插入排序 Insertion-sort

输入：n 个数（a1,a2, … ,an）

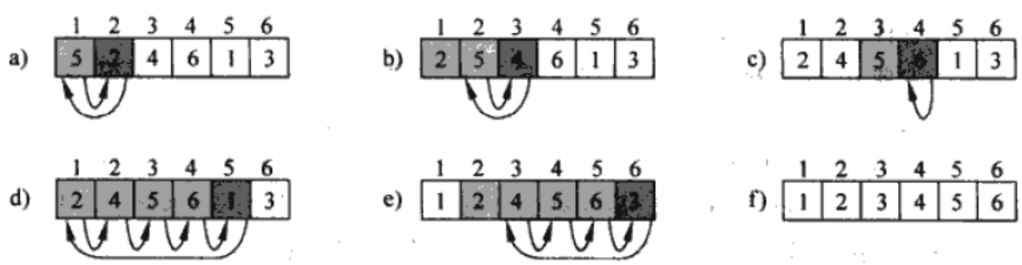
输出: 重新排序后的数列(a’1, a’2, … ,a’n)使得 a’1 <= a’2 <= … <= a’n

想象你在打牌，一开始你的左手上没有牌，你开始一次拿起一张牌，并把它插入到左手的合适位置。为了找到这张牌合适的位置，你需要将它与你手中已有的每一张牌从右往左进行比较。无论何时，你左手上的牌都是排好序的。这就是插入排序。

下面是实现插入排序的伪代码:

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1..j−1].
       i ← j−1
       while i>0 and A[i]>key
           do A[i+1] ← A[i]
              i ← i−1
       A[i+1] ← key
```

下图便于理解：



在第一次循环中，key 被赋的值‘2’，接下来需要做的就是把 key 插入到合适的位置。While 循环的判断条件被满足，所以，原序列的第二位置的‘2’ 进行了 ‘do A[a+1] ← A[i]‘操纵。第 2 位置被赋予了新的值’5‘。至此，while 循环不再满足，跳出循环后，第一位置的‘5’被赋予新的值‘2’。

插入排序的运行时间

INSERTION-SORT(A)	cost	times
1 **for** $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
2 **do** *key* $\leftarrow A[j]$	c_2	$n-1$
3 ▷ Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $j \leftarrow j-1$	c_4	$n-1$
5 **while** $i>0$ and $A[i]>key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j-1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j-1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

对于算法的总运行时间的计算，其计算方法是对它每一对 cost 与 times 之积求和。在插入排序中:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$$

插入排序存在最佳情况和最坏情况。其实这很好理解，如果输入数据都是已经排好序的话，你每一个拿到的数据，在第五行里都不需要花费额外时间，即 tj = 1。在这种情况下：

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

它的时间复杂度为 O(n)。

那么，当最坏的情况发生时，它的时间复杂度如何变化呢？最坏情况发生在当输入数据为刚好倒序排列时，在这种情况下，第五行需要做的（或者说是需要花费的时间）就变成了

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \quad 和 \quad \sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2}$$

这时，插入排序的时间复杂度变成了：

$$
\begin{aligned}
T(n) = {} & c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
& + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
= {} & \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)
\end{aligned}
$$

即 O(n^2)。

最坏情况和平均情况分析

一个算法的最坏情况反映出了它在任何输入下运行时间的上限，也就是说，我们可以保证它不会比这更坏了。

在某些特定情况下，我们也许会对一个算法的平均情况感兴趣。但有些时候，平均情况得到的结果与最坏情况是相同的。

我们一般只考虑公式中的最高次项，理由是当 n 很大时，低阶项相对来说不再重要。接着，忽略最高次项的常数系数，因为在考虑较大规模输入下的计算效率时，相对于增长率来说，系数是次要的。