Discussion

In a broad sense, the results from the tests were as to be expected. Bubble and insertion sort were expected to run in n^2 time in most cases with the exception of the already sorted array being sorted in n time. The 10 percent shuffled array also did slightly better but still not as well as the fully sorted array as seen in tables 1 and 3. The selection sort also performed with the expected n^2 time in every case. You can see this in table 2 because every test for a single size will be raised to the same power signifying that the complexity for each test is approximately the same. Merge sort also exhibited this behavior as seen in table 4. This is to be expected because merge sort is in θ(n log n). The final test represented in the quick sort in table 5. Time complexity for quick sort is n^2 in the worst case and $nlogn$ in the best case. The end time complexity of the sort is determined on the pivot element that is chosen for the sort which is why there is variation between tests for how the algorithm performs. From the data set it is clear that both the merge sort and the quick sort are better choices than the three more basic algorithms. However, which algorithm is better between the merge and the quick sort is depended on the requirements of the larger program using the sort. Merge sort is better from a time complexity standpoint because it is guaranteed to run in $nlogn$ time but the space complexity can range from O(n) to O(n) + O(log n) because of the extra space needed for the sub arrays that are created. Conversely, the quick sort is not always as efficent from a time perspective – although it still runs in an average of $nlogn$ time – but since it sorts the array in-place it has a space complexity of O(1) (constant space complexity). Therefore, if time is the most valuable resource then it is best to use merge sort and if space is the most valuable resource then it is best to use quick sort.

Statistical Analysis

As discussed in the previous section, the means for each algorithm was more or less within the order of magnitudes that were expected from the theoretical analysis. For selection and bubble sort the general trend for the standard deviation seemed to be that it grew as the size of the array grew. This means that as the array got bigger the data points tended to fall further from the average. The insertion sort seemed to be more erratic than the selection and the bubble sort, although in general the standard deviation still grew with the size of the array. Mergesort seems to follow the same pattern of insertion sort but the quick sort is considerably more erratic.

Conclusion

In conclusion the best sorting algorithms were the quick sort and the merge sort. Each algorithm performed slightly differently and throughout the different tests that we ran on them. This project was also an excellent oppertunity to improve knowledge of object oriented design.