CPSC 223: Project Part #2 *Due Tuesday, November 19th*

Overview. In this part of the project, you will implement one of your algorithms and create a "test harness" to begin measuring performance.

Instructions: With your group, complete the following steps.

1. Implement at least one of your sorting algorithms. Your algorithm should be written using a function that takes an array and an array length as arguments. It is okay to use helper functions as well. It is also okay to assume that your algorithm works over an integer array (i.e., that the values being sorted are integers).

2. Implement a main program that runs your implementation over a given test case and generates timing numbers. Your main program should accept a file name as an argument (containing the data set for the test case) and print the timing results of running the algorithm over the test case. Your main program should be implemented in a file called "project.cpp". You are free to use the HW6 performance test program as a starting point.

3. Generate a set of input data sets of different sizes to use as a testing "benchmark". One approach for doing this is shown below. You will be generating a relatively large number of datasets to run your algorithms on.

4. Turn in the hardcopy of your code, a description of the test files you generated, and the results of executing your first sorting algorithm over the files (i.e., the output of running your program over the data files). Note that the results you produce in this part are only to demonstrate that everything is working.

Benchmark Test Data. You are free to use whatever approach you would like for generating test data including writing a separate program or using the tools described below. For the project, you will need to create at least two "batches" of test files. One batch should be for small-to-medium sized data sets (e.g., in the thousands, tens, or hundreds of thousands), and one batch should be for large-sized data sets (e.g., in the millions, tens of millions, hundreds of millions, etc). Each batch should have around 9 or so data set sizes. For example, if your first batch is in the thousands, you should have data set sizes of 1000, 2000, 3000, etc., up to 9000. For each data set size, you should have a "random" case and a "worst" case to run your algorithms on. This means you will have around 36 total test data sets. Note that you may need a fewer or larger number of data sets in each batch. In particular, the goal is for you to use enough data sets to graphically show the algorithm's general runtime complexity (which isn't always the case using our five test cases, for example). Also note

that you will have two sets of graphs—one for the "average" case and one for the "worst" case.

Tools for Generating Data Sets. One approach for generating test data sets is to use the following built-in unix/linux command-line tools (available on `ada` as well as the department virtual machine). The command-line tool `shuf` (which stands for "shuffle") can generate a list of (pseudo) random numbers. For instance, the following outputs a list of 5 random numbers between 50 and 100.

```
bowers@ada:~$ shuf -i 50-100 -n 5
58
94
67
73
72
```

To generate a larger set of random numbers change the `-n` argument (e.g., from 5 to 1000) as well as the range of numbers to select values from via the `-i` argument (e.g., to `1-1000`). The above command will not output duplicate values, and so if the value of `-n` is larger than the range of values given by the `-i` argument, then the number of outputs will be the same as the size of the range given (still in random order). To save the numbers generated to a file, redirect the output to a file using the `>` operator:

```
bowers@ada:~$ shuf -i 50-100 -n 5 > data.txt
```

If the `data.txt` file does not exist it will be created by the above command. If it does exist, it will be overwritten with the values output by `shuf`. To output a sorted list of n numbers you can use the `seq` command. For example:

```
bowers@ada:~$ seq 5
1
2
3
4
5
```

Again, the result can be redirected to a data file as shown above. Finally, to obtain a list of numbers in descending order, you can combine the `seq` command with the `tac` command (where `tac` is `cat` reversed). For example:

```
bowers@ada:~$ seq 5 | tac
5
4
```

```
3
2
1
```

To redirect the output to a file you can do the following:

```
bowers@ada:~$ seq 5 | tac > data.txt
```

To see the contents of a file, you can use `cat`, `more`, or `less` command followed by the file name. The latter two will show only one screen's amount of lines of the file at a time. The `more` command will allow you to move forward through the file one screen at a time by pressing the space bar. The `less` command allows fine-grained movement through the file by using space to move one screen at a time or the up or down arrows for one line at a time. To exit the `more` and `less` commands type `q` (for quit).