



第十章 排序

概述

- 排序(Sorting)是计算机程序设计中的一种重要操作，其功能是对一个数据元素集合或序列重新排列成一个按数据元素关键字的值有序的序列。

学号	姓名	数学
0512114	张卓	114
0512087	王力	98
0512096	李华	123

学号	姓名	数学
0512087	王力	98
0512096	李华	123
0512114	张卓	114

数据元素集合



按照学号排序



概述

- 关键字的取值情况，会使排序结果可能不唯一。
- 按照**主**关键字排序，排序结果**唯一**；按照**次**关键字排序，排序结果**可能不唯一**。
- 若对任意的数据元素序列，使用某个排序方法，对它按关键字进行排序：若相同关键字元素间的位置关系，排序前与排序后保持一致，称此排序方法是**稳定**的；而不能保持一致的排序方法则称为**不稳定**的。



学号	姓名	数学
0512114	张卓	114
0512087	王力	98
0512096	李华	114

数据元素集合

学号	姓名	数学
0512087	王力	98
0512096	李华	114
0512114	张卓	114

按数学成绩排序

排序前后数学成绩为114分的两个是数据元素李华和张卓的相对位置发生了变化

不稳定!



排序分类

- 内排序和外排序。
- 内排序：指待排序列完全存放在内存中所进行的排序过程，适合不太大的元素序列。
- 外排序：指排序过程中还需访问外存储器，足够大的元素序列，因不能完全放入内存，只能使用外排序。
- 本章只介绍内排序



排序分类

- 根据排序策略：

- 1 插入排序

- 2 选择排序

- 3 交换排序

- 4 归并排序

- 5 计数排序



排序分类

■ 根据排序过程所需的工作量区分：

1 简单的排序方法： $O(n^2)$

2 先进的排序方法： $O(n \log n)$

3 基数排序： $O(d.n)$

通常，排序（除基数排序外）需要两种基本操作：

(1) 比较两个关键字的大小；

(2) 将数据元素从一个位置移动至另一个位置。

排序算法（除基数排序外）的时间效率：主要看比较和数据移动的次数

待排序数据元素的存放方式:

typedef struct

{

ElemType r[Max];//待排序的数据存放于数组r

int length;

}SqList;

SqList L;

L.r[0].key

L.length==3

L.r[0]	0512114	张卓	114
L.r[1]	0512087	王力	98
L.r[2]	0512096	李华	123



插入排序

- **基本思想**：将待排序的数据按其关键字的大小依次插入到前面已排序的数据序列的适当位置上。
- **常见的插入排序方法**：
直接插入排序，二分插入排序，希尔排序



直接插入排序

方法：在插入第 i 个数据元素时， R_1, R_2, \dots, R_{i-1} 已经排序，用**顺序查找**方法找出 R_i 应该插入的位置，将 R_i 插入。


- 53 27 36 15 69 42
- 第一趟排序：27 53 36 15 69 42
- 第二趟排序：27 36 53 15 69 42
- 第三趟排序：15 27 36 53 69 42
- 第四趟排序：15 27 36 53 69 42
- 第五趟排序：15 27 36 42 53 69

n 个数据元素 $n-1$ 趟直接插入排序完成排序

n个数据元素n-1趟直接插入排序完成排序



直接插入排序



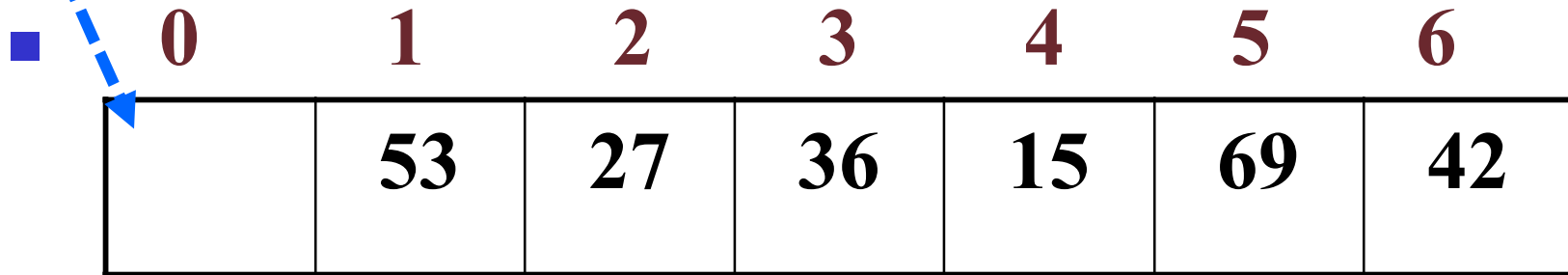
0	1	2	3	4	5	6
53	27	36	15	69	42	

插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处——哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序

哨兵



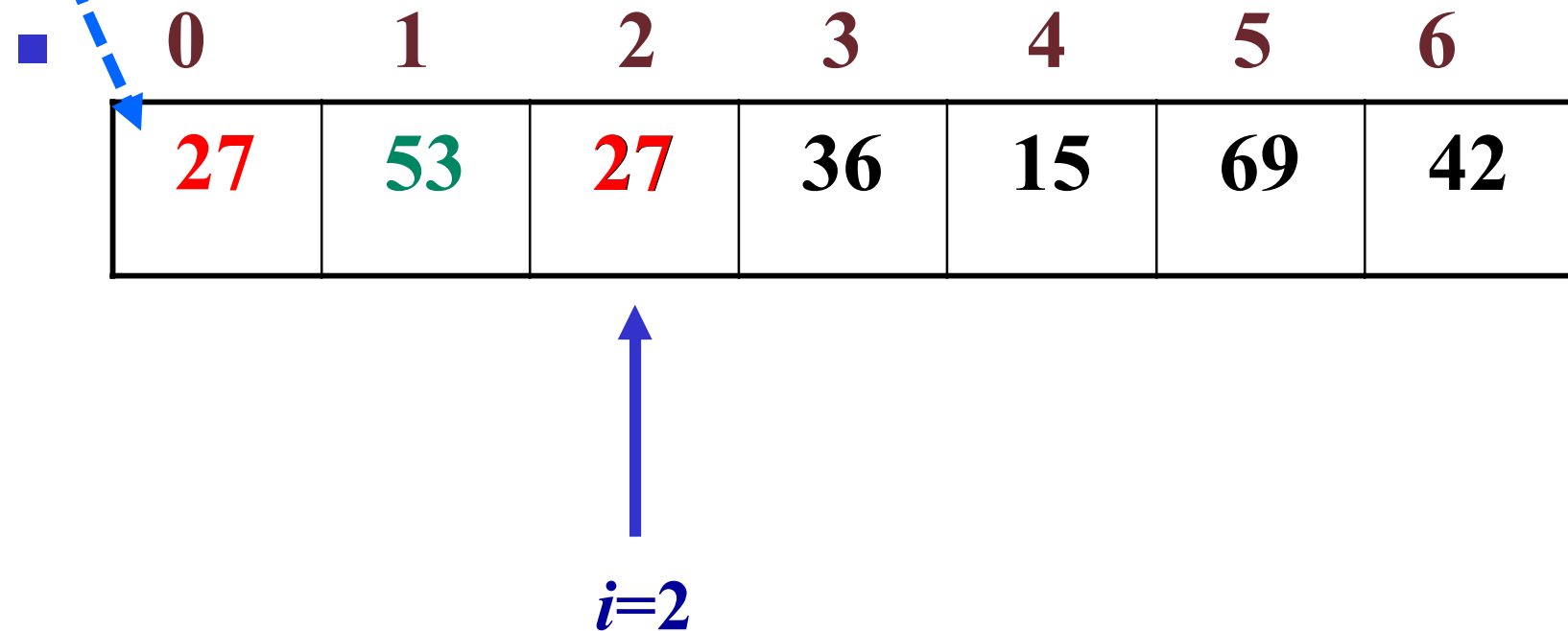
0	1	2	3	4	5	6
	53	27	36	15	69	42

插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处——哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第一趟

