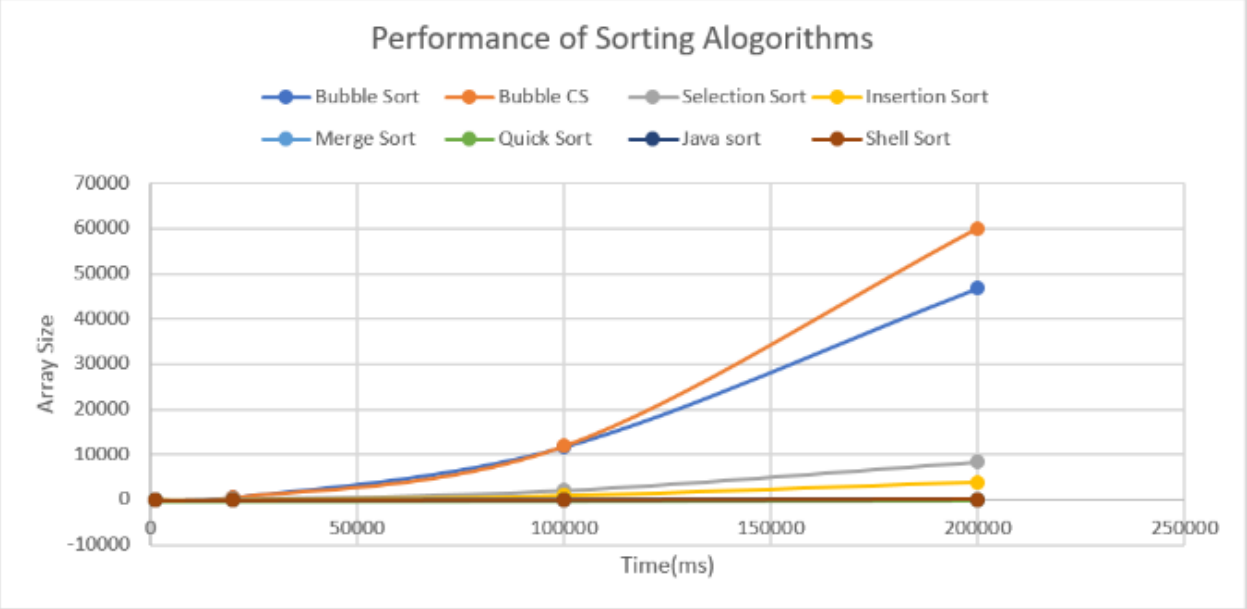
Name: Brennen Calato
LSU ID: 895007806
Date: 4/23/2020

		Time Milliseconds		
Array Size	**1000**	**20,000**	**100,000**	**200,000**
Bubble Sort	5	502	11,730	46,855
Bubble CS	4	517	11,900	60,035
Selection Sort	2	157	2116	8426
Insertion Sort	2	37	978	3956
Merge Sort	0	4	17	33
Quick Sort	1	2	7	15
Java Sort	0	3	8	18
Shell Sort	0	4	12	26



a) What is the complexity of the sorting algorithm you chose?

The time complexity of this version of shell sort is O(n^2)

b) *Why did your algorithm perform faster/slower than other algorithms?*

Shell sort is essentially an enhanced version of Insertion Sort. So, it is fast than sorting algorithms slower than insertion sort for the same reason. It keeps shifting the value as long as the value to its left is greater until it hits a smaller value. However, the difference between Shell sort and Insertion sort that makes shell sort faster is that it swaps items at greater distances over a “gap” that grows smaller over time rather than swapping positions one at a time. Therefore, the item being swapped is more likely to be much closer to its final position after a singular swap in shell sort than insertion sort. Once the array is almost in order, the “gap” would become 1 making shell sort essentially insertion sort, which works at very fast speeds in partially sorted arrays. Shell sort is slower than algorithms such as Quick Sort and Java Sort as it struggles with larger arrays, as larger arrays are more likely to be disorganized where shell sort works at maximum speed once arrays are partially sorted. It still goes very fast according to the project timers, but in many instances, it takes a little longer than quick sort and java sort.