Hypothesis:

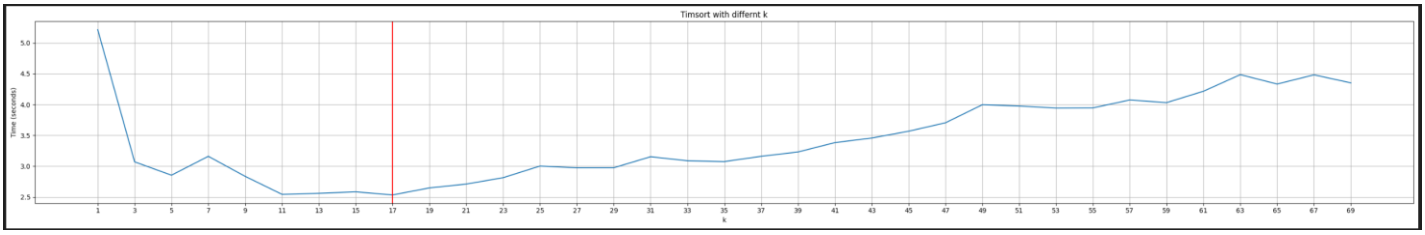
I believe that when looking at the best K values a value similar to what was found in question 1 will work best that value being about 35.

Methods:

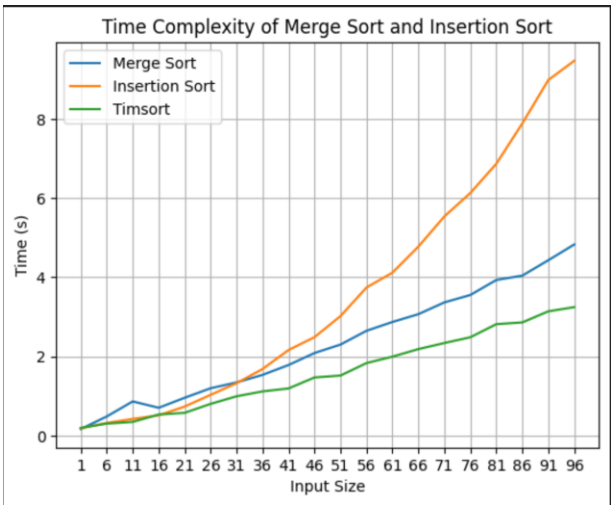
Source Code: https://github.com/leohoerdemann/HW5/blob/main/Q2.ipynb

First, I created the function for Tim sort using a jupyter notebook on python 3.11.3. This function would take in an array to sort as well the K value to use. I then ran it though a loop testing various k values each 1000 times and with arrays from size 1 to 100. I used timeit on this loop to get the time taken with each K value. Finally, I graphed the output using matplotlib and an extra function to add find the K value with the lowest time. I also carried over the graphing code from the previous question. However, this time I also graphed Tim sort and removed the crossover point calculations.

Results:



Times of Tim sort done with different K values (the lower the better).



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

Discussion:

The K value I found to be best (that being 17) was about half of the value that was found in the previous question (35). I thought this was interesting as my original thought was that it should be about the same. I think the recursive nature of being put into merge sort could affect the time complexity of insertion sort making the ideal K value half it would be otherwise. I also saw in the second graph that for almost every size of the array Tim sort was faster than both insertion sort and merge sort.

Conclusions:

Under the testing conditions I found that my original hypothesis was not supported. The ideal K value for Tim sort sits around half of what the recommended size from the first question gave.