Q1

(a) Insertion sort on a sorted file of size n will perform $n-1$ comparisons and 0 interchanges.

(b) Insertion sort on a reverse order file of size n will perform $n(n-1)/2$ comparisons and interchanges.

(c) Insertion sort on a file in which x[0], x[2], x[4]... are the smallest elements in sorted order, and in which x[1], x[3], x[5]... are the largest elements in sorted order will perform the following number of comparisons and interchanges.

$$i(n) = \frac{a(a+1)}{2} \quad \text{where} \quad a = \begin{cases} \frac{n-1}{2}, & n = \text{odd} \\ \frac{n-2}{2}, & n = \text{even} \end{cases}$$

$$c(n) = \frac{a(a+1)}{2} + (n-1) \quad \text{where} \quad a = \begin{cases} \frac{n-1}{2}, & n = \text{odd} \\ \frac{n-2}{2}, & n = \text{even} \end{cases}$$

These equations correctly produce the following table.

n	1	2	3	4	5	6	7	8	9	10	11	12
c	0	1	3	4	7	8	12	13	18	19	25	26
i	0	0	1	1	3	3	6	6	10	10	15	15

Q2

(a) Shell sort (gap 2 and 1) on a sorted file of size n will perform $2n-3$ comparisons and 0 interchanges.

(b) Shell sort (gap 2 and 1) on a reverse order file of size n will perform the following number of comparisons and interchanges. We will make extensive use of the formula for sum of consecutive integers, given by:

$$\text{sum} = \frac{n(n-1)}{2}$$

First, calculate the number of comparisons/interchanges at gap 2.

$$\text{even:} \qquad c(n) = i(n) = n(n-1)$$

$$\text{odd:} \qquad c(n) = i(n) = \frac{(\frac{n}{2} - \frac{1}{2})(\frac{n}{2} - \frac{3}{2})}{2} + \frac{(\frac{n}{2} + \frac{1}{2})(\frac{n}{2} - \frac{1}{2})}{2}$$

This results in two ordered lists that are then reassembled. Now, calculate the number of comparisons/interchanges at gap 1.

$$\text{even:} \quad c(n) = \frac{a(a+1)}{2} + (n-2) \quad \text{AND} \quad i(n) = \frac{a(a+1)}{2} \quad \text{where} \quad a = \frac{n-2}{2}$$

$$\text{odd:} \quad c(n) = n-1 \quad \text{AND} \quad i(n) = 0$$

Finally, sum the two parts to get:

even:

$$c(n) = n(n-1) + \frac{a(a+1)}{2} + (n-2) \quad \text{where} \quad a = \frac{n-2}{2}$$

$$i(n) = n(n-1) + \frac{a(a+1)}{2} \quad \text{where} \quad a = \frac{n-2}{2}$$

odd:

$$c(n) = \frac{(\frac{n}{2} - \frac{1}{2})(\frac{n}{2} - \frac{3}{2})}{2} + \frac{(\frac{n}{2} + \frac{1}{2})(\frac{n}{2} - \frac{1}{2})}{2} + (n-1)$$

$$i(n) = \frac{(\frac{n}{2} - \frac{1}{2})(\frac{n}{2} - \frac{3}{2})}{2} + \frac{(\frac{n}{2} + \frac{1}{2})(\frac{n}{2} - \frac{1}{2})}{2}$$

(c) Shell sort (gap 2 and 1) on a file in which x[0], x[2], x[4]... are the smallest elements in sorted order, and in which x[1], x[3], x[5]... are the largest elements in sorted order will perform the number of comparisons from standard insertion sort (Question 1c), plus two. The exact same number of interchanges are made as standard insertion sort.

Q3

(a) In this method, you compare the smallest value of each file and transfer the smaller value to the end of the new sorted file. In case a, a[0] < b[0] and is transferred to the sorted file. Next, a[1] and b[0] are compared. b[0] < a[1] and is transferred to the sorted file. This continues until one file is "empty", at which point the contents of the non-empty file are transferred to the end of the sorted file. In this way, there are $2m - 1$ or $2n - 1$ comparisons made, depending on whether m or n is smaller. Since $m = n$, the equation is simply $m + n - 1$.

