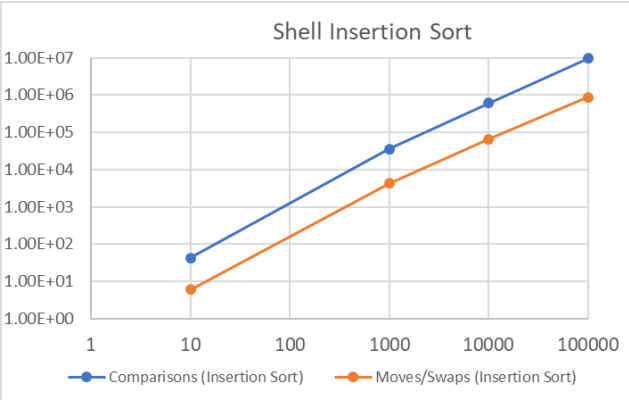
Yash Nain (ynain)
00270-62448

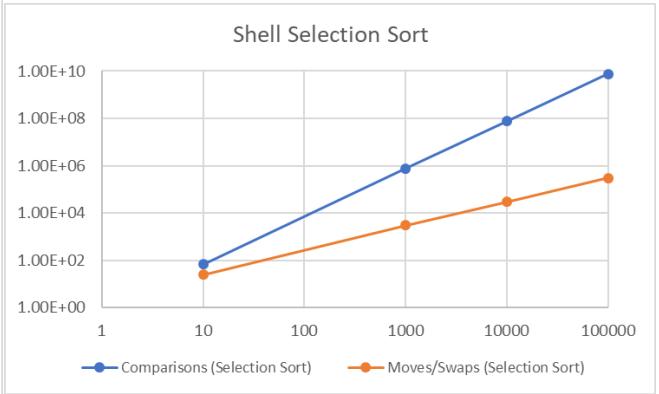
This program implements two shell sort algorithms, with one implementing an optimized insertion sort, while the other utilizes an improved selection sort. Firstly, a gap sequence is generated by the program based on the length of the array in order to subdivide the array. The sorting algorithm is applied to each subdivided array, then the gap sequence decrements to the next value, until the gap sequence = 1 and a normal sorting takes place.

1.) Firstly, the length of the gap sequence is calculated, and an appropriate amount of space is allocated into an array. Next, the gap sequence is generated in a series of for loops which "constructs" the pyramid of values by calculating the exponential constants of 2^p3^q, until the gap value exceeds the length of the array itself. The overall time complexity of the calculation ends up being c * O(n) time since the sequence does not diverge as the length of the array increases, and only increases by a factor of 2 as the size of the array itself increases by a factor of 10. Since the array will take a limited amount of space, and the size of the sequence is far smaller than the size of the array itself, the space complexity is O(log n).

2.) As the size of the array increases, the number of moves and comparisons increases with it exponentially. The best case time complexity, when the array is already in order and no swaps need to occur is O(n log n), while the worst-case time complexity is $O(n^2)$, where the array is in reverse order, and every element needs to be swapped. With each gap sequence, the time complexities are summed to $n^2/1 + n^2/2 + n^2/3 + n^2/4 + n^2/6 + \ldots + n^2/n$ to approximately = $3n^2/2$.
However, since by the last few gap sequences more like elements are closer together, the most any element will have to move will be one space, and the number of overall swaps is lessened drastically. For the selection sort algorithm, the overall time complexity is O(n^2), and is much more inefficient since every element in each subarray is incremented over more than it needs to be, by the nature of the algorithm itself.

	Shell Insertion Sort			**Shell Selection Sort**		
N	Comparisons	Moves/Swaps	Sorting Time	Comparisons	Moves/Swaps	Sorting Time
10	4.3E+1	6.0E+0	< 0.0E-6	6.7E+1	2.5E+1	< 0.0E-6
1000	3.52E+4	4.311E+3	< 0.0E-6	7.49E+5	3.0E+3	< 0.0E-6
10000	6.15E+5	6.48E+4	< 0.0E-6	7.50E+7	3.0E+4	2.1E-1
100000	9.48E+6	8.79E+5	2.0E-2	7.5E+9	3.0E+5	1.19E+1





3.) The sorting algorithm itself only requires enough space to hold the array, making it O(n). Besides the space required for the gap sequence, no other memory is used outside of temporary variables for swaps and comparisons, so the overall space complexity ends up being O(n).