1. Methodology

i. Algorithms and Sample Sizes

- Bubble sort, selection sort, insertion sort, quick sort, and heap sort
- Six sample sizes: 1000, 5000, 10000, 15000, 20000, 50000

ii. Random Number Generation

- 25 random numbers will be generated for each sample size
- Random seed to ensure the same random numbers are used for each sample size
- Range of random numbers: -1000 ~ 1000

iii. Time Calculation

- The average time will be calculated for each sorting algorithm for each sample size
- The time will be measured in seconds using Python's time module

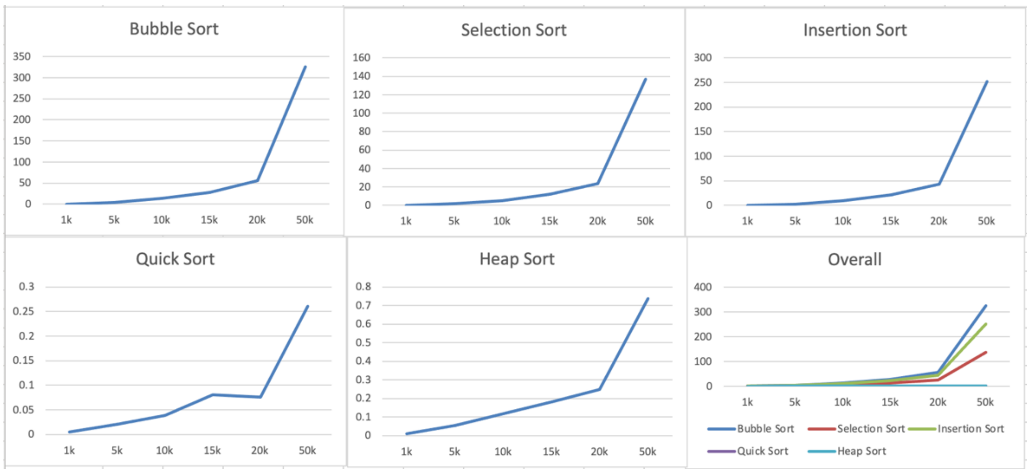
iv. Programming Language and Libraries

- Python (in Spyder)

v. Results Presentation and Export

- Dataframe with rows representing the sorting algorithms and columns representing the sample sizes
- The dataframe will be exported to an Excel file for visualization

2. Graphical Results



3. Discussion

Based on the data presented, we can observe that quick sort and heap sort have the lowest average time among the five sorting algorithms for all sample sizes. Bubble sort consistently has the highest average time, with a significant increase in time as the sample size increases. Selection sort and insertion sort have intermediate average times between bubble sort and quick sort/heap sort.

Additionally, we can see that as the sample size increases, the average time for all sorting algorithms also increases. This is expected, as the time complexity of most sorting algorithms is O(n^2) or O(n log n), meaning that as the input size (sample size) increases, the time taken to execute the algorithm increases exponentially.

Overall, these results can be used to make informed decisions about which sorting algorithm to use based on the size of the input data. For small sample sizes, all algorithms can be used interchangeably, but for larger sample sizes, quick sort and heap sort may be more appropriate due to their faster average times.

4. Challenges and Solutions

One of my experiences was that I didn't notice it was acceptable to run different sample sizes based on my computer's CPU, despite the homework introduction mentioning we needed to run through sample sizes of N=50000, 100000, 150000, 200000, 250000, and 300000. I felt stressed about the large sample size, as my computer struggled to run the bubble sort with 25 series of 50000. It took me a whole day to run through all the sorting algorithms with just N=50000, let alone the larger sample sizes. After two days of setbacks, I revisited the homework introduction and discovered that we could adjust the sample size based on our personal computer CPU capabilities, which relieved my stress.

Through implementing the various sorting algorithms, I gained a deeper understanding of why calculating time complexity is essential. In today's digital world, there is an abundance of data generated in every corner, and N=50000 is a relatively small sample size compared to the hundreds of thousands of data that exist. Initially, the concept of Big O notation was challenging to comprehend. I struggled to understand what computer efficiency meant, how long O(n^2) would take, whether O(n!) was truly a lengthy process, and how fast O(nlogn) would be. As someone who only interacted with computers to obtain answers or solutions to my queries, I had never considered the underlying algorithms or how they worked.

As an international business student without a background in information management, this homework assignment was novel for me. Most of my projects revolved around developing business strategies, which were abstract concepts to help companies or deal with people in terms of business management. These aspects of business are less related to computer science.