数据结构

（C语言版）（第2版）

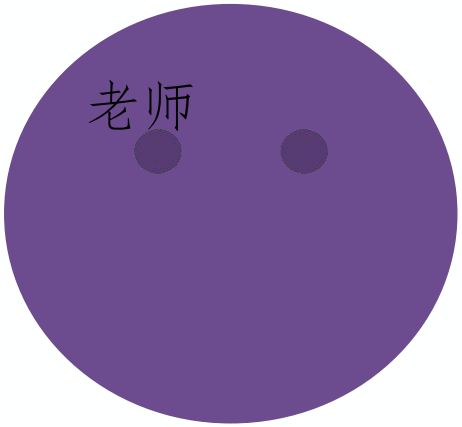
排序

插入排序

主讲教师：汪红松

教 学 内 容 Contents

一、直接插入排序
二、折半插入排序
三、希尔排序



插入排序

基本思想：

每步将一个待排序的对象，按其关键码大小，插入到前面已经排好序的一组对象的适当位置上，直到对象全部插入为止。

即边插入边排序，保证子序列中随时都是排好序的。

插入排序

不同的具体实现方法导致不同的算法描述。

最简单的排序法！

- 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）

一、直接插入排序

例（13，6，3，31，9，27，5，11）

排序过程：整个排序过程为n-1趟插入，即先将序列中第1个记录看成是一个有序子序列，然后从第2个记录开始，逐个进行插入，直至整个序列有序。

【13】, 6, 3, 31, 9, 27, 5, 11

【6, 13】, 3, 31, 9, 27, 5, 11

【3, 6, 13】, 31, 9, 27, 5, 11

【3, 6, 13，31】, 9, 27, 5, 11

【3, 6, 9, 13，31】, 27, 5, 11

【3, 6, 9, 13，27, 31】, 5, 11

【3, 5, 6, 9, 13，27, 31】, 11

【3, 5, 6, 9, 11，13，27, 31】

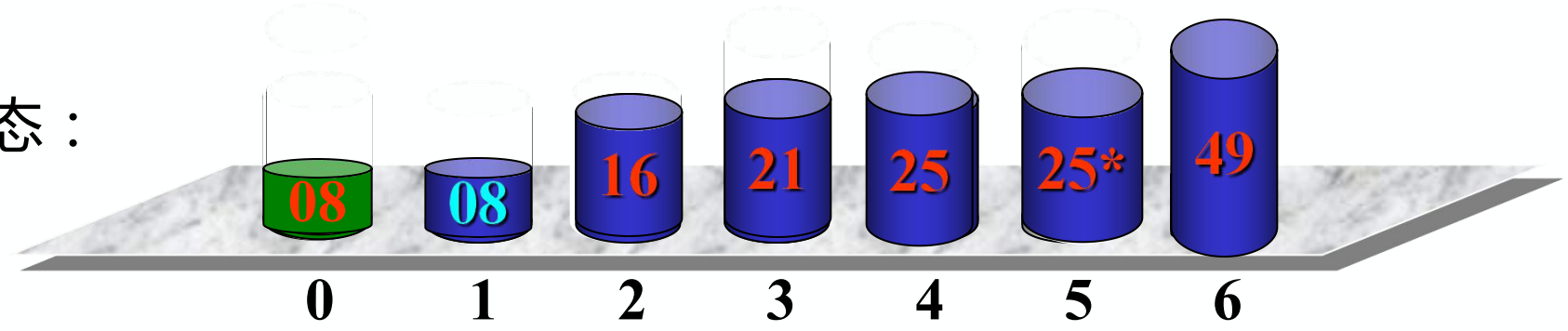
一、直接插入排序

*表示后一个25

(21 , 25 , 49 , 25* , 16 , 08)

