About Tutorial Challenges
Many of the challenges on HackerRank are difficult and assume that you already know the relevant algorithms. These tutorial challenges are different. They break down algorithmic concepts into smaller challenges so that you can learn the algorithm by solving them. They are intended for those who already know some programming, however. You could be a student majoring in computer science, a self-taught programmer, or an experienced developer who wants an active algorithms review. Here's a great place to learn by doing!

The first series of challenges covers sorting. They are listed below:

Tutorial Challenges - Sorting

Insertion Sort challenges

- Insertion Sort 1 - Inserting
- Insertion Sort 2 - Sorting
- Correctness and loop invariant
- Running Time of Algorithms

Quicksort challenges

- Quicksort 1 - Partition
- Quicksort 2 - Sorting
- Quicksort In-place (advanced)
- Running time of Quicksort

Counting sort challenges

- Counting Sort 1 - Counting
- Counting Sort 2 - Simple sort
- Counting Sort 3 - Preparing
- Full Counting Sort (advanced)

There will also be some challenges where you'll get to apply what you've learned using the completed algorithms.

About the Challenges
Each challenge will describe a scenario and you will code a solution. As you progress through the challenges, you will learn some important concepts in algorithms. In each challenge, you will receive input on STDIN and you will need to print the correct output to STDOUT.

There may be time limits that will force you to make your code efficient. If you receive a "Terminated due to time out" message when you submit your solution, you'll need to reconsider your method. If you want to test your code locally, each test case can be downloaded, inputs and expected results, using *hackos*. You earn hackos as you solve challenges, and you can spend them on these tests.

For many challenges, helper methods (like an array) will be provided for you to process the input into a useful format. You can use these methods to get started with your program, or you can write your own input methods if you want. Your code just needs to print the right output to each test case.

Sample Challenge
This is a simple challenge to get things started. Given a sorted array (arr) and a number (V), can you print the index location of V in the array?

For example, if $arr = [1, 2, 3]$ and $V = 3$, you would print 2 for a zero-based index array.

If you are going to use the provided code for I/O, this next section is for you.

Function Description

Complete the *introTutorial* function in the editor below. It must return an integer representing the zero-based index of V.

introTutorial has the following parameter(s):

- *arr*: a sorted array of integers

- V: an integer to search for

The next section describes the input format. You can often skip it, if you are using included methods or code stubs.

Input Format

The first line contains an integer, V, a value to search for.
The next line contains an integer, n, the size of arr. The last line contains n space-separated integers, each a value of $arr[i]$ where $0 \leq i < n$.

Output Format
Output the index of V in the array.

The next section describes the constraints and ranges of the input. You should check this section to know the range of the input.

Constraints

- $1 \leq n \leq 1000$
- $-1000 \leq V \leq 1000, V \in arr$
- It is guaranteed that V will occur in arr exactly once.

This "sample" shows the first input test case. It is often useful to go through the sample to understand a challenge.

Sample Input 0

```
4
6
1 4 5 7 9 12
```

Sample Output 0

```
1
```

Explanation 0
$V = 4$. The value 4 is the 2^{nd} element in the array, but its index is 1 since in this case, array indices start from 0 (see array definition under *Input Format*).