constraints**

* *1 < n ≤ 105*
* *1 ≤ wallPositions[i], wallHeights[i] ≤ 109 (where 0 ≤ i < n)*

Input Format For Custom Testing

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

Sample Case 0

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

3    →   wallPositions[] size n = 3

1    →   wallPositions = [1, 3, 7]

3

7

3    →   wallHeights[] size n = 3

4    →   wallHeights = [4, 3, 3]

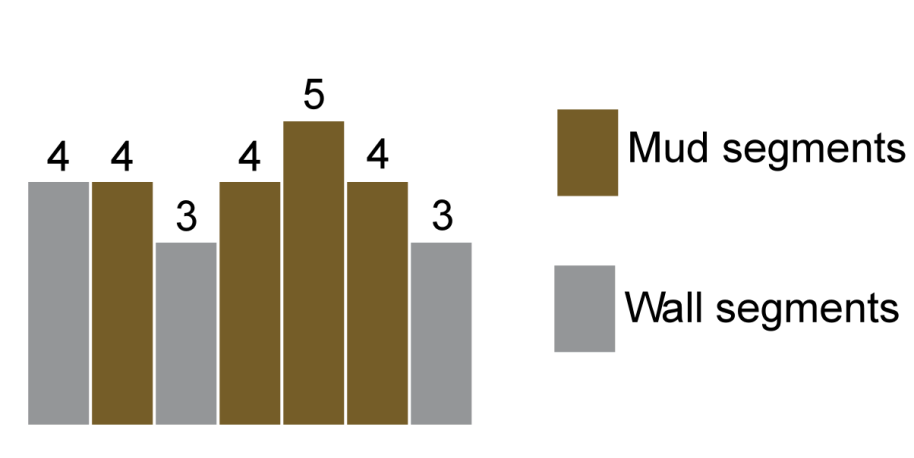
3

3

**Sample Output**

5

**Explanation**

****

The *wallPositions = [1, 3, 7]* and *wallHeights = [4, 3, 3].*  There can be a segment of height *4* at position *2* supported by walls of heights *4* and *3.*  Between positions *3* and *7*, there can be a segment of height *4* at positions *4* and *6.*  Between them, a segment can be built of height *5* at position *5.*

Sample Case 1

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

2    →   wallPositions[] size n = 2

1    →   wallPositions = [1, 10]

10

2    →   wallHeights[] size n = 2

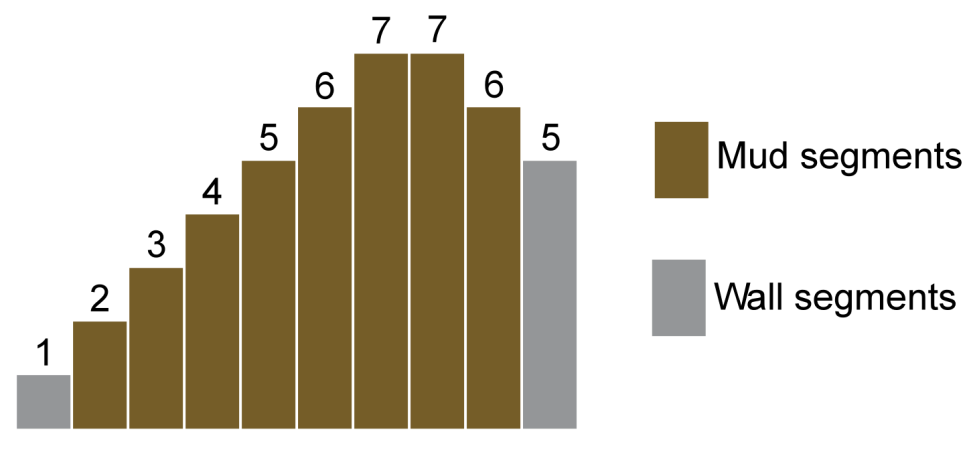
1    →   wallHeights = [1, 5]

5

**Sample Output**

7

**Explanation**



The *wallPositions = [1, 10]* and *wallHeights = [1, 5].* The heights of the mud segments from positions *2* through *9* are *[2, 3, 4, 5, 6, 7, 7, 6].*

*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 'maxHeight' function below.*

*\**

*\* The function is expected to return an INTEGER.*

*\* The function accepts following parameters:*

*\* 1. INTEGER\_ARRAY wallPositions*

*\* 2. INTEGER\_ARRAY wallHeights*

*\*/*

*public static int maxHeight(List<Integer> wallPositions, List<Integer> wallHeights) {*

*// 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 wallPositionsCount = Integer.parseInt(bufferedReader.readLine().trim());*

*List<Integer> wallPositions = IntStream.range(0, wallPositionsCount).mapToObj(i -> {*

*try {*

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

*} catch (IOException ex) {*

*throw new RuntimeException(ex);*

*}*

*})*

*.map(String::trim)*

*.map(Integer::parseInt)*

*.collect(toList());*

*int wallHeightsCount = Integer.parseInt(bufferedReader.readLine().trim());*

*List<Integer> wallHeights = IntStream.range(0, wallHeightsCount).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.maxHeight(wallPositions, wallHeights);*

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

*bufferedWriter.newLine();*

*bufferedReader.close();*

*bufferedWriter.close();*

*}*

*}*

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