Bubble sort, being the simplest algorithm to implement and conceptualize, made it inefficient compared to the other sorts. This is due to the fact that it goes through the entire array swapping values, pushing the greatest value up. Because it has to go through the array, and again but with one less element, it makes the time complexity O(n^2) in its average case. Through research and class time I learned that we can optimize bubble sort by stopping the loop if it didn't make a swap, saving a lot of time that would otherwise have been used to go through the loop until the one last element is checked.

Shell sort is a variation of insertion sort. Instead of doing gaps of 1 throughout, shell sort utilizes an array of gaps. These gaps between values allow the algorithm to move values that are far away from each other faster, instead of going one by one like a bubble or insertion sort. When dealing with a large array, I observed that shell sort has a significant amount of fewer moves than bubble sort. One of the things I learned was the importance of the gap in shell sort. The average case time complexity really depends on the gap being implemented, An optimized gap can slightly improve the time complexity of the sort, with the best known being a ratio of 2.2.

Quicksort is by far the most efficient sort, as far as my testing goes. With elements of 100 to 10000, quicksort, compared to the other 3 sorts covered in the assignment, made relatively fewer moves and compares. Quicksort's average time complexity is O(n log n), although it can be made much more complex when the pivot value that is chosen happens to be an extremity, either the largest or smallest number in the list. This makes partitioning of the array extremely off-balanced, having one side with only one element. This can be solved by computing the median of the array and using that as the pivot, or choosing a random index to use as the pivot to avoid the chances of picking an extremity.

Binary insertion sort is an optimized version of insertion sort, utilizing binary search instead of moving it one by one. Through tests, I observed that binary insertion and bubble sort both have similar, if not the same, number of moves. However, binary insertion performs better than bubble sort due to the fact that it utilizes binary search, so it uses fewer comparisons, essentially cutting it in half. Although the comparisons is O(logn), since it is essentially cut by half using binary search, the overall average time complexity is still O(n^2), due to the fact that it is still swapping

items next to each other, which explains why the number of moves compared to bubble sort is similar.