Din Bostandzic
CSE 13S

Assignment 5 - Sorting Program Writeup

The results obtained in this lab were based on the implementations of the binary insertion sort, quick sort, shell sort, and bubble sort. Each sort performed on the array respective sorts and ordered the data in ascending order, producing the same outputs, however where they varied lies in the counters of swaps and comparisons being made on the elements in the array. These stats are helpful as they are indicative of what actually goes on “under the hood” and how efficient and thus fast each of these sorts are. The binary insertion sort has worst case time complexity of O(nlogn) which shows it is improved based on just an insertion sort, as the compares are reduced. However looking at the data, this seems to be one of the worst ones when it comes to the number of swaps being made as it has the same amount of swaps as a bubble sort would make, which is 769875 swaps on an array of 1000 items. Yet it tops any of the sorts performed when it comes to the number of comparisons done, as it has the lowest comparisons made out of all.

The worst case time complexity of quick sort is O(n^2), which occurs when the pivot point is picked as first or last index so that the sort has to traverse the entire array. This would become very intense as with bubble sort, which has worst case time complexity of O(n^2) when the array is hundreds of thousands of elements, meaning that it would have to go all the way to length of array - 1 to sort that value to beginning. However, quick sort in terms of swaps made has the least amount of all the sorts, due to its recursive nature. The bubble sort is overall way far the worst with the greatest amount of swaps and comparisons being made (besides shell sort for numbers of comparisons). Finally the shell sort has the worst time complexity of O(n^5/3). This is highly dependent and varies on the gap intervals produced by the gap equation, so if less gaps are possible then the sort can be faster. This sort has the most amount of comparisons out of all the sorts.

I learned that all these sorting algorithms achieve the same end goal of sorting the arrays but should be used depending on what the size of input is and program specifications to get the right one for the right task, increasing efficiency to the maximum. If the number of inputs is less than a few thousand then the change is not as drastic and any of these algorithms would be fine to use as sorting, but as the input increases to millions, this is where the algorithm choice is significant. So using bubble sort would not be the appropriate case if the array is completely unsorted, instead a binary insertion sort or a shell sort with low enough gaps would be used. Or a quick sort as it does not require additional memory to be used on arrays, if an efficient pivot is picked to prevent the worst case. Each sort was experimented by changing around the inputs and total arrays sizes and then comparing each to see what the number of swaps and compares are and how they all match up to one another in terms of efficiency.

Time complexity of shell sort and binary insertion sort taken from notes on sorting lecture.