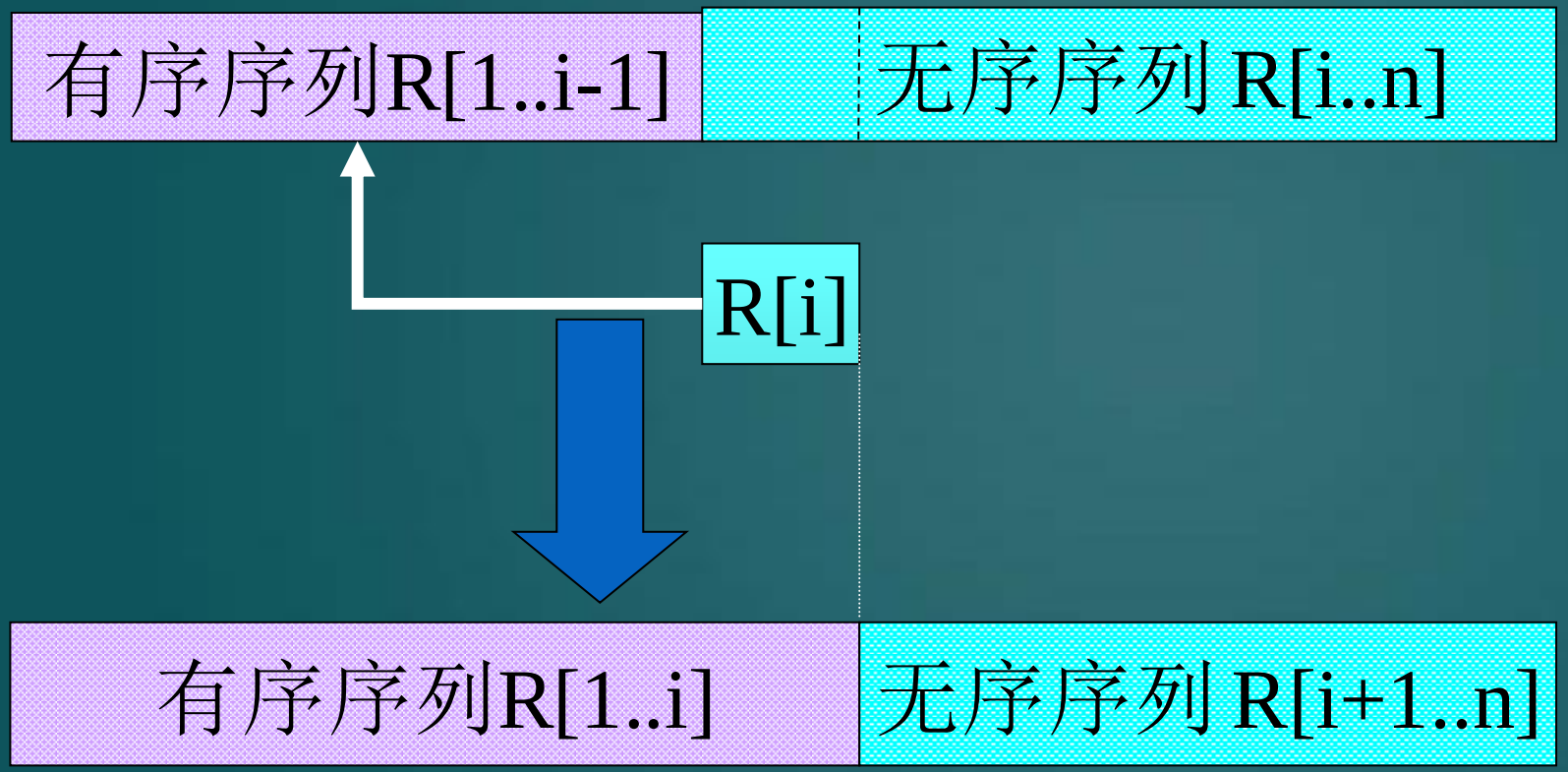
10. 2 插入排序

一趟插入排序的基本思想：



实现“一趟插入排序”可分三步进行：

1．在R[1..i-1]中**查找**R[i]的插入位置，

R[1..j].key ≤ R[i].key < R[j+1..i-1].key；

2．将R[j+1..i-1]中的所有**记录**均**后移**一个位置；

3．将R[i] **插入**(复制)到R[j+1]的位置上。

插入排序的方法有：

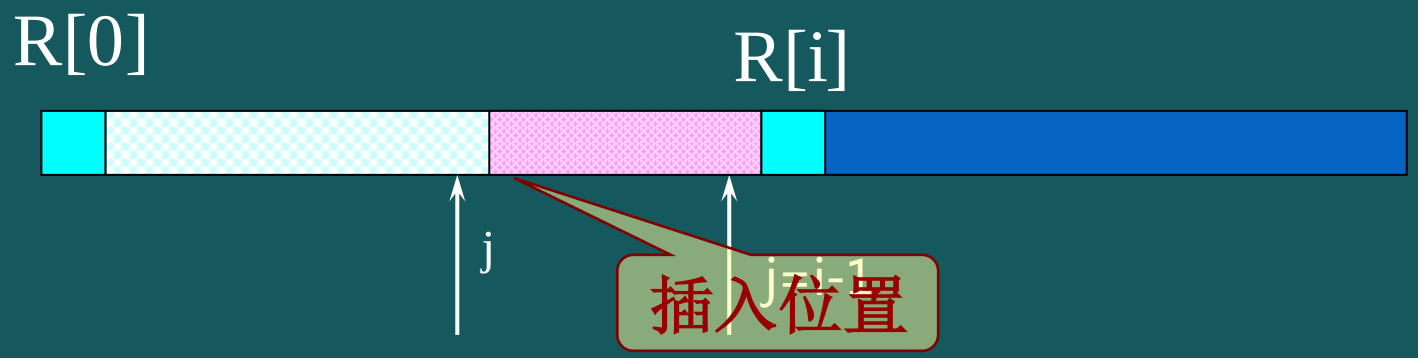
- 直接插入排序（基于顺序查找）
- 折半插入排序（基于折半查找）
- 希尔排序（基于逐趟缩小增量）

10.2.1 直接插入排序

利用 “**顺序查找**”实现“在R[1..i-1]中**查找**R[i]的插入位置”。

算法的实现要点：

从R[i-1]起向前进行顺序查找，监视哨设置在R[0]；



R[0] = R[i]; // 设置“哨兵”

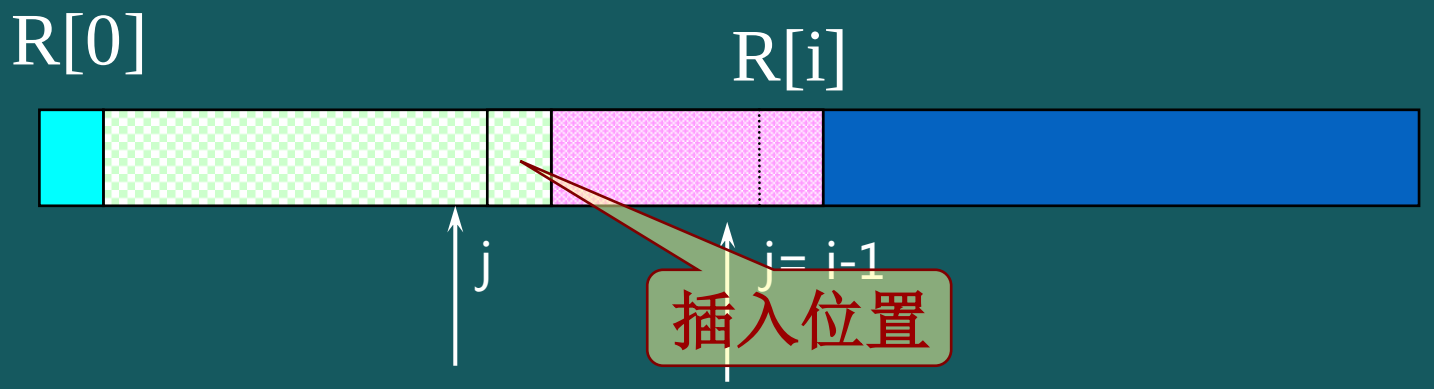
for (j=i-1; R[0].key<R[j].key; **--**j);

// 从后往前找

循环结束表明R[i]的插入位置为***j +1***

- 对于在查找过程中找到的那些关键字不小于R[i].key的记录，并在查找的同时实现记录向后移动；

```
for (j=i-1; R[0].key<R[j].key; --j);
    R[j+1] = R[j]
```



- 上述循环结束后可以直接进行“插入”

令 i = 2，3，…, n,

实现整个序列的排序。

```
for ( i=2;  i<=n;  ++i )
  if (R[i].key<R[i-1].key)
    {  在 R[1..i-1]中查找R[i]的插入位置;
       插入R[i] ;
    }
```

```
void InsertionSort ( SqList &L ) {
  // 对顺序表 L 作直接插入排序。
   for ( i=2; i<=L.length; ++i )
       if (L.r[i].key < L.r[i-1].key) {
       L.r[0] = L.r[i];              // 复制为监视哨
       for ( j=i-1; L.r[0].key < L.r[j].key;  -- j )
           L.r[j+1] = L.r[j];        // 记录后移
       L.r[j+1] = L.r[0];            // 插入到正确位置
    }
} // InsertSort
```

