

目 录

1 基本概念

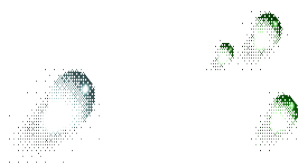
2 插入排序

3 交换排序

4 选择排序

5 归并排序

退出



1 基本概念

排序:是计算机程序设计中的一重要操作, 其功能是指一个数据元素集合或序列重新排列成一个按数据元素某个数据项值有序的序列。

排序码(关键码):排序依据的数据项。

稳定排序:排序前与排序后相同关键码元素间的位置关系, 保持一致的排序方法。

不稳定排序:排序前与排序后相同关键码元素间的相对位置发生改变的排序方法。

排序分为两类:

(1) **内排序:**指待排序列完全存放在内存中所进行的排序。内排序大致可分为五类: 插入排序、交换排序、选择排序、归并排序和分配排序。本章主要讨论内排序。

(2) **外排序:**指排序过程中还需访问外存储器的排序。

为了以后讨论方便, 我们直接将排序码写成一个一维数组的形式, 并且在没有声明的情形下, 所有排序都按排序码的值递增排列。

2 插入排序

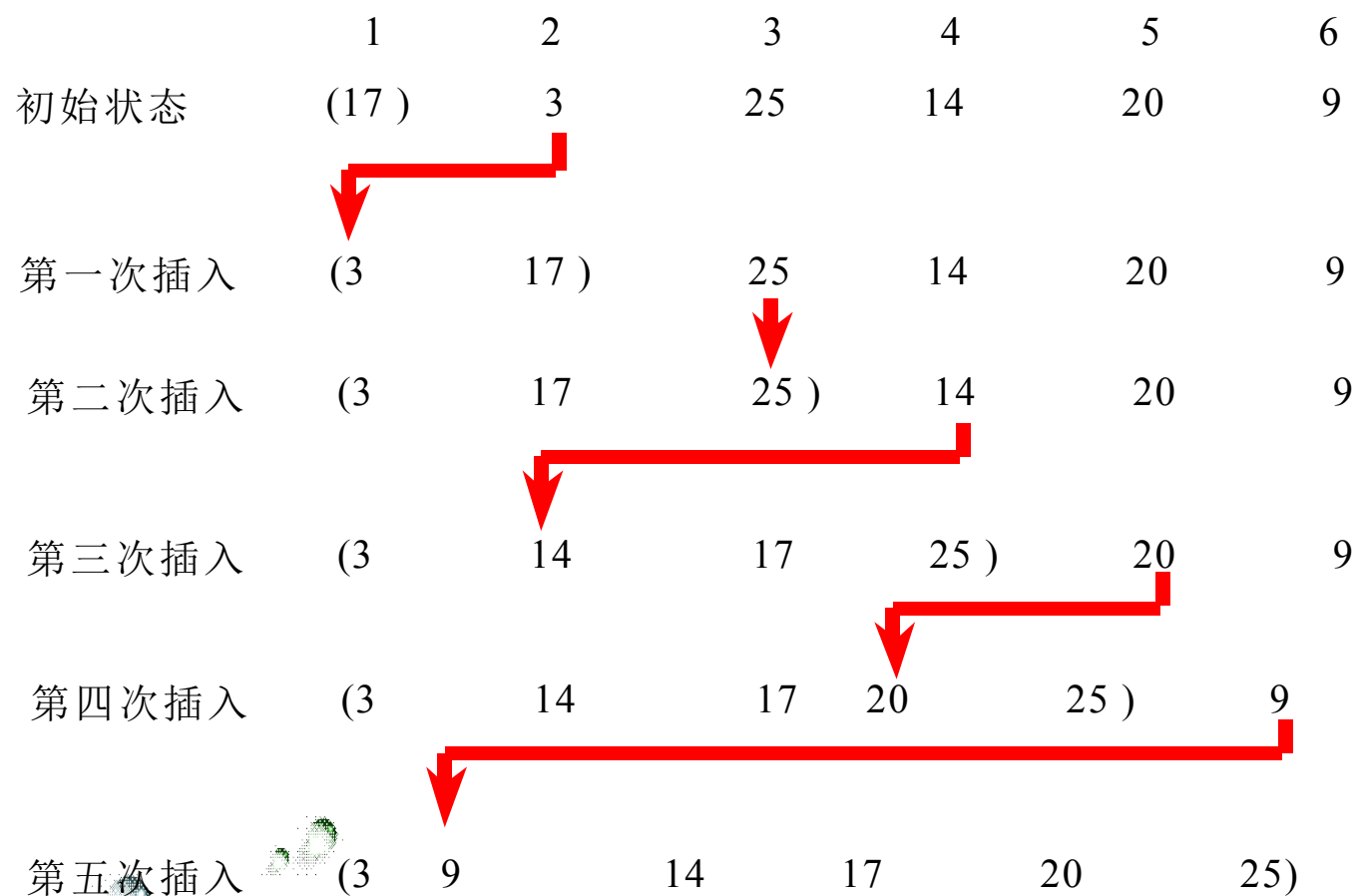
基本思想：每次将一个待排序的元素，按其关键字的大小插入到前面已经排好序的子文件的适当位置，直到全部记录插入完成为止。

2.1 直接插入排序

直接插入排序的基本思想是：把 n 个待排序的元素看成为一个有序表和一个无序表，开始时有序表中只包含一个元素，无序表中包含有 $n-1$ 个元素，排序过程中每次从无序表中取出第一个元素，把它的排序码依次与有序表元素的排序码进行比较，将它插入到有序表中的适当位置，使之成为新的有序表。

直接插入排序举例

例如， $n=6$ ，数组**R**的六个排序码分别为：**17，3，25，14，20，9**。它的直接插入排序的执行过程如下：



直接插入排序算法

```
void D-InsertSort(datatype R[ ],int n)
```

```
/*待排序的n个元素放在数组R中，用直接插入法进行排序*/
```

```
{ for ( i=2; i<=n; i++) /*i控制第i-1次插入,最多进行n-1次插入*/
```

```
    if (R[i].key<R[i-1].key) /*小于时,需将R[i]插入有序表*/
```

```
        { R[0]=R[i]; /*为统一算法设置监视哨*/
```

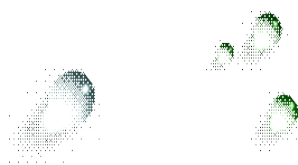
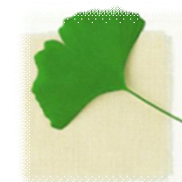
```
            for ( j=i-1; R[0].key<R[j].key;j--)
```

```
                R[j+1]=R[j]; /*元素后移*/
```

```
                R[j+1]= R[0]; /*将放在R[0]中的第i个元素插入到有序表中*/
```

```
        }
```

```
}
```






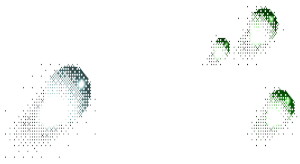
直接插入排序的效率分析



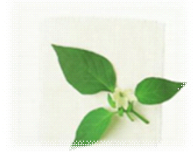
首先从空间来看，它只需要一个元素的辅助空间，用于元素的位置交换。从时间分析，首先外层循环要进行 **$n-1$** 次插入，每次插入最少比较一次(正序)，移动**0**次；最多比较 **i** 次(包括同监视哨 **$R[0]$** 的比较)，移动 **$i+1$** 次(逆序)(**$i=2,3,\dots,n$**)。因此，直接插入排序的时间复杂度为 **$O(n^2)$** 。



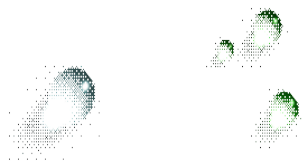
直接插入算法的元素移动是顺序的，该方法是稳定的。



2.2 希尔排序(缩小增量排序)



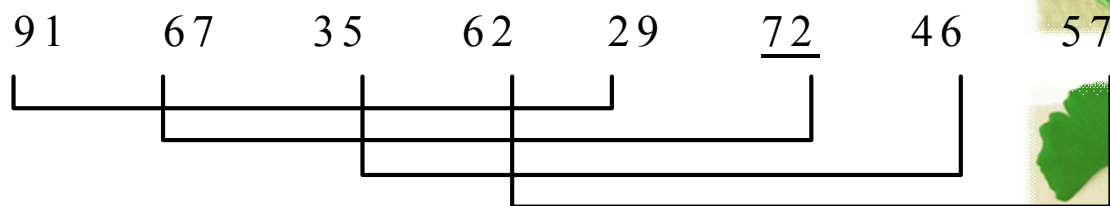
希尔排序的基本思想：先将整个待排元素序列分割成若干个子序列（由相隔某个“增量”的元素组成的）分别进行直接插入排序，待整个序列中的元素基本有序（增量足够小）时，再对全体元素进行一次直接插入排序。因为直接插入排序在元素基本有序的情况下（接近最好情况），效率是很高的，因此希尔排序在时间效率上有较大提高。



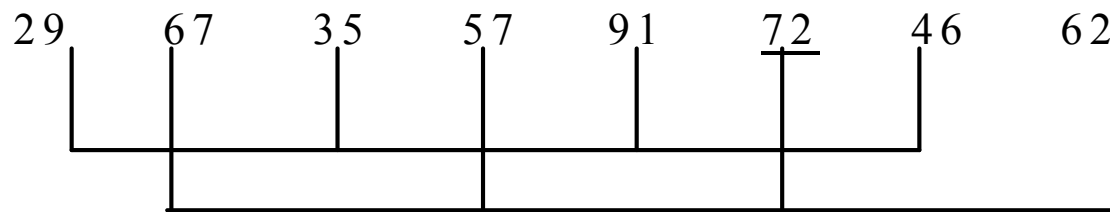
希尔排序过程举例

八个元素的关键码分别为：**91, 67, 35, 62, 29, 72, 46, 57**，希尔排序算法的执行过程为：

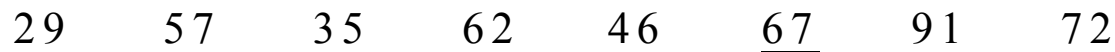
初始状态， $d_1=4$



第一趟结果， $d_2=2$



第二趟结果， $d_3=1$



第三趟结果



希尔排序算法

```
void ShellSort(datatype R[],int d[],int t)
```

/* 按增量序列 $d[0]$ 、 $d[1]$ 、 $d[2]$ 、...、 $d[t-1]$ 对顺序表 $R[1]$ 、 $R[2]$ 、...、 $R[n]$ 作希尔排序，注意 $d[0]$ 、 $d[1]$ 、 $d[2]$ 、...、 $d[t-1]$ 除1之外不能有公因子，且 $d[t-1]$ 必须为1 */

```
{ for(k=0;k<t;k++)
```

```
    ShellInsert(R,d[k]);
```

```
}
```

```
void ShellInsert(datatype R[],int dk)
```

/* 对顺序表 $R[1]$ 、 $R[2]$ 、...、 $R[n]$ 进行一趟插入排序， dk 为增量、步长 */

```
{ for(i=dk+1;i<=n;i++)
```

```
    if( R[i].key<R[i-dk].key)
```

```
        { R[0]=R[i]; /* 存放待插入的记录 */
```

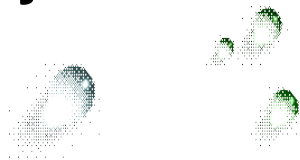
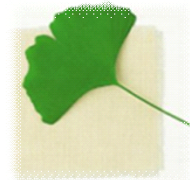
```
          for(j=i-dk;(j>0)&&(R[0].key<R[j].key);j=j-dk)
```

```
            R[j+dk]=R[j]; /* 记录后移 */
```

```
            R[j+dk]=R[0]; /* 插入到正确位置 */
```

```
        }
```

```
}
```





希尔排序的效率分析



虽然我们给出的算法是三层循环，最外层循环为 $\log_2 n$ 数量级，中间的for循环是 n 数量级的，内循环远远低于 n 数量级，因为当分组较多时，组内元素较少；此循环次数少；当分组较少时，组内元素增多，但已接近有序，循环次数并不增加。因此，希尔排序的时间复杂性在 $O(n \log_2 n)$ 和 $O(n^2)$ 之间，大致为 $O(n^{1.3})$ 。

由于希尔排序对每个子序列单独比较，在比较时进行元素移动，有可能改变相同排序码元素的原始顺序，因此希尔排序是不稳定的。

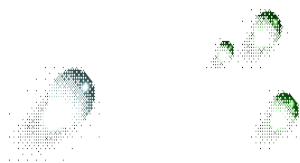
3 交换排序

主要是通过排序表中两个记录关键码的比较，若与排序要求相逆（不符合升序或降序），则将两者交换。

3.1 冒泡排序

冒泡排序的基本思想：通过对待排序序列从前向后，依次比较相邻元素的排序码，若发现逆序则交换，使排序码较大的元素逐渐从前部移向后部。

因为排序的过程中，各元素不断接近自己的位置，如果一趟比较下来没有进行过交换，就说明序列有序，因此要在排序过程中设置一个标志**swap**判断元素是否进行过交换。从而减少不必要的比较。



冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	49	38	65	97	76	13	27	49

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	97	76	13	27	49

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	97	76	13	27	49

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	97	76	13	27	49

冒泡排序的具体过程


若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	97	76	13	27	49



冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	76	97	13	27	49

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	76	97	13	27	49

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	76	13	97	27	49

冒泡排序的具体过程


若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	76	13	97	27	49



冒泡排序的具体过程


若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	76	13	27	97	49



冒泡排序的具体过程


若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	76	13	27	97	49



冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49
	38	49	65	76	13	27	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	76	13	27	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	76	13	27	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	76	13	27	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	76	13	27	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	13	76	27	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	13	76	27	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	13	27	76	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49	97
	38	49	65	13	27	76	49	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	76	13	27	49'	97
	38	49	65	13	27	49'	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	65	13	27	49	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	65	13	27	49	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	65	13	27	49	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	13	65	27	49	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	13	65	27	49	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	13	27	65	49	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	13	27	65	49	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	65	13	27	49	76	97
	38	49	13	27	49	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	13	27	49	65	76	97
	38	49	13	27	49	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	13	27	49	65	76	97
	38	49	13	27	49	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	13	27	49	65	76	97
	38	13	49	27	49	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	13	27	49	65	76	97
	38	13	49	27	49	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	13	27	49	65	76	97
	38	13	27	49	49	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	49	13	27	49'	65	76	97
	38	13	27	49	49'	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	13	27	49	49'	65	76	97
	38	13	27	49	49'	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	13	27	49	49'	65	76	97
	13	38	27	49	49'	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	13	27	49	49'	65	76	97
	13	38	27	49	49'	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确,则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	13	27	49	49'	65	76	97
	13	27	38	49	49'	65	76	97

冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	38	13	27	49	49'	65	76	97
	13	27	38	49	49'	65	76	97

冒泡排序的具体过程


若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	13	27	38	49	49'	65	76	97
	13	27	38	49	49'	65	76	97



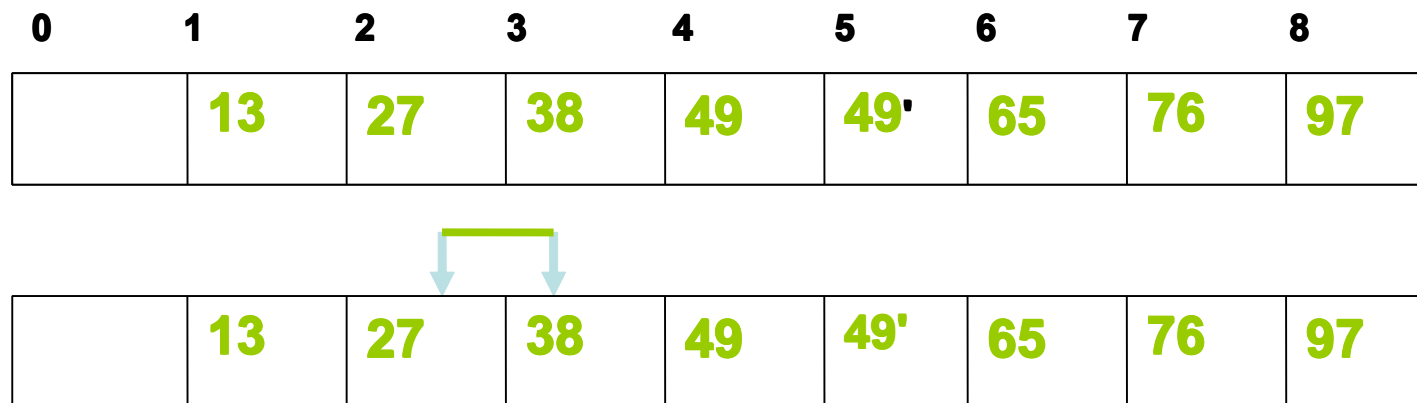
冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序



冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序

0	1	2	3	4	5	6	7	8
	13	27		49	49'	65	76	97
	13	27	38	49	49'	65	76	97

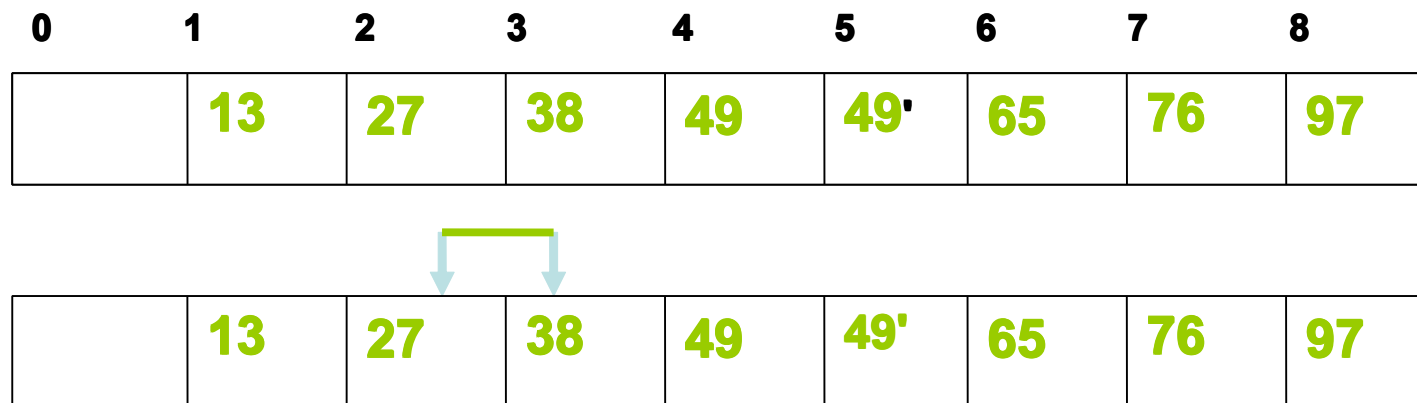
冒泡排序的具体过程

若序列中有 n 个元素，通常进行 $n - 1$ 趟。第 1 趟，针对第 $R[1]$ 至 $R[n]$ 个元素进行。第 2 趟，针对第 $R[1]$ 至 $R[n-1]$ 个元素进行。..... 第 $n-1$ 趟，针对第 $R[1]$ 至 $R[2]$ 个元素进行。

每一趟进行的过程：从第一个元素开始，比较两个相邻的元素。若相邻的元素的相对位置不正确，则进行交换；否则继续比较下面两个相邻的元素。

结束条件：在任何一趟进行过程中，未出现交换。

如：将序列 **49、38、65、97、76、13、27、49** 用起泡排序的方法进行排序



冒泡排序算法的实现

```
void Bubble-sort( datatype R[],int n)
```

```
{ int i, j, swap; /* 当swap为0则停止排序 */
```

```
  for ( i=1; i<n; i++) /* i 表示趟数，最多n-1趟 */
```

```
  { swap=0;          /* 开始时元素未交换 */
```

```
    for ( j=1; j<=n-i; j++)
```

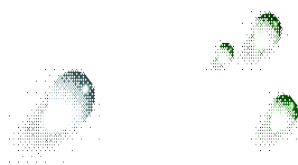
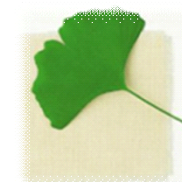
```
      if (R[j].key>R[j+1].key) /* 发生逆序 */
```

```
        { R[0]=R[j];R[j]=R[j+1];R[j+1]=R[0];
```

```
          swap=1; }          /* 交换，并标记发生了交换 */
```

```
      if(swap==0) break;  }
```

```
}
```



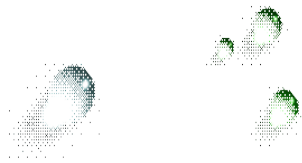


冒泡排序的效率分析



从冒泡排序的算法可以看出，若待排序的元素为正序，则只需进行一趟排序，比较次数为 $(n-1)$ 次，移动元素次数为 0 ；若待排序的元素为逆序，则需进行 $n-1$ 趟排序，比较次数为 $(n^2-n)/2$ ，移动次数为 $3(n^2-n)/2$ ，因此冒泡排序算法的时间复杂度为 $O(n^2)$ 。由于其中的元素移动较多，所以属于内排序中速度较慢的一种。

因为冒泡排序算法只进行元素间的顺序移动，所以是一个稳定的算法。



3.2 快速排序(分区交换排序)

快速排序（Quick Sorting）是迄今为止所有内排序算法中速度最快的一种。它的基本思想是：任取待排序序列中的某个元素作为标准（**也称为支点、界点**，一般取第一个元素），通过一次划分，将待排元素分为左右两个子序列，左子序列元素的排序码均小于基准元素的排序码，右子序列的排序码则大于或等于基准元素的排序码，然后分别对两个子序列继续进行划分，直至每一个序列只有一个元素为止。最后得到的序列便是有序序列。

假设：[49 38 65 97 76 13 27 49]

第1趟 [27 38 13] 49 [76 97 65 49]

第2趟 [[13] 27 [38]] 49 [[49' 65] 76 [97]]

第3趟 [[13] 27 [38]] 49 [[49'[65]] 76 [97]]

最后结果 13 27 38 49 49' 65 76 97

一次划分的具体过程

1. low 指向待划分区域首元素, $high$ 指向待划分区域尾元素;
2. $R[0]=R[low]$ (为了减少数据的移动,将作为标准的元素暂存到 $R[0]$ 中, 最后再放入最终位置);
3. $high$ 从后往前移动直到 $R[high].key < R[0].key$;
4. $R[low]=R[high]$, $low++$;
5. low 从前往后移动直到 $R[low].key \geq R[0].key$;
6. $R[high]=R[low]$, $high--$;
7. goto 3;
8. 直到 $low==high$ 时, $R[low]=R[0]$ (即将作为标准的元素放到其最终位置)。

概括地说, 一次划分就是从表的两端交替地向中间进行扫描, 将小的放到左边, 大的放到右边, 作为标准的元素放到中间。

快速排序的排序过程

例

初始关键字: 27 38 13 5 97 37 76 97 13 65 27 50

Diagram showing the initial array with pivot 'x' above the first element (27) and arrows indicating the partitioning process. The pivot is 27. Elements less than 27 (13, 5) are moved to the left, and elements greater than 27 (38, 97, 37, 76, 97, 65) are moved to the right. The final partitioned array is (13) 27 (38) 49 (50 65) 76 (97).

完成一趟排序: (27 38 13) 49 (76 97 65 50)

分别进行快速排序: (13) 27 (38) 49 (50 65) 76 (97)

快速排序结束: 13 27 38 49 50 65 76 97



0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49

	49	38	65	97	76	13	27	49'
--	-----------	-----------	-----------	-----------	-----------	-----------	-----------	------------

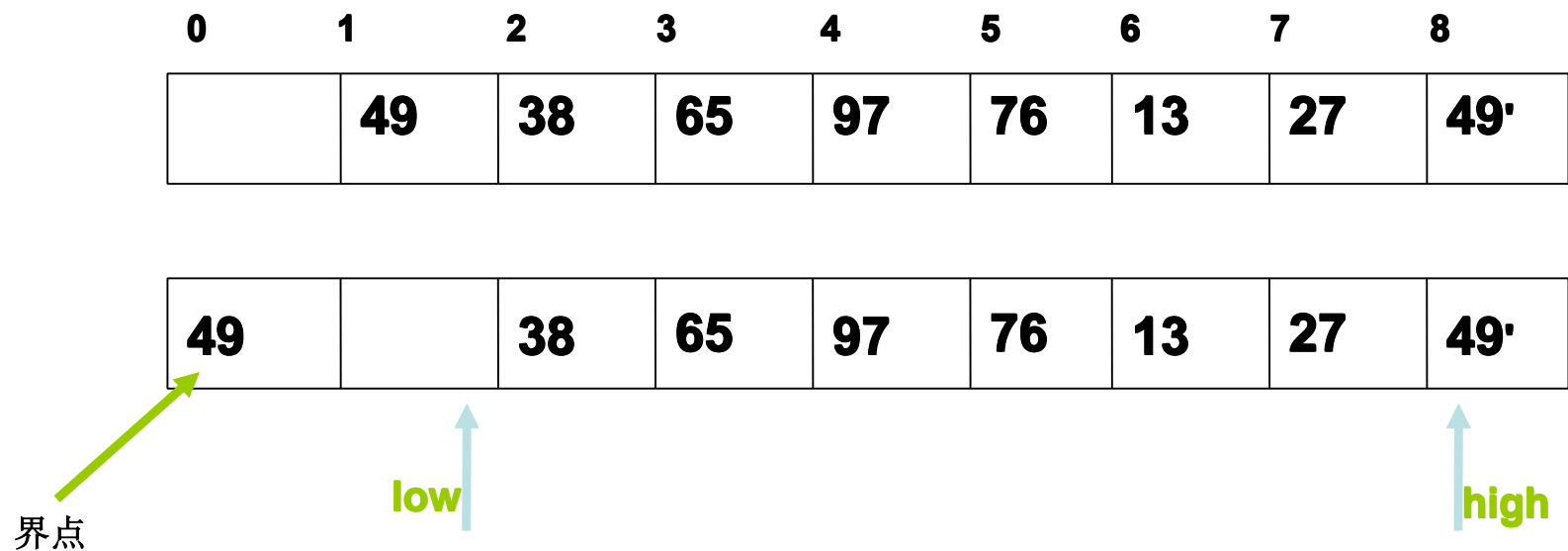
high

一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：

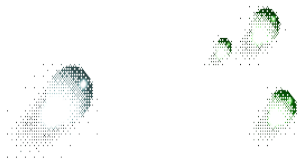
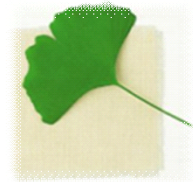
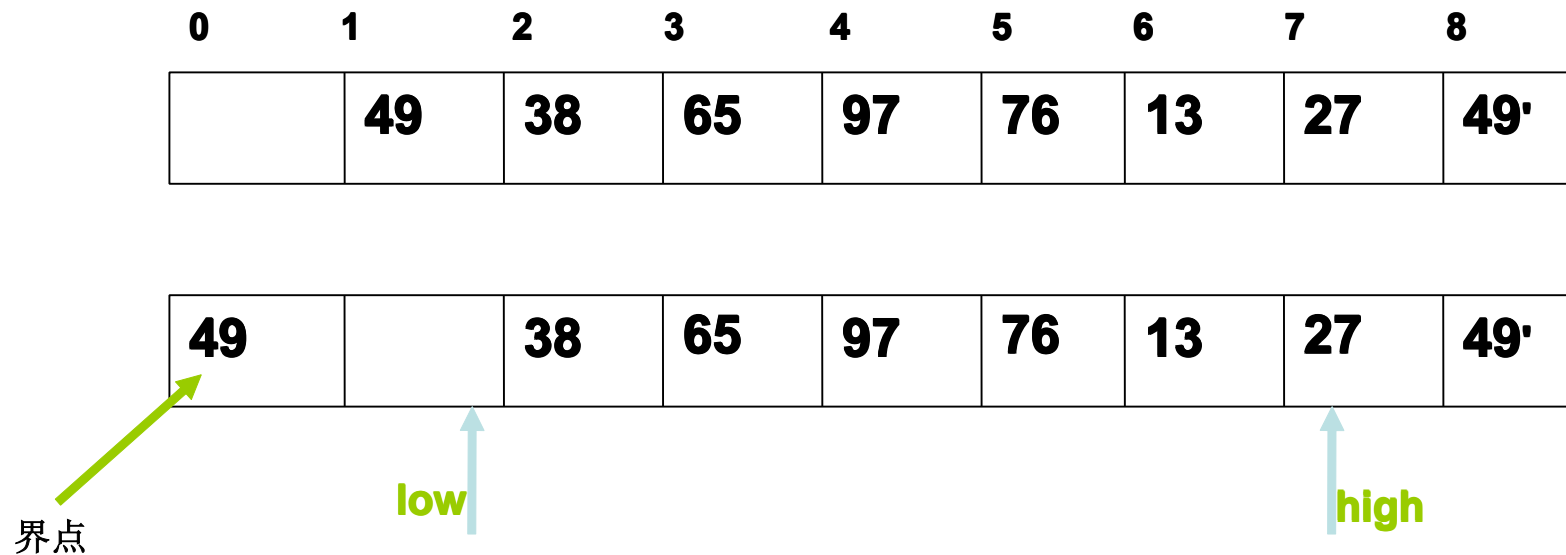


