1: For this project there were four algorithms, two to generate the sequences used and 2 that were used in sorting. For both sequences, I generated values iteratively. In the first, values are generated until the last integer in the sequence is the largest possible without exceeding the size of the array. The first three values are 1, 2, 3 respectively. Pointers are set at 2 and 3. Each successive integer is then determined by taking the smallest value obtained by multiplying the value at pointer 2 by 2 and the value at pointer 3 by three. After it is used to generate a value, the pointer is moved up a value in the sequence. In the second sequence, each value is determined by dividing the previous value by 1.3, discarding anything after the decimal point. The first value is obtained by using the array size. If the sequence value is 9 or 10, it is turned into 11. For the sorting algorithms, the shell sort was achieved by implementing a loop that iterates through each value in the sequence. This dictates the gap value to use for splitting the array into smaller sections. Inside this loop an insertion sort that sorts each of the sub-arrays. The bubble sort is achieved in a similar fashion, with an outer loop determining the gap values to use. Instead of an insertion sort, the arrays are sorted using a bubble sort with optimization. It has been optimized to detect when the sorted portion of the array has been reached so the number of comparisons are reduced.

2: For the first sequence, the time-complexity is O(n) and the space complexity is O(log(n)). Differentials in the time elapsed is dependent on the length of a single for loop that iterates from 1 to the array size. The memory required is dependent on the number of values in the sequence. Values in the sequence are selected by multiplying previous values by 2 or 3. The farther in the sequence, the greater the difference between adjacent values. The number of values used for sorting is determined by how many values are less than the size. As size grows larger, the rate at which memory required increases slows down.
For the second sequence, the time-complexity and the space complexity are both O(log(n)). The time-complexity is determined by a while loop that iterates through each sequence value until it is less than 1. The space-complexity is also dependent on the number of values in the sequence. Similar to the first sequence, each value is selected by dividing previous values by a factor. The number of values will grow slower than the array size. Since both time and space complexities are dependent on the size of the sequence, they will both be O(log(n)).
3.

Size:	10000		100000		1000000	
	Shell	Bubble	Shell	Bubble	Shell	Bubble
Elapsed time(sec)	0.000000	0.000000	0.020000	0.040000	0.340000	0.610000
# Comparisons:	615529	949298	484124	14113444	135697411	19772106
# Moves:	64848	62606	878713	816180	11710260	10079332

When increasing the size by factors of 10, the time elapsed increased dramatically. The smaller samples of 10000 quickly so that the registered time was 0.000000. For both larger samples, the time for the bubble sort was around two times the time for the shell sort. The number of comparisons was also larger for the bubble for all sizes, but the number of moves was smaller. The time, comparisons, and moves increase by a lot for each successive size and all increase at a greater rate than the size.

4: Without the sequence generation, the only memory required for both of the sorting algorithms is just for local variables. This memory is constant for any size array so without the sequence generation the space-complexity for both the shell and bubble sorts is O(1). As described in section 2, the space-complexity for both sequences is O(log(n)). This makes the overall space-complexity of both sorts O(log(n)).