例如：

图10.1 直接插入排序示例

内部排序的**时间分析**：

实现内部排序的**基本操作**有两个：

（1）**“比较”**序列中两个关键字的大小；

（2）**“移动”**记录。

对于直接插入排序：

最好的情况（关键字在记录序列中顺序有序）：

“比较”的次数：

$$\sum_{i=2}^{n} 1 = n - 1$$

“移动”的次数：

$$\mathbf{0}$$

最坏的情况（关键字在记录序列中逆序有序）：

“比较”的次数：

$$\sum_{i=2}^{n}(i+1) = \frac{(n+4)(n-1)}{2}$$

“移动”的次数：

$$\sum_{i=2}^{n}(i+1) = \frac{(n+4)(n-1)}{2}$$

10.2.2 其他插入排序

1.**折半插入排序**

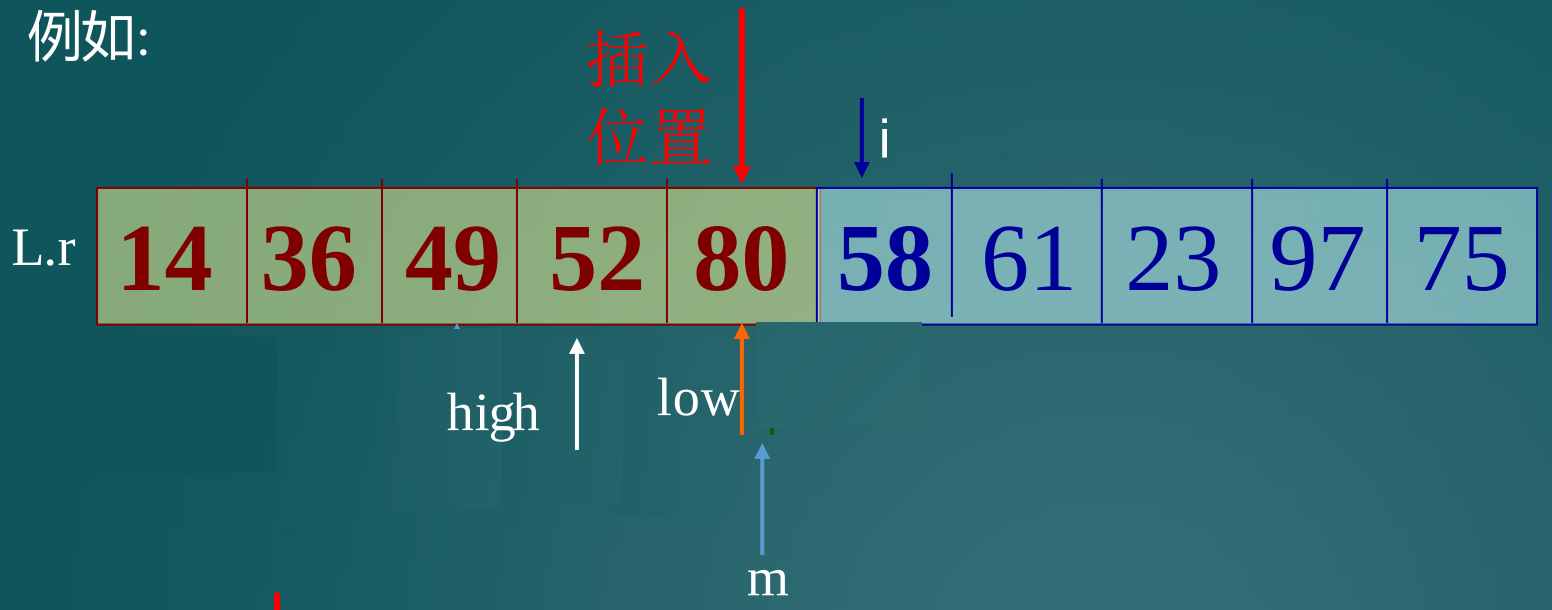
因为 R[1..i-1] 是一个按关键字有序的有序序列，则可以利用**折半查找**实现“在R[1..i-1]中**查找**R[i]的插入位置”，如此实现的插入排序为**折半插入**排序。