哨兵 将27插入已经排好序的[53]中

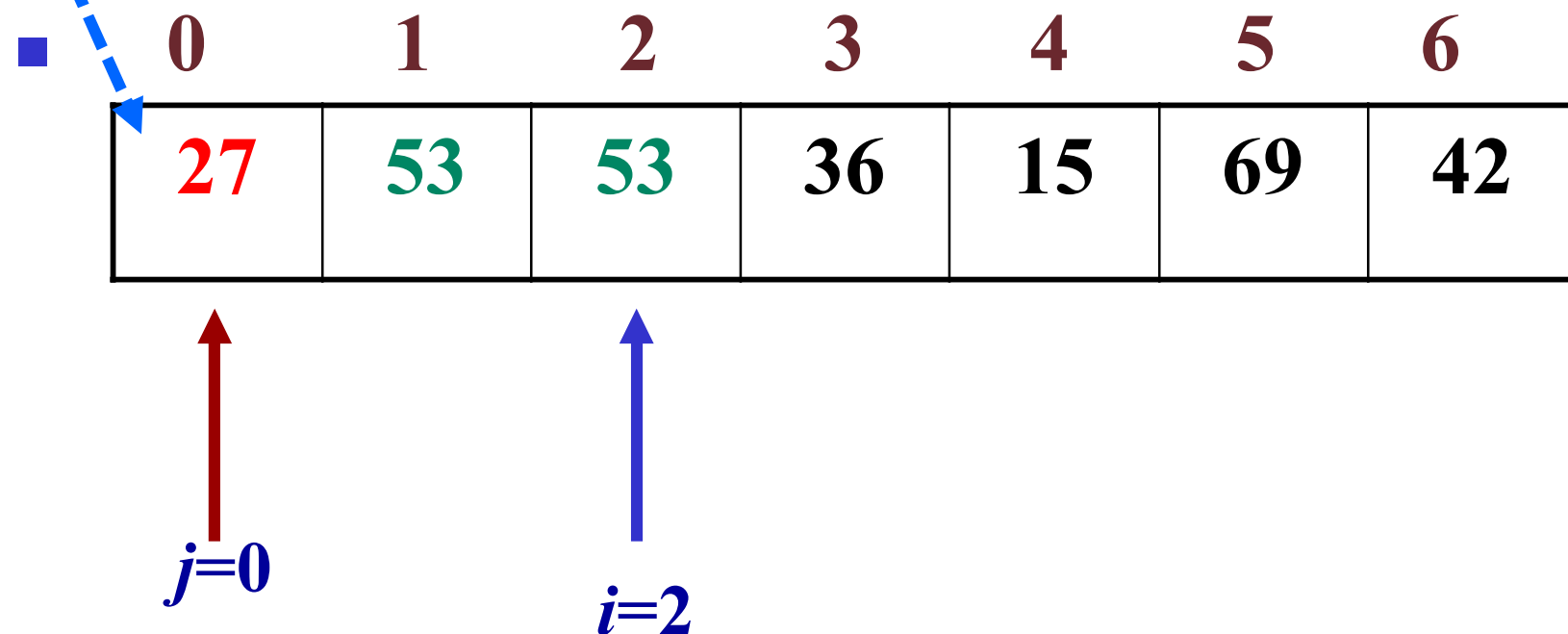


插入第 i 个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第一趟

哨兵 将27插入已经排好序的[53]中

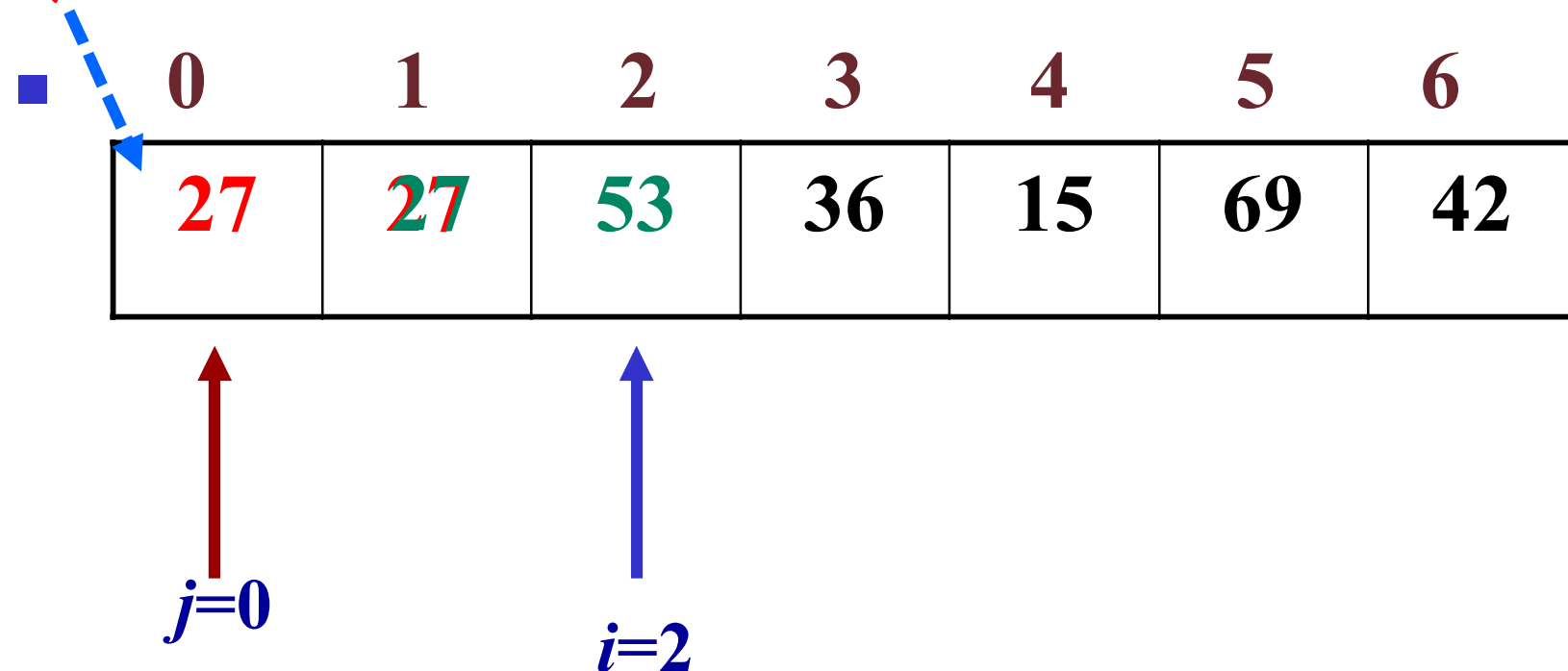


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序——第一趟

哨兵 将27插入已经排好序的[53]中

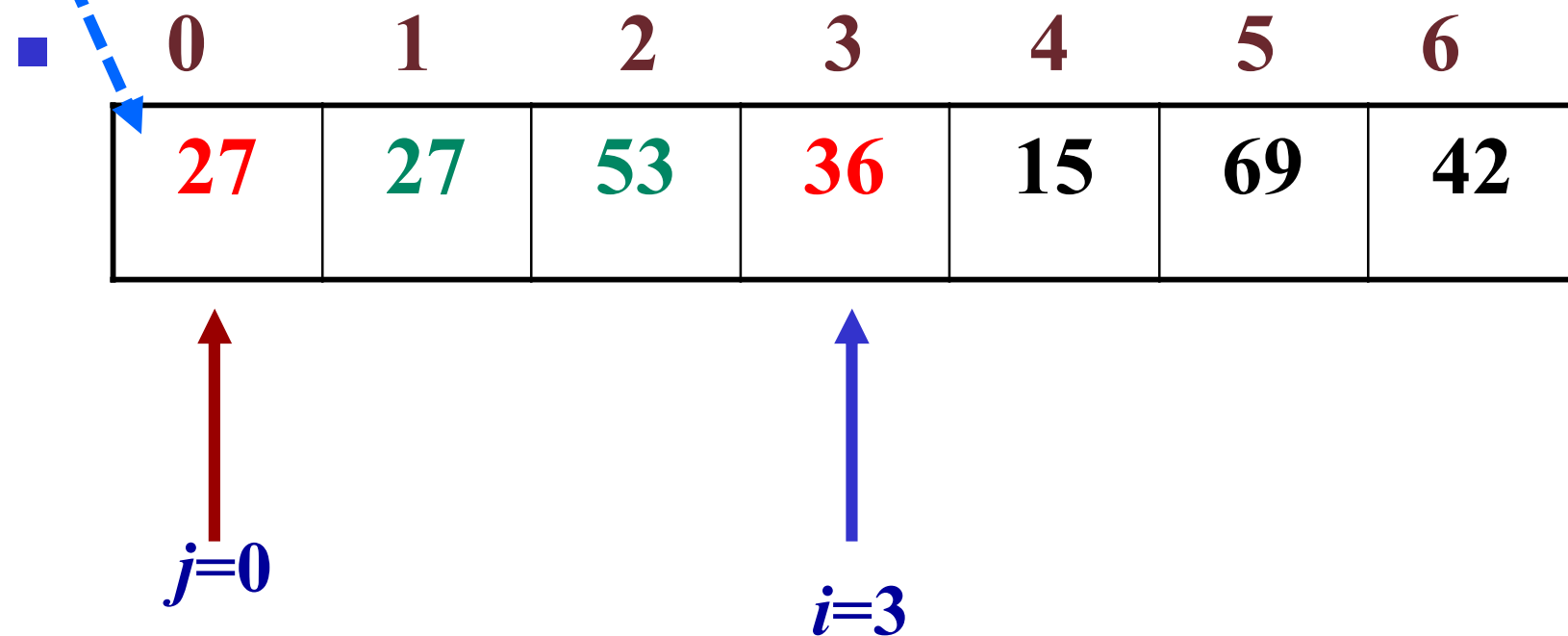


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处——哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第二趟

