**Summary**

Standard sort takes the shortest time to run.

Merge sort and quick sort take a relatively short time to run and have the lowest number of comparisons and swaps, ultimately putting them at the top for most efficient sorting algorithms among the 6 implemented.

Merge sort, quick sort, and selection sort have a steady and small number of swaps, while bubble sort and insertion swaps happen at a much higher volume.

**Thesis**

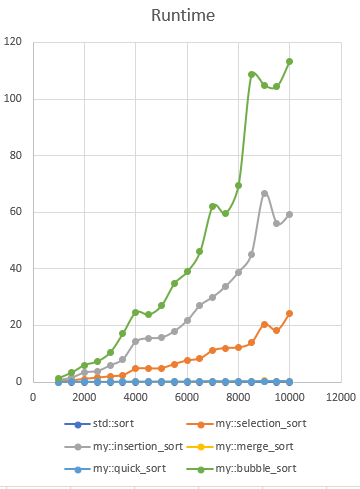
Big-O analysis for algorithms (average case scenario):

Selection sort: O( Insertion sort: O() Merge sort: O()

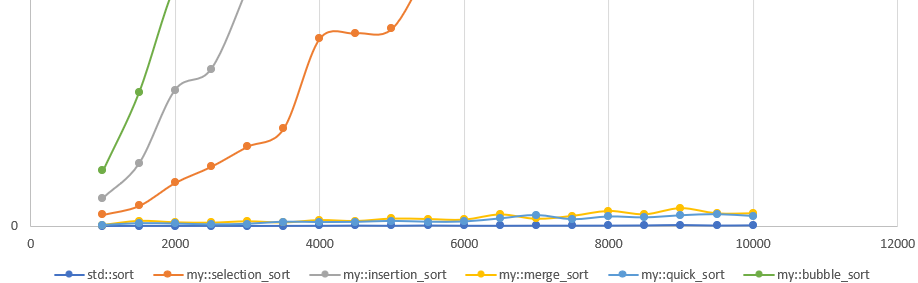
Quick sort: O( Bubble sort: O()

According to the theses, selection sort, bubble sort, and insertion sort should take about the same time complexity, with the chance of insertion sort being more efficient because of its best case scenario of O(n). Merge sort and quick sort’s time complexity should be relatively similar and smaller than that of selection and insertion sort ( < ); they should also be the least time-consuming among the 6.

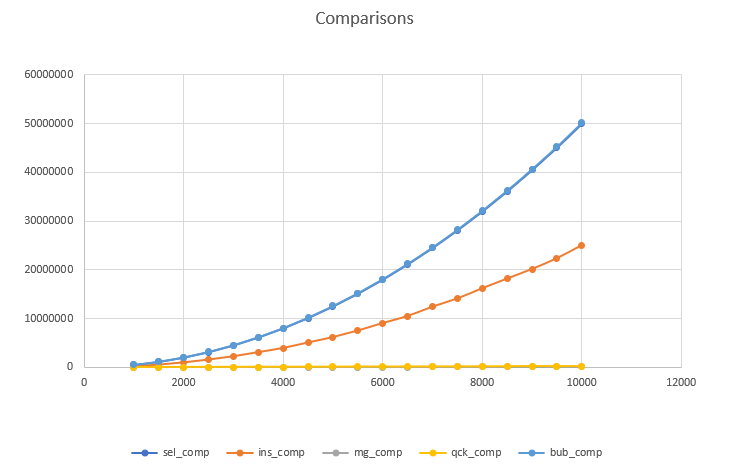
**Data**

****

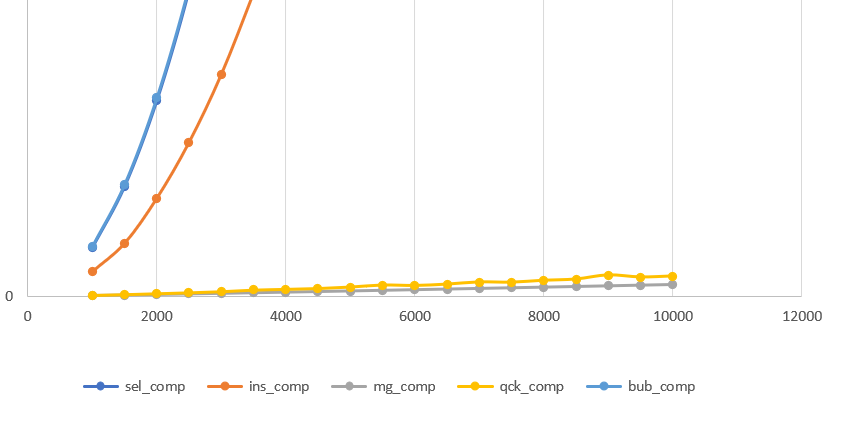
Close-up:



