Jackson Nahom
4/7/17
CS 344
Profesor Maciel

Sorting Algorithms Testing Experiment

The purpose of this project is to experiment and compare the running times of comparison-based sorting algorithms that have been taught in this course so far. This lead me to implement four sorting algorithms. However, I also used selection sort and mergesort with code provided in class. On top of that, I used the STL sorting algorithm as well for testing. The four sorting algorithms that I implemented were an iterative version of insertion sort, quicksort using the first element as the pivot, a randomized quicksort, and a randomized quicksort using in-place partitioning algorithm. I implemented these four arrays and they all sort the arrays given to them correctly.

After implementing the arrays I created a class of Integer objects with the purpose of testing the runtime of the previous mentioned sorting algorithms. The main purpose of the class Integer is to keep track of every time an Integer is compared or copied to keep track of that number with a global variable Integer_count. Finally came the time to test the algorithms with the Integer class. I did this by testing three kinds of arrays: sorted, reversed, and random. This lead to a table:

	SS	IS	MS	DQS	RQS	RQSIP	STL
Sorted Array	50044885	29994	599072	300059991	297610029	300009997	178926
Reversed Array	50044885	124930014	606456	203201240	121110022	241435621	758841
Random Arrays	50044885	112714135	651242	30305446	30271431	30278861	351246

SS = Selection sort   IS = Insertion Sort MS = Mergsort DQS = Deterministic quicksort

RQS = Random quicksort RQSIP = Random quicksort in place   STL = STL's sorting algorithm

The above table shows how many times each algorithm for each array either compares or copies an Integer. The numbers above are are the Integer_counts for each run of the algorithm with the n being used 10,000 for 10,000 elements in each array. The random arrays are an average of three random arrays.

Some of the results are expected, however most are not as expected. All the algorithms relate to Insertion sort in some way. I start by calling T the sorted array of Insertion sort, therefore T = 29994. Selection sort is suppose to be $T^2/2$ $= 449,820,018$ for each kind of array; however SS is 50,044,885 for each array, that is a lot less Integer counts than expected. As for Insertion sort the reversed array is suppose to be $T^2/2$ $= 449,820,018$ and the random arrays $T^2/4$ $= 224,910,009$. Insertion sort runs faster than expected for both reversed and random arrays. For merge sort it should run at $T logn$ $= 119,976$. While all the mergesort run in similar integer counts, they run much slower than expected. The Deterministic quicksort is suppose to be

$T^2/2$ for sorted and reverse and $T \log n$ for random arrays. The integer counts for every array is much slower than expected. Finally I get to the Random quicksort, Random quicksort in place, and STL's sort algorithms. All these algorithms are suppose to be $T \log n$. Both the Random quicksort and Random quicksort in place run much slower than expected, however the STL sorting algorithm is the most close to the expected outcome.

There are some interesting results from this experiment. The most interesting results are from the quicksort arrays. My only logical conclusion for the quicksort arrays is that I did not implement the quicksort algorithms completely correct. I am also a little confused on why the STL sorting array is not more similar to the expected results. In the end this was a very informative experiment, in which I now understand these sorting algorithms more fluently now.