一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key < R[0].key**;

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：



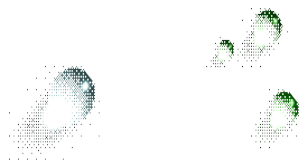
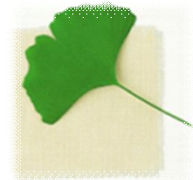
一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key < R[0].key**;

4. **R[low]=R[high], low++;**

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：



一次划分的具体过程示例

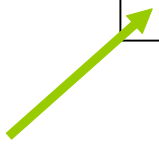


一次划分的具体过程为：

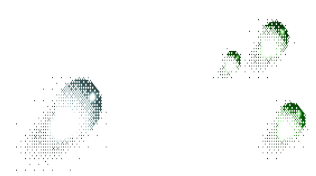
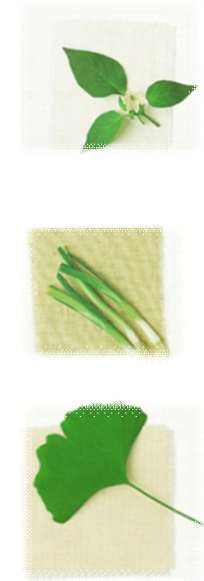
1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key < R[0].key**;
4. **R[low]=R[high], low++**;
5. **low**从前往后移动直到**R[low].key >= R[0].key**;

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49'

49	27	38	65	97	76	13		49'
-----------	-----------	-----------	-----------	-----------	-----------	-----------	--	------------

界点   



一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key < R[0].key**；
4. **R[low]=R[high], low++**；
5. **low**从前往后移动直到**R[low].key >= R[0].key**；
6. **R[high]=R[low], high--**；

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：

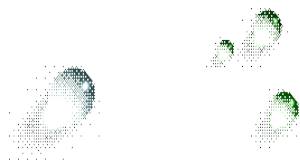
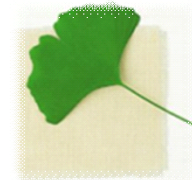
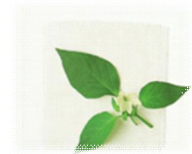
0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49'

49	27	38		97	76	13	65	49'
----	----	----	--	----	----	----	----	-----

界点

low

high



一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key<R[0].key**；
4. **R[low]=R[high], low++**；
5. **low**从前往后移动直到**R[low].key>=R[0].key**；
6. **R[high]=R[low], high--**；
7. **goto 3**；

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：

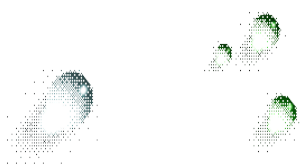
0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49'

49	27	38		97	76	13	65	49'
-----------	-----------	-----------	--	-----------	-----------	-----------	-----------	------------

界点

low

high



一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key<R[0].key**;
4. **R[low]=R[high], low++**;
5. **low**从前往后移动直到**R[low].key>=R[0].key**;
6. **R[high]=R[low], high--**;
7. **goto 3**;

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：

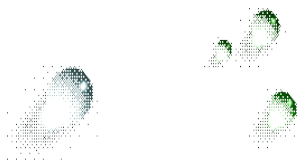
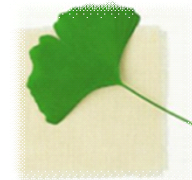
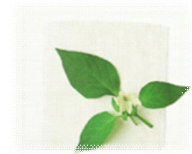
0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49'

49	27	38	13	97	76		65	49'
-----------	-----------	-----------	-----------	-----------	-----------	--	-----------	------------

界点

low

high



一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动, 将作为标准的元素暂存到**R[0]**中, 最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key<R[0].key**;
4. **R[low]=R[high], low++**;
5. **low**从前往后移动直到**R[low].key>=R[0].key**;
6. **R[high]=R[low], high--**;
7. **goto 3**;

如：将序列 **49、38、65、97、76、13、27、49'** 一次划分的过程为：

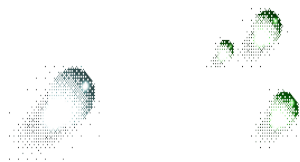
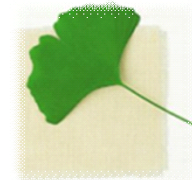
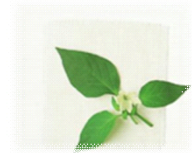
0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49'

49	27	38	13		76	97	65	49'
-----------	-----------	-----------	-----------	--	-----------	-----------	-----------	------------

界点

low

high



一次划分的具体过程示例

一次划分的具体过程为：

1. **low**指向待划分区域首元素，**high**指向待划分区域尾元素；
2. **R[0]=R[low]** (为了减少数据的移动，将作为标准的元素暂存到**R[0]**中，最后再放入最终位置)；
3. **high**从后往前移动直到**R[high].key<R[0].key**；
4. **R[low]=R[high], low++**；
5. **low**从前往后移动直到**R[low].key>=R[0].key**；
6. **R[high]=R[low], high--**；
7. **goto 3**；
8. 直到**low==high**时，**R[low]=R[0]** (即将作为标准的元素放到其最终位置)。

如：将序列 **49、38、65、97、76、13、27、49'** 进行一次划分的过程为：

0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49'

	27	38	13	49	76	97	65	49'
--	-----------	-----------	-----------	-----------	-----------	-----------	-----------	------------

low **high**

概括地说，一次划分就是从表的两端交替地向中间进行扫描，将小的放到左边，大的放到右边，作为标准的元素放到中间。

快速排序算法

```
void Quick_Sort(datatype R[],int s,int t)  /* 对R[s]到R[t]的元素进行排序 */
```

```
/*
```

```
{ if (s<t)
```

将表一分为二

```
    { i=Partition(R,s,t);
```

对左子序列进行快速排序

```
        Quick_Sort(R,s,i-1);
```

```
        Quick_Sort(R,i+1,t);    }
```

对右子序列进行快速排序

```
}
```

```
int Partition(datatype R[],int low,int high)
```

```
{ R[0]=R[low];  /* 暂存界点元素到R[0]中*/
```

在右端扫描

```
while(low<high) /* 从表的两端交替地向中间扫描 */
```

```
    { while(low<high&&R[high].key>=R[0].key)  high--;
```

把比界点小的元素放到界点前面

```
        if(low<high) { R[low]=R[high];  low++;  }
```

```
        while(low<high&&R[low].key<R[0].key)  low++;
```

在左端扫描

```
        if (low<high) { R[high]=R[low]; high--; }
```

```
    }
```

```
    R[low]=R[0]; /* 将界点元素放到其最终位置 */
```

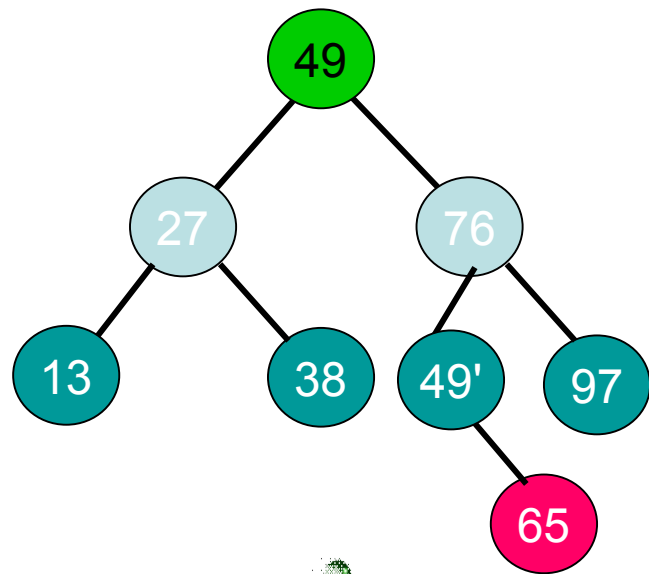
把比界点大的元素放到界点后面

```
    return low; /* 返回界点元素所在的位置*/
```

```
}
```

快速排序的递归树

快速排序的递归过程可用一棵二叉树形象地给出。下图为待排序列49、38、65、97、76、13、27、49' 所对应的快速排序递归调用过程的二叉树(简称为快速排序递归树)。



从快速排序算法的递归树可知，快速排序的趟数取决于递归树的高度。

快速排序的时间复杂度

如果每次划分对一个对象定位后，该对象的左子序列与右子序列的长度相同，则下一步将是对两个长度减半的子序列进行排序，这是最理想的情况。

假设 n 是2的幂， $n=2^k$ ($k=\log_2 n$)，假设支点位置位于序列中间，这样划分的子区间大小基本相等。

$$n+2(n/2)+4(n/4)+\dots+n(n/n)=n+n+\dots+n=n*k=n*\log_2 n$$

因此，快速排序的最好时间复杂度为 $O(n\log_2 n)$ 。而且在理论上已经证明，快速排序的平均时间复杂度也为 $O(n\log_2 n)$ 。实验结果表明：就平均计算时间而言，快速排序是所有内排序方法中最好的一个。

在最坏的情况，即待排序对象序列已经按其排序码从小到大排好序的情况下，其递归树成为单支树，每次划分只得到一个比上一次少一个对象的子序列(蜕化为冒泡排序)。必须经过 $n-1$ 趟才能把所有对象定位，而且第 i 趟需要经过 $n-i$ 次排序码比较才能找到第 i 个对象的安放位置，总的排序码比较次数将达到

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

因此，快速排序的最坏时间复杂度为 $O(n^2)$ 。

快速排序的空间复杂度及稳定性



快速排序是递归的，需要有一个栈存放每层递归调用时的指针和参数。最大递归调用层数与递归树的高度一致。理想情况为 $\lceil \log_2(n+1) \rceil$ ；最坏情况即待排序对象序列已经按其排序码从小到大排好序的情况下，其递归树成为单支树，深度为 n 。因此，快速排序最好的空间复杂度为 $O(\log_2 n)$ ，最坏的空间复杂度为 $O(n)$ （即快速排序所需用的辅助空间）。

快速排序是一种不稳定的排序方法。可用 $3, 2, 2'$ 序列来验证。



4 选择排序

基本原理： 将待排序的元素分为已排序（初始为空）和未排序两组，依次将未排序的元素中值最小的元素放入已排序的组中。

两种常见的选择排序

- ✓ 简单选择排序
- ✓ 堆排序

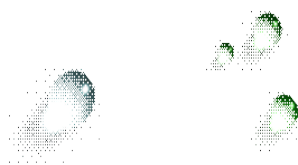
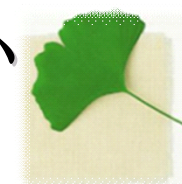
4.1 简单选择排序



简单选择排序的基本过程为：



- 在一组元素 $R[i]$ 到 $R[n]$ 中选择具有最小关键码的元素
- 若它不是这组元素中的第一个元素，则将它与这组元素中的第一个元素对调。
- 除去具有最小关键字的元素，在剩下的元素中重复第(1)、(2)步，直到剩余元素只有一个为止。



简单选择排序示例

简单选择排序
过程为：

(1) 在一组元素 $R[i]$ 到 $R[n]$ 中选择具有最小关键码的元素

(2) 若它不是这组元素中的第一个元素，则将它与这组元素中的第一个元素对调。

(3) 除去具有最小关键字的元素，在剩下的元素中重复第(1)、(2)步，直到剩余元素只有一个为止。

49	38	65	97	76	13	27	49'
----	----	----	----	----	----	----	-----

初始状态

13	38	65	97	76	49	27	49'
----	----	----	----	----	----	----	-----

第1趟

13	27	65	97	76	49	38	49'
----	----	----	----	----	----	----	-----

第2趟

13	27	38	97	76	49	65	49'
----	----	----	----	----	----	----	-----

第3趟

13	27	38	49	76	97	65	49'
----	----	----	----	----	----	----	-----

第4趟

13	27	38	49	49'	97	65	76
----	----	----	----	-----	----	----	----

第5趟

13	27	38	49	49'	65	97	76
----	----	----	----	-----	----	----	----

第6趟

13	27	38	49	49'	65	76	97
----	----	----	----	-----	----	----	----

第7趟

简单选择排序算法



```
Void Select-Sort(datatype R[], int n)
```

```
/*对R[1]到R[n]的元素进行排序*/
```

```
{ for (i=1; i<n; i++)
```

进行n-1趟排序



```
{ k=i;
```

```
  for (j=i+1; j<=n; j++) {
```

```
    if (R[j].key<R[k].key) k=j;
```

寻找最小
元素下标k

```
  }
```

```
  if (k!=i)
```

```
  { R[0]=R[k];
```

```
    R[k]=R[i];
```

```
    R[i]=R[0];
```

元素交换

```
  }
```

```
}
```

```
}
```


简单选择排序的效率分析

1. 无论初始状态如何，在第*i* 趟排序中选择最小关键码的元素，需做*n-i*次比较，因此总的比较次数为：


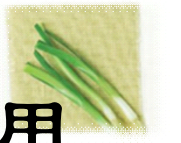
$$\sum_{i=1}^{n-1} (n-i) = n(n-1)/2 = O(n^2) \text{ (即时间复杂度)}$$

2. 最好情况：序列为正序时，移动次数为0，最坏情况：序列为反序时，每趟排序均要执行交换操作，总的移动次数取最大值 $3(n-1)$ 。

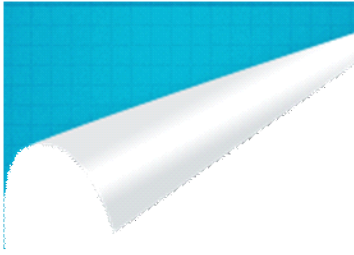
3. 由于在直接选择排序中存在着不相邻元素之间的互换，因此，直接选择排序是一种不稳定的排序方法。例如，给定排序码为3，7，3'，2，1，排序后的结果为1，2，3'，3，7。



选择排序——堆排序的引入



堆排序是简单选择排序的改进。用直接选择排序从 n 个记录中选出关键字值最小的记录要做 $n-1$ 次比较，然后从其余 $n-1$ 个记录中选出最小者要作 $n-2$ 次比较。显然，相邻两趟中某些比较是重复的。为了避免重复比较，可以采用**树形选择排序**比较。



(a) 求出最小关键字3 (b) 求出次小关键字11

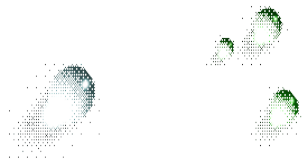
图 8.8 树形选择排序



选择排序——堆排序的引入



树形选择排序总的比较次数为 $O(n \log_2 n)$ ，与直接选择排序比较，减少了比较次数，但需要增加额外的存储空间存放中间比较结果和排序结果。



堆排序

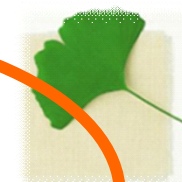
1. 堆的定义

n 个元素的序列 $\{k_1, k_2, \dots, k_n\}$, 当且仅当满足

$$\begin{array}{ll} k_i \leq k_{2i} & \\ (1) \quad k_i \leq k_{2i+1} & \text{或} \quad (2) \quad k_i \geq k_{2i} \\ \left\{ \right. & \left\{ \right. \\ & k_i \geq k_{2i+1} \end{array} \quad (i=1, 2, \dots, \lfloor n/2 \rfloor)$$

称之为堆。

若将此排序码按顺序组成一棵完全二叉树, 则 (1) 称为小顶堆 (二叉树的所有结点值小于或等于左右孩子的值), (2) 称为大顶堆 (二叉树的所有结点值大于或等于左右孩子的值)。



小顶堆和大顶堆示例

序列{ 12, 36, 24, 85, 47, 30, 53, 91} 满足

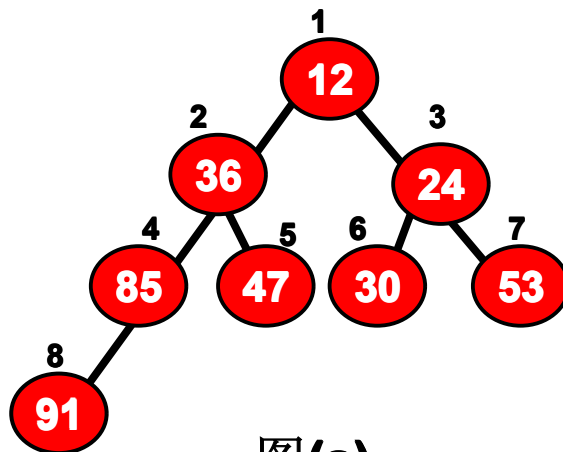
$$\begin{cases} k_i \leq k_{2i} \\ k_i \leq k_{2i+1} \quad (i = 1, 2, \dots, \lfloor n/2 \rfloor) \end{cases}$$

为小顶堆, 如图(a)。

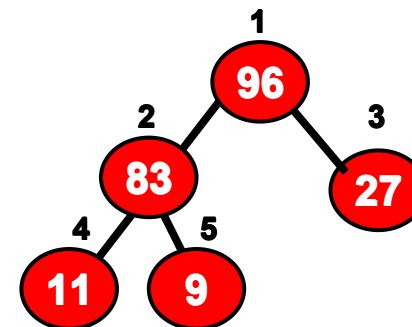
序列{ 96, 83, 27, 11, 9} 满足

$$\begin{cases} k_i \geq k_{2i} \\ k_i \geq k_{2i+1} \quad (i = 1, 2, \dots, \lfloor n/2 \rfloor) \end{cases}$$

为大顶堆, 如图(b)。



图(a)



图(b)

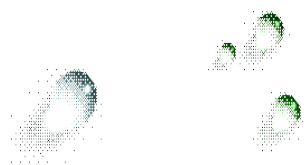
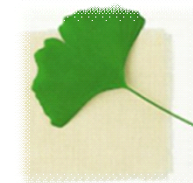
2. 堆排序的基本思想

(1) 建初始堆

将排序码 $k_1, k_2, k_3, \dots, k_n$ 表示成一棵完全二叉树，然后从第 $\lfloor n/2 \rfloor$ 个排序码(即树的最后一个非终端结点)开始**筛选**，使由该结点作根结点组成的子二叉树符合**堆的定义**，然后从第 $\lfloor n/2 \rfloor - 1$ 个排序码重复刚才操作，直到第一个排序码止。这时候，该二叉树符合堆的定义，初始堆已经建立。

(2) 堆排序

将堆中第一个结点（二叉树根结点）和最后一个结点的数据进行**交换**（ k_1 与 k_n ），再将 $k_1 \sim k_{n-1}$ **重新建堆**，然后 k_1 和 k_{n-1} 交换，再将 $k_1 \sim k_{n-2}$ 重新建堆，然后 k_1 和 k_{n-2} 交换，如此重复下去，每次重新建堆的元素个数不断减1，直到重新建堆的元素个数仅剩一个为止。这时堆排序已经完成，则排序码 $k_1, k_2, k_3, \dots, k_n$ 已排成一个有序序列。





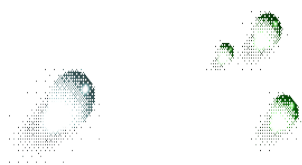
堆排序的两大步骤

堆排序过程分为两大步骤:

(1) 根据初始输入数据形成初始堆

;

(2) 通过一系列的元素交换和重新调整堆进行排序。

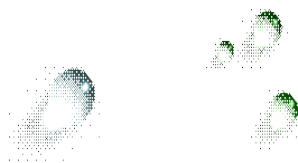
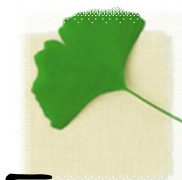




堆排序的关键问题

堆排序需解决的两个问题：


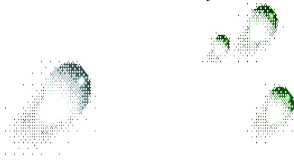
- ◆ 如何由一个无序序列建成一个堆？
- ◆ 如何在输出堆顶元素之后，调整剩余元素，使之成为一个新的堆？





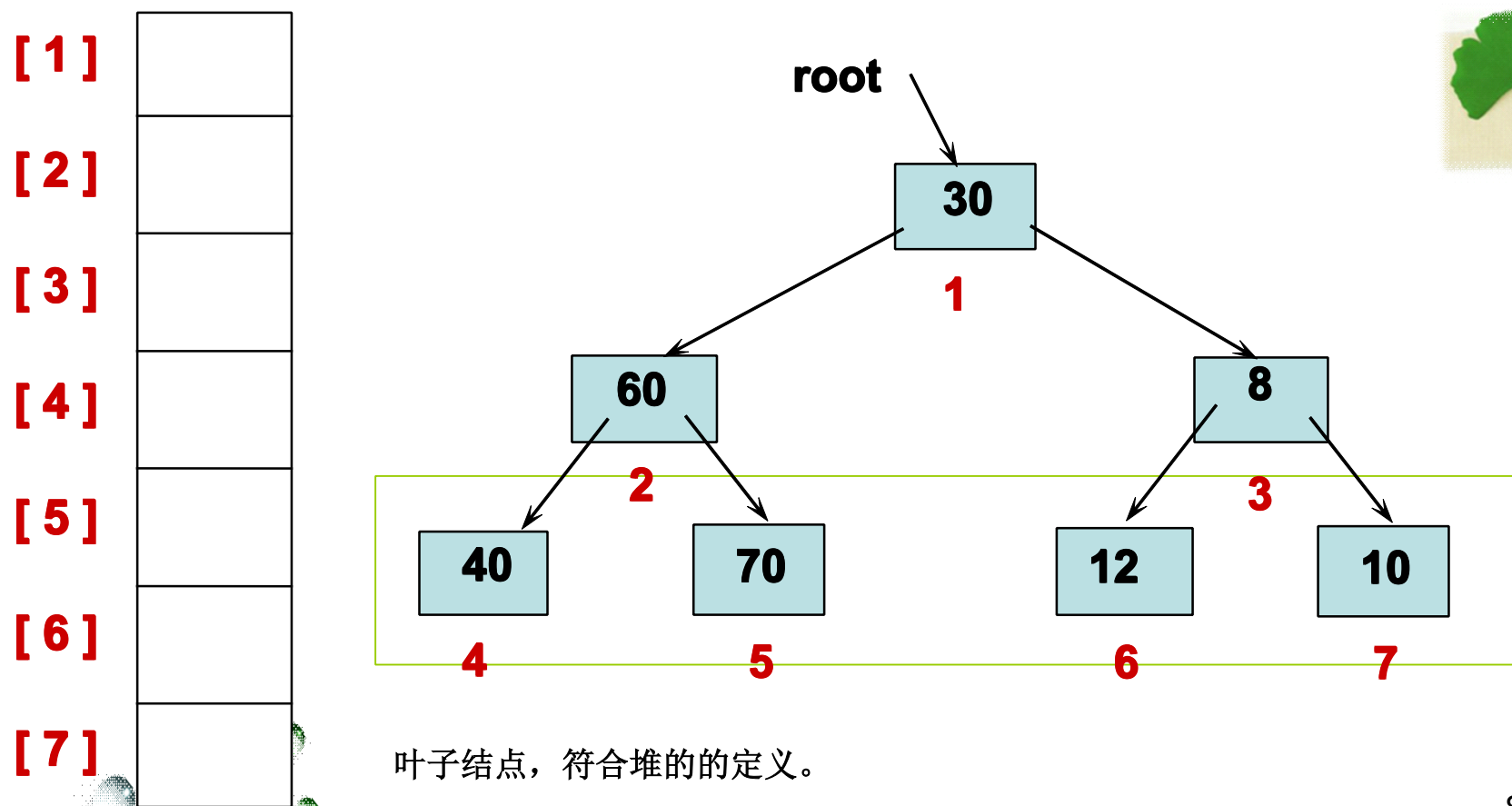
堆排序的关键问题



- 第二个问题解决方法——筛选
 - 方法：输出堆顶元素之后，以堆中最后一个元素替代之；然后将根结点值与左、右子树的根结点值进行比较，并与其中小者进行交换；重复上述操作，直至叶子结点，将得到新的堆，称这个从堆顶至叶子的调整过程为“筛选”。
 - 第一个问题解决方法——建堆
 - 方法：从无序序列的第 $\lfloor n/2 \rfloor$ 个元素（即此无序序列对应的完全二叉树的最后一个非终端结点）起，至第一个元素止，进行反复筛选。
- 

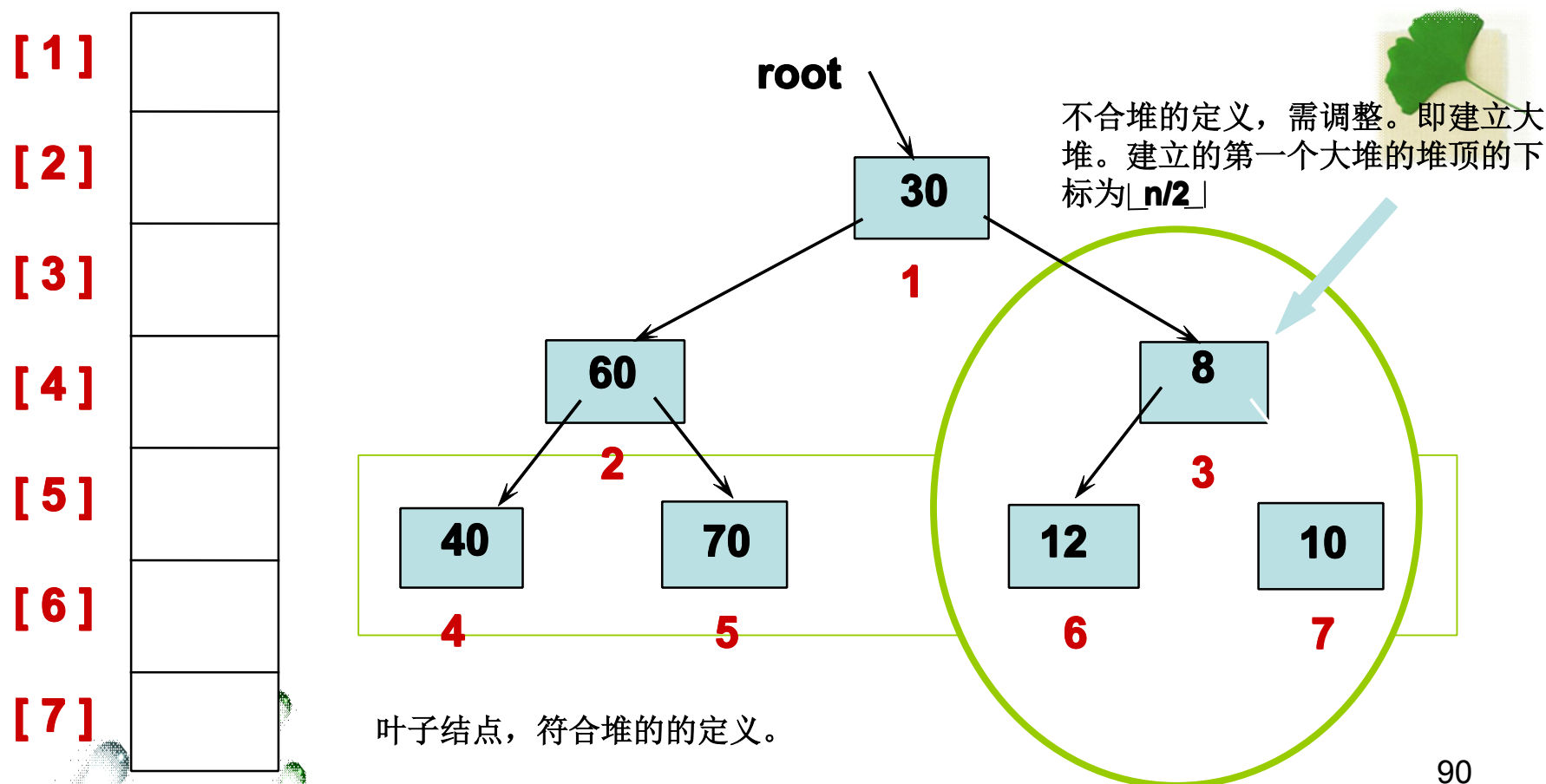
堆排序示例(以大顶堆为例)

