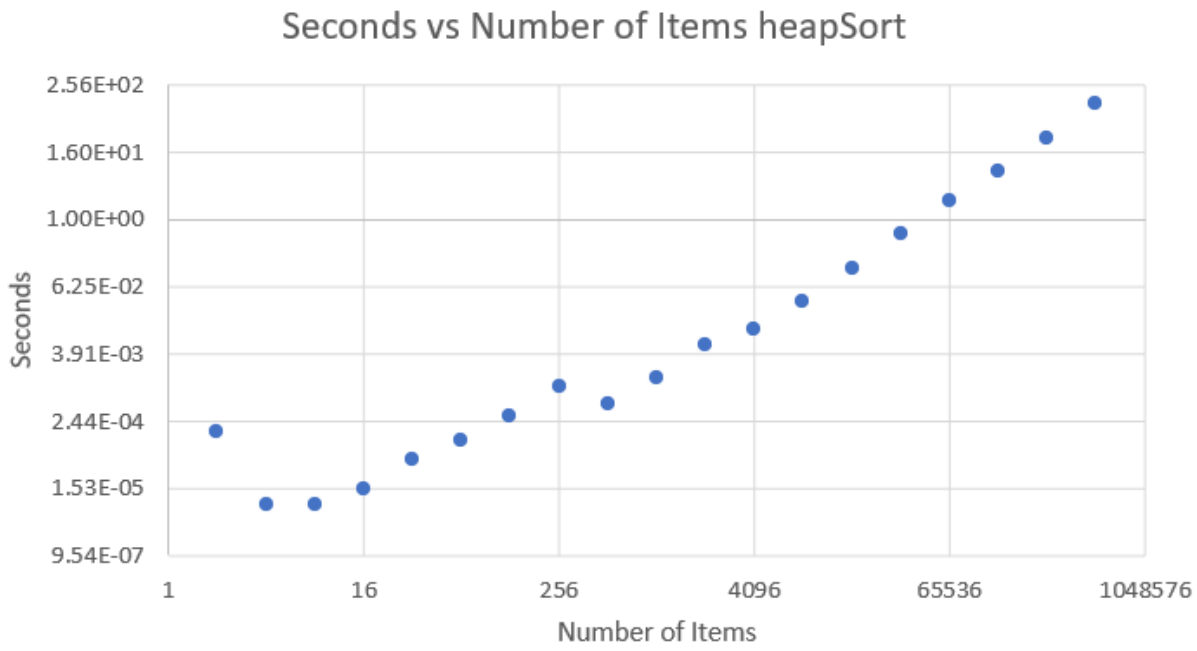
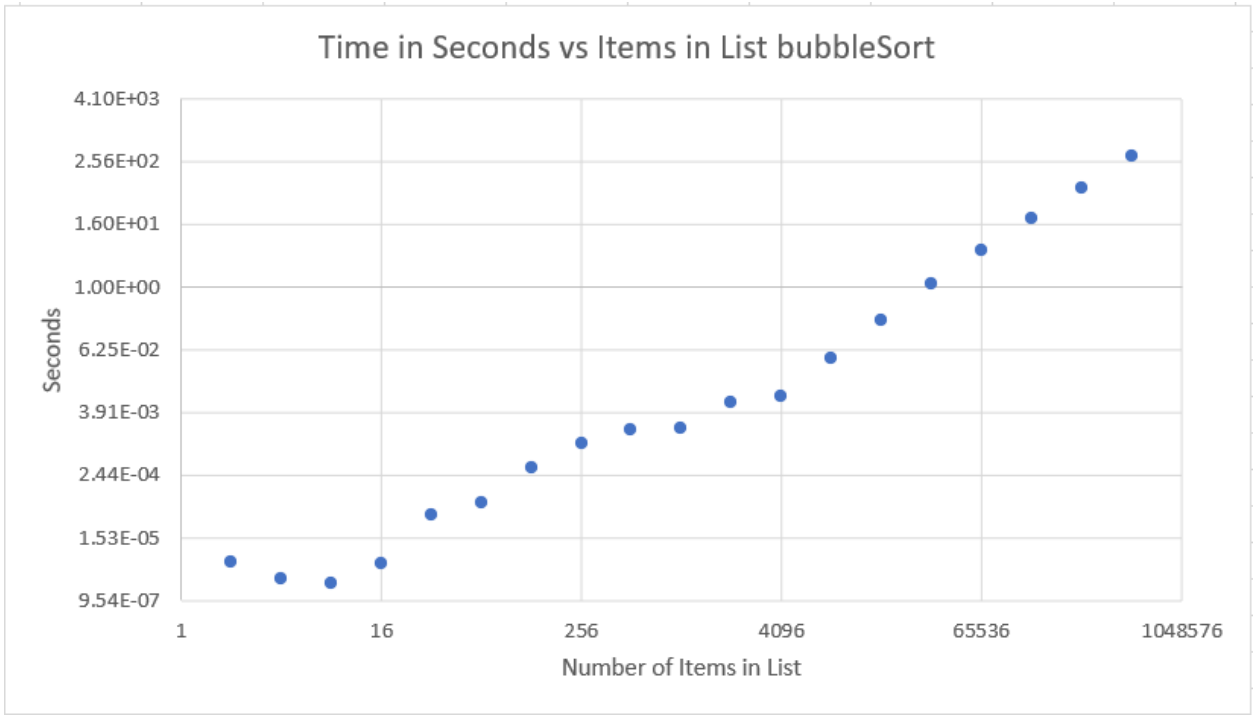
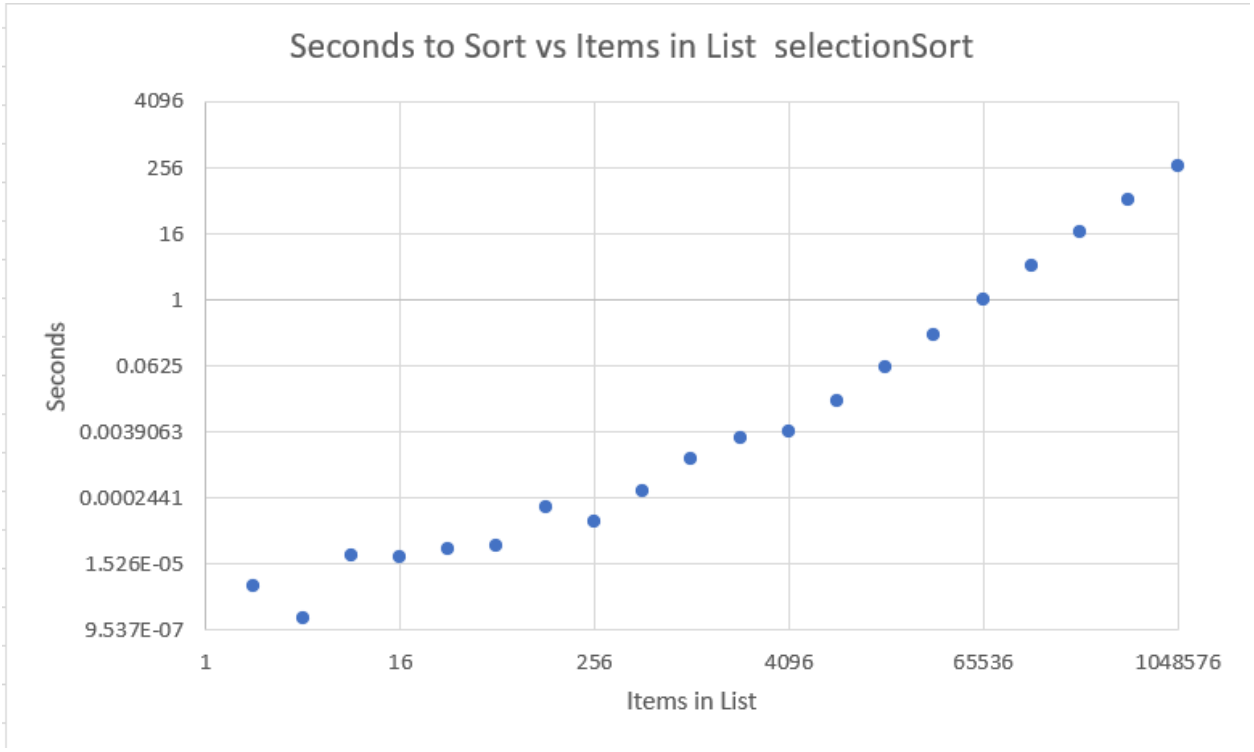
Intro:

