**Selection & Insertion Sorts**

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

The goal in sorting is to put a list of items in order. We do not know the initial ordering, so the items could be totally random or already in order or mostly in order or backwards—the initial order will make a difference when we talk about efficiency.

If we are to have any hope of sorting a list, we must be able to compare any two items and determine their proper order. For primitive types we just use the built-in less-than operator (<). For objects we either use compareTo(), if the class *implements the Comparable interface*, or compare() if the class *implements the Comparator interface*. If none of these apply, then the objects don't have any order, and it makes no sense to sort the list.

There are many ways to sort a list of items. Two of the simpler algorithms are the Selection Sort and the Insertion Sort.

# Selection Sort

Find the largest item in the list and swap it to the end. That item is now in its correct position and is no longer considered. Now look at the sublist containing the first N–1 items. On that sublist, find the largest item and swap it to the end (the next to last position of the overall list). That item is now in its correct position and is no longer considered. Then move to the sublist containing the first N–2 items, and repeat.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| begin | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 |
| pass 1 |  |  |  |  |  |  |  |  |
| pass 2 |  |  |  |  |  |  |  |  |
| pass 3 |  |  |  |  |  |  |  |  |
| pass 4 |  |  |  |  |  |  |  |  |
| pass 5 |  |  |  |  |  |  |  |  |
| pass 6 |  |  |  |  |  |  |  |  |
| pass 7 |  |  |  |  |  |  |  |  |
| pass 8 |  |  |  |  |  |  |  |  |

# 

The selection sort (in its resource class) will need these methods:

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

**private static int** findMax(**double**[] array, **int** n)

**private static void** swap(**double**[] array, **int** a, **int** b)

# Question: why are these methods in the resource class marked static? What difference does static make?

# Insertion Sort

This 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. To create a sorted sublist of length 2, we examine 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 the first position. (Be careful! As the length of the sublists grow, swapping items is not part of the general solution.) To create a sorted sublist of length 3, we begin with our sorted sublist of length two. So, if the third item in the list 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 item in its correct location.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| begin | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 |
| pass 1 |  |  |  |  |  |  |  |  |
| pass 2 |  |  |  |  |  |  |  |  |
| pass 3 |  |  |  |  |  |  |  |  |
| pass 4 |  |  |  |  |  |  |  |  |
| pass 5 |  |  |  |  |  |  |  |  |
| pass 6 |  |  |  |  |  |  |  |  |
| pass 7 |  |  |  |  |  |  |  |  |
| pass 8 |  |  |  |  |  |  |  |  |

# 

The insertion sort (in its resource class) will need methods

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

**private static int** shift(**double**[] array, **int** index, **double** value)

# Lab Assignments

Part 1: The driver classes are called SelectionSort\_Driver and InsertionSort\_Driver. Complete the two *resource classes*, Selection and Insertion which sort ints. Complete the boolean isAscending methods. (You can comment the Comparable method headers and the calls to those methods to test Part 1.)

Part 2: In the appropriate resource classes, implement each sort to sort Strings, which are Comparables. To test your sorts, complete the isAscending methods for Comparables and sort the first 100 words in declaration.txt. Notice that SuppressWarnings("unchecked") removes the warning messages.