折半插入排序的算法如下页算法10.2

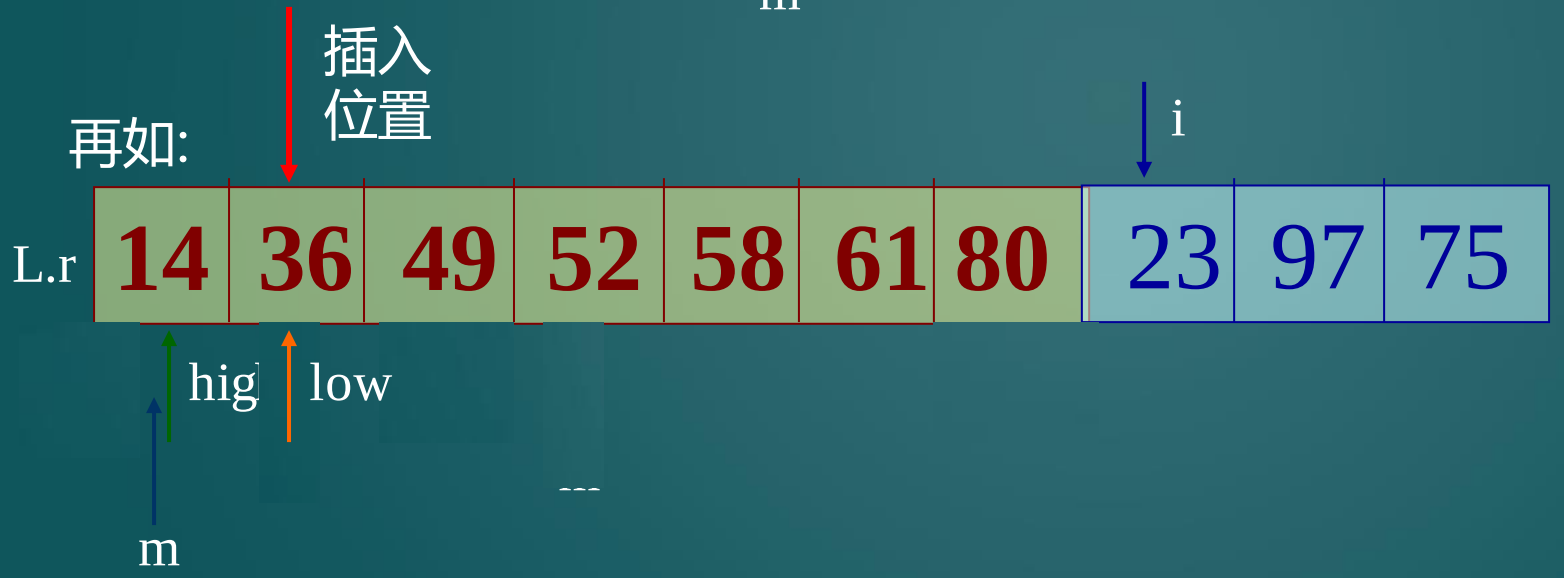
```
void BiInsertionSort ( SqList &L ) {//对顺序表作折半插入排序
  for ( i=2; i<=L.length; ++i ) {
      L.r[0] = L.r[i];           // 将 L.r[i] 暂存到 L.r[0]
      low = 1;   high = i-1;
      while (low<=high) {//在r[low..high]中折半查找插入位置
          m = (low+high)/2;          // 折半
          if (L.r[0].key < L.r[m].key)    high = m-1;   // 低半区查找
          else  low = m+1;           // 在高半区查找
      } // while
      for ( j=i-1;  j>=high+1;  --j )
           L.r[j+1] = L.r[j];          // 记录后移
      L.r[high+1] = L.r[0];        // 插入
  } // for
} // BInsertSort
```

算法 10.2

例如:


插入
位置
i
L.r
14 36 49 52 80 58 61 23 97 75
high
low
m

再如:


