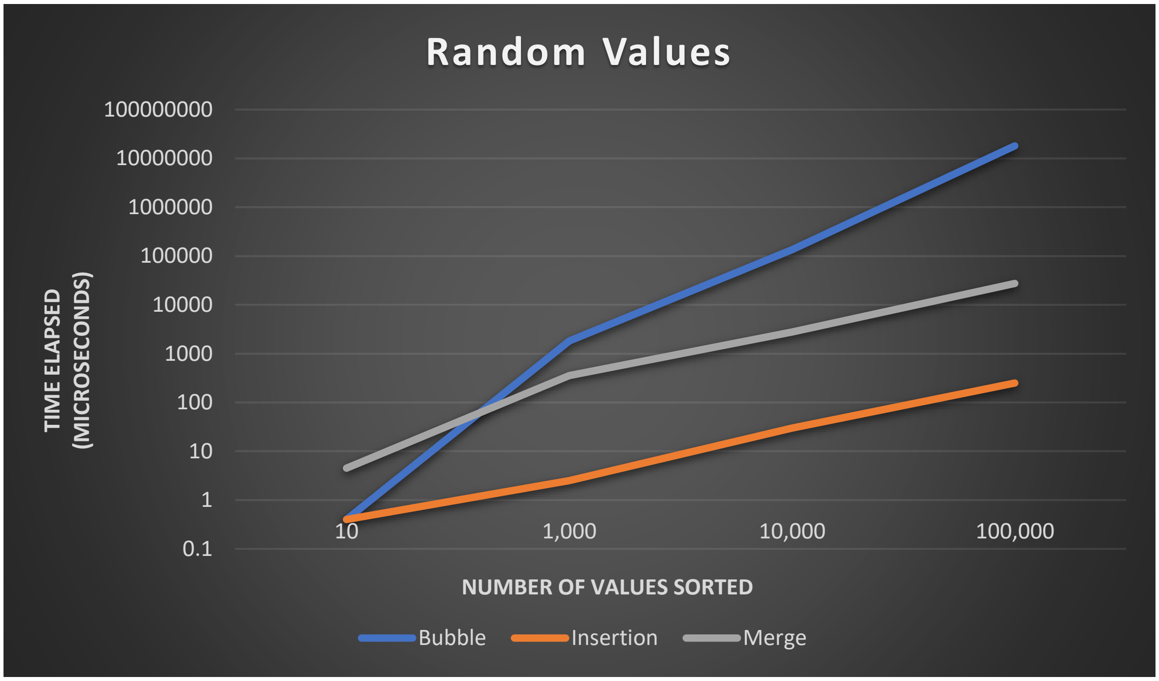
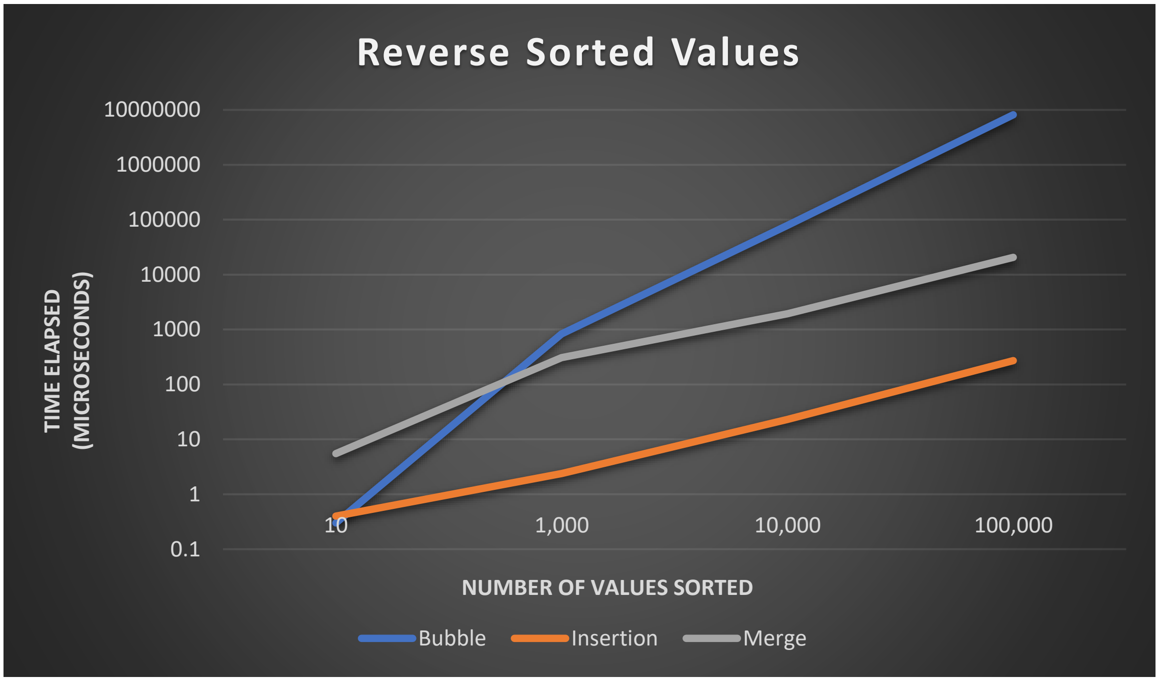
