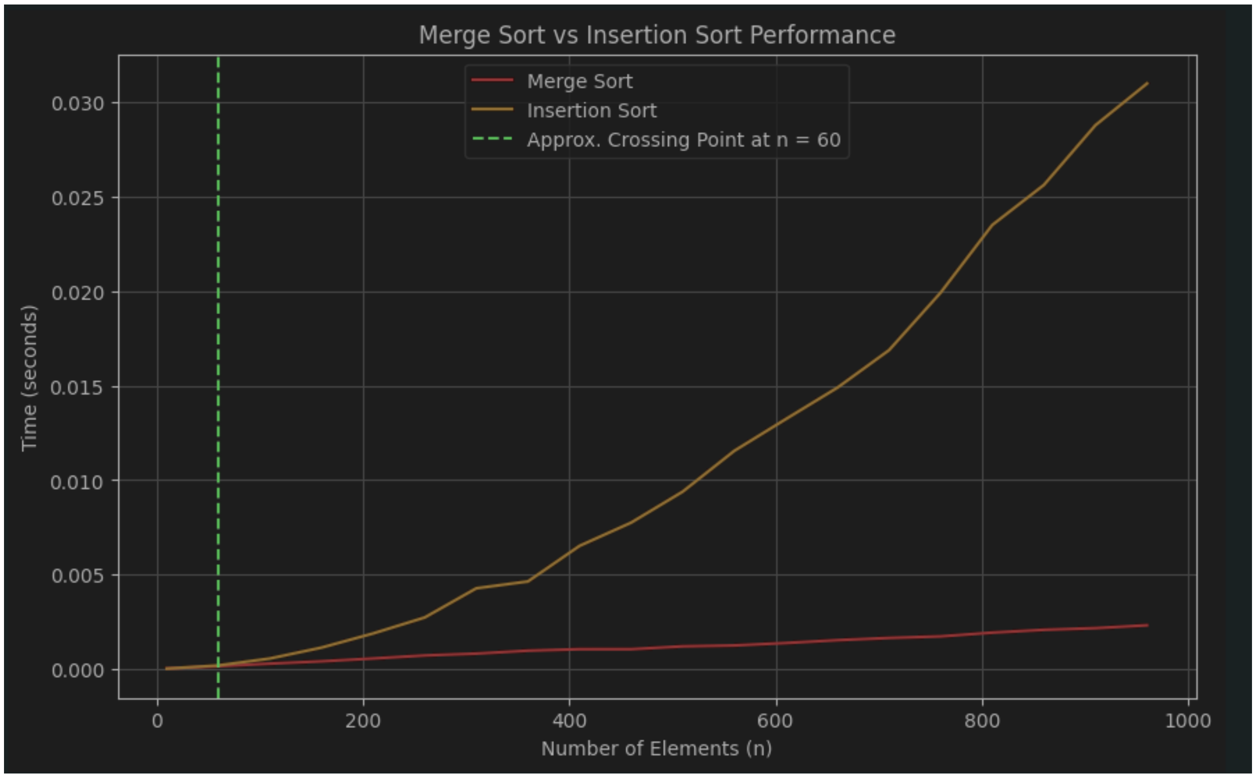
Algorithm Analysis: Merge Sort vs Insertion Sort