哨兵 将36插入已经排好序的[27, 53]中

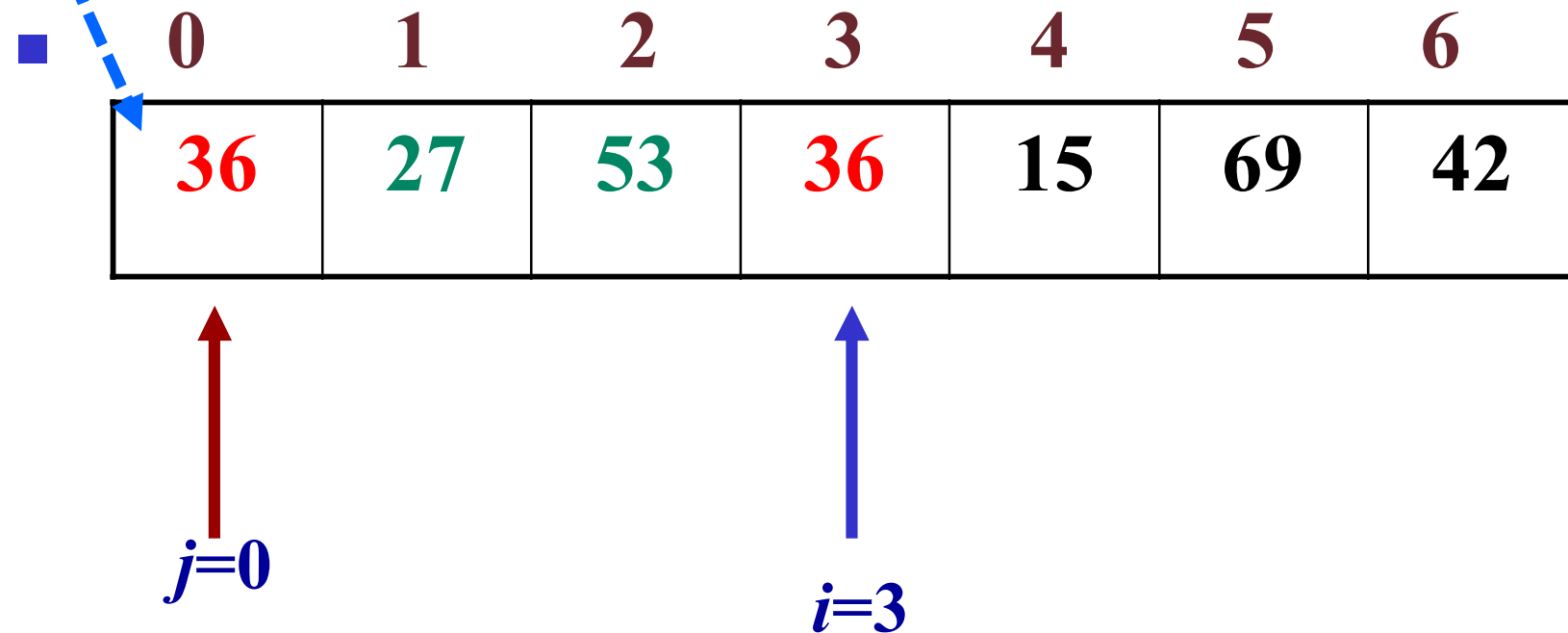


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第二趟

哨兵 将36插入已经排好序的[27, 53]中

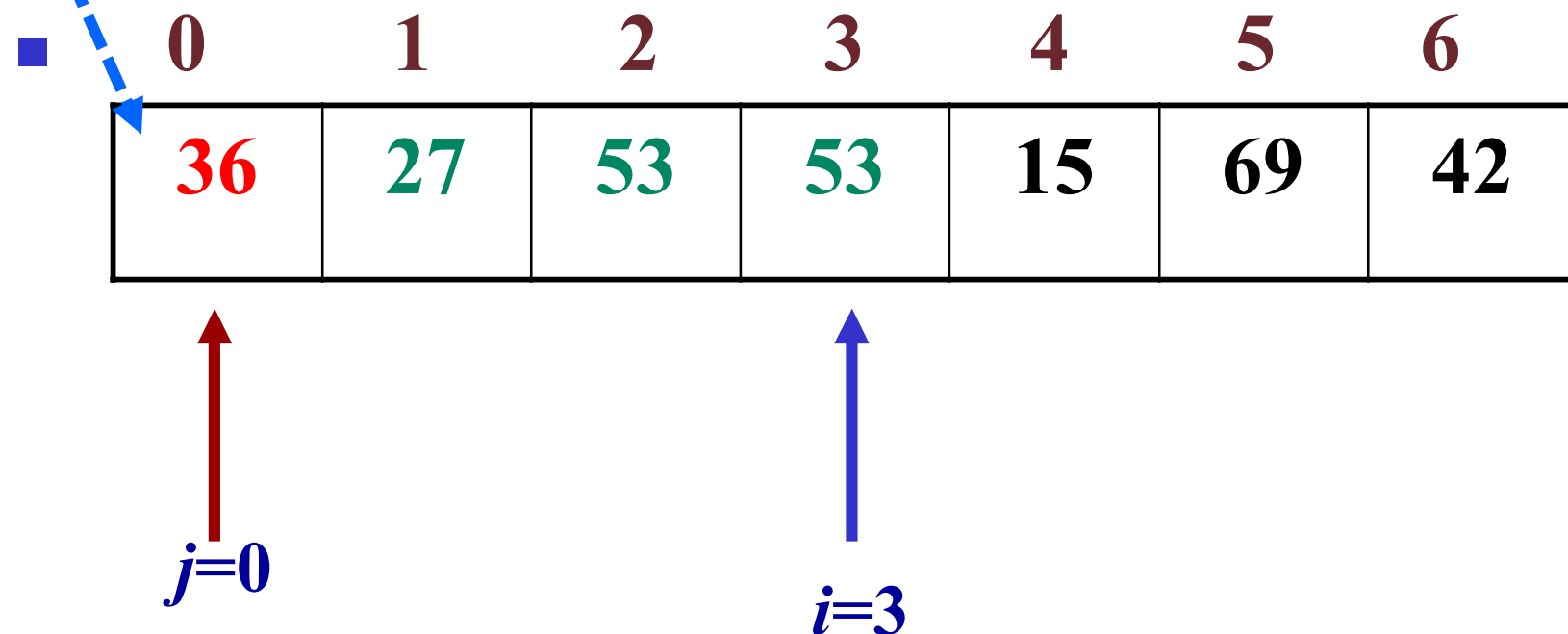


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第二趟

哨兵 将36插入已经排好序的[27, 53]中

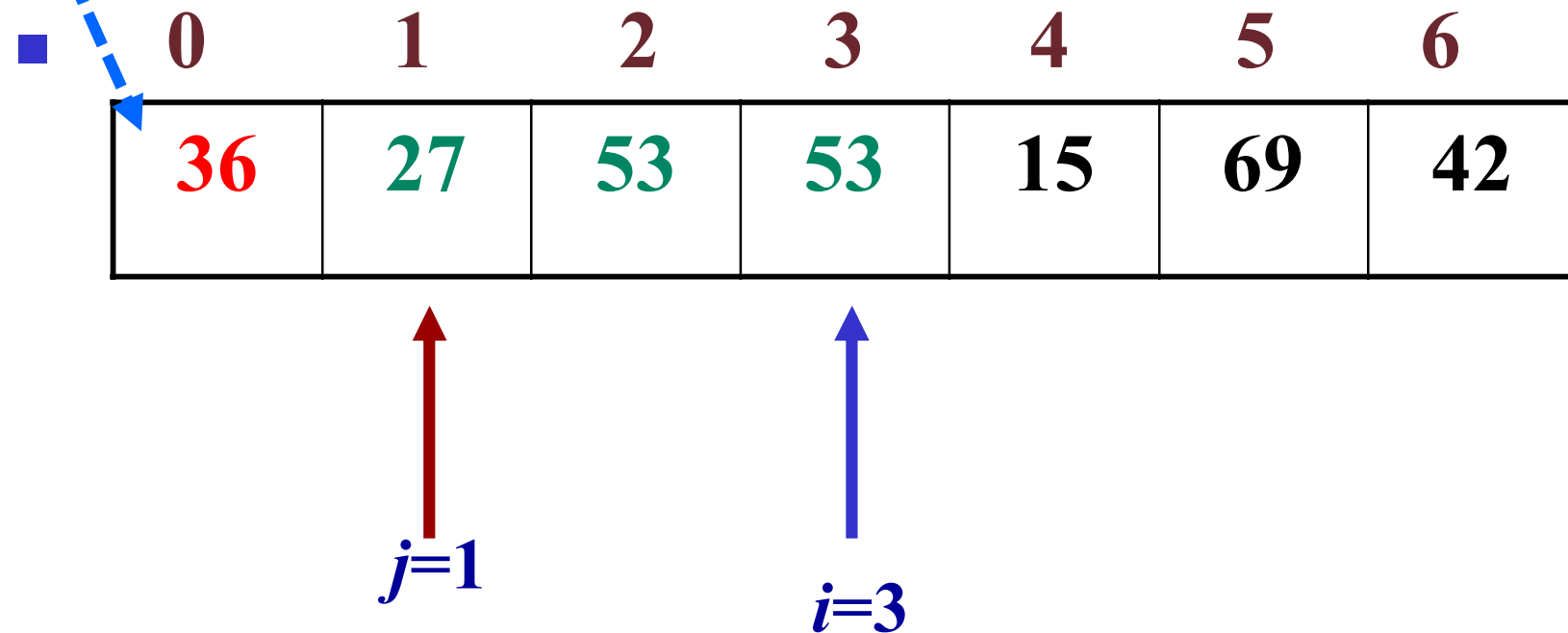


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第二趟

哨兵 将36插入已经排好序的[27, 53]中

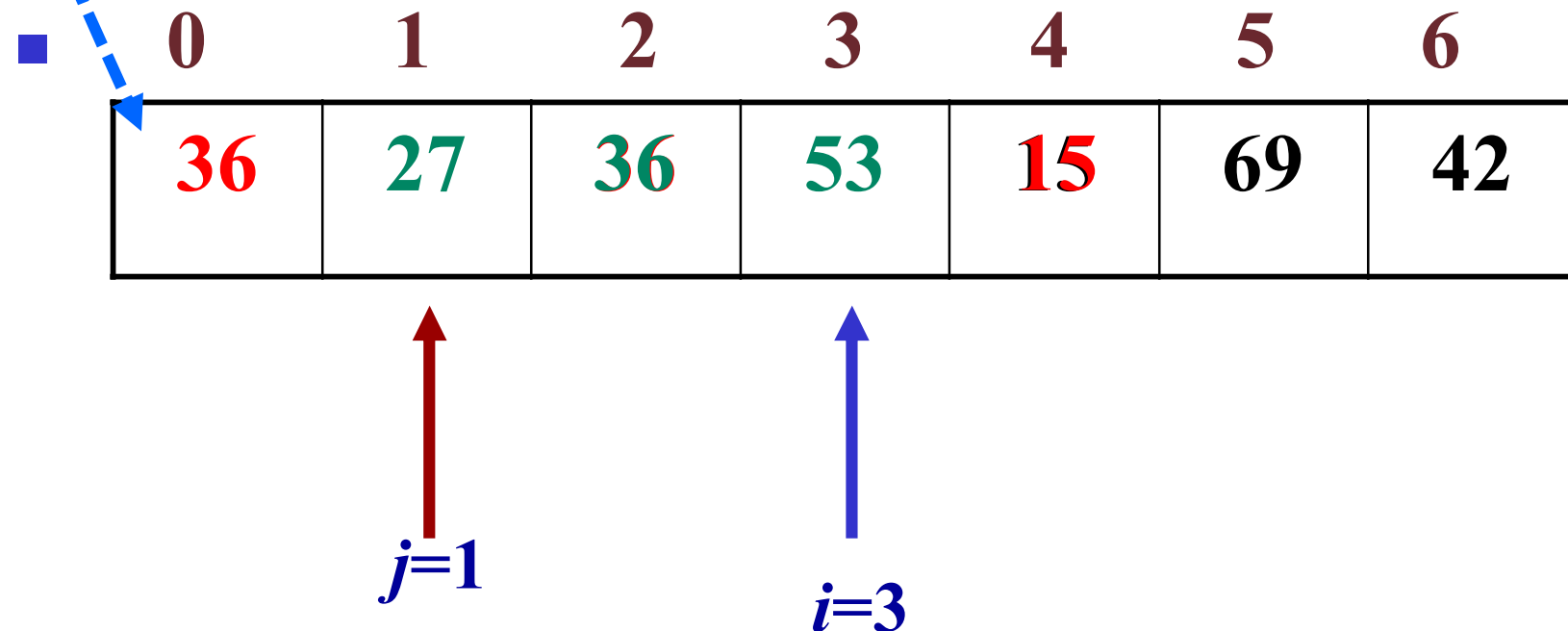


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第二趟

哨兵 将36插入已经排好序的[27, 53]中

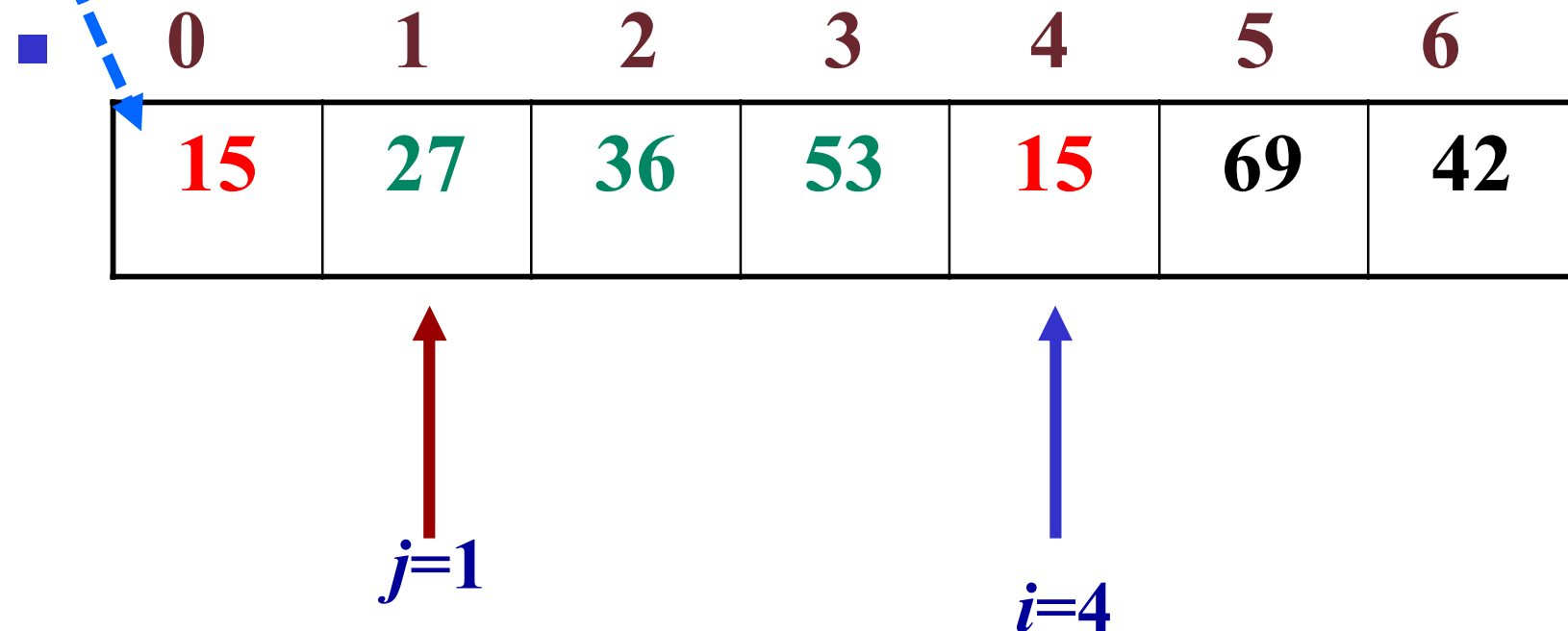


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

哨兵 将15插入已经排好序的[27, 36, 53]中

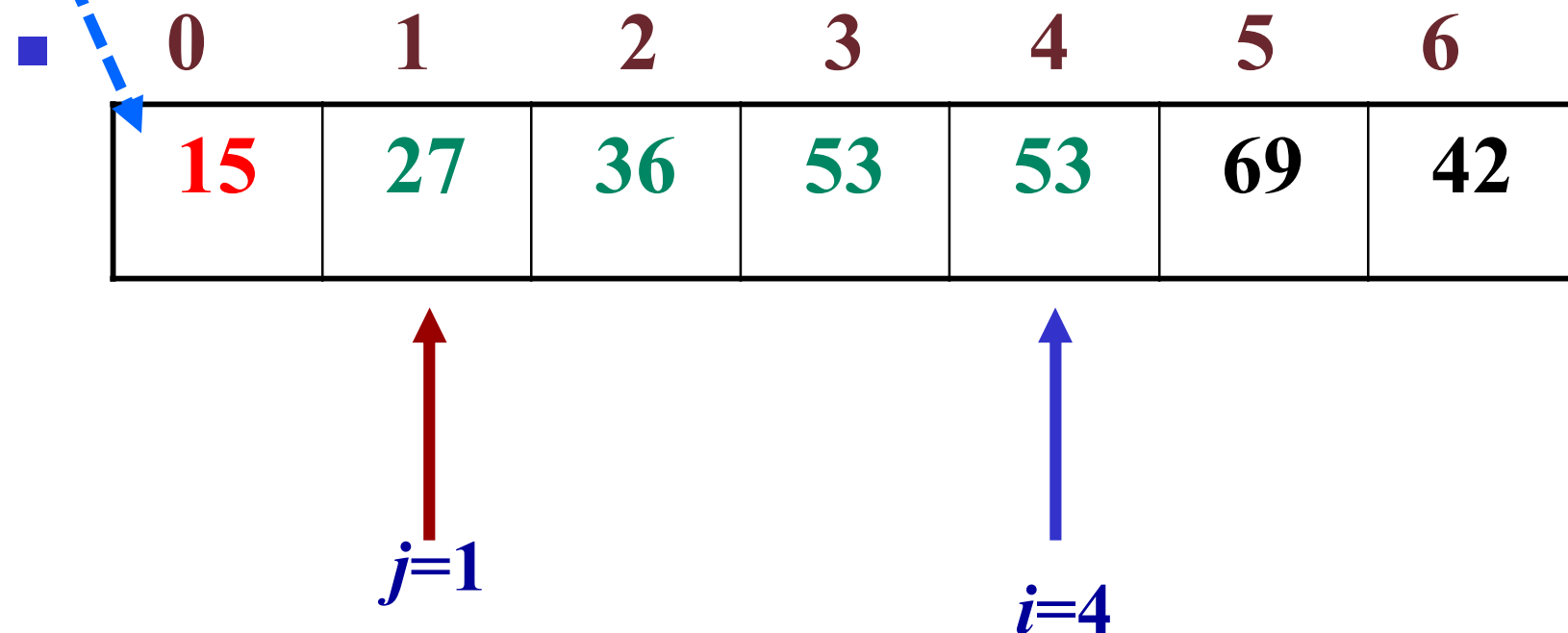


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

哨兵 将15插入已经排好序的[27, 36, 53]中



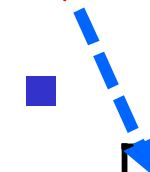
插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

