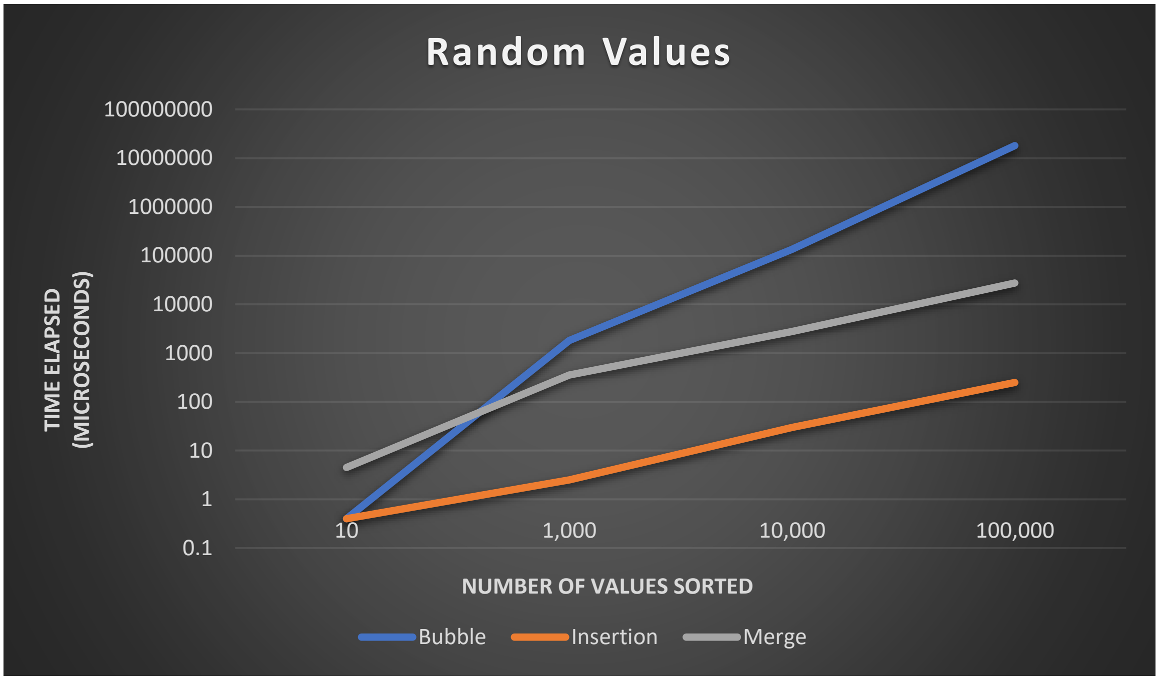
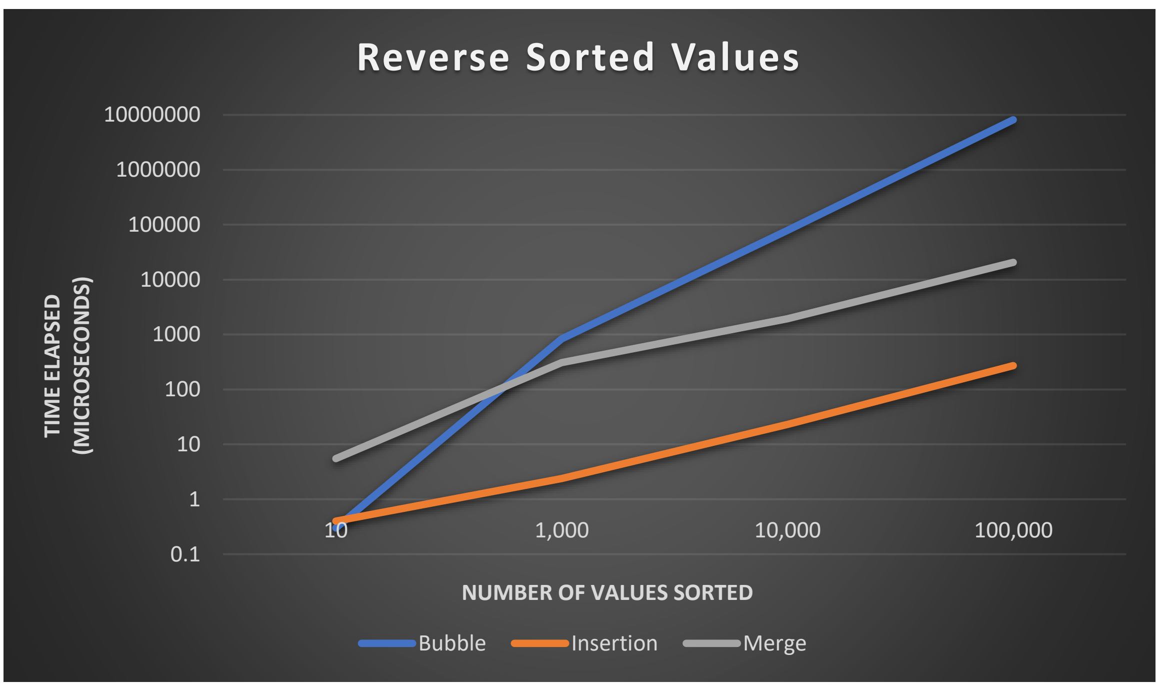