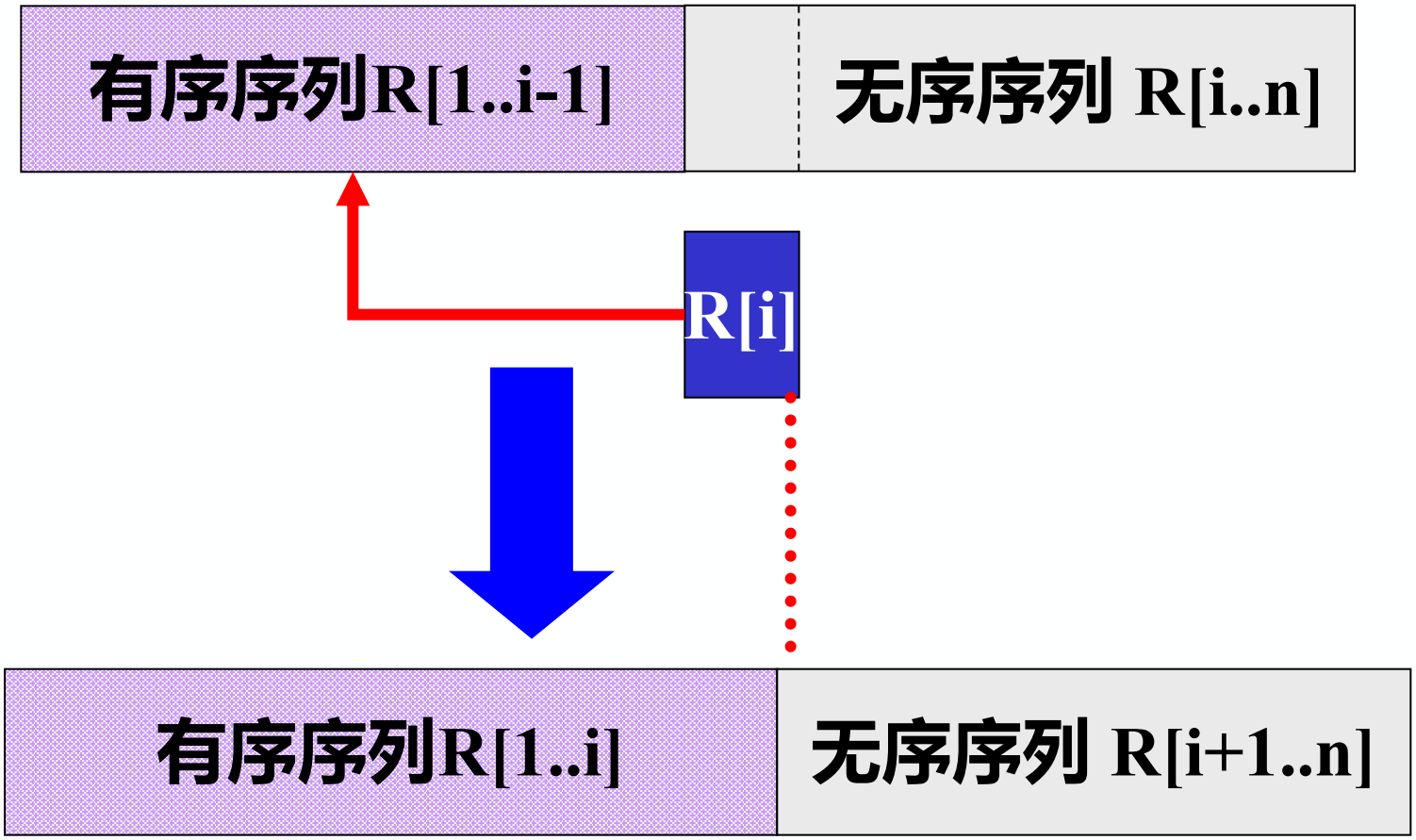
将序列存入顺序表L中，将L.r[0]作为哨兵

初态：



完成!

一、直接插入排序　　1.插入排序的基本思想



一、直接插入排序　2.插入排序的基本步骤

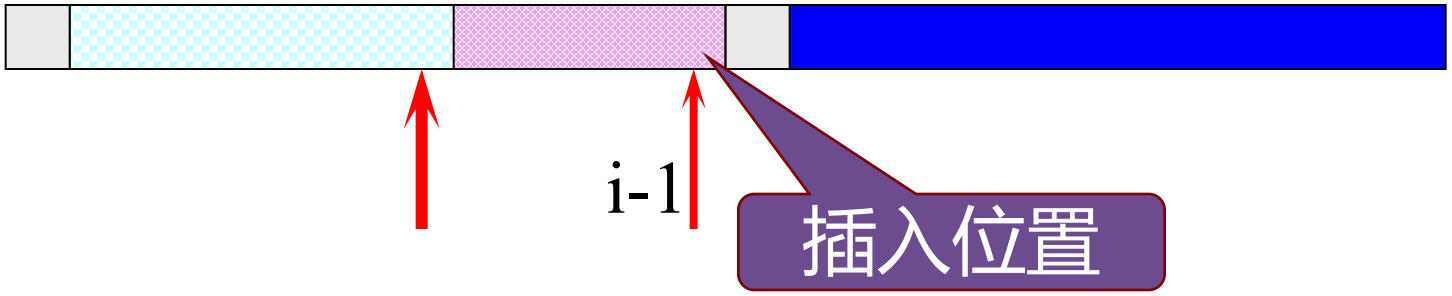
01 OPTION　在R[1..i-1]中查找R[i]的插入位置，R[1..j].key　R[i].key< R[j+1..i-1].key；