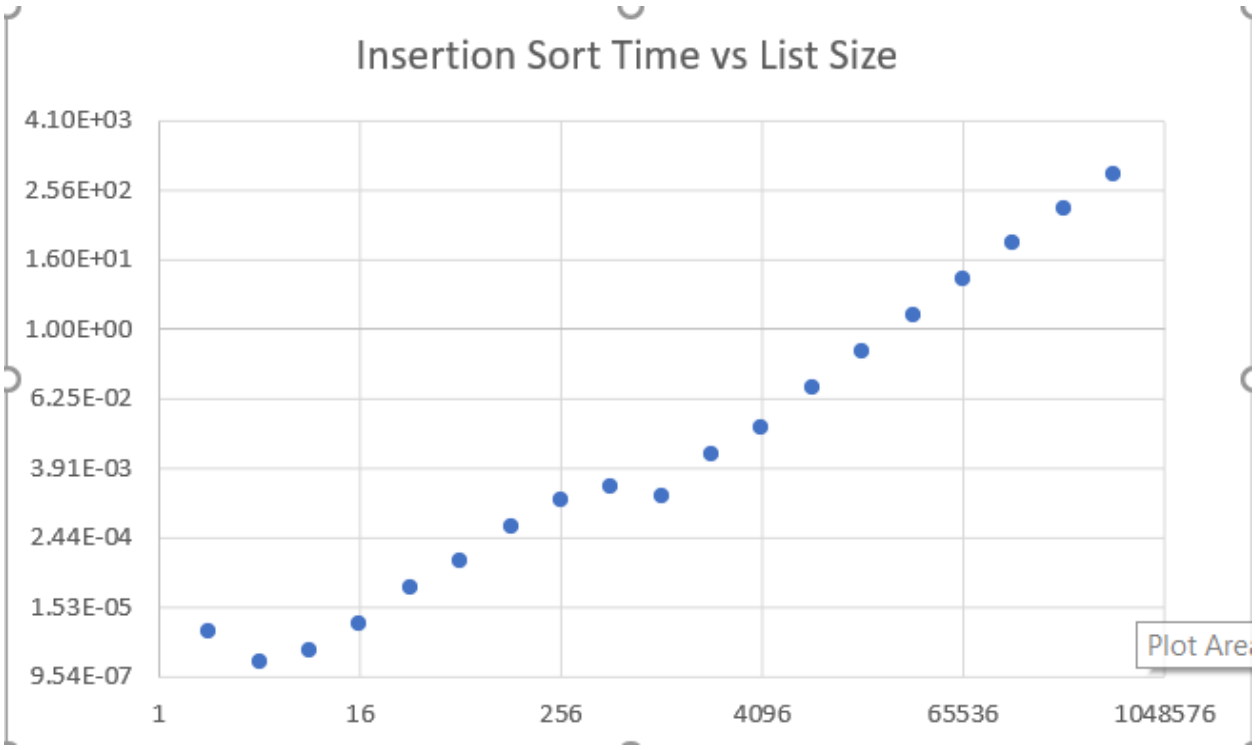
The assignment was to write and benchmark four different sorting algorithms in Java: Bubble Sort, Selection Sort, Insertion Sort, and one sort that had a time complexity better than $O(n^2)$. I decided to go with Heap Sort because I think sorting things with heaps sounds kind of cool and also the Geeks for Geeks page I read about said they were made with 'Complete Binary Heaps' and that sounds even cooler!!! Afterwards, we had to graph the performance of each of the sorts. Since our fourth sort had to perform with better time complexity than $O(n^2)$, I predicted that my Heap Sort would look better on the graphs when compared to the rest of the sorts.

Time Comparisons:





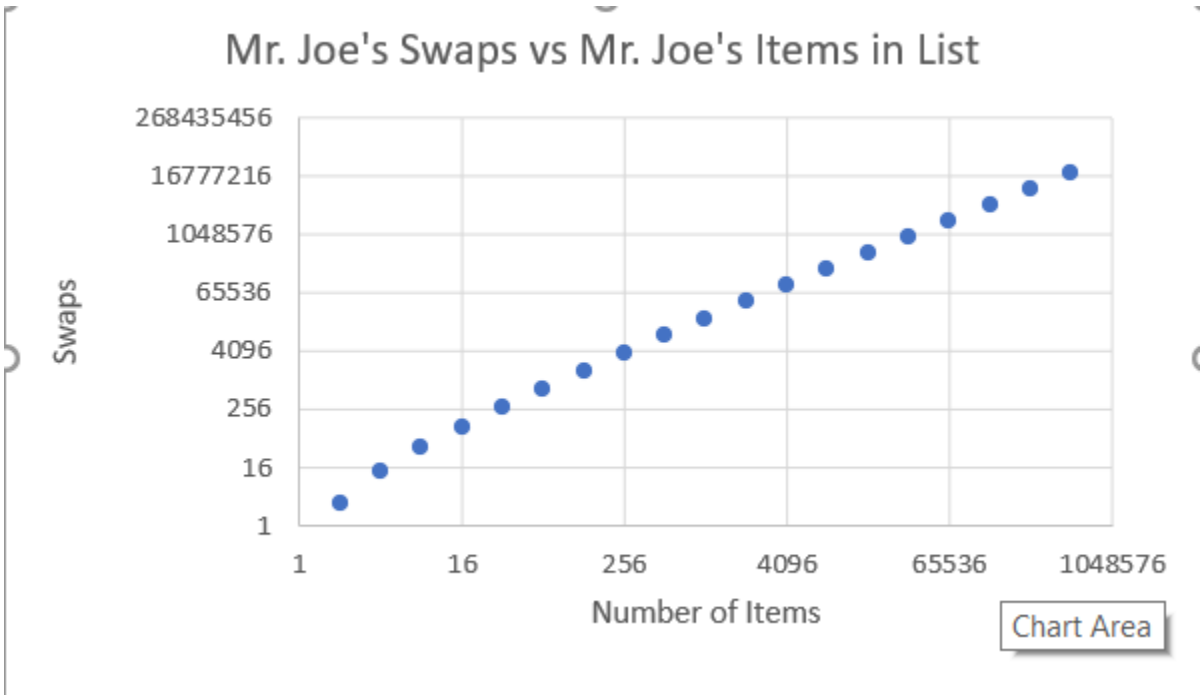


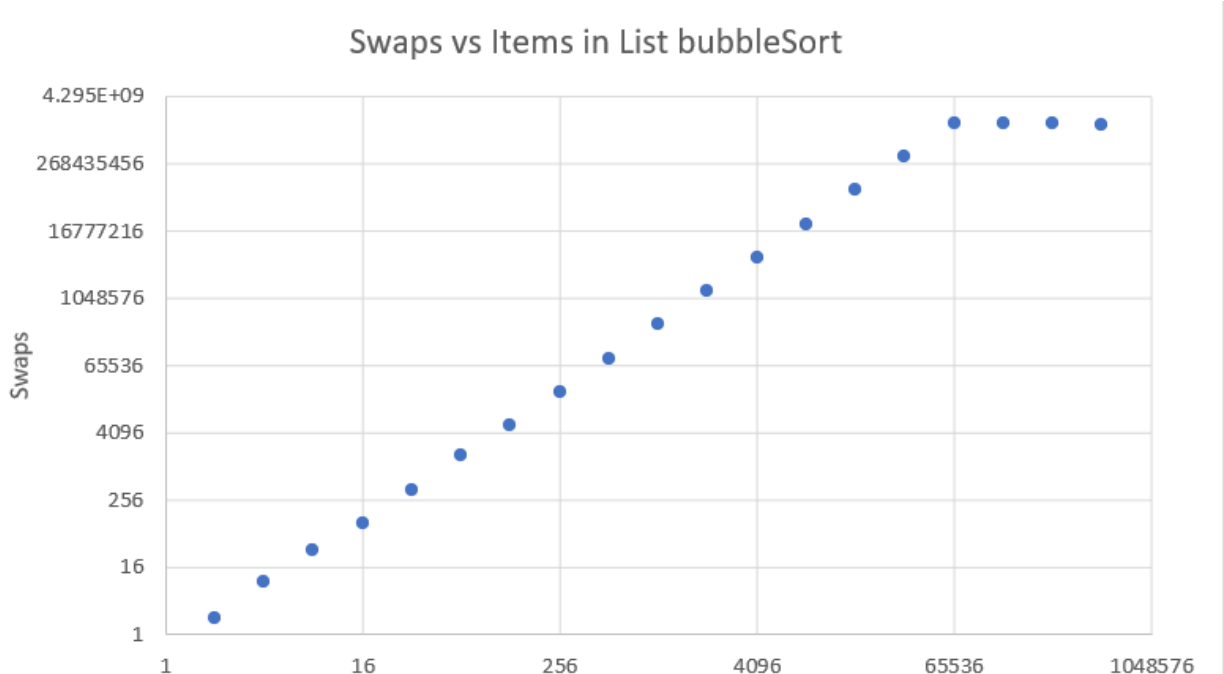


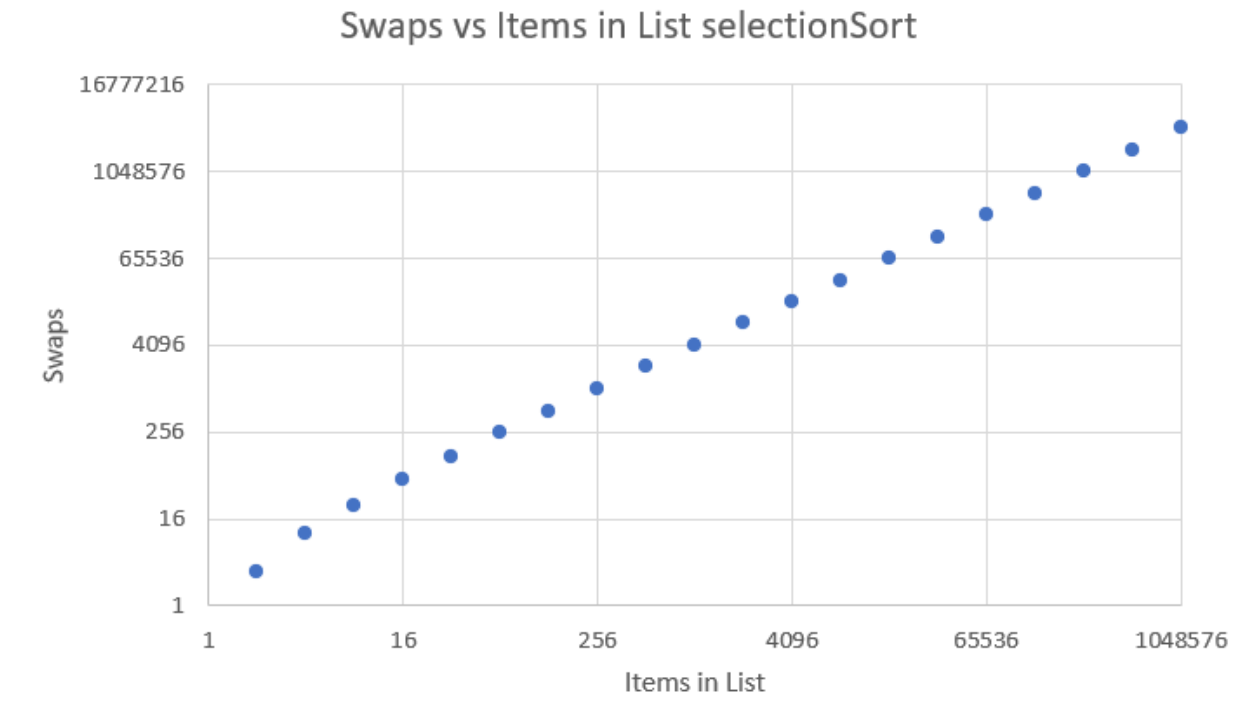
In order, it looks like the quickest sorts under five minutes were: heap, selection, bubble, and insertion. I think it's easy to see why insertion sort is the slowest operation, as it is a double