将15插入已经排好序的[27, 36, 53]中

哨兵



0	1	2	3	4	5	6
15	27	36	53	53	69	42

$j=2$

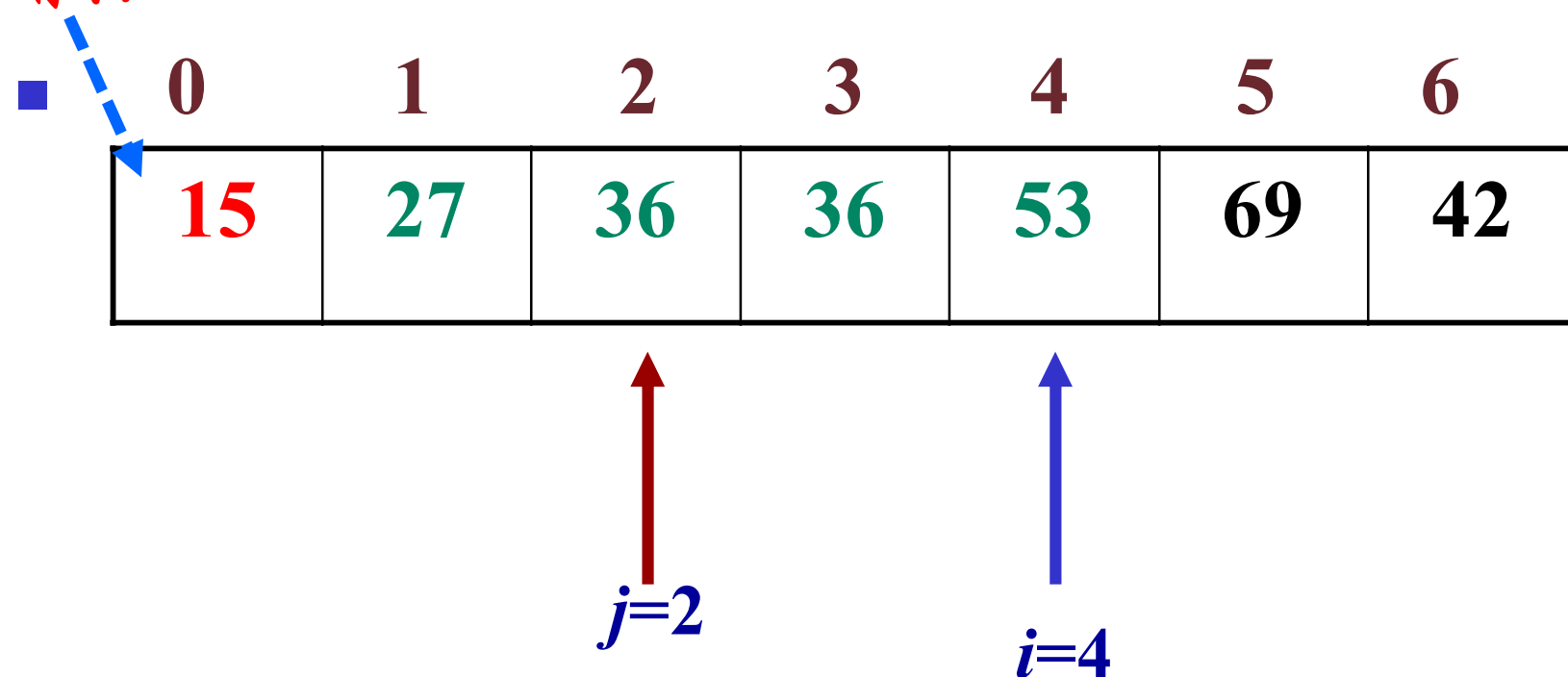
$i=4$

插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

哨兵 将15插入已经排好序的[27, 36, 53]中

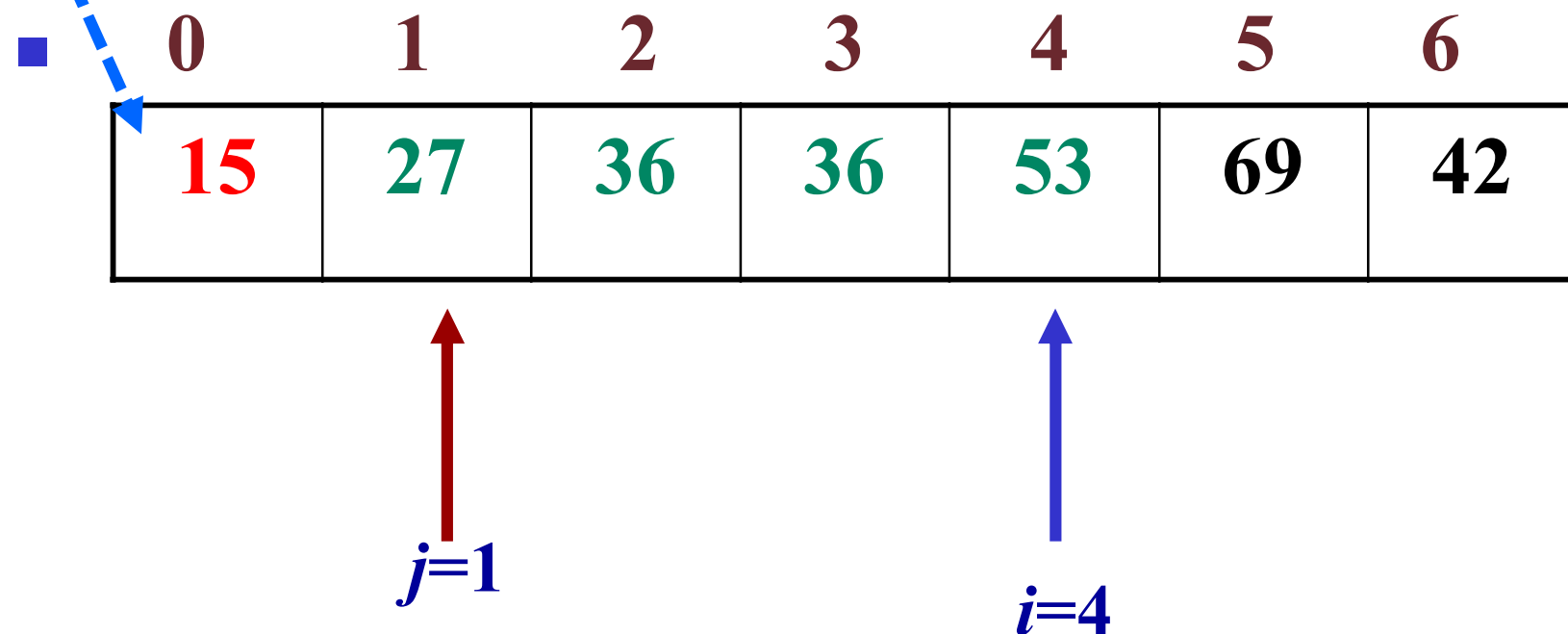


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

哨兵 将15插入已经排好序的[27, 36, 53]中

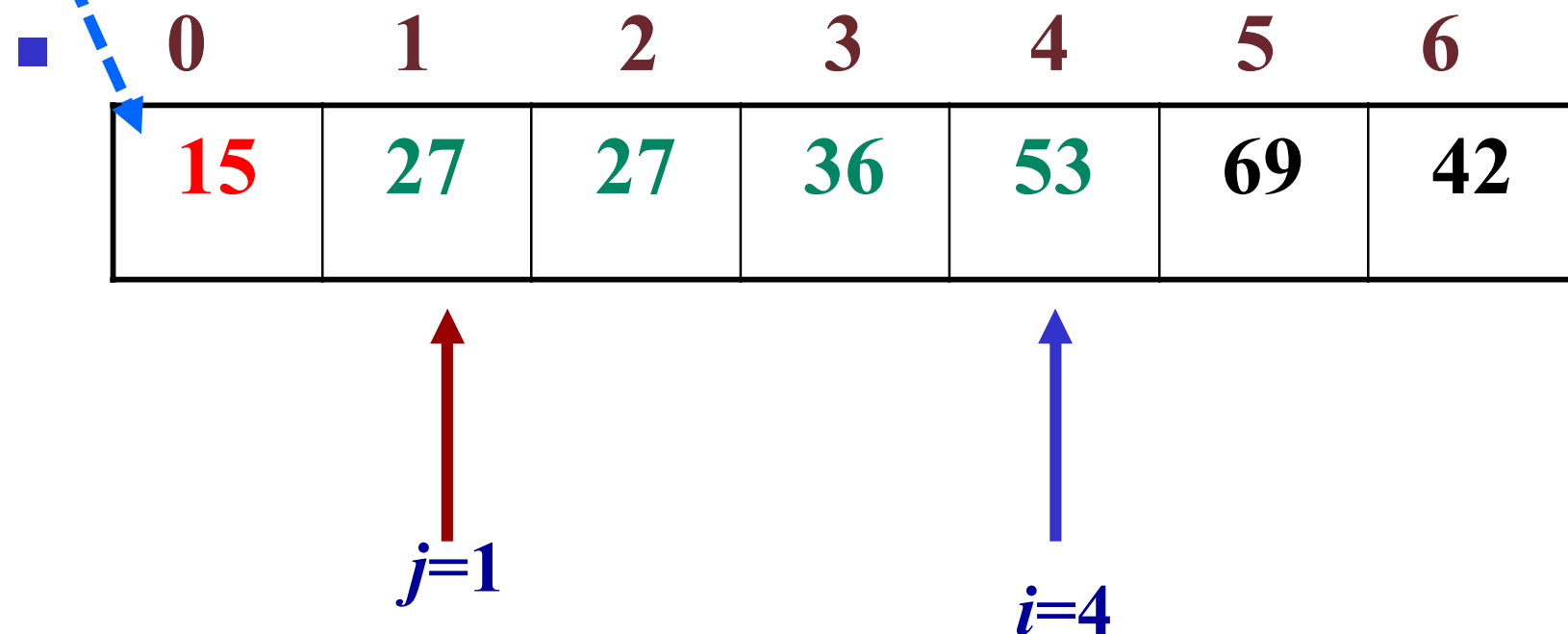


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

哨兵 将15插入已经排好序的[27, 36, 53]中

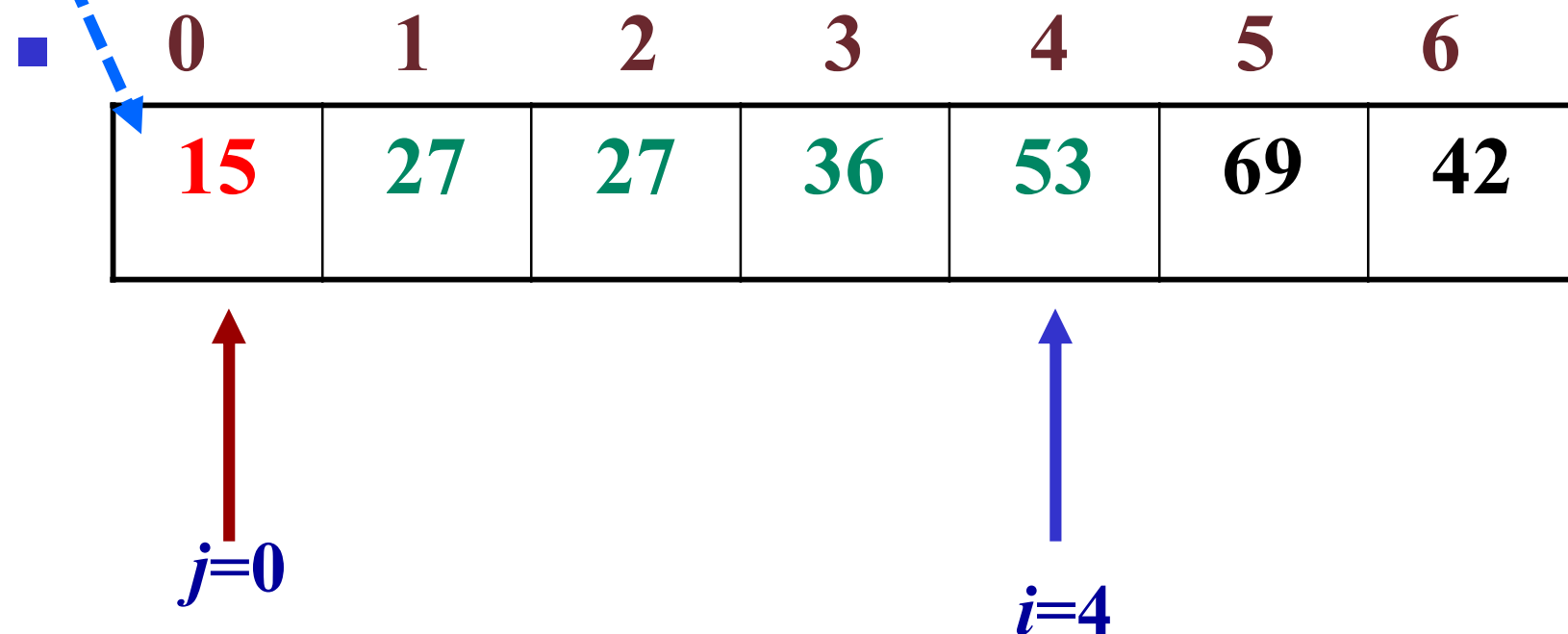


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

