$$\Theta(n \lg n + n \lg n - n \lg \lg n) = \Theta(2n \lg n - n \lg \lg n) ,$$

which, by taking just the high-order term and ignoring the constant coefficient, equals $\Theta(n \lg n)$.

d. In practice, k should be the largest list length on which insertion sort is faster than merge sort.

Solution to Problem 2-2

a. We need to show that the elements of A' form a permutation of the elements of A.

b. **Loop invariant:** At the start of each iteration of the **for** loop of lines 2–4, $A[j] = \min\{A[k] : j \le k \le n\}$ and the subarray $A[j \mathinner{..} n]$ is a permutation of the values that were in $A[j \mathinner{..} n]$ at the time that the loop started.

Initialization: Initially, $j = n$, and the subarray $A[j \mathinner{..} n]$ consists of single element $A[n]$. The loop invariant trivially holds.

Maintenance: Consider an iteration for a given value of j. By the loop invariant, $A[j]$ is the smallest value in $A[j \mathinner{..} n]$. Lines 3–4 exchange $A[j]$ and $A[j-1]$ if $A[j]$ is less than $A[j-1]$, and so $A[j-1]$ will be the smallest value in $A[j-1 \mathinner{..} n]$ afterward. Since the only change to the subarray $A[j-1 \mathinner{..} n]$ is this possible exchange, and the subarray $A[j \mathinner{..} n]$ is a permutation of the values that were in $A[j \mathinner{..} n]$ at the time that the loop started, we see that $A[j-1 \mathinner{..} n]$ is a permutation of the values that were in $A[j-1 \mathinner{..} n]$ at the time that the loop started. Decrementing j for the next iteration maintains the invariant.

Termination: The loop terminates when j reaches i. By the statement of the loop invariant, $A[i] = \min\{A[k] : i \le k \le n\}$ and $A[i \mathinner{..} n]$ is a permutation of the values that were in $A[i \mathinner{..} n]$ at the time that the loop started.

c. **Loop invariant:** At the start of each iteration of the **for** loop of lines 1–4, the subarray $A[1 \mathinner{..} i-1]$ consists of the $i-1$ smallest values originally in $A[1 \mathinner{..} n]$, in sorted order, and $A[i \mathinner{..} n]$ consists of the $n-i+1$ remaining values originally in $A[1 \mathinner{..} n]$.

Initialization: Before the first iteration of the loop, $i = 1$. The subarray $A[1 \mathinner{..} i-1]$ is empty, and so the loop invariant vacuously holds.

Maintenance: Consider an iteration for a given value of i. By the loop invariant, $A[1 \mathinner{..} i-1]$ consists of the i smallest values in $A[1 \mathinner{..} n]$, in sorted order. Part (b) showed that after executing the **for** loop of lines 2–4, $A[i]$ is the smallest value in $A[i \mathinner{..} n]$, and so $A[1 \mathinner{..} i]$ is now the i smallest values originally in $A[1 \mathinner{..} n]$, in sorted order. Moreover, since the **for** loop of lines 2–4 permutes $A[i \mathinner{..} n]$, the subarray $A[i+1 \mathinner{..} n]$ consists of the $n-i$ remaining values originally in $A[1 \mathinner{..} n]$.

Termination: The **for** loop of lines 1–4 terminates when $i = n$, so that $i - 1 = n - 1$. By the statement of the loop invariant, $A[1 \mathinner{..} i-1]$ is the subarray

$A[1\,..\,n-1]$, and it consists of the $n-1$ smallest values originally in $A[1\,..\,n]$, in sorted order. The remaining element must be the largest value in $A[1\,..\,n]$, and it is in $A[n]$. Therefore, the entire array $A[1\,..\,n]$ is sorted.

Note: Tn the second edition, the **for** loop of lines 1–4 had an upper bound of $A.\mathit{length}$. The last iteration of the outer **for** loop would then result in no iterations of the inner **for** loop of lines 1–4, but the termination argument would simplify: $A[1\,..\,i-1]$ would be the entire array $A[1\,..\,n]$, which, by the loop invariant, is sorted.

d. The running time depends on the number of iterations of the **for** loop of lines 2–4. For a given value of i, this loop makes $n-i$ iterations, and i takes on the values $1, 2, \ldots, n-1$. The total number of iterations, therefore, is

$$\begin{aligned}\sum_{i=1}^{n-1}(n-i) &= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \\ &= n(n-1) - \frac{n(n-1)}{2} \\ &= \frac{n(n-1)}{2} \\ &= \frac{n^2}{2} - \frac{n}{2} .\end{aligned}$$

Thus, the running time of bubblesort is $\Theta(n^2)$ in all cases. The worst-case running time is the same as that of insertion sort.

Solution to Problem 2-4
This solution is also posted publicly

a. The inversions are $(1,5)$, $(2,5)$, $(3,4)$, $(3,5)$, $(4,5)$. (Remember that inversions are specified by indices rather than by the values in the array.)

b. The array with elements from $\{1, 2, \ldots, n\}$ with the most inversions is $\langle n, n-1, n-2, \ldots, 2, 1\rangle$. For all $1 \le i < j \le n$, there is an inversion (i, j). The number of such inversions is $\binom{n}{2} = n(n-1)/2$.

c. Suppose that the array A starts out with an inversion (k, j). Then $k < j$ and $A[k] > A[j]$. At the time that the outer **for** loop of lines 1–8 sets $\mathit{key} = A[j]$, the value that started in $A[k]$ is still somewhere to the left of $A[j]$. That is, it's in $A[i]$, where $1 \le i < j$, and so the inversion has become (i, j). Some iteration of the **while** loop of lines 5–7 moves $A[i]$ one position to the right. Line 8 will eventually drop *key* to the left of this element, thus eliminating the inversion. Because line 5 moves only elements that are greater than *key*, it moves only elements that correspond to inversions. In other words, each iteration of the **while** loop of lines 5–7 corresponds to the elimination of one inversion.

d. We follow the hint and modify merge sort to count the number of inversions in $\Theta(n \lg n)$ time.