02 OPTION　将R[j+1..i-1]中的所有记录均后移一个位置；

03 OPTION　将R[i] 插入到R[j+1]的位置上。

直接插入排序

- 从R[i-1]向前进行顺序查找，监视哨设置在R[0]



- 关键字大于R[i].key的记录向后移动

```
if( L.r[i].key<L.r[i-1].key){
    R[0] = R[i]; // 设置“哨兵”          R[i] = R[i-1];
 for (j=i-2; R[0].key<R[j].key; --j)      R[j+1] = R[j];
```

- 循环结束表明R[i]的插入位置为 j+1

```
L.r[j+1]=L.r[0]; //插入到正确位置
```

一、直接插入排序

```
void InsertSort(SqList &L)
 {int i,j;
   for(i=2;i<=L.length;++i)
     if( L.r[i].key<L.r[i-1].key)//将L.r[i]插入有序子表
       { L.r[0]=L.r[i]; // 复制为哨兵
         L.r[i]=L.r[i-1];
         for(j=i-2; 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
       }
}
```

- 设对象个数为n，则执行n-1趟
- 比较次数和移动次数与初始排列有关

最好情况下：

每趟只需比较 1 次，不移动

总比较次数为 n-1


```
for(i=2;i<=L.length;++i)
    if( L.r[i].key<L.r[i-1].key)
```

一、直接插入排序

3.算法分析

最坏情况下：第 i 趟比较 i 次，移动 $i+1$ 次。

比较次数: $\sum_{i=2}^{n} i = \frac{(n+2)(n-1)}{2}$

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

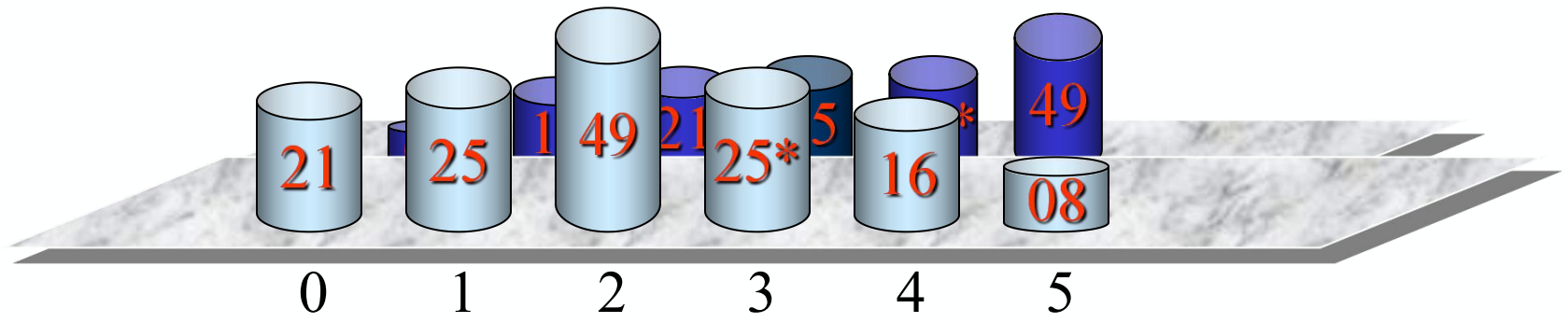
```
if( L.r[i].key<L.r[i-1].key)      {
   L.r[0]=L.r[i]; // 复制为哨兵
   L.r[i]=L.r[i-1];
   ……
   L.r[j+1]=L.r[0]; //插入到正确位置 }
```

