Zach Stephens

ECE 368

Project 1

In Project 1, a shell sort algorithm was used to show the improvement in performance of insertion sort. Shell sort was applied to a bubble sort algorithm. The required functions included: file I/O functions, the traditional shell sort, a bubble sort with a shell component, and functions used for saving the sequences used in sorting. Two additional functions I used to carry out these tasks were a sequence generating function I used for the shell sort, named my_pq, and a simple power function I used in place of math.h 's pow() function named my_pow. Other than the required optimization of utilizing shell sort to improve the bubble sort, I did not make any other improvements to the algorithm.

In generating sequence 1 I used iteration to find the gaps. For the iteration, the time complexity is O(the summation as j goes from 0 to $\log_2(N)$ of 2^($\log_2(N)$-j). After finding all possible gaps, I used a simple bubble sort (O(N)) and then filtered for the indices that were greater than N. Noting that the combinations where logarithmic, I allocated an array of 10,000 integers.

Generating Sequence 2 was implicit in the bubble sort algorithm. No extra memory was used other than the file used to write $\log_{1.3}N$ values to a file. With the space complexity O(1) and time complexity O($\log_{1.3}N$).

Insertion sort (shell sort) N= 10000
Elapsed Time (sec): 0.01 # Comparisons: 1672446 # Moves: 10000
Sorting by Improved Bubble Sort with Sequence XXXXX N=10000
Elapsed Time (sec): 0.37 # Comparisons: 147968597 # Moves: 14234394
Insertion sort (shell sort) N= 100000
Elapsed Time (sec): 0.07 # Comparisons: 27465864 # Moves: 879291.000000
Sorting by Improved Bubble Sort with Sequence XXXXX N=100000
Elapsed Time (sec): 25.55 # Comparisons: 14854308889 # Moves: 0.000000
Insertion Sort(shell sort) N=1,000,000
Elapsed Time (sec): 1.69 # Comparisons: 39415666 # Moves: 11709852

Even with the optimizations used for the bubble sort, values of time, comparisons and moves are orders of magnitudes larger than the shell sort's. When N=1,000,000 the time for the bubble sort was too large for values to be rendered. Comparing to previous rates of growth the time estimated for N= 1000000 was ~45minutes.

Both the shell and bubble sorts are in place and only require constant amount of memory for variables. The bubble sort's gaps are implicitly calculated and therefore require no extra space. I made use of an auxillary function (my_pq) that requires space for 10,000 integers. This is the only extra space used in the sorting algorithms. This static space was used noting that the project goal was to examine the sorting algorithms themselves and under the assumption fewer than 10,000 different sized gaps would be needed for the shell sort .