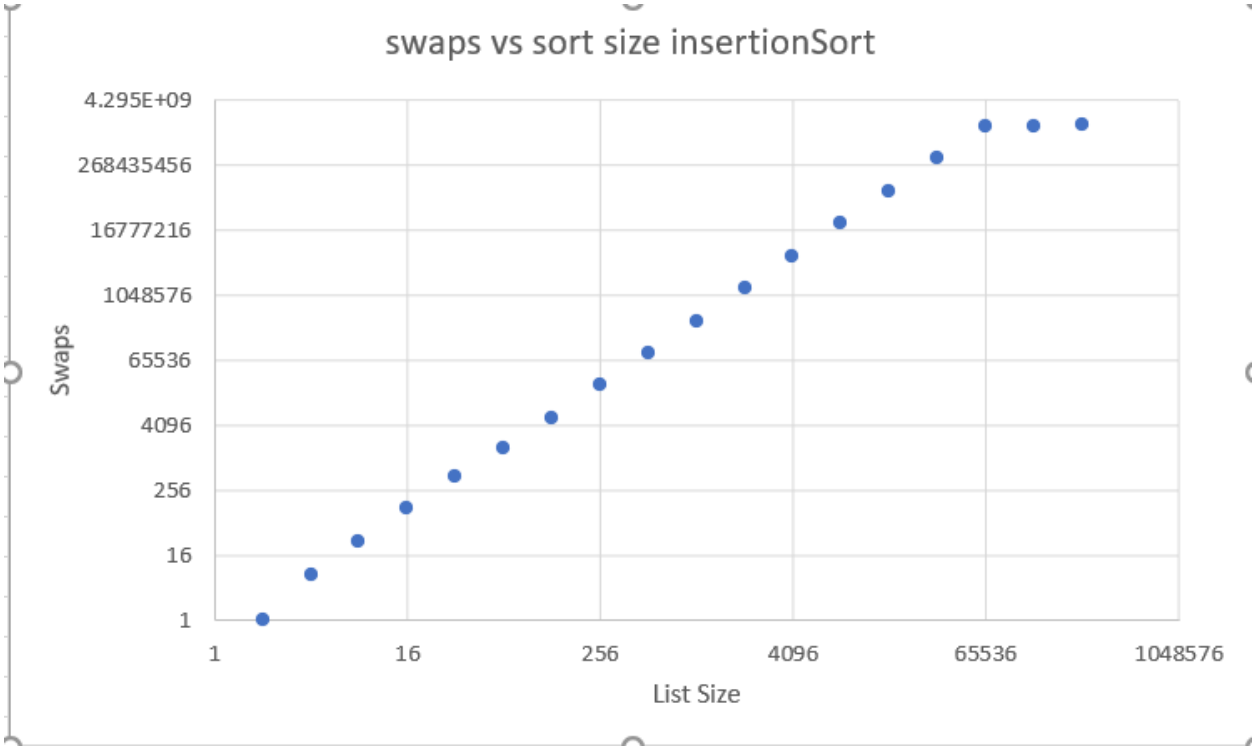
for-loop and a while-loop for as long as the previous element in an array is greater than the currently selected element. On the same token, heap sort only involves creating a heap (which has a time complexity of N) and removing the root from the heap (which is log(N)). Interestingly, there is a point in both Selection Sort and Heap Sort where sorting a small array, about two elements, is significantly slower compared to an array of about four or eight sizes. There is similar behavior in the other sorts, but I thought it was more noticeable in those two.