(1) **建立初始堆** (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)



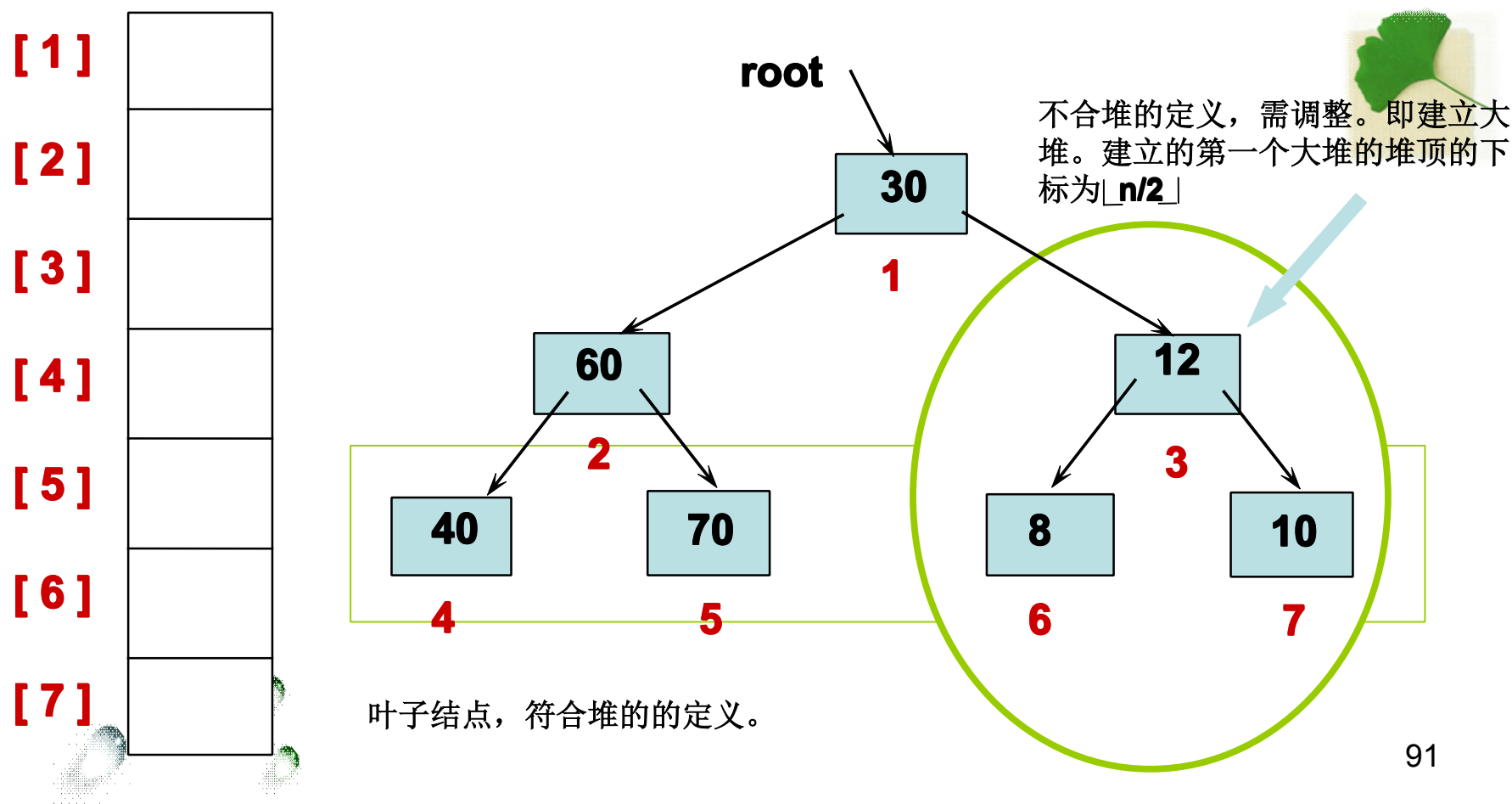
堆排序示例(以大顶堆为例)

(1) **建立初始堆** (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)



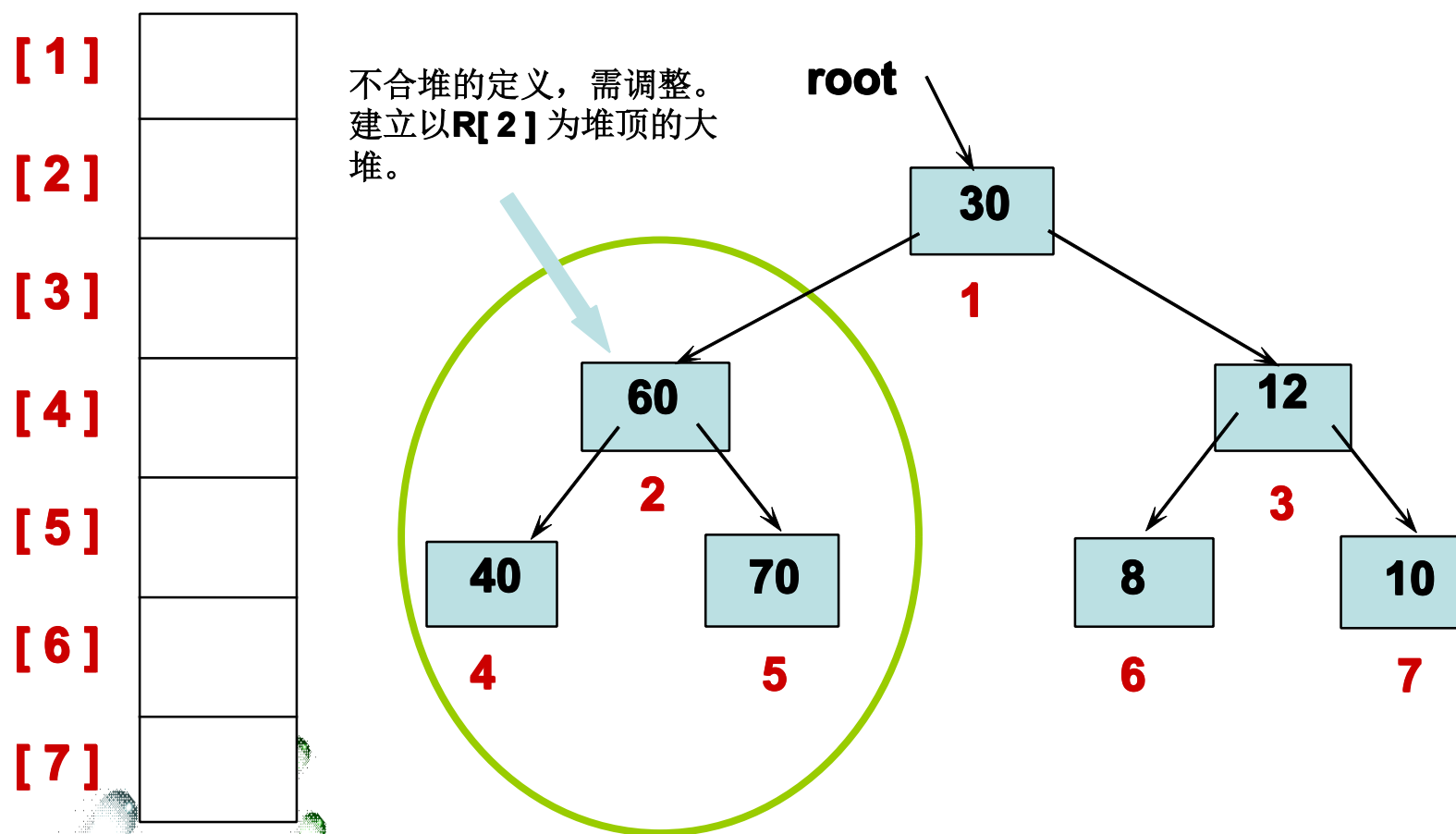
堆排序示例(以大顶堆为例)

(1) 建立初始堆 (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)



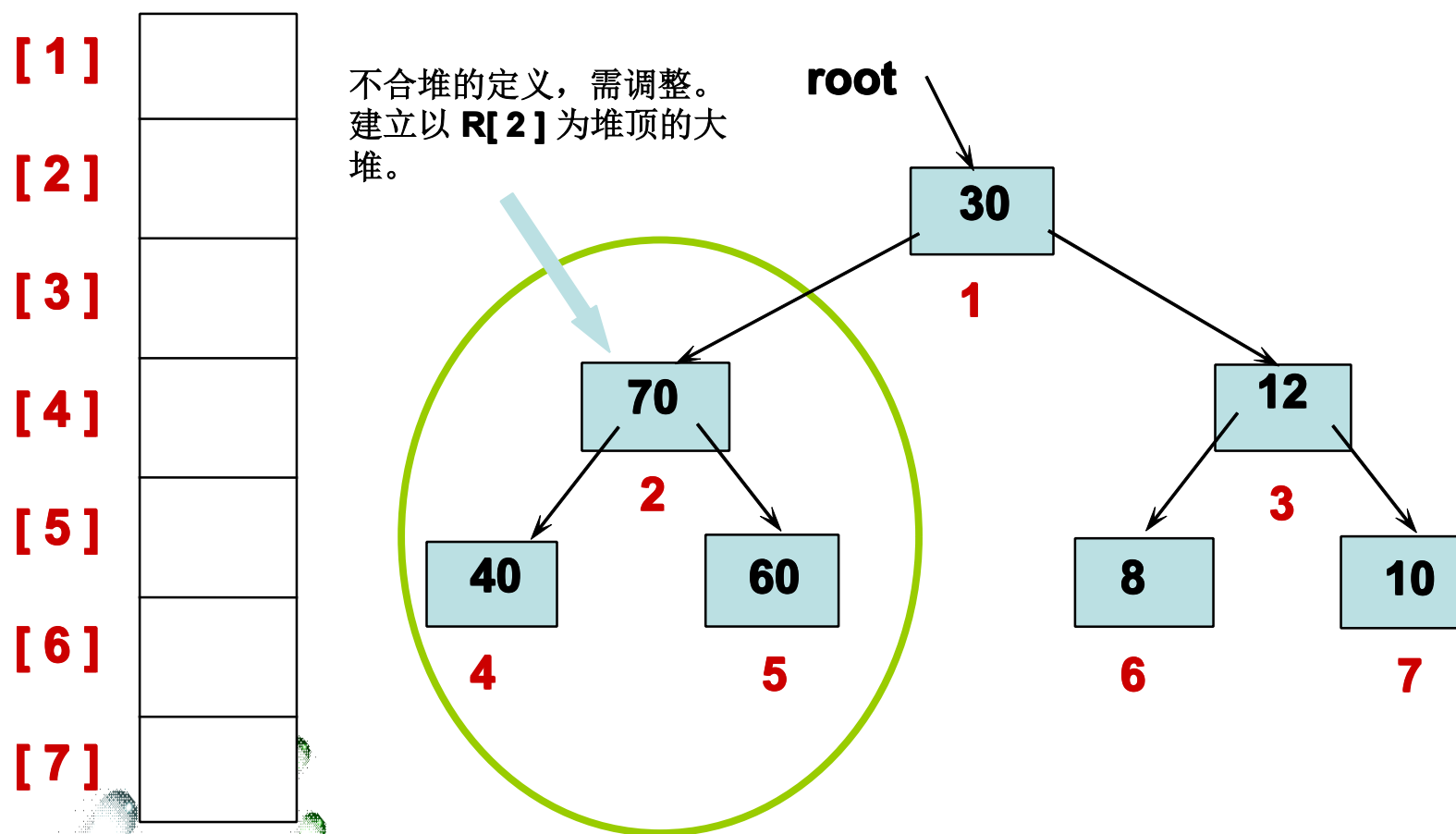
堆排序示例(以大顶堆为例)

(1) 建立初始堆 (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)



堆排序示例(以大顶堆为例)

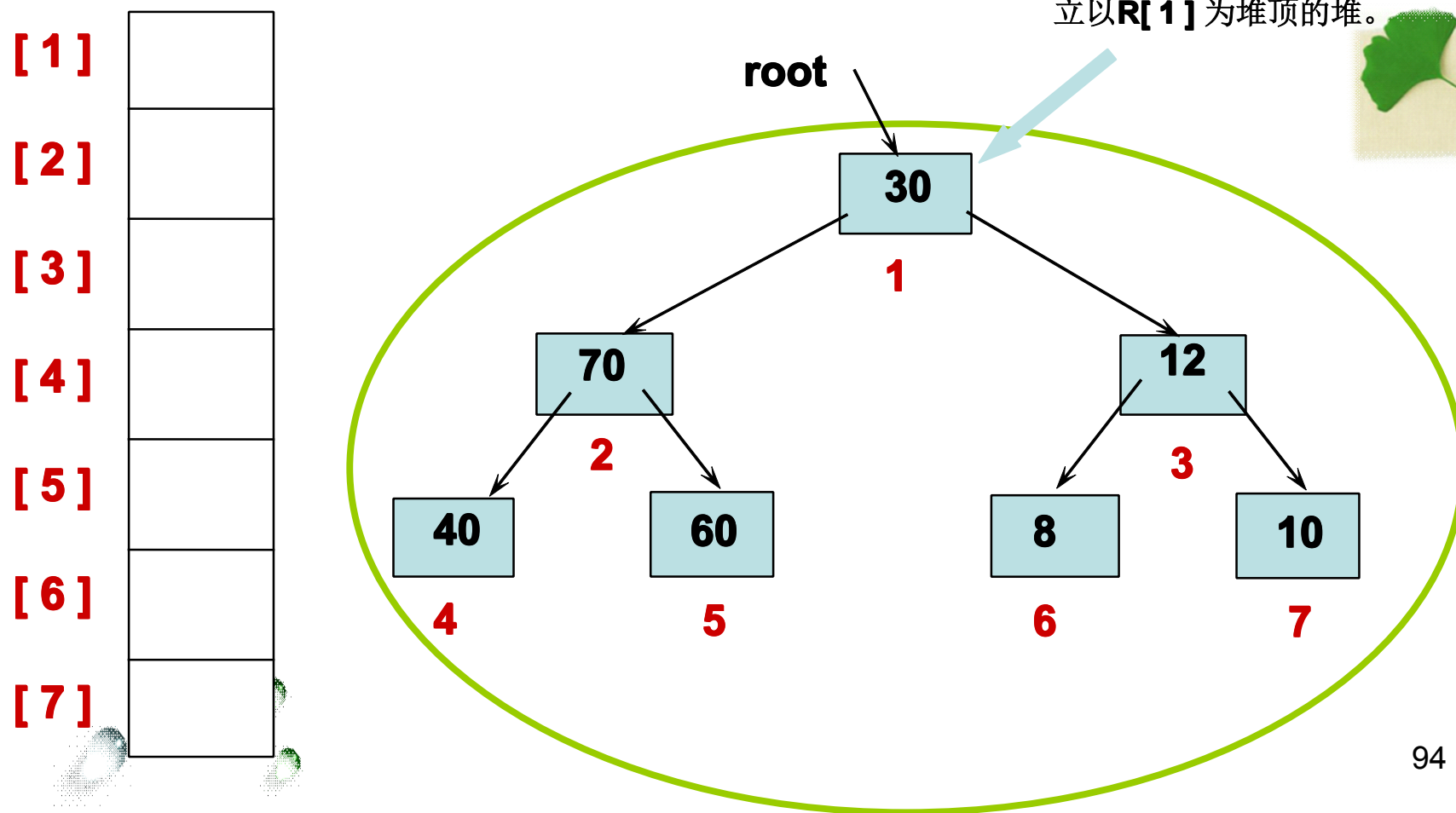
(1) 建立初始堆 (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)



堆排序示例(以大顶堆为例)

(1) 建立初始堆 (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)

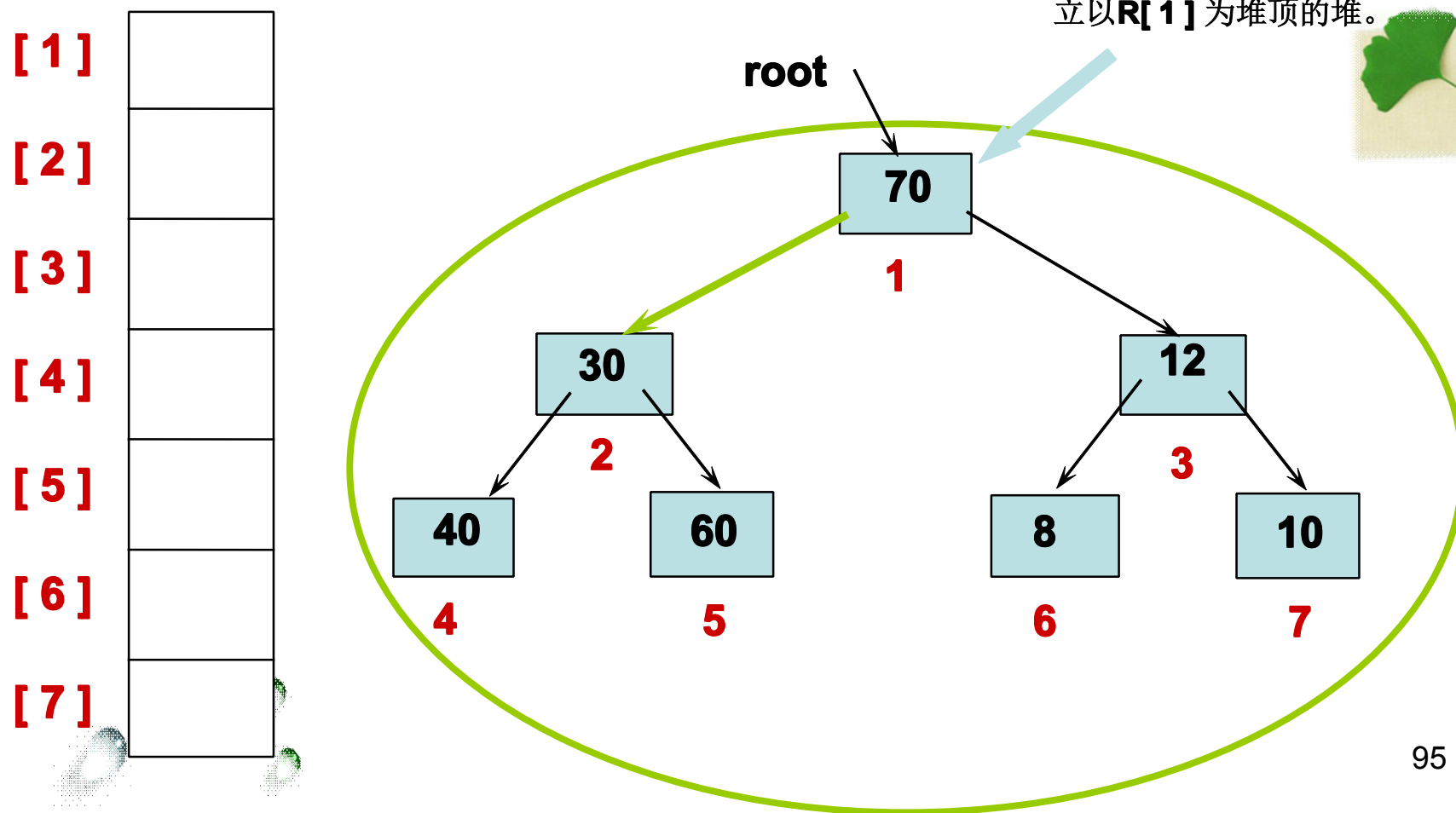
不合堆的定义, 需调整。建立以 **R[1]** 为堆顶的堆。



堆排序示例(以大顶堆为例)

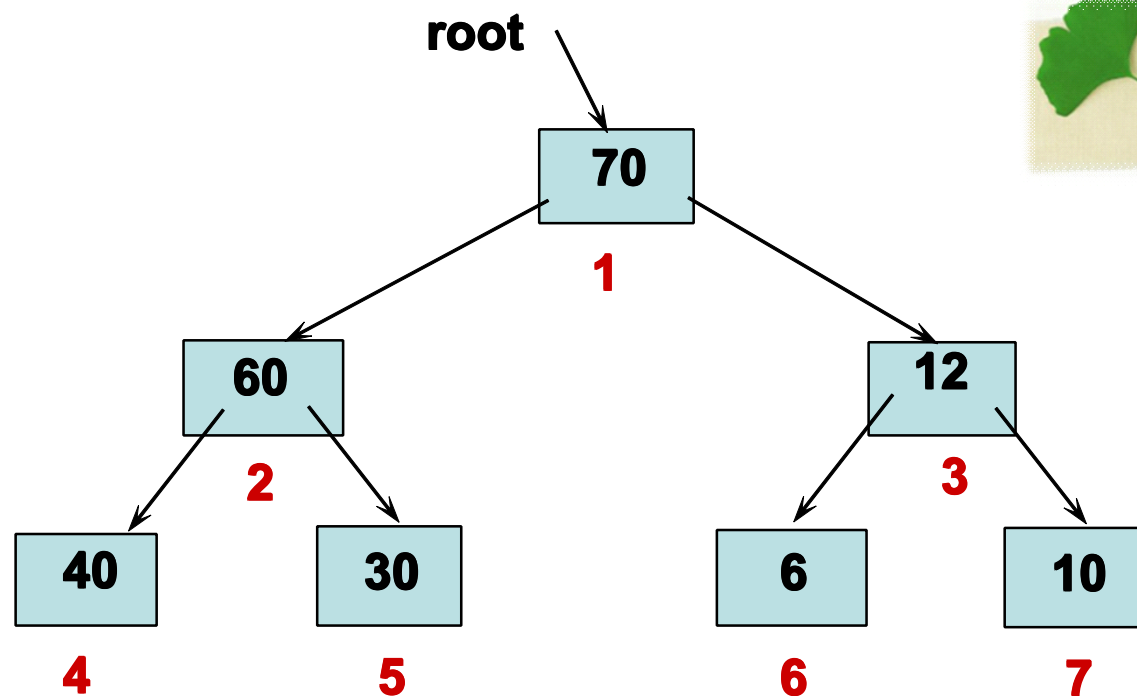
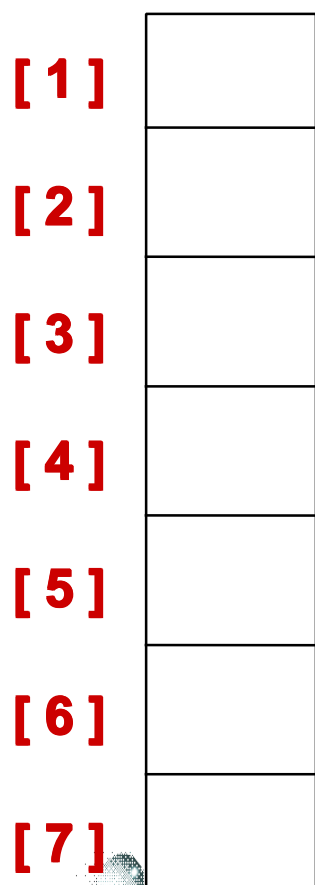
(1) 建立初始堆 (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)

不合堆的定义, 需调整。建立以 **R[1]** 为堆顶的堆。



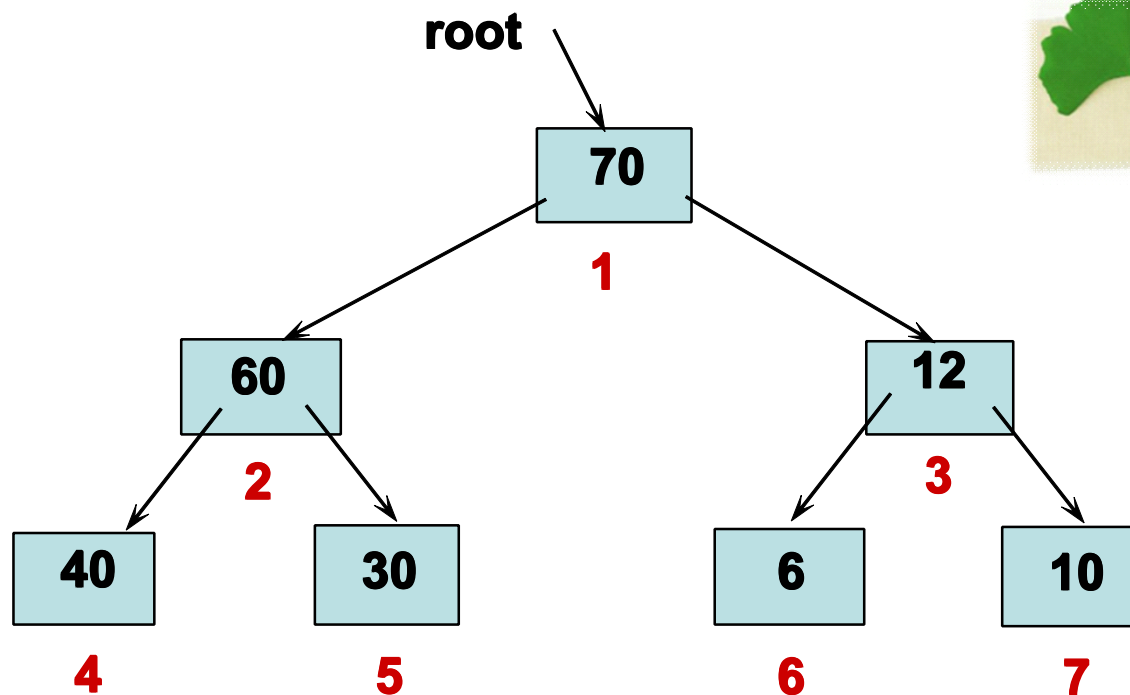
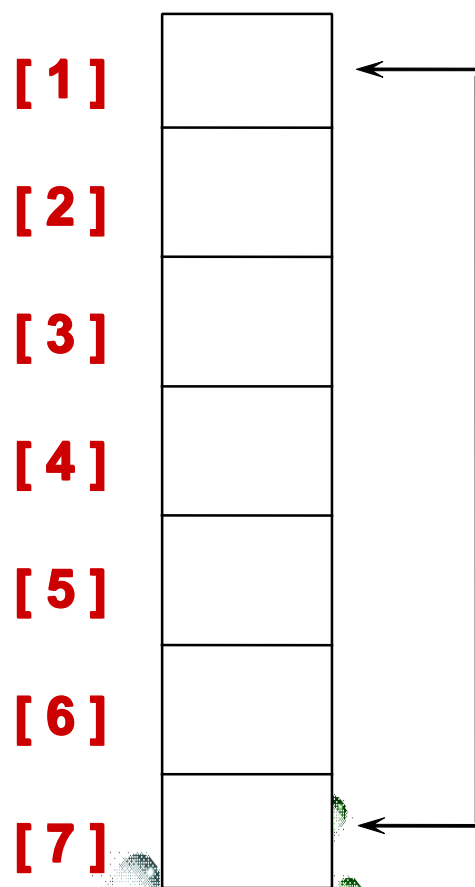
堆排序示例(以大顶堆为例)

(1) 建立初始堆 (把放在数组里的元素的序列看成是一棵完全二叉树, 对该二叉树进行调整, 使之成为堆)