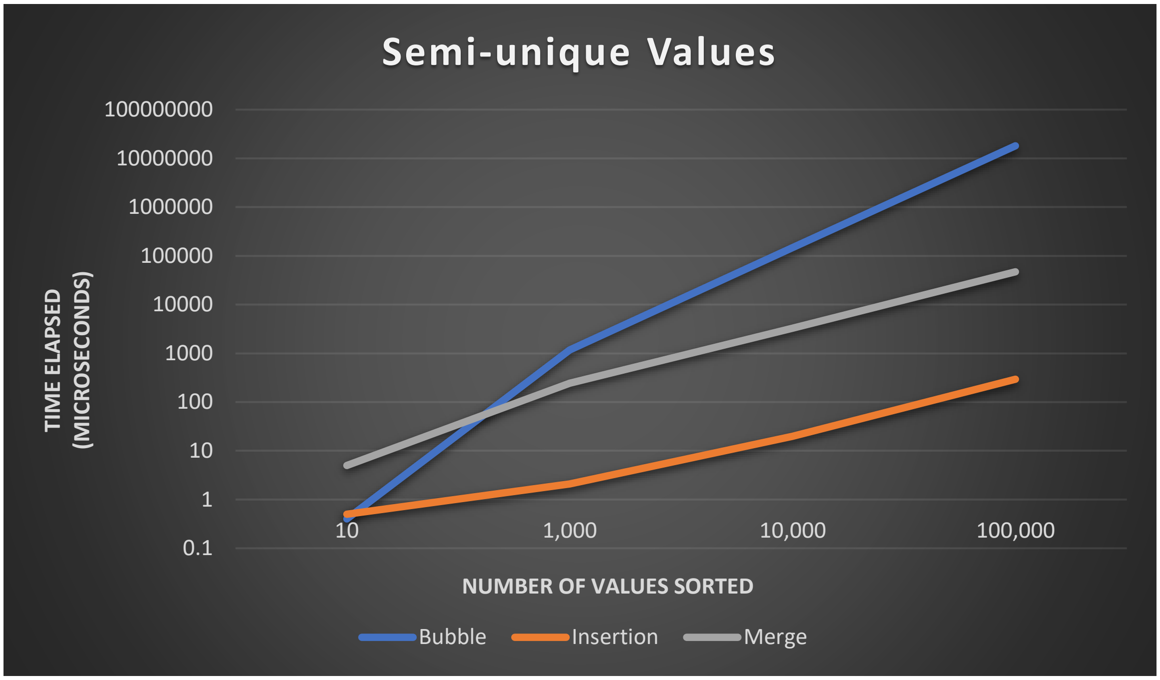
Nicholas Crothers

