

1 Introduction

In this lab you will be doing a counting analysis (not timing) of the sorting algorithms you saw in COSC 120 and 220. Specifically the bubble, insertion, selection, merge, and quick sorts. You already know the complexities of all these sorts so the results of this exercise will not be very surprising. The main point of this is to look at one of the possible techniques for counting when the mathematics of doing the analysis exactly becomes difficult. That is, we place counters in the algorithm implementations and run the code on different sizes of problems and look for trends in the counts.

In this exercise you will take the implementations of these algorithms that are given in the Sorts.h file, have each function return the number of pertinent operations that were done and return these counts. You will then graph your results and submit a short report of your findings.

2 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Lab02, put each programming exercise into its own subdirectory of this directory, zip the entire Lab02 directory up into the file Lab02.zip, and then submit this zip file to Lab #2.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Make sure that each project includes a makefile that will compile the program on the Linux HPCL lab machines.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyCLasses.

In this lab you will also be doing a counting analysis (not timing) of the sorting algorithms you saw in COSC 120 and 220. Specifically the bubble, insertion, selection, merge, and quick sorts. For these you will be

1. Plotting your results using LibreOffice Calc.
2. Transferring the charts to a LibreOffice Writer document.
3. Answering any questions about the analysis which are to be written in the LibreOffice Writer document.
4. Export the LibreOffice Writer document to a PDF file (there is a button in the toolbar to do that).
5. This document will be zipped up with your code files into one zip file to be uploaded to MyClasses.

3 Exercises

1. Take the Sorts.h header file that was supplied with the lab and edit each of the functions so that,
 - (a) They return a long integer that will be the operation count.
 - (b) Count each arithmetic operation, this includes incrementing, decrementing, addition, subtraction, multiplication, division, etc. Count each comparison and count each assignment. The total of these is what the function should return.
 - (c) You do not need to count logical operations like `&&`, `||`, `!`, `^`, or their bit-wise counterparts.

- (d) You do not need to count data allocations or parameter passing (although these are assignments under the hood). If when passing a parameter there is an arithmetic operation or a conditional statement in the parameter, these are to be counted.
- (e) Do not count the operations that the counter is doing.
2. Write a main program that will ask the user for the array size, the type of array (Best, Worst, or Average), and if they would like to include the slower sorts. Our arrays will simply be arrays of integers. Best case is an array that is already sorted, worst case is an array in reverse order, and average is a random array with integer entries in the range of [0, 1, 000, 000]. The output of the run will be operation counts for each sort. For example,

```
Array size: 1000
Array type (1 = Best, 2 = Worst, 3 = Average): 1
Include slow sorts (1 = Yes, 2 = No): 1
```

```
Results:
Bubble Sort Operation Count: 3003996
Selection Sort Operation Count: 1509502
Insertion Sort Operation Count: 9002
Merge Sort Operation Count: 95906
Quick Sort Operation Count: 25173
```

```
Array size: 10000000
Array type (1 = Best, 2 = Worst, 3 = Average): 1
Include slow sorts (1 = Yes, 2 = No): 2
```

```
Results:
Merge Sort Operation Count: 2030135794
Quick Sort Operation Count: 530948147
```

3. Do runs for best, worst, and average cases for all the sorts using array sizes of 1000, 5000, 10000, 20000, and 30000.
4. Do runs for best, worst, and average cases for just the two fast sorts using array sizes of 1000000, 2000000, 4000000, 5000000, 7500000, and 10000000.
5. For each of the three slower sorts, bubble, insertion, and selection, graph the three different array type scenarios on one graph. That is, three graphs with three lines each on them. So one graph will be the best worst and average counts for bubble sort. The second is the same for the insertion sort and the third is for selection.
- Copy the chart over to the LibreOffice Writer document (copy-paste operation — Ctrl+C then Ctrl+P or use the mouse or menu).
 - Discuss any trends you see in the results from the graphs.
6. For each of the two faster sorts, merge and quick, graph the three different array type scenarios on one graph. That is, two graphs with three lines each on them. So one graph will be the best worst and average counts for the merge sort. The second is the same for the quick sort. Use the data from the larger runs you did.
- Copy the chart over to the LibreOffice Writer document (copy-paste operation — Ctrl+C then Ctrl+P or use the mouse or menu).

- Discuss any trends you see in the results from the graphs.
7. Graph just the average case for the three slower sorts on one graph. So one graph and three lines. One line for bubble, one for insertion, and one for selection.
- Copy the chart over to the LibreOffice Writer document (copy-paste operation — Ctrl+C then Ctrl+P or use the mouse or menu).
 - Discuss any trends you see in the results from the graph.
8. Graph just the average case for the two faster sorts on one graph. Use the data from the larger runs you did.
- Copy the chart over to the LibreOffice Writer document (copy-paste operation — Ctrl+C then Ctrl+P or use the mouse or menu).
 - Discuss any trends you see in the results from the graph.
9. PDF the document and include it with your main and altered Sorts.h file. As usual, include a make file that will compile the program on the Linux lab computers.