(b) The same general principle applies here. Values are compared and then added to the sorted file until one file is empty, at which point the remainder of values from the non-empty file are added. In this case, all values in a are smaller than the smallest value in b; thus, only n comparisons are made before file a is "empty" and the contents of file b are added to the end of the sorted file.

Q4

(a) In this case, the first half of a will be compared and then moved to the sorted file. After this, each item in b will be compared to $a[\frac{n}{2} + 1]$ and then moved to the sorted file. At this point, b is "empty" and the remainder of a is added to the sorted list. Thus, the number of comparisons is $\frac{n}{2} + m$.

(b) In this case, only a single comparison is made. The sole element in b is compared to a, found to be less than a, and consequently moved to the sorted file. At this point, b is "empty" and all of a can be moved to the sorted file.

(c) In this case, a total of n comparisons are made. Each item in a is compared to b[1], found to be less than b[1], and consequently moved to the sorted file. At this point, a is "empty" and the sole item in b can be moved to the sorted file.

Q5

If no record with the key *Item* is present, sequential search will perform N comparisons if the list is unordered. The worst-case performance is O(n).

If the list is ordered, the best case will require a single comparison, the worst case will require N comparisons, and the average case will require $\frac{N}{2}$ comparisons. The worst-case performance is O(n).

Q6

If one record with the key *Item* is present and only one is sought, for an unordered list sequential search will perform a single comparison in the best case, N comparisons in the worst case, and $\frac{N}{2}$ comparisons in the average case. The worst-case performance is O(n).

If the list is ordered, the best case will require a single comparison, the worst case will require N comparisons, and the average case will require $\frac{N}{2}$ comparisons. The worst-case performance is O(n).

Q7

If more than one record with the key Item is present and it is desired to find only the first, for an unordered list sequential search will perform a single comparison in the best case, $N - m$ comparisons in the worst case, and $\frac{N}{2}$ comparisons in the average case. The worst-case performance is O(n).

If the list is ordered, the best case will require a single comparison, the worst case will require $N - m$ comparisons, and the average case will require $\frac{N}{2}$ comparisons. The worst-case performance is O(n).

Q8

If more than one record with the key Item is present and it is desired to find them all, for an unordered list sequential search will perform a m comparisons in the best case, $N - m$ comparisons in the worst case, and $\frac{N}{2}$ comparisons in the average case. The worst-case performance is O(n).

If the list is sorted, then any record with the same key as the first found will be located immediately after the first record. Thus, the performance is the same; that is, the best case will require a single comparison, the worst case will require $N - m$ comparisons, and the average case will require $\frac{N}{2}$ comparisons. The worst-case performance is O(n).

Q9

The size of the output[] array needs to be at minimum size n. No addtional space is required, but the minimum must be satisfied in order to store all initial values. A method for sorting could be as follows:

```
/* Java/pseudocode for alternative sort method */
public int[] altSort(int[] data) {
   size = data.length;
   int[] A = data;
   int[] count = new int[size];
   int[] output = new int[size];
   int counter;

   for ( int i = 0; i < size; i++ ) {  // iterate through items
      counter = 0;
      for ( int j = 0; j < size; j++ ) {  // iteratively compare items
         if ( A[j] < A[i] ) {
            counter++;  // increment if A[j] < A[i]
         }
      }
      count[i] = counter;  // number of items less than A[i]
   }
   for ( int i = 0; i < size; i++ ) {  // iteratively fill output array
      output[count[i]] = A[i];
   }
   return output;  // return output
}
```

Q10

In the sort method I wrote, the best-case performance and worst-case performance are the same - $\Omega(n^2)$ and $O(n^2)$. This is because the method utilizes a nested for-loop to iterate through each item in the array, and iteratively compare to each other item in the array. This performance will not vary with ordered/unordered/random/reverse data, because all iterations occur regardless.