一、直接插入排序

3.算法分析

- 若出现各种可能排列的概率相同，则可取最好情况和最坏情况的平均情况；
- 平均情况比较次数和移动次数为$n^2/4$。

- 时间复杂度为 o(n^2)
- 空间复杂度为 o(*1*)
- 是一种稳定的排序方法



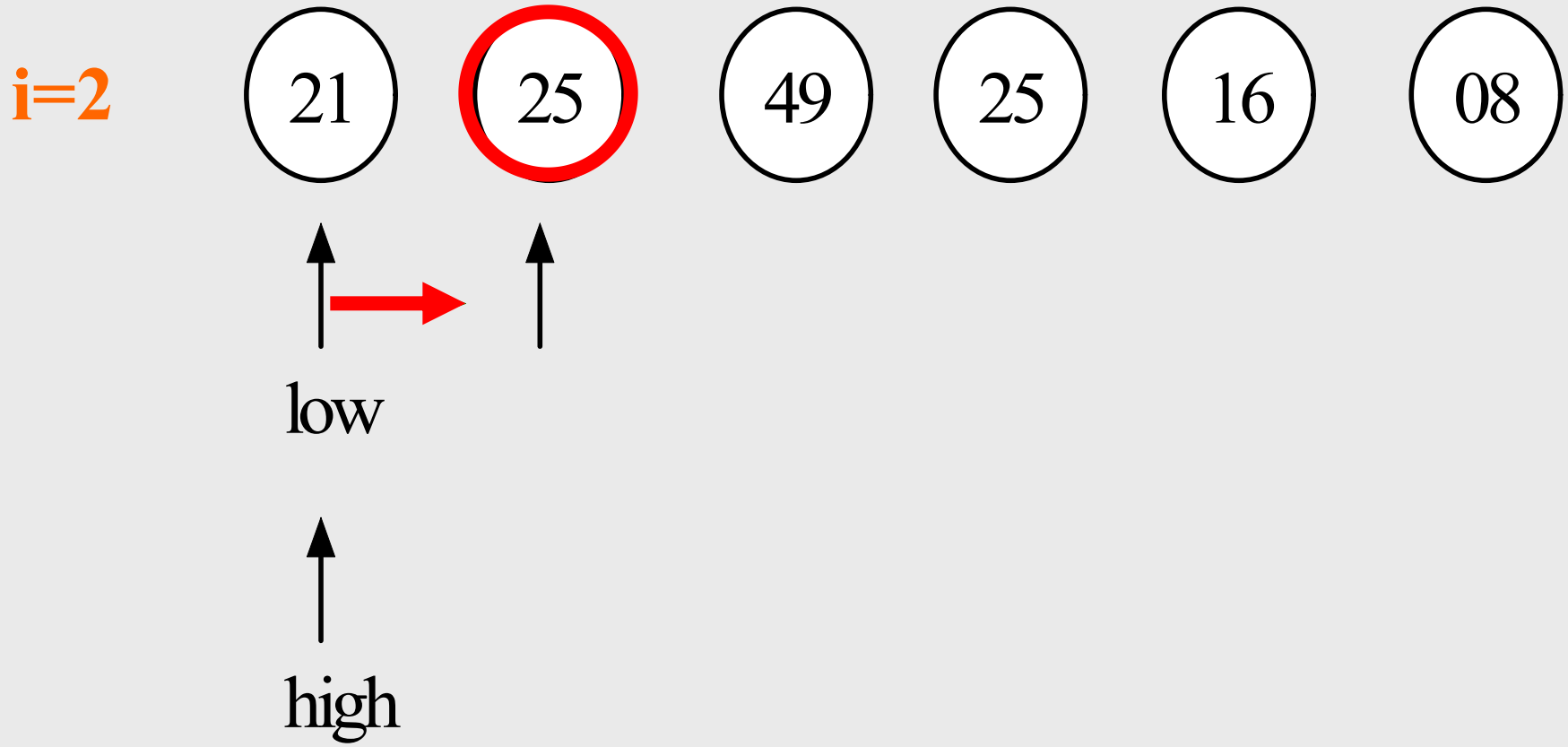
直接插入排序

减少关键字间的比较次数

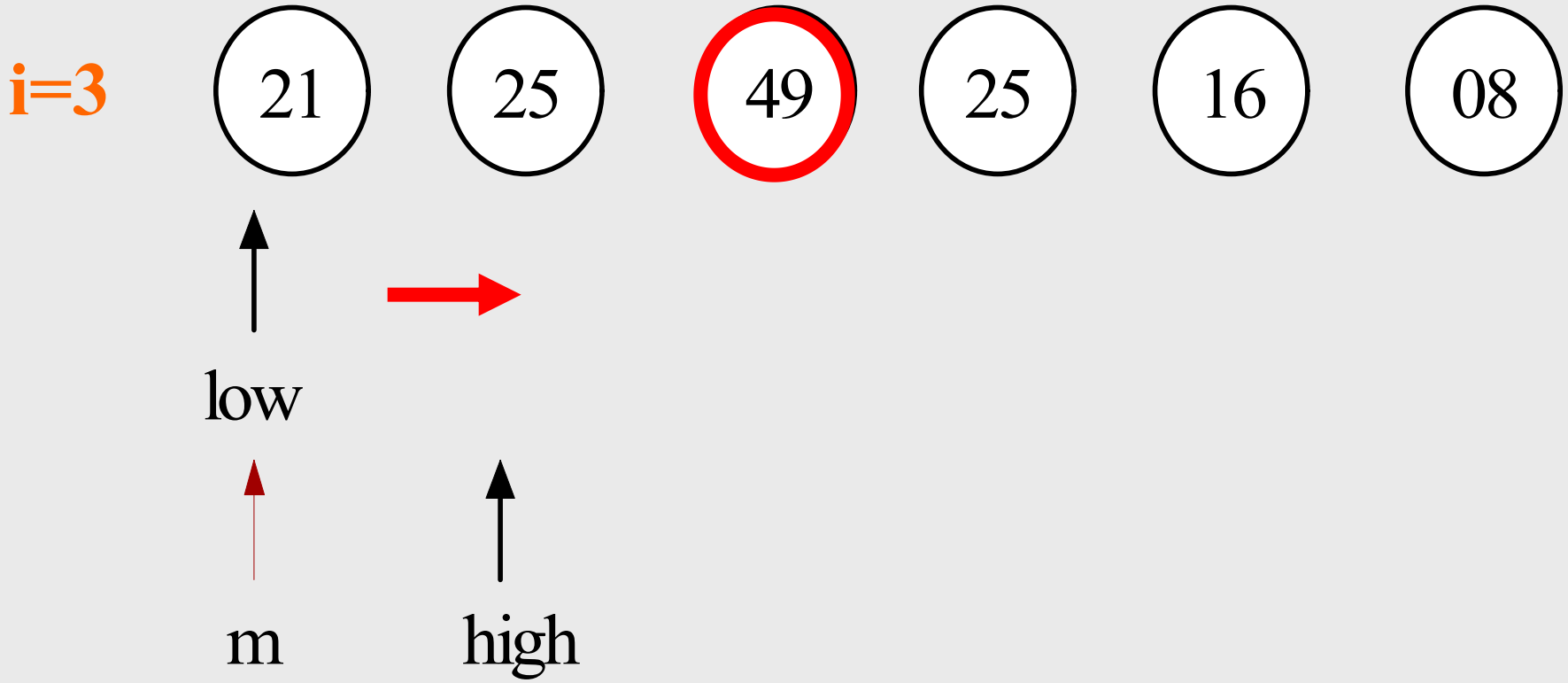
折半插入排序

在插入 $r[i]$ 时，利用折半查找法寻找 $r[i]$ 的插入位置。

二、折半插入排序

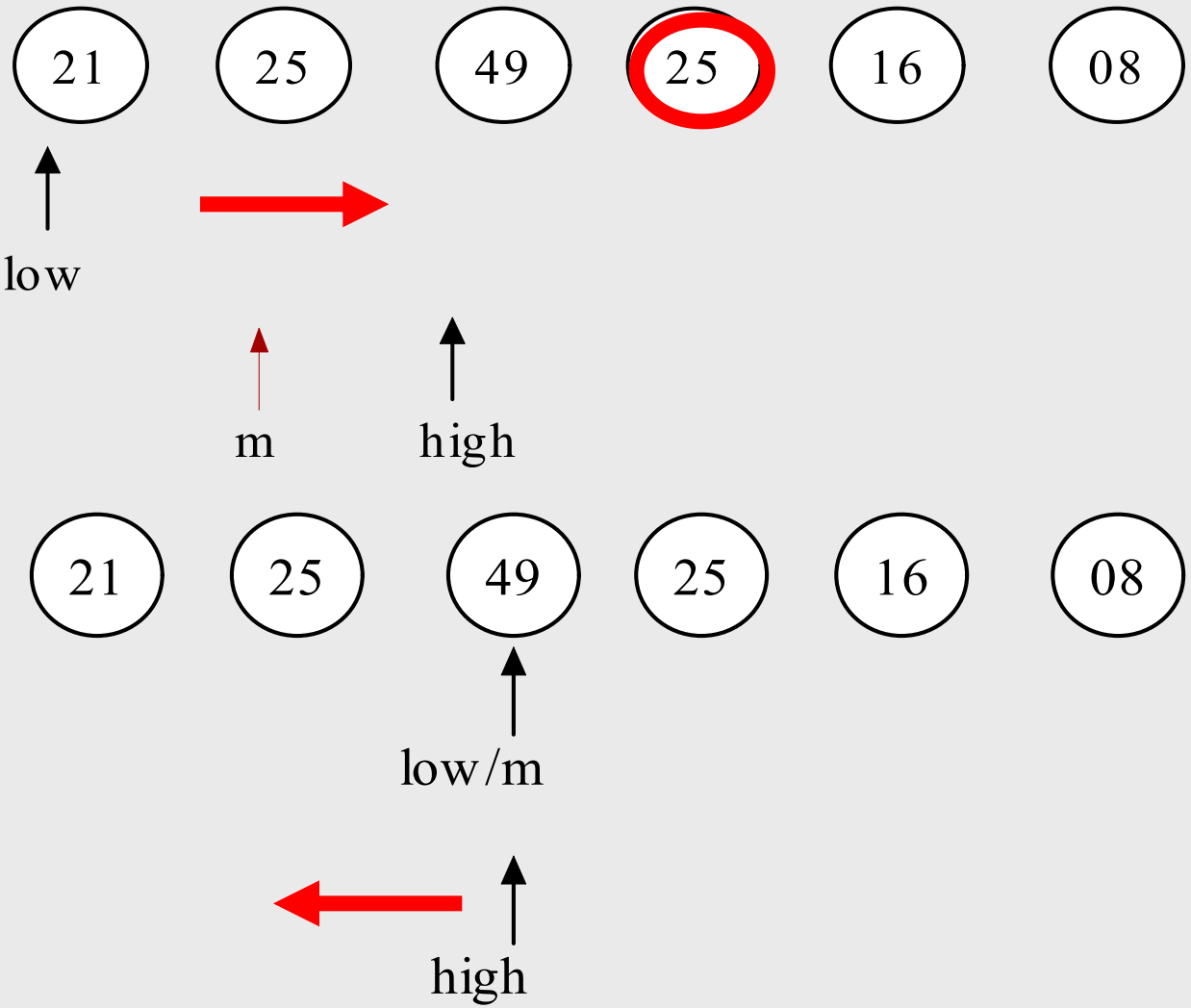


二、折半插入排序

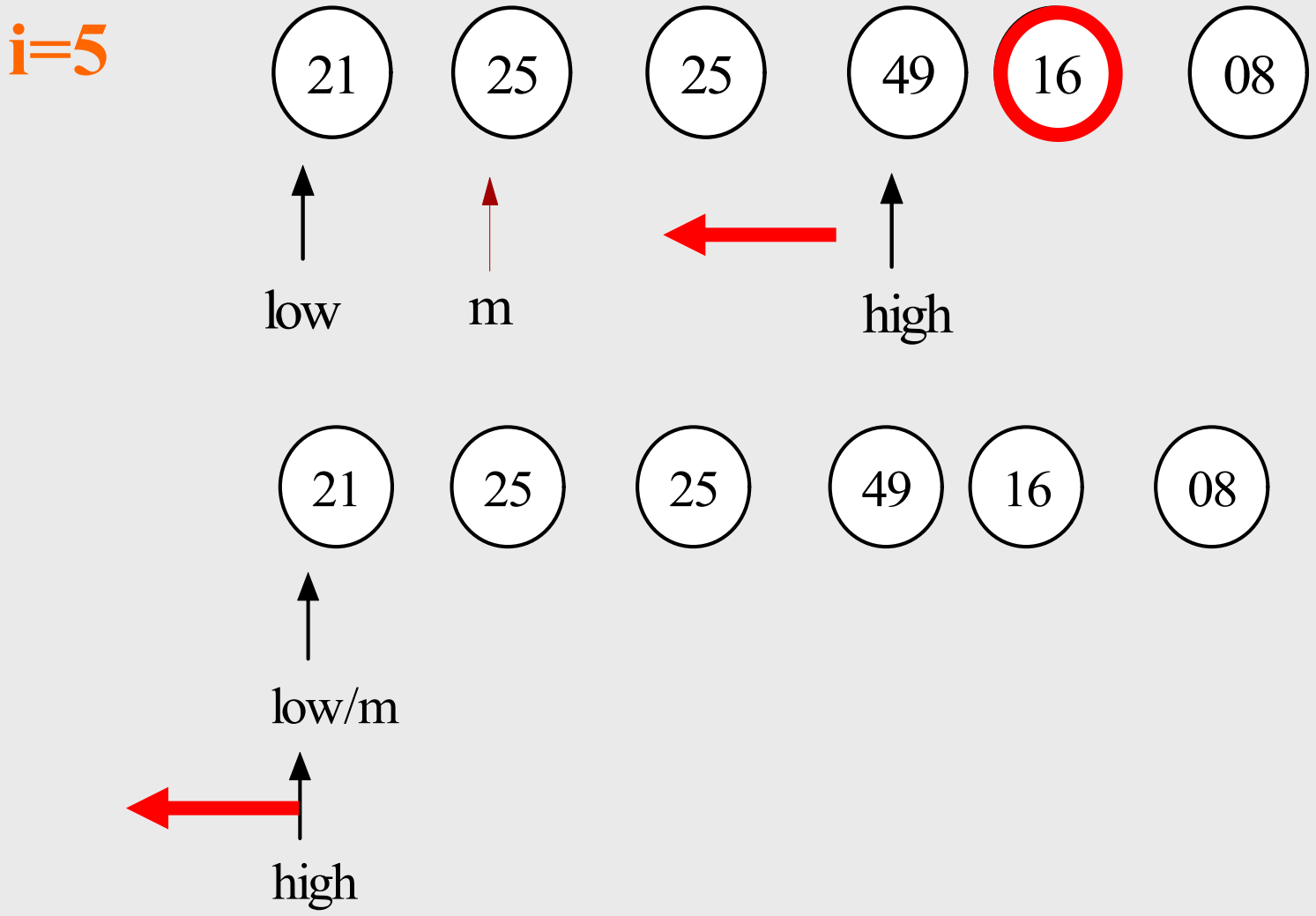


二、折半插入排序

i=4

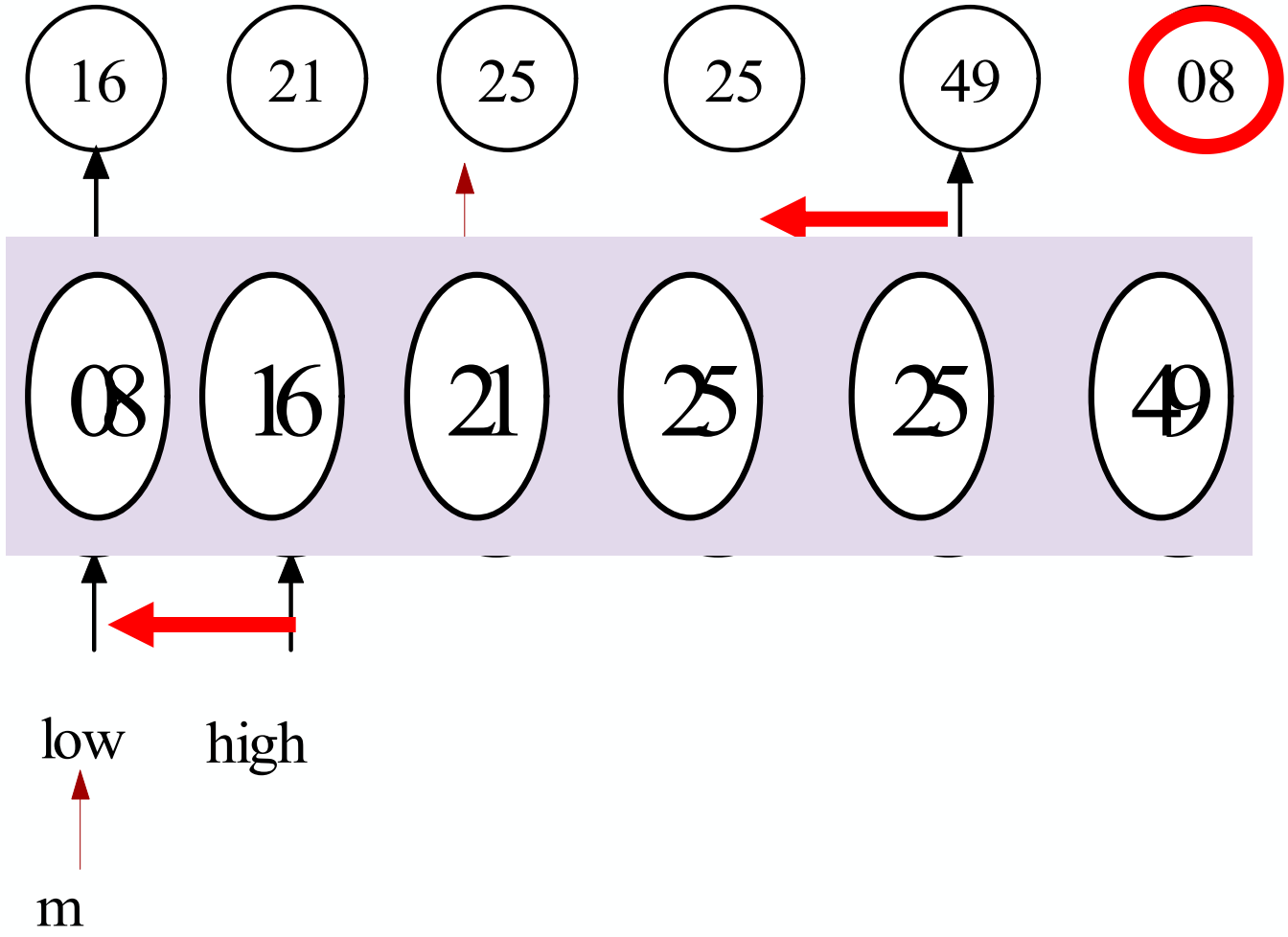


二、折半插入排序



二、折半插入排序

i=6



二、折半插入排序

```
void  BInsertSort ( SqList &L )
 { for ( i = 2;  i <= L.length ; ++i )
