Chapter 2 Getting Started

2.1 Insertion Sort

Sorting problem: Given a sequence of n numbers $(a_1, a_2, \ldots, a_n)$, produce a premutation $(a'_1, a'_2, \ldots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

We present a first solution to the sorting problem, the *insertion sort* algorithm, as a subroutine which takes as parameter an array $A[1..n]$ containing a sequence of length n to be sorted. When the procedure INSERTION-SORT is finished, it rearranges the elements within A so that they are sorted.

INSERTION-SORT(A)

```
1  for j = 2 to A.length do
2      key = A[j]
       // Insert A[j] into the sorted sequence A[1..j − 1]
3      i = j − 1
4      while i > 0 and A[i] > key do
5          A[i + 1] = A[i]
6          i = i − 1
7      end while
8      A[i + 1] = key
9  end for
```

Loop Invariants and Correctness: In order to prove for correctness of some algorithms, we can use a *loop invariant*, which is a property of the state of the algorithm which holds during all iterations of the loop. We must show three things in order to prove that a loop invariant holds:

- **Initialization:** It is true prior to the first iteratin of the loop.
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.
- **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

When the first two properties hold, we show that the loop invariant holds prior to every iteration of the loop. The third property helps us use the loop invariant to prove correctness (typically using the loop invariant along the condition which made the loop to terminate).

We can now give a loop invariant of the given INSERTION-SORT algorihtm: *At the start of each iteration of the* ***for*** *loop of lines 1 - 8 the subarray* $A[1..j-1]$ *consists of the elements originally in* $A[1..j-1]$*, but in sorted order.*

Let us see the loop invariant holds and see how it can prove correctness of the sorting algorithm.

- *Initialization:* Before the first iteration, at $j = 2$, the subarray is just $A[1]$ which trivially contains the elements of $A[1]$ in sorted order.
- *Maintenance:* Informally, the body of the **for** loop works by moving $A[j-1]$, $A[j-2]$, $A[j-3]$, and so on by one position to the right until it finds the proper position for $A[j]$. Which leaves the subarray $A[1..j]$ consisting in elements originally in $A[1..j]$ but in sorted order. Incrementing j for the next iteration of the loop preserves the loop invariant. (A more formal proof would require to prove another loop invariant for this **while** loop.)
- *Termination:* The loop terminates when $j > A.length = n$. Because each iteration increments j by one, we must have $j = n + 1$ at that time. Since the loop invariant holds at the termination time, we see that the array $A[1..n]$ consists of the elements originally in $A[1..n]$ but in sorted order. Hence the entire array A is sorted at the end, hence the algorithm is correct.