Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
- PRACTICE-11_FUN-PTR
- LAB-PRAC-11_FUN-PTR
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC
  - Predecessor and Successor
  - Insertion Sort
  - Link a List
  - The United Sums of Arrays
  - Bubble Sort
  - Pretty Queues Revisited
  - Just About Sorted
  - Brick Sort
  - All My Descendants
  - Mr C likes a Majority
  - Cocktail Sort
  - All My Descendants - Part II

Insertion Sort

LAB-PRAC-14_SORT-MISC

Insertion Sort [20 marks]

Problem Statement
There are several sorting algorithms that have been developed over the years. We saw some during the lectures and will explore others here. One popular algorithm for sorting small to medium arrays is called Insertion Sort.

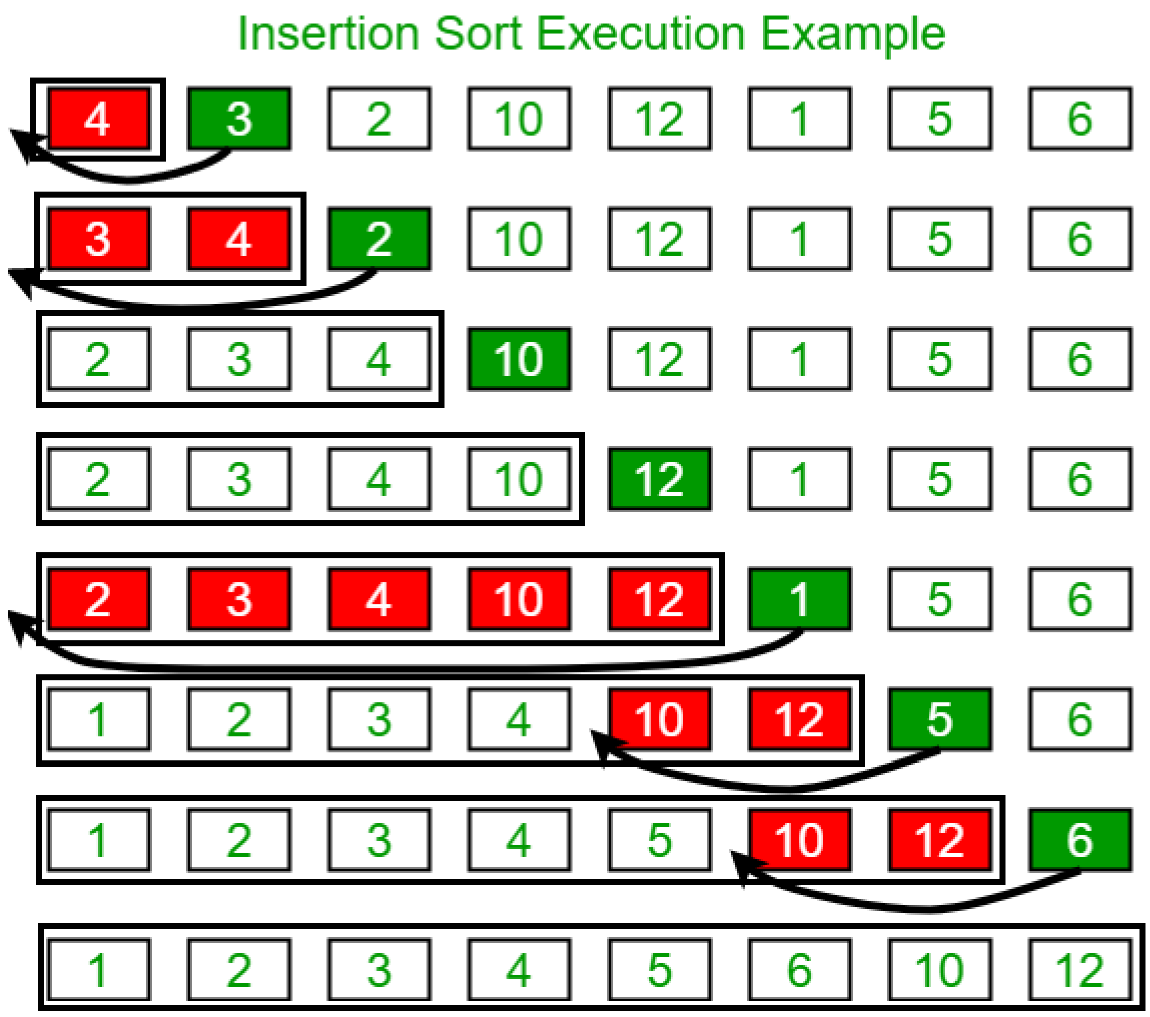
Insertion sort maintains the following invariant - at the end of the i-th iteration of the algorithm, the first i positions of the array must be sorted in non-decreasing order. If there are n elements in the array, then the algorithm runs for n iterations. Say after i iterations, the first i elements are sorted in non-decreasing order. Then in the (i+1)-th iteration, the element currently at index i in the array (i.e. the (i+1)-th element from the left in the array) is *inserted* into its appropriate position with respect to the first i elements, thus creating an array of (i+1) elements sorted in non-decreasing order and also fulfilling the promise of the invariant..

This is where the algorithm gets its name. At the i-th iteration, the i-th element of the array is inserted into its correct location with respect to the previous (i-1) elements to create a sorted array of i elements. The first line of the input will give you n, a strictly positive integer and the second line will give you n integers, separated by a space. Store these numbers in an array of size n.

In your output, you have to print the array after each iteration of insertion sort, on a separate line. Print the array by printing each number in the array from left to right with two numbers separated by a single space. However, there should be no space at the end of a line.

The image below shows you how insertion sort works. Notice that at each step, a larger and larger set of elements (indicated by the black box) gets sorted in non-decreasing order. Also notice that at some steps, nothing needs to be done but in other steps, the element just outside the box needs to be inserted in the correct position to expand the sorted box.

Image courtesy: geeksforgeeks.org (image modified for better clarity)



Caution

1. Please do not try to cheat by using library functions like qsort(). These will not sort the array in the order insertion sort will and hence you will not get partial marks for printing the intermediate steps of the algorithm.
2. The n numbers we give you may be positive, negative or zero. The same number may occur twice in the list too.
3. The number of elements n can be any strictly positive number. Your output must have exactly n lines.
4. As you can imagine, the first iteration of the algorithm has to do nothing since it only has to ensure that the array of the first 1 elements is sorted. However, a single element is always sorted so nothing needs to be done. However, you must still print the array after the first iteration.
5. Be careful about extra/missing lines and extra/missing spaces in your output. There should be no space at the end of any line in your output, nor should there be any extra newlines at the end of your output.

EXAMPLE:
INPUT
4
3 5 4 2

OUTPUT:

```
3 5 4 2
3 5 4 2
3 4 5 2
2 3 4 5
```

Explanation: At the end of iteration 1, the first 1 element of the array is correctly sorted. Since, it is a single element, it is sorted by default. At the end of the 2nd iteration, the first two elements are sorted. Since they were sorted already, nothing had to be done. After the third iteration, the first 3 elements must have to be in sorted order. However, this requires 4 to be inserted between 3 and 5 since that is its correct place in the sorted order of the first 3 elements in the array. In the final step, all the elements must be sorted, and hence, 2 is placed before 3.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

Start Solving! (/editor/practice/6286)