Project Summary

1. This project asked us to find code 4 different sorting algorithms, find out how long each one took, how many moves they made and how many comparisons on average they each took. To test all the methods, we created a driver class that used a switch statement to pick one of the sorts and inheritance to connect all the classes together. The input was simply a letter that corresponded to the sort method and a number 1 - 6. The driver class from there created five test cases for sizes 10^1 - 10^n (the number inputed). It then puts the each array in the sort class selected and finds how long, how many comparisons, and moves the sort makes while sorting it into an increasing array.
2. For this project, to connect all the classes, an interface was implemented in each of the sorting methods. It also asked us to test and verify four different algorithms, insertion, merge, heap, and quick sort. Insertion sort was the simplest, and straight forward to code. In the sort method we had to implement, I used a for loop and while loop to look through each element and find the largest element. For merge sort, it recursively broke the array into smaller ‘arrays’(it really only changed indexes) and then once they were individual elements, compared the elements and sorted them into a new array. After it finishes sorting it, it copies it back into the old array. Heap sort, takes in an array, reforms it into a max heap by taking the last element and putting it at the top(the top element switches with it), and sifts it down. Once the max heap is found, the algorithm is supposed to take off the first element(the top one), put it into a new array at the very back, ‘deletes’ it from the old array, and then reorganize the heap by again taking the last element, putting it at the top, sifting it down until it becomes max heap again. Quick sort does like the opposite of merge sort. It finds a random element(for mine it finds the middle element of the unsorted array, the pivot) and then sorts them by whether or not they are bigger or smaller than the pivot. The pivot then switches places with the right most element so all the elements can be looked at in a row without getting in the way. When it looks at a new element, if it is smaller than the pivot, it swaps places with the element that is designated to keep the boundary, the first element that was bigger than the pivot. If the new element is larger than the pivot, it stays in its place. At the end, the pivot then swaps with the element separating the two sets of numbers and recursively calls the right and left side of the pivot until sorted.
3. All the different sorts had different classes. A class was dedicated to keeping track of all the moves, comparisons, and time elapsed of each array that was tested(one class mind you that was multiple times). The driver class called on all the other classes and the interface SortAlgorithm tied all the classes together so they were similar.
4. I tried following the given pseudocode and tested my classes individually several times to make sure all cases were covered. The general outline looked like the pseudocode, I knew I was on the right track. I also used the debugger for the first time to help me find mistakes for mergeSort.
5. I learned again how much time these projects take too late, but anyways… I learned how the code actually looks like for each algorithm. Like how heap actually works. I wasn't too clear about how it went about sorting itself even though I saw animations and stuff before, but now I get it. The same goes for merge and quick sort. Oh wait, this is supposed to be like the analysis part. Ok so, insertion sort, even though to the human mind is the most straight forward or easiest to see sort, takes the longest, moves and comparisons, because it has to go through the whole array several times. Quick sort and merge sort are cool because it can go pretty fast if its already sorted. Quick sort, in the best case would copy the elements from the array and then back without a lot of moves. Heap is very for ordering elements. Graphs and tables can be found in the graphs and tables document.