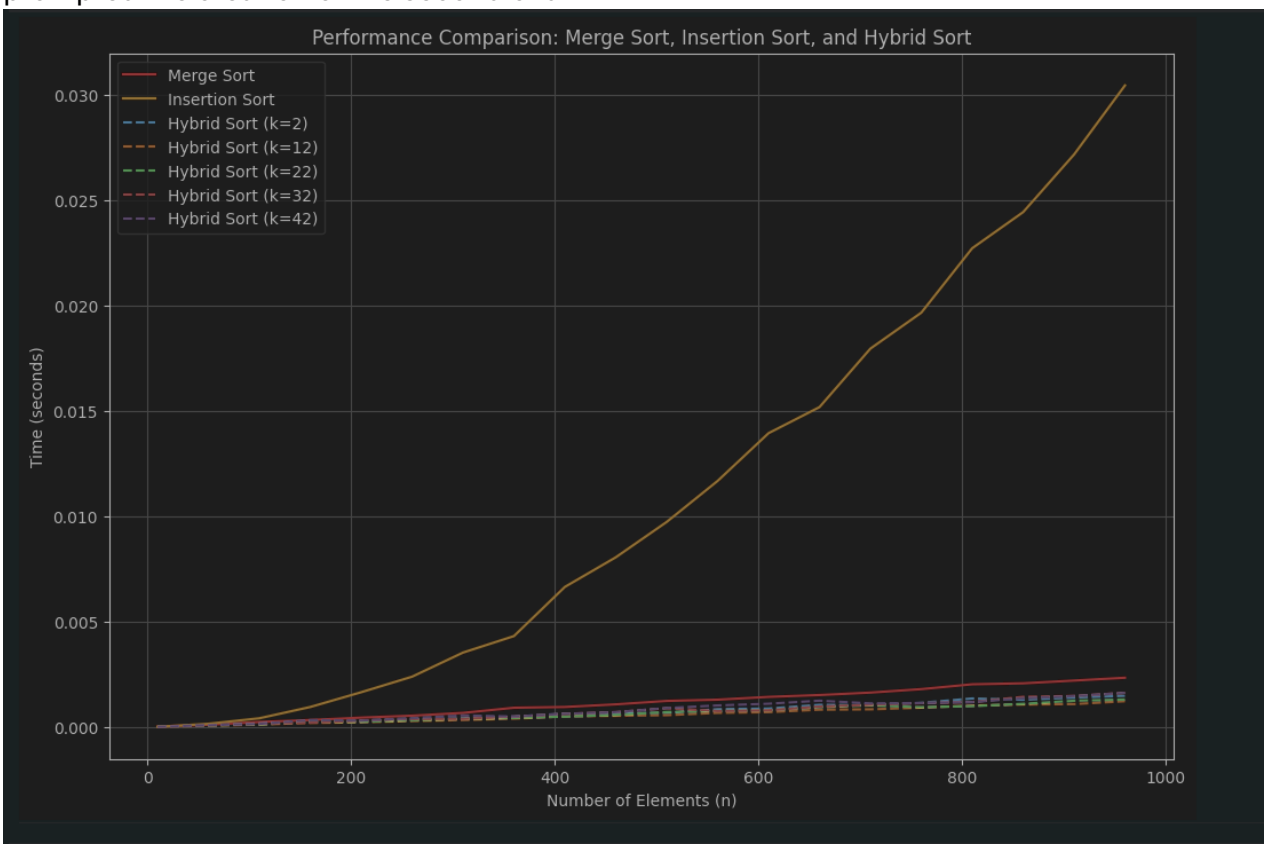
Algorithm Analysis: Hybrid Merge Sort

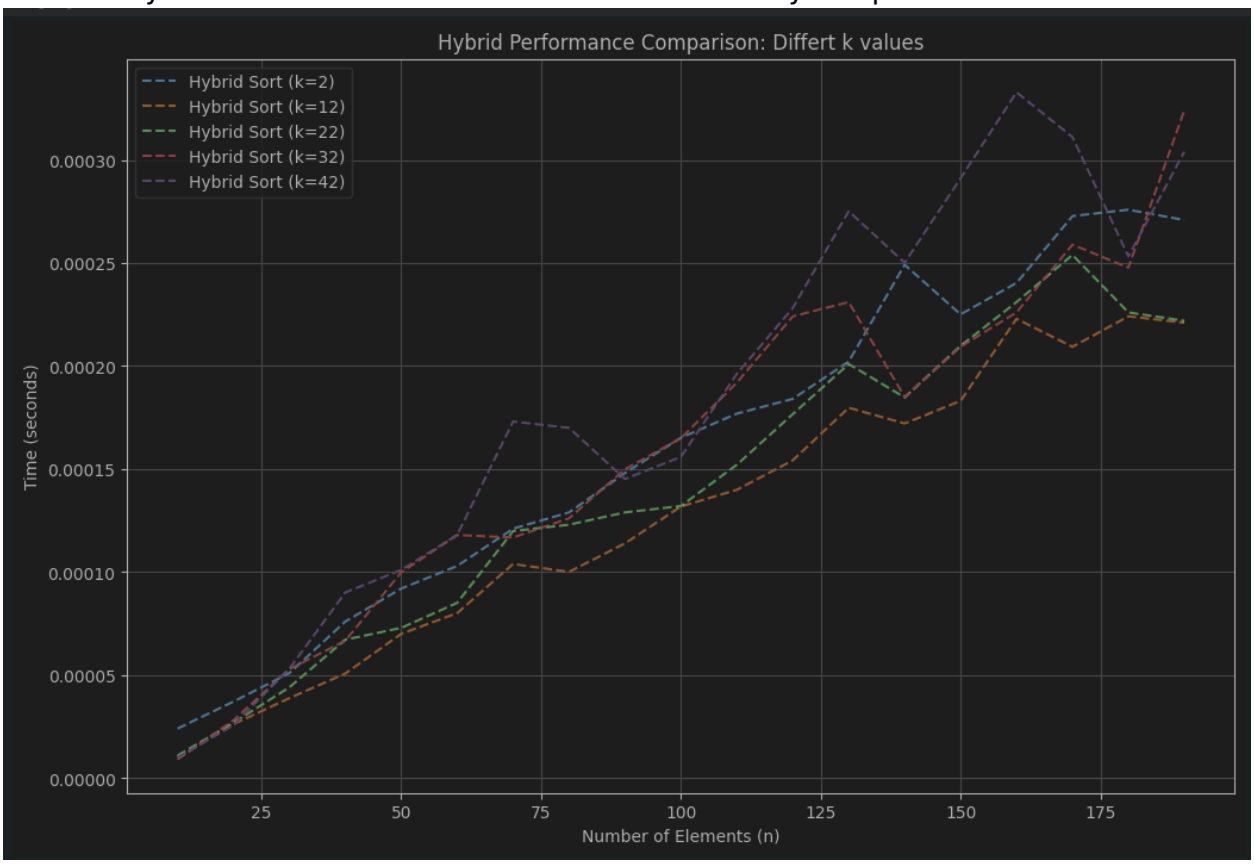
Hypothesis: I predict that for a partition size of k around 50 the hybrid merge sort will be optimized.

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. I followed a similar setup, using code from the previous question for graphing and timing the algorithms. I generated two plots, one showing merge sort, insertion sort, and hybrid merge sort (Tim sort) for different k partition values. For both tests, a for loop loops over the number of values, generates data for each value, then times both algorithms. For the Tim sort algorithm, I followed the approach by performing insertion sort if the length of the array was less k and performing merge sort for arrays greater than the partition. The results were graphed using a for loop to randomly generate the data over a set amount of k values (2, 12, 22, 32, and 42) with the first graph going up to n=1000 and the second, more zoomed in result going up to 200 (exclusive for both). Chat GPT was used for generating both plots for a clean and interpretable graph. 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:
Shown below is the first result of the analysis up to n=1000 (exclusive) with a step size of 10. This result shows the big picture, comparing all three sorting algorithms. The result is as expected, with insertion sort performing at the worst time, then merge sort, then the hybrid merge with the best time for varying k levels. While this graph is great at highlighting the main theme, it does not show well how differing levels of the partition perform, which prompted the creation of the second chart.



Shown in the second chart is a performance comparision of different k values for the hybrid merge sort algorithm up to n values of 200 (exclusive) with a step size of 10. The closest the different partitions come to intersecting one another (all are the same speed) is at n less than 25, around 15. This differs from my previous analysis as the crossing point of insertion sort and merge sort was a n=60. From this graph, it is shonw that a partion value of k=12 is the optimal choice for the hybrid merge sort. Over the number of elements shown, it consistenly beats the alternate choices and sorts the array the quickest.



Discussion: My initial hypothesis of a partition size of k around 50 the hybrid merge sort will be optimized was correct. From my experiment, I observed that a partition value of k=12 is the optimal value from my list of 5 different partition values. While my guess was too high, I was on with the idea that a small partition would outperform a large one. The reason I hypothesized this is because insertion sort outperforms merge sort for arrays of small size, so using hybrid merge sort with a small partition to use insertion sort would perform superior to using hybrid merge sort with a high partition value.

Conclusions: Under the conditions tested, the optimal partition value for Hybrid Merge Sort (Tim Sort) is k=12. Overall, Tim Sort outperformed both Merge Sort and Insertion Sort in terms of time complexity with its ability to utilize the best conditions of both Merge Sort and Insertion Sort.