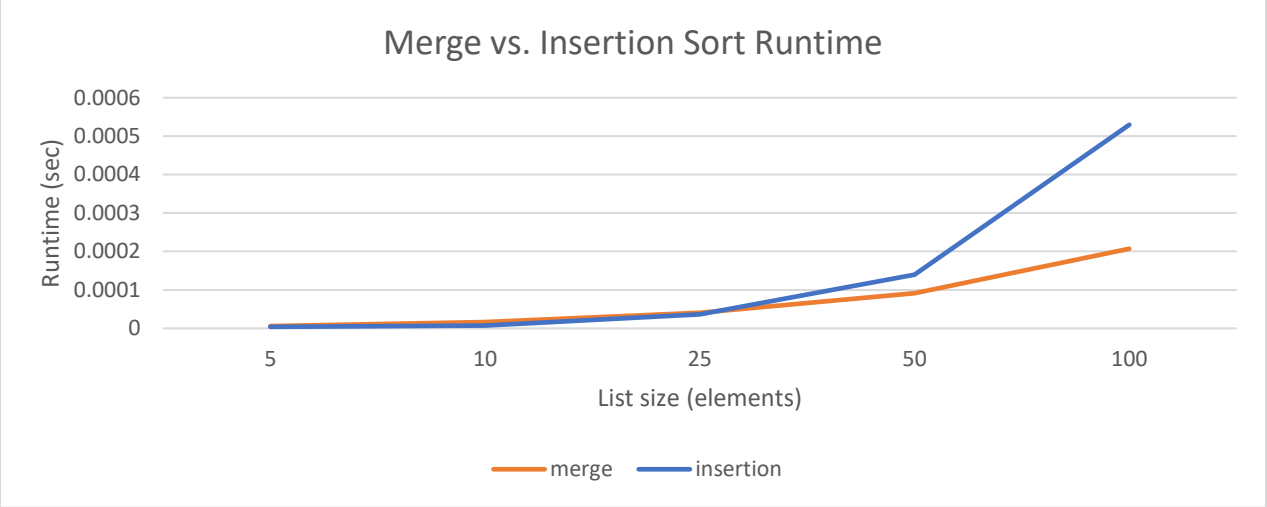
Alex Holt – A57959647

Hypothesis: When comparing merge sort and insertion sort runtimes, I predict that insertion sort will perform better up until n=1000 where merge sort will start to outperform insertion sort.

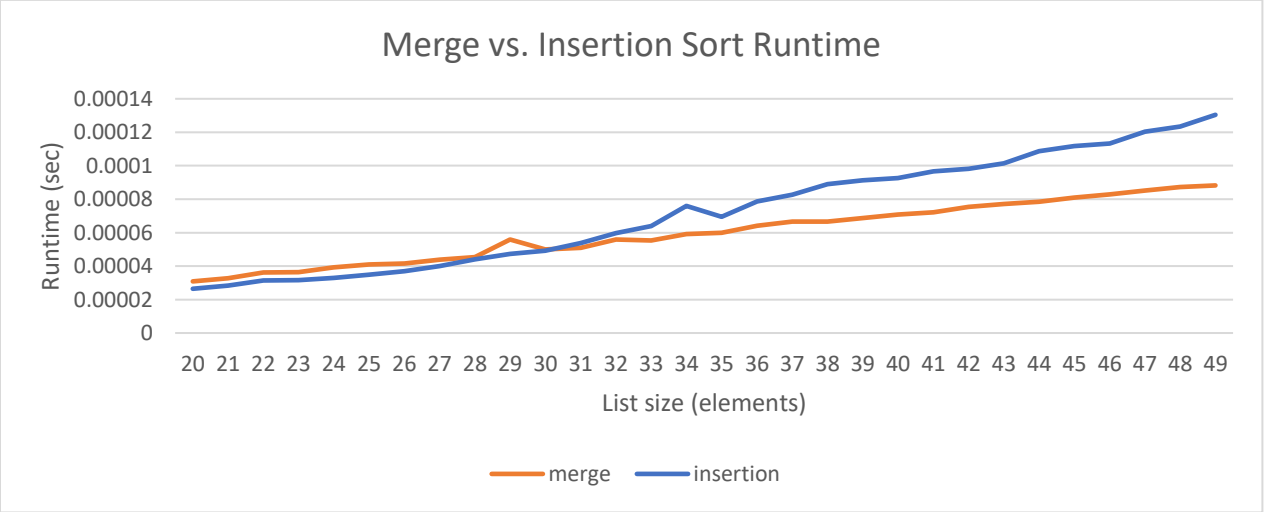
Methods: To solve this problem, I wrote the code at the following link: https://github.com/Holtster2000/cse431_hw4/blob/master/q1.py

I first implemented a merge sort and insertion sort method in python. I then used a random generated list of integers for each sorting function and used the timeit module at varying sizes.

Results: From the output of the program, I plotted the runtimes as a line graph shown below.



As shown, the size that merge sort starts out performing insertion sort is much smaller than I anticipated. From the initial results, somewhere between 25 and 50 elements is the sweet spot for insertion sort. I went back and did a more thorough test to find the exact place where this change happens shown in the graph below.



From the second analysis, we can see that if a list has greater than 30 elements, merge sort is the faster option. These results were observed using 1000 iterations of each sorting function.

Alex Holt – A57959647

Discussion: I was surprised how quickly merge sort outperformed insertion sort. When I hear small in terms of data, I know that can be quite a large value so I assumed a larger list would perform well. I also found out I needed to run a lot more iterations than I thought to get a smooth graph. When I ran my program at 100 iterations for the second graph, the data was very rough and there were multiple points where merge sort and insertion sort outperformed each other.

Another challenge was using timeit effectively. I needed to make sure I wasn't timing the time it takes to regenerate the list of random integers and only measuring the sorting process.

Conclusions: Under the conditions tested, insertion sort performs better for lists of size 30 or less while merge sort is preferable for lists larger than 30 elements.