Geoffrey Cramer
ECE368 – Summer 2017

Programming Assignment 2

Results:

Shell Sort with inner Insertion Sort

Size	I/O Time	Sorting Time	Total Time
15	4.5e-4	2.9e-5	4.79e-4
1,000	8.72e-4	1.62e-3	2.49e-3
10,000	4.27e-3	2.57e-2	3.0e-2
100,000	2.6e-2	5.59e-1	5.85e-1
1,000,000	4.87e-1	2.316e1	2.36e1

Time Complexity:

It seems that the time complexity is $O(n)$ for all test cases except for 1,000,000. Changing from input size 100,000 to 1,000,000 increased the sorting time by a factor of 100 instead of 10 (resulting in $O(n^2)$ complexity). This change could be explained by the "sortedness" of the lists provided and also by the fact that insertion sort tends to be faster for "smaller" input sizes.

Space Complexity:

First, an array of k values must be generated. The maximum k value is $n-1$ and it was determined through observation that the size of the sequence is $O(n)$. This was determined by plotting the length of the sequence for each n.

For each value of k, a list structure is created (the size of which is k) and the original array is moved to the new structure for sorting purposes. Given that the maximum size of any k value is $n-1$, the space complexity for the list structure itself is $O(n)$.

In conclusion, the array of k values and the linked list of linked lists result in a total space complexity of $O(n)$.

Comparison with PA 1:

Comparing the time complexity with PA1, it can be seen that the I/O time using linked lists is significantly faster (even an order of magnitude faster for larger n values). This is likely due to the fact that the size of the input first needs to be determined before the array is allocated while the size isn't needed ahead of time for linked lists.

The time complexity of the sorting times is also vastly different. The PA1 implementation is significantly faster in every test case. This is likely due to an unoptimized sort but in theory, the array implementation should be faster as accesses are $O(1)$ for arrays but $O(n)$ for linked lists.