Swap Comparisons





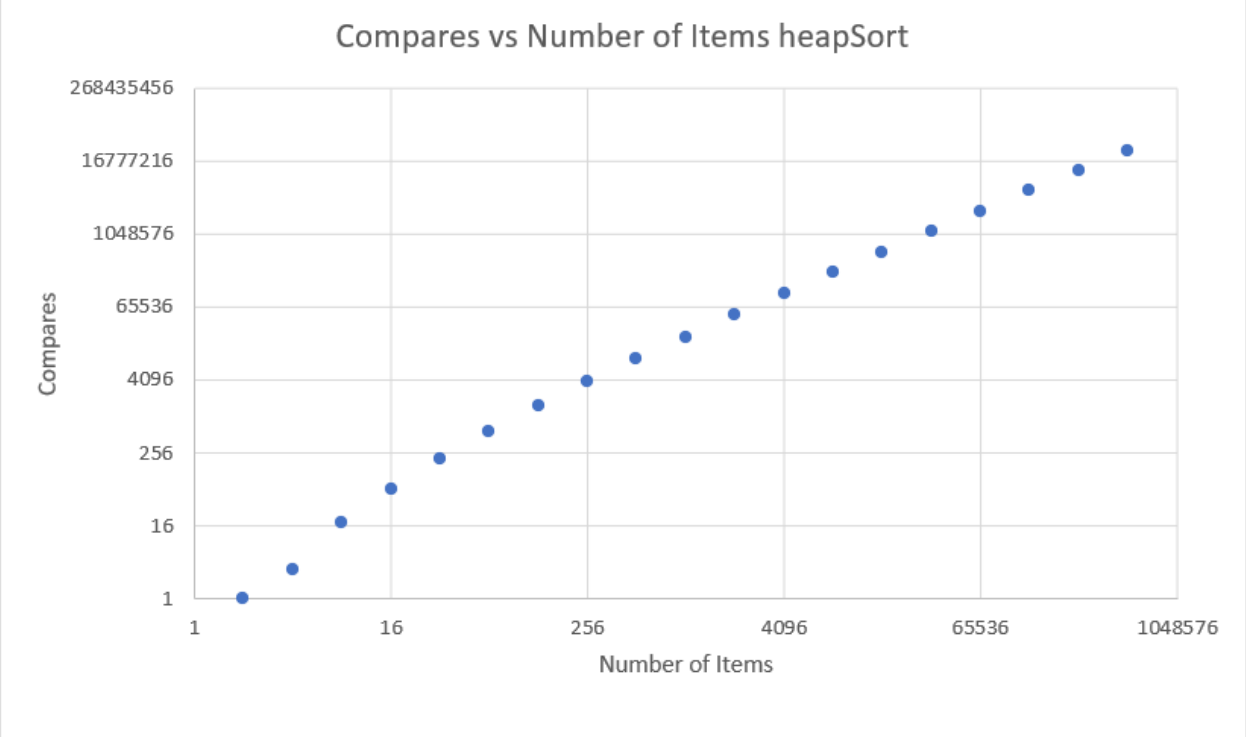


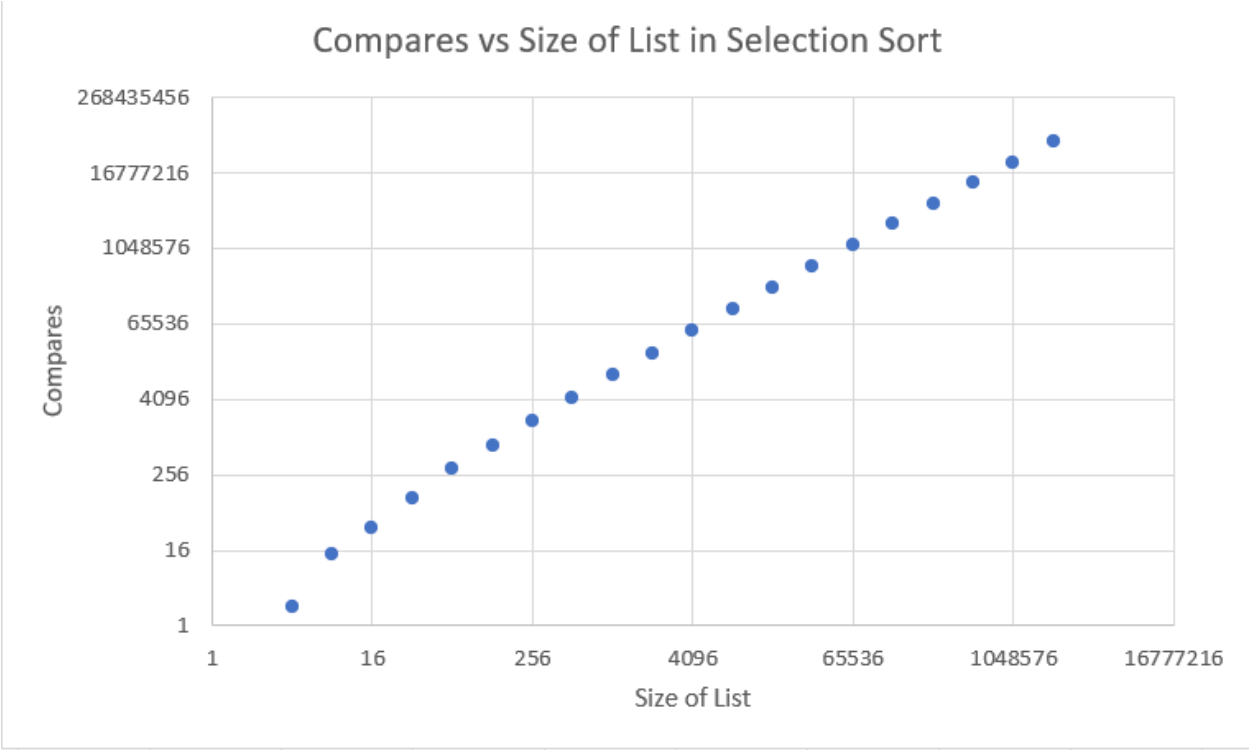


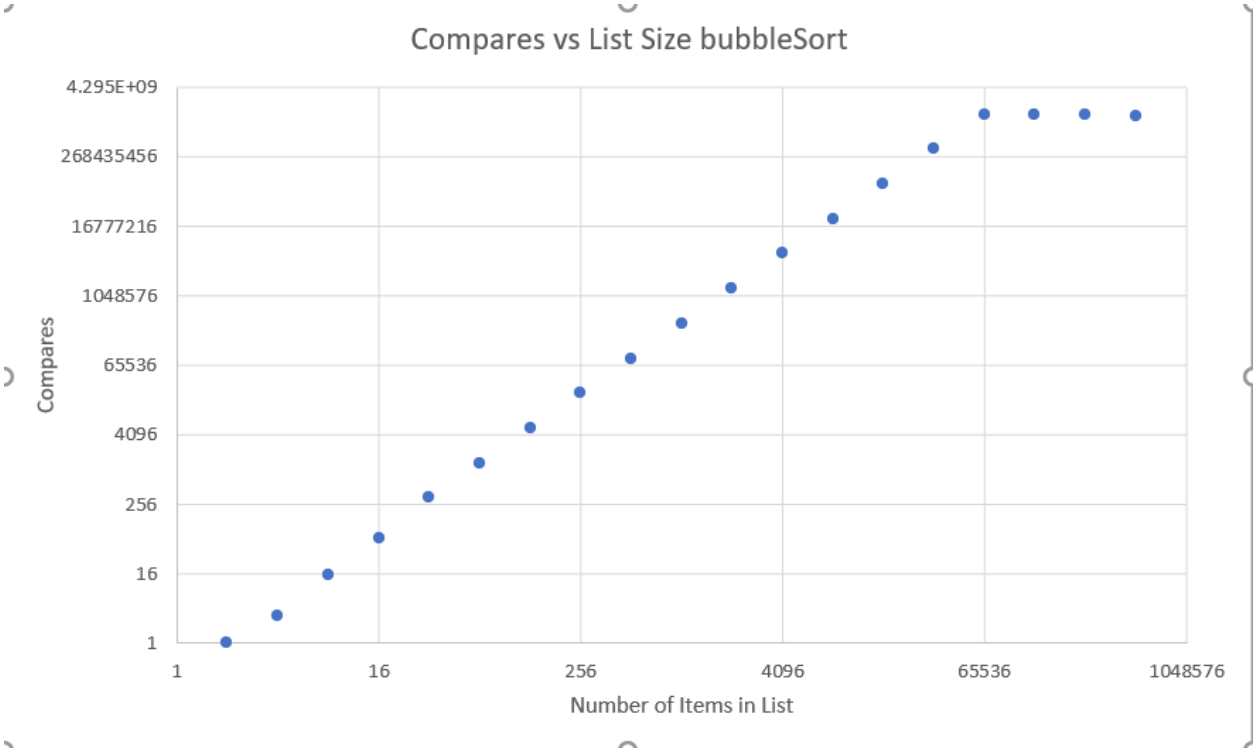
I expected the worse-performing sorts to have more swaps when compared because doing more swaps would mean doing more operations which could slow the sort down. The graphs

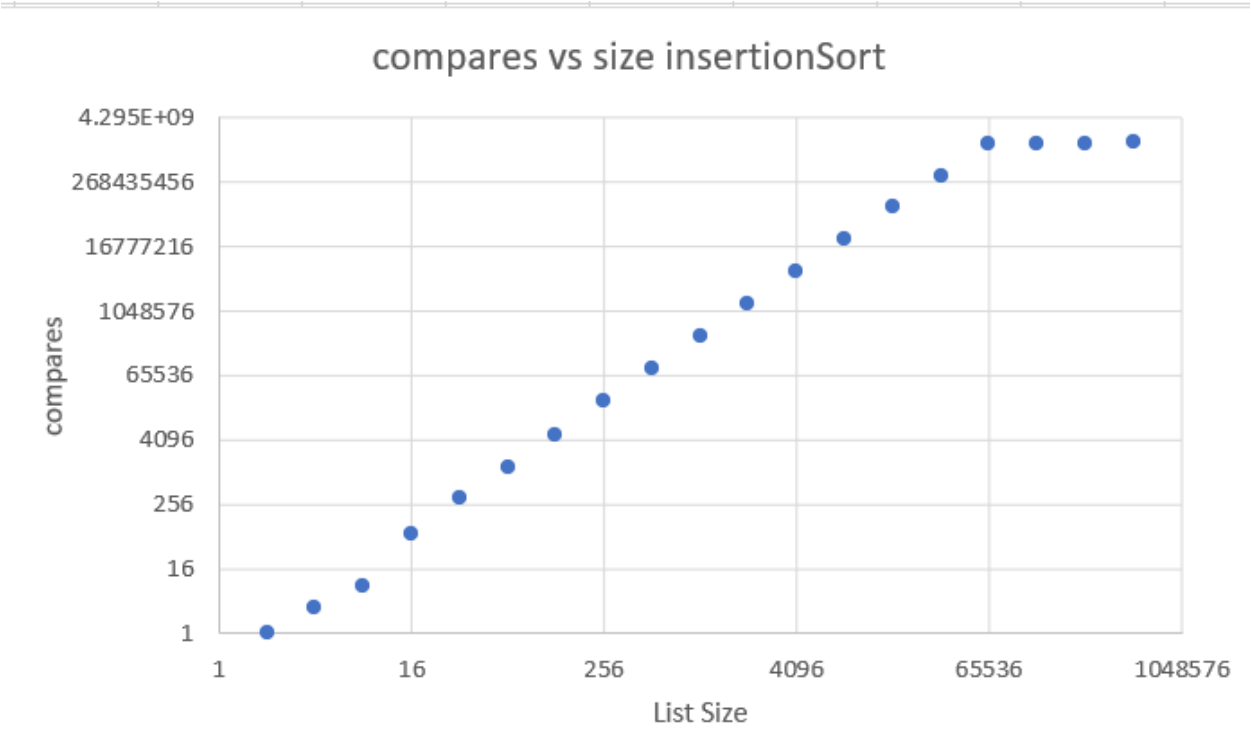
mostly reflect this, except with Selection Sort and Heap Sort (Heap Sort being the Mr. Joe graph). This was a little surprising to me because in the time comparisons Heap Sort appears to perform visibly better compared to Selection Sort. Then again, the differences in them on the array sizes tested was relatively small. Also, once I thought about how you had to maximize a heap (if that is the correct term) I began to see how there could be the slight differences in swap counts.

Compare Counts:




Compares vs Size of List in Selection Sort
Compares
268435456
16777216
1048576
65536
4096
256
16
1
1
16
256
4096
65536
1048576
16777216
Size of List


Compares vs List Size bubbleSort
Compares
4.295E+09
268435456
16777216
1048576
65536
4096
256
16
1
1
16
256
4096
65536
1048576
Number of Items in List



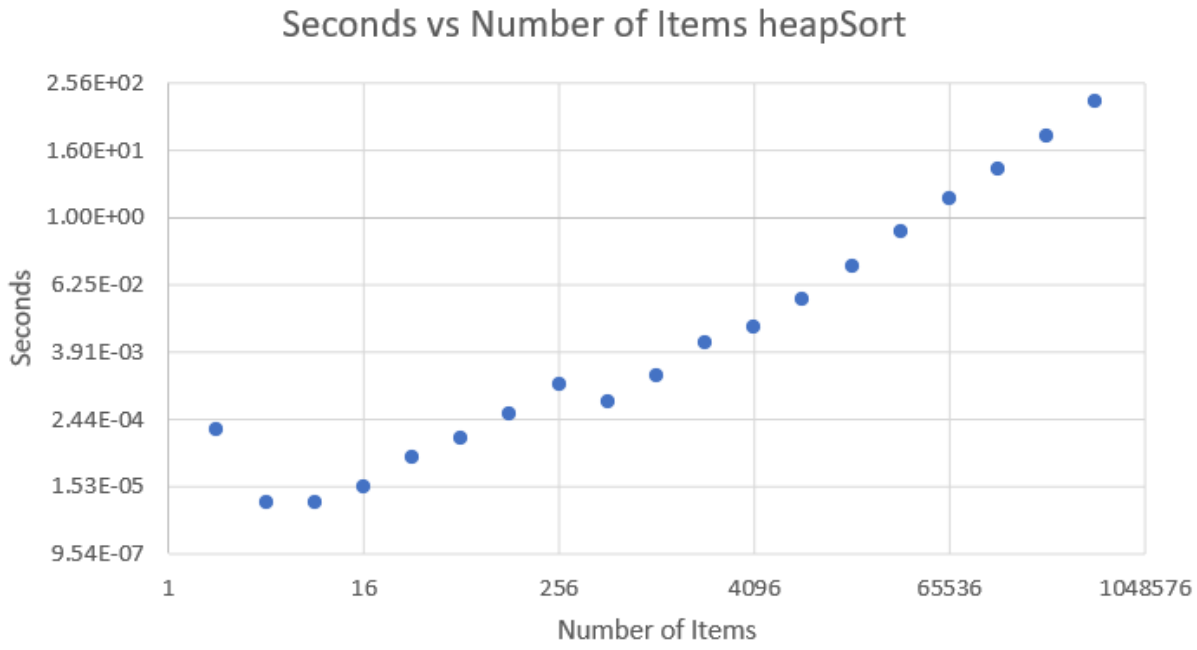
I didn't know what to expect for these graphs; all sorts seemed like they did a lot of comparing, and so to be honest I didn't really keep track of them to think about which ones would be betterer later. I think it's interesting that Insertion Sort and Bubble Sort straight up flatten out, though. I checked their time graphs to see if they appeared to flatten out there, too, but they did not. If the sorting times did not take so long when I was graphing them, and if my battery wasn't about to die at the time, I would have investigated that further to see if I could make the lists bigger and see how the compare count continued to behave on the graph. This was also how the graphs looked for the swap counts, too.

