Dan Suciu

Milestone 1

To approach this problem the first thing I will have to do is understand the gap sequencing for the shell sort. In the project documentation, we are told to use the Pratt Sequencing method, in which we have a sequence of gaps which are every combination of p and q from N to 1 for the value of the expression 2^p*3^q, where p and q start at zero and increase. This is the slowest growing gap sequencing method. I will find this gap sequence Seq1 with a nested for loop, where the outside loop controls p starting at 0, and the inside loop controls q starting at 0. The inside loop will evaluate 2^p*3^q and check to see if it is greater than N. If it is not, we will save it into Seq1, if it is we will exit the inside loop and increment p until 2^p*1 is greater than N. At the end of the function we will sort sequence so that our gaps are in order from greatest to least.

For the gap sequence of the bubble sort, Seq2, we will use a similar method. A loop will control the power of 1.3, starting at 1. Inside the loop we will take N and divide it by 1.3^power and take the floor of this whole expression, then saving it to Seq2 starting at index (counter = 0) as long as it is not equal to or greater than the last gap size to avoid duplicates and errors. As an extra condition, if the floor(N/(1.3^power)) is equal to 9 or 10, we will save 11 into Seq2 instead. The previous duplicate check will apply to these 11 values as well. The loop will stop when the floor(N/(1.3^power)) is equal to 1 for the first time, and then we will save 1 into the next spot in Seq2. These are already in order so there is no need to sort them.

In both our shell sort and our improved bubble sort we will use the standard method of starting from gap = Seq1[0] or Seq2[0] respectively, being the biggest gap, and moving up in indices until we have reached a gap of 1, being the last pass of the sort. Inside our control statements we will increment the value stored at *NComp and *NMove starting at 0 directly after every operation that fits the description in the project documentation. The outer control of these sorts will be very similar in this way. The inner control will be different however, as when we are looking at our values corresponding to each gap in the array, the shell sort will use the insertion method and the improved bubble sort will use the bubble sort method obviously.

In terms of using the gaps, we will start every set of values and the array index 0, and add gap to the index until the index is greater than Size, the last of which we will not include as it is out of bounds. Then we will do our sort operations on those indices, and when we have made our one pass, we will shift all indices 1 to the right and preform the same operations. This will stop once the greatest index equals Size, and then we will move on to the next gap.