插入
位置
i
L.r
14 36 49 52 58 61 80 23 97 75
hig
low
m

10.2.3 希尔排序

希尔排序（又称缩小增量排序）

基本思想：对待排记录序列先作“宏观”调整，再作“微观”调整。

所谓“宏观”调整，指的是，“跳跃式”的插入排序。

具体做法为：

将记录序列分成若干子序列，分别对每个子序列进行插入排序。待整个序列中的纪录“基本有序”时，再对全体记录进行一次直接插入排序。

例如：将 n 个记录分成 d 个子序列：

{ R[1]，R[1+d]，R[1+2d]，…，R[1+kd] }

{ R[2]，R[2+d]，R[2+2d]，…，R[2+kd] }

…

{ R[d]，R[2d]，R[3d]，…，R[kd]，R[(k+1)d] }

其中，**d** 称为增量，它的值在排序过程中从大到小逐渐缩小，直至最后一趟排序减为 1。

例如：

16	25	12	30	47	11	23	36	9	18	31

第一趟希尔排序，设增量 **d =5**

11	23	12	9	18	16	25	36	30	47	31

第二趟希尔排序，设增量 **d = 3**

9	18	12	11	23	16	25	31	30	47	36

第三趟希尔排序，设增量 **d = 1**

9	11	12	16	18	23	25	30	31	36	47

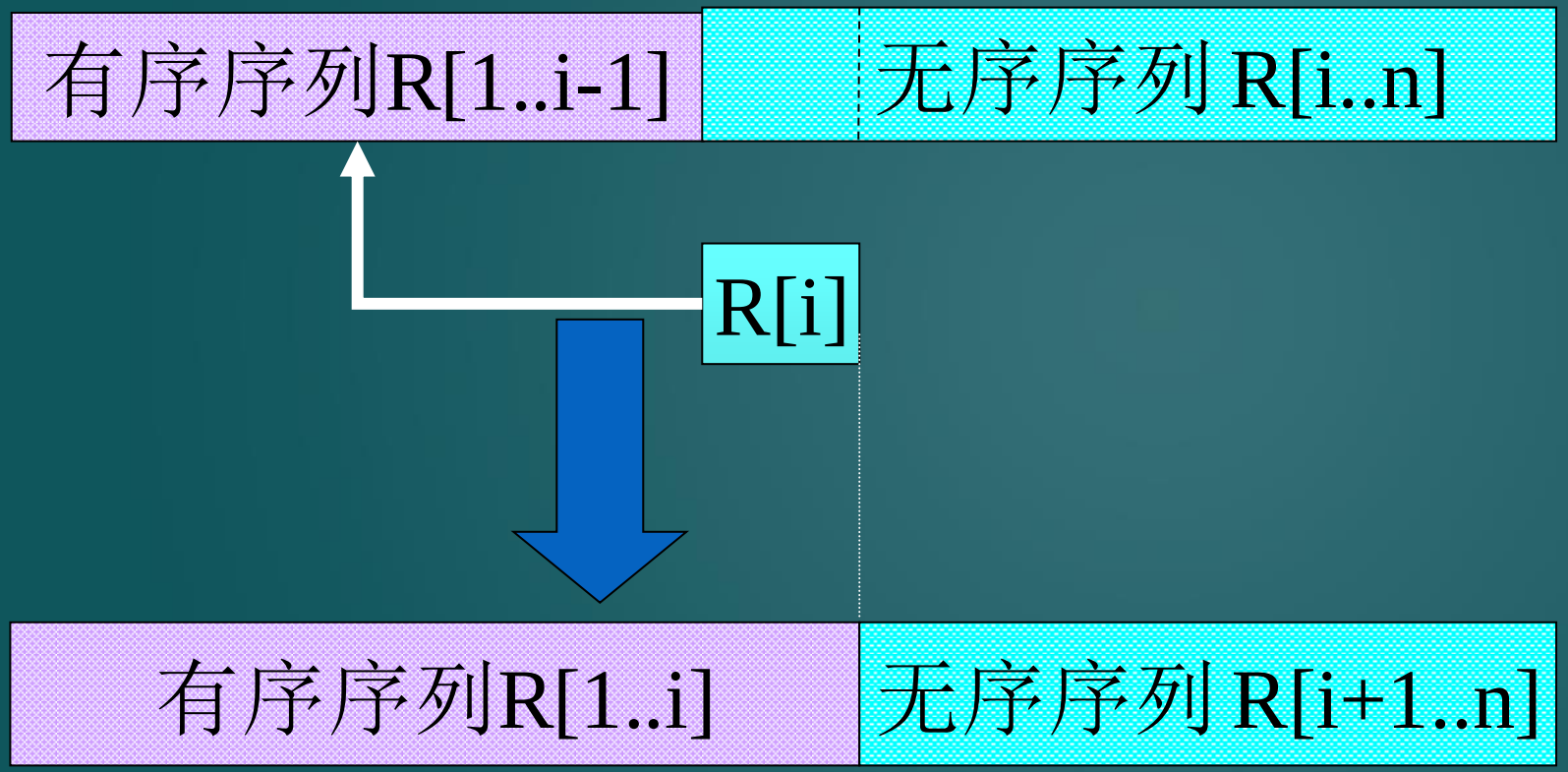
```
void ShellInsert ( SqList &L, int dk ) {//对顺序表L作一
//趟希尔插入排序，dk 为增量
   for ( i=dk+1; i<=n; ++i )
      if ( L.r[i].key< L.r[i-dk].key) {
        L.r[0] = L.r[i];            // 暂存在L.r[0]
        for (j=i-dk;  j>0&&(L.r[0].key<L.r[j].key); j-=dk)
           L.r[j+dk] = L.r[j];  // 记录后移，查找插入位置
        L.r[j+dk] = L.r[0];                // 插入
      } // if
} // ShellInsert
```

```
void ShellSort (SqList &L, int dlta[], int t)
{  // 按增量序列dlta[0..t-1]对顺序表L作希尔排序
    for (k=0; k<t; ++t)
        ShellInsert(L, dlta[k]);
            //一趟增量为dlta[k]的插入排序
} // ShellSort
```