哨兵 将15插入已经排好序的[27, 36, 53]中

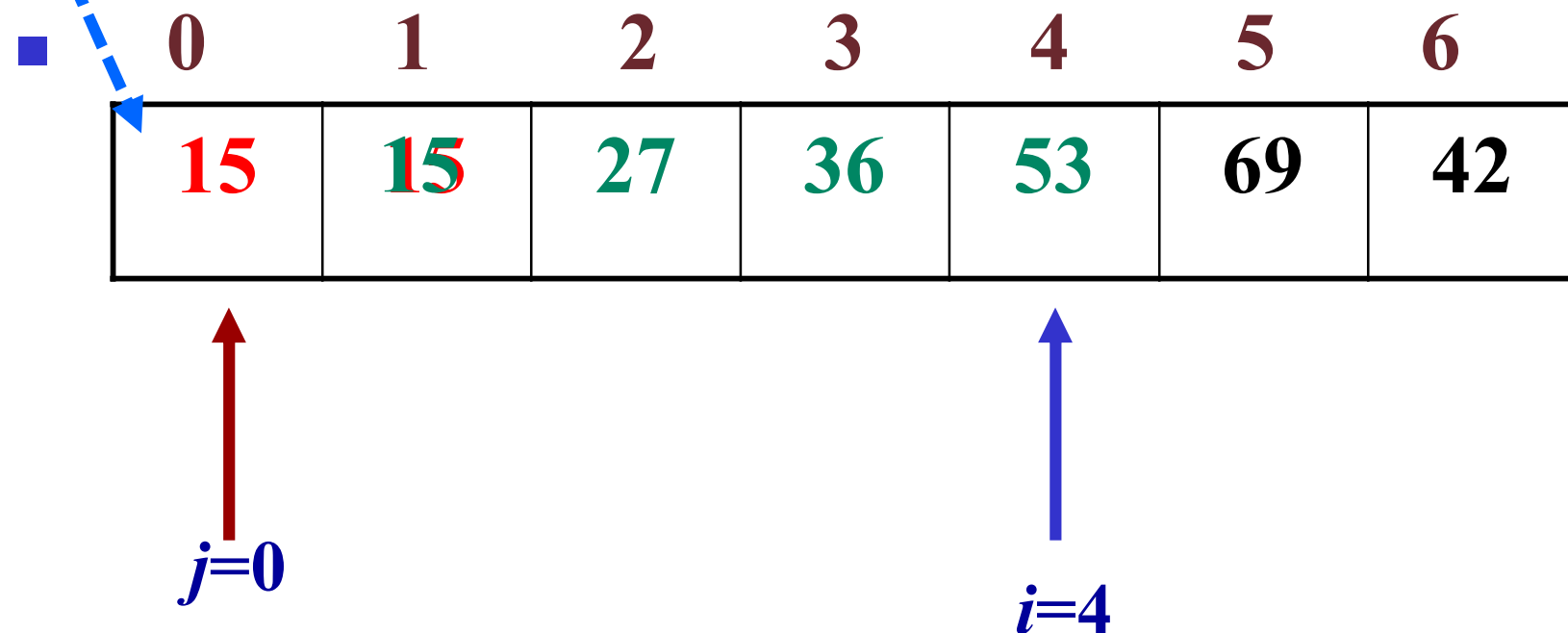


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第三趟

哨兵 将15插入已经排好序的[27, 36, 53]中

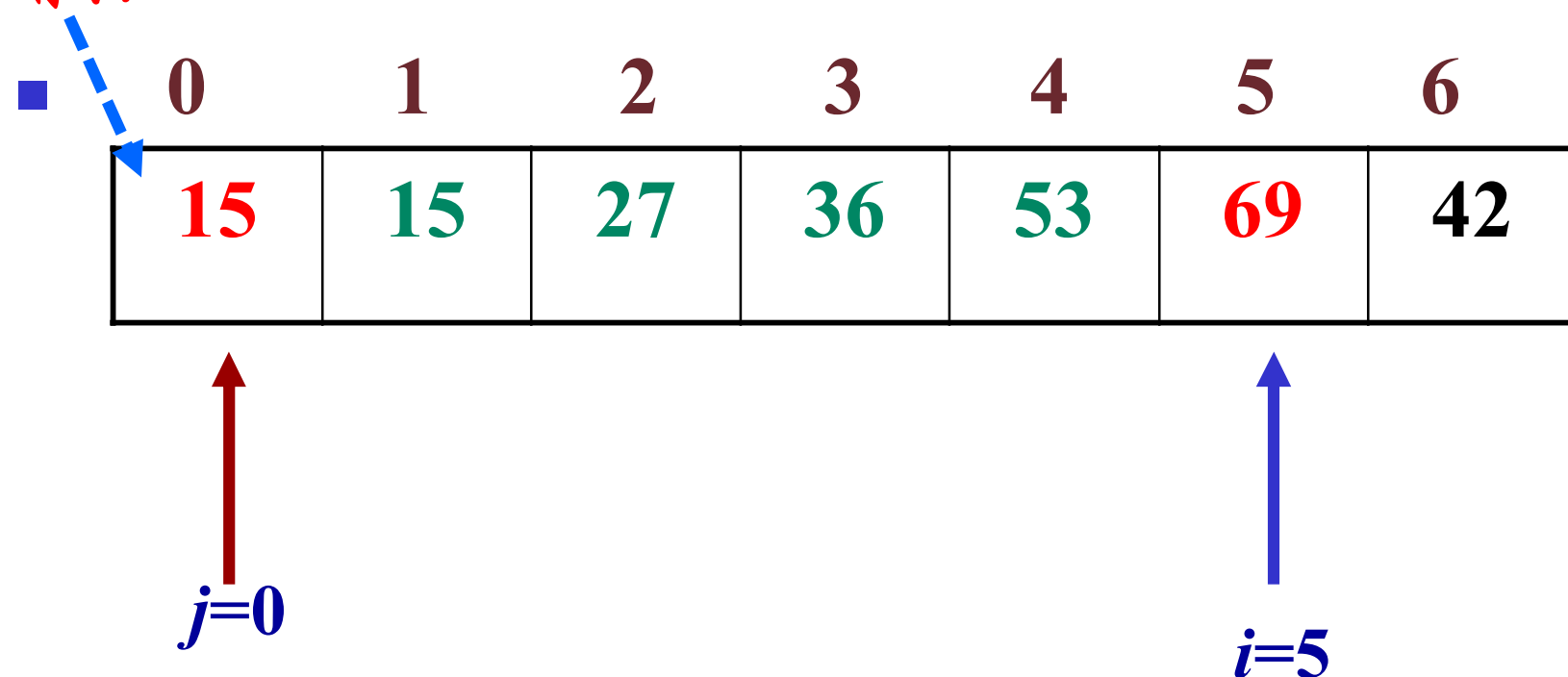


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第四趟

哨兵 将69插入已经排好序的[15, 27, 36, 53]中

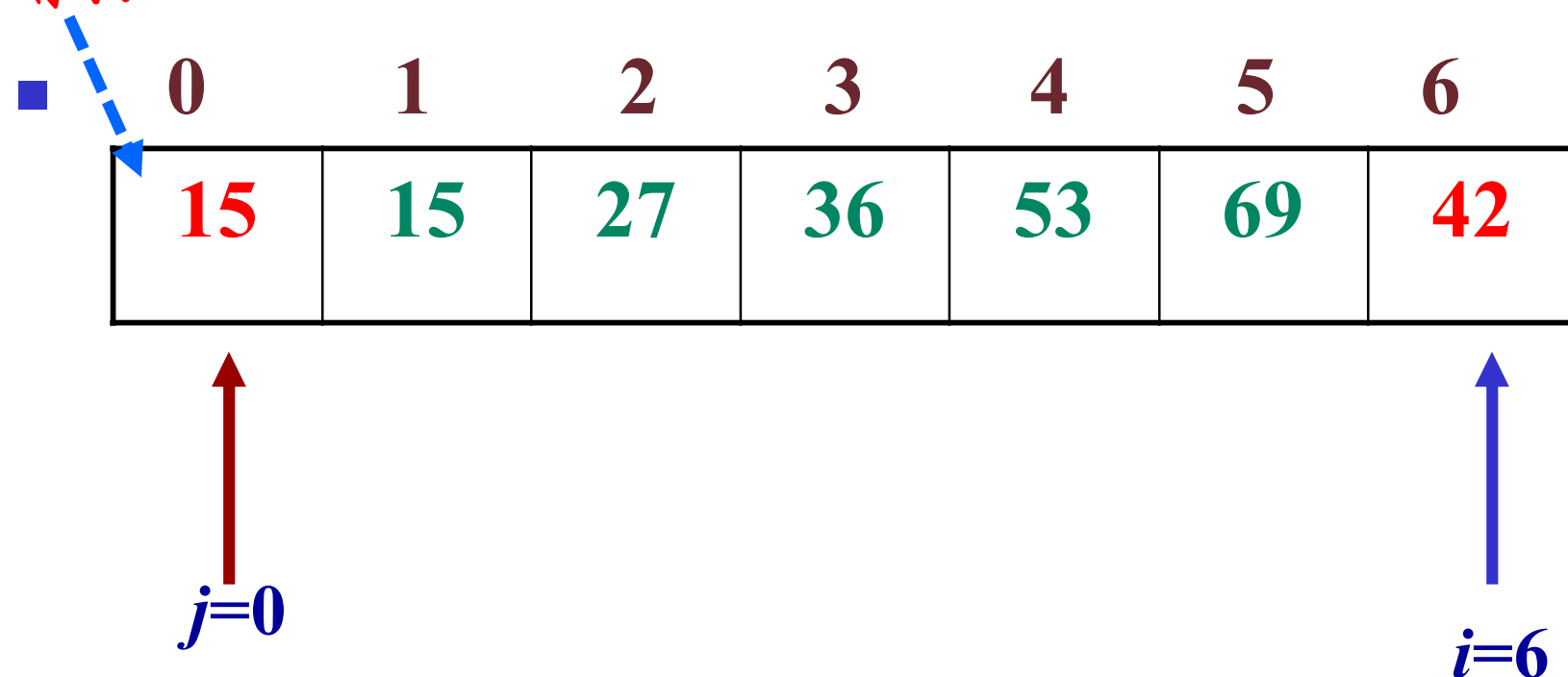


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第五趟

哨兵 将42插入已经排好序的[15, 27, 36, 53, 60]中



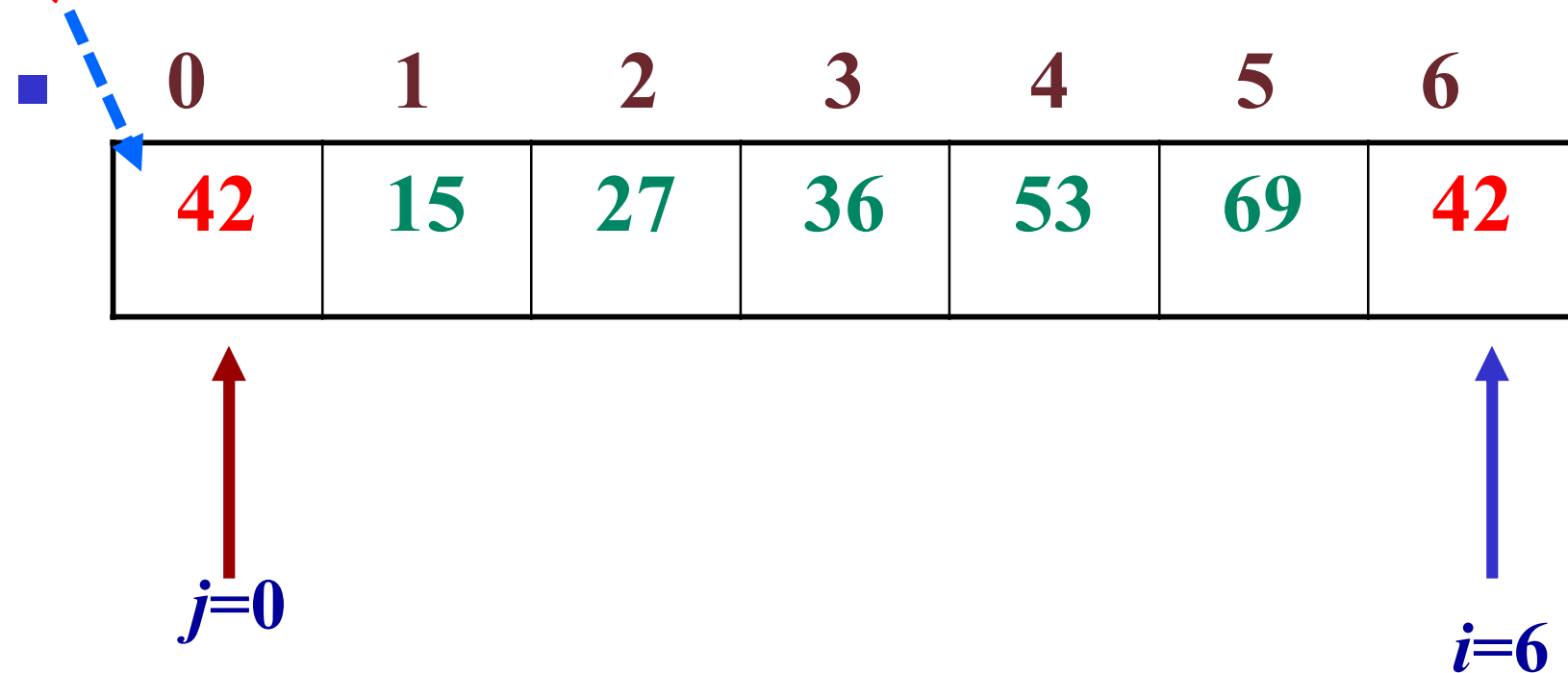
插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第五趟

