第 8 章习题

8.1 直接插入排序算法分别在什么情况下可以达到最好和最坏的情况？分别要比较和移动多少次？相应的时间复杂度分别是多少？

【解】假设待排序列有 n 个元素。

最好情况：原始序列为有序序列，目标序列与原始序列顺序相同，比如原序列为不降序列，进行增排序。

比较 n-1 次；

每次比较 2 次移动元素，总移动次数为 2（n-1）次；

时间复杂度：0（n）。

最坏情况：原始序列为有序序列，目标序列与原始序列顺序相反，比如原序列为不降序列，进行降排序。

循环控制变量为 i 时，有序区有 i-1 个元素。则与有序区元素比较次数为 i-1，加上和监视哨 1 次，共比较 i 次；有序区元素移动 i-1 次，加上 i 元素存入临时变量和从临时变量插入目标位置，共移动 i-1+2=i+1 次，

比较 $\sum_{i=2}^{n} i$ =(n+2)(n-1)/2 次；

移动 $\sum_{i=2}^{n}(i+1)$ =(n+4)(n-1)/2 次；

时间复杂度：$O(n^2)$。

8.2 直接插入排序能否保证在每一趟都能至少将一个元素放在其最终的位置上？是否是稳定的排序算法？

【解】

不能保证每趟至少一个元素进入最终位置。

稳定排序。

8.3 对下面数据表，写出采用希尔排序算法排序的每趟的结果。

(78 100 120 25 85 40 90 15 60 35 105 50 30 10 28 12)

【解】

d=8

60 35 105 25 30 10 28 12 78 100 120 50 85 40 90 15

d=4

30 10 28 12 60 35 90 15 78 40 105 25 85 100 120 50

d=2

28 10 30 12 60 15 78 25 85 35 90 40 105 50 120 100

d=1

10 12 15 25 28 30 35 40 50 60 78 85 90 100 105 120

8.4 对下面数据表，写出采用冒泡排序算法排序的每趟的结果，并标明数据移动情况。

(105 50 30 25 85 40 100 12 10 28)

【解】

10 105 50 30 25 85 40 100 12 28

10 12 105 50 30 25 85 40 100 28

10 12 25 105 50 30 28 85 40 100

10 12 25 28 105 50 30 40 85 100

10 12 25 28 30 105 50 40 85 100

10 12 25 28 30 40 105 50 85 100

10 12 25 28 30 40 50 105 85 100

10 12 25 28 30 40 50 85 105 100

10 12 25 28 30 40 50 85 100 105

8.5 对下面数据表，写出采用快速排序算法排序的每趟的结果，并标明每趟的数据移动情况。

(50 30 120 25 85 40 100 12 90 15 60 35 105 78 10 28)

【解】

x=50

(28 30 120 25 85 40 100 12 90 15 60 35 105 78 10)

(28 30 25 85 40 100 12 90 15 60 35 105 78 10 120)

(28 30 10 25 85 40 100 12 90 15 60 35 105 78 120)

(28 30 10 25 40 100 12 90 15 60 35 105 78 85 120)

(28 30 10 25 35 40 100 12 90 15 60 105 78 85 120)

(28 30 10 25 35 40 12 90 15 60 100 105 78 85 120)

(28 30 10 25 35 40 15 12 90 60 100 105 78 85 120)

(28 30 10 25 35 40 15 12 90 60 100 105 78 85 120)

(28 30 10 25 35 40 15 12) 50 (90 60 100 105 78 85 120)

x=28，x=90

(12 30 10 25 35 40 15) 50 (85 60 100 105 78 120)

(12 10 25 35 40 15 30) 50 (85 60 105 78 100 120)

(12 15 10 25 35 40 30) 50 (85 60 78 105 100 120)

(12 15 10 25 40 35 30) 50 (85 60 78 100 105 120)

(12 15 10 25) 28 (40 35 30) 50 (85 60 78) 90 (100 105 120)

x=12，40，85，100

(10 15 25) 28 (30 35) 50 (78 60) 90 (105 120)

(10 15 25) 28 (30 35 40) 50 (78 60 85) 90 (100 105 120)

(10 12 15 25) 28 (30 35 40) 50 (78 60 85) 90 (100 105 120)
(10) 12 (15 25) 28 (30 35) 40 50 (78 60) 85 90 100 (105 120)
x=10, 15, 30, 78, 105
10 12 (15 25) 28 (30 35) 40 50 (60) 85 90 100 (105 120)
10 12 15 (25) 28 30 (35) 40 50 (60 78) 85 90 100 105 (120)
x=25, 35, 60, 120
10 12 15 25 28 30 35 40 50 (60 78) 85 90 100 105 120
10 12 15 25 28 30 35 40 50 60 (78) 85 90 100 105 120
x=78
10 12 15 25 28 30 35 40 50 60 78 85 90 100 105 120

