Question 4

We started the problem by making a simple swap function for the bubble sort on the array. After reviewing some material on bubble sort, and some help from GeeksforGeeks, we were able to figure out and understand the implementation of a bubble sort and created a function for it. There is a nested loop in which the array will iterate itself and see if the value in front of it is less than the current one, if it is then it will swap. Then after you reach the end of the array, the next iteration will begin. In the next iteration the array will only go up to 1 less than the position it went up to before because the last value has already been swapped. This iteration will continue as long as I is less than the size of the array but more importantly, as long as j is less than the amount of elements remaining to be checked. In the bubble sort for linked list, we do something similar. We will continue swapping the current node with the node ahead of it as long as it is bigger. We have the same for loops as the array problem because the algorithm is the same. The loops will continue checking for the index ahead of the current one and will keep reducing the next iteration by 1 since those values have already been checked and swapped. Insertion sort was definitely trickier than bubble sort but we managed to make it work. In our insertion sort algorithm for an array we check multiple values against the current index, if the values are above this index, then they will be pushed up by one. This will continue to happen until the current index is the last one in the array. Our linked list does something very similar by pushing previous nodes that contain greater values up in the list. Then the current node would be incremented and the algorithm would simple repeat itself until the current node was the last one. In conclusion, the insertion sorts were much fast than the bubble sorts for both data structures. On average the insertion sorts were 2-3 times faster than the bubble sorts.