The first functions for saving and writing to the file are trivial file functions similar to many that I implemented in ECE264. Depending on the function you either print or scan in the first line using the size pointer variable, then you simply reference the size variable in a for loop while you either scan from or print to the new file. In load file you will have to allocate memory for the array and in save file you will have to free up that memory to avoid any leaks. I would create a function to generate the sequence 1 based upon the scanned size of the array and return sequence 1 in an array to be referenced for shell sort. This memory would be freed at the end of shell sort. Sequence 2 is simply dividing repeatedly so it can be taken care of at each iteration. For the shell sort function, I would just implement code similar to what has been discussed in the lectures. Using my sequence 1 array I could reference the index and add that to the position x amount of times (to make sure I can check the whole array and get many variables). Then I will repeat that with the initial index +1 until the entire array has been considered. After all comparisons and swaps have been made I can then move to the next value in sequence one and repeat what I mentioned above until the gap size is one, which will then simply perform insertion sort on the final array. For my optimization I would do something very similar, however I will not have an array to reference. I will calculate N for each step that I am I then I will add it to a position x times until I have gone the length of the array. Then you repeat for the next several indexes as mentioned above. Now you perform standard bubble sort on the operation by comparing the now “adjacent” items in the array and swapping them. I may use an optimized version of bubble sort to improve my time complexity slightly, however I will begin with standard insertion and bubble sort to avoid bugs and work my way to optimized versions from there.