   {  L.r[0] = L.r[i]; low = 1 ; high = i-1 ;
     while ( low <= high )
       {  m = ( low + high ) / 2 ;
         if  (  L.r[0].key < L.r[m]. key  )  high = m -1 ;
        else  low = m + 1;
        }
     for ( j=i-1; j>=high+1; - - j ) L.r[j+1] = L.r[j];
     L.r[high+1] = L.r[0];
   }
 }  // BInsertSort
```

二、折半插入排序

1.算法分析

折半查找比顺序查找快，所以折半插入排序就平均性能来说比直接插入排序要快。

它所需要的关键码比较次数与待排序对象序列的初始排列无关，仅依赖于对象个数。在插入第 i 个对象时，需要经过 $\lfloor \log_2 i \rfloor + 1$ 次关键码比较，才能确定它应插入的位置。

二、折半插入排序

1.算法分析

当 n 较大时，总关键码比较次数比直接插入排序的最坏情况要好得多，但比其最好情况要差。

在对象的初始排列已经按关键码排好序或接近有序时，直接插入排序比折半插入排序执行的关键码比较次数要少。

折半插入排序的对象移动次数与直接插入排序相同，依赖于对象的初始排列。

二、折半插入排序　　1.算法分析

- 减少了比较次数，但没有减少移动次数；
- 平均性能优于直接插入排序。

时间复杂度为 o(n2)

空间复杂度为 o(1)

是一种稳定的排序方法

三、希尔排序

算法思想的出发点：

直接插入排序在基本有序时，效率较高；
在待排序的记录个数较少时，效率较高。

基本思想：

先将整个待排记录序列分割成若干子序列,分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行一次直接插入排序。

三、希尔排序

技巧

子序列的构成不是简单地“逐段分割”将相隔某个增量dk的记录组成一个子序列让增量dk逐趟缩短（例如依次取5,3,1）直到dk = 1为止。

优点

小元素跳跃式前移最后一趟增量为1时，序列已基本有序平均性能优于直接插入排序。

三、希尔排序

例：关键字序列 T=(49，38，65，97, 76, 13, 27, 49*，55，04)