8.6 已知数组 A[n]中的元素为整型，设计算法将其中所有的奇数调整到数组的左边，而将所有的偶数调整到数组的右边，并要求时间复杂度为 O(n)。

【解】与快速排序算法的分区算法类似，

先从左端寻找第一个偶数，存入临时变量 A[0]；然后从右往左找到一个奇数，存入 A[low]，再从左端找一个偶数放入右端 A[high]，这两个操作交替执行，直到 low==high；再把临时变量中存放的数据存入 A[low]，划分完成，算法描述如下：

```
void OddEvenPartition( elementType A[])
{
    int low,high;  //第一和最后一个元素下标
    low=1;
    high=listLen;

        //左端找到第一个偶数，转存 A[0]
    while(low<high && A[low].key%2==1)   //奇数，low 后移
        low++;
       //左端出现第一个偶数，存入 A[0]
    A[0]=A[low];

    while( low<high )
    {      //从右端找一个奇数，存入左端
        while(low<high && A[high].key%2==0)  //偶数，high 前移
            high--;
        //右端出现一个奇数，存入左端
        A[low]=A[high];

           //左端找一个偶数，存到右端
        while(low<high && A[low].key%2==1)  //奇数，low 右移
            low++;
        //左端出现一个偶数，存入右端
        A[high]=A[low];
```

```
    }
        //此时，low==high，是A[0]元素的存放位置
    A[low]=A[0];
}
```

8.7 已知数组A[n]中的元素为整型，设计算法将其调整为三部分，其中左边所有元素为3的倍数，中间所有元素除3余1，右边所有元素除3余2，并要求时间复杂度为O(n)。

【解】借用快速排序分区思想，分两步完成，第一步先将余数为0的元素与其他情况进行二分划分，即余数为0元素存入左边子表，余数为1和2的元素存在表的右端；第二部再将右端子表按余数1和2进行二分划分即可。

第一步：

从左端找到第一个余数不为0（余数为1或2）的元素，存入临时单元A[]；

以下步骤循环执行，直到low==high：

从高端找到一个余数为0的元素，存入A[low]；

从左端找到一个余数不为0的元素，存入A[high]；

循环结束后，low==high，在此位置存入A[0]元素，即：

A[low]=A[0]，或A[high]

上面处理结束后，余数为0的元素存入表的左端，右端为余数为1或2的元素，且A[low]为右端子表的第一个元素。

接下来再将右子表A[low...n]，按余数为1和2进行二分划分，此时，low数值不变，high重新初始化为n。算法描述如下：

```
void Remainder3Partition(elementType A[])
{
    int low,high;
    low=1;
    high=listLen;

        //第一步：按余数为0，余数为1和2进行二分划分
        //左端找出第一个余数不为0的元素（余数为1或2），存入A[0]
    while(low<high && A[low].key%3 ==0)  //余数不为0，low右移
        low++;
        //找到第一个余数不为0的元素，存入A[0]
    A[0]=A[low];
          //余数为0，和余数1、2的元素划分出来
    while(low<high)
    {
            //右端找出一个余数为0的元素，存到左端
        while(low<high && A[high].key%3 !=0)
            high--;
        A[low]=A[high];
        while(low<high && A[low].key%3==0)  //余数为0，low右移
```

```
            low++;
        A[high]=A[low];
    }
    A[low]=A[0];  //A[low]中是余数不为 0 的第一个元素

        //第二部：将右端子表按余数 1 和 2 进行二分划分
        //下面再按余数 1 和 2，对右端子表进行二分划分,子表为 A[low...n]
        //low 指在第一个余数 1 或 2 的元素上，不需改动；high 重新初始化
    high=listLen;

        //从子表左端找出第一个余数为 2 的元素，存到 A[0]
    while(low<high && A[low].key%3==1)
        low++;
    A[0]=A[low];

    while(low<high)
    {
        while(low<high && A[high].key%3==2)
            high--;
        A[low]=A[high];
        while(low<high && A[low].key%3==1)
            low++;
        A[high]=A[low];
    }
    A[low]=A[0];
}
```

8.8 设计算法以实现如下功能：不用完整排序，求解出按大小关系为第 k 位的元素。

【解】

堆排序：建立 k 个元素的小根堆。先顺序取出前 k 个元素，建立小根堆。对剩下元素循环，依次扫描，比当前堆顶值小的元素跳过，比堆顶值大的元素，用以取代堆顶，重新调整为小根堆。循环结束时，此小根堆上就是表中前 k 个最大的元素，堆顶就是第 k 位的元素。时间复杂度约为：$O(n\log_2 k)$。