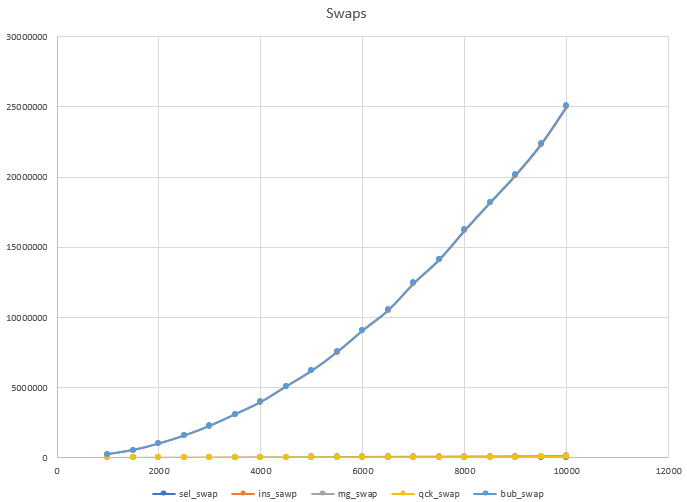
From the chart, we can see that most of our theses were correct in the time complexity, except for insertion sort. Merge sort and quick sort were the best in time-efficiency, only being outperformed by standard sort. Initially, we thought insertion sort would take less time to run than the selection sort because it was doing only one swap after reading through the list of elements each time. However, we made an educated guess about how it also could be doing more reading and writing every time it compares. The result based on this guess is also shown in the number of comparisons and swaps it takes to get the end result, which will be discussed later. The algorithm that surprised me the most is bubble sort with an absurdly high runtime. I had predicted that it would only take O(to run, but it looks more like Oin the results. I think this might have to do with changing the variable swapped within the function because it has to go back and initialize swapped to false every time it moves up an element. I’m not sure if it has a different worst or best case scenario compared to the average case.



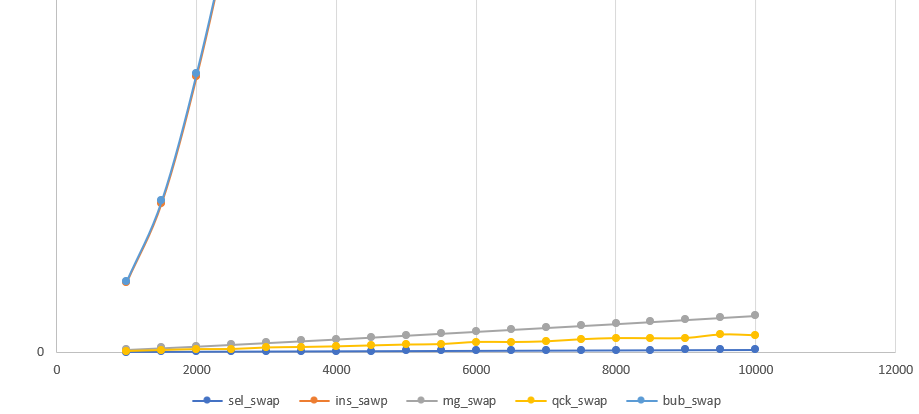
Close-up:



Selection sort and bubble sort take about the same number of comparisons (high), while merge sort and quick sort do not seem to take as much (low).



Close-up:



Here, the results are interesting because insertion sort goes through many swaps. This is probably what makes its runtime so high. The same goes for bubble sort.

**Conclusion**

Bubble sort is the worst to use when one is looking for time efficiency. Instead, they should look for implementation of quick sort and merge sort.