# CSCE 310 Data Structures & Algorithms

Fall 2004
Steve Goddard

Homework 3, October 7, 2004
(Total of 100 points)

Chapter 4 and 5.1 (Insertion Sort)

Due: 9:00pm Thursday, October 21, 2004

---

## Exercises From the Book (40 points):

4.1 #2 (15 pts)
4.2 #6 (10 pts)
4.2 #10 (10 pts)
5.1 #6 – Explain your answers (5 pts)

---

## Programming Exercise (60 points):

For this assignment you will be reading in the inventory file, and sorting the contents using a hybrid sorting algorithm.

The inventory file format is as shown at http://cse.unl.edu/~goddard/Courses/CSCE310J/Project/InventoryFile.csv. For this assignment, you should know that the first line of the file consists of header information. There is a long series of empty headers spaces which corresponds to a set of subheaders in the second line. For the purposes of this assignment, know that the sortable headers are on the first line, and the non-sortable headers are on the second line. The third line is where the actual data starts.

Your job is to read this file in and store each product in an object. Then, you must write an algorithm that can sort the items based on which fields that the user wants the data sorted by. This algorithm must be a hybrid sorting algorithm that uses quicksort to perform sorting until the partitions being sorted reach a size that you (the programmer) choose, at which point you sort the remaining part of the partition using insertion sort. You should be able to justify the cutoff point you choose in the analysis.

The user will invoke the program **inventory_sort** using the following command line arguments:

```
inventory_sort filename field1sort [field2sort] [field3sort] ...
```

For example if `inventory filename "Production Quantity" "Primary Vendor"` is run, it will sort each product by `Production Quantity` first. Then, if any products have the same `Production Quantity`, they will be sorted by `Primary Vendor`. If the products are otherwise identical, your sorting algorithm should be **stable**, so if `Product1` was before `Product2` in the original file, the resulting output should also have `Product1` before `Product2`.

As output, your program should print all data in the original file to the console, including the header and subheader information. The output should be sorted, of course.

**Notes**:
In order to make this assignment gradable I will be using an autograding script. This means you must follow the formatting of the input file EXACTLY for your program to output results that the grader will recognize as correct. This means your program should not output extra spaces after commas, extra spaces at the end of lines, or extra lines at the end of the file. Such extraneous characters will result in the loss of points.

You will also notice that in the sample input file, any `Item Name` that contains a quote character in the name is surrounded by quotes itself, with the actual quote character represented by a double-quote (look at the `"Bolt (1/2"")"`). Also, the `Acquisition Preferences` field is surrounded by quotes, but only if there is a comma involved in the list (since there are more than 1 preference). Therefore, in all cases your program should:

1. Evaluate strings that are surrounded by quotes by remove the surrounding quotes and evaluating "" to a single ".
2. Sort based on the evaluated strings (i.e. sort `Bolt (1/2")` and not `"Bolt (1/2"")"`)
3. Reformat strings for outputting. This means that if the string contains a comma or a quote, then you should add quotes around the whole string and replace any " with "" inside the string.

Note that you do NOT need to know specifically which field the string is in to perform these 3 operations. The program should respond in exactly the same manner if, for example, a vendor name contains a comma and is therefore surrounded by quotes (i.e. if `Frannie's Fancy Emblem Mfg.` was actually `"Frannie's Fancy Emblem, Inc."`, you would need to add the quotes because the name itself contains a comma).

**Analysis**: Prepare a report in which the following are considered.

- What cutoff point did you choose for the transition between using quicksort and using insertion sort?
- Why did you choose the cutoff point you did for the insertion sort?
- Give a quantifiable reason for why your cutoff is better than other points and better than a quicksort that doesn't use insertion sort. You may want to use a comparison of efficiency classes.

This analysis report should also (as in previous assignments) describe how your program works, what design choices (if any) were made and why, and any known defects. You may also discuss enhancements you think should be made, but were beyond the scope of the assignment (i.e., inadequacy of the specifications).

**Deliverables**:
You should turn in your analysis document, all code, and a makefile to build your program into the web-handin. You should also submit a hard copy of your analysis to me. I should be able to type `make` to make your program.

This problem will be scored at follows:

| | |
|---|---|
| Program correctness (as defined) | 30% (18 pts) |
| Quality of design/readability | 15% (9 pts) |
| In-line documentation/coding standard | 20% (12 pts) |
| Design and analysis document | 25% (15 pts) |
| Thoroughness of test cases | 10% (6 pts) |