Hypothesis:

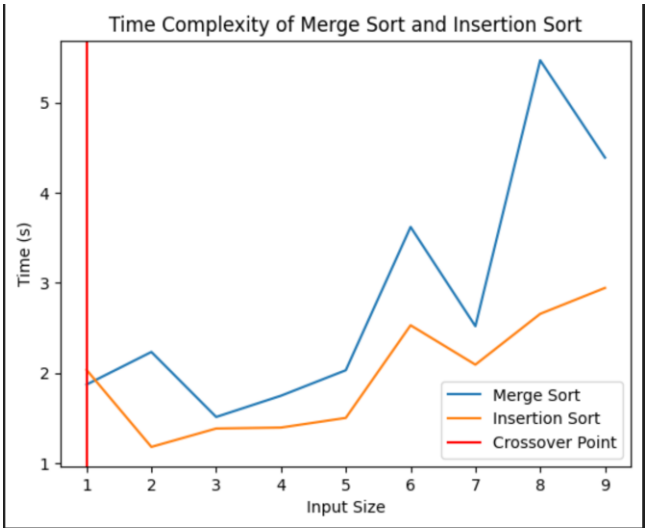
I believe that when comparing Merge sort and Insertion sort insertion sort will only be faster when n is as small as possible so about 1-3

Methods:

Source Code: https://github.com/leohoerdemann/HW5/blob/main/Q1.ipynb

I created the functions for both merger sort and insertion sort with python 3.11.3 in a jupyter notebook as well as a function to find the point when the crossover of times would occur. After that I put the functions inside other functions that would generate n random numbers in a list and run the sorting algorithms on those lists. This was done to make sure the list was new each time. After this I generated a list of the different n amounts to use and ran both sorting algorithms on those amounts using timeit to get the run time of 100000 times and putting those in an output list for each algorithm. Finally, I graphed the outputs.

Results:



Times of each sort done 100000 times vs the input size.

Discussion:

The output graphs consistently showed that the time of insertion sort is only faster on an array of size 1. However, there were a few times when the insertion sort was faster for sizes up to 9. While sizes up to 3 had insertion sort faster a fair but of times it still was not as consistent as 1. The rest though were outliers, and I was not able to reproduce these consistently at all.

Conclusions:

Under the testing conditions I found that my original hypothesis was supported. Insertion sort is faster on arrays of sizes around 1-3 the closer to one being the more consistently faster. On the other hand insertion sort was slower than merge sort on everything else.