Hypothesis:

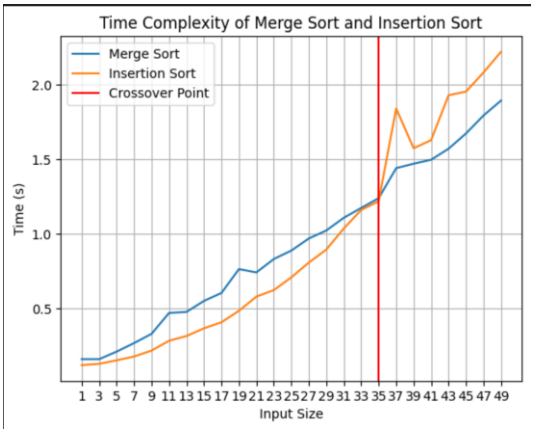
I believe that when comparing Merge sort and Insertion sort insertion sort will only be faster when n is as small as possible so about 1-3

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 10000 times and putting those in an output list for each algorithm. Finally, I graphed the outputs.

Results:



Times of each sort done 10000 times vs the input size (lower is better)

Discussion:

The output graphs consistently showed that the time of insertion sort is faster than merge sort on arrays of around size 35 and under. While there were sometimes where that number varied, it normally sat around 30-37 and most consistently sat at 35. Using larger ranges for n showed the same thing where merge sort would become faster at sizes around 35.

Conclusions:

Under the testing conditions I found that my original hypothesis was not supported. Insertion sort is faster on arrays of sizes around 35 and lower. On the other hand insertion sort was slower than merge sort on everything above 35.