**LI: Insertion sort**

InsertionSort(A, n): // Sort A[1..n].

1: \textbf{for} \ j ← 2 \ \textbf{to} \ n \ \textbf{do}
2: \quad \textit{key} ← A[j]
3: \quad i ← j - 1
4: \quad \textbf{while} \ i > 0 \ \textbf{and} \ A[i] > key \ \textbf{do}
5: \quad \ A[i + 1] ← A[i]
6: \quad i ← i - 1
7: \quad A[i + 1] ← key

**Analysis of while loop:**

Precondition to the while loop: \( key = A[j], \ i = j - 1, \ \text{and}, \ A[1..j - 1] \) is sorted.

LI for the while loop: The elements of \( A[i + 1..j - 1] \) have shifted right one step, in the same order, now occupying \( A[i + 2..j] \). Also, \( key < A[i + 1..j - 1] \). \( A[1..i] \) is unaffected.

Initialization: When the loop is entered for the first time, \( i = j - 1 \), and \( A[j] \) has been copied to \( key \). So, the LI is true. Note that, for \( i = j - 1 \), \( A[i + 1..j - 1] = A[j..j - 1] = \emptyset \).

Maintenance: If the body of the while loop is executed, then, \( A[i] > key \). The algorithm copies \( A[i] \) into \( A[i + 1] \), and decrements \( i \). So, the LI is valid when the loop ends.

Termination: The while loop has 2 conjunctive conditions, so there are 2 cases to consider.

Case 1: The while loop ended because \( i = 0 \). In this case, from the LI, \( A[1..j - 1] \) have been shifted right to \( A[2..j] \), and \( key < A[2..j] \). The algorithm copies \( key \) to \( A[1] \). By the precondition, \( A[1..j - 1] \) was sorted when the loop was entered, and the elements have been retained in the same order. \( A[1] \) has the element that was at \( A[j] \) in the beginning of the loop, and it is smaller than all elements of \( A[2..j] \). Therefore \( A[1..j] \) is now in sorted order.

Case 2: The while loop ended because \( A[i] \leq key \). The subarray \( A[1..i] \) is in sorted order, and so \( A[i] \leq key \) implies that \( A[1..i] \leq key \). Also, we know that \( A[i + 2..j] > key \). The algorithm now copies the remaining element, \( key \) into \( A[i + 1] \), and the elements \( A[1..j] \) are now in sorted order.

In both cases, the while loop satisfies the postcondition that \( A[1..j] \) are in sorted order. The algorithm does not access \( A[j + 1..n] \) and so those elements just stay in their places.

**Analysis of for loop:**

Precondition to the for loop: none.

LI of the for loop: \( A[1..j - 1] \) is in sorted order, \( A[j..n] \) is unprocessed.

Initialization: At the start of the for loop, \( j = 2 \). The LI says \( A[1..1] \) is in sorted order, which is trivially true, since \( A[1..1] \) has only one element, \( A[1] \). Also, \( A[2..n] \) is unprocessed.

Maintenance: The LI of the for loop and the execution of the first two statements inside the loop, the Preconditions to the while loop are satisfied. From the proof above, the postcondition to the while loop is that \( A[1..j] \) is in sorted order, which satisfies the LI for the next iteration of the for loop.

Termination: At the end of the for loop, \( j = n + 1 \), and by the LI, \( A[1..n] \) is in sorted order, thus proving the correctness of the algorithm. Postcondition: \( A[1..n] \) is sorted.