也可用简单选择排序实现，时间复杂度约为：O(n*k)。

8.9 判断下列各序列是否是堆：

(a) (100, 60, 80, 90, 40, 20, 30, 70, 35)

(b) (100, 80, 90, 60, 40, 20, 70, 30, 35)

(c) (100, 90, 80, 40, 20, 70, 30, 60, 35)

(d) (20, 30, 40, 35, 60, 80, 90, 100, 70)

【解】b 和 c 为堆，其中，b 为大根堆，d 为小根堆。

8.10 将下面数据表分别调整为大根堆和小根堆。

(50 30 120 25 85 40 100 12 90 15 60 35 105 78 10 28)

【解】

调整为小根堆，往上筛的子树根结点用红色字体表示；往下筛的路径用蓝色字体表示。大根堆请同学自己完成。

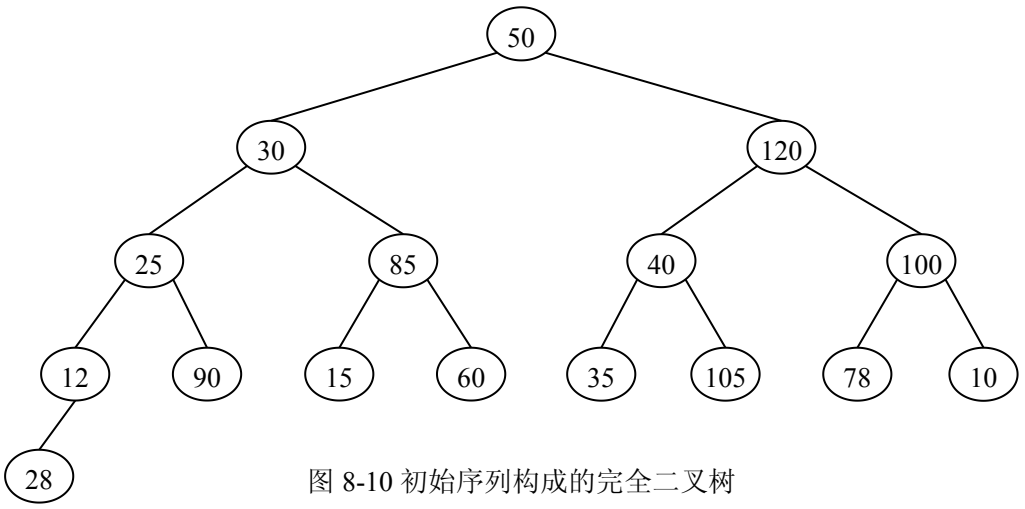

图 8-10 初始序列构成的完全二叉树

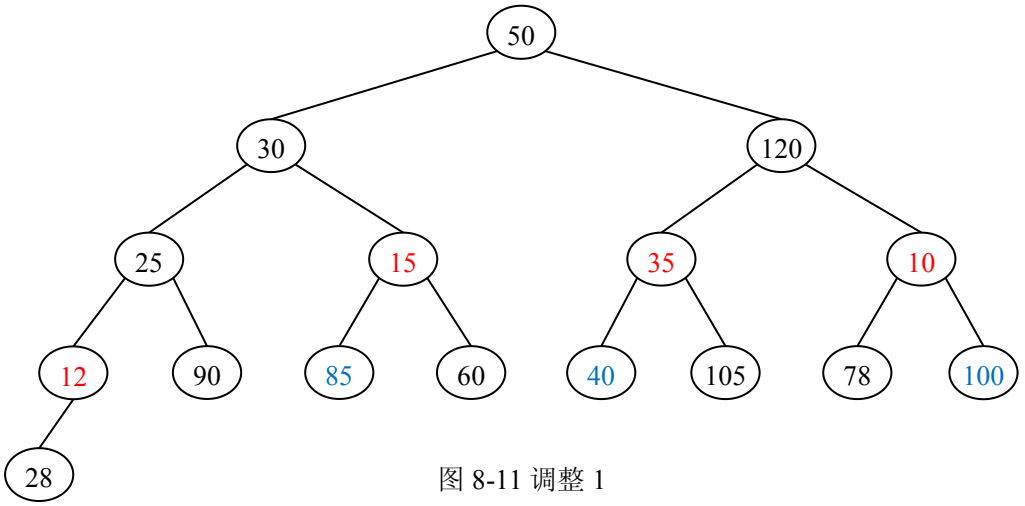

图 8-11 调整 1

