

# Correctness and the Loop Invariant

## Problem Statement

In the previous challenge, you wrote code to perform an *Insertion Sort* on an unsorted array. But how would you prove that the code is correct? I.e. how do you show that for any input your code will provide the right output?

## Loop Invariant

In computer science, you could prove it formally with a *loop invariant*, where you state that a desired property is maintained in your loop. Such a proof is broken down into the following parts:

- *Initialization*: It is true (in a limited sense) before the loop runs.
- *Maintenance*: If it's true before an iteration of a loop, it remains true before the next iteration.
- *Termination*: It will terminate in a useful way once it is finished.

## Insertion Sort's Invariant

Say, you have some InsertionSort code, where the outer loop goes through the whole array  $A$ :

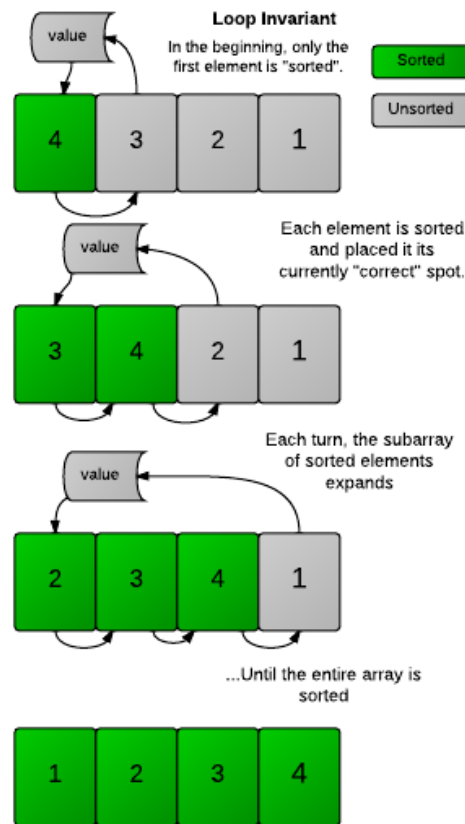
```
for(int i = 1; i < A.length; i++){  
    //insertion sort code
```

You could then state the following loop invariant:

At the start of every iteration of the outer loop (indexed with  $i$ ), the subarray until  $ar[i]$  consists of the original elements that were there, but in sorted order.

To prove Insertion Sort is correct, you will then demonstrate it for the three stages:

- *Initialization* - The subarray starts with the first element of the array, and it is (obviously) sorted to begin with.
- *Maintenance* - Each iteration of the loop expands the subarray, but keeps the sorted property. An element  $V$  gets inserted into the array only when it is greater than the element to its left. Since the elements to its left have already been sorted, it means  $V$  is greater than all the elements to its left, so the array remains sorted. (In *Insertion Sort 2* we saw this by printing the array each time an element was properly inserted.)
- *Termination* - The code will terminate after  $i$  has reached the last element in the array, which means the sorted subarray has expanded to encompass the entire array. The array is now fully sorted.



You can often use a similar process to demonstrate the correctness of many algorithms. You can see [these notes](#) for more information.

## Challenge

In the InsertionSort code below, there is an error. Can you fix it? Print the array only once, when it is fully sorted.

## Details

The Input format and the constraints are the same as in the previous challenges and are presented below.

### Input Format

There will be two lines of input:

- $s$  - the size of the array
- $ar$  - the list of numbers that makes up the array

### Output Format

Output the numbers in order, space-separated.

### Constraints

$$1 \leq s \leq 1000$$

$$-1500 \leq V \leq 1500, V \in ar$$

### Sample Input

```
6
1 4 3 5 6 2
```

Sample Output

1 2 3 4 5 6