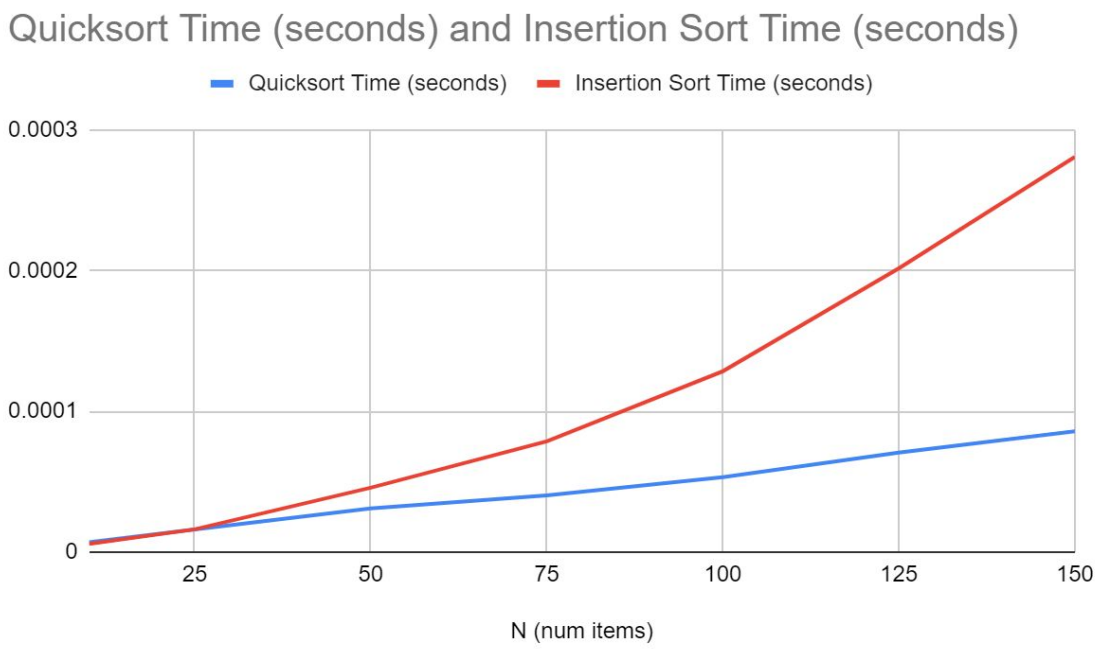
- *Hypothesis*: I believe that quicksort will become faster than insertion sort around N = 10 (10 integers). Anything larger than 25 I would expect quicksort to do much quicker than insertion sort, given that quicksort is O(nlog(n)) and insertion sort is O(n^2)
- *Methods*: My methodology for conducting the experiment is as follows:

  1.) I found c++ implementations of both quicksort and insertion sort, and wrote a main file that tested the clock time before and after it sorted an array populated with random integers with both algorithms.
  2.) Due to extraneous data in testing this on sorting only one array, I made a loop of 1000 test cases that generated random in each iteration, then took the average of the timings in the output. This gave more expected results.
  3.) After compiling my script with g++ and no options/optimization flags, I ran the code and changed the number of items that were being sorted in order to develop a scatter plot in the future. I tested at N = 10, 25, 50, 75, 100, 125, and 150.
  4.) Finally, I input this data into a google spreadsheet for modeling later.

- *Results*:

N (num items)	Quicksort Time (seconds)	Insertion Sort Time (seconds)
10	0.00000699	0.00000583
25	0.00001599	0.00001613
50	0.00003103	0.0000457
75	0.00004031	0.00007863
100	0.00005319	0.00012842
125	0.00007063	0.00020155
150	0.00008579	0.00028073



    - Explanation: I plotted the time it took each algorithm to sort the array vs the number of items in the array. The insertion sort curve is in red, and the quicksort curve is in blue.
- *Discussion*: Before sorting *arrays* of integers, I first tried to sort with *vectors* of integers, in hopes of dynamically allocating the size of the vector for each iteration up to 150 items, to get more data to plot later. However, my code consistently produced results that said that quicksort was *always* faster than insertion sort, no matter how small N was. Since I was aware that this should not be the case, I then tried using arrays (as stated in the methodology above) and ran the code on MSU's Rasperry Pi shell and got much better data, despite not being able to dynamically allocate the array. What I found was that while $N < 25$, insertion sort always performed faster than quicksort, and at $N = 25$, the two seemed to take equal time. With $N > 25$, quicksort always performed faster than insertion sort, however, which was to be expected. However, I did not expect insertion sort to be faster up until $N = 25$. Initially, I thought that my hypothesis of $N = 10$ being the cutoff was actually too high, so finding that insertion was faster until $N = 25$ caught my attention and helped me realize that, in the future, for sorting small data sets, using the easier-to-implement insertion sort is probably the best bet. However, for generality and taking into account an ever-growing data set size, quicksort is certainly a better-performing option.

- *Conclusions*: For $N < 25$, insertion sort is faster than quicksort for sorting integers. At $N = 25$ is where both algorithms take roughly the same amount of time. For $N > 25$, quicksort is always faster, and gets exponentially faster compared to the timing of insertion sort.