算法 10.5

10. 2 插入排序

一趟插入排序的基本思想：



实现“一趟插入排序”可分三步进行：

1．在R[1..i-1]中**查找**R[i]的插入位置，

R[1..j].key ≤ R[i].key < R[j+1..i-1].key；

2．将R[j+1..i-1]中的所有**记录**均**后移**一个位置；

3．将R[i] **插入**(复制)到R[j+1]的位置上。

插入排序的方法有：

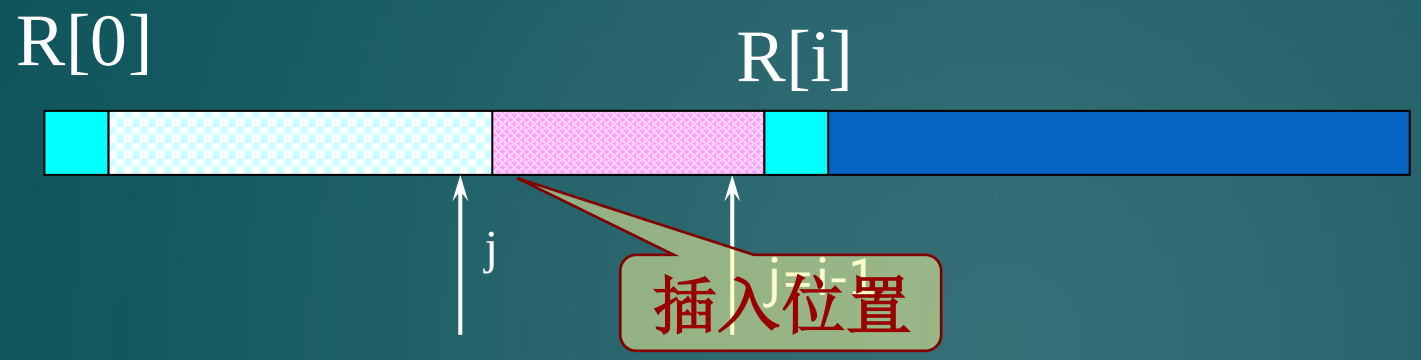
- 直接插入排序（基于顺序查找）
- 折半插入排序（基于折半查找）
- 希尔排序（基于逐趟缩小增量）

10.2.1 直接插入排序

利用 “**顺序查找**”实现“在R[1..i-1]中**查找**R[i]的插入位置”。

算法的实现要点：

- 从R[i-1]起向前进行顺序查找，监视哨设置在R[0]；



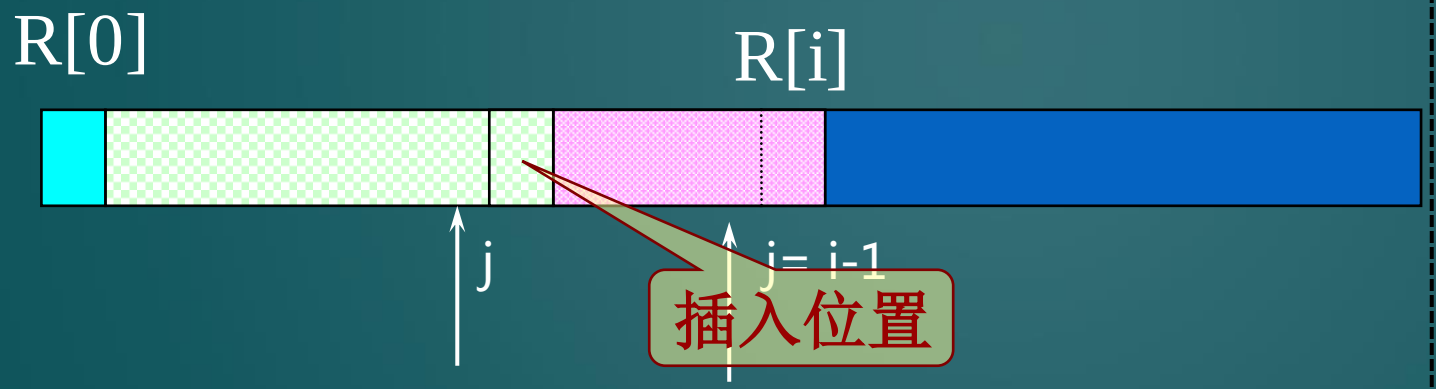

```
R[0] = R[i];            // 设置“哨兵”

for (j=i-1; R[0].key<R[j].key; --j);
                        // 从后往前找
```

循环结束表明R[i]的插入位置为 ***j +1***

- 对于在查找过程中找到的那些关键字不小于R[i].key的记录，并在查找的同时实现记录向后移动；

```
for (j=i-1; R[0].key<R[j].key; --j);
    R[j+1] = R[j]
```



- 上述循环结束后可以直接进行“插入”

令 i = 2，3，…, n,

实现整个序列的排序。

```
for ( i=2;  i<=n;  ++i )
  if (R[i].key<R[i-1].key)
    {  在 R[1..i-1]中查找R[i]的插入位置;
       插入R[i] ;
    }
```

```
void InsertionSort ( SqList &L ) {
  // 对顺序表 L 作直接插入排序。
   for ( i=2; i<=L.length; ++i )
       if (L.r[i].key < L.r[i-1].key) {
       L.r[0] = L.r[i];              // 复制为监视哨
       for ( j=i-1; L.r[0].key < L.r[j].key;  -- j )
           L.r[j+1] = L.r[j];        // 记录后移
       L.r[j+1] = L.r[0];            // 插入到正确位置
    }
} // InsertSort
```