将42插入已经排好序的[15, 27, 36, 53, 69]中

哨兵

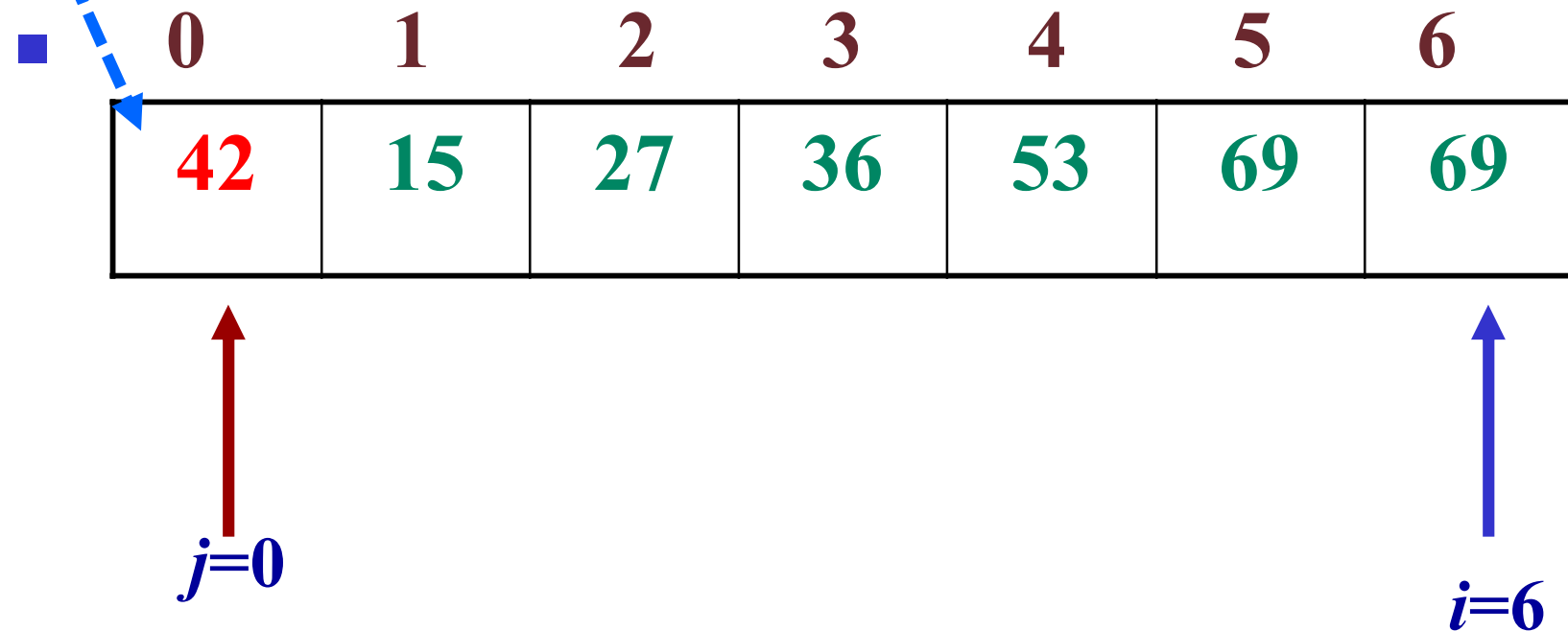


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第五趟

哨兵 将42插入已经排好序的[15, 27, 36, 53, 69]中

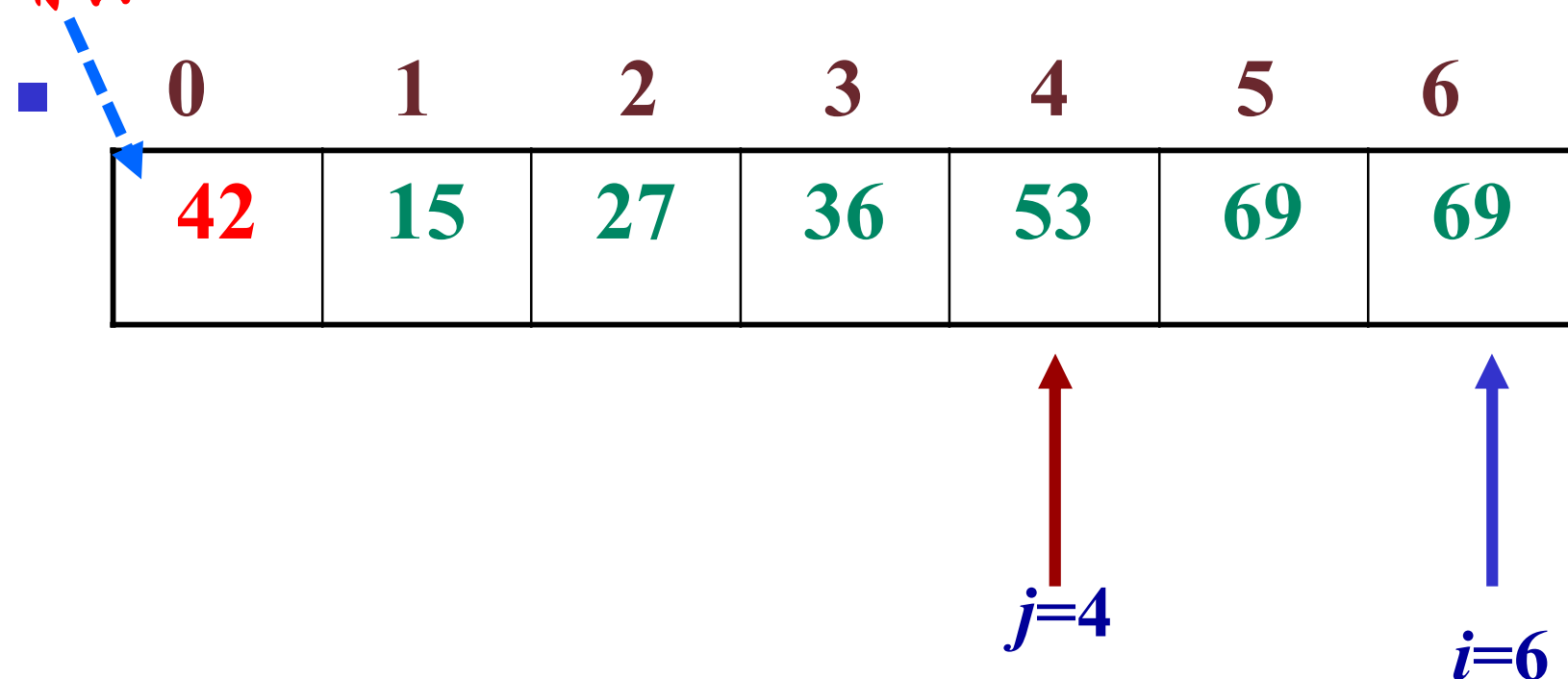


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第五趟

哨兵 将42插入已经排好序的[15, 27, 36, 53, 69]中

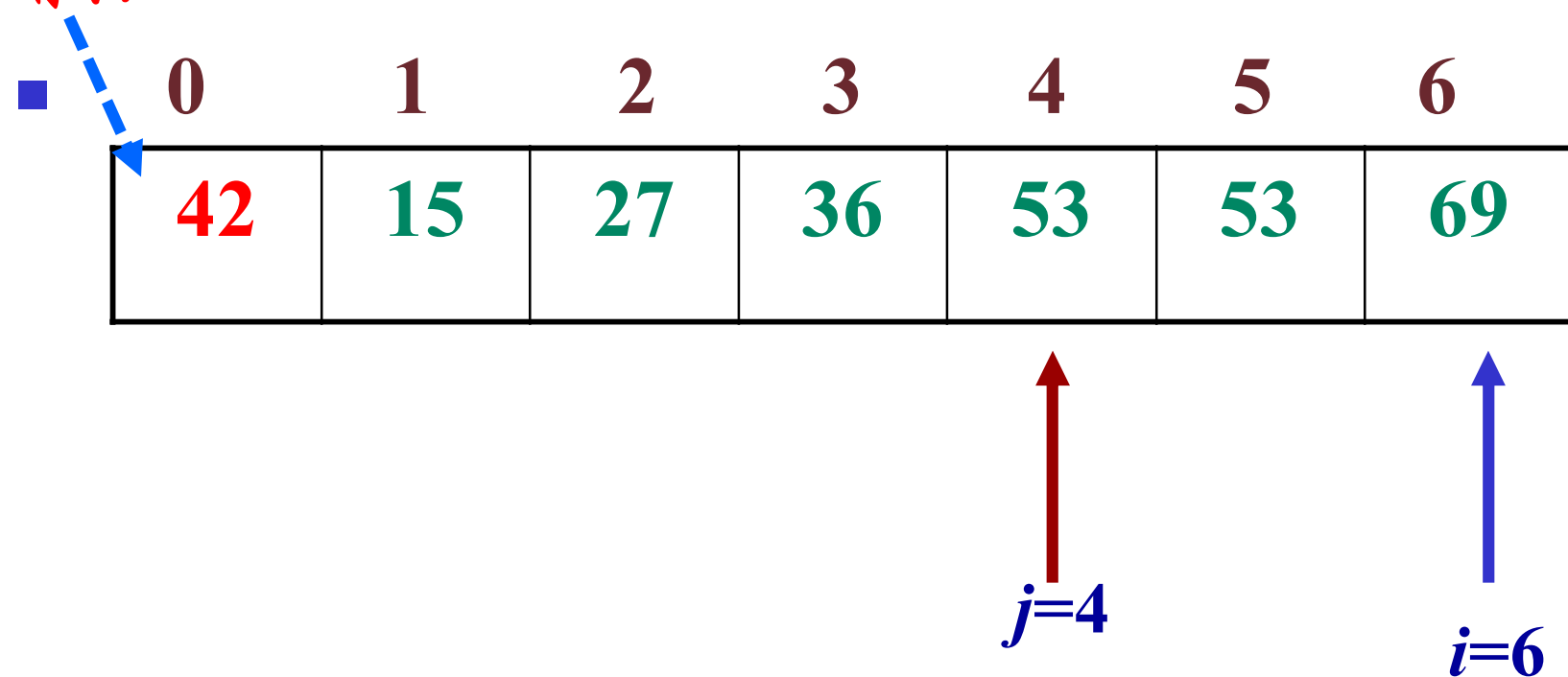


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第五趟

哨兵 将42插入已经排好序的[15, 27, 36, 53, 69]中

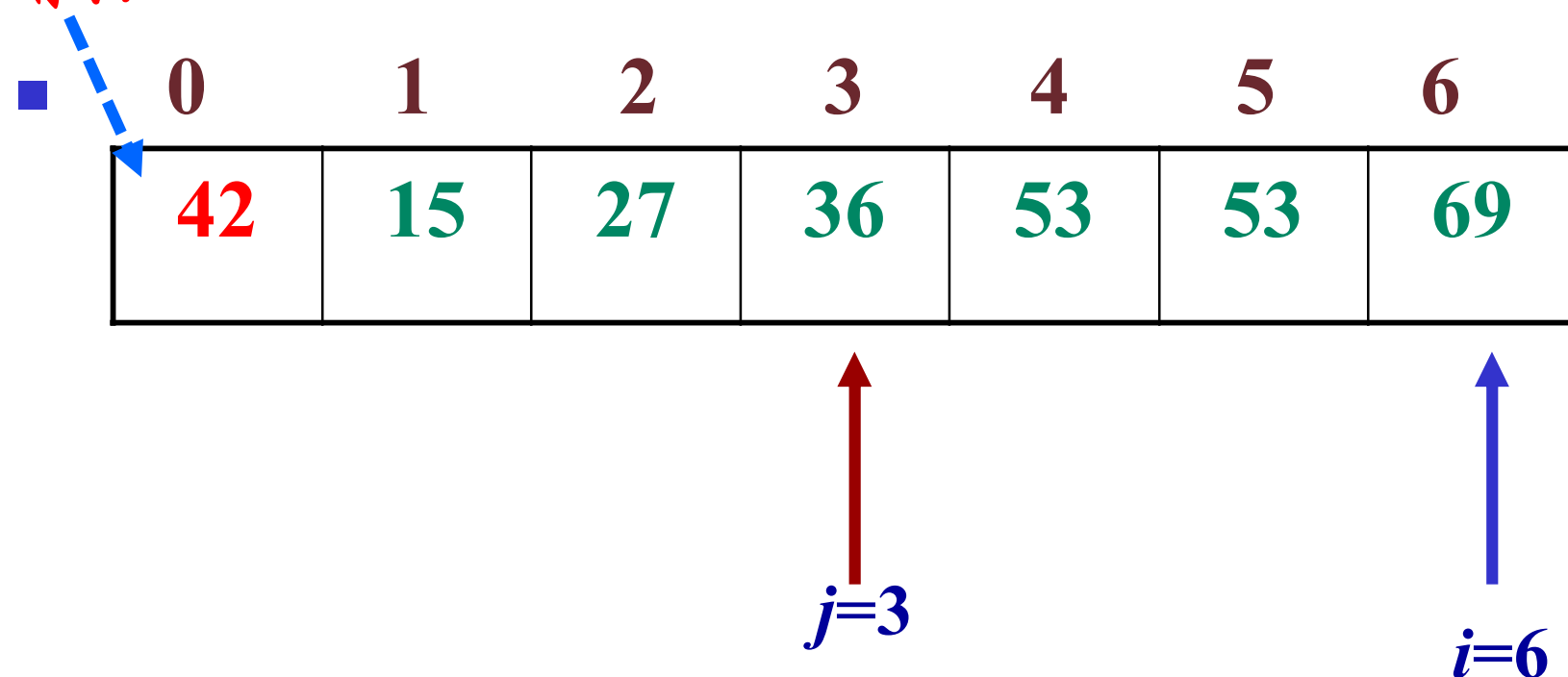


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第五趟

哨兵 将42插入已经排好序的[15, 27, 36, 53, 69]中

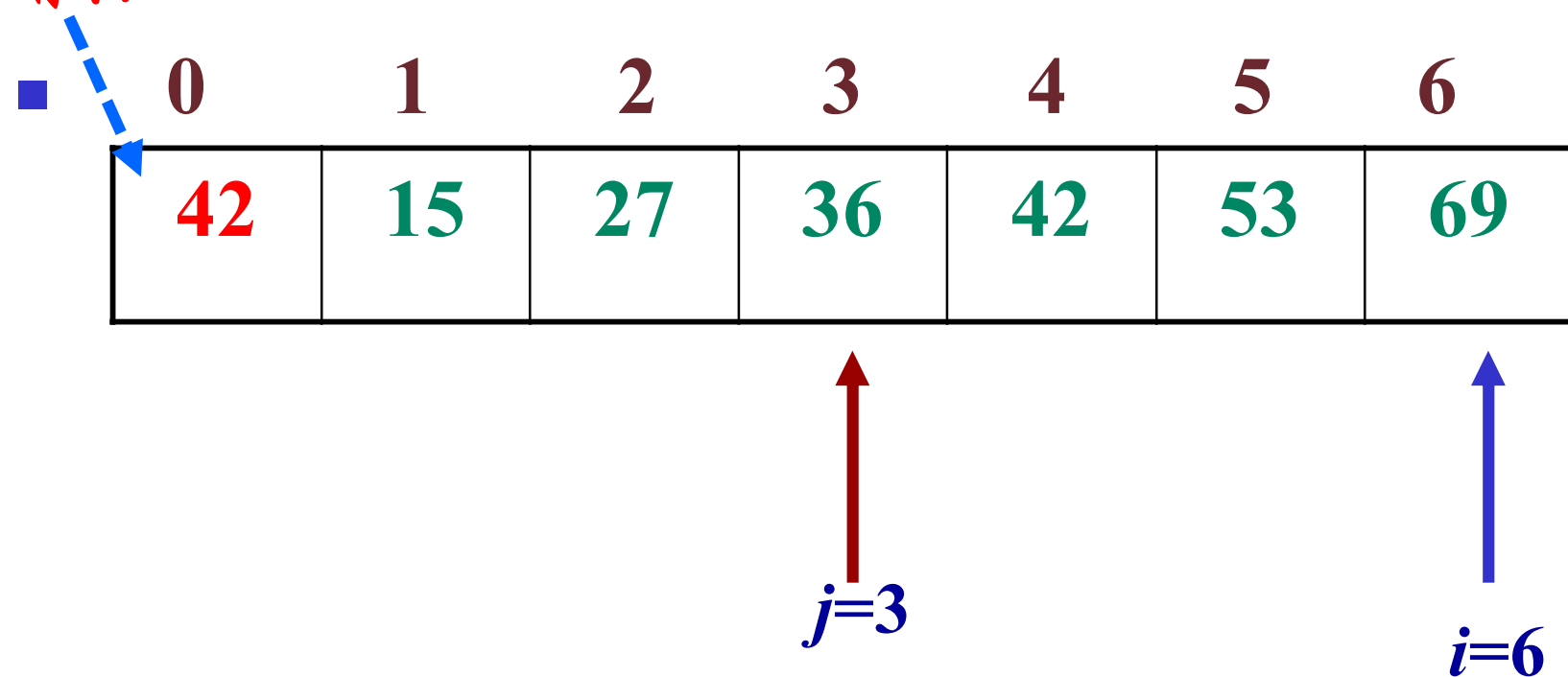


插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项

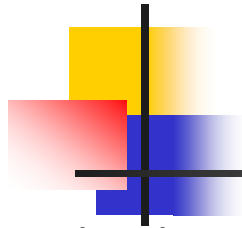
n个数据元素n-1趟直接插入排序完成排序

直接插入排序—第五趟

哨兵 将42插入已经排好序的[15, 27, 36, 53, 69]中



插入第*i*个数据元素时，为了方便查找合适的插入位置，待排序的数据元素从下标为1的数组元素开始存放。下标为0处—哨兵项



直接插入排序

```
void insertSort(SqList &L)
```

```
{  int i, j;
```

```
    for(i=2; i<=L.length; i++)//从第2个数据开始插入, n=L.length
```

```
    if(L.r[i].key<L.r[i-1].key) //第i个数据比前面已经有序的i-1个数据最大的小
```

```
    {  L.r[0]=L.r[i]; //将第i个数据放入哨兵位置
```

```
        L[i]=L.r[i-1];
```

```
        for( j=i-2; L.r[0].key<L.r[j].key; --j )//L.r[0]存放的是此次要插入的第i个数据
```

```
        L.r[j+1]=L.r[j];
```

```
        L.r[j+1]=L.r[0];  }
```

```
}
```



直接插入排序

- 直接插入排序是一个**稳定**的排序方法
- 一个额外的辅助空间, $O(1)$ 。
- 对于有 **n** 个数据元素的待排序序列, 插入操作要进行 **$n-1$** 趟。
- **最好**情况: **$n-1$** 次数据比较, **0** 次数据移动。待排序的数据已经有序。
-



直接插入排序

- 最坏情况：待排序的数据逆序。

i	比较次数	移动次数
2	2	3
3	3	4
.....		
n	n	$n+1$

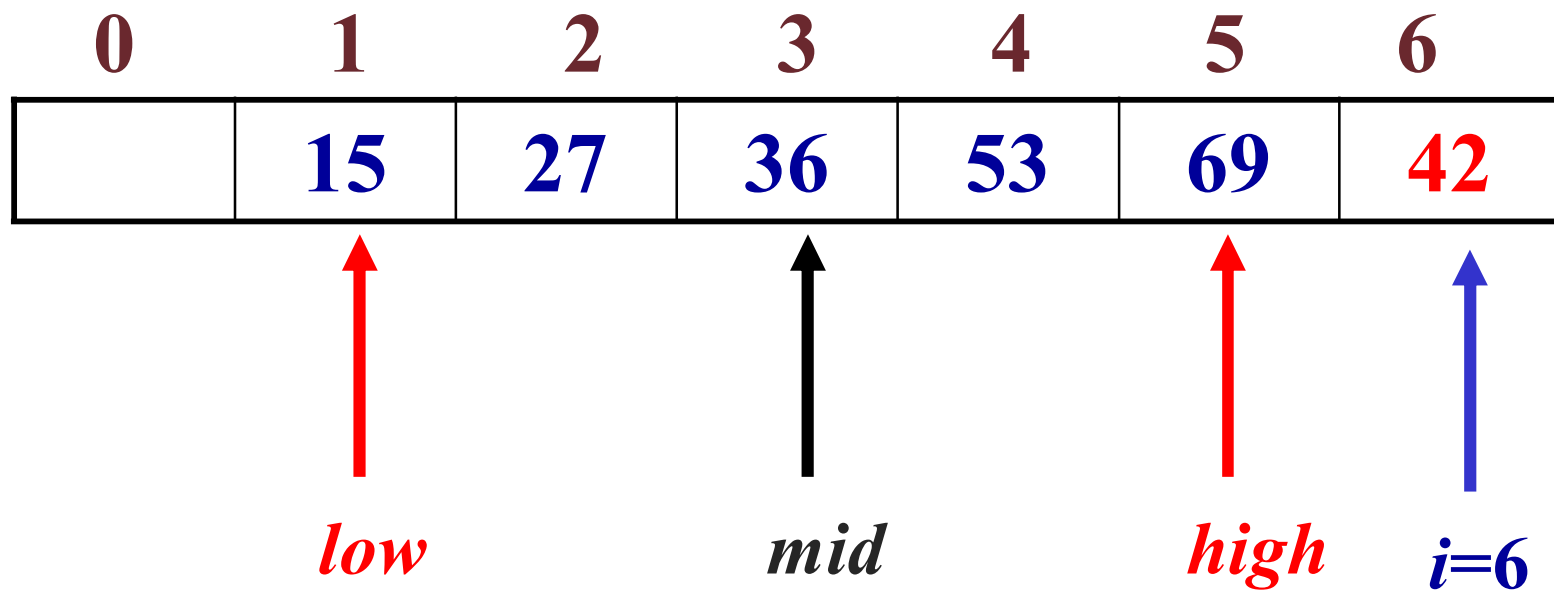
- 数据比较次数= $(n+2)(n-1)/2$
- 数据移动次数= $(n+4)(n-1)/2$
- 时间复杂度为 $O(n^2)$
- 数组和链表均可



