Justin-Anthony Joco 0028483667
Programming Assignment #1 Report ECE 368

For this assignment, the goal was to sort a binary file with n long integers and outputting the sorted array into an output file. Instead of sorting the entire array at once, Shellsort breaks down the array into subarrays of differing lengths (each length is an element within a sequence calculated from array size). Each subarray is sorted through whatever sorting method is chosen; in this project, we are to use Insertion and Selection sort.

To facilitate this, a K sequence must be created. For increasing values of n, my algorithm generates the K sequence quickly, only increasing slightly for every increase in order of magnitude. However, due to the while and for loops, the time complexity must not be O(log(n)) but O((log(n))^2). Unfortunately, such a low time-complexity led to a high space complexity of O(n) due to the allocation of memory to store the array. Figure 1 below shows the data from the code's entire runtimes at increasing values of n.

Figure 1: Algorithm Output Information

Number of Integers	**I/O Run-Time (sec)**		**Number of Comparisons**		**Number of Moves**	
	Insertion	**Selection**	**Insertion**	**Selection**	**Insertion**	**Selection**
15	0s	0	75	45	45	123
1,000	0s	.03	3.09e4	3.64e04	1.36e4	4.26e04
10,000	0s	2.94	5.51e5	7.31e05	2.13e5	6.16e5
100,000	.6s	419	8.6e6	1.27e7	3.01e6	8.31e06
1,000,000	.83s	Long time	1.24e8	Unknown	4.04e7	Unknown

According to the table, the numbers of moves and comparisons of selection sort were higher than those of insertion because selection iterates over unsorted elements; whereas, insertion iterates over sorted elements. Thus, insertion compares and moves less than selection. As order of magnitude of n increased, number of comparisons and moves increased 16 times the amount before.

However, the run-times were significantly different: for increasing orders of magnitude of n, insertion sort run-time increased fairly little; whereas, the selection sort runtime increases exponentially due to inefficiency. I may have written many lines of code that could have accomplished the same task with one or two lines. Another way to optimize the selection was to use recursion, but recursion does not always guarantee a smaller runtime. Though selection is supposed have a time complexity of O(n^2), my algorithm seems to have much more than that. As a result, it would take days for my selection algorithm to solve the n=1,000,000 data set.

However, I may have made some errors in terms of keeping track of number of comparisons, moves, and memory consumption. I found that both sorts used very similar amounts of memory at about 15n; thus, O(n). This does not make sense since both sorts' runtimes are different; thus, their memory usage should also be. Thus, space complexity of selection should be about O(1) or O(logn).