算法 10.1

例如：

图10.1 直接插入排序示例

内部排序的**时间分析**：

实现内部排序的**基本操作**有两个：

（1）**“比较”**序列中两个关键字的大小；

（2）**“移动”**记录。

对于直接插入排序：

最好的情况（关键字在记录序列中顺序有序）：

“比较”的次数：

$$\sum_{i=2}^{n} 1 = n - 1$$

“移动”的次数：

$$\mathbf{0}$$

最坏的情况（关键字在记录序列中逆序有序）：

“比较”的次数：

$$\sum_{i=2}^{n}(i+1) = \frac{(n+4)(n-1)}{2}$$

“移动”的次数：

$$\sum_{i=2}^{n}(i+1) = \frac{(n+4)(n-1)}{2}$$

10.2.2 其他插入排序

1.**折半插入排序**

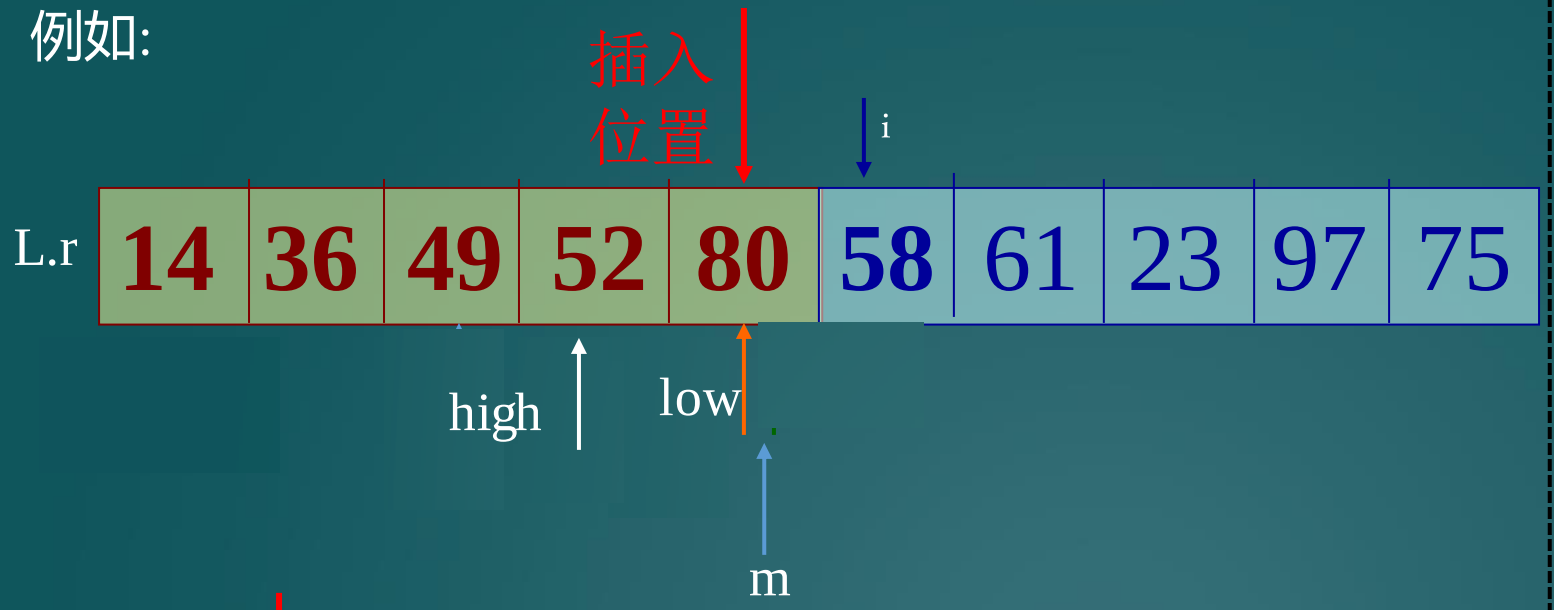
因为 R[1..i-1] 是一个按关键字有序的有序序列，则可以利用**折半查找**实现“在R[1..i-1]中**查找**R[i]的插入位置”，如此实现的插入排序为**折半插入**排序。