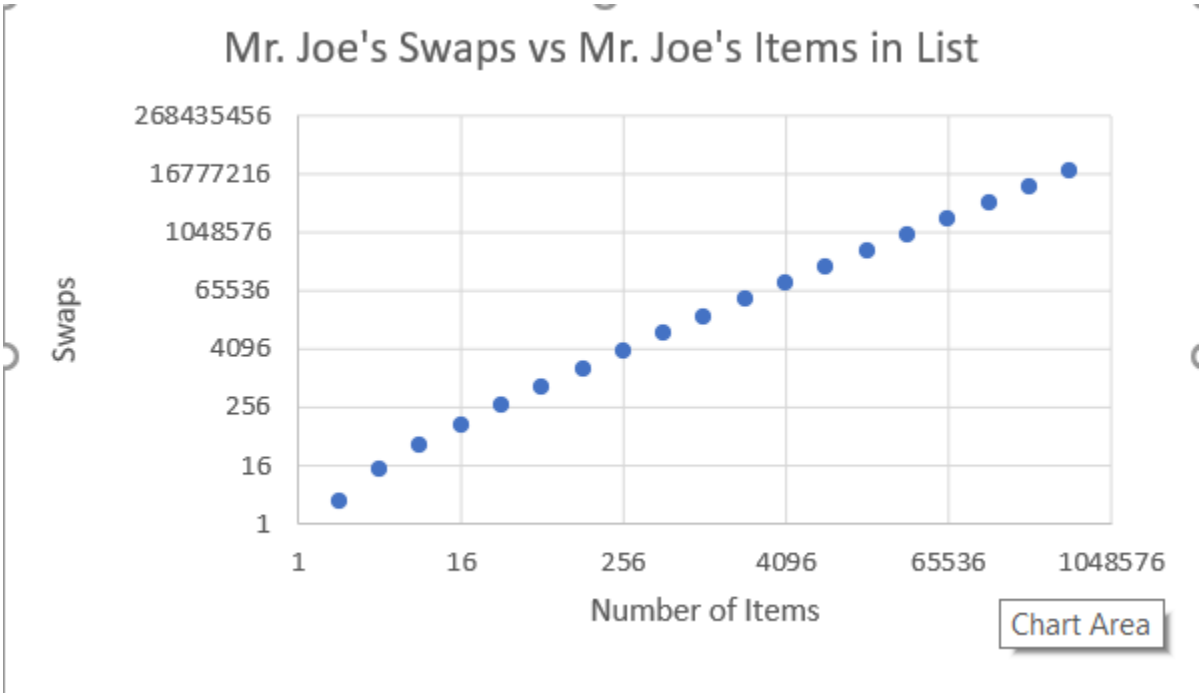
Conclusion:

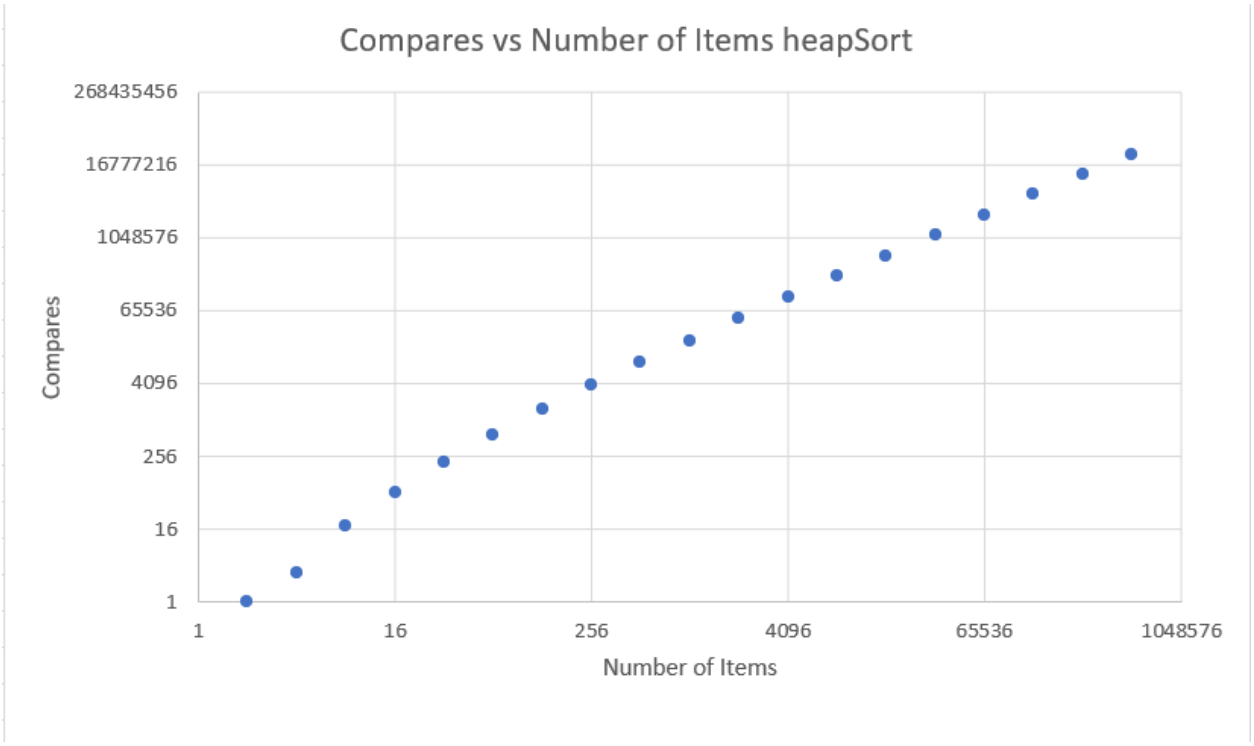
Besides just one exception or two, I think I was right to say that Heap Sort would perform better than the rest of the other sort algorithms I did. The exception was in the swap count for

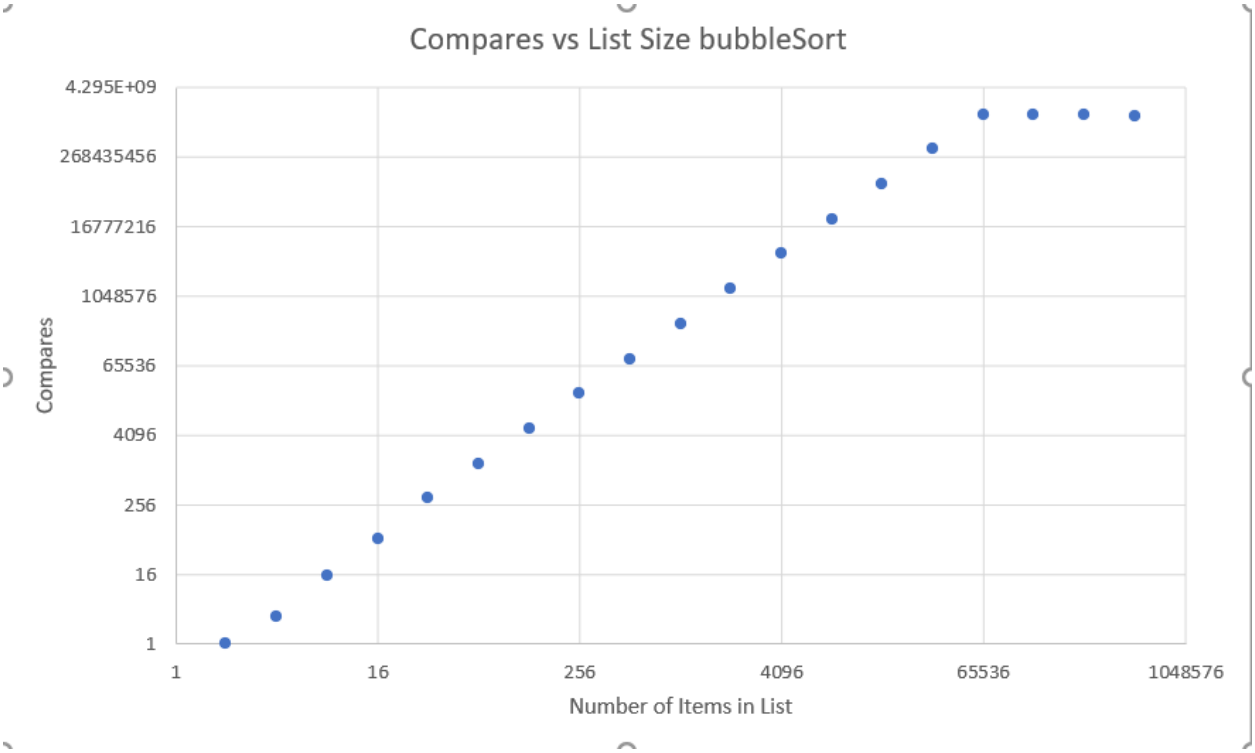
Selection Sort, which had slightly fewer swaps than the Heap Sort did. I did not expect that at all, because in my code all selection sort has is a swap function in a for-loop, whereas Heap relies on some conditional logic before it does swaps. If I think about it hard enough I can see it, though.

