1.
The shell sort algorthm uses the conecpt of gaps to sort the array across multiple gaps with each pass. The gap size keeps decreasing with each pass till it reaches 1 and every element is sorted. First I generate the sequence for the different gaps. Then I pass through the gaps in descending order and complete an insertion sort for each gap.

The improved bubble sort also uses the concept of gaps, but instead performs a bubble sort on each gap instead of an insertion sort. It also generates the sequence in the sorting algorithm itself than seperately. The concept of no swaps is used to terminate the algorithm without necessarily going through all the steps once the array is sorted.

2.
Depending on the meaning we assign to N, the time and space complexity varies for each sequence generation algorithm.
If N denotes the number of elements in the array to be sorted, the orders are as follows:
Sequence 1:
Space-Complexity = O((log(N) ^ 2)) [Max (Log Base 2 of N * Log Base 3 Of N) Elements in Sequence less than N]
Time Complexity = O((log(N) ^ 2)) [O(1) time to generate each element in Sequence]

Sequence 2:
Space-Complexity = O(log N) [Max (Log Base 1.3 of N) Elements in Sequence greater than 1]
Time Complexity = O(log N) [O(1) time to generate each element in Sequence]

If N denotes the number of elements in the sequence, the orders are as follows:
Sequence 1:
Space-Complexity = O(N) [N elements in Sequence]
Time Complexity = O(N) [O(1) time to generate each element in Sequence]

Sequence 2:
Space-Complexity = O(N) [N elements in Sequence]
Time Complexity = O(N) [O(1) time to generate each element in Sequence]

3.
Shell Sort
N:1000   Comparisions:4311   Moves:66221   TimeTaken:0.000424
N:10000   Comparisions:64818   Moves:1166240   TimeTaken:0.006302
N:100000   Comparisions:878713   Moves:18089535   TimeTaken:0.078522
N:1000000 Comparisions:11710260 Moves:259684562 TimeTaken:1.158119

The number of Comparisions increases by about 13 times for an increase of 10 times in the # of elements. This suggests that the number of Comparisions can be upper bounded by O(N*log(N)).
The number of Moves increases by about 15 times for an increase of 10 times in the # of elements. This suggests that the number of Moves can be upper bounded by O(N*log(N)*log(N)).
The number of Moves increases by about 15 times for an increase of 10 times in the # of elements. This suggests that the Time taken can be upper bounded by O(N*log(N)*log(N)).

Bubble Sort
N:1000   Comparisions:18713   Moves:13497   TimeTaken:0.000223
N:10000   Comparisions:276739   Moves:189927   TimeTaken:0.003066
N:100000   Comparisions:3666745   Moves:2437695   TimeTaken:0.029284
N:1000000 Comparisions:45666766 Moves:30132165   TimeTaken:0.359441

The number of Comparisions increases by about 13 times for an increase of 10 times in the # of elements. This suggests that the number of Comparisions can be upper bounded by O(N*log(N)).
The number of Moves increases by about 12 times for an increase of 10 times in the # of elements. This suggests that the number of Moves can be upper bounded by O(N*log(N)).
The number of Moves increases by about 10 times for an increase of 10 times in the # of elements. This suggests that the Time Taken can be upper bounded by O(N).

4.
For Shell Sort, we only need to allocate memory to hold the sequence, however, the sequence can be generated in the sorting algorithm itself than separetly. Therefore additional space complexity is O(1).
For Improved Bubble Sort, similarly the additional space complexity is O(1).