Problem 1

1. The sorting algorithm I chose was mergeSort. Since it only moves switches values when they are smaller than the preceeding values, it maintains relative order:

**if**(arr[leftInd] <= arr[rightInd])

leftInd++; ←- if the value at the left index is smaller, it doesn’t swap any values

**while**(leftInd <= midInd && rightInd <= end) ←- the index where the smaller values should be in the sorted array never exceeds the index where the larger values should be.

2. The trivial best case of this algorithm is when the array only contains one element and the time complexity is O(1).

The best case of this mergeSort is NlogN where N is the number of elements in the array.

Since it recursively splits the array into two before sorting:

*mergeSort*(arr, start, midInd);

*mergeSort*(arr, midInd+1, end);

The time complexity of the splitting is logN. Within this, the loop requires the elements to shift N times:

System.*arraycopy*(arr, leftInd, arr, leftInd+1, rightInd-leftInd);

So the total time complexity is: NlogN.

3. The worst case time complexity of mergesort is the same as the average case—NlogN time. This is because even if the array is already sorted, the algorithm will still recursively check each half of the array.

*mergeSort*(arr, start, midInd);

*mergeSort*(arr, midInd+1, end);