Heap Sort Figures:

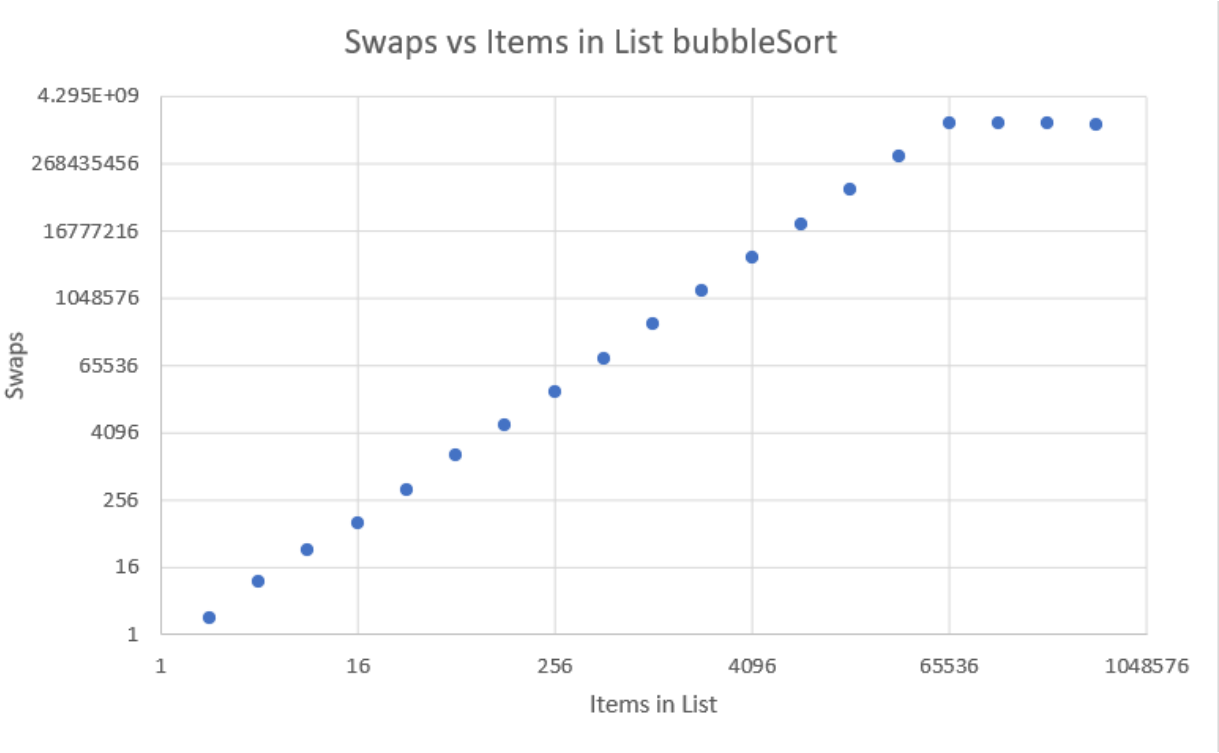


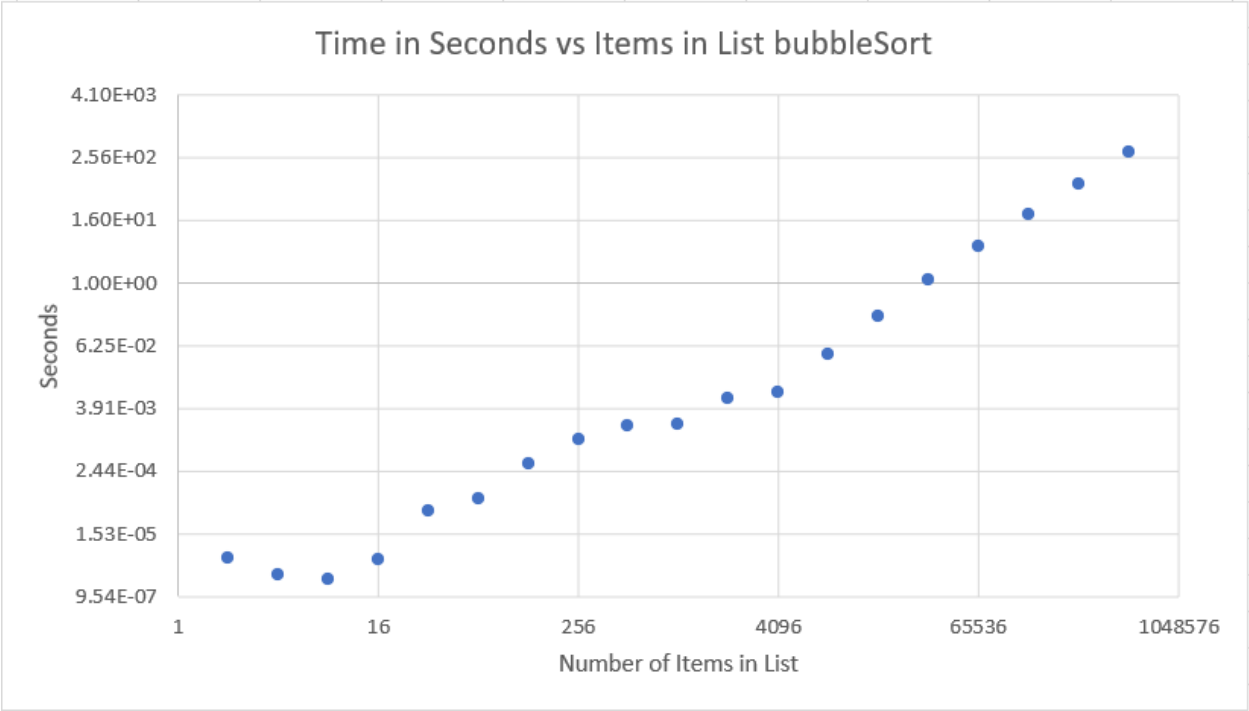


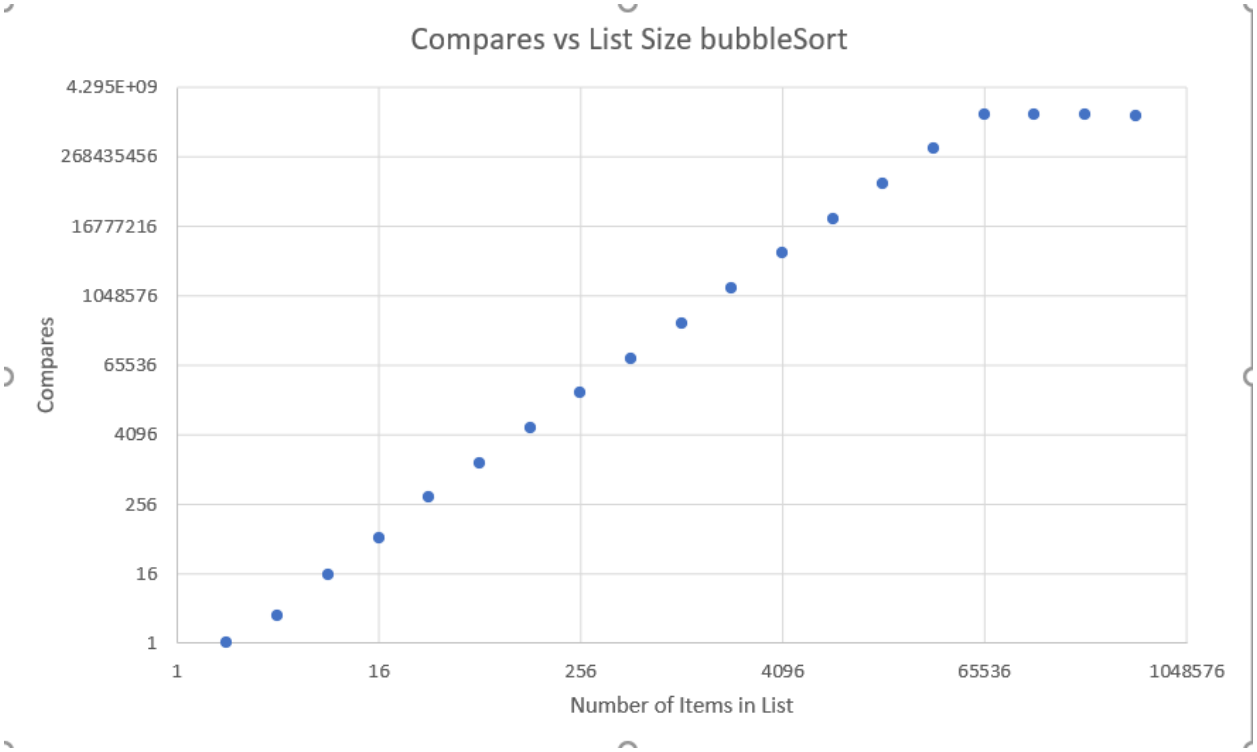

Compares vs Number of Items heapSort
268435456
16777216
1048576
65536
4096
256
16
1
Compares
1
16
256
4096
65536
1048576
Number of Items


Compares vs List Size bubbleSort
4.295E+09
268435456
16777216
1048576
65536
4096
256
16
1
Compares
1
16
256
4096
65536
1048576
Number of Items in List

