CSCI 3412 – Algorithms
Dr. Williams
Program 1 – Sorting Algorithms

Due: September 24, 2018

In Program 1, each student will select three (3) sorting algorithms to analyze and implement. There will be static analysis of the algorithms – based on the written pseudo-code – as well as analysis of the performance of the algorithms – based on the execution of the code.

I will give you several input files that will allow us to analyze various aspects of the code. All the data are integers between 0 and 999 inclusive and you will be given the following data sets:

1. shuffled.txt – the integers 0-999 in random order – i.e., shuffled
2. sorted.txt – the integers 0-999 in sorted order (ascending order)
3. nearly-sorted.txt – the integers 0-999 in nearly sorted order
4. unsorted.txt – the integers 0-999 in unsorted order (descending order)
5. nearly-unsorted.txt – the integers 0-999 in nearly unsorted order
6. duplicate.txt – 1000 integers 0, 10, 20, …, 990 – i.e., many duplicates – in random order
7. one-million-randoms – 1,000,000 integers 0-999 in random order

All the files have the following format with one entry per line:

Description of the data set
<n>
A[1]
A[2]
…
A[n]

Part 1 – Select Sorting Algorithms

Each student needs to select three (3) sorting algorithms to analyze and implement. The first algorithm needs to have an expected running time that is $O(n^2)$ – i.e. a naive sorting algorithm like bubble sort or selection sort. The second algorithm needs to have an expected running time that is $O(n \lg n)$ – the "normal" rate of growth for comparison sorts like merge sort or quick sort. The third sort algorithm needs to have an expected running time that is $O(n)$ – which is only possible for non-comparison-based sorts where we know – and take advantage of – some aspect of the data that allows us to avoid comparing the elements to be sorted. [In this case, the fact that all the values to be sorted are integers between 0 and 999 inclusive.]

Part 2 – Write Pseudo-Code for each of the algorithms

Write (in the same style as the text) the pseudo-code for their selected algorithms. Most – if not all – of these are in the text or available from other sources. Properly document the source of your algorithms.

Part 3 – Static Analysis

Write a loop invariant statement for the main loop for each of your algorithms. [Just the main loop is fine here.]

Show that the algorithms are correct by showing that the loop invariant holds for the first iteration of the main loop, is true for any iteration of the main loop assuming it was true for the previous iteration and use the loop invariant to show that the algorithm performs as expected at the termination of the main loop. [Note that your algorithm may be specified iteratively – using actual loop statements – or recursively – using recursive calls.]

Part 4 – Implementation

Implement each of the three algorithms in a programming language of your choice. To perform the analysis in Part 5, you will need to instrument the algorithms to count the number of comparisons performed and the number of array assignments – essentially, the number of times elements are exchanged. [There may not be an exchange, per se, since one of the values may be in a local variable during an inner loop.]

Part 5 – Analysis

Analysis of Growth

Using the one-million-randoms.txt file, show the rate of growth of comparisons and array assignments for each algorithm by plotting these values for each power of ten – 1*, 10, 100, 1000, 10000, 100000, 1000000. [For 1, the comparisons and assignments should be 0 since the algorithm has nothing to do. We are just including it to give a common starting point for the comparisons.] Does it meet your expectations in terms of growth?

Comparison Across Data Sets

Using the other six data sets, which each have 100 values, show how each algorithm performs against the different data sets – shuffled, sorted, nearly sorted, unsorted, nearly unsorted, and multiple duplicates.