Report by Zhakypbek Adil

Seq1gen:

To generate 2^p 3^q sequence I started with the first value in the sequence array to be equal to 1. Then I run a for loop, where I would multiply previous value in the sequence by 2 or 3 and whichever is less went as the next value in the array.

Seq2gen:

I made a do/while loop until N reaches 1, which divided Size by 1.3 and added the result to the sequence's array. I also implemented an if statement to work with Ns that are equal to 9 or 10.

Shell_Insertion_Sort:

Called seq1gen to generate correct sequence. Run in a loop, changing the gap size. Worst case Time complexity is O(n*(log(n))^2)

Improved_Bubble_Sort:

Called seq2gen to generate correct sequence. Run in a loop from both ends to increase efficiency. Time complexity is O(n * log(n)). Worst case O(n^2)

	Size	Time (seconds)	Comparisons	Moves
Shell Insertion Sort	1000	0	249115	252083
Improved Bubble Sort		0	4361	13083
Shell Insertion Sort	10000	0.28	24806134	25476076
Improved Bubble Sort		0.01	62626	187878
Shell Insertion Sort	100000	26.12	2495504627	2505604526
Improved Bubble Sort		0.08	816427	2449281

Conclusion:

As we can see the time tends to increase as the Size increases. Number of Comparisons and Moves also increase, which is how it supposed to be.

Additionally, for when I allocated memory for my sequences I allocated (Size*int) by default. To improve space efficiency, it is possible to use realloc. However, it wasn't specified, so I ignored that. I also included helper functions: seq1gen, seq2gen and swap to avoid dry coding and increase readability of the code.