Maximizing the Final Element

Given an array of integers, perform certain operations in order to satisfy some constraints. The constraints are as follows:

1. The first array element must be 1.
2. For all other elements, the difference between adjacent integers must not be greater than 1. In other words, for 1 ≤ *i* < *n*, *arr[i] - arr[i-1]*≤ 1.
3. To accomplish this, the following operations are available:

* Rearrange the elements in any way.
* Reduce any element to any number that is at least 1.

What is the maximum value that can be achieved for the *final* element of the array by following these operations and constraints?

****Example****

*arr = [3, 1, 3, 4]*

1. Subtract 1 from the first element, making the array [2, 1, 3, 4].
2. Rearrange the array into [1, 2, 3, 4].
3. The final element's value is 4, the maximum value that can be achieved. Therefore, the answer is 4.

**Function Description**

Complete the function *getMaxValue* in the editor below.

getMaxValue has the following parameter:

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

Returns:

    int: the maximum value that can be achieved for the final element of the array given the conditions above

**Constraints**

* 1 ≤ *n* ≤ 105
* 1 ≤ *arr[i]* ≤ 109

Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *arr*.

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

Sample Case 0

**Sample Input For Custom Testing**

STDIN     Function

-----     --------

4 → arr[] size n = 4

1 → arr = [1, 3, 2, 2]

3

2

2

**Sample Output**

3

**Explanation**

These elements can be rearranged to become [1, 2, 2, 3], which results in a maximum value of 3 for the final element. Notice how this array follows the constraints that (1) the first element is 1, and (2) the difference between each pair of adjacent integers is no more than 1.

Sample Case 1

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

4    →   arr[] size n = 4

3    →   arr = [3, 2, 3, 5]

2

3

5

**Sample Output**

4

**Explanation**

These elements can be rearranged to become [2, 3, 3, 5]. Then, the heights can be adjusted to become [1, 2, 3, 4]. Therefore, the maximum value of the final element is 4.

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 'getMaxValue' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static int getMaxValue(List<Integer> arr) {

// 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 arrCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = IntStream.range(0, arrCount).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.getMaxValue(arr);

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

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

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