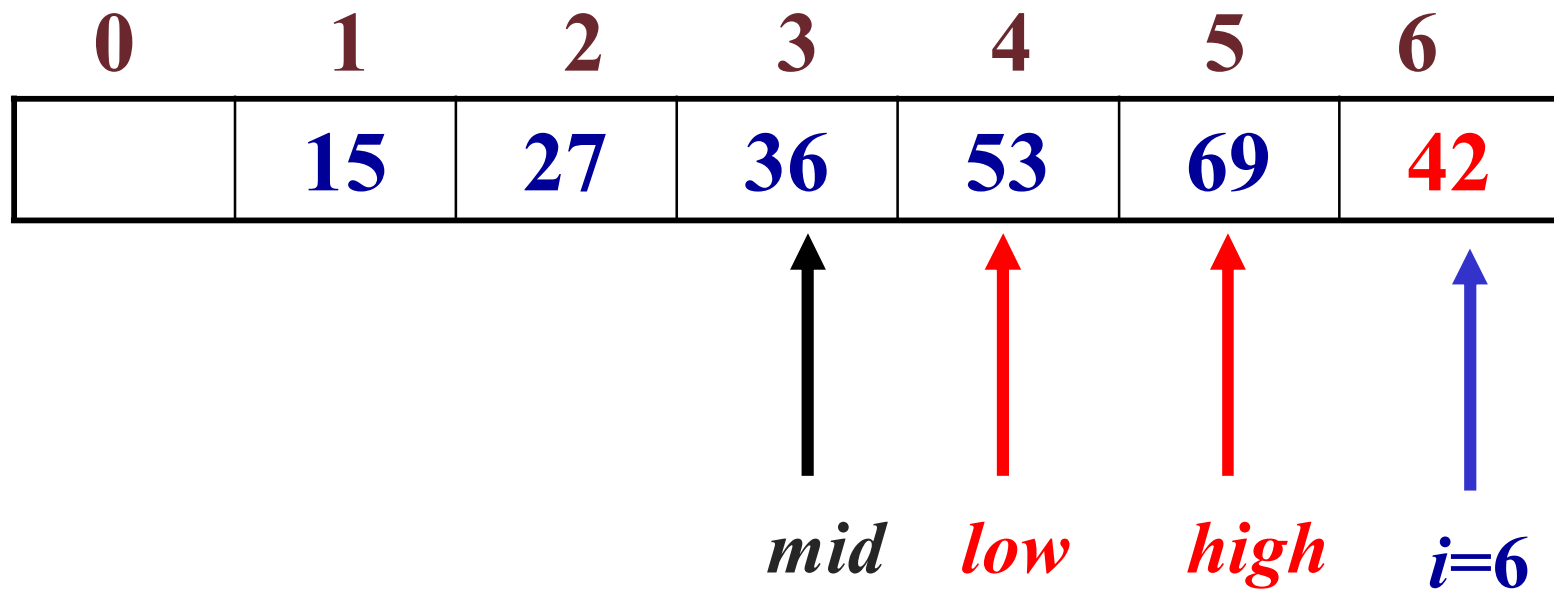
二分插入排序

- 方法：在插入第 i 个数据时， R_1, R_2, \dots, R_{i-1} 已经排序，用**二分查找**方法找出 R_i 应该插入的位置，将 R_i 插入。
- 并把键值大于 K_i 的数据后移一个位置，空出该位置将 R_i 插入。
- 二分插入排序也为稳定的排序法。

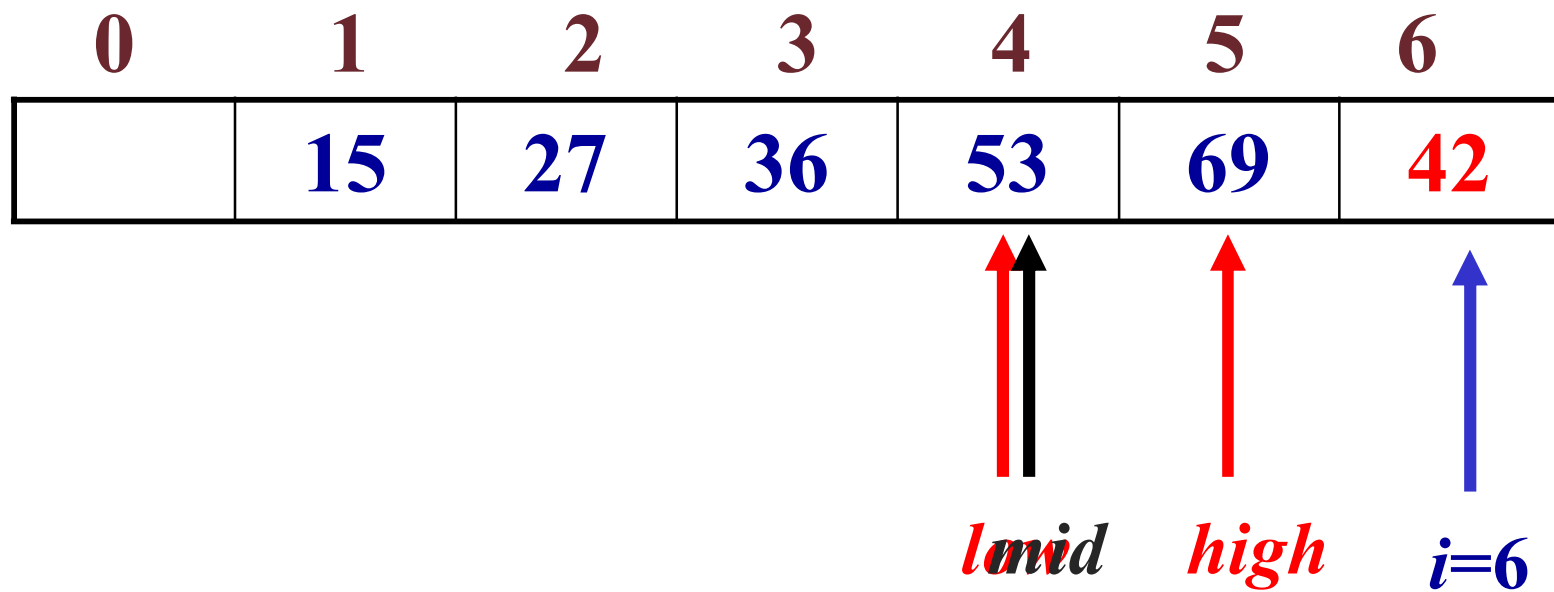
二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。



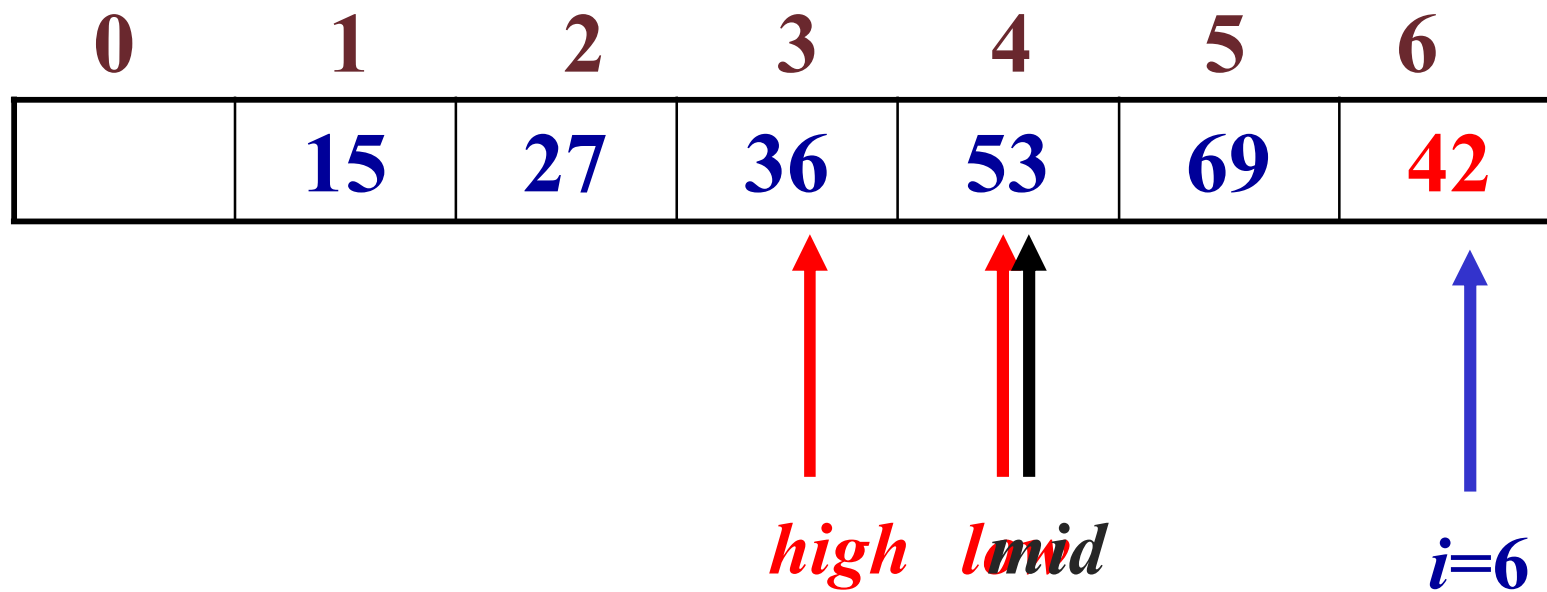
二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。



二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。

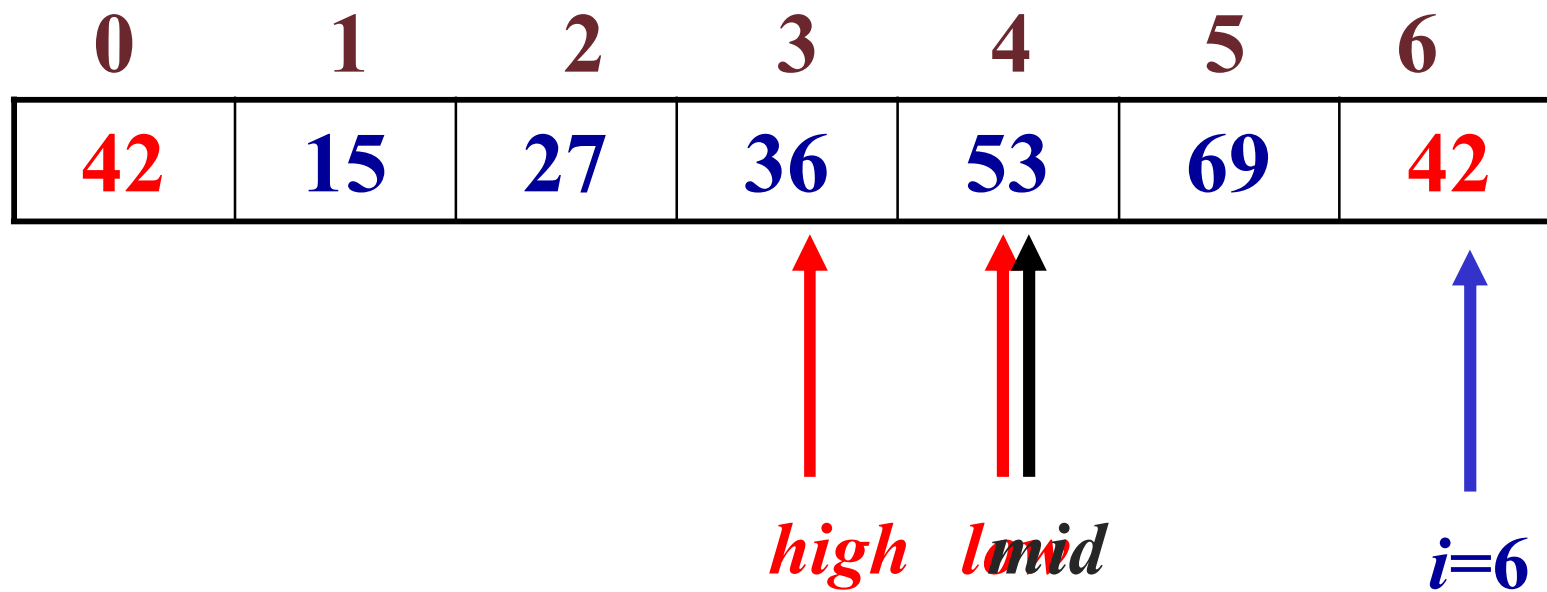


二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。



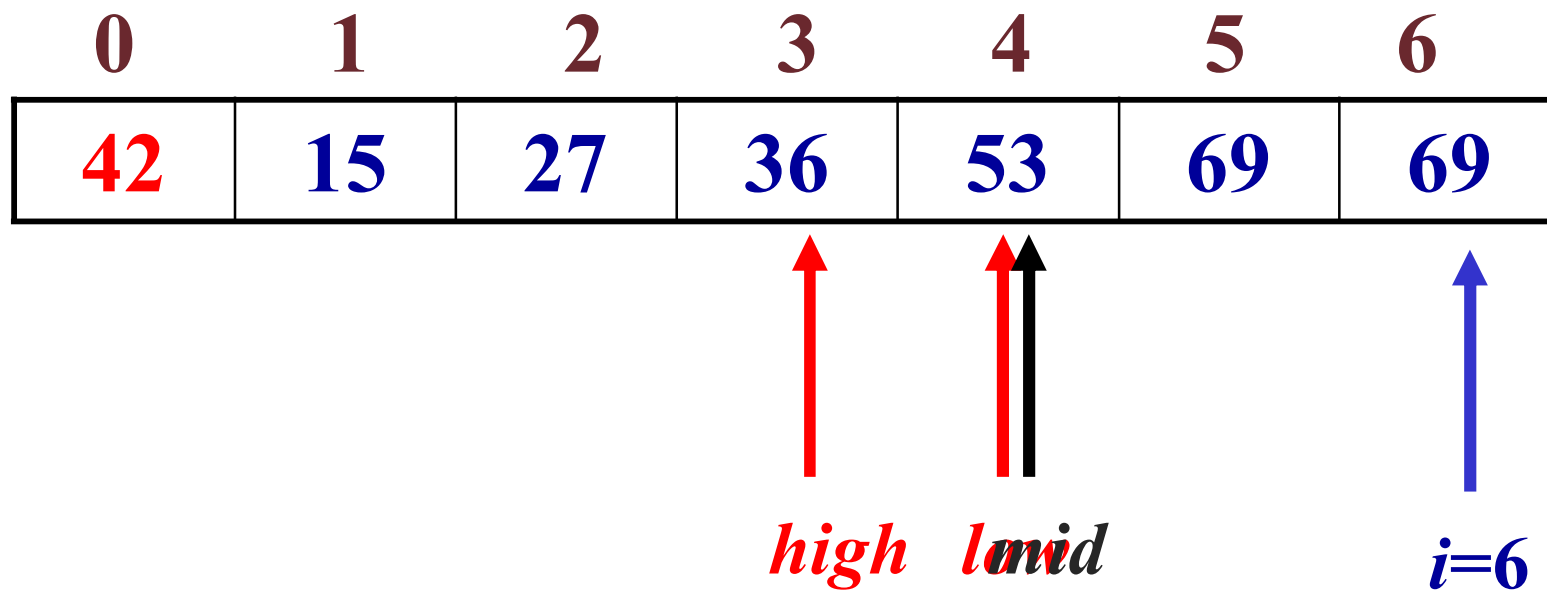
($high < low$, 查找结束, 插入位置为 low 或 $high+1$)

