Project 1

Algorithms and Data Structures project on implementing various sorting algorithms to then capture and compare running times on distinct sorted and unsorted arrays

Group Members:

Yoel Ben-Avraham

Kristina Kolibab

Class:

CS 344 - Algorithms and Data Structures

REFLECTIONS:

In completing this project, we implemented several sorting algorithms and produced a test driver to evaluate the efficiency of the various algorithms on various types of arrays (IE sorted vs unsorted). As 'efficiency' here is measurable by the number of operations performed on our Integer objects, we analyzed our results as a function if the number of operations performed when sorting arrays of 10,000 elements - a graph of this can be seen below:

	SS	IS	MS	DQS	RQS	RQSIP	STL
Sorted Array	50024997	29997	262832	150034995	490357	435492	201563
Reversed Array	50024997	100009998	271632	150034995	500586	477335	169098
Random Array (averaged)	5.0025e+07	5.02914e+07	268312	510816	490706	431949	274400

We also analyzed our actual results versus those which we expected as per the project guidelines. See our results below:

Let's make the integer T = Operations performed by insertion sort of an array of increasing sequential numbers = 29997. This should be our smallest number of operations.

Insertion sort on a reversed array performs 100009998 operations, which should be about 4.4991e+08. We can see that this is about four times smaller than we expected.

Insertion sort on random arrays performs 5.02914e+07 operations, which should be about 2.24955e+08. We can see again that this is more than 4 times smaller than we expected.

Selection sort should perform about 4.4991e+08 operations on every array - actually: 50024997, 50024997, and 5.0025e+07. We can see that we are near to ten times smaller for our experimental data!

Mergesort should do about 276283 operations - actually: 262832, 271632, and 268312
Notice that our experimental data is almost perfect here!

Quicksort One (A deterministic version) should perform about 8.9982e+08 operations on the sorted and reversed arrays. It actually performs 150034995 and 150034995 operations respectively. This is about a tenth of our expected value!

Deterministic quicksort should also perform about 276283 operations on random arrays - it actually performed an average of 510816. This is nearly twice as long as expected!

Both types of randomized quicksort, as well as the STL algorithm, should perform 276283 operations on every array. Actual values for randomized quicksort are 490357, 500586, and 490706. Actual values for randomized quicksort with the in-place partitioning algorithm are 435492, 477335, and 431949. Actual values for the STL algorithm are 201563, 169098, and 274400. As you can see, our predictions were spot on for the STL algorithm. Our own implementations of randomized quicksort, however, took about twice as long as expected to run.

In summary, we can see that our run times were not quite matched with the expected values, but not altogether too far away. Our maximum differences were about 10 times larger/smaller than we expected, but most were closer to 2 or 4 times smaller/larger than expected. There are many potential explanations for this relatively minor variation. Some amount of randomness was expected from the start of this project, and our array length of ten thousand, while not small, is still notably finite - our results would likely conform even more closely to our expectations had we run our tests on larger arrays. Finally worth consideration is the possibility that we implemented our own algorithms imperfectly. This seems unlikely because we have tested and debugged all of our implemented algorithms in completing this project, but we would be remiss not to mention the possibility. Since our results appear valid, we may surmise that our project is now complete and fully functional. If we were to continue work, however, our next step would probably be to test our algorithms on larger arrays to further minimize mathematical error, and perhaps to run additional tests on our algorithms to cement our trust in the correctness of their implementation. In completing this project we found it particularly interesting and surprising that our errors tended towards a smaller runtime than we expected. While we hypothesize that this is due to our limited array lengths, this is merely an educated guess and would require additional experimentation to verify or disprove.