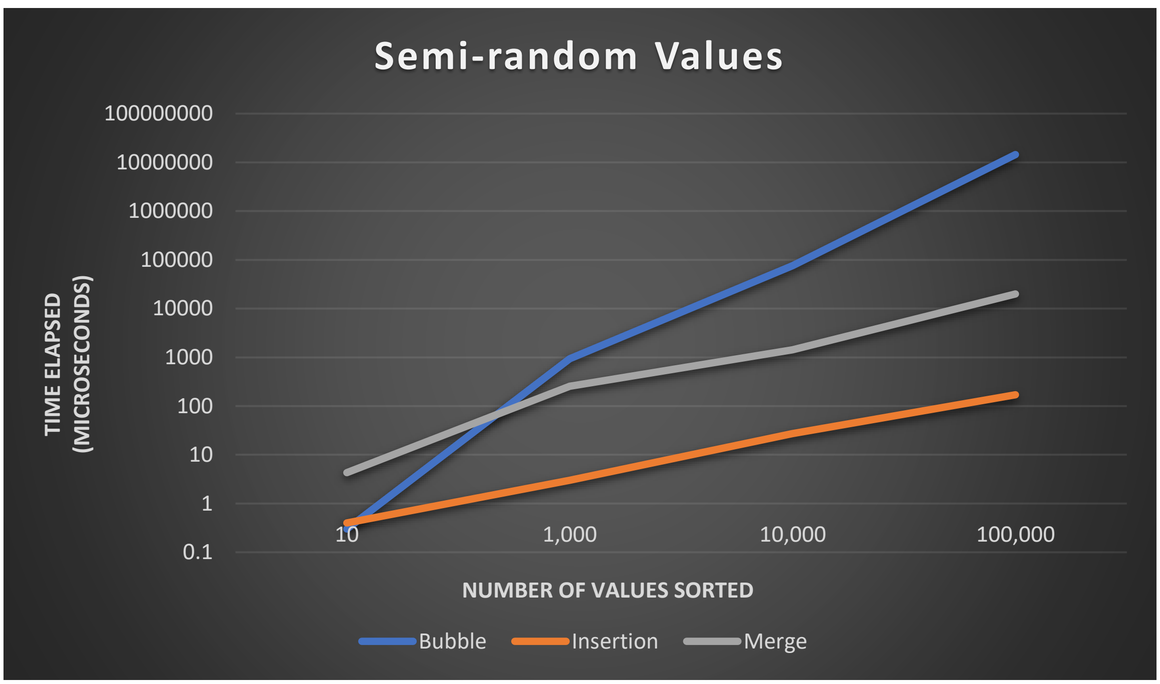
CS 3353 Lab 1 Report

Graph Data Representation:






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


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

Data Summary Tables:

10 Values in milliseconds		Sort Algorithm		
		Bubble	Insertion	Merge
Data Types	Random	0.0052	0.0046	0.0113
	Reversed	0.0071	0.005	0.0226
	Semi-unique	0.0039	0.0018	0.0141
	Semi-random	0.005	0.0025	0.0135

1,000 Values in milliseconds		Sort Algorithm		
		Bubble	Insertion	Merge
Data Types	Random	56.3777	15.1305	1.16800
	Reversed	85.2403	38.2647	1.11950
	Semi-unique	42.2023	8.84440	1.50380
	Semi-random	52.2609	17.4374	2.38360

10,000 Values in milliseconds		Sort Algorithm		
		Bubble	Insertion	Merge
Data Types	Random	5704.08	1685.56	13.1897
	Reversed	7635.61	3275.57	13.4480
	Semi-unique	4188.88	839.05	15.6017
	Semi-random	5268.94	1643.85	14.6798

100,000 Values in milliseconds		Sort Algorithm		
		Bubble	Insertion	Merge
Data Types	Random	522072	159083	147.463
	Reversed	718720	319301	144.301
	Semi-unique	414997	81817	195.561
	Semi-random	528332	160274	148.239

Data Analysis:

As expected, the bubble sort algorithm performed the worst on the datasets larger than ten elements and became exponentially more inefficiency 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.

Bubble sort had the worst performance on the reversed dataset, as I had expected, due to the sheer number of swaps needed to completely reverse the array. It performed the best on the semi-unique dataset, which makes sense given that there are 5 "correct" spots for each value since there are 5 of each value. Random and semi-random performed similarly.

Insertion sort performed the next worst, a lot better than bubble but still very poor performance on the 100,000 value dataset. As with bubble sort, the insertion sort algorithm performed worst on the reversed dataset due to the amount of shifting that needed to happen. It performed the best on the semi-unique dataset, likely for a similar reason in the bubble sort. Random and semi-random performed similarly.

Merge sort performed the best, as expected, given its time complexity is $O(n \log(n))$. Because it is a divide and conquer algorithm, it performs virtually the same on all the datasets since it breaks it up into smaller and smaller pieces and performs the same number of array element transfers. It performed in a fraction of a second on all the dataset sizes.