Algorithms

For the shell insertion sort, first I calculate the smooth number. Then use the smooth number, cut the array to several pieces and pick the nth number from each and do the insertion sort. Finally, return the sorted array.

For the improve sort, I also calculate the sequence number at first. Then use the number in the sequence to cut the array. Instead of the insertion sort. I do the cocktail sort to the nth number of each piece. Finally, return the sorted array.

Structure

For the shell insertion sort, first use a while loop to create a smooth number array have Size of items. Then, I use another while loop to find the biggest number that smaller than Size. For sorting part, I use one for loop to cut the array, inside this loop I build another for loop to keep the number move in each piece of array and a while loop to compare and move the number in the array.

For the improved bubble sort, I create a while loop to control the sequence number and update itself. Then inside it, a for loop to move the number in each piece and while loop to cut and get out the number I need to sort in each turn. After this, a basic cocktail in a while loop to sort the number I picked out. Finally, another while loop to put those sorted number back into the array.

Optimization

For the shell insertion sort, the optimization is not so obviously, since I have nothing to compare in my code. However, it seems work better that simple bubble sort or insertion sort in move and compare times. Space and time complexity is better in some gap sequence.

For the improved bubble sort, it has much smaller move time than the shell insertion sort. This make it working time also better than shell insertion.

Seq1: Time complexity: O(n) Space complexity: O(n) //n is the amount of number in the seq1

Seq2: Time complexity: O(1) Space complexity: O(1)

	1000	10000	100000	1000	10000	100000
run-time	0	0	0	0	0	0
Ncomp	34384	605405	9374603	42928	667073	10024448
Nmove	66146	1166089	18089353	13449	189948	2447484

For the shell insertion sort (the first part): run-time can't show since too small, the number of compare increase in linear(N = kx), number of movement is also increase in linear(N = kx).

For the improved bubble sort (the second part): run-time also can't show. The number of compare increase as N = x + ky. The number of movement increase as N = x +ky.

The complexity of my sorting routines is O(n). //n is the Size number. I create two n size array.