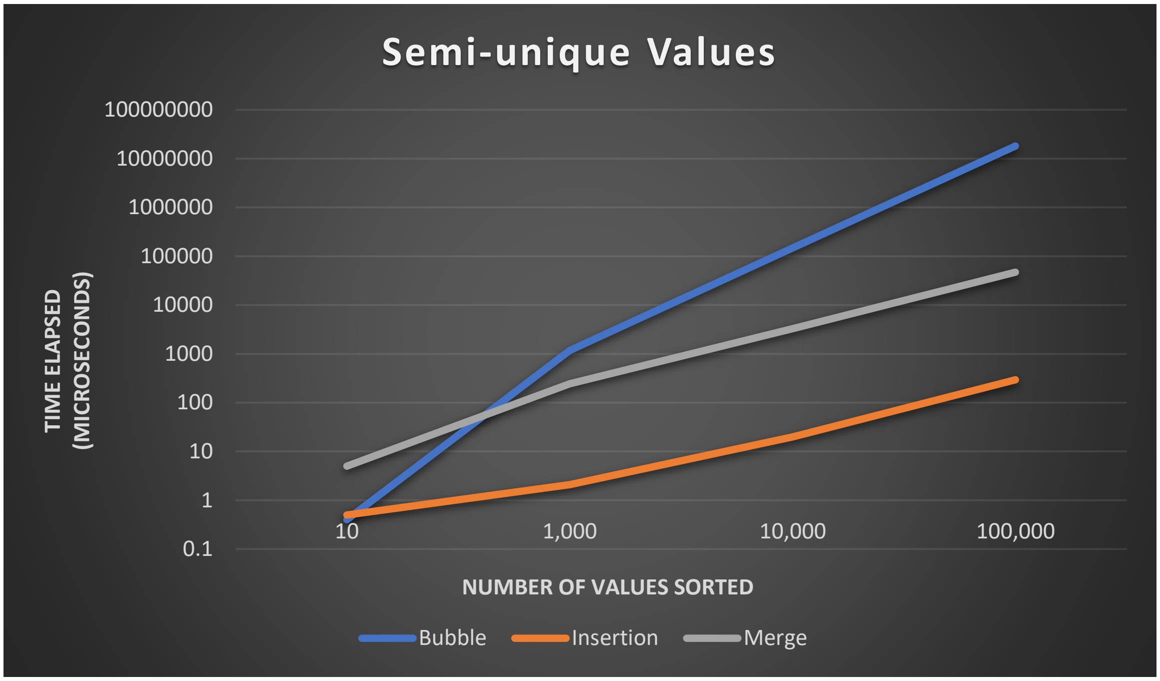
Nicholas Crothers

