图 7-1 表达了一般的策略。在第 p 躺，我们将位置 p 上的元素向左移动，直到它在前 $p+1$ 个元素中的正确位置被找到的地方。图 7-2 中的程序实现这种策略。第 12 行到第 15 行实现数据移动而没有明显地使用交换。位置 p 上的元素储存于 tmp，而(在位置 p 之前)所有更大的元素都被向右移动一个位置。然后 tmp 被置于正确的位置上。这是与在二叉堆实现时所用到的相同技巧。

```
1   /**
2    * Simple insertion sort.
3    * @param a an array of Comparable items.
4    */
5   public static <AnyType extends Comparable<? super AnyType>>
6   void insertionSort( AnyType [ ] a )
7   {
8       int j;
9
10      for( int p = 1; p < a.length; p++ )
11      {
12          AnyType tmp = a[ p ];
13          for( j = p; j > 0 && tmp.compareTo( a[ j - 1 ] ) < 0; j-- )
14              a[ j ] = a[ j - 1 ];
15          a[ j ] = tmp;
16      }
17  }
```

图 7-2 插入排序例程

7.2.2 插入排序的分析

由于嵌套循环的每一个都花费 N 次迭代，因此插入排序为 $O(N^2)$，而且这个界是精确的，因为以反序的输入可以达到该界。精确计算指出，图 7-2 内循环中元素的比较次数对于 p 的每个值最多是 $p+1$ 次。对所有的 p 求和得到总数为

$$\sum_{i=2}^{N} i = 2 + 3 + 4 + \cdots + N = \Theta(N^2)$$

另一方面，如果输入数据已预先排序，那么运行时间为 $O(N)$，因为内层 for 循环的检测总是立即判定不成立而终止。事实上，如果输入几乎被排序(该术语将在下一节更严格地定义)，那么插入排序将运行得很快。由于这种变化差别很大，因此值得我们去分析该算法平均情形的行为。实际上，和各种其他排序算法一样，插入排序的平均情形也是 $\Theta(N^2)$，详见下节的分析。

7.3 一些简单排序算法的下界

成员是数的数组的逆序(inversion)即具有性质 $i<j$ 但 $a[i]>a[j]$ 的序偶(a[i],a[j])。在上节的例子中，输入数据 34, 8, 64, 51, 32, 21 有 9 个逆序，即(34, 8), (34, 32), (34, 21), (64, 51), (64, 32), (64, 21), (51, 32), (51, 21)以及(32, 21)。注意，这正好是需要由插入排序(隐含)执行的交换次数。情况总是这样，因为交换两个不按顺序排列的相邻元素恰好消除一个逆序，而一个排过序的数组没有逆序。由于算法中还有 $O(N)$ 量的其他工作，因此插入排序的运行时间是 $O(I+N)$，其中 I 为原始数组中的逆序数。于是，若逆序数是 $O(N)$，则插入排序以线性时间运行。