CS 344 Project 1:

Damon Gwinn, Eric Sognefest.

Introduction and overview

For the project, we implemented Selection Sort (SS), Insertion Sort (IS), Mergesort (MS), Quicksort where the first element is the pivot (DQS), a randomized Quicksort without an in place partition (RQS), and a randomized Quicksort using an in place partition (RQSIP). We compared the results for each and also included the Sort algorithm from the Standard Template Library (STL). For the comparisons, we used the number of operations as a baseline for comparison and timed the algorithms using the system time library.

Comparison Table

We tested each sorting algorithm on arrays of size 10,000 with random elements in the range 1-500. These were created as per instruction. The number of operations and the time taken (ms) are recorded for each sorting algorithm on a sorted, reversed, and random array. All of the data was taken from a single Apple computer.

	SS	IS	MS	DQS	RQS	RQSIP	STL
Sorted Array	50044996	39997	341840	880876	740020	409332	70006
Time (ms)	270.052	0.211886	3.40566	5.55894	5.15449	2.11615	0.320356
Reversed Array	50044996	100019998	346240	776182	783169	466900	145093
Time (ms)	263.638	441.422	3.43302	5.11717	6.12773	2.41741	0.6801
Random Array	50044996	50023168	397701	905107	904045	458468	441208
Time (ms)	264.08	214.334	4.41032	6.71306	6.7094	2.78934	2.62921

Analysis

A) The smallest number of operations achieved by Insertion Sort on as sorted array with only 39,997 operations. It also had the shortest running time at 0.211886 ms. This was our expected result.

B) For Insertion Sort on a reversed array, if we take $\mathrm{T} = 39{,}997$, the expected amount of operations is $\frac{T^2}{2} = 799{,}880{,}004.5$. Our data shows 100,019,998 operation which yields a large percentage of error. However the operations are on the same order as the expected result, that is, off by a constant factor of approximately 8. Thus our data proves reliable to the expected result.

C) On random arrays, the expected amount of operations is $\frac{T^2}{4} = 399{,}940{,}002.3$. This is half the expected number of operations from part B. If we compare this to our actual results of 50,023,168 operations, it is roughly half of our results from part B. This displays the expected trend. Our original analysis of the expected results from part B appears to be overly pessimistic, however, only by a constant factor.

D) The expected output of selection sort is $\frac{T^2}{2} = 799{,}880{,}004$. We compared this to our result of 50,044,996 operations for every array, which is on the order of $\frac{\mathrm{T}^2}{4}$. This is faster than expected.

E) Our analysis of Mergesort yielded approximately 346,240 operations for each array and a running time of around 3.5 ms. The expecting number of operations is 159,988. Our overly pessimistic analysis of Insertion Sort in part B could be the culprit.

F) Deterministic quick sort for all arrays was around 850,000 operations. This was rather perplexing. After checking our code to ensure everything was done right, we have decided that there must be some hidden optimizations done by the GNU and Clang compilers that cause this to run faster than we expected.

G) On a random array, Deterministic Quicksort, performs Tlog(n) operations as we expected. We concluded the overly pessimistic analysis of Insertion Sort on a sorted array to be the reason behind being off by a constant factor.

H) The randomized version of Quicksort along with the STL Sort algorithms perform around Tlog(n) operations. This is the trend we observed with the STL algorithms being very fast surprising us in many ways with its speed.

Our data follows most of the trends expected from our analysis. It looks like our analysis of part B was overly pessimistic causing our data to be off by a constant factor. We decided that for Deterministic Quicksort, there must be some sort of optimizations being done with the GNU and Clang compilers. We are perplexed by it's efficiency.