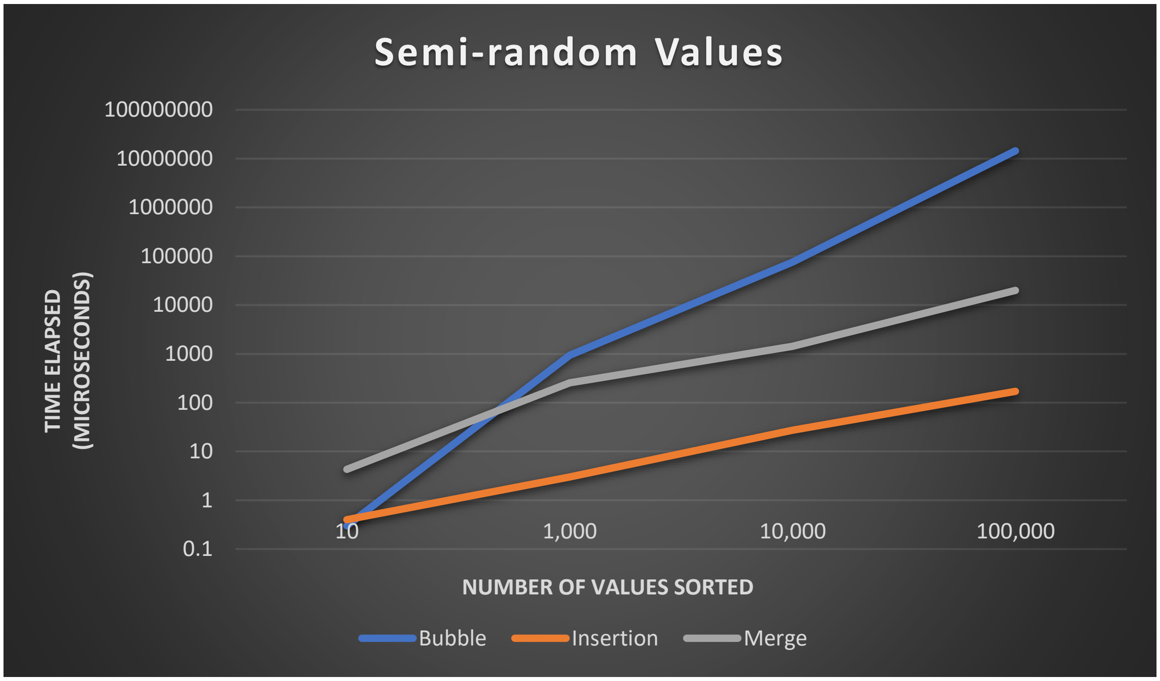
# CS 3353 Lab 1 Report

## Graph Data Representation:






Semi-unique Values
TIME ELAPSED (MICROSECONDS)
100000000
10000000
1000000
100000
10000
1000
100
10
1
0.1
10
1,000
10,000
100,000
NUMBER OF VALUES SORTED
Bubble
Insertion
Merge


Semi-random Values
TIME ELAPSED (MICROSECONDS)
100000000
10000000
1000000
100000
10000
1000
100
10
1
0.1
10
1,000
10,000
100,000
NUMBER OF VALUES SORTED
Bubble
Insertion
Merge

Data Summary Tables:

| 10 Values $(\mu\ seconds)$ | | Sort Algorithm | | |
|---|---|---|---|---|
| | | Bubble | Insertion | Merge |
| Data Types | Random | 0.4 | 0.4 | 4.5 |
| | Reversed | 0.3 | 0.4 | 5.5 |
| | Semi-unique | 0.4 | 0.5 | 5 |
| | Semi-random | 0.3 | 0.4 | 4.3 |

| 1,000 Values $(\mu\ seconds)$ | | Sort Algorithm | | |
|---|---|---|---|---|
| | | Bubble | Insertion | Merge |
| Data Types | Random | 1803.2 | 2.5 | 357.9 |
| | Reversed | 833.7 | 2.4 | 308.5 |
| | Semi-unique | 1176.8 | 2.1 | 244.8 |
| | Semi-random | 930.7 | 3 | 254.8 |

| 10,000 Values $(\mu\ seconds)$ | | Sort Algorithm | | |
|---|---|---|---|---|
| | | Bubble | Insertion | Merge |
| Data Types | Random | 133922.6 | 29.9 | 2754.2 |
| | Reversed | 78459 | 23.2 | 1931 |
| | Semi-unique | 146331.5 | 19.8 | 3280.9 |
| | Semi-random | 75682.2 | 27.2 | 1417.6 |

| 100,000 Values $(\mu\ seconds)$ | | Sort Algorithm | | |
|---|---|---|---|---|
| | | Bubble | Insertion | Merge |
| Data Types | Random | 18071379.2 | 249.6 | 27427.1 |
| | Reversed | 8150815 | 271.5 | 20621.2 |
| | Semi-unique | 18102849.5 | 293.6 | 47022.1 |
| | Semi-random | 14441395.3 | 170.3 | 19930 |

## Data Analysis:

As expected, the bubble sort algorithm performed the worst on the datasets larger than ten elements and became less and less efficient as the number of values increased. This to be expected given the algorithm's worst-case time complexity of $O(n^2)$, where the time complexity increases exponentially as the number of elements increases. With a very low number of values, like ten, it makes very few comparisons, so this executes rather quickly.

I had predicted that bubble sort would perform the worst with the reversed dataset due to the sheer number of swaps that needed to occur, but it performed the best in almost every dataset size. It performed similarly in both the random and semi-unique datasets.

Surprisingly, the insertion sort algorithm performed the best overall. The algorithm sorted the datasets virtually instantly, even the dataset with 100,000 elements, which took the algorithm between 200 and 300 microseconds to sort. Within each set of total data elements, such as the 10,000 elements group, the algorithm performed best on a specific data type; in this case, it was the semi-unique dataset. However, this does not hold true for other datasets with a different number of elements.

The merge sort algorithm performance was surprising. It performed much worse than insertion sort, especially on the larger datasets. However, even with 10 elements it performed much worse than both bubble and insertion. I believe this is because of the way the algorithm is implemented, with the merge() function copying the left and right side into new arrays every time it is called, causing a large overhead which is only magnified with larger datasets.