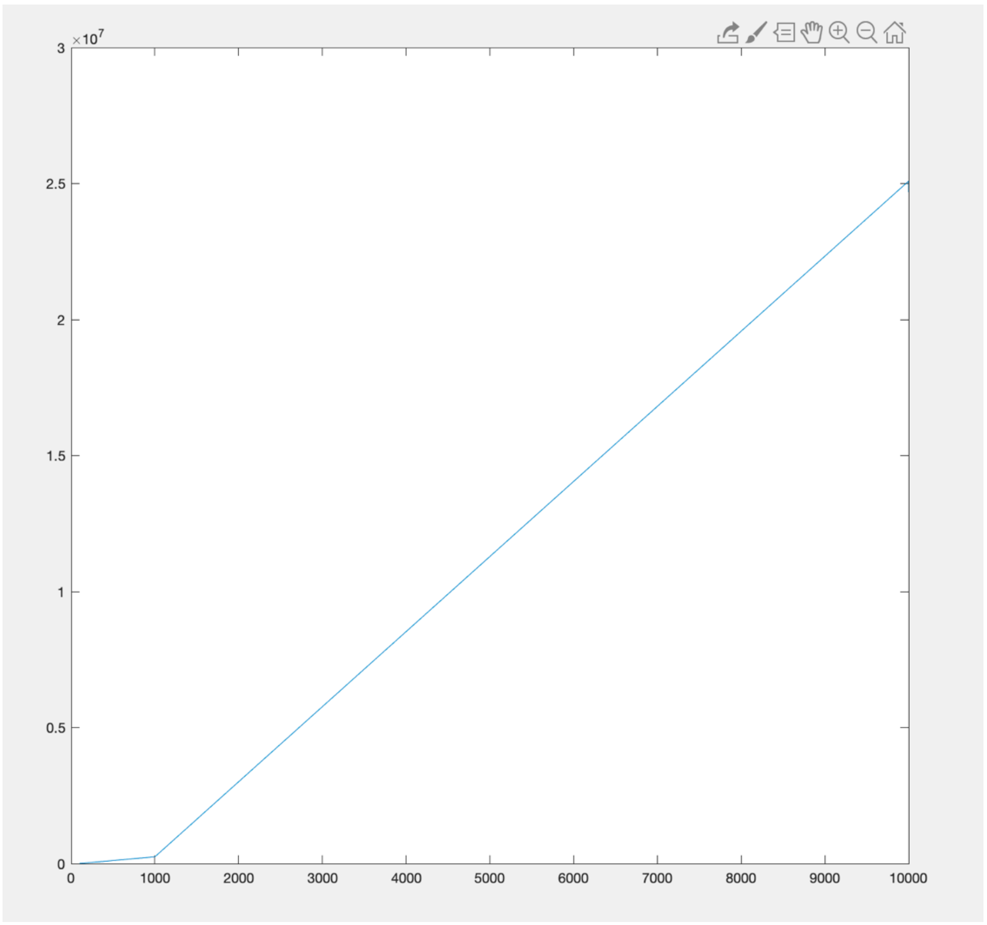
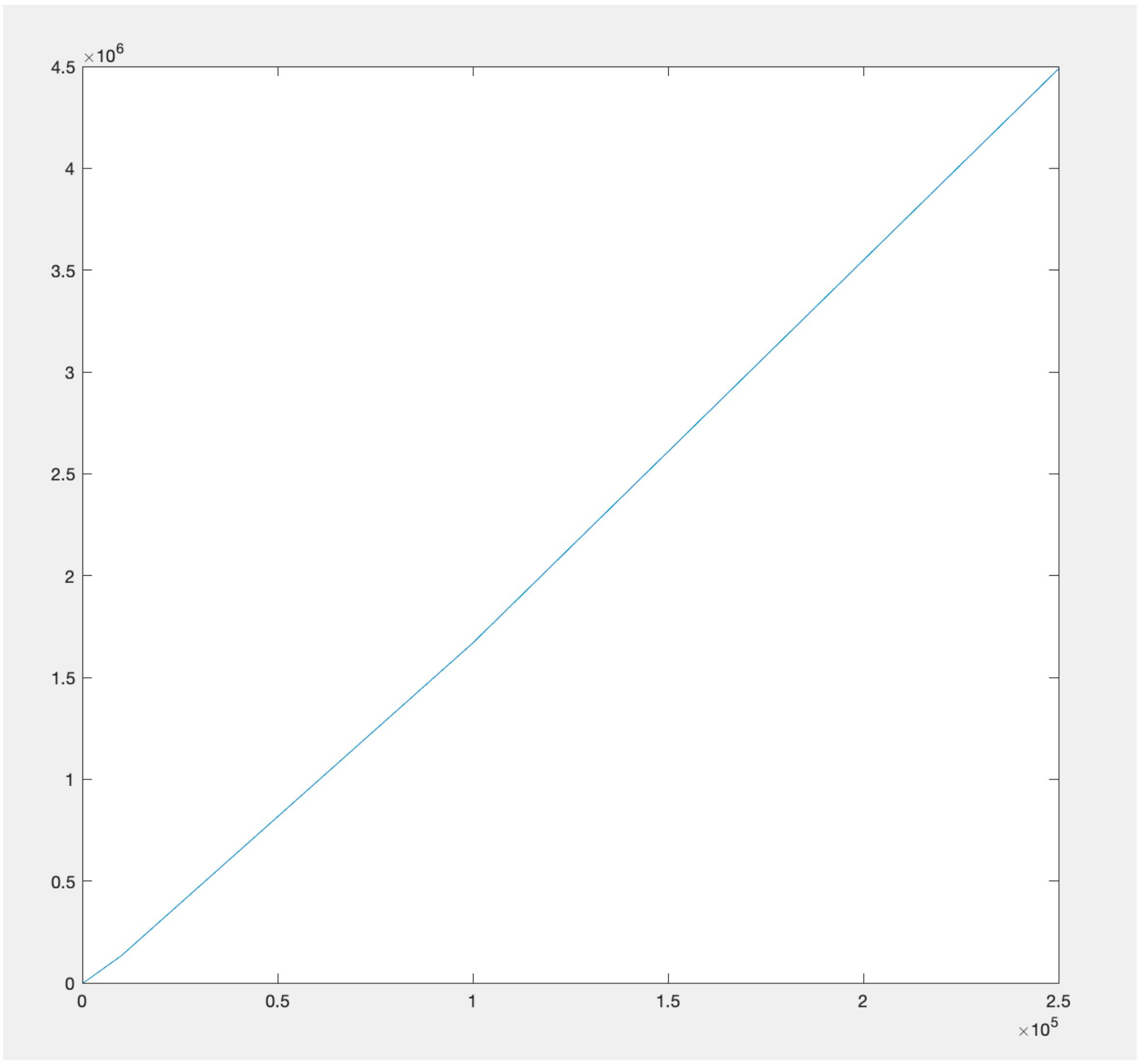
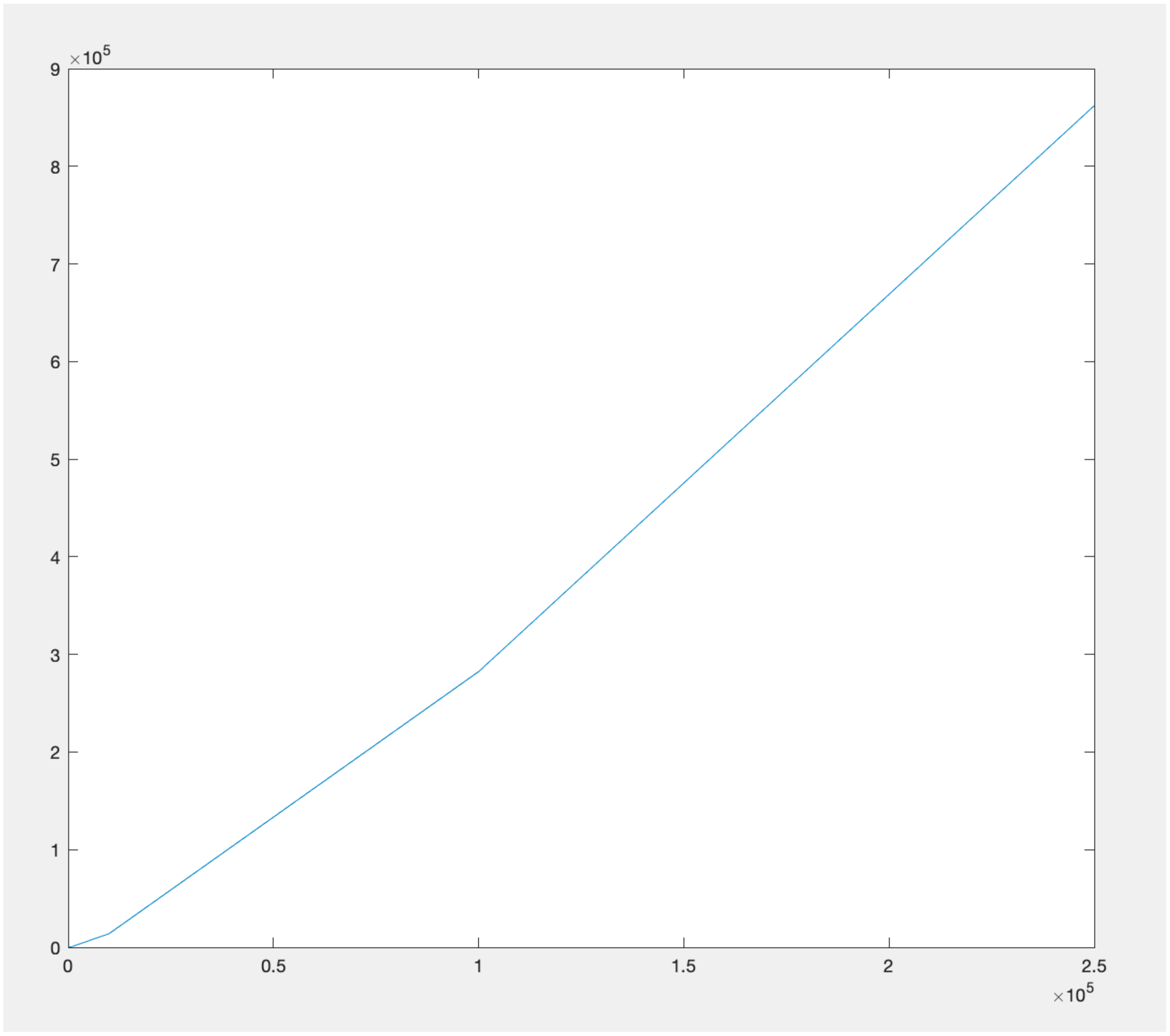
To complete these algorithms, I had to write three different algorithms. Each algorithm returns the list inserted into it. Because Merge and Quick were both recursive, I made the counter a static variable, and wrote a static method to return the counter after it had been manipulated upon the running of the algorithm. The algorithm then set the variable back to 0. For the Bubble sort, the counter was included in the algorithm, and so I did not need to call any additional function in the main(). Furthermore, I tried to be fair with my counters in each algorithm, and nested a counter in inner-most loops, such that the counter increase everyone the program has to loop through something.
The three graphs below were made with MATLAB. Also, I commented out the tests for Bubble sort on cases of 100,000 and 250,000 because it just takes way too long to compile; thus, the graph for Bubble only accounted for 3 array sizes. That said, it should still be enough to get the gist of how the graph looks.