r[i]	0	1	2	3	4	5	6	7	8	9	10
初态：		49	38	65	97	76	13	27	49*	55	04
第1趟 (dk=5)		13	27	49*	55	04	49	38	65	97	76
第2趟 (dk=3)		13	04	49*	38	27	49	55	65	97	76
第3趟 (dk=1)		04	13	27	38	49*	49	55	65	76	97

- ✓ dk 值较大，子序列中对象较少，速度较快；
- ✓ dk 值逐渐变小，子序列中对象变多，但大多数对象已基本有序，所以排序速度仍然很快。

三、希尔排序

1.希尔排序算法（主程序）

dk值依次装在dlta[t]中

```
void   ShellSort(SqList &L，int dlta[ ]，int t){
      //按增量序列dlta[0…t-1]对顺序表L作Shell排序
   for(k=0；k<t；++k)
      ShellInsert(L，dlta[k])；
       //增量为dlta[k]的一趟插入排序
}  // ShellSort
```

三、希尔排序

2.希尔排序算法（其中某一趟的排序操作）

```
void   ShellInsert(SqList &L，int dk) {
 //对顺序表L进行一趟增量为dk的Shell排序，dk为步长因子
for(i=dk+1；i<=L.length； ++ i) //开始将r[i] 插入有序增量子表
     if(r[i].key < r[i-dk].key) {
      r[0]=r[i]；            //暂存在r[0]
      for(j=i-dk; j>0 &&(r[0].key<r[j].key); j=j-dk)
   r[j+dk]=r[j]；        //关键字较大的记录在子表中后移
      r[j+dk]=r[0]；   //在本趟结束时将r[i]插入到正确位置
     }
}
```

三、希尔排序

3.算法分析

- 时间复杂度是n和d的函数：

$$O(n^{1.25}) \sim O(1.6n^{1.25})$$ —经验公式

- 空间复杂度为 $o(1)$
- 一种不稳定的排序方法

✓如何选择最佳d序列，目前尚未解决；

✓最后一个增量值必须为1，无除1以外的公因子；

✓不宜在链式存储结构上实现。

小结

1. 直接插入排序
2. 折半插入排序
3. 希尔排序