1 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, \( A[j] = \min\{A[k]: j \leq k \leq n\} \) and the subarray \( A[j..n] \) is a permutation of the values that were in \( A[j..n] \) at the time that the loop started.

Initialization: Initially, \( j = n \), and the subarray \( A[j..n] \) 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 \( A[j..n] \). 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 \( A[j-1..n] \) afterward. Since the only change to the subarray \( A[j-1..n] \) is this possible exchange, and the subarray \( A[j..n] \) is a permutation of the values that were in \( A[j..n] \) at the time that the loop started, we see that \( A[j-1..n] \) is a permutation of the values that were in \( A[j-1..n] \) 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, \( A[i] = \min\{A[k]: i \leq k \leq n\} \) and \( A[i..n] \) is a permutation of the values that were in \( A[i..n] \) at the time that that loop started.

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

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

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

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, \( A[1..i-1] \) 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.

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, ..., \( n-1 \). The total number of iterations, therefore, is \( \sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n(n-1) - \frac{n(n-1)}{2} = \frac{n^2 - n}{2} \). Thus, the running time of bubble sort is \( \Theta(n^2) \) in all cases. The worst-case running time is the same as that of insertion sort.
2 Solution to Exercise 3.1-3

Let the running time be \( T(n) \). \( T(n) \geq O(n^2) \) means that \( T(n) \geq f(n) \) for some function \( f(n) \) in the set \( O(n^2) \). This statement holds for any running time \( T(n) \), since the function \( g(n) = 0 \) for all \( n \) is in \( O(n^2) \), and running times are always nonnegative. Thus, the statement tells us nothing about the running time.

3 Solution to Exercise 3.1-4

\( 2^{n+1} = O(2^n) \), but \( 2^{2n} \neq O(2^n) \).

To show that \( 2^{n+1} = O(2^n) \), we must find constants \( c, n_0 > 0 \) such that
\[
0 \leq 2^{n+1} \leq c \cdot 2^n \text{ for all } n \geq n_0.
\]
Since \( 2^{n+1} = 2 \cdot 2^n \) for all \( n \), we can satisfy the definition with \( c = 2 \) and \( n_0 = 1 \).

To show that \( 2^{2n} \neq O(2^n) \), assume there exist constants \( c, n_0 > 0 \) such that
\[
0 \leq 2^{2n} \leq c \cdot 2^n \text{ for all } n \geq n_0.
\]
Then \( 2^{2n} = 2^n \cdot 2^n \leq c \cdot 2^n \Rightarrow 2^n \leq c \). But no constant is greater than all \( 2^n \), and so the assumption leads to a contradiction.