This is the graph for the Bubble sort made it MATLAB. It is a little bit hard to tell, but the graph shows the inefficiency of the algorithm. Just observe the scale of the x-ais and y-axis. As you can see, the Bubble sort because increasingly inefficient as the array size increases. This is because of how the Bubble sort works. It must cycle through a portion the array for every single variable in the array, making it extremely bad for large data sizes. The advantage to Bubble sort is that it is relatively simple and easy to understand. The Bubble sorts complexity is O(n^2).



This is the graph for Merge sort.   As you can see by the graph, it becomes less efficient as data size increases, but not nearly to the same degree as the Bubble sort. The result is that it is much better for larger numbers than the Bubble sort. Another interesting note about this algorithm is that (based of my results at least), it takes the same number of steps on any 2 inputs of the same size. The way the merge sort works is that it reclusively breaks down the input down to arrays of size one, and then merges them all together, one and one, then two and two, then four and four, etc…. Merge sort has a complexity of O(n logn).



This is the graph for the Quick sort. As you can see, the graph for the Quick sort looks very similar to that for the Merge sort. It should be noted, however, that the y-axis is on a different scale. The Quick sort, like the Merge sort, is a recursive search algorithm. The way Quick sort works is by picking an arbitrary pivot and partitioning each element into one of two sub-arrays, one in which the elements are larger than the pivot, one in which the elements are smaller than the pivot. It does this recursively. The complexity of the Quick sort is the same as the Merge sort, O(n logn).