**COMP3270 Section 002, Fall 2014**

**Programming Assignment 1**

**Due Oct 23, 2014, 11:59pm**

**Introduction**

In this assignment you will compare the performance of three sorting algorithms: insertion sort, heap sort, and quick sort, on files of integers. You will implement the sorting algorithms according to the pseudo-code given in the textbook, adding code to read and write the input files, and instrumentation code to compute the number of comparisons of elements of the files to be sorted and the number of assignments to array elements by each of the three sorting algorithms. Sort from smallest to largest.

\*\*\* This requirements document may be clarified, updated, or corrected during the period of the assignment. It supersedes any comments made in class \*\*\*

**Input File Format**

Each file will consist of one or more integers between 1 and 32767, possibly with duplicates. No file will contain more than 10,000 integers. Each integer will be on a line by itself, without any commas, decimal points, or negative signs (except the very last line -- the end of the file will be denoted by a line with a negative one (-1) on it. Four test files along with the correct output will be provided for you to test with, but your submission will also be tested on files conforming to the above specification which were not provided to you. Your program should read the file **from the standard input.**

**Output File Format**

For each input file, your program should write output **to the standard output** in the following format. First should come the integers from the sorted input file in sorted order, in the format:

*userid algorithm #i: n*

where *userid*  is replaced by your Auburn userid (e.g. “roc0017”), where *algorithm* is replaced by one of the words: “insertion”, “heap”, or “quick” (in that order)**,** where *i* is replaced by the index of the value you are printing in sorted order (e.g. “1” for the smallest element in the file that you have sorted), and where *n* is replaced by a value from the sorted list. Do not include the terminating -1 in the sorted list or produce it in the output.

Following each of the three copies of the sorted list, you will produce summary statistics about the number of comparisons and assignments that sort required. That output should be of the format

*userid algorithm* **comparisons**: *n* **assignments:** *m*

where *userid* and *algorithm* are as above, *n* is the number of comparisons made by the algorithm, and *m* is the number of assignments. Calculate assignments as follows:

* Count any assignment where an element of the array to be sorted is on the left hand side of the assignment statement
* Do not count assignments to temporary variables, or loop index variables

Count comparisons as follows:

* Count any comparison between two array elements using <, >, ==, !=, >=, or <=
* Count comparisons between an array element and a temporary variable
* Count comparisons between two temporary variables IF they contain array elements rather than indexes.
* Do not count comparisons of loop index variables, either **for** or **while** loops

**Example of Format**

An input file might look like

10

1021

45

32000

5

-1

The output files should look like

roc0017 insertion #1: 5

roc0017 insertion #2: 10

roc0017 insertion #3: 45

roc0017 insertion #4: 1021

roc0017 insertion #5: 32000

roc0017 insertion comparisons: 25 assignments: 12

roc0017 heap #1: 5

roc0017 heap #2: 10

roc0017 heap #3: 45

roc0017 heap #4: 1021

roc0017 heap #5: 32000

roc0017 heap comparisons: 15 assignments: 38

roc0017 quick #1: 5

roc0017 quick #2: 10

roc0017 quick #3: 45

roc0017 quick #4: 1021

roc0017 quick #5: 32000

roc0017 quick comparisons: 21 assignments 15

NOTE: The numbers of comparisons and assignments are NOT correct in the output, which is intended to show format only.

**Code Guidelines**

You should implement the sorting algorithms in Java. I suggest a single class with all the algorithms, but the object-oriented analysis is left up to you. You should name each method as it is named in the textbook, and your code should mirror the textbook pseudo-code in terms of the parameters and return values, and line-by-line in the body of each method except for the code to update the counts of comparisons and assignments. You may add a parameter for the array size if you want, for insertion sort and heap sort. You should copy the input array to another array before running each sort algorithm, since all three of them sort the array in place.