29.26 more points to get your next star!

Rank: **154191** | Points: **445.74/475**

Problem Solving

Insertion Sort - Part 2

Problem Submissions Leaderboard

In *Insertion Sort Part 1*, you inserted one element into an array at its correct sorted position. Using the same approach repeatedly, can you sort an entire array?

Guideline: You already can place an element into a sorted array. How can you use that code to build up a sorted array, one element at a time? Note that in the first step, when you consider an array with just the first element, it is already sorted since there's nothing to compare it to.

In this challenge, print the array after each iteration of the insertion sort, i.e., whenever the next element has been inserted at its correct position. Since the array composed of just the first element is already sorted, begin printing after placing the second element.

For example, there are $n = 7$ elements in $arr = [3, 4, 7, 5, 6, 2, 1]$. Working from left to right, we get the following output:

```
3 4 7 5 6 2 1
3 4 7 5 6 2 1
3 4 5 7 6 2 1
3 4 5 6 7 2 1
2 3 4 5 6 7 1
1 2 3 4 5 6 7
```

Function Description

Complete the *insertionSort2* function in the editor below. At each iteration, it should print the array as space-separated integers on a separate line.

insertionSort2 has the following parameter(s):

- *n*: an integer representing the length of the array arr
- *arr*: an array of integers

Input Format

The first line contains an integer, n, the size of arr.
The next line contains n space-separated integers $arr[i]$.

Constraints

$1 \leq n \leq 1000$
$-10000 \leq arr[i] \leq 10000, 0 \leq i < n$

Output Format

On each line, output the entire array at every iteration.

Sample Input

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

Sample Output

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

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

Explanation

Skip testing 1 against itself at position 0. It is sorted.

Test position 1 against position 0: $4 > 1$, no more to check, no change.

Print arr

Test position 2 against positions 1 and 0:

- $3 < 4$, new position may be 1. Keep checking.
- $3 > 1$, so insert 3 at position 1 and move others to the right.

Print arr

Test position 3 against positions $2, 1, 0$ (as necessary): no change.

Print arr

Test position 4 against positions $3, 2, 1, 0$: no change.

Print arr

Test position 5 against positions $4, 3, 2, 1, 0$, insert 2 at position 1 and move others to the right.

Print arr

C#

```
using System.Globalization;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization;
using System.Text.RegularExpressions;
using System.Text;
using System;

class Solution {

    // Complete the insertionSort2 function below.
    static void insertionSort2(int n, int[] arr) {

        for (int i = 1; i < n; i++)
        {
            for (int j = 0; j < i; j++)
            {
               if (arr[j] > arr[i])
                {
                    var temp = arr[j];
                    arr[j] = arr[i];
                    arr[i] = temp;
                }
            }
            Console.WriteLine(String.Join(" ",arr));
        }
    }

    static void Main(string[] args) {
```

Line: 24 Col: 11