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

Cocktail Sort

LAB-PRAC-14_SORT-MISC

Cocktail 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 simple algorithm is called Cocktail Sort and the reason it is called this name is due to a back and forth behavior it has which is reminiscent of the shaking motion bartenders perform when preparing cocktails at a bar. The Cocktail Sort is a close cousin of another simple sorting algorithm known as Bubble Sort. As the name suggests, bubble sort causes elements to "bubble" up to their correct position in the array.

Cocktail Sort maintains the following invariant. After the first iteration, the largest element in the array should be in the last position of the array (as it should be in a non-decreasing sorted order). After the second iteration, the smallest element in the array must be in the first position of the array (as it should be in a non-decreasing sorted order). After the third iteration, the second-largest element in the array should be in the second-last position of the array (as it should be in a non-decreasing sorted order). After the fourth iteration, the second-smallest element in the array must be in the second position of the array (as it should be in a non-decreasing sorted order) - this process repeats for a total of n iterations after which the array must get sorted.

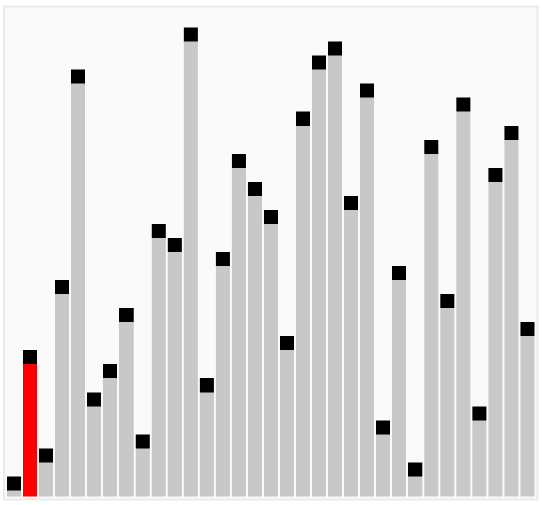
The way cocktail sort accomplishes the above is actually very cute. In odd iterations i.e. 1st iteration, 3rd iteration etc, - it goes from left to right, comparing adjacent elements. If it finds an element that is strictly larger than the element to its immediate right (i.e. the elements are out of order), it swaps the two elements. In even iterations i.e. 2nd iteration, 4th iteration etc, - it goes from right to left, comparing adjacent elements. If it finds an element that is strictly larger than the element to its immediate right (i.e. the elements are out of order), it swaps the two elements.

Verify that if we do this, then in the first iteration, we would end up transporting the largest element of the array to the last position in the array. Also, in the second iteration, we would end up transporting the smallest element of the array to the first position in the array.

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 cocktail 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 animation below shows you how cocktail sort works. Notice that first the algorithm does a left to right sweep in the first iteration, then a right to left sweep in the second iteration and then a left to right sweep again in the third iteration and so on, for n iterations.

Image courtesy: wikipedia.org



Caution

1. Cocktail sort goes over the array from left to right, starting from the index 0, comparing adjacent locations and swapping them if they are out of order, then going from right to left and doing the same thing and repeating this over and over again. However, you should be able to deduce that it does not need to go all the way till the left most and right most indices in later iterations.
2. There may be iterations where nothing needs to be done. However you still have to print the array after that iteration. Your output must contain n lines.
3. Please do not try to cheat by using library functions like qsort(). These will not sort the array in the order bubble sort will and hence you will not get partial marks for printing the intermediate steps of the algorithm.
4. The n numbers we give you may be positive, negative or zero. The same number may occur twice in the list too.
5. The number of elements n can be any strictly positive number, even 1. Your output must have exactly n lines.
6. 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 4 1 2
```

OUTPUT:
```
3 1 2 4
1 3 2 4
1 2 3 4
1 2 3 4
```

Explanation: Initial state of the array is 3 4 1 2
Iteration 1:

1. 3 and 4 get compared. They are in correct order so no change

2. 4 and 1 get compared. They are out of order so they get swapped. 3 1 4 2
3. 4 and 2 get compared. They are out of order so they get swapped. 3 1 2 4

State of array after iteration 1: 3 1 2 4
Iteration 2:

1. 2 and 4 get compared. They are in correct order so no change
2. 1 and 2 get compared. They are in correct order so no change
3. 3 and 1 get compared. They are out of order so they get swapped. 1 3 2 4

State of array after iteration 2: 1 3 2 4
Iteration 3:

1. 1 and 3 get compared. They are in correct order so no change
2. 3 and 2 get compared. They are out of order so they get swapped. 1 2 3 4
3. 3 and 4 get compared. They are in correct order so no change

State of array after iteration 3: 1 2 3 4
Iteration 4:

1. 1 and 2 get compared. They are in correct order so no change
2. 2 and 3 get compared. They are in correct order so no change
3. 3 and 4 get compared. They are in correct order so no change

State of array after iteration 4: 1 2 3 4

Notice that after iteration 1, 4 (the largest element in the array) reaches its correct position in the sorted array. After iteration 2, 1 (the smallest element in the array) reaches its correct position in the sorted array and so on.

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/6295)