Insertion Sort

Go to <http://math.hws.edu/eck/js/sorting/xSortLab.html> to watch the sorts in action.

The Insertion Sort is an entirely different algorithm for sorting. It is not just a different way to code the Selection Sort! Think of picking up a hand of cards, one at a time. The first card is automatically in order. Pick up the second card. **Insert** it in order with the first card, **sliding** over if necessary. Pick up the third card, and repeat. At each step we produce a sorted hand of cards, then get the next item, until we have inserted each item in its proper place.

The first item of data is automatically a sorted sublist of length 1. Now look at the second item in the list. If the second item is smaller then we **slide** the first item over, and put the second item in its place. (Be careful! As the length of the sublists grow, swapping items is not part of the general solution.) Now look at the third item. If the third item is larger than the second then it must be larger than the first also, and the third item is in its correct location. On the other hand, if it’s smaller than the second then we move the second over, and compare it to the first. It might now be in the correct location, but if it is smaller than the first, we move the first over, and so on. Eventually we insert the third item in its correct location. And so on. A list with N items needs N-1 passes.

| **begin** | **3** | **1** | **4** | **1** | **5** | **9** | **2** | **6** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **pass 1** | 1 | 3 | 4 | 1 | 5 | 9 | 2 | 6 |
| **pass 2** | 1 | 3 | 4 | 1 | 5 | 9 | 2 | 6 |
| **pass 3** | 1 | 1 | 3 | 4 | 5 | 9 | 2 | 6 |
| **pass 4** |  |  |  |  |  |  |  |  |
| **pass 5** |  |  |  |  |  |  |  |  |
| **pass 6** |  |  |  |  |  |  |  |  |
| **pass 7** |  |  |  |  |  |  |  |  |

# **Exercises**

# Using the Insertion Sort algorithm, what does this array look like after 3 passes?

| 4 | 3 | 7 | 1 | 3 | 8 | 5 | 6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

# Using the Insertion Sort algorithm, what does this array look like after 3 passes?

| -4 | -3 | -7 | -1 | -3 | -8 | -5 | -6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

| 3 | 2 | 1 | 4 | 6 | 5 |
| --- | --- | --- | --- | --- | --- |

# Let’s think about the Big-O of the Insertion Sort in its best case, average case, and worst case. We will focus on the number of comparisons made by the algorithm. Here is a 6-item array in random order:

# How many comparisons are made to sort this array in ascending order? \_\_\_\_\_\_

| 1 | 2 | 3 | 4 | 5 | 6 |
| --- | --- | --- | --- | --- | --- |

# Here is a 6-item array already in sorted order (ascending order):

# How many comparisons are made to sort this array in ascending order? \_\_\_\_\_\_

| 6 | 5 | 4 | 3 | 2 | 1 |
| --- | --- | --- | --- | --- | --- |

# Here is a 6-item array in reverse order (descending order):

# How many comparisons are made to sort the array in ascending order? \_\_\_\_\_\_

# Conclusion: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Know thy Big-O: the Insertion Sort is \_\_\_\_\_ in the best case, \_\_\_\_\_ in the average case, and \_\_\_\_\_ in the worst case.

# **Lab Assignment**

The code for the insertion sort uses these methods:

**public static void** sort(**double**[] array)

# //returns the index in which to put the new item **private static int** shift(**double**[] array, **int** index, **double** value)

# 

The *driver class* is called InsertionSort\_Driver. It tests your code with both doubles and Strings (the first 100 words of the Declaration of Independence).

Your job is to complete the *resource class* Insertion. A set of methods will sort doubles. Another set of methods will sort Comparables. Complete both isAscending methods.

Notice that SuppressWarnings("unchecked") removes the warning messages.