PROJECT 1 REPORT

Name: Chul Woo Park
Homework ID: 4600081

1. Description of Algorithm

I implemented insertion sort algorithm in my shell sort. For the sequence of gap values, I called Generate_Seq1 function, which returns the array that consist gap values. However, the gap array is in ascending order so that I started from the end of it. There are 4 for-loops nested in my Shell Sort algorithm. The first for loop is for the gap selection from the gap array. The next for loop is for sub-array selection based on the selected gap value. Therefore, this for loop will iterate the number of sub-arrays. The next for loop is for sub-array element selection. However, important note in this for loop is that it starts from the second element in sub-array. The reason is that Insertion sort algorithm assumes that the first element is sorted and insert the rest of the elements to the left part in the exact place. Then, it stores the current element in temporary variable and runs another for loop, which searches where to insert that element in the left part of the array (sorted portion). I implemented bubble sort algorithm by using comb sort with gap sequence of N/1.3. Same as my shell sort, I called Generate_Seq2 function, which returns the array of gap values. I nested two for loops in this algorithm. The first for loop is for the gap selection from the gap array. The inner for loop compares the neighbor elements and swap if needed. Important note here is that it only goes through the entire array just once, which is quite different from original bubble sort. However, those iterations are done certain gap sequence. Therefore, at this point, the array would be roughly sorted, but not perfectly. Therefore, it runs another for loop until no moves occur. Existence of moves is indicated by the assertion of repeat flag. In every iteration, the last element gets excluded from getting compared for the next iteration because the last element gets sorted for sure at the end of the iteration. These are my brief description of my two sorting algorithms and these information is heavily commented in the code.

2. Complexity Analysis of Sequence Generation Algorithm

Since generating sequence_1 and sequence_2 only malloc once with sizeof(int) * N for seqarray, I would say that space complexity for both is [n]. For the time complexity of sequence_1, it would be big-O[n] because it only has one while-loop. For sequence_2, the time complexity would be (log_3)^2. The reason is that N is divided by 1.3 repeatedly and that gives such behavior.

3. Analysis of Shell Sort and Improved Bubble Sort

	# of elements	Shell Sort (Insertion)	Improved Bubble Sort
Comparisons	100,000	10253395	3966739
	1,000,000	146182457	49666756
Moves	100,000	18089535	2438928
	1,000,000	259684562	30144183
Time	100,000	0.13 sec	0.04 sec
	1,000,000	2.14 sec	0.47 sec

It seems like comparisons, moves, and time increase more rapidly as the input size increases. The reason is that when input size was 1,000 and 10,000 run time for both sort algorithm took less than 0.1 seconds, but when it gets to million, it seems to take much more time than that. In addition, I would say that my improved bubble sort is better in efficiency than shell sort. The reason is that it has less comparisons, less moves, and even much less run-time. My shell sort has a time complexity around big-O[n^4] and improved bubble sort has a time complexity of around big-O[n^2], which makes sense why my improved bubble sort is faster in run-time.

4. Space Complexity Summary

For both of my sort routines, the space complexity is n. The reason is that I only allocated memory in heap for the initial array that contains the elements. In addition, I have allocated memory for my seqarray, but that is also n, which still makes the space complexity of n.