Badge Progress (Details)   Points: 1624.64 Rank: 9555

Running Time of Algorithms 🔖

H by **HackerRank**

| Problem | Submissions | Leaderboard | Discussions |

In the previous challenges you created an *Insertion Sort* algorithm. It is a simple sorting algorithm that works well with small or mostly sorted data. However, it takes a long time to sort large unsorted data. To see why, we will analyze its running time.

Running Time of Algorithms

The running time of an algorithm for a specific input depends on the number of operations executed. The greater the number of operations, the longer the running time of an algorithm. We usually want to know how many operations an algorithm will execute in proportion to the size of its input, which we will call N.

For example, the first two input files for Insertion Sort were small, so it was able to sort them quickly. But the next two files were larger and sorting them took more time. What is the ratio of the running time of Insertion Sort to the size of the input? To answer this question, we need to examine the Insertion Sort algorithm.

Analysis of Insertion Sort

For each element V in an array of N numbers, Insertion Sort shifts everything to the right until it can insert V into the array.

How long does all that shifting take?

In the best case, where the array was already sorted, no element will need to be moved, so the algorithm will just run through the array once and return the sorted array. The running time would be directly proportional to the size of the input, so we can say it will take N time.

However, we usually focus on the worst-case running time (computer scientists are pretty pessimistic). The worst case for Insertion Sort occurs when the array is in reverse order. To insert each number, the algorithm will have to shift over that number to the beginning of the array. Sorting the entire array of N numbers will therefore take $1 + 2 + \ldots + (N - 1)$ operations, which is $N(N - 1)/2$ (almost $N^2/2$). Computer scientists just round that up (pick the dominant term) to N^2 and say that Insertion Sort is an "N^2 time" algorithm.
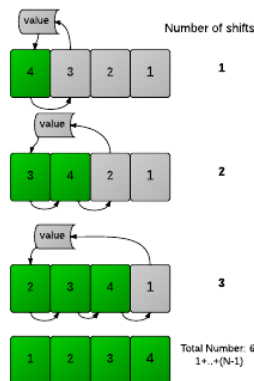
What this means

As the size of an input (N) increases, Insertion Sort's running time will increase by the square of N. So Insertion Sort can work well for small inputs, but becomes unreasonably slow for larger inputs. For large inputs, people use sorting algorithms that have better running times, which we will examine later.

Challenge

Can you modify your previous Insertion Sort implementation to keep track of the number of shifts it makes while sorting? The only thing you should print is the number of shifts made by the algorithm to completely sort the array. A shift occurs when an element's position changes in the array. Do not shift an element if it is not necessary.

Input Format

The first line contains N, the number of elements to be sorted. The next line contains N integers $a[1]$, $a[2]$, ..., $a[N]$.

Output Format

Output the number of shifts it takes to sort the array.

Constraints

$1 \leq N \leq 1001$
$-10000 \leq x \leq 10000, x \in a$

Sample Input

```
5
2 1 3 1 2
```

Sample Output

```
4
```

Explanation

The first *1* is shifted once. The *3* stays where it is. The next *1* gets shifted twice. The final *2* gets shifted once. Hence, the total number of shifts is **4**.

Task

For this problem, you can copy your code from the InsertionSort problem, and modify it to keep track of the number of shifts instead of printing the array.

f   y   in

Submissions: 33021
Max Score: 30
Difficulty: Easy

Rate This Challenge:
☆ ☆ ☆ ☆ ☆

More

Current Buffer (saved locally, editable)   🔀 🕘   C#   ⌄   ⤢   ⚙

```csharp
using System;
using System.Collections.Generic;
using System.IO;
class Solution {
    static void insertionSort(int[] ar)
        {
            int shifts = 0;

            for (int i = 1; i < ar.Length; i++)
            {
                int indice = i;
                while (indice > 0 && ar[indice - 1] > ar[indice])
                {
                    shifts++;
                    int temp = ar[indice];
                    ar[indice] = ar[indice - 1];
                    ar[indice - 1] = temp;
                    indice--;
                }

            }
            Console.WriteLine(shifts);
        }



        static void Main(string[] args)
        {

            int n = int.Parse(Console.ReadLine());
            int[] arr = Array.ConvertAll(Console.ReadLine().Split(' '), e => int.Parse(e));

            insertionSort(arr);

            //Console.ReadLine();
        }
}
```

Line: 34 Col: 1

⬆ <u>Upload Code as File</u>   ☐ Test against custom input   Run Code   Submit Code

Congrats, you solved this challenge!

✔ Test Case #0   ✔ Test Case #1   ✔ Test Case #2

✔ Test Case #3   ✔ Test Case #4   ✔ Test Case #5

✔ Test Case #6

Next Challenge

Join us on IRC at #hackerrank on freenode for hugs or bugs.

Contest Calendar | Interview Prep | Blog | Scoring | Environment | FAQ | About Us | Support | Careers | Terms Of Service | Privacy Policy | Request a Feature