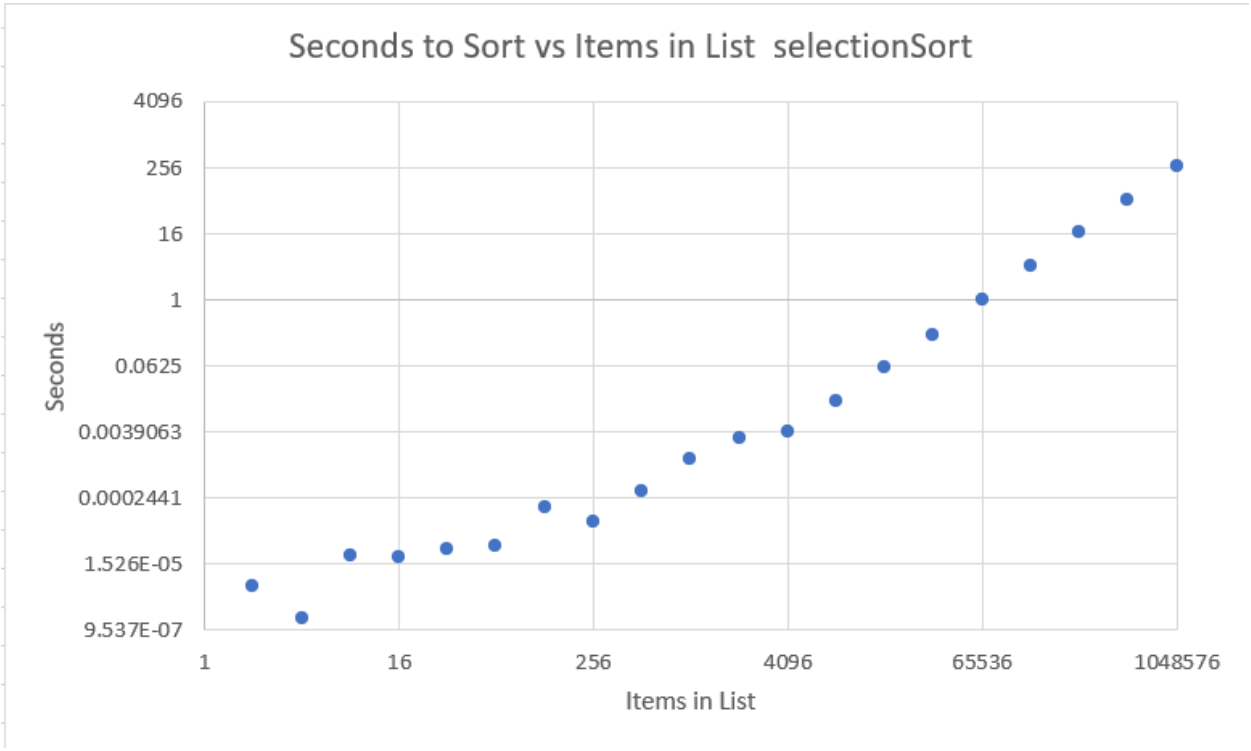
Bubble Sort Figures:

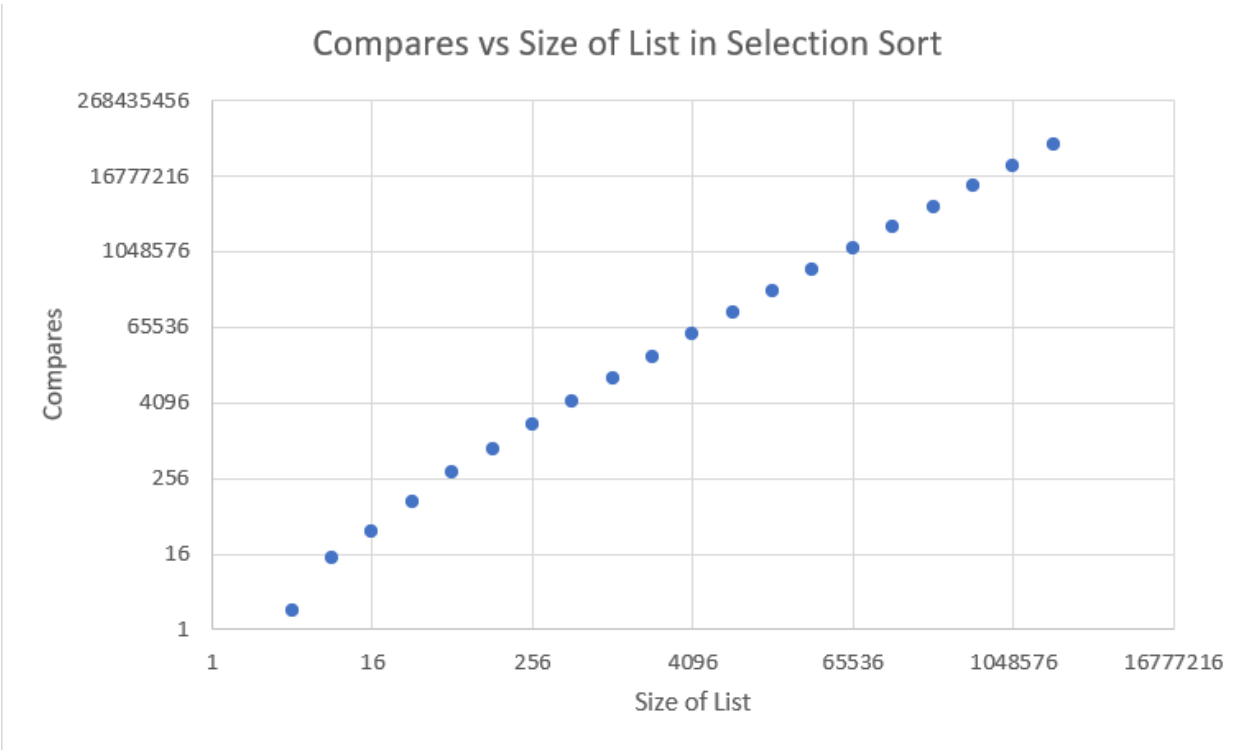


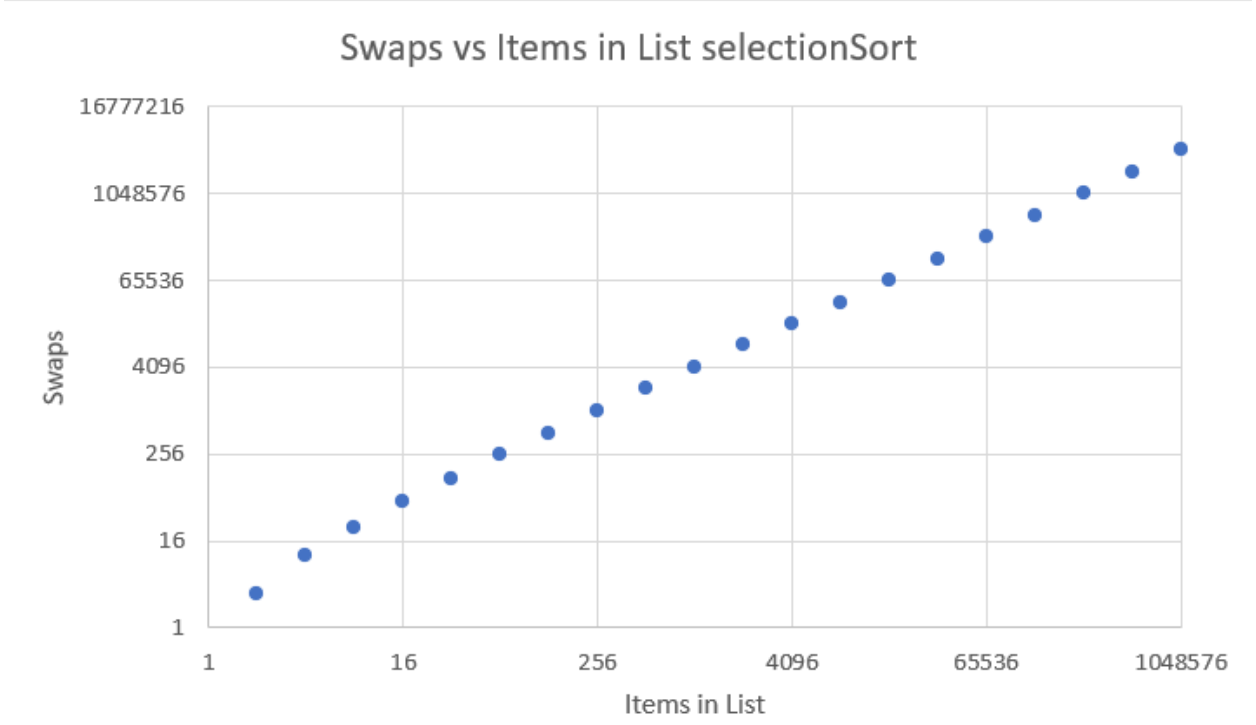


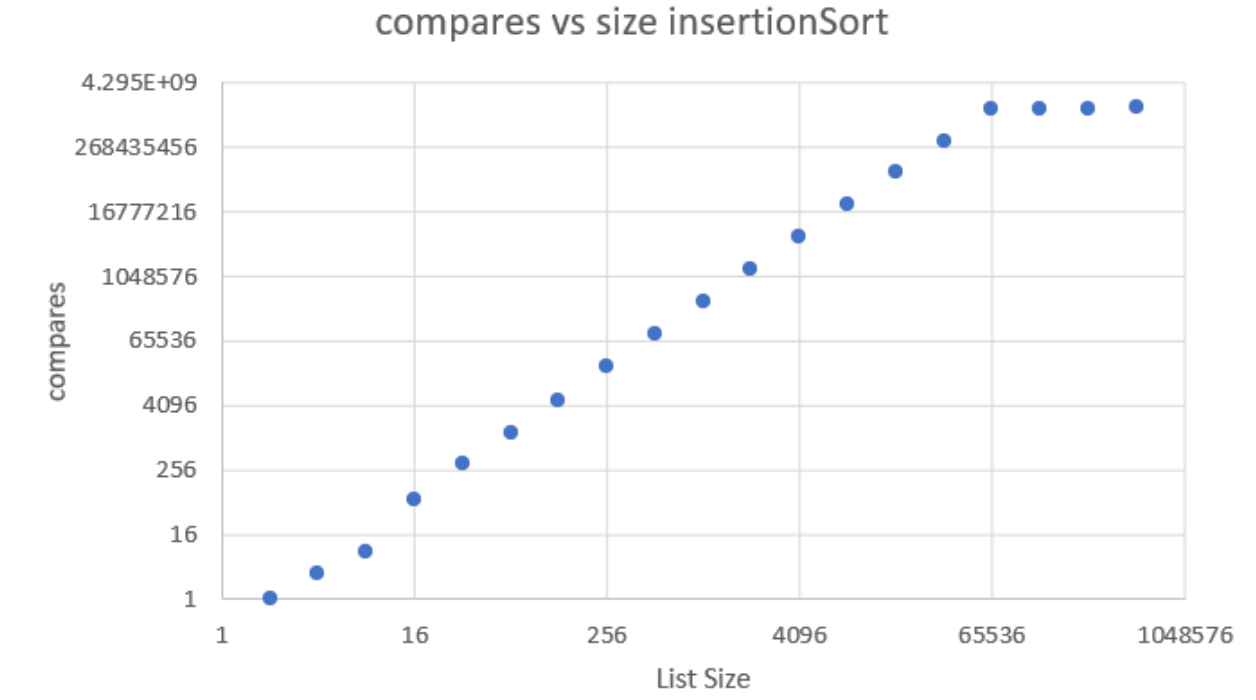

Compares vs List Size bubbleSort
4.295E+09
268435456
16777216
1048576
65536
4096
256
16
1
Compares
1
16
256
4096
65536
1048576
Number of Items in List

