10.2.3 希尔(Shell)排序

希尔排序（缩小增量排序）**基本思路**：

- 先取定一个小于n的整数d_1作为第一个增量，把表的全部记录分成d_1个组，所有距离为d_1的倍数的记录放在同一个组中，在各组内进行直接插入排序。
- 然后取第二个增量d_2（$<d_1$），重复上述的分组和排序，直至所取的增量$d_t=1$（$d_t<d_{t-1}<\ldots<d_2<d_1$），即所有记录放在同一组中进行直接插入排序为止。

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。

例如：将 n 个记录分成 d 个子序列：

$\{ R[0], \quad R[d], \quad R[2d], \ldots, \quad R[kd] \}$

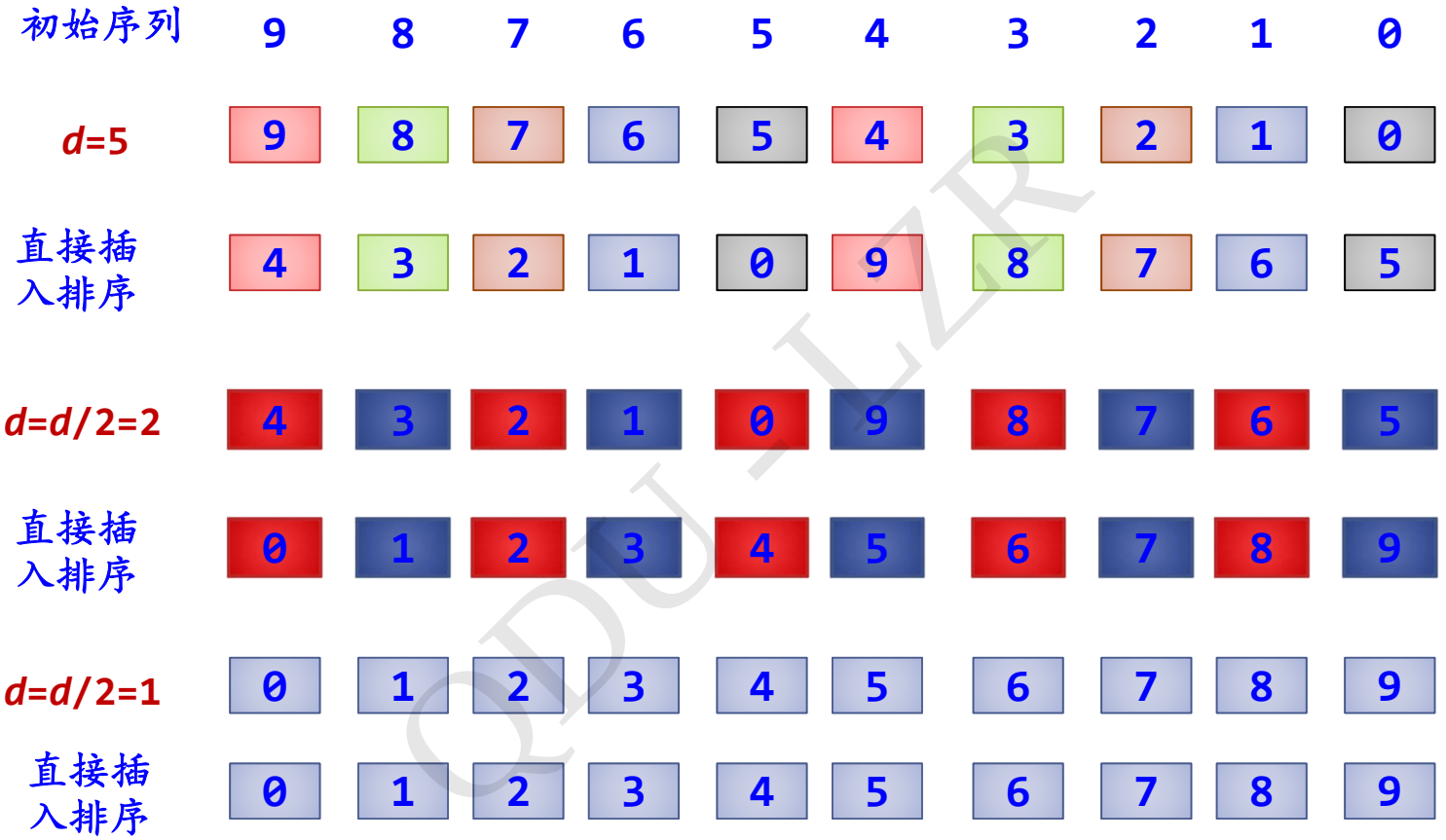
$\{ R[1], \quad R[1+d], \quad R[1+2d], \ldots, R[1+kd] \}$

…

$\{ R[d-1], R[2d-1], R[3d-1], \ldots, R[(k+1)d-1] \}$

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。

【示例-1】



注意：对于d=1的一趟，排序前的数据已将近正序！

取$d_1 = n/2$，$d_{i+1} = \lfloor d_i/2 \rfloor$时的希尔排序的算法如下：

```
void ShellSort(RedType R[],int n)
{
   int i,j,d;
   RedType   tmp;
   d=n/2;                  //增量置初值
   while (d>0)
   {  for (i=d;i<n;i++)
      {  tmp=R[i];         //对所有相隔d位置的记录组采用直接插入排序
         j=i-d;
         while (j>=0 && tmp.key<R[j].key)
         {  R[j+d]=R[j];
            j=j-d;
         }
         R[j+d]=tmp;
      }
      d=d/2;               //减小增量
   }
}
```

希尔排序的算法如下：

```
void ShellInsert(SqList &L,  int  dk)
{
  // 对顺序表L作一趟希尔插入排序。本算法是和一趟直接插入排序相比，
  // 作了以下修改：
  // 1.前后记录位置的增量是dk，而不是1;
  // 2.r[0]只是暂存单元，不是哨兵。当j<=0时，插入位置已找到。算法10.4
  int i, j;
  for(i = dk + 1; i <= L.length; ++i)
    if (LT(L.r[i].key, L.r[i - dk].key) ) {
      // 需将L.r[i]插入有序增量子表
      L.r[0] = L.r[i];                    // 暂存在L.r[0]
      for(j = i - dk; j > 0 && LT(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
    }
}
```

- 希尔排序的分析比较复杂，涉及一些数学上的问题，其时间是所取的“增量”序列的函数。希尔排序的时间复杂度约为$O(n^{1.3})$。

- **希尔排序特点**

子序列的构成不是简单的“逐段分割”，而是将相隔某个增量的记录组成一个子序列。

- **希尔排序可提高排序速度，原因是：**
  - 分组后n值减小，n^2更小，而$T(n)=O(n^2)$，所以T(n)从总体上看是减小了；
  - 关键字较小的记录跳跃式前移，在进行最后一趟增量为1的插入排序时，序列已基本有序。

- **增量序列取法**
  - 无除1以外的公因子；
  - 最后一个增量值必须为1。

归纳起来，希尔排序算法的性能如表所示。

时间复杂度			空间复杂度	稳定性
最好情况	最坏情况	平均情况		
—	—	$O(n^{1.3})$	O(1)	不稳定

END