***Insertion Sort vs Quick Sort***

Insertion Sort and Quick Sort are two of the most used sorting algorithms. Insertion sort has an expected run time complexity of ), while Quick Sort has an expected time complexity of ). At first glance, it may seem that Quick Sort will always be the faster sorting algorithm, but in practice, the results may differ based on the size of the input.

**Hypothesis**

Given the theoretical time complexities of insertion sort and quick sort, it seems that quick sort will outperform insertion sort. However, in practice, the performance these sorting algorithms may vary based on the size of *n.*

Insertion sort will likely outperform quicksort for lower values of n due to less overhead in terms of code. Insertion sort only needs to compare values and make swaps, while Quick Sort must compare and swap values, do recursive calls, and use a partitioning function. As a result, Insertion Sort will have lower time and space complexity than Quick Sort for small values of *n*.

On the other hand, Quick Sort will likely outperform Insertion Sort for large data sets due to its faster time complexity. For large values of *n*, the initial overhead to implement Quick Sort will be negligible compared to the performance of ) over Insertion Sort’s ). This is assuming that a good partitioning scheme is chosen and the data is not manipulated to abuse the partitioning scheme.

**Methods**

***Full code can be found at*** [***this GitHub repo***](https://github.com/sidb70/Algorithm-Engineering-Course-Project/blob/main/Insertion%20Sort%20vs%20Quick%20Sort/main.py)

*Design:* For this study, we compared run times to execute insertion sort and quick sort on arrays of varying sizes of *n*, ranging from 5 to 200. The array data was populated using random number generation with a constant seeding value to ensure results were unbiased.

*Data collection*:

*Sources and Tools*:

<https://seaborn.pydata.org/generated/seaborn.swarmplot.html>