Mitch Bouma
ECE 368 Project 1 Report
Compile with: gcc -Wall -Werror -O3 -lm sorting.c sorting_main.c -o proj1

Description of Algorithms

My Shell_Insertion_Sort works by first generating the array of gaps. To do so, it uses a nested loop incrementing p and q by one each time through. 2^q * 3^p is calculated for each q and p until the product is greater than the size of the array. When the number of total gaps is found, an array is malloc'd with that number, and the same double loop runs to input the products into that array. Now that the gap array is generated, qsort is run on it to set it in descending order. Using this gay array, insertion sort is performed on each gap until a gap of one which just becomes a regular insertion sort.

My Improved_Bubble_Sort works in a similar way to the shell sort by first generating the gap sequences. This time, it sets gap to size/1.3 and keeps dividing by 1.3 again for each successive gap. Each division is floor'd to ensure that they are integers. Also, as specified, a gap of 9 or 10 is converted to 11, and anything less than 1 is converted to 1. Then these gaps are used in a modified bubble sort (comb sort) until a gap of 1 where it runs until it is fully sorted. One drawback to this method is that I could have sorted each gap as they were calculated, therefore eliminating the need for the array. However, both methods work.

Analysis of Time/Space Complexity of Sequence Generation

Space complexity of the shell gap sequence is O(log(N)^2). This is because the size of the array is determined by how many times N can be divided by 2 (call this Q) and summing 1+2+3…Q. This series has a sum of (n*(n+1))/2 which simplifies to O(log(N)^2). The time complexity would be O(N*log(N)^2) from the nested loops.

Bubble sort's space complexity is O(log(N)) as the size of the array is the number of times 1.3 is multiplied by itself before it is greater than N. Time complexity is O(log(N)) because the loop will only iterate as many times as the size of the sequence.

Algorithm Comparisons

The improved bubble sort was much faster than the shell sort. The number of moves and comparisons for bubble sort only increased by a factor of 1 for each magnitude of 1 of the input. Shell sort moves and comparisons increased much faster.

Algorithm	Input Size	Run Time (s)	Moves	Comparisons
Shell	1,000	0.00	66,221	30,955
Bubble	1,000	0.00	13,197	21,704
Shell	10,000	0.00	1,166,240	550,711
Bubble	10,000	0.00	186,408	306,727
Shell	100,000	0.05	18,089,535	8,605,411
Bubble	100,000	0.02	2,438,928	3,966,742
Shell	1,000,000	0.62	259,684,562	123,987,151
Bubble	1,000,000	0.20	30,144,183	49,666,762

Analysis of Space Complexity of Sorting Algorithms

My bubble sort's space complexity is also O(log(N)) because I generate the gap array in the sorting function. However, no temp arrays are required.

My shell sort's space complexity is O(log(N)^2). This is because I again generate the entire array of gaps in the sorting function itself. No temp arrays are required.