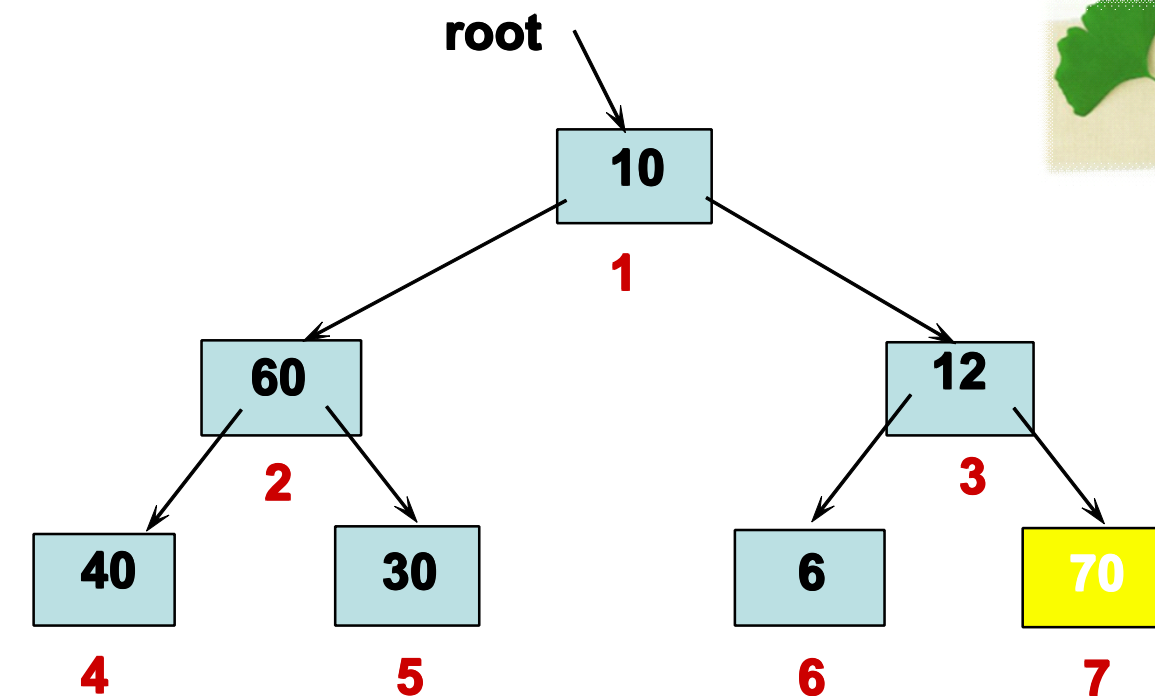
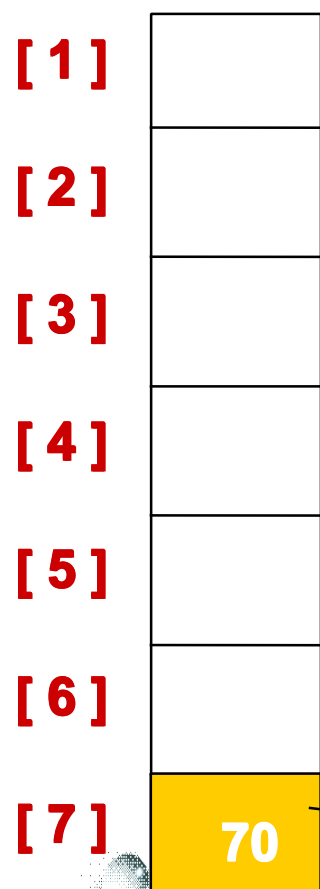
堆排序示例(以大顶堆为例)

(2) 堆排序



堆排序示例(以大顶堆为例)

(2) 堆排序

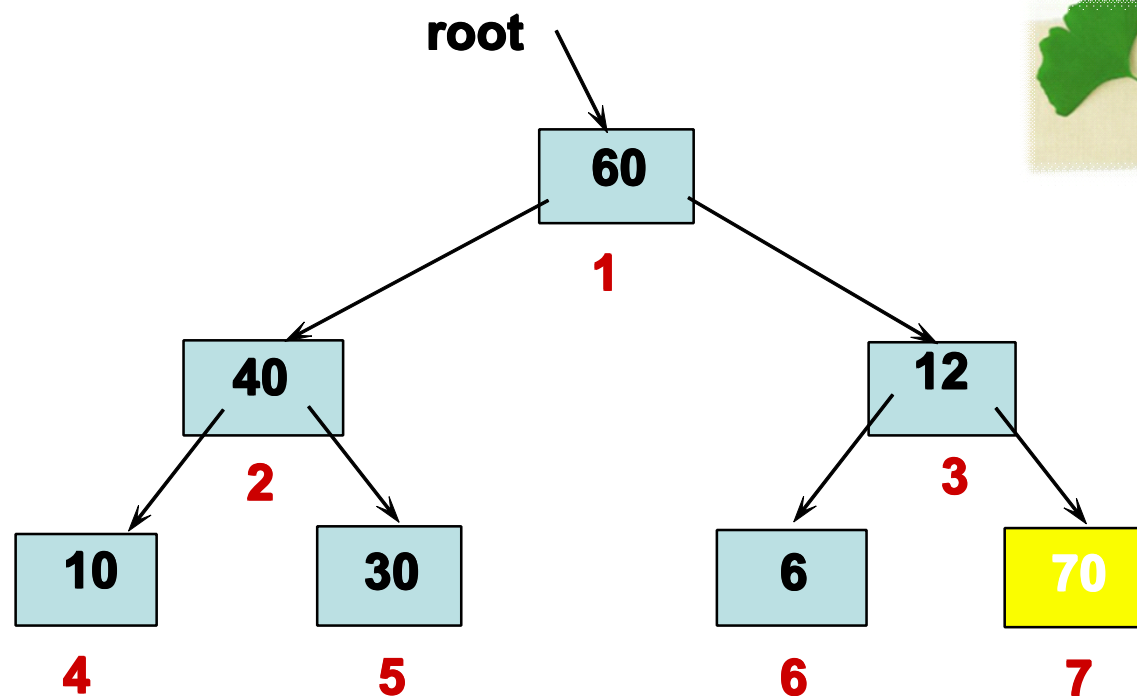


NO NEED TO CONSIDER AGAIN(就位)

堆排序示例(以大顶堆为例)

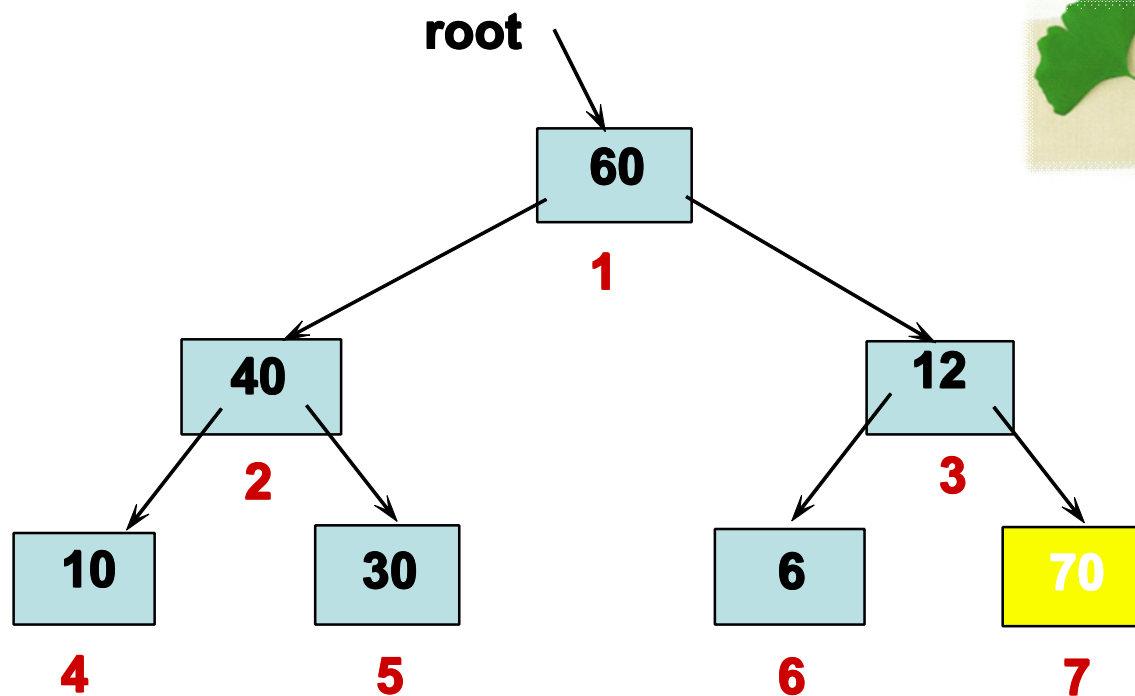
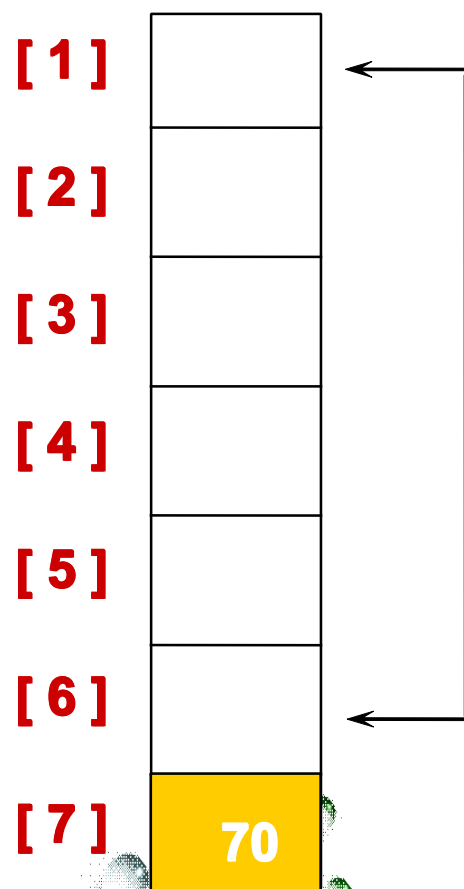
(2) 堆排序

[1]	
[2]	
[3]	
[4]	
[5]	
[6]	
[7]	70



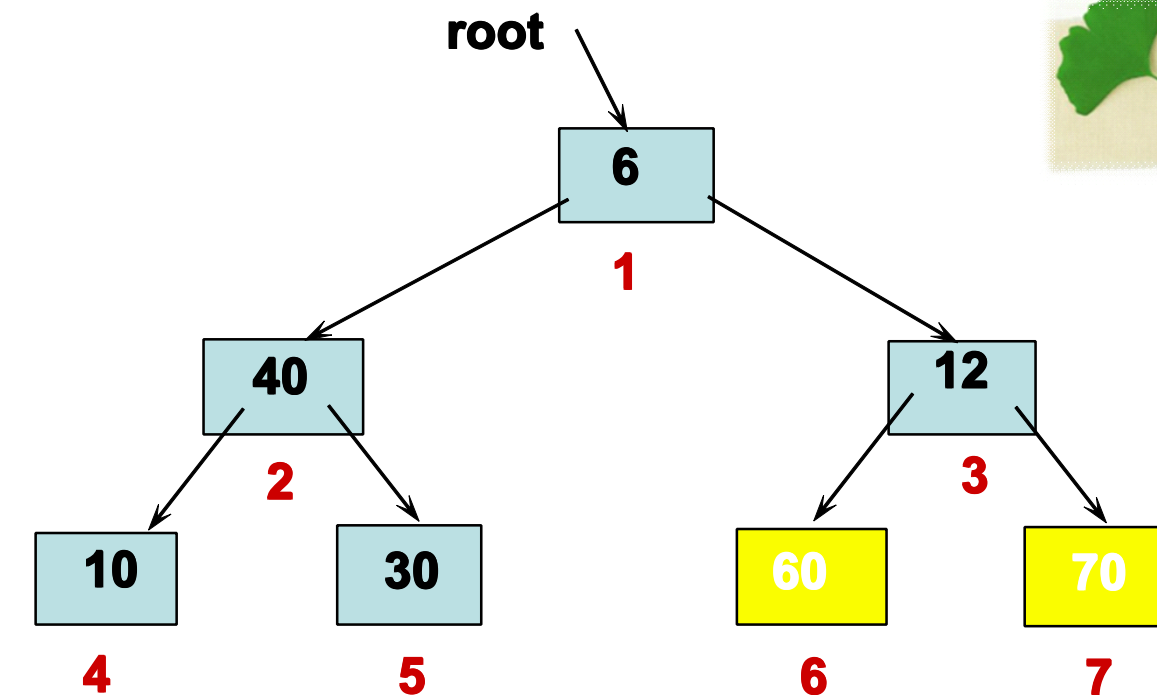
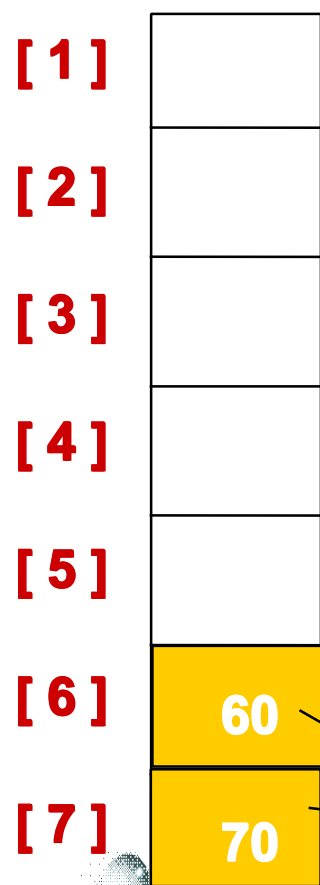
堆排序示例(以大顶堆为例)

(2) 堆排序

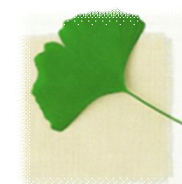


堆排序示例(以大顶堆为例)

(2) 堆排序



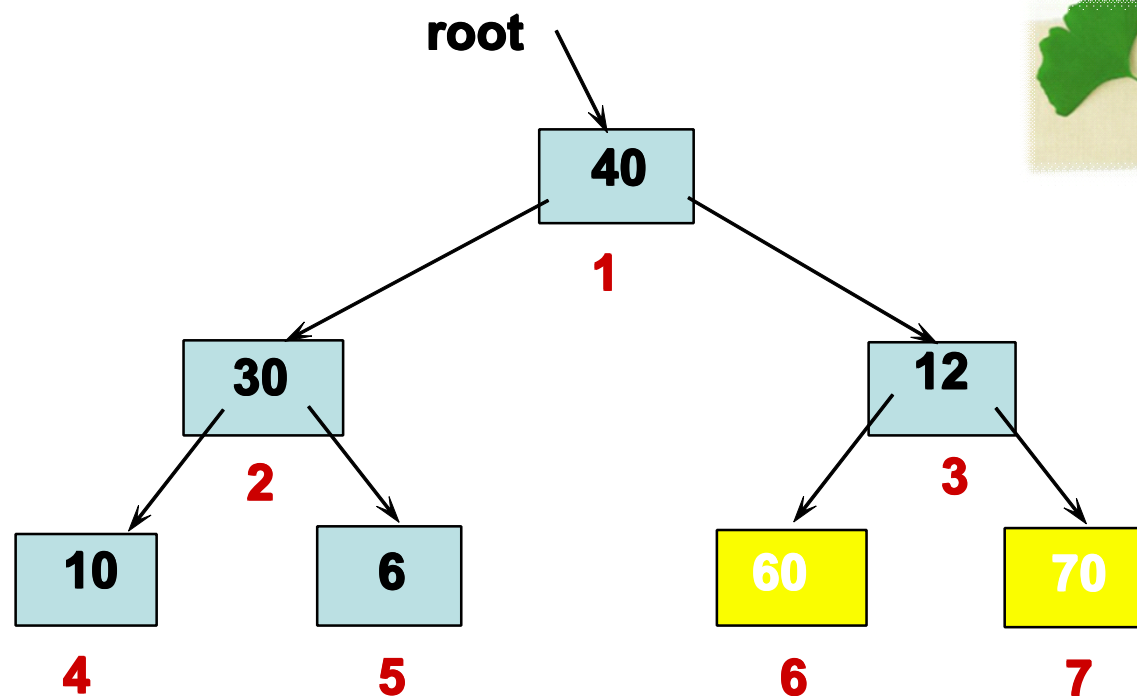
NO NEED TO CONSIDER AGAIN



堆排序示例(以大顶堆为例)

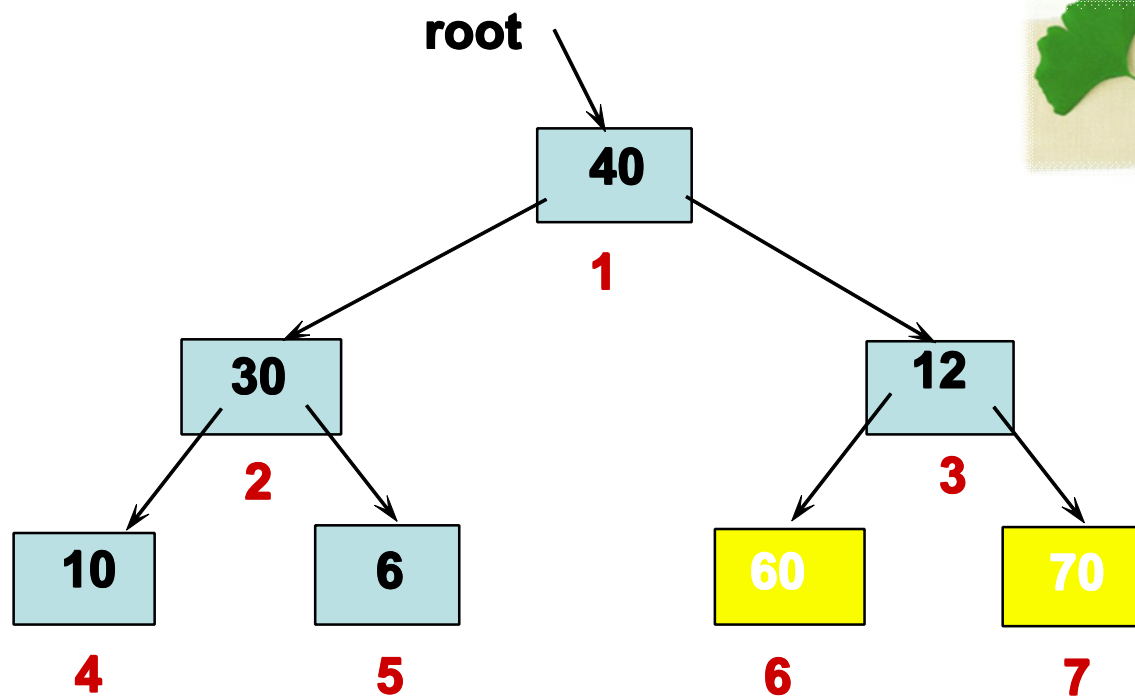
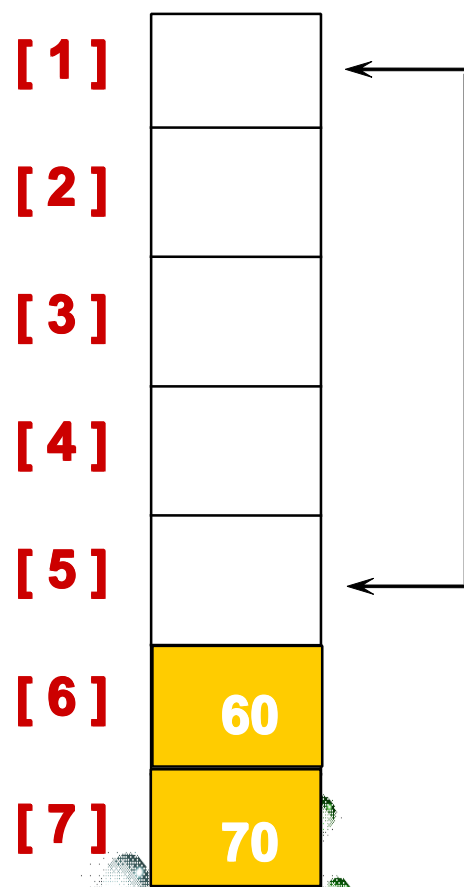
(2) 堆排序

[1]	
[2]	
[3]	
[4]	
[5]	
[6]	60
[7]	70



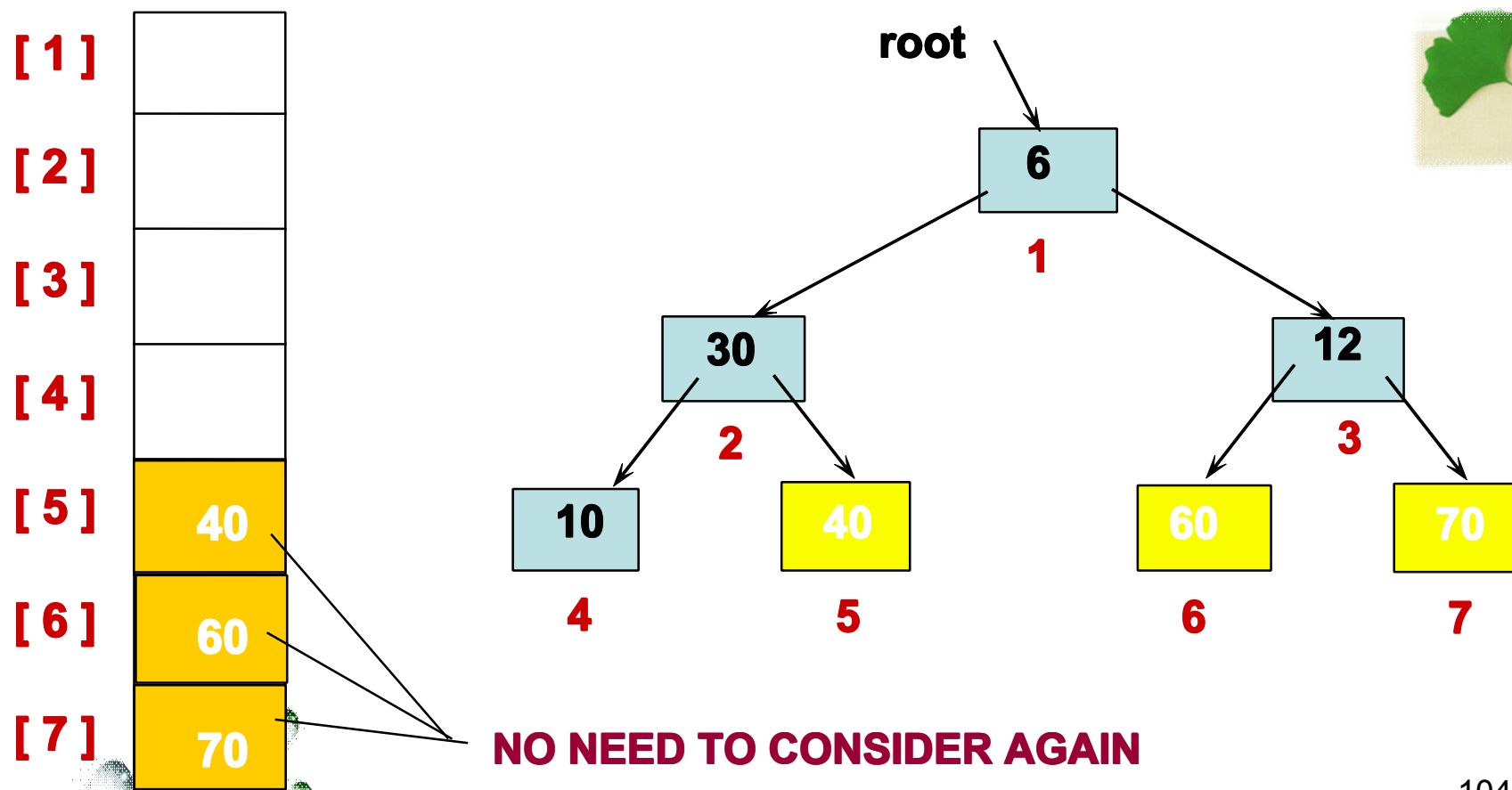
堆排序示例(以大顶堆为例)

(2) 堆排序



堆排序示例(以大顶堆为例)

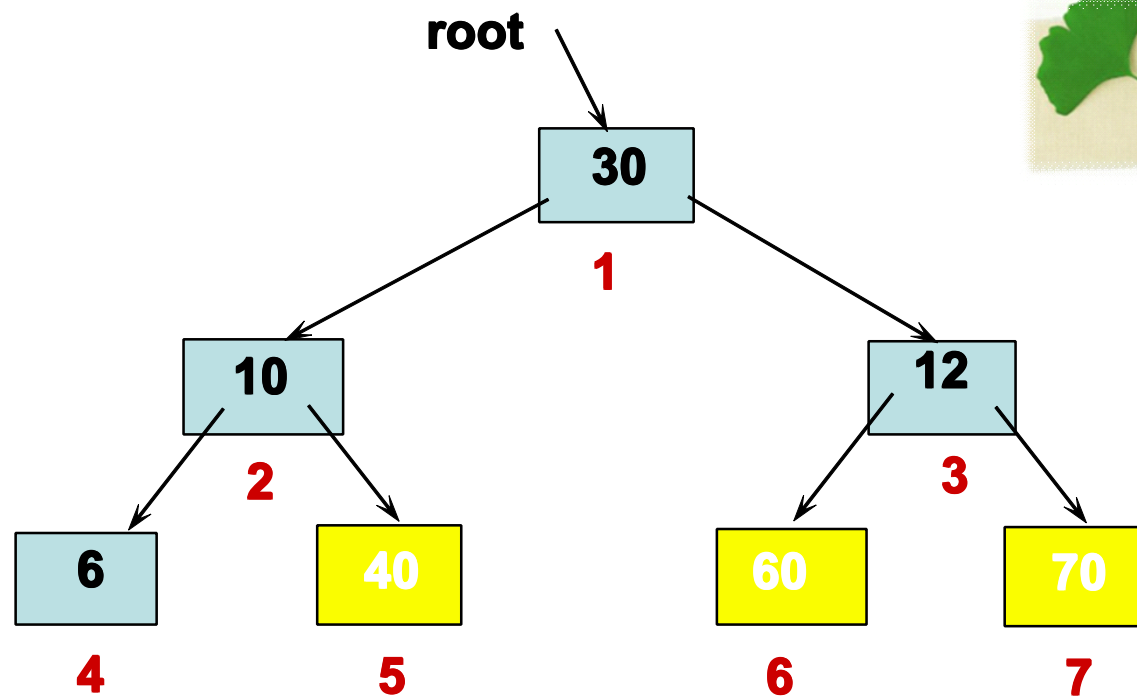
(2) 堆排序



堆排序示例(以大顶堆为例)

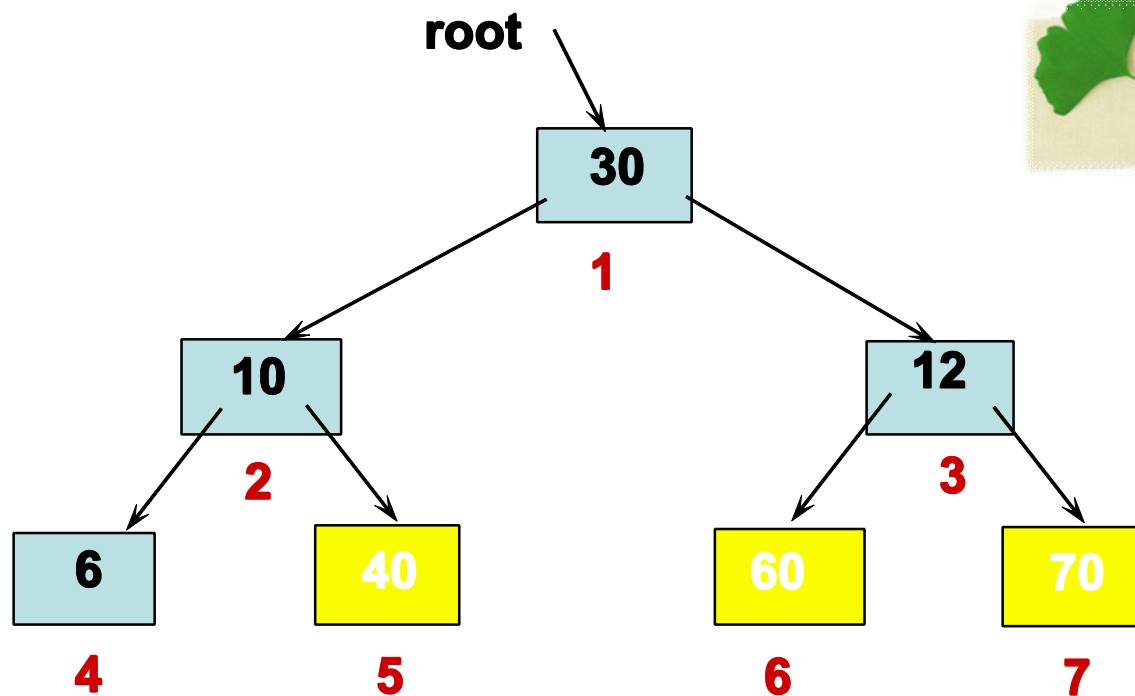
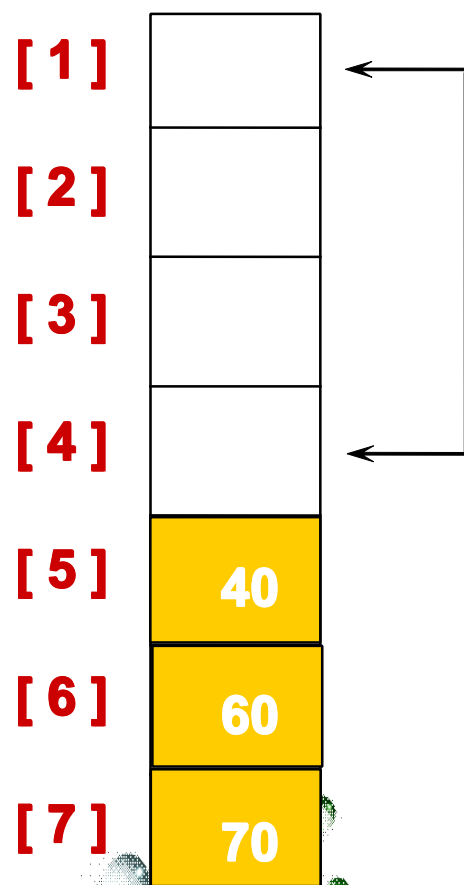
(2) 堆排序

