Assigned: Friday, 28 March 2014, **Due: Sunday, 6 April, 11:59pm.**

Three Sorts

1. Overview

This assignment will test your ability to write three different sorting algorithms in C++. We have discussed a number of different sorts, and you will have the freedom to choose from the myriad of different sorting algorithms that have been developed.

2. Implementation Specifics

You need to write three programs that are able to sort integers that are input through the "standard input" (stdin) stream. In other words, numbers can either be typed into the command shell, or they can be read in from a text file of integers (see below for an example of the latter). The numbers will go into an array in the following `struct`:

```
struct arrayWithLength{
        int length;
        int *arr;
};
```

As you can see, the struct gives you the length of the integer array as well as a pointer to the array itself.

Output from the sorted array will be printed to standard out (stdout) using the `printArray()` function, which you need to write.

3. The Three Sorts

The following is a breakdown of the sorting programs you can write:

A) One of your sorts must be **selection sort**. The sample code you will be given has a SelectionSort class that you should use to write the code, but the algorithm is in our class notes or online (see Section 6 for a list of references)

B) One of your sorts must be from the following list:

shell sort, merge sort, quicksort, insertion sort

C) Your final sort can be from the previous list, *or* it can be any other reasonable sorting algorithm, except for **bubble sort** or **heap sort** (only because we just wrote heap sort in lab). "Reasonable" is defined as a sort that can be completed for all of our test inputs in three minutes or less. That precludes sorts such as Bogosort (see: http://en.wikipedia.org/wiki/Bogosort), sleep sort (see: http://stackoverflow.com/questions/6474318/what-is-the-time-complexity-of-the-sleep-sort), or other sorts that, while in principal could sort, are not feasible as general purpose sorts. Ideas for out-of-the-ordinary (or non-comparison) sorts are **radix sort, tim sort, bucket sort, library sort, quick3**.

4. Describing Your Sorts, and Finding Sorting Algorithms

In your `readme.txt` file, you need to **describe the algorithm for each sort in your own words**. This is especially important if you chose an out-of-the ordinary sort. **Include a discussion of the worst case and best case complexity, and justify your response.** If you include an average complexity, please provide a reference or a proof. You should have a section in your `readme.txt` file dedicated to each sort. You should also have a list of references where you found the algorithm for your sorts.

You may use offline or online resources to find the algorithm for your sorts. **DO NOT** copy C++ code from the internet! Almost all sorts have very detailed algorithms online, written in both C++ and other languages. We can't stop you from looking at code samples of the algorithms, but you are on your honor to translate only algorithms and not specific code into C++. In other words, you are free to look at algorithms (indeed, you must), but you can't copy specific code.

5. Other Comments

- The `Makefile` (provided) will create three executable files:
    - selectionSort
    - sort2
    - sort3
- See the Makefile itself for details on the names for the files you need to produce.
- Testing for this assignment will include timed tests as well as functionality tests, with varying input sizes and element orderings. If your sorts do not finish within the three minute time-limit for our input data, we will consider them to be incorrect. You should test your code on $O(n^2)$ sorting algorithms on input sizes up to 100,000 elements. You will have test input files that have that many input integers.
- In order to test your sorts, you can either type (or paste) a list of integers (separated by newlines) into the terminal (followed by ctrl-D), or you can input them directly from test files, as follows:

```
./selectionSort < randList100000.txt
```

- As always, direct any specific questions to the Piazza class site.

6. References

Feel free to use any of the following sites to discover sorting algorithms:

- http://en.wikipedia.org/wiki/Sorting_algorithm
- http://www.sorting-algorithms.com
- http://betterexplained.com/articles/sorting-algorithms/

7. Low Level Details

Getting the files

The files are located at `/comp/15/public_html/assignments/hw4/files`

To get the files: `mkdir hw4 && cd hw4 && cp /comp/15/public_html/assignments/hw4/files/* .`

Compiling and providing

1. Use the included `Makefile`:
   - To make all:
     ```
     make
     ```
   - To make individual sorts:
     ```
     make selectionSort
     ```
     (or `sort2` or `sort3`)

2. At the command prompt, enter “`make provide`”