二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。



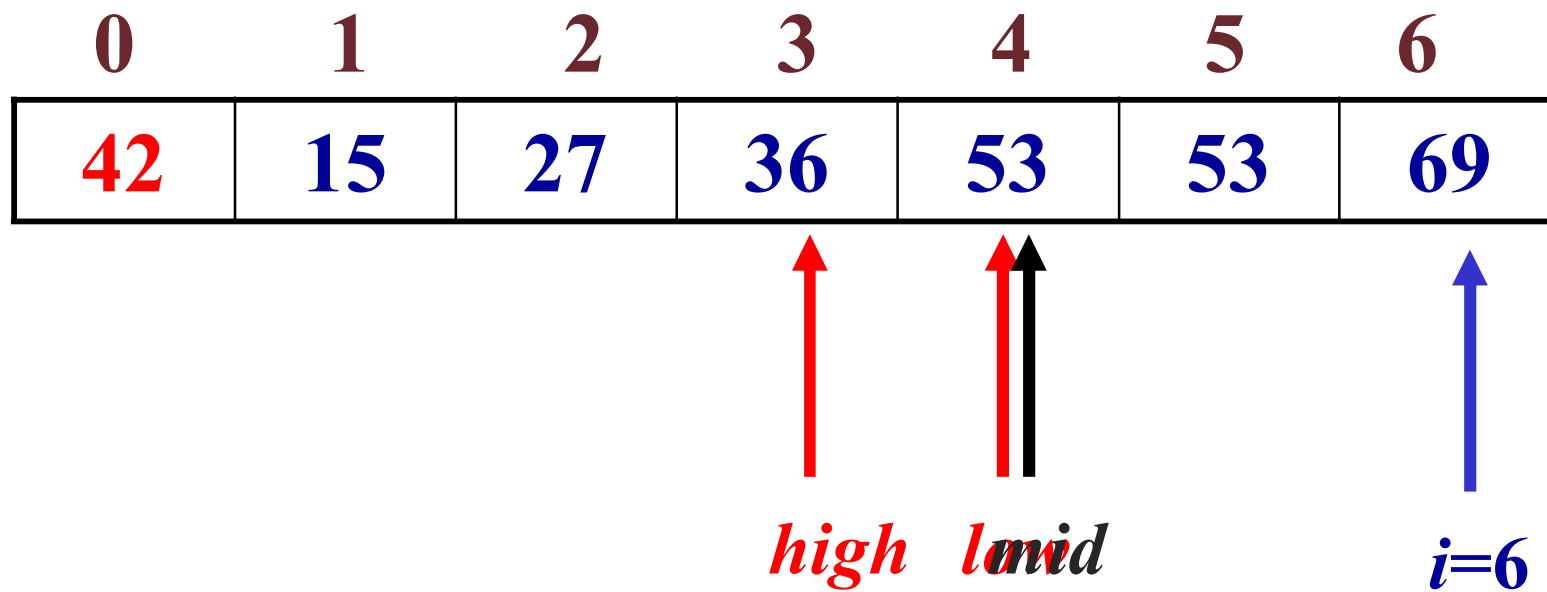
($high < low$, 查找结束, 插入位置为 low 或 $high+1$)

二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。



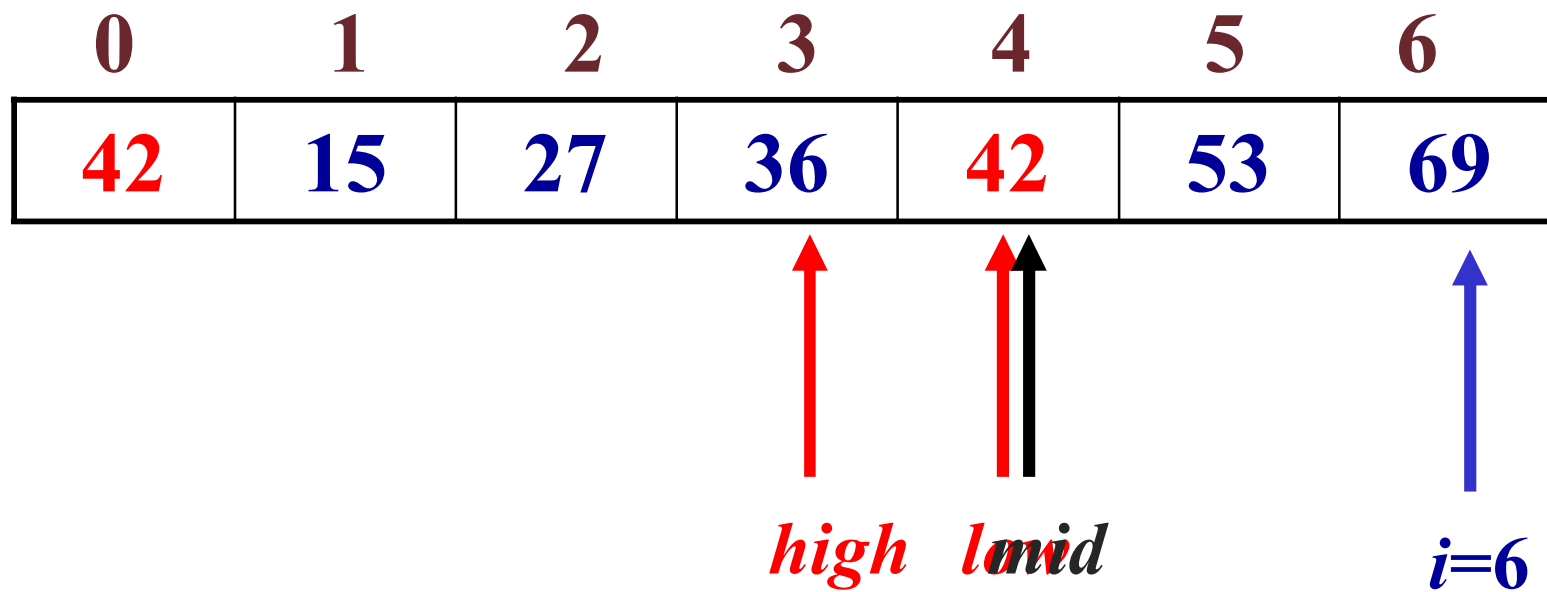
($high < low$, 查找结束, 插入位置为 low 或 $high+1$)

二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。



($high < low$, 查找结束, 插入位置为 low 或 $high+1$)

二分插入排序过程示例：假设有6个数据元素，前5个已排序的基础上，对第6个数据排序。



($high < low$, 查找结束, 插入位置为 low 或 $high+1$)



```
void BinsertSort(SqList &L)
```

```
{  int i,low,high,mid;
```

```
    for(i=2; i<=L.length; i ++)
```

```
        if(L.r[i].key<L.r[i-1].key)
```

```
        {
```

```
            L.r[0]=L.r[i];  low=1; high=i-1;
```

```
            while(low<=high)
```

```
            {
```

```
                mid=(low+high)/2;
```

```
                if (L.r[0].key<L.r[mid].key) high=mid-1;
```

```
                else low=mid+1;  }
```

```
            for( j=i-1; j>=high+1; j -- )
```

```
                L.r[j+1]=L.r[j];
```

```
                L.r[high+1]=L.r[0];
```

```
            }
```

```
    }
```



二分插入排序

- 二分插入排序减少了关键字的比较次数，但数据元素的移动次数不变，其时间复杂度与直接插入排序相同。
- 时间复杂度： $O(n^2)$
- 待排序的数据元素必须存放于数组



直接插入排序

- 特点1:

平均时间复杂度为: $O(n^2)$,

最好的情况当待排序的数据**基本有序**时: $O(n)$

- 特点2:

可证平均时间复杂度约为: $n^2/4$

当 $n < 16$ 时, $n^2/4 < n \log n$

理论值当 $n < 16$ 时, 直接插入比 $O(n \log n)$ 的排序方法快!

利用上述2个特点, **希尔排序**采用将**大问题**分割为**小问题**,
每一**小问题**采用直接插入排序, 所有**小问题**的完成使得整个待排序的数据基本有序时, 再一起插入排序-----**提高效率**



希尔排序

39, 80, 76, 41, 13, 29, 50, 78, 30, 11, 100, 7

$d_1=4$, 分成4组, 在每组内进行直接插入

- 又称缩小增量法。设待排序的数据有 n 个。
- 首先设定正整数增量 $d_1 < n$, 将待排序的数据分成 d_1 组
- 所有距离为 d_1 倍数的数据元素放在一组
- 第1、 d_1+1 、 $2d_1+1$, ...个数据元素为一组
- 第2, d_1+2 , $2d_1+2$, ...个数据元素为一组, 等等
- 各组内进行插入排序
- 然后取增量 $d_2 < d_1$, 重复上述分组和排序工作, 直至取 $d_i=1$, 即所有数据元素放在一个组内进行排序

增量序列可以有各种取法, 但应注意: 应使增量序列中的值没有除1的之外的公因子, 并且最后一个增量值必须等于1。



希尔排序-第一趟

■ $d_1=4$

39, 80, 76, 41, 13, 29, 50, 78, 30, 11, 100, 7



希尔排序-第一趟

■ $d_1=4$

39, 80, 76, 41, 13, 29, 50, 78, 30, 11, 100, 7
~~39, 80, 76, 41, 13, 29, 50, 78, 30, 11, 100, 7~~



希尔排序-第一趟

■ $d_1=4$

39, 80, 76, 41, 13, 29, 50, 78, 30, 11, 100, 7
39, 80, 76, 41, 13, 29, 50, 78, 30, 11, 100, 7
13, 80, 76, 41, 30, 29, 50, 78, 39, 11, 100, 7
13, 11, 76, 41, 30, 29, 50, 78, 39, 80, 100, 7
13, 11, 50, 41, 30, 29, 76, 78, 39, 80, 100, 7
13, 11, 50, 7, 30, 29, 76, 41, 39, 80, 100, 78

希尔排序-第一趟 结果:

13, 11, 50, 7, 30, 29, 76, 41, 39, 80, 100, 78



希尔排序-第二趟

■ $d_2=2$ (

■ 说明：红色数据一组，黑色数据一组，
每组内做直接插入排序

13, 11, 50, 7, 30, 29, 76, 41, 39, 80, 100, 78

13, 11, 30, 7, 39, 29, 50, 41, 76, 80, 100, 78

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

希尔排序-第二趟 结果：

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80



希尔排序-第三趟

- $d_3=1$

- 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80



希尔排序-第三趟

- $d_3=1$

- 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80



希尔排序-第三趟

- $d_3=1$

- 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 30, 39, 29, 50, 41, 76, 78, 100, 80



希尔排序-第三趟

■ $d_3=1$

■ 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 30, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 50, 41, 76, 78, 100, 80



希尔排序-第三趟

■ $d_3=1$

■ 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 30, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 41, 50, 76, 78, 100, 80



希尔排序-第三趟

■ $d_3=1$

■ 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 30, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 41, 50, 76, 78, 100, 80



希尔排序-第三趟

■ $d_3=1$

■ 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 30, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 41, 50, 76, 78, 100, 80



希尔排序-第三趟

- $d_3=1$

- 说明：所有数据元素均在一组，一起做直接插入

13, 7, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 13, 30, 11, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 30, 39, 29, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 50, 41, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 41, 50, 76, 78, 100, 80

7, 11, 13, 29, 30, 39, 41, 50, 76, 78, 80, 100

希尔排序稳定否? No!



希尔排序

```
void shell(SqList &L,int d)
{ for (i=d+1;i<=L.length;i++)
    if(L.r[i].key<L.r[i-d].key)
    {L.r[0]=L.r[i];
      for(j=i-d;j>0&&(L.r[0].key<L.r[j].key) ; j=j-d )
        L.r[j+d]=L.r[j];
      L.r[j+d]=L.r[0];
    }
}
```



希尔排序

```
void shell(SqList &L,int d)
{ for (i=d+1;i<=L.length;i++)
    if(L.r[i].key<L.r[i-d].key)
    {L.r[0]=L.r[i];L.r[i]=L.r[i-d];
      for(j=i-2d;j>0&&(L.r[0].key<L.r[j].key) ; j=j-d )
        L.r[j+d]=L.r[j];
      L.r[j+d]=L.r[0];
    }
}
```