[1]	
[2]	
[3]	
[4]	
[5]	40
[6]	60
[7]	70



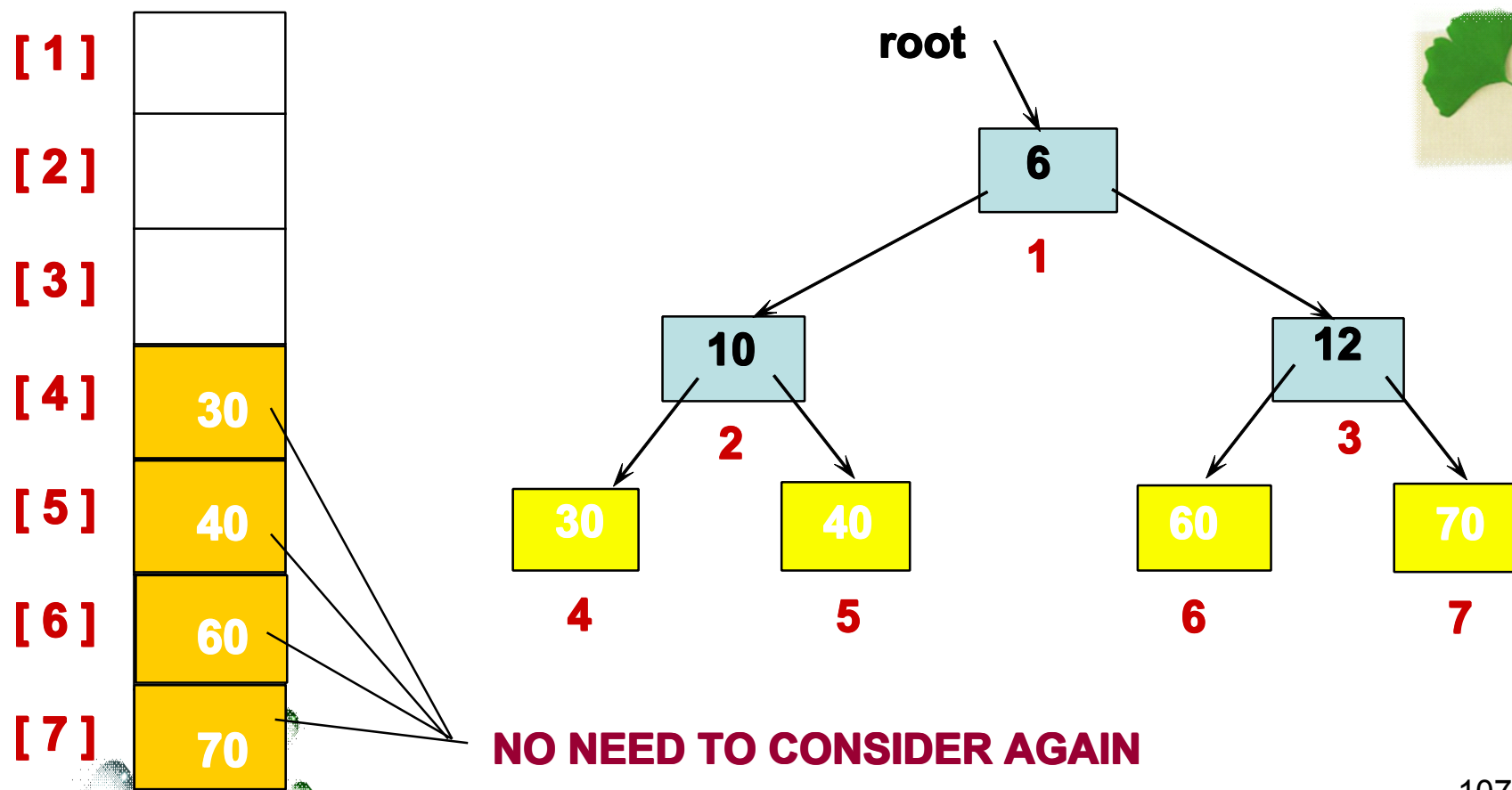
堆排序示例(以大顶堆为例)

(2) 堆排序



堆排序示例(以大顶堆为例)

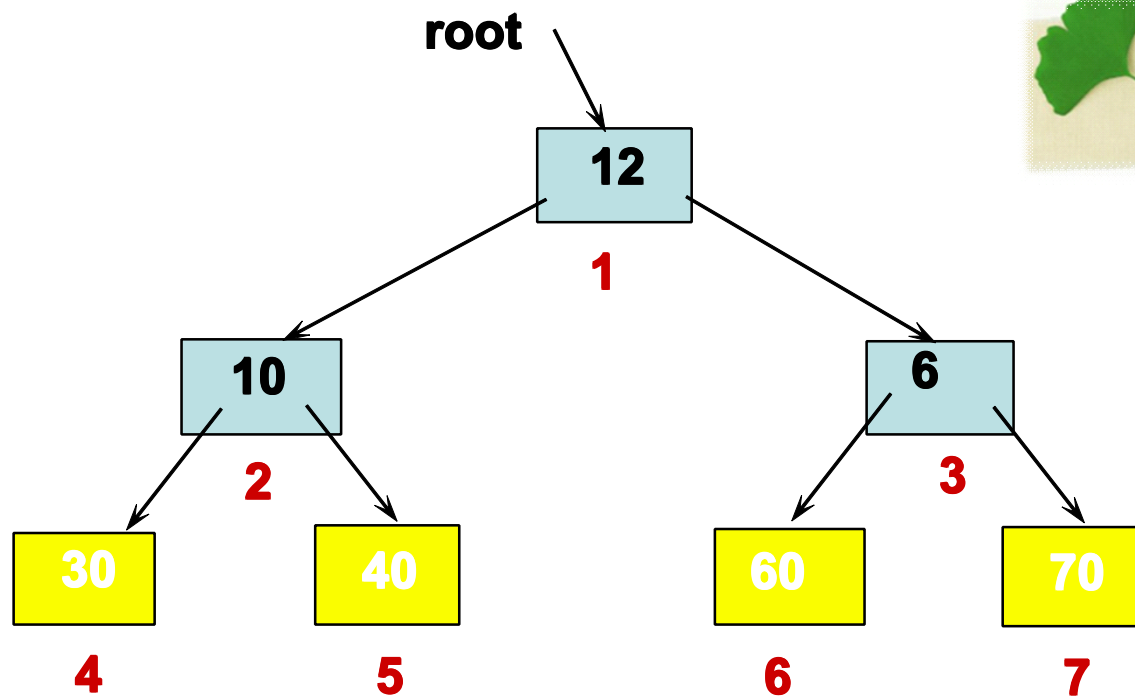
(2) 堆排序



堆排序示例(以大顶堆为例)

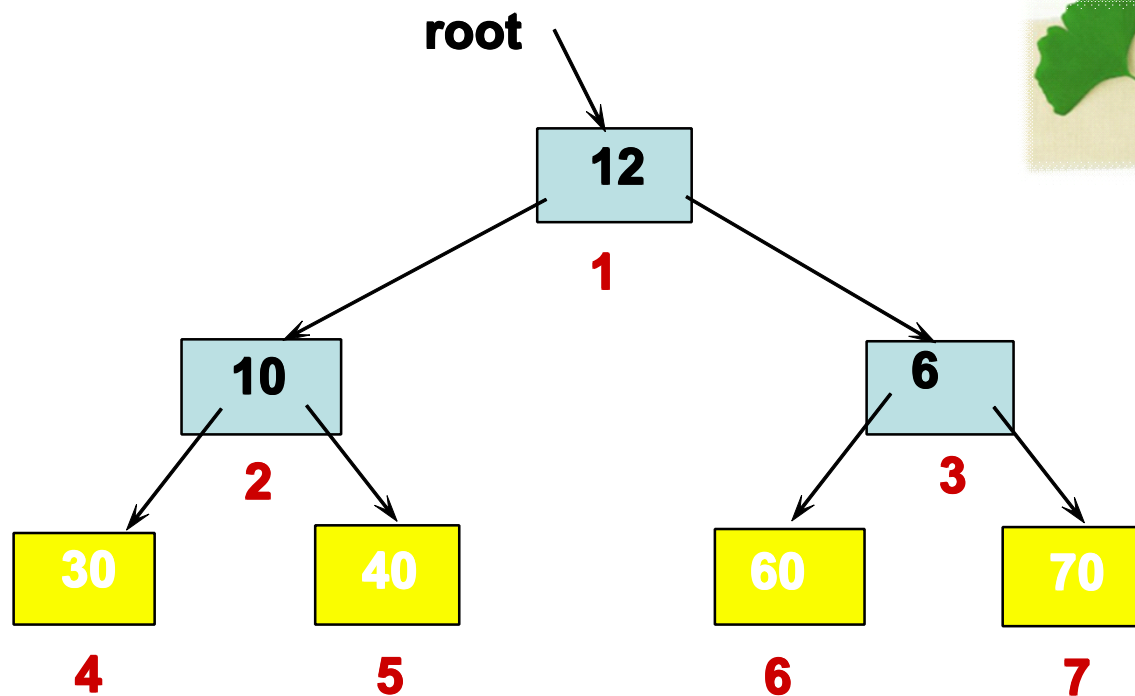
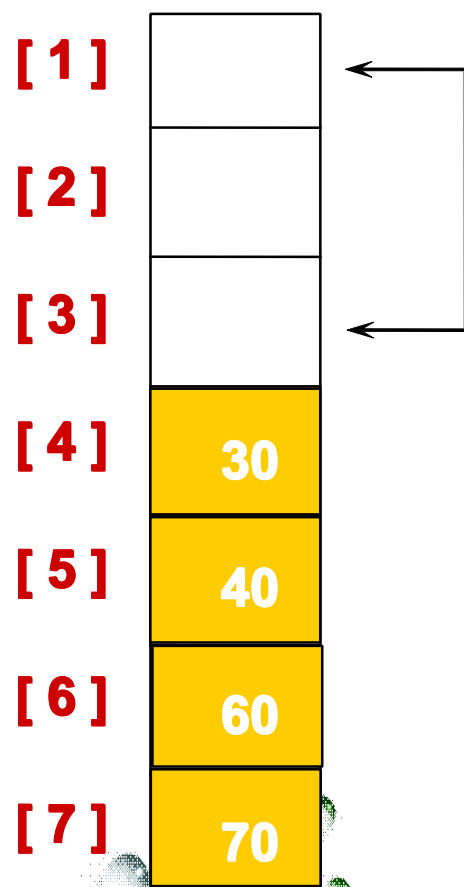
(2) 堆排序

[1]	
[2]	
[3]	
[4]	30
[5]	40
[6]	60
[7]	70



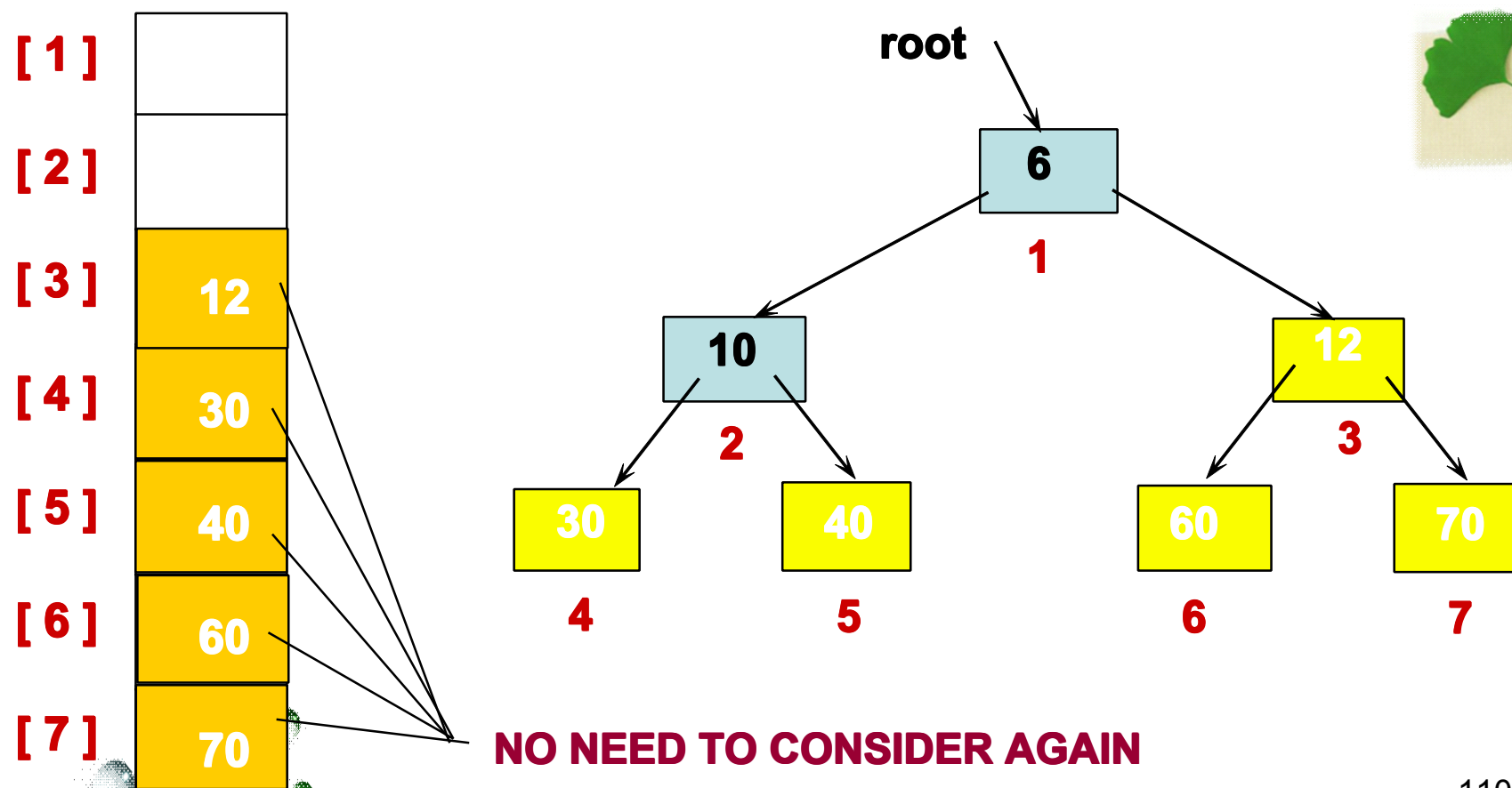
堆排序示例(以大顶堆为例)

(2) 堆排序



堆排序示例(以大顶堆为例)

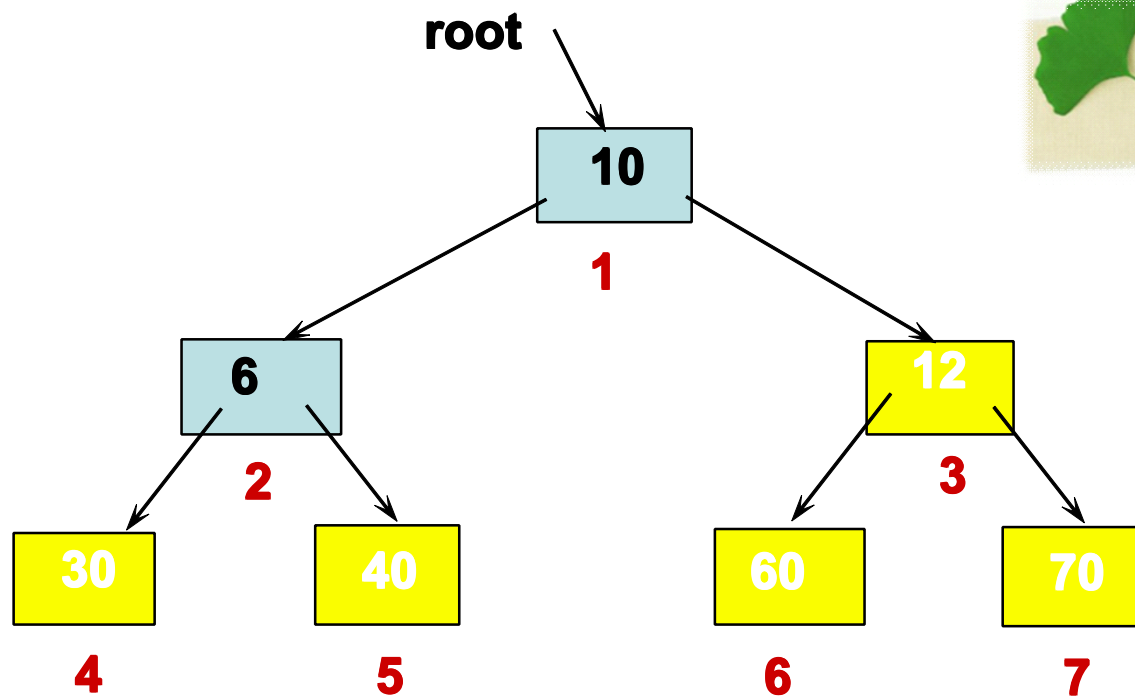
(2) 堆排序



堆排序示例(以大顶堆为例)

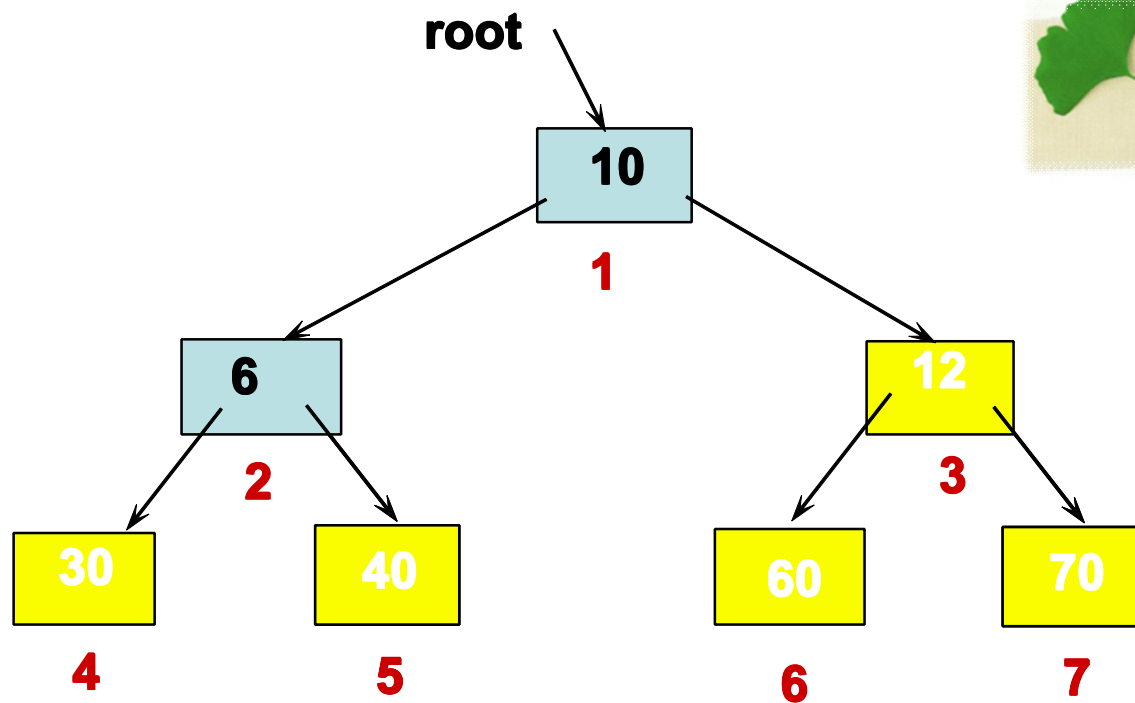
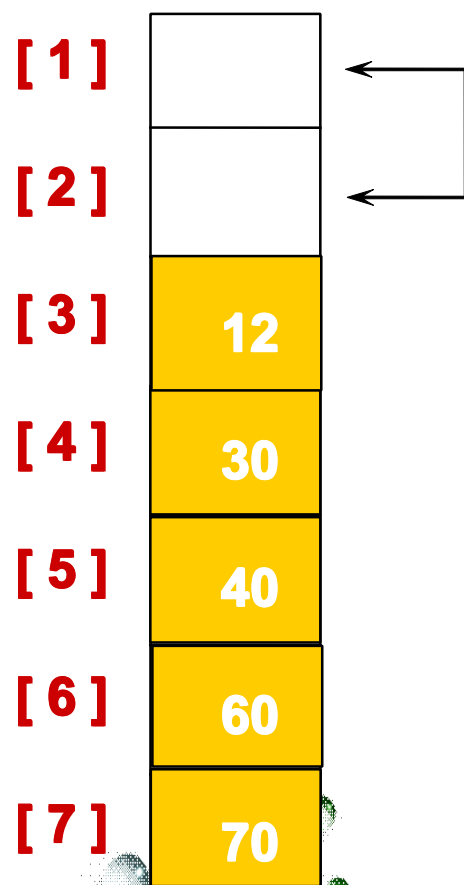
(2) 堆排序

[1]	
[2]	
[3]	12
[4]	30
[5]	40
[6]	60
[7]	70



堆排序示例(以大顶堆为例)

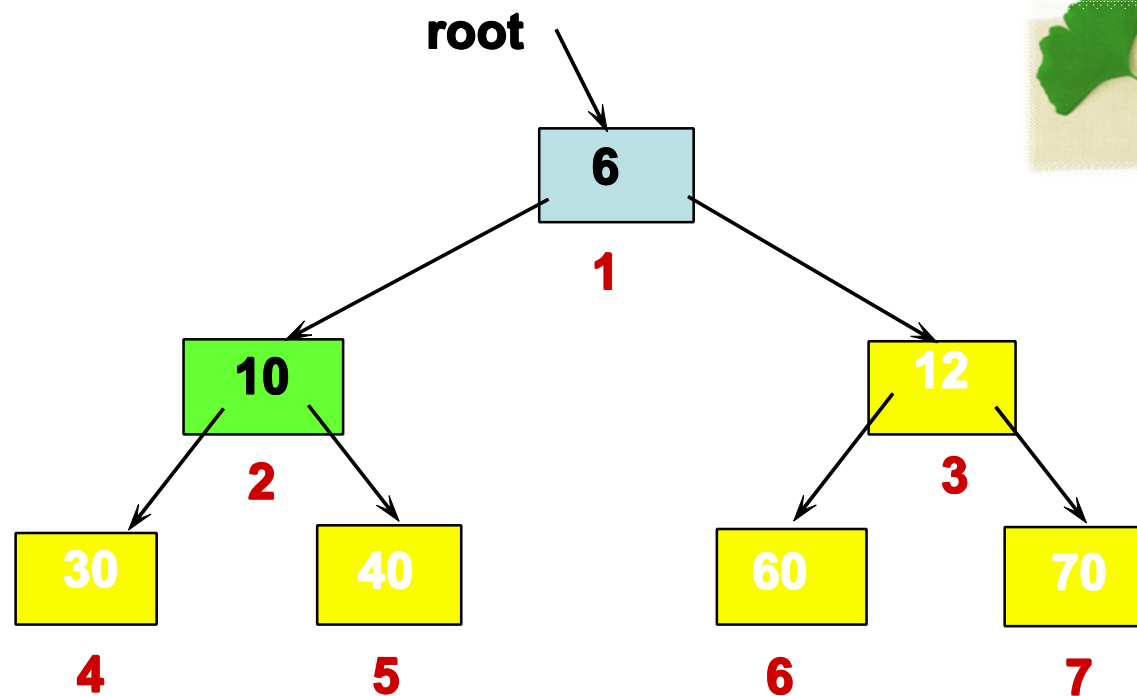
(2) 堆排序



堆排序示例(以大顶堆为例)

(2) 堆排序

[1]	
[2]	10
[3]	12
[4]	30
[5]	40
[6]	60
[7]	70



ALL ELEMENTS ARE SORTED

3. 堆排序算法(以大顶堆为例)

将完全二叉树
调整成一个堆(
即建初始堆)

```
void heapsort(int x[],int n)
{ for(i= n/2;i>=1;i--)
    HeapAdjust(R,i,n);
  for(i=n;i>1;i--)
  { R[0]=R[1];
    R[1]=R[i];
    R[i]=R[0];
    HeapAdjust(R,1,i-1);
  }
}
```

元素交换(
将堆顶元
素放到其
最终位置)

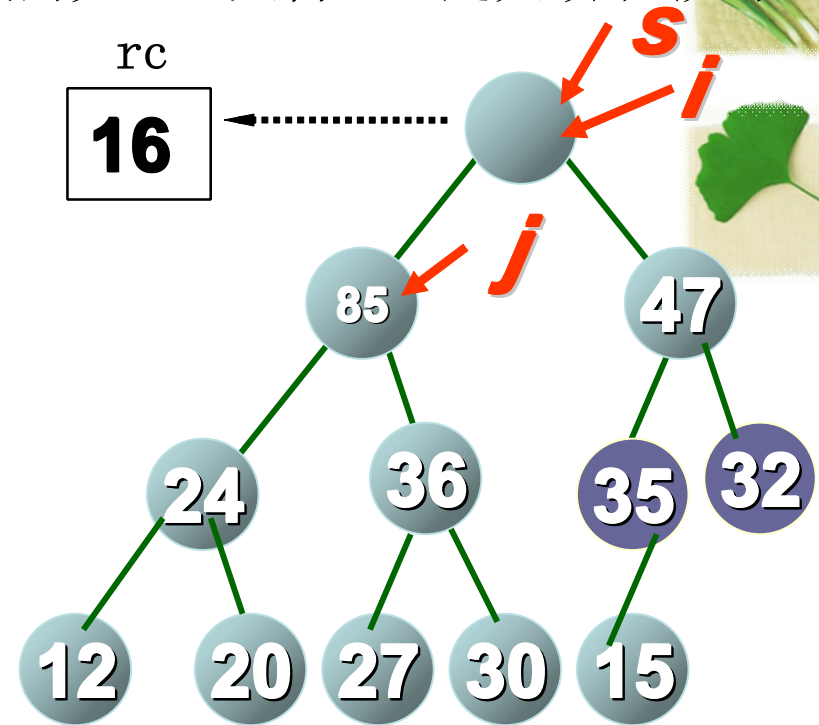
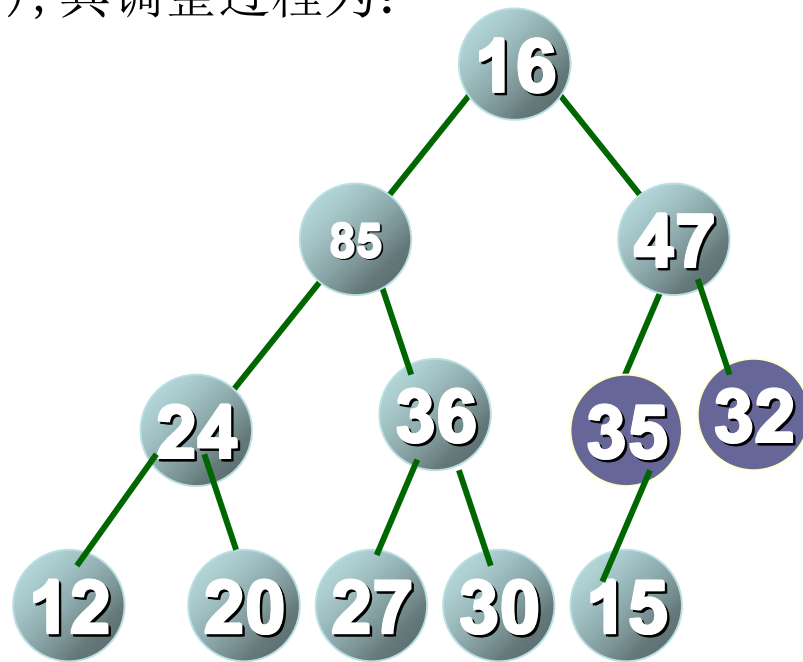
将第1个到第
i-1个元素调
整成一个堆

```
void HeapAdjust(datatype R[],int s,int t)
{ rc=R[s]; /*用rc暂存R[S]*/
  i=s; j=2*i;
  while (j<=t)
  { if (j<t && R[j+1].key>R[j].key)
      j=j+1;
    if (rc.key>R[j].key)
      break;
    else
    { R[i]=R[j]; /*j号元素上移*/
      i=j;
      j=2*i; }
  }
  R[i]=rc; /*将rc中的元素放到R[i]中*/
}
```