折半插入排序的算法如下页算法10.2

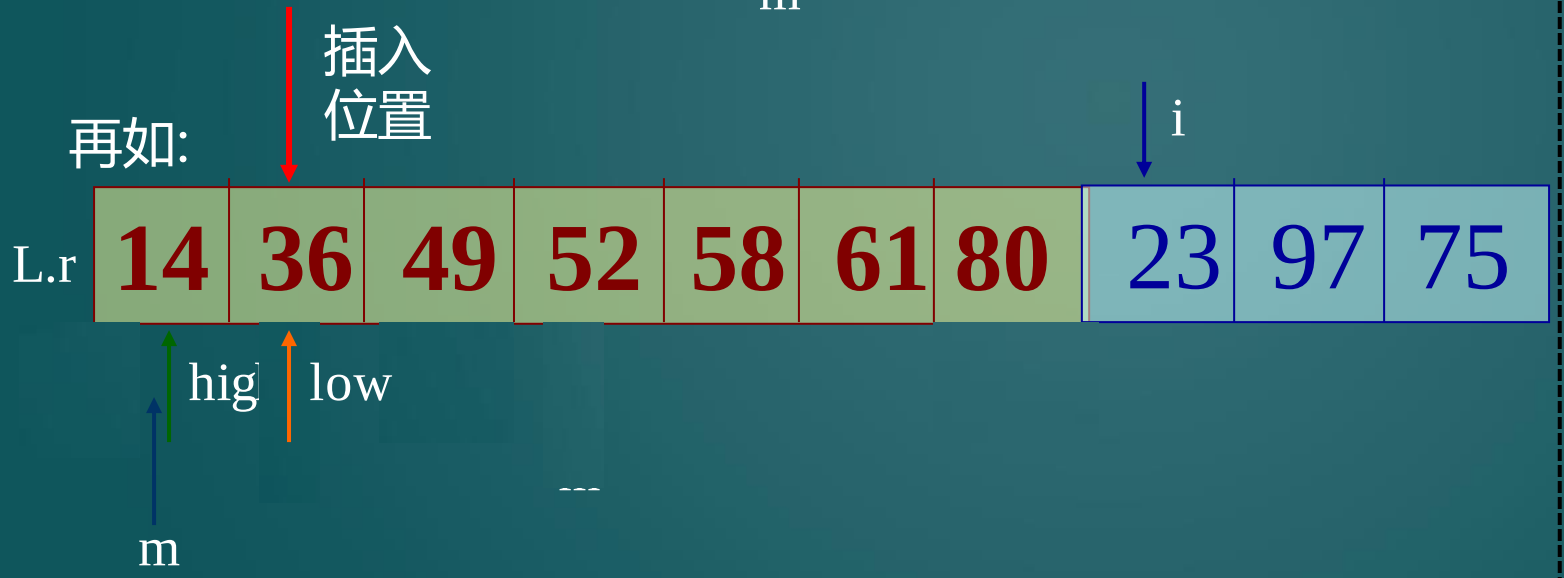
```
void BiInsertionSort ( SqList &L ) {//对顺序表作折半插入排序
  for ( i=2; i<=L.length; ++i ) {
      L.r[0] = L.r[i];           // 将 L.r[i] 暂存到 L.r[0]
      low = 1;   high = i-1;
      while (low<=high) {//在r[low..high]中折半查找插入位置
          m = (low+high)/2;          // 折半
          if (L.r[0].key < L.r[m].key)    high = m-1;   // 低半区查找
          else  low = m+1;           // 在高半区查找
      } // while
      for ( j=i-1;  j>=high+1;  --j )
           L.r[j+1] = L.r[j];         // 记录后移
      L.r[high+1] = L.r[0];       // 插入
  } // for
} // BInsertSort
```

算法 10.2

例如:


插入
位置
i
L.r
14 36 49 52 80 58 61 23 97 75
high
low
m

再如:


插入
位置
i
L.r
14 36 49 52 58 61 80 23 97 75
hig
low
m

10.2.3 希尔排序

希尔排序（又称缩小增量排序）

基本思想：对待排记录序列先作“宏观”调整，再作“微观”调整。

所谓“宏观”调整，指的是，“跳跃式”的插入排序。

具体做法为：

将记录序列分成若干子序列，分别对每个子序列进行插入排序。待整个序列中的纪录“基本有序”时，再对全体记录进行一次直接插入排序。

例如：将 n 个记录分成 d 个子序列：

{ R[1]，R[1+d]，R[1+2d]，…，R[1+kd] }

{ R[2]，R[2+d]，R[2+2d]，…，R[2+kd] }

…

{ R[d]，R[2d]，R[3d]，…，R[kd]，R[(k+1)d] }

其中，**d** 称为增量，它的值在排序过程中从大到小逐渐缩小，直至最后一趟排序减为 1。

例如：

16	25	12	30	47	11	23	36	9	18	31

第一趟希尔排序，设增量 **d =5**

11	23	12	9	18	16	25	36	30	47	31

第二趟希尔排序，设增量 **d = 3**

9	18	12	11	23	16	25	31	30	47	36

第三趟希尔排序，设增量 **d = 1**

9	11	12	16	18	23	25	30	31	36	47

```
void ShellInsert ( SqList &L, int dk ) {//对顺序表L作一
//趟希尔插入排序，dk 为增量
   for ( i=dk+1; i<=n; ++i )
      if ( L.r[i].key< L.r[i-dk].key) {
        L.r[0] = L.r[i];            // 暂存在L.r[0]
        for (j=i-dk;  j>0&&(L.r[0].key<L.r[j].key); j-=dk)
           L.r[j+dk] = L.r[j];  // 记录后移，查找插入位置
        L.r[j+dk] = L.r[0];                // 插入
      } // if
} // ShellInsert
```

```
void ShellSort (SqList &L, int dlta[], int t)
{  // 按增量序列dlta[0..t-1]对顺序表L作希尔排序
    for (k=0; k<t; ++t)
        ShellInsert(L, dlta[k]);
            //一趟增量为dlta[k]的插入排序
} // ShellSort
```

算法 10.5