Special Topic 14.1

Insertion Sort

Insertion sort is another simple sorting algorithm. In this algorithm, we assume that the initial sequence

```
a[0] a[1] . . . a[k]
```

of an array is already sorted. (When the algorithm starts, we set `k` to `0`.) We enlarge the initial sequence by inserting the next array element, `a[k + 1]`, at the proper location. When we reach the end of the array, the sorting process is complete.

For example, suppose we start with the array

11	9	16	5	7

Of course, the initial sequence of length 1 is already sorted. We now add `a[1]`, which has the value 9. The element needs to be inserted before the element 11. The result is

9	11	16	5	7

Next, we add `a[2]`, which has the value 16. As it happens, the element does not have to be moved.

9	11	16	5	7

We repeat the process, inserting `a[3]` or 5 at the very beginning of the initial sequence.

5	9	11	16	7

Finally, `a[4]` or 7 is inserted in its correct position, and the sorting is completed.

The following class implements the insertion sort algorithm:

```
public class InsertionSorter
{
   private int[] a;

   /**
      Constructs an insertion sorter.
      @param anArray the array to sort
   */
   public InsertionSorter(int[] anArray)
   {
      a = anArray;
   }

   /**
      Sorts the array managed by this insertion sorter.
   */
   public void sort()
   {
      for (int i = 1; i < a.length; i++)
      {
         int next = a[i];
         // Find the insertion location
         // Move all larger elements up
         int j = i;
         while (j > 0 && a[j - 1] > next)
         {
            a[j] = a[j - 1];
            j--;
         }
         // Insert the element
         a[j] = next;
      }
   }
}
```

How efficient is this algorithm? Let n denote the size of the array. We carry out n – 1 iterations. In the kth iteration, we have a sequence of k elements that is already sorted, and we need to insert a new element into the sequence. For each insertion, we need to visit the elements of the initial sequence until we have found the location in which the new element can be inserted. Then we need to move up the remaining elements of the sequence. Thus, $k + 1$ array elements are visited. Therefore, the total number of visits is

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

Insertion sort is an $O(n^2)$ algorithm.

We conclude that insertion sort is an $O(n^2)$ algorithm, on the same order of efficiency as selection sort.

Insertion sort has a desirable property: Its performance is $O(n)$ if the array is already sorted—see Exercise R14.13. This is a useful property in practical applications, in which data sets are often partially sorted.