寻找结点i
的大孩子j

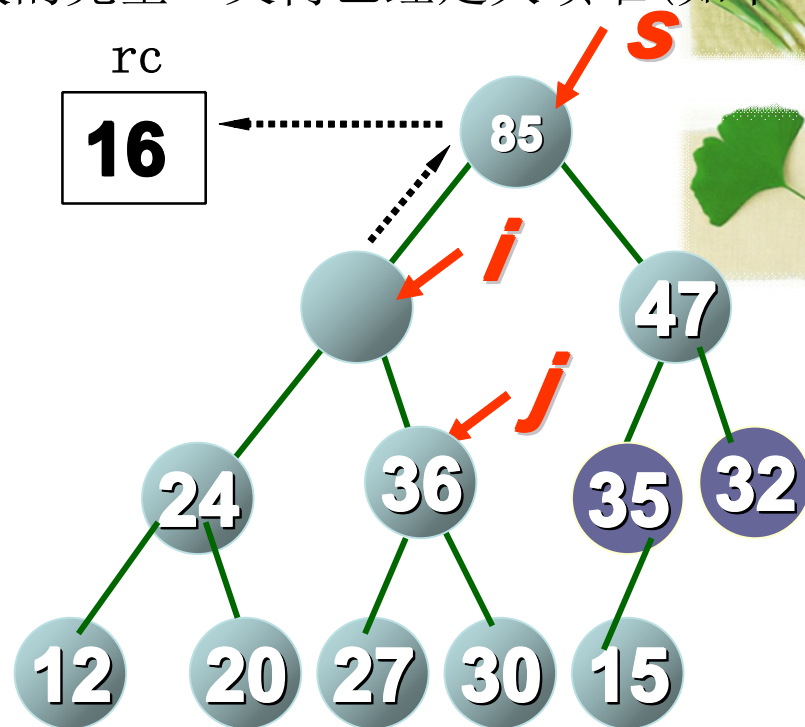
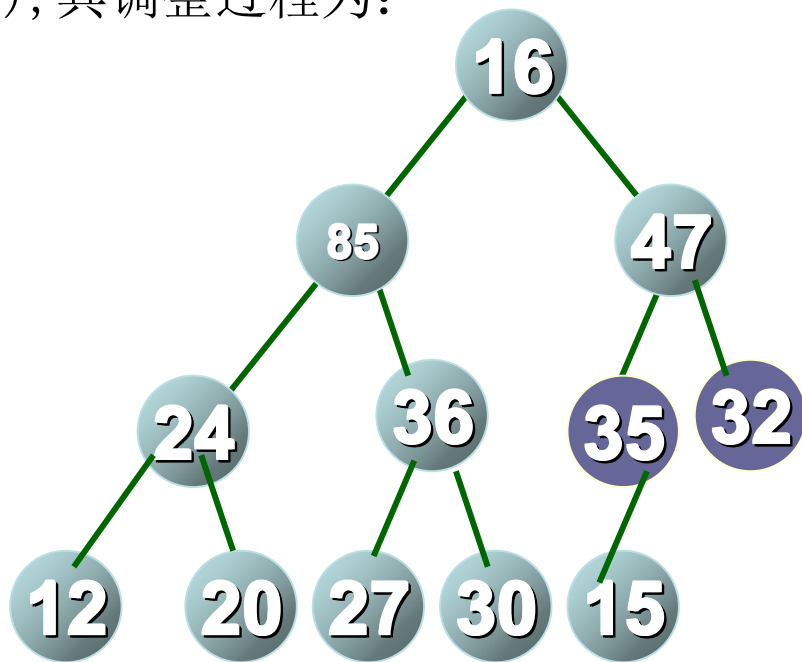
HeapAdjust算法的执行过程

算法HeapAdjust的作用是将以第s个结点为根的完全二叉树调整成一个大顶堆,其前提是以第s个结点左右孩子为根的完全二叉树已经是大顶堆。假定 $s=1$,算法HeapAdjust的作用是将以第1个结点为根的完全二叉树调整成一个大顶堆,其前提是以第2、3个结点为根的完全二叉树已经是大顶堆(如下图),其调整过程为:



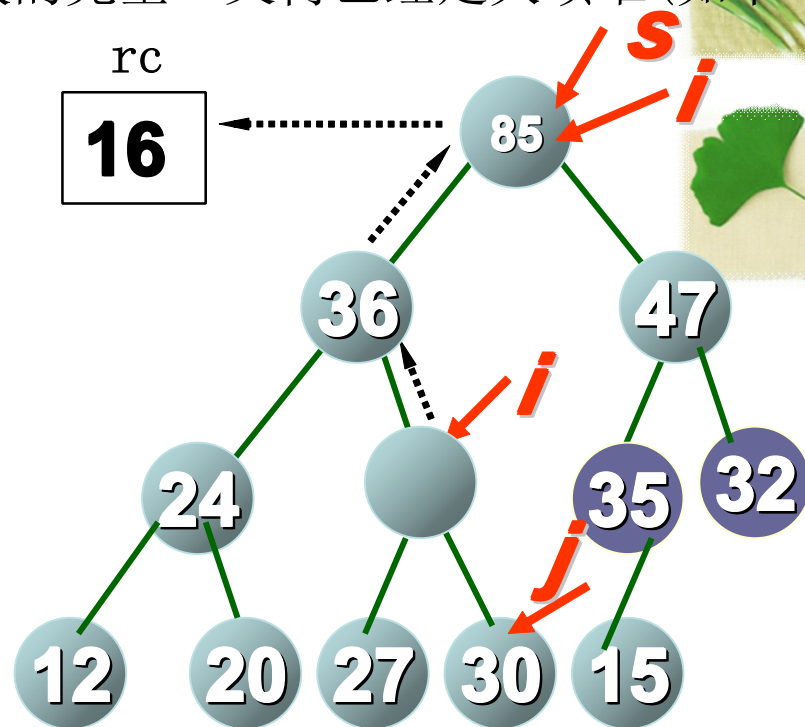
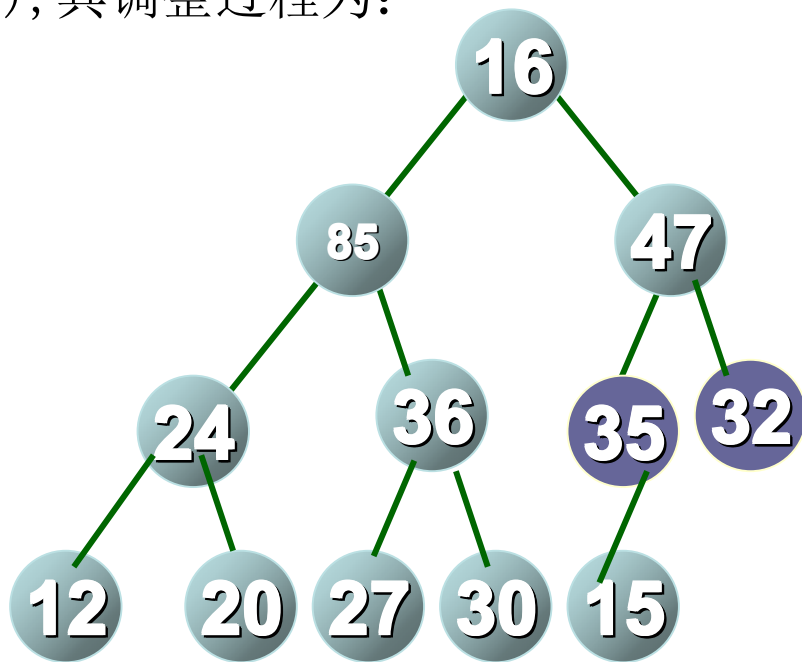
HeapAdjust算法的执行过程

算法HeapAdjust的作用是将以第s个结点为根的完全二叉树调整成一个大顶堆,其前提是以第s个结点左右孩子为根的完全二叉树已经是大顶堆。假定 $s=1$,算法HeapAdjust的作用是将以第1个结点为根的完全二叉树调整成一个大顶堆,其前提是以第2、3个结点为根的完全二叉树已经是大顶堆(如下图),其调整过程为:



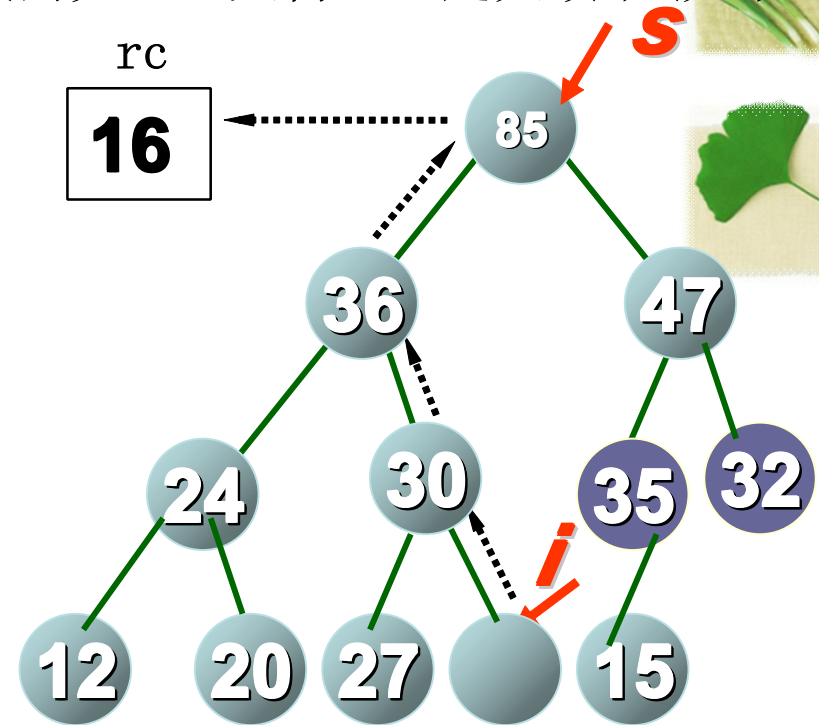
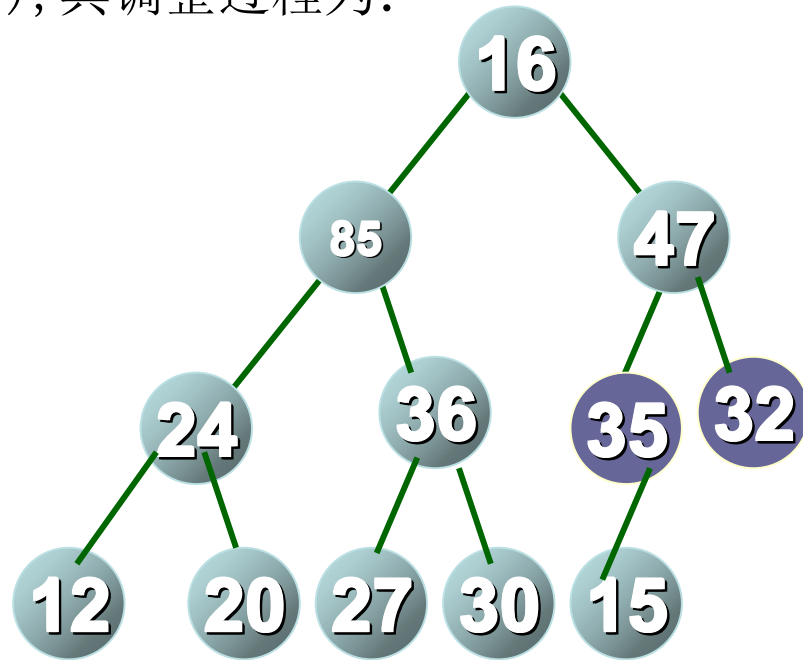
HeapAdjust算法的执行过程

算法HeapAdjust的作用是将以第s个结点为根的完全二叉树调整成一个大顶堆,其前提是以第s个结点左右孩子为根的完全二叉树已经是堆。假定 $s=1$,算法HeapAdjust的作用是将以第1个结点为根的完全二叉树调整成一个大顶堆,其前提是以第2、3个结点为根的完全二叉树已经是堆(如下图),其调整过程为:



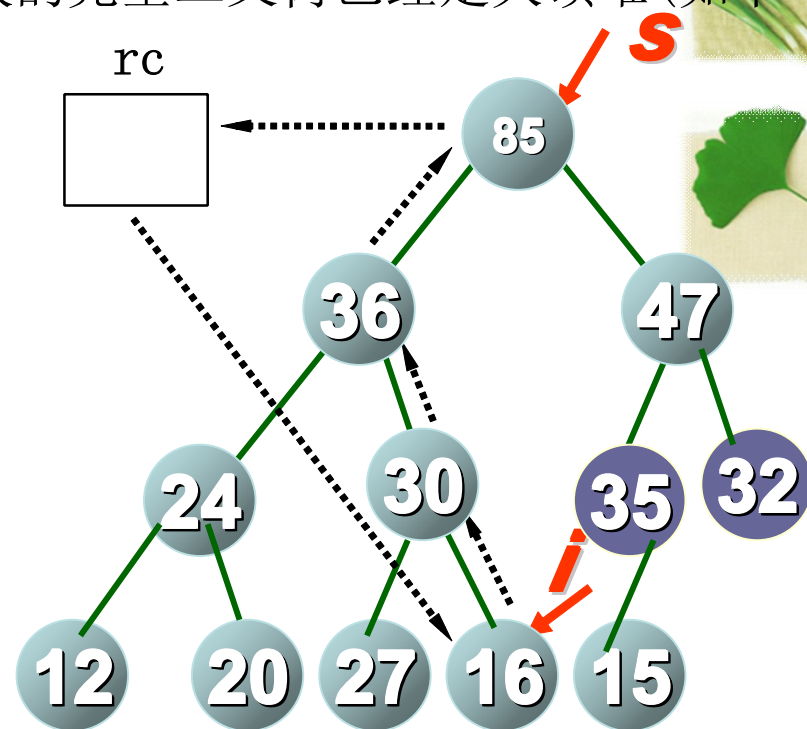
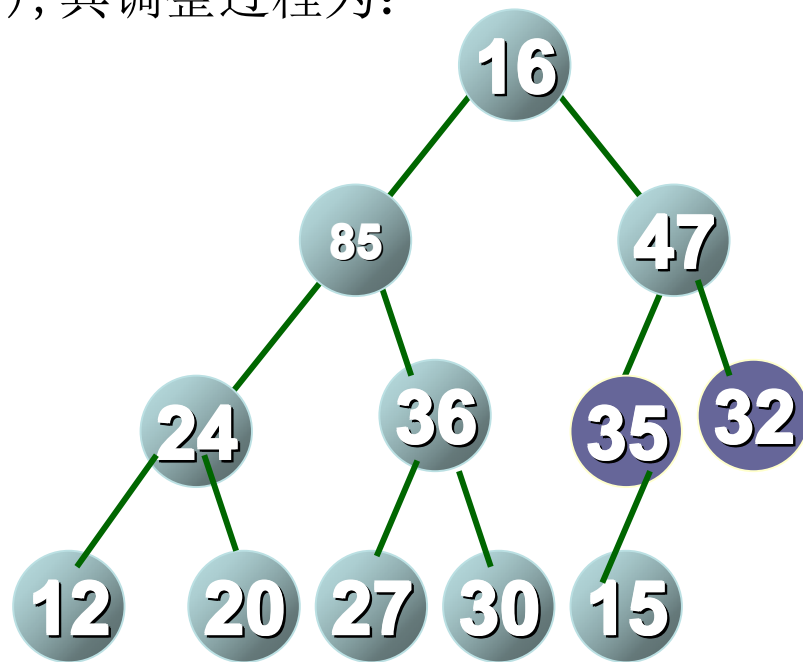
HeapAdjust算法的执行过程

算法HeapAdjust的作用是将以第s个结点为根的完全二叉树调整成一个大顶堆,其前提是以第s个结点左右孩子为根的完全二叉树已经是堆。假定 $s=1$,算法HeapAdjust的作用是将以第1个结点为根的完全二叉树调整成一个大顶堆,其前提是以第2、3个结点为根的完全二叉树已经是堆(如下图),其调整过程为:



HeapAdjust算法的执行过程

算法HeapAdjust的作用是将以第s个结点为根的完全二叉树调整成一个大顶堆,其前提是以第s个结点左右孩子为根的完全二叉树已经是大顶堆。假定 $s=1$,算法HeapAdjust的作用是将以第1个结点为根的完全二叉树调整成一个大顶堆,其前提是以第2、3个结点为根的完全二叉树已经是大顶堆(如下图),其调整过程为:



可见, HeapAdjust的执行过程是先用rc暂存R[S], 然后以元素的移动代替元素间的直接交换。这样做可大大减少所需执行的语句, 提高算法的效率。如直接交换3(5)次需执行9(15)个语句, 而采用这种方法仅需执行5(7)个语句。在排序算法中这种技巧得到了广泛的应用。

堆排序的效率分析

在整个堆排序中，共需要进行 $\lfloor n/2 \rfloor + n - 1$ 次筛选运算，每次筛选运算进行双亲和孩子或兄弟结点的排序码的比较和移动次数都不会超过完全二叉树的深度 $\lfloor \log_2 n \rfloor + 1$ ，所以，每次筛选运算的时间复杂度为 $O(\log_2 n)$ ，故整个堆排序过程的时间复杂度为 $O(n \log_2 n)$ 。

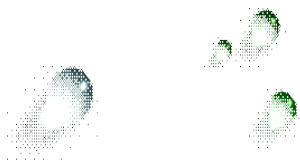
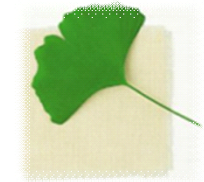
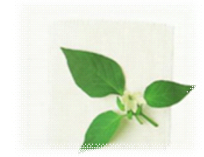
堆排序的时间复杂度不管序列初始状态如何皆为 $O(n \log n)$ ，但比较次数与初始状态有关。

堆排序占用的辅助空间为1（供交换元素用），故它的空间复杂度为 $O(1)$ 。

堆排序是一种不稳定的排序方法，例如，给定排序码：2，1，2'，它的排序结果为：1，2'，2。

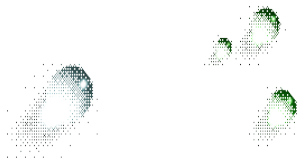
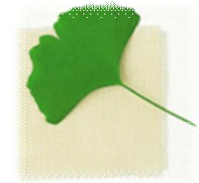


5 2-路归并排序



5 2-路归并排序

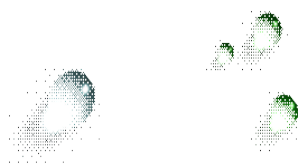
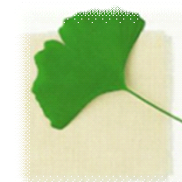
二路归并排序的基本思想:将两个有序表合并成一个有序表。



5 2-路归并排序

二路归并排序的基本思想:将两个有序表合并成一个有序表。

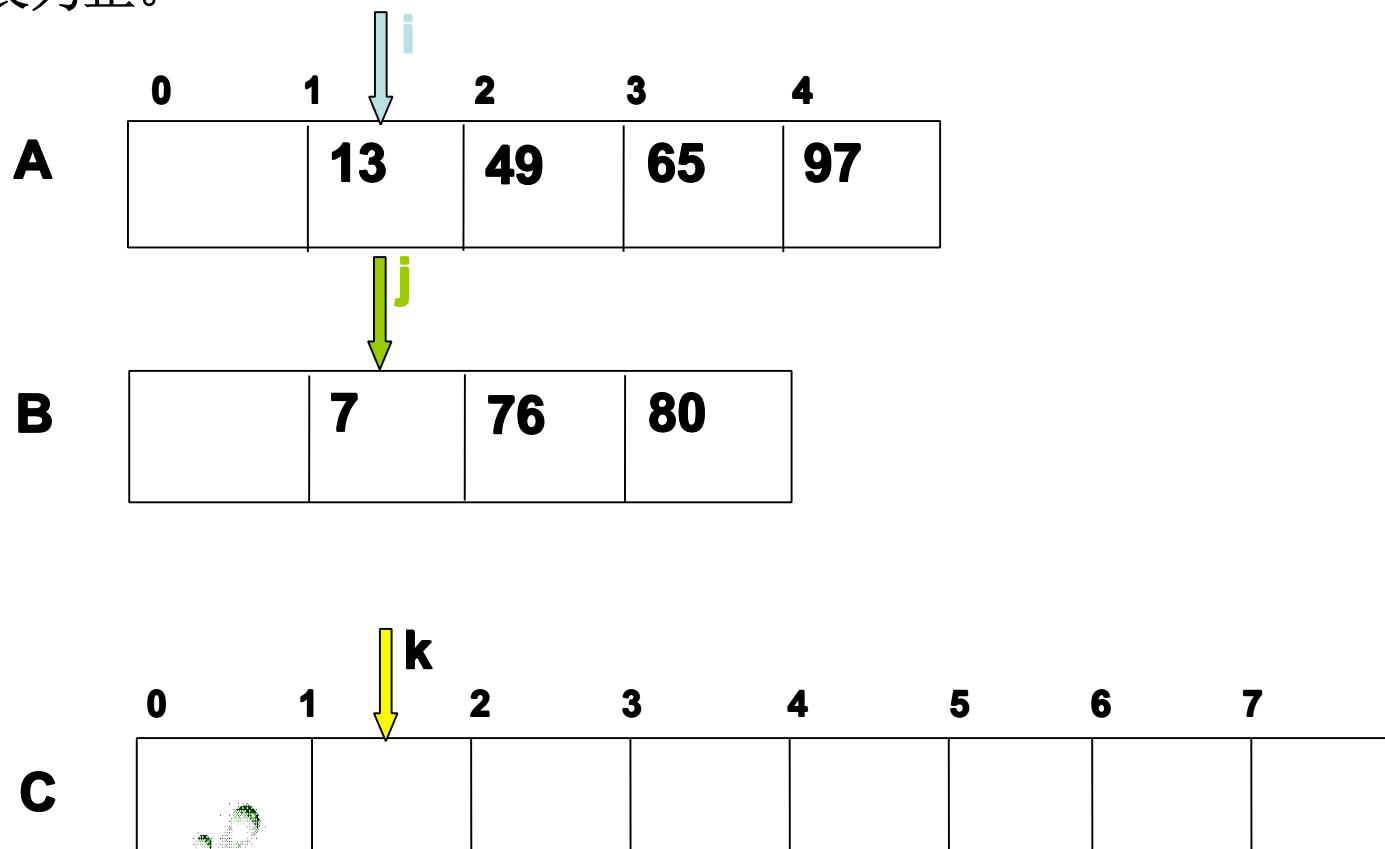
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

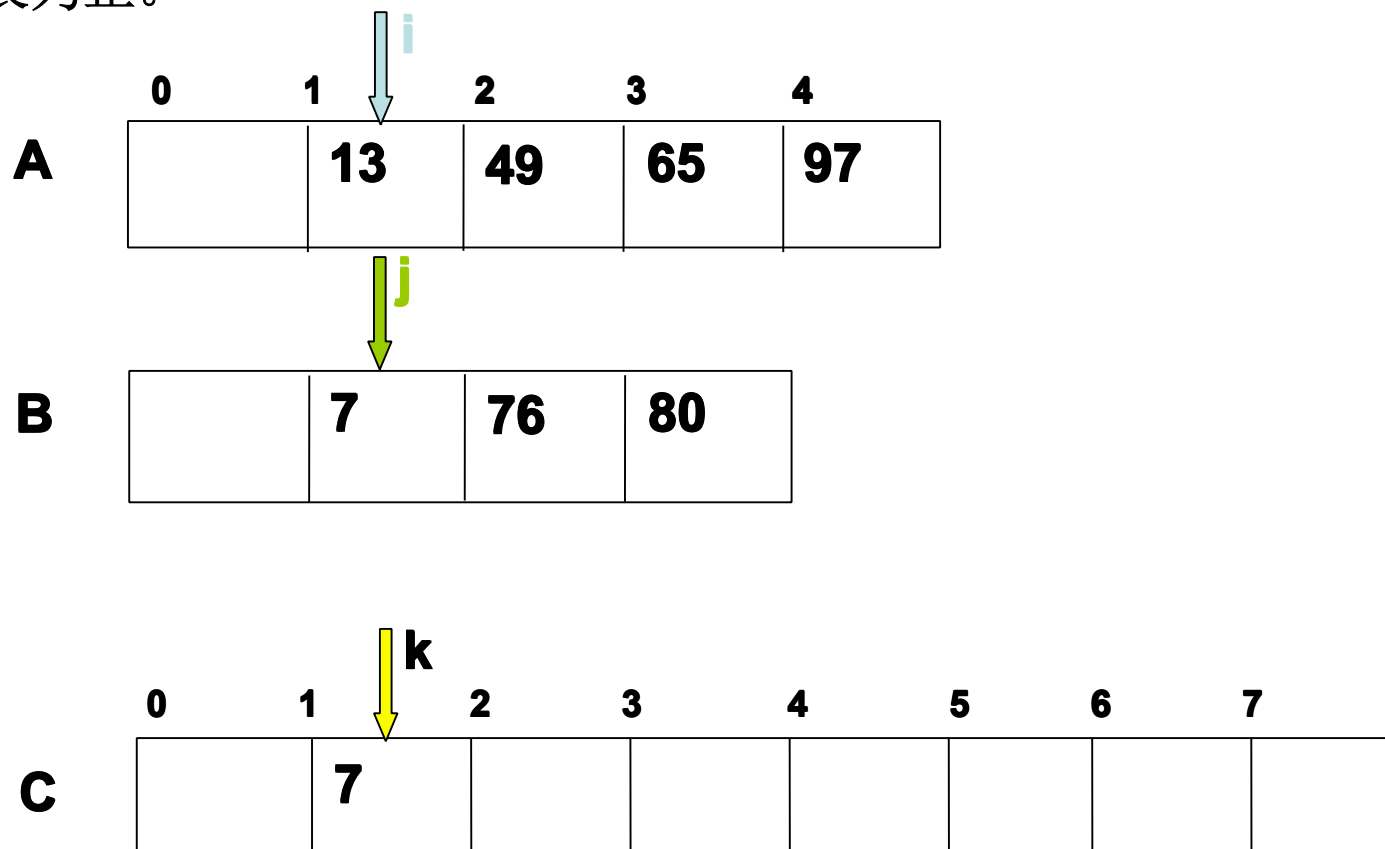
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

