Max Chi
ECE 368 Project 1 Milestone

The purpose of project 1 for ECE 368 is to understand and implement Shell Sort in order to improve the functionality and performance of sorting functions such as Insertion and Bubble sort. The use of the following functions will be necessary in successfully completing this assignment: Load_File and Save_File will be used not to sort but rather transfer the items that will be sorted. When the Load_File function is called, the function will read the first line as the number of integers that require sorting and the remaining lines as the integers that will be sorted. The sorting process will begin with the use of the Shell_Insertion_Sort function in that the program will have the integers sorted in ascending order within and Array. I have not yet thought of a way to implement this shell sort function to help improve Bubble sort, but based of notes and lectures slides, Shell Sort could be implemented into Bubble sort by limiting the need for the algorithm to traverse through an array each time it completes 1 iteration of a swap. Since by the end of a bunch of swaps, usually toward the end, the values have usually been sorted into clumped groups, making there no need to re-compare the values and then proceed to swap them. This should ideally cut Bubble Sort's comparisons in half, making it twice as fast as the original. Finally, the Save_File function will return the newly sorted integers back into the same format as it was scanned in as.

There is always the potential for larger and larger unsorted items to appear. The larger the problem, the more iterations, comparisons and moves sorting functions will have. The most ideal sorting function, not matter what the problem size is, would be able to easily sort any problem by using the least amount of space as well as having minimal amounts of comparisons, and being used primarily to swapping the values. The hopes of the "improved" Bubble Sort is to find a balance that doesn't sacrifice much time or space for a sorting function to be accurate.

When evaluating any sorting algorithm, it is important to take into account the amount of time the algorithm requires to run for a certain amount of unsorted values, as well as the amount of space that the algorithm requires to successfully complete the sorting process. Space-complexity wise, Bubble Sort is actually very simple in the fact that there is only really a need for an extra variable to temporarily hold the values of a variable. So there is no need to improve that aspect.

As for the actual time comparison, Bubble Sort is an algorithm that typically has its run time fall into the quadratic range since its time is N*N or O(N^2). N^2 being the worst case scenario where all the values are sorted in reverse order before actual sorting begins. My attempt to cut the comparisons in half should have the "improved" version of Bubble Sort's run time to be around N*(N+1)/2. This will effectively cut down sorting time for Bubble Sort, but still the algorithm will have its run time remain quadratic.