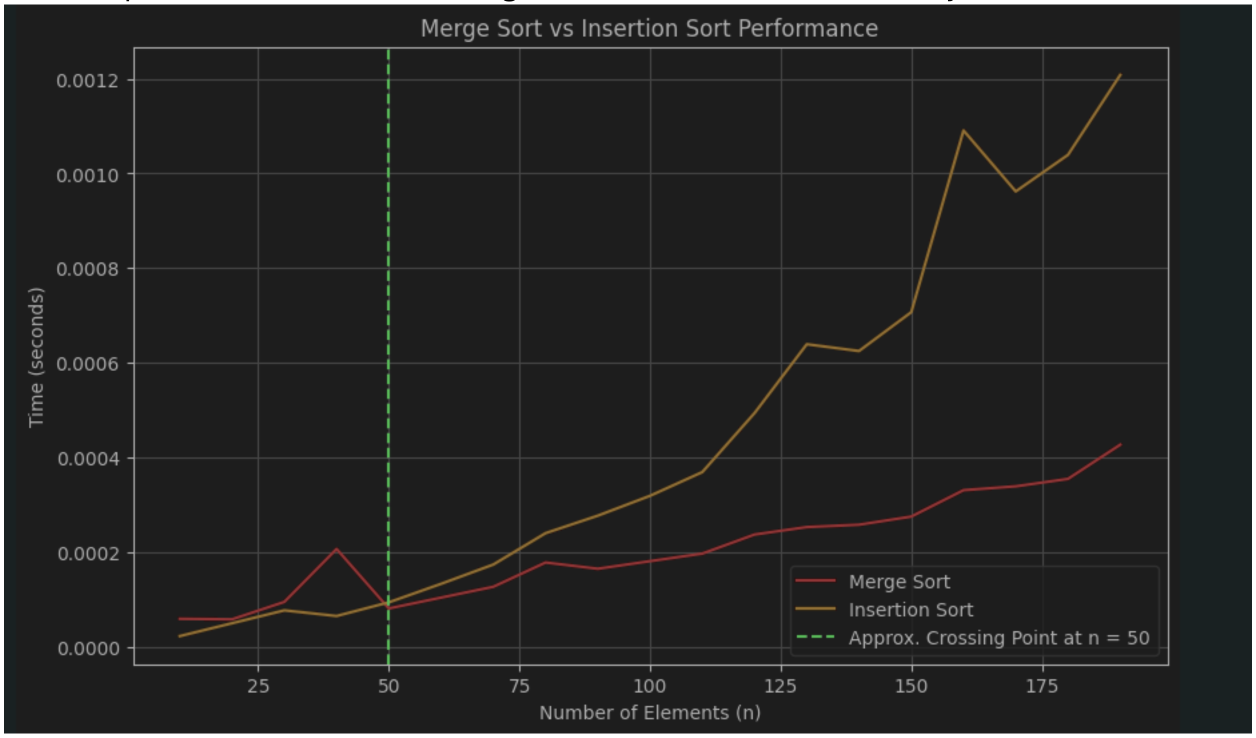
Hypothesis: Knowing that merge sort is faster for a large n while insertion sort works better for a small n, I hypothesize that insertion sort will perform better than merge sort for input size of n less than or equal to 100. I believe the graphs will reach a distinction at this n value where it will be easy to see the exponential run time of insertion sort and the logarithmic run time of merge sort, with some value less than 100 being where the graphs intersect.

Methods: I conducted this analysis using a Python 3 Jupyter Notebook making use of Numpy, Time, and Matplotlib to generate data, time the algorithms, and view the result respectively. GitHub Copilot generated the implementations of Merge Sort and Insertion Sort and I verified to make sure they are correct. Two separate tests were conducted to prove the point that insertion sort is faster for small n but slower for large n compared to merge sort. One test goes from 0 to 1000 (exclusive) to show the big picture, and one goes from 0 to 200 (exclusive) to show the intersection and speed differences at small inputs. The code uses the .copy() function on the generated data to ensure each algorithm used the unsorted list and randomizes the order of testing using a threshold of 0.5 and a random number generated each iteration, deterring if insertion sort or merge sort would be the first algorithm to sort the data. For both tests, a for loop loops over the number of values, generates data for each value, then times both algorithms. Graphs were then generated showing the crossing point of the algorithms with the time and input size on the y and x axis. Seeds were set for both to ensure reproducibility and the link to the code is here: https://github.com/edemott/Merge-Sort-vs-Insertion-Sort-Time-Complexity

Results:



This was the initial test I conducted and the respective output. This graph highlights the big picture that insertion sort has an expected run time of n^2 and that merge sort has an expected run time of nlog(n). The approximate crossing point for this graph is at n=60 (approximate due to step size of input sizes). The code used to create this graph calculates the time taken for different inputs of step size 50 up to 1000 for merge sort and insertion sort and plots the time it takes the algorithm to sort the unsorted array.



This was the result of the second test I conducted and was conducted for the purpose of showing that insertion sort performs better at smaller input size n. For the seed I set, the intersection of the algorithms is at n = 50 and after that merge sort performs better than insertion sort. This graph goes from 0 to 200 (exclusive) taking a step size of 10.

Discussion: Surprisingly, my hypothesis of insertion sort performing better for values under n = 100 was not to far off. In my results, it was shown that graph intersected at n = 50, which aligned with my guess of the graph intersecting at some value less than 100. For the data that was randomly generated, insertion sort took less time than merge sort for input values of n less than 50.

Conclusions: Under the conditions tested, Insertion sort is the faster sorting algorithm for n < 50, while for n > 50 Merge sort is the faster sorting algorithm.