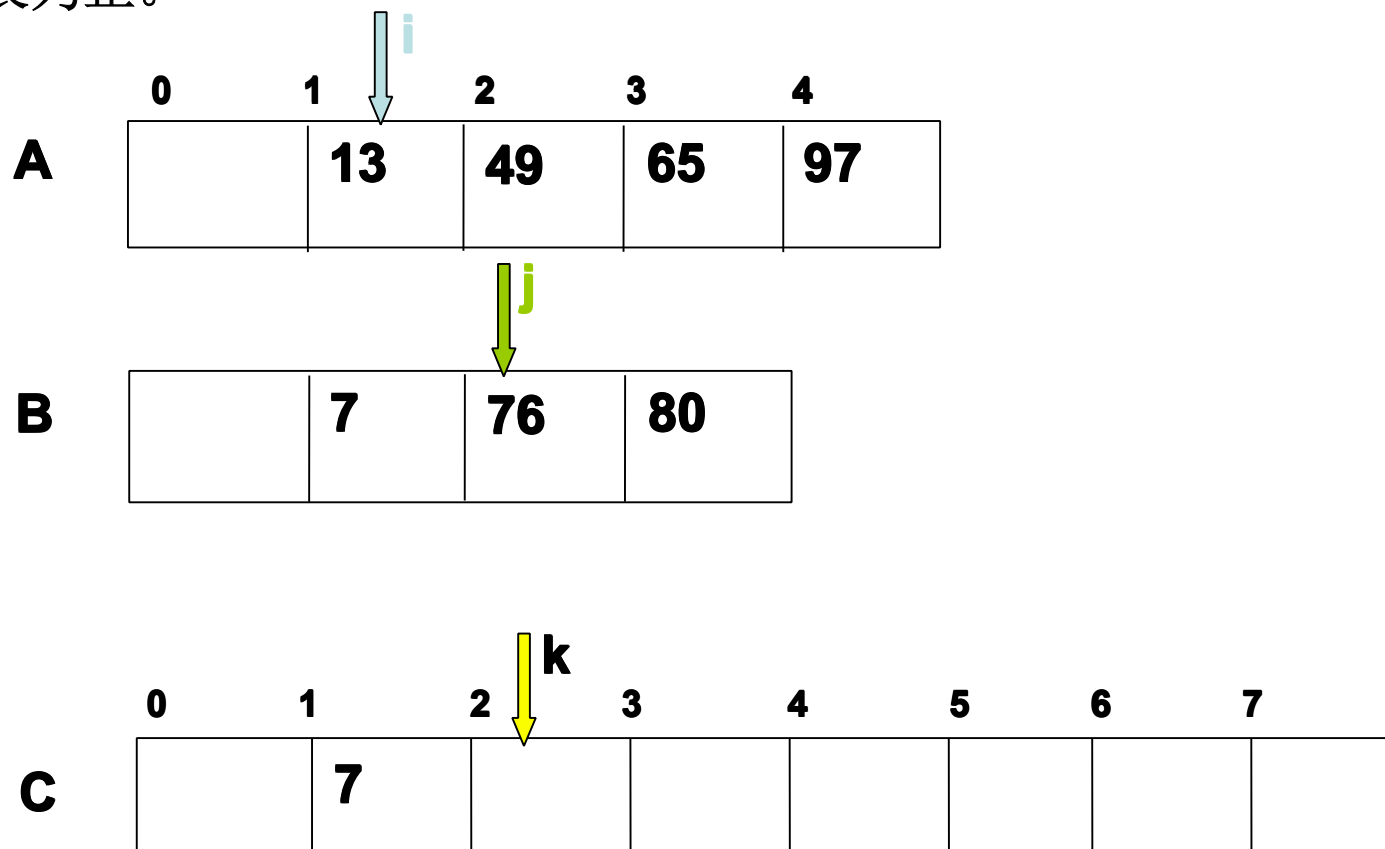
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想:将两个有序表合并成一个有序表。

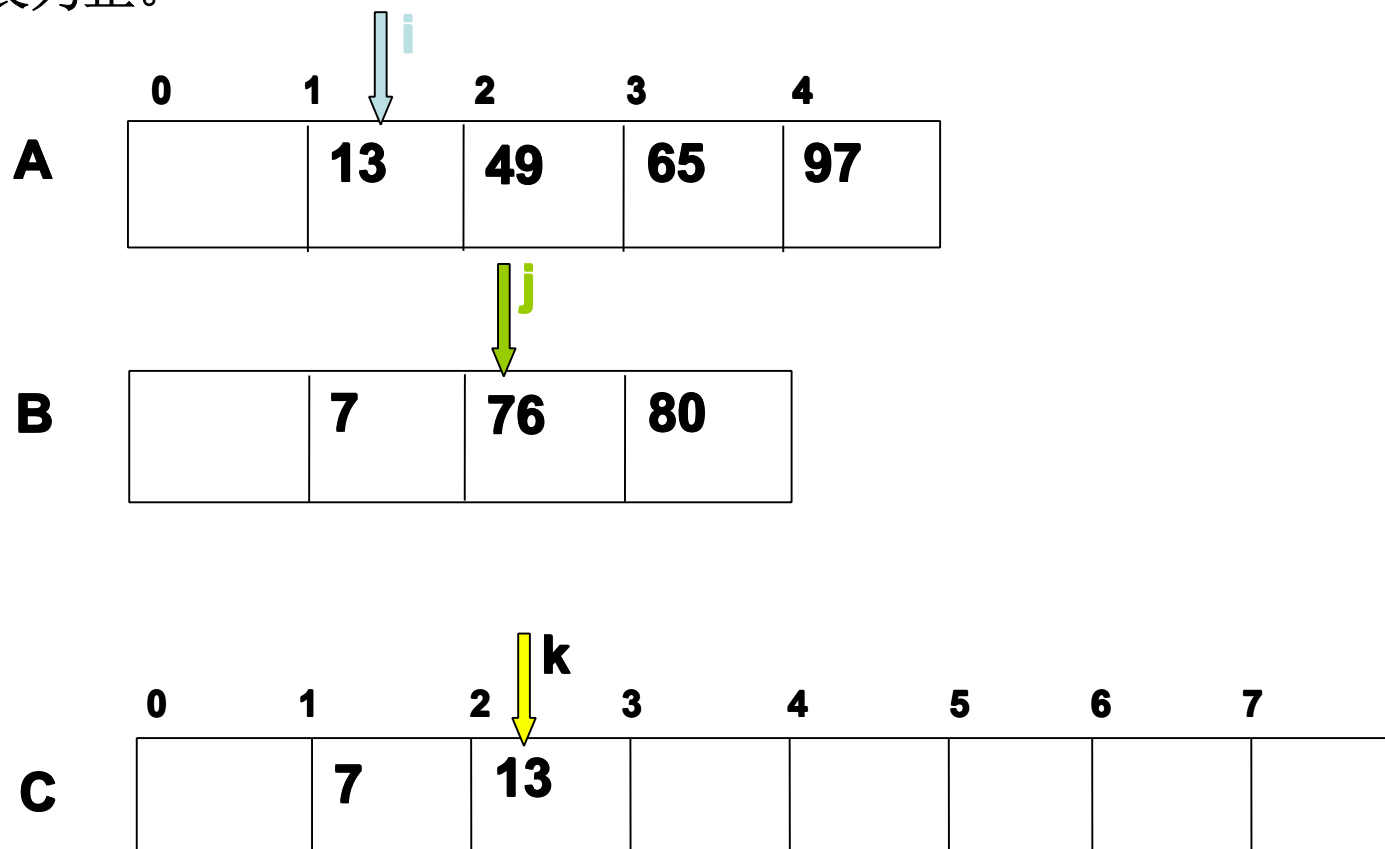
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

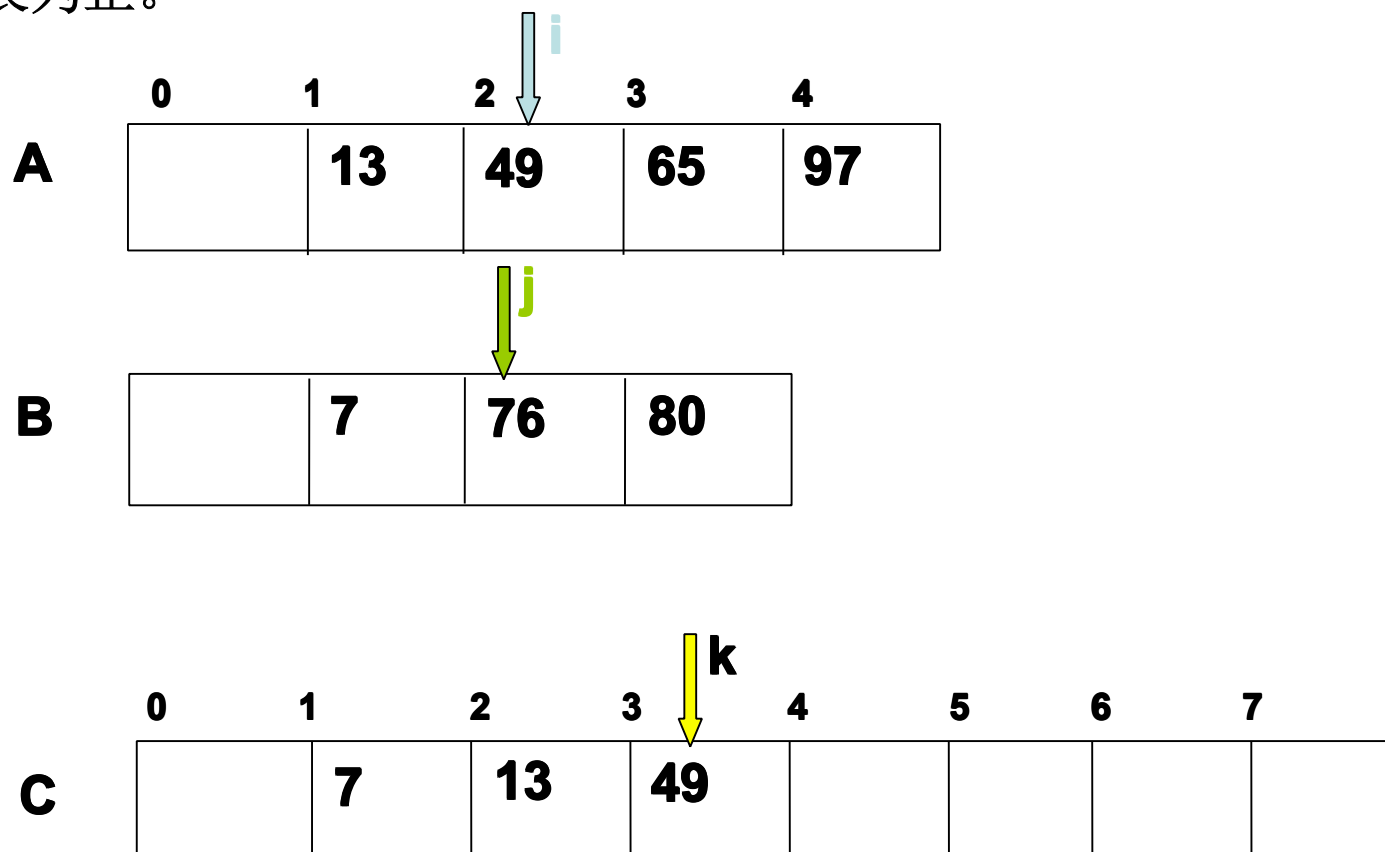
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

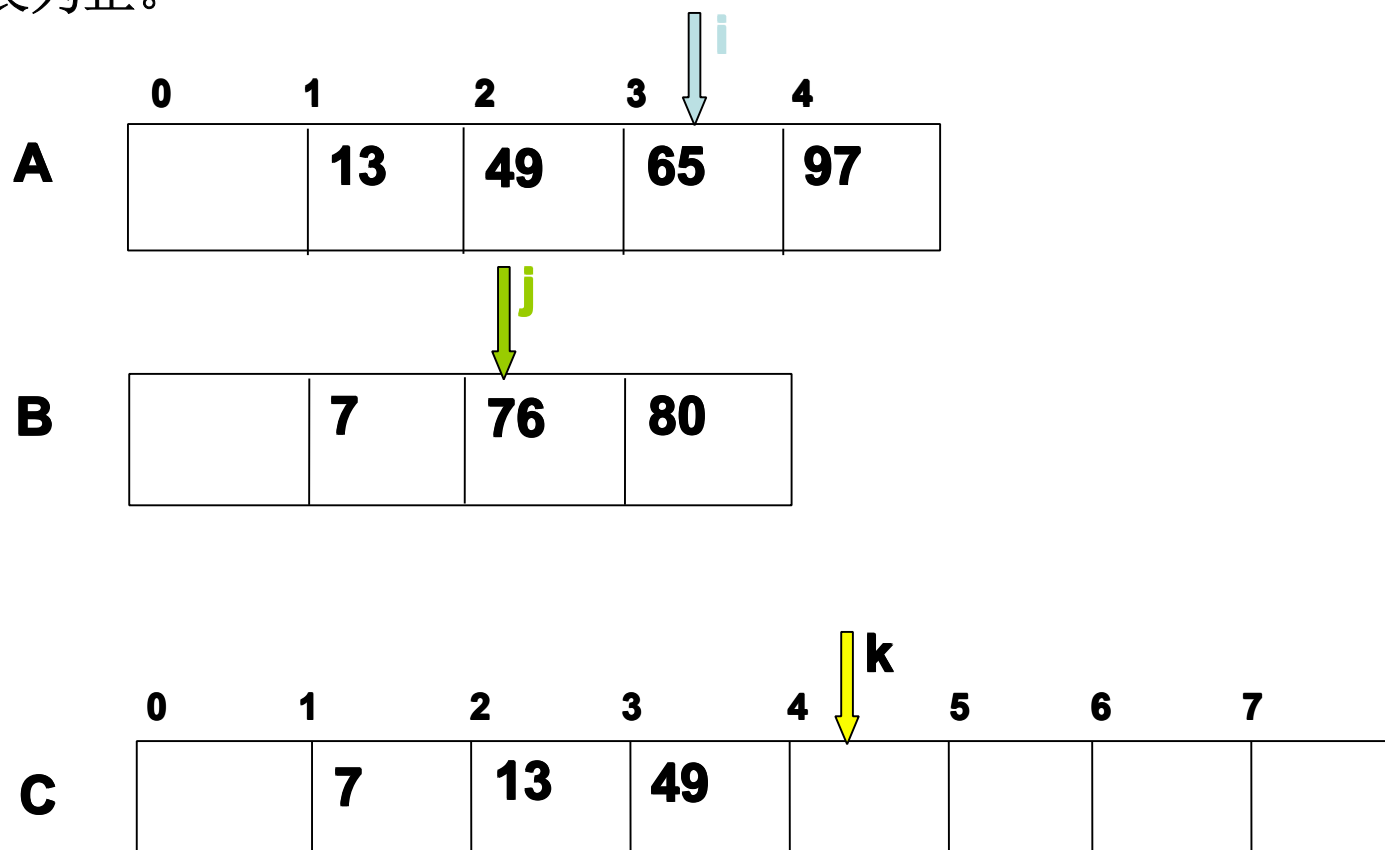
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

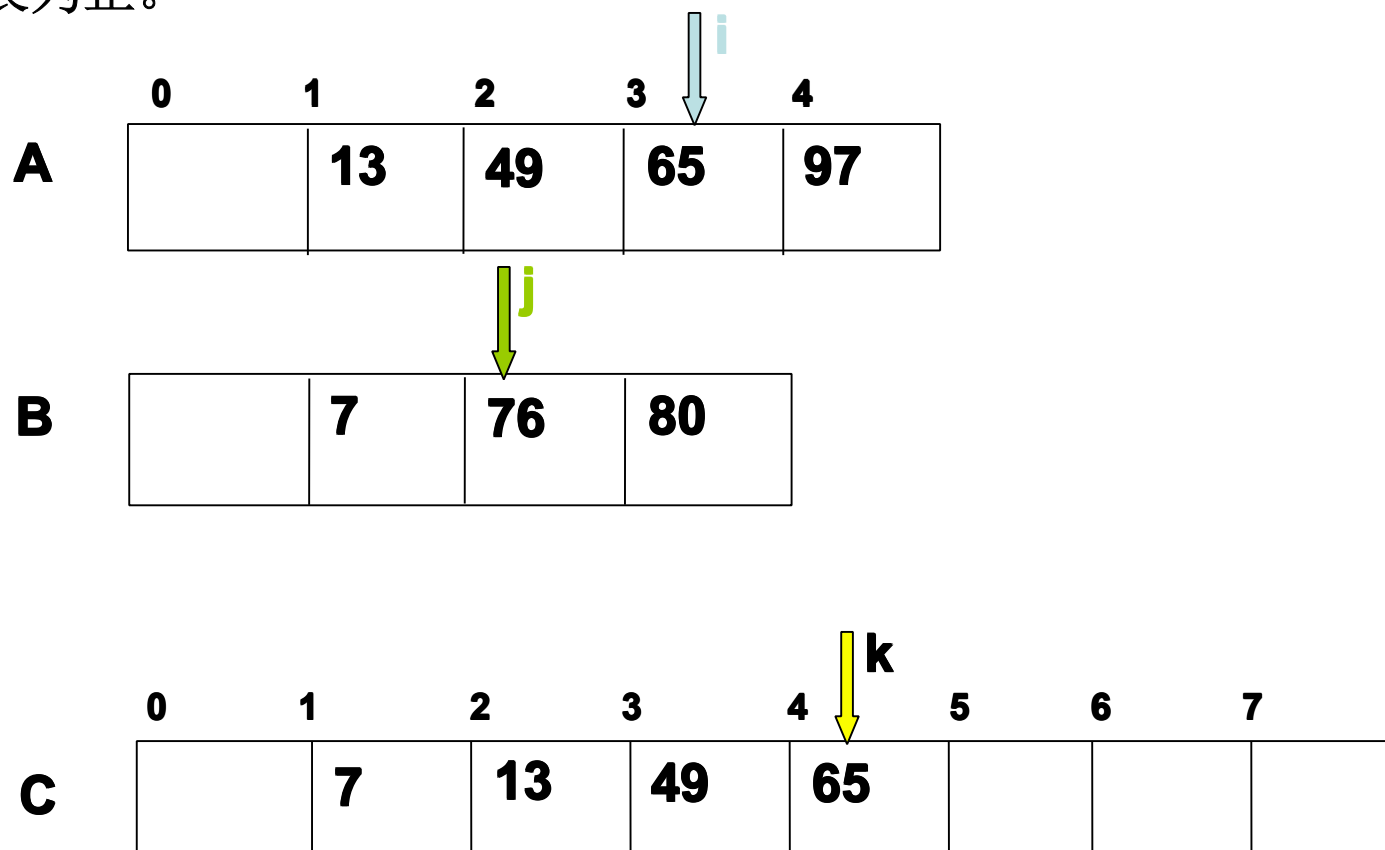
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

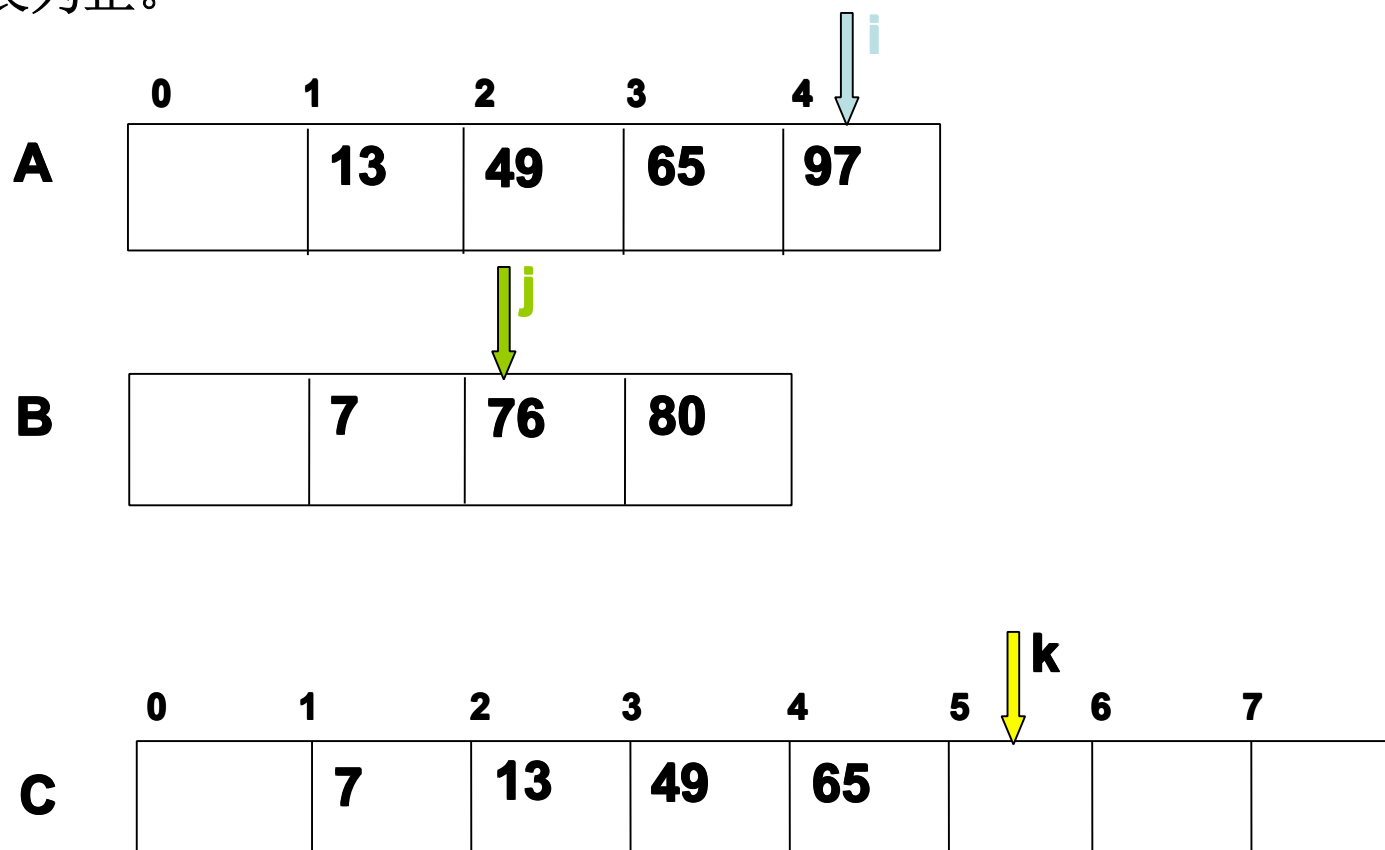
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

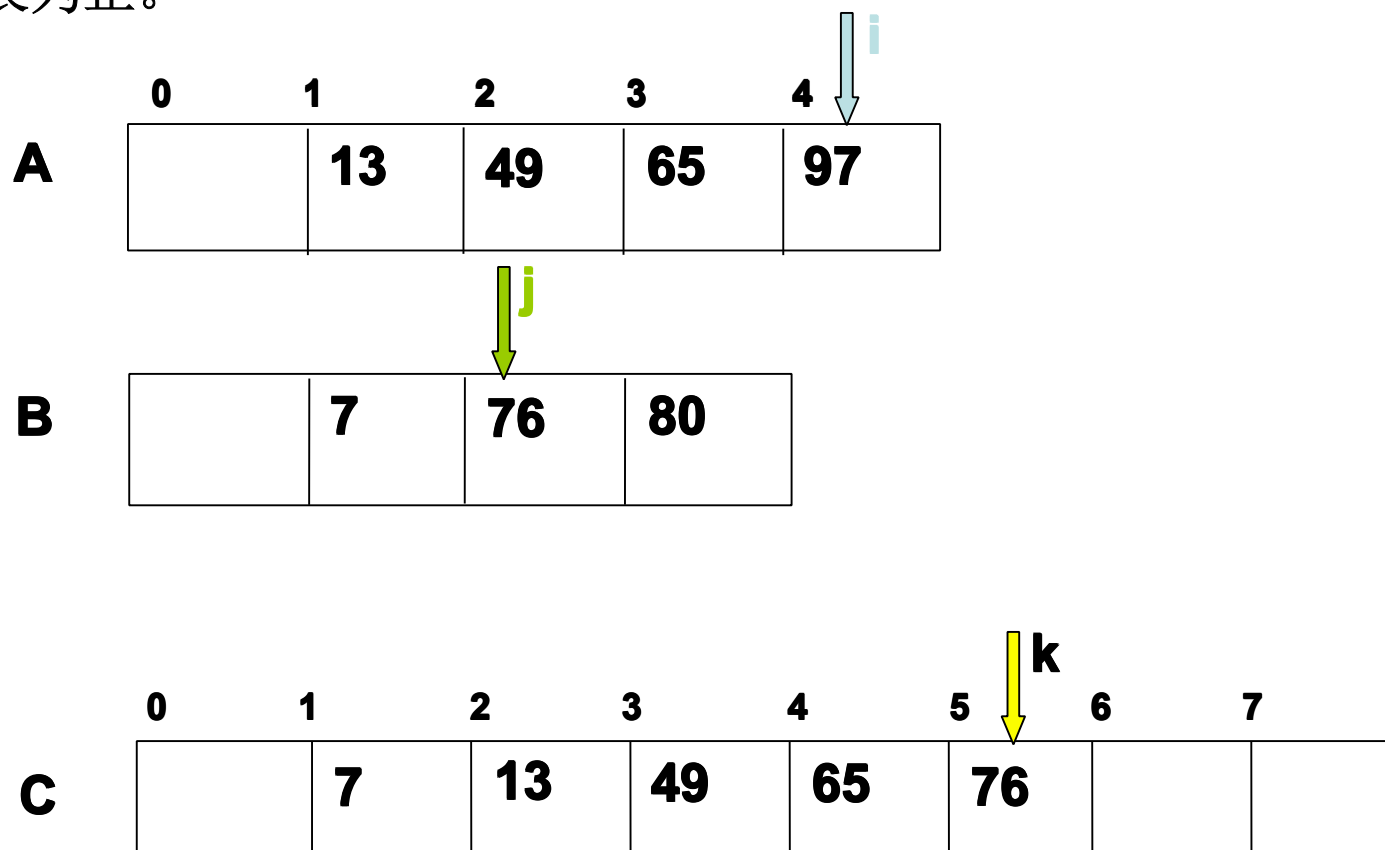
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想:将两个有序表合并成一个有序表。

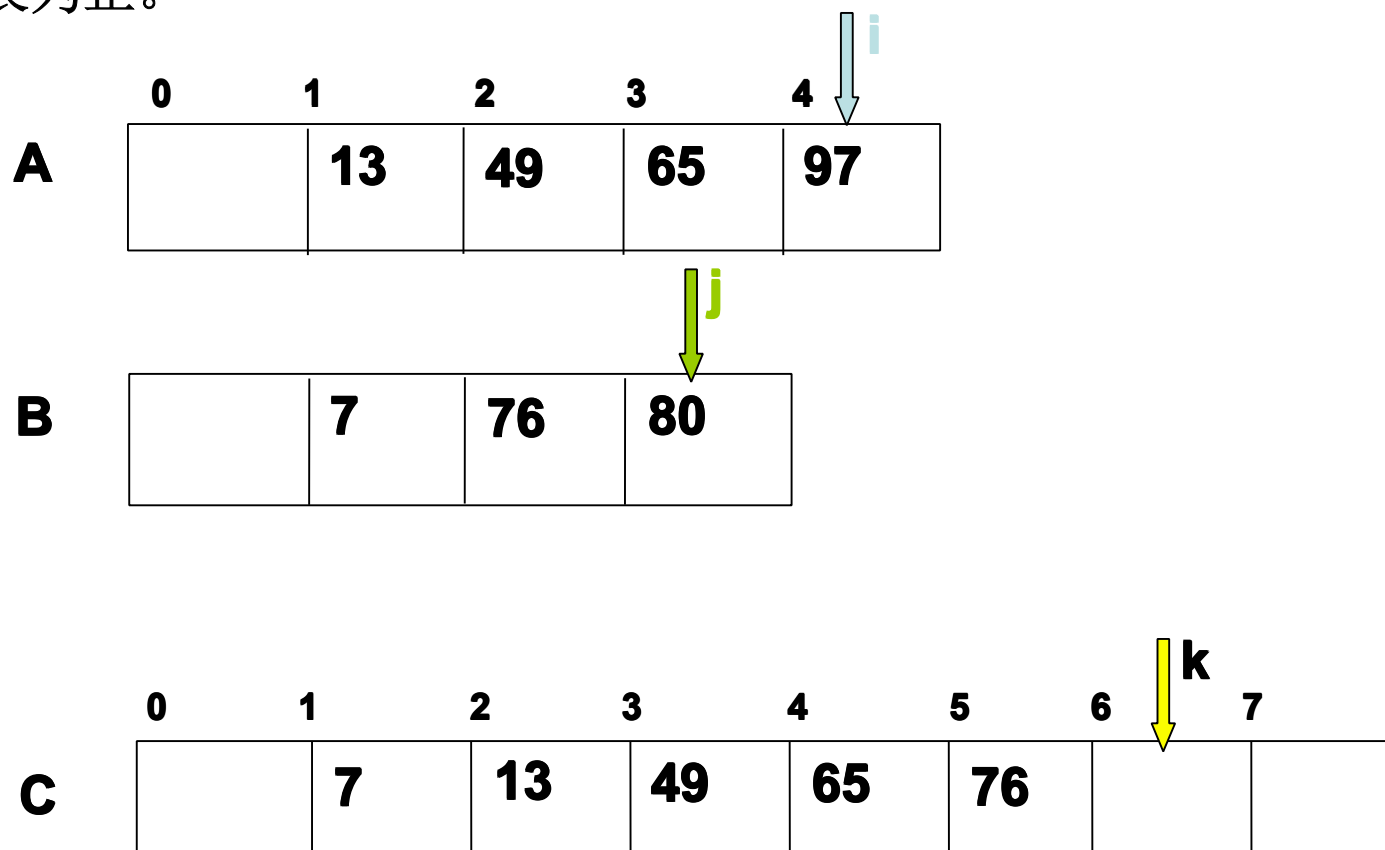
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想:将两个有序表合并成一个有序表。

