1. Algorithms

(1) Load_File: Read the first integer in the file as the size of the data (N), allocate an array with N spaces, and use a while loop to copy the data from the file into the array.

(2) Save_File: Use for loop to write N and all the elements in the array to the file.

(3) Shell_Insertion_Sort:

a. Malloc an array for the sequence 1 with N spaces, which is sequence 1 (seq1[ ]).

b.Use a for loop to create sequence 1: If seq1[P2]*2 or seq1[P3]*3 equals to the last element stored in seq1[ ], then increase P2 or P3 by 1. Compare seq1[P2]*2 and seq1[P3]*3. If the larger one is less than N, save the larger one as next item in the array and increase the corresponding index. Otherwise, stop the loop.

c. Use a for loop for each gap k from the sequence 1 (from higher k to lower k): For each N/k groups, use an inner loop to insertion sort the first element of each group, the second element of each group, and so on until the last element of each group.

(4) Improved_Bubble_Sort :

a. Use a while loop to count the size of the sequence (gap_n) and malloc an array for the sequence 2 with gap_n spaces, which is seq2[ ].

b. Use a for loop to create sequence 2: Let the gap equals to the floor number of the previous gap divided by 1.3 and put each of them into the array from the end of the array to the head of the array. Whenever the gap equals to 9 or 10, replace it by 11.

c. Use a for loop for each gap k from the sequence 2 (from higher k to lower k): For each N/k groups, use bubble sort the first element of each group, and then the second element of each group, and so on. Until the last element of each group.

2. Time and space-complexity of the algorithms to generate the two sequences:

(1) Time complexity to generate sequence 1 is O((p+1)*(q+1)+1): If (2^p)*(3^q) is the largest integer smaller than N, then the sequencr will have (p+1)*(q+1)+1.

(2) Time complexity to generate sequence 2 is O(log1.3(N)), at which 1.3 is the base of the log.

Size of data	1000000	
Type of Sequence	Sequence1	Sequence2
Run-time(second)	0.001	0
Space complexity	O(N)	O(log1.3(N))

3. Runtime, number of comparisons, and number of moves of sorting routines:For shell insertion sort, when N grows by 10 times, its run-time grows by 200 times, and iits number of comparisons and number of moves grow by 100 times. For improved bubble sort, when N grows by 10 times, its run-time grows by 150 times, its number of comparisons grows by 100 times, and its number of moves grows by about 15 times. Thus, improved bubble sort can decrease the number of moves, but at the same time increase the number of comparisons.

Size of data	1000		10000		100000	
Type of Sorting	Shell_Insertion_Sort	Improved_Bubble_Sort	Shell_Insertion_Sort	Improved_Bubble_Sort	Shell_Insertion_Sort	Improved_Bubble_Sort
Run-time(second)	0.004	0.003	0.279	0.416	47.073	62.781
number of comparisons	749,139	1,367,680	74,991,211	139,830,699	7,499,905,978	13,678,115,376
number of moves	454,707	16,275	45,633,744	243,360	4,579,016,736	3,205,941

4. Space complexity of sorting routines: Since the data are sorted in place, the extra memory is only allocated for sequences.

Type of Sorting	Shell_Insertion_Sort	Improved_Bubble_Sort
Space complexity	O(N)	O(log1.3(N))