CS 158 Lab 21 April 20, 2010
Timing Sorting (concluded)

1. Begin preparation on a report that compares all of the following sorting algorithms: heap sort, binary search tree sort, bubble sort, quick sort, merge sort, and the version of quick sort provided in **java.util.Arrays**. For each sorting algorithm, start with 10,000 random integers, find the average time to sort with 10 replications, then repeatedly double the amount of data until either it takes too long or you run into memory problems. Try to make sure that all the experiments are done on the same machine, although you may want to compare the results on the Suns with how they are on your own computer. Turn in your printed report on Monday in class. You should have heap sort and binary search tree sort ready from the previous lab. Notes on the other algorithms are below.

2. Create new packages for the remaining algorithms: **bubblesort**, **quicksort**, **mergesort**, and **javasort**. In **quicksort**, create a new class called **QuickSort**. Put the following file into one of your folders: http://faculty.valpo.edu/jcaristi/cs158/IntArraySorts.java. This file contains several static sorting methods that work by passing an array as an argument in the method. I would like you to change any of the methods you use so that instead of passing an array, they work with an array that is a member variable, as we did in the previous two labs. Extract from that file just the methods needed to do the quick sort: **partition**, **swap**, and **quickSort**. Change these methods so that they do NOT use an array argument, but instead use an array that you set up as a member variable. Create JUnit tests for each of these methods. For example, to test partition, you know that after you call partition, the value returned is the subscript of the location where the pivot element will end up, so you can assert that everything to the left of that position must be smaller than the pivot and everything to the right of that position must be greater than the pivot. Set up a main method to run your experiments similarly to the way in which the earlier ones were done.

3. Go back to the file you downloaded and extract the methods required to do a merge sort (**internalMergeSort** and **merge**), and place them into the **mergesort** package in a new class called **MergeSort**. The author has included a print statement that you definitely want to remove before you start running this program with a lot of data items. As in the quicksort step, convert these methods so that they use an encapsulated array. In this case, you will need two arrays: inputArray and tempArray. Also, you should find a couple of extraneous variables the author has created that are never used, and you can remove them too. Do all of the experiments for this algorithm.

4. Move on to the sort that Java provides in **java.util.Arrays**. In the package **javasort**, create a class called **JavaSort** and figure out how to use the **sort** method that is provided for sorting integer arrays. Run the same experiments in a main method of this class.

5. Finally, in the **bubblesort** package create a class called **Bubble** and add an array member variable and a constructor with an integer argument as usual. Add a **bubbleSort** method that sorts the array using the bubble sort algorithm. Try to code this yourself without looking it up or copying code from elsewhere. Ask for help if you get stuck. Add a JUnit test that calls **bubbleSort** and asserts that all the elements are in order. When this all works for an array of size 100, add a main method to the **Bubble** class and run the same experiment as you did for the other algorithms.