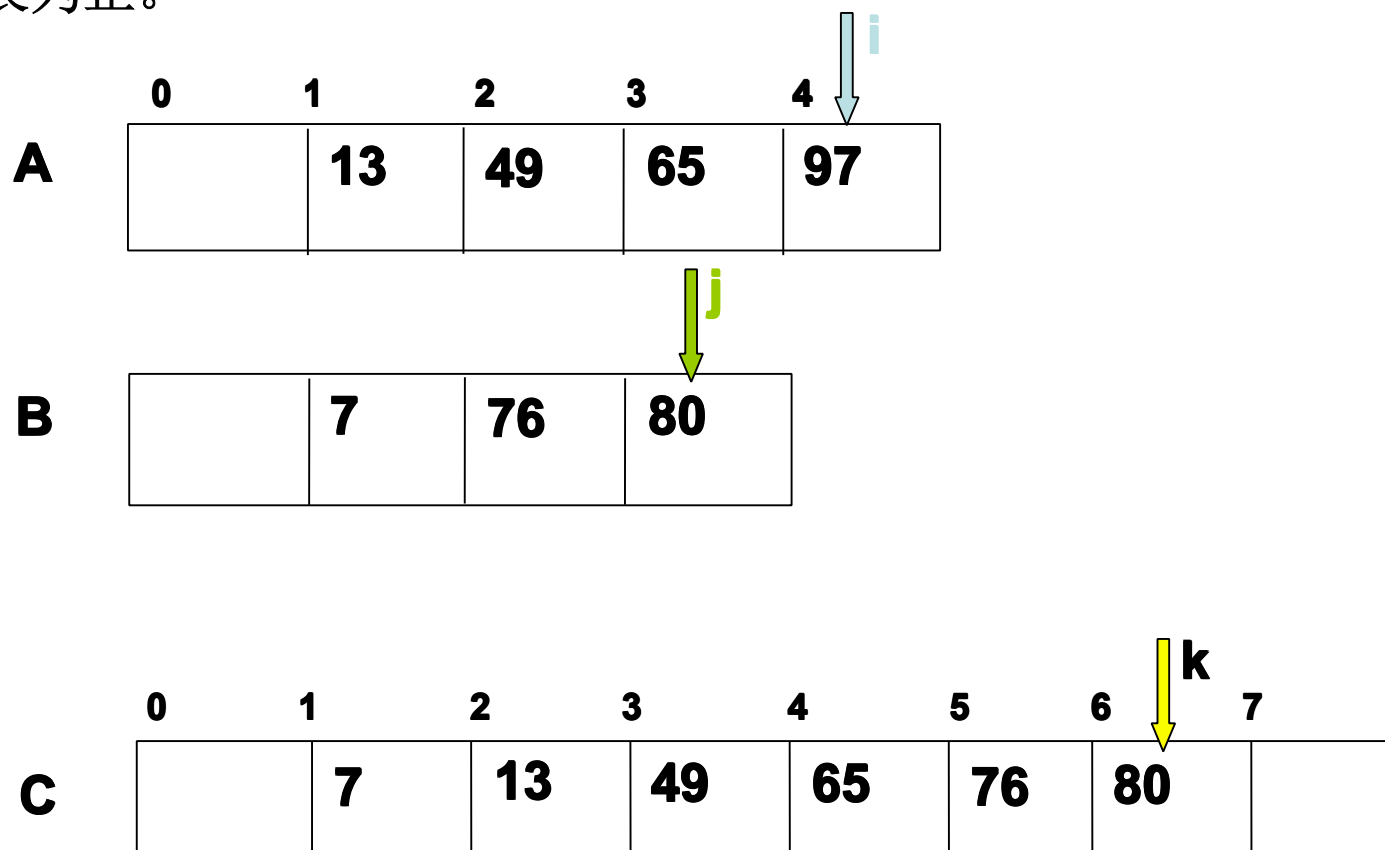
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

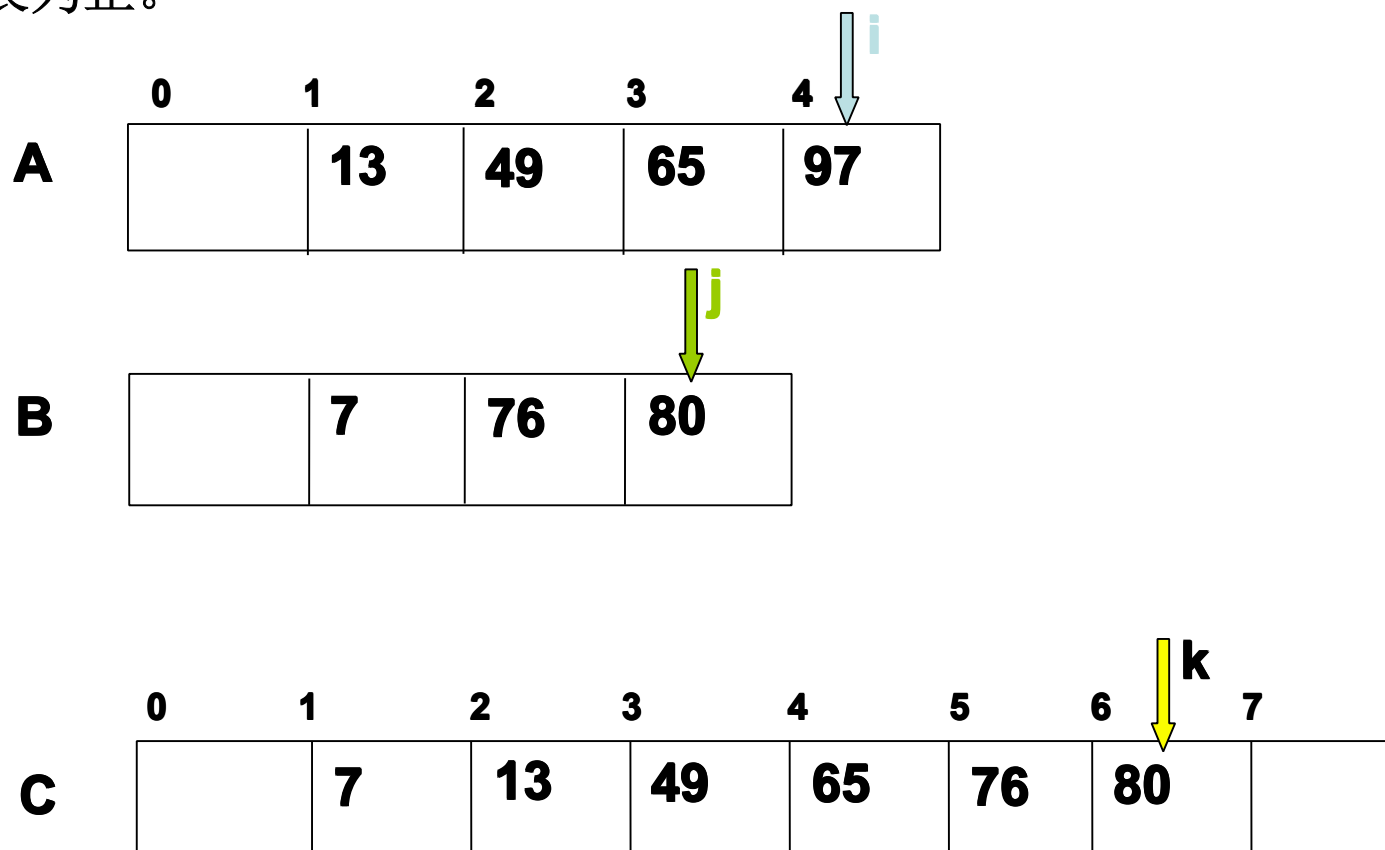
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想:将两个有序表合并成一个有序表。

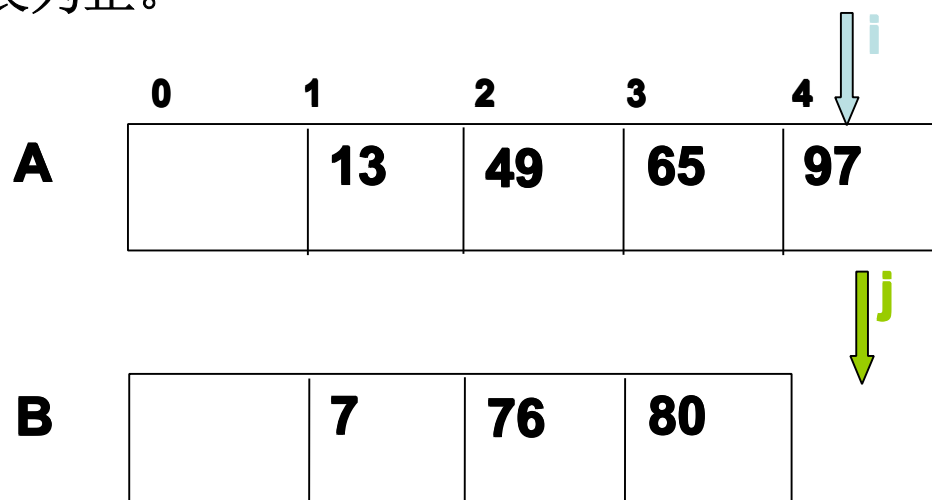
例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



5 2-路归并排序

二路归并排序的基本思想:将两个有序表合并成一个有序表。

例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。

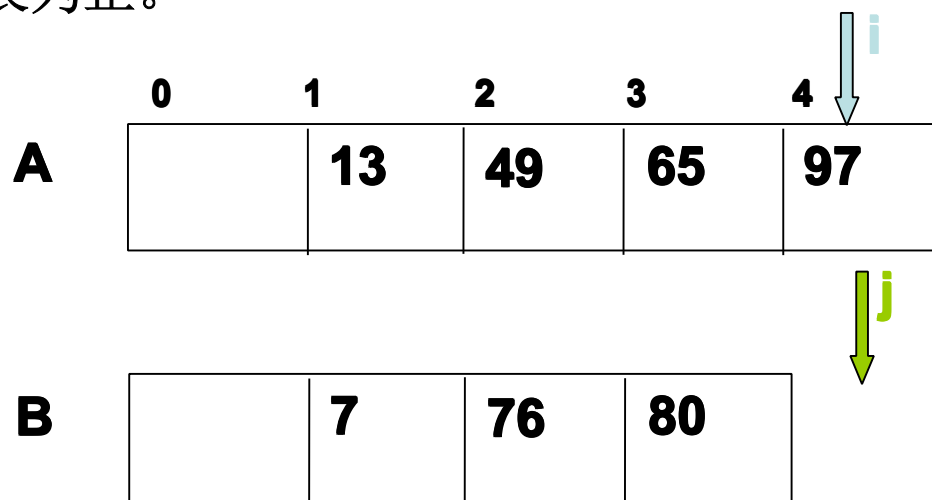


至此 **B** 表的元素都已移入 **C** 表, 只需将 **A** 表的剩余部分移入 **C** 表即可。

5 2-路归并排序

二路归并排序的基本思想: **将两个有序表合并成一个有序表。**

例如, 将下列两个已排序的顺序表合并成一个已排序表。顺序比较两者的相应元素, 小者移入另一表中, 反复如此, 直至其中任一表都移入另一表为止。



至此 **B** 表的元素都已移入 **C** 表, 只需将 **A** 表的剩余部分移入 **C** 表即可。



两个有序表的归并算法



2-路归并排序过程

二路归并排序的基本思想是将两个有序表合并成一个有序表。给定排序码**46, 55, 13, 42, 94, 05, 17, 70**，二路归并排序过程为：

初始状态: [46] [55] [13] [42] [94] [05] [17] [70]
└──┘ └──┘ └──┘ └──┘

一趟归并: [46 55] [13 42] [05 94] [17 70]
└────────┘ └────────┘

二趟归并: [13 42 46 55] [05 17 70 94]
└────────────────┘

三趟归并: [05 13 17 42 46 55 70 94]

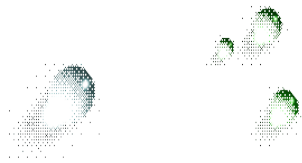
2-路归并的迭代算法

```
void MergeSort(datatype R[], int n)
/* 对R进行2-路归并结果仍在R中 */
{ int len=1;
  while (len<n)
  { MergePass (R, R1, len, n);
    /* 一趟归并结果在R1中 */
    for (i=1;i<=n;i++)
      R[i]=R1[i];
    /* 将R1[1..n] 复制到R[1..n]中,为
       下一步操作做准备*/
    len=2*len;
  }
}
```

一趟归并排序算法

```
void MergePass(datatype R[],
               datatype R1[], int len, int n)
/* len是本趟归并中有序表的长度, 从R[1..n]
   归并到R1[1..n]中 */
{for (i=1;i+2*len-1<=n;i+2*len)
  Merge (R, R1, i, i+len-1, i+2*len-1);
  /* 对两个长度为len的有序表进行合并 */
  if (i+len-1<n)
    Merge (R, R1, i, i+len-1, n);
    /* 对剩下的两个有序表（后一个有序表长
       度小于len）进行合并 */
  else
    while (i<=n) R1[i++]=R[i++];
    /*将剩余的最后一个有序表复制到R1中*/
}
```

2-路归并的递归算法



2-路归并排序的效率分析

2-路归并排序的时间复杂度等于归并趟数与每一趟时间复杂度的乘积。对 **n** 个元素的表，将这 **n** 个元素看作叶结点，若将两两归并生成的子表看作它们的父结点，则归并过程对应由叶向根生成一棵二叉树的过程。所以归并趟数等于二叉树的高度减**1**，即 $\lfloor \log_2 n \rfloor$ 。每一趟归并需移动 **n** 个元素，即每一趟归并的时间复杂度为 **$O(n)$** 。因此，**2-路归并排序的时间复杂度为 **$O(n \log_2 n)$**** 。

利用二路归并排序时，需要利用与待排序数组相同的辅助数组作临时单元，故该排序方法的**空间复杂度为 **$O(n)$**** ，比前面介绍的其它排序方法占用的空间大。

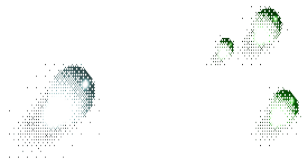
由于二路归并排序中，每两个有序表合并成一个有序表时，若分别在两个有序表中出现有相同排序码，则会使前一个有序表中相同排序码先复制，后一个有序表中相同排序码后复制，从而保持它们的相对次序不会改变。所以，**2-路归并排序是一种稳定的排序方法**。



*基数排序



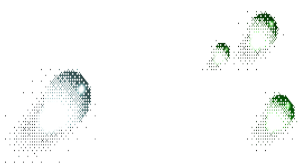
基数排序是和前面所述的各种排序方法完全不同的一种排序方法。前面介绍的几种排序方法，都是根据关键字之间的比较和移动记录来实现的，基数排序不需要进行记录关键字间的比较，而是根据组成关键字的各位值，即借助于多关键字排序的思想，用“分配”和“收集”的方法进行排序。





多关键字的排序



- 假设有 n 个记录的序列
 - $\{ R_1, R_2, \dots, R_n \}$
 - 每个记录 R_i 中含有 d 个关键字 $(K_i^0, K_i^1, \dots, K_i^{d-1})$, 则称上述记录序列对关键字 $(K_i^0, K_i^1, \dots, K_i^{d-1})$ 有序是指: 对于序列中任意两个记录 R_i 和 $R_j (1 \leq i < j \leq n)$ 都满足下列(词典)有序关系:
 - $(K_i^0, K_i^1, \dots, K_i^{d-1}) < (K_j^0, K_j^1, \dots, K_j^{d-1})$
 - 其中 K^0 被称为“最主”位关键字, K^{d-1} 被称为“最次”位关键字。
- 

实现多关键字排序通常有两种作法:

- **最高位优先MSD法**: 先对 K_0 进行排序, 并按 K_0 的不同值将记录序列分成若干子序列之后, 分别对 K_1 进行排序, ..., 依次类推, 直至最后对最低位关键字排序完成为止。
- **最低位优先LSD法**: 先对 K_{d-1} 进行排序, 然后对 K_{d-2} 进行排序, 依次类推, 直至对最主位关键字 K_0 排序完成为止。排序过程中不需要根据“前一个”关键字的排序结果, 将记录序列分割成若干个(“前一个”关键字不同的)子序列。

例如：学生记录含三个关键字：系别、班号和班内的序列号，其中以系别为最主位关键字。

LSD的排序过程如下：

无序序列 3,2,30 1,2,15 3,1,20 2,3,18 2,1,20

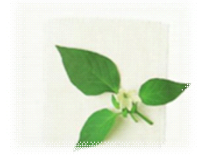
对K2排序 1,2,15 2,3,18 3,1,20 2,1,20 3,2,30

对K1排序 3,1,20 2,1,20 1,2,15 3,2,30 2,3,18

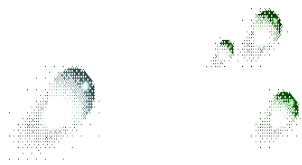
对K0排序 1,2,15 2,1,20 2,3,18 3,1,20 3,2,30



MSD法和LSD法的比较



比较MSD法和LSD法，一般来讲，LSD法要比MSD法来得简单，因为LSD法是从头到尾进行若干次分配和收集，执行的次数取决于构成关键字值的成分为多少；而MSD法则要处理各序列与子序列的独立排序问题，就可能复杂一些。



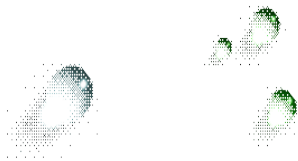


链式基数排序

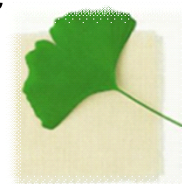
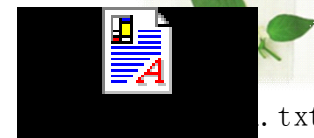


- 假如多关键字的记录序列中，每个关键字的取值范围相同，则按LSD法进行排序时，可以采用“分配-收集”的方法，其好处是不需要进行关键字间的比较。

对于数字型或字符型的单关键字，可以看成是由多个数位或多个字符构成的多关键字，此时可以采用这种“分配-收集”的办法进行排序，称作基数排序法。

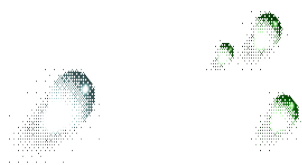


链式基数排序



在计算机上实现基数排序时，应采用链表作存储结构，即链式基数排序，具体作法为：

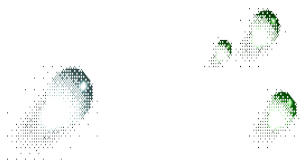
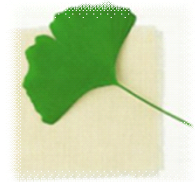
- 待排序记录以指针相链，构成一个链表；
- “分配”时，按当前“关键字位”所取值，将记录分配到不同的“链队列”中，每个队列中记录的“关键字位”相同；
- “收集”时，按当前关键字位取值从小到大将各队列首尾相链成一个链表；
- 对每个关键字位均重复2)和3)两步。



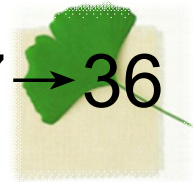


链式基数排序

- 例如:
- $p \rightarrow 369 \rightarrow 367 \rightarrow 167 \rightarrow 239 \rightarrow 237 \rightarrow 138 \rightarrow 230 \rightarrow 139$
- 第一次分配得到
- $f[0] \rightarrow 230 \leftarrow r[0]$
- $f[7] \rightarrow 367 \rightarrow 167 \rightarrow 237 \leftarrow r[7]$
- $f[8] \rightarrow 138 \leftarrow r[8]$
- $f[9] \rightarrow 369 \rightarrow 239 \rightarrow 139 \leftarrow r[9]$
- 第一次收集得到
- $p \rightarrow 230 \rightarrow 367 \rightarrow 167 \rightarrow 237 \rightarrow 138 \rightarrow 368 \rightarrow 239 \rightarrow 139$




- 第二次分配得到
- $f[3] \rightarrow 230 \rightarrow 237 \rightarrow 138 \rightarrow 239 \rightarrow 139 \leftarrow r[3]$
- $f[6] \rightarrow 367 \rightarrow 167 \rightarrow 368 \leftarrow r[6]$
- 第二次收集得到
- $p \rightarrow 230 \rightarrow 237 \rightarrow 138 \rightarrow 239 \rightarrow 139 \rightarrow 367 \rightarrow 167 \rightarrow 368$
- 第三次分配得到
- $f[1] \rightarrow 138 \rightarrow 139 \rightarrow 167 \leftarrow r[1]$
- $f[2] \rightarrow 230 \rightarrow 237 \rightarrow 239 \leftarrow r[2]$
- $f[3] \rightarrow 367 \rightarrow 368 \leftarrow r[3]$
- 第三次收集之后便得到记录的有序序列
- $p \rightarrow 138 \rightarrow 139 \rightarrow 167 \rightarrow 230 \rightarrow 237 \rightarrow 239 \rightarrow 367 \rightarrow 368$





链式基数排序分析



链式基数排序算法对数据进行 d 趟扫描，每趟需时间 $O(n+j)$ 。因此总的计算时间为 $O(d(n+j))$ 。对于不同的基数 j 所用的时间是不同的。当 n 较大或 d 较小时，这种方法较为节省时间。

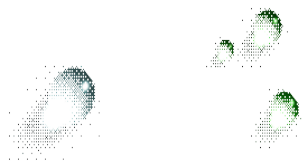
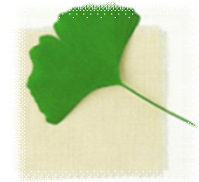
基数排序适用于链式存储结构的记录的排序，它要求的附加存储量是 j 个队列的头、尾指针。所以，需要 $O(n+j)$ 辅助空间。

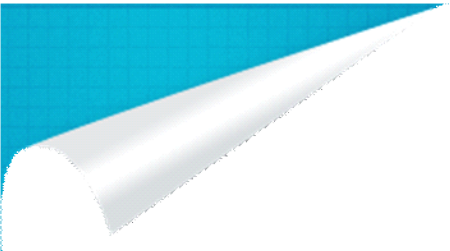
基数排序是一种稳定的排序方法。



7 外排序简介

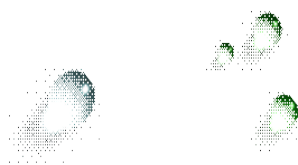
如果待排序的记录数很大，无法将所有记录都装入内存，只能将它们存放在外存上，我们称这时的排序为外排序。





外存信息的存取——磁带

磁带是涂上一层磁性材料的窄带，磁带卷在带盘上，带盘安装在磁带驱动器的转轴上。驱动器控制磁带盘转动，带动磁带移动，通过读/写磁头进行读/写信息的操作。

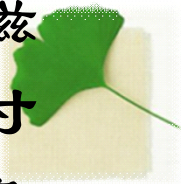





外存信息的存取——磁盘



磁盘存取信息时，首先要确定信息所在的柱面，再将磁头移动到所需磁道的位置上，移动磁头所需的时间称为磁头定位时间或称为寻道时间。然后等待磁道上的信息所在位置随着磁盘的转动而转到磁头下面，这段时间称为等待时间。由于磁盘高速运转（2400~3600转/分），所以，等待时间是极短的。磁盘的存取时间主要花在磁头定位时间上。

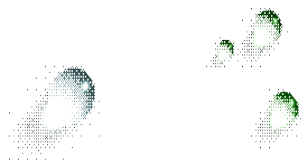




外部排序的基本方法



外排序的基本方法是归并排序。由两个阶段组成：第一阶段，产生顺串。即把要排序的元素分成若干组，把每一组装入内存，进行内排序，每一组经过内排序后再写到外存上，我们把经过内排序的每一组叫一个顺串，这样在外存上就产生了许多顺串。第二阶段，归并。可用两路或多路进行归并（可用三台磁带机或磁盘机实现两路归并、也可用四台磁带机或磁盘机实现三路归并等），使顺串的长度逐渐由小至大，直至变成一个顺串（即使所有元素按关键码有序）为止。







外部排序的基本方法



一般可依据所使用的外存设备将外部排序分为磁盘文件排序和磁带文件排序。磁盘排序和磁带排序基本相似，区别在于初始归并段在外存储介质中的分布方式不同。磁盘是直接存取设备，而磁带是顺序存储设备，读取信息块的时间与所读信息块的位置关系极大。故在磁带上进行文件排序时，研究归并段信息块的分布是个极为重要的问题。







外部排序的基本方法



最简单的归并排序方法与内排序中的二路归并类似。假设一具有 n 个记录的文件，先把该文件看作是由 n 个长度为1的顺串组成，然后在此基础上进行两两归并。经过 $\log_2 n$ 趟归并后，当文件中只含有一个长度为 n 的顺串时，整个文件的排序就完成了。在每一趟排序过程中都需要进行记录的内、外存交换。

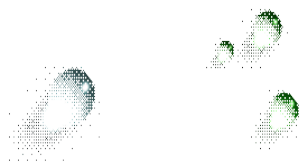




外部排序的基本方法



还有一种常用的外部排序方法是多路归并排序。由于在外部排序过程中，数据的内外存交换所需的时间比记录的内部归并所需的时间大得多，所以可以通过减少数据内外存交换的次数来提高外部排序的效率。为了不增加内部归并时所需进行关键字比较的次数，在具体实现时通常不用选择排序的方法，而用“败者树”来实现。





排序小结



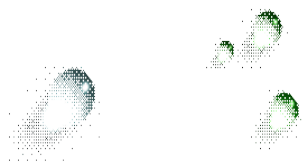
本章重点掌握内排序。不仅要掌握各种内



排序的基本思想（过程），而且要掌握各种内排序方法的时间复杂度、辅助存储空间及稳定性，尤其要会根据具体情况和要求选择相应的内排序方法。



各种内部排序方法的比较和选择如下：



各种内部排序方法的比较

	平均时间性能	最好时间性能	最坏时间性能	辅助空间	稳定性
直接插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	√
希尔排序	$O(n^{1.3})$	$O(n \log_2 n)$	$O(n^2)$	$O(1)$	×
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	√
快速排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n^2)$	$O(\log_2 n)$	×
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	×
堆排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$	×
归并排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n)$	√
基数排序	$O(d(n+rd))$	$O(d(n+rd))$	$O(d(n+rd))$	$O(rd)$	√



各种内部排序方法性能比较



- 1) 从平均时间而言：快速排序最佳。但在最坏情况下时间性能不如堆排序和归并排序。
- 2) 从算法简单性看：由于直接选择排序、直接插入排序和冒泡排序的算法比较简单，将其认为是简单算法，都包含在上图中的“简单排序”中。对于希尔排序、堆排序、快速排序和归并排序算法，其算法比较复杂，认为是复杂排序。
- 3) 从稳定性看：直接插入排序、冒泡排序和归并排序是稳定的；而希尔排序、直接选择排序、快速排序和堆排序是不稳定排序。
- 4) 从待排序的记录数 n 的大小看， n 较小时，宜采用简单排序；而 n 较大时宜采用改进排序。



选择排序的方法



- (1) 当待排序记录数 n 较大时，若要求排序稳定，则采用归并排序。
- (2) 当待排序记录数 n 较大，关键字分布随机，而且不要求稳定时，可采用快速排序；
- (3) 当待排序记录数 n 较大，关键字会出现正、逆序情形，可采用堆排序（或归并排序）。
- (4) 当待排序记录数 n 较小，记录已接近有序或随机分布时，又要求排序稳定，可采用直接插入排序。
- (5) 当待排序记录数 n 较小，且对稳定性不作要求时，可采用直接选择排序。