ECE368 PA1 Report

Ben Jiang

My original plan for the sequence generation was to have them in two separate functions that would be called by the sorting functions and the sequence saving functions. However, during the programming process. I discovered more issues with this approach as it made the algorithms run slower and created difficulties with debugging. So instead, I incorporated them into the sorting and sequence saving functions. The sequence for shell sort was generated with an array with the first element set to 1 and then goes through a while loop that checks the consecutive numbers generated by multiplying the previous number by 2 and 3 to create a sequence with the proper order while eliminating the possibility for duplicated numbers. The sequence for bubble sort is easier with just a while loop that divides an array consecutively by 1.3 starting from the number of elements to be sorted till it reaches 1. The time complexity for both sequence generations is O(log(n)). The space complexity for both sequence generations is O(n) since in both cases an array of size n is created.

The shell sort algorithm has a for loop that goes through each gap number in the sequence, then goes through the list of numbers with another for loop while checking to see if the number has reached the proper location, if not then it would continually shift the elements within the sub array.

The bubble sort algorithm has a similar structure with a for loop going through the sequence of gap numbers and another for loop that goes through the numbers to be sorted. A while loop in the center checks for when the consecutive numbers needs to be swapped and performs the swap when needed. I was able to incorporate the swap comparison of the algorithm within the same while loop for the shell sort like modification of the algorithm. This optimization significantly reduced the runtime for sorting.

Table 1: runtime data

		1000	10000	100000	1000000
Shell sort	time (sec)	0.000000	0.010000	0.090000	0.700000
	comparisons	35266	615529	9484124	135697411
	moves	66221	1166240	18089535	259684562
Bubble sort	Time (sec)	0.000000	0.000000	0.030000	0.160000
	comparisons	24071	349335	4482925	55746098
	moves	13095	187818	2448540	30237996

For both algorithms the runtime and comparison/move count increase as the sample size increases. The time complexity is better than O(n^2) but worse than O(n).
The space complexity for both sorting algorithms is O(1). This is because the sorting routines don't require more memory than the amount already allocated for the arrays. They simply perform comparison and swaps on the arrays.