Selection Sort Figures:

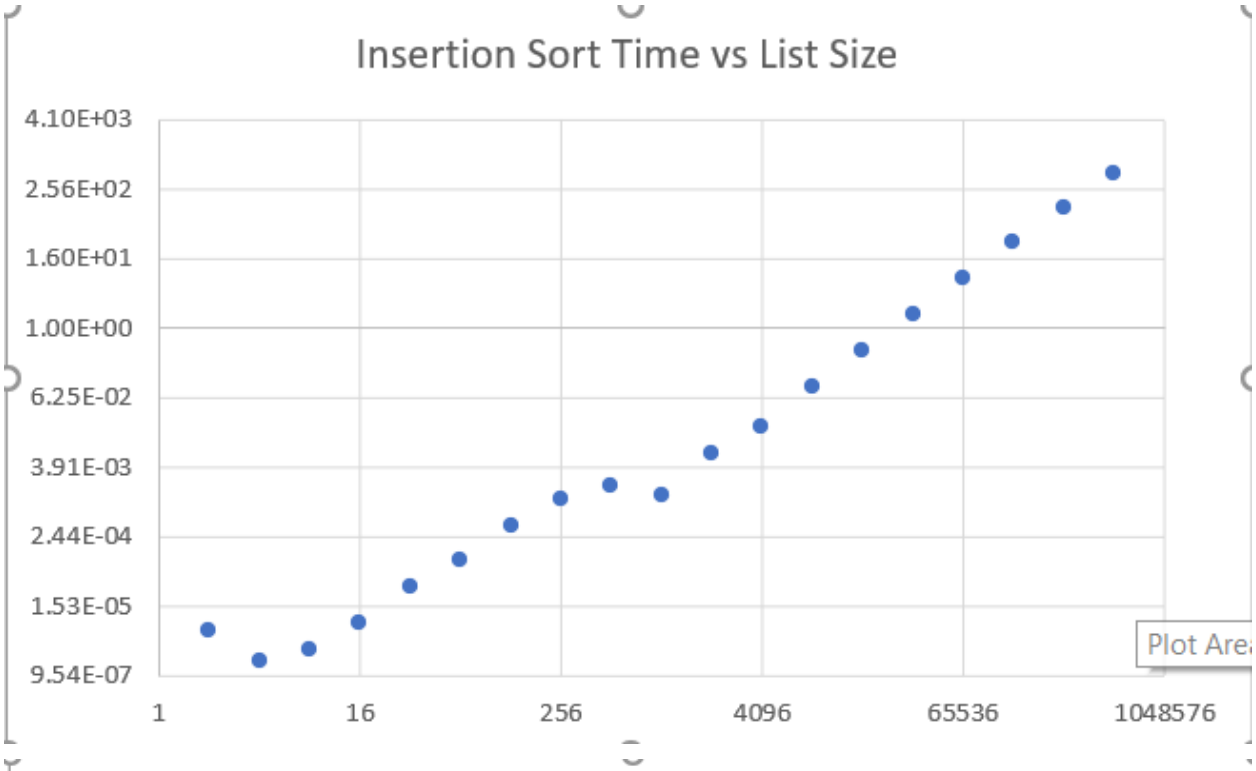


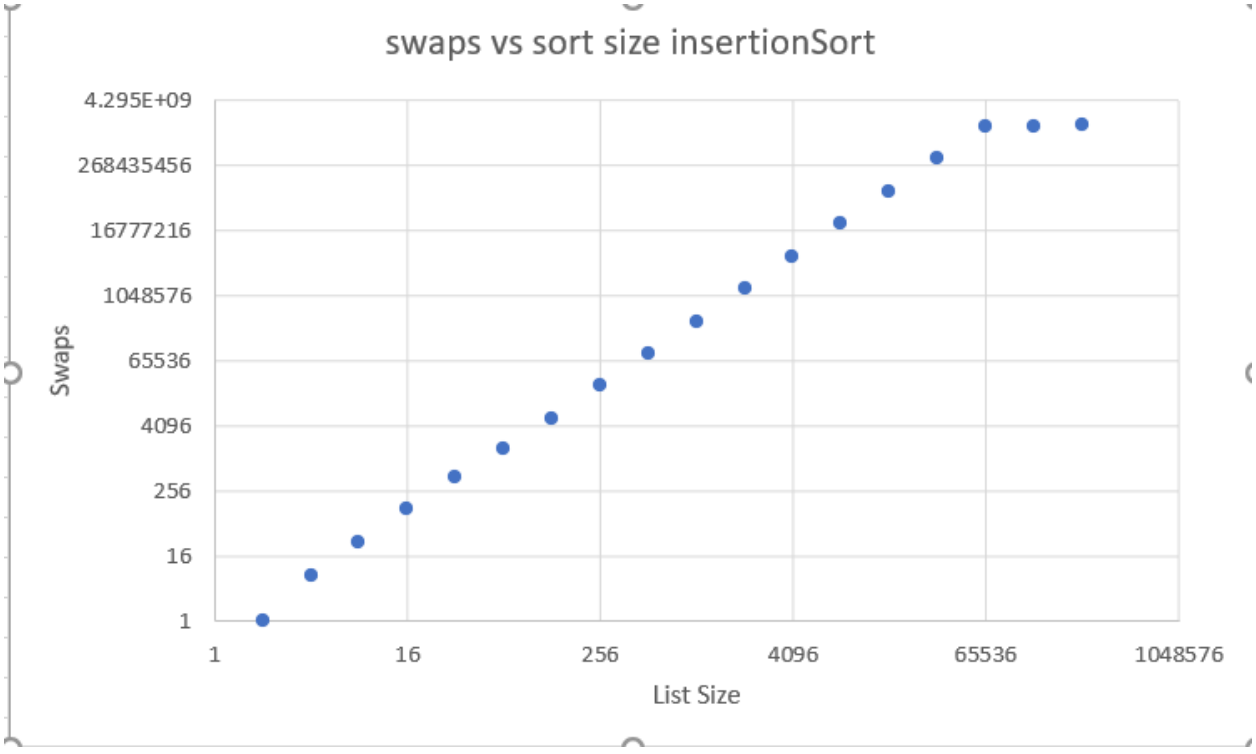


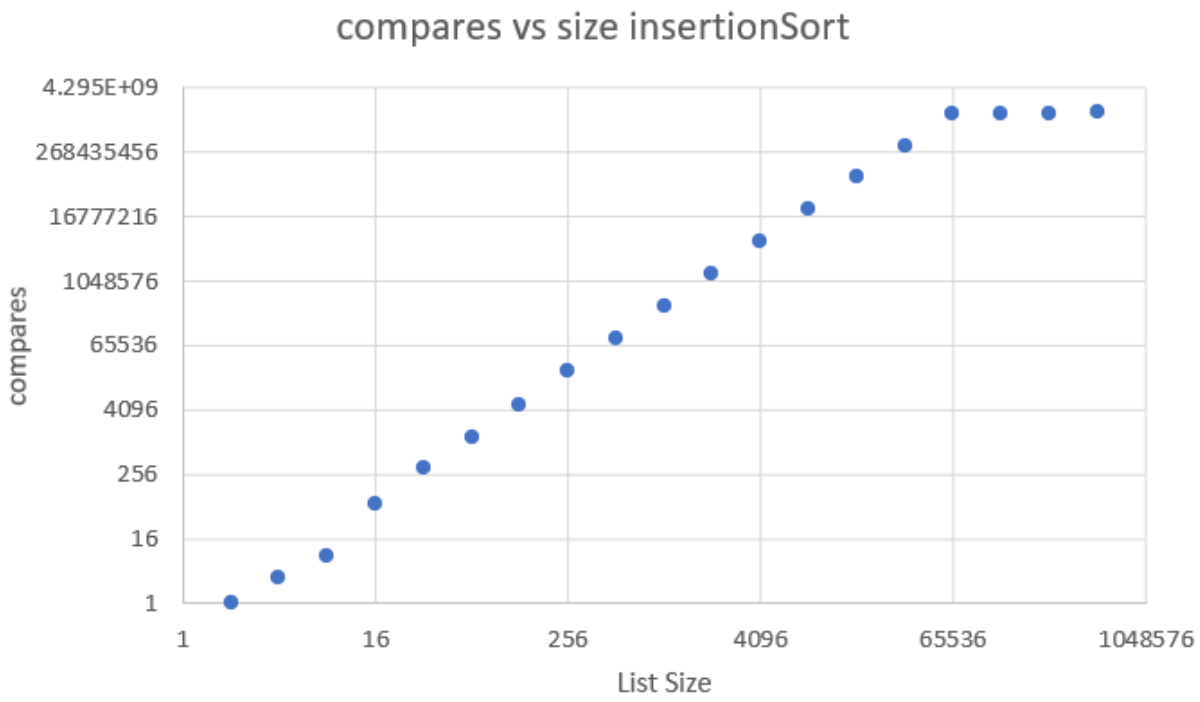

Swaps vs Items in List selectionSort
16777216
1048576
65536
4096
256
16
1
Swaps
1
16
256
4096
65536
1048576
Items in List


compares vs size insertionSort
4.295E+09
268435456
16777216
1048576
65536
4096
256
16
1
compares
1
16
256
4096
65536
1048576
List Size

Insertion Sort Figures:






compares vs size insertionSort
4.295E+09
268435456
16777216
1048576
65536
4096
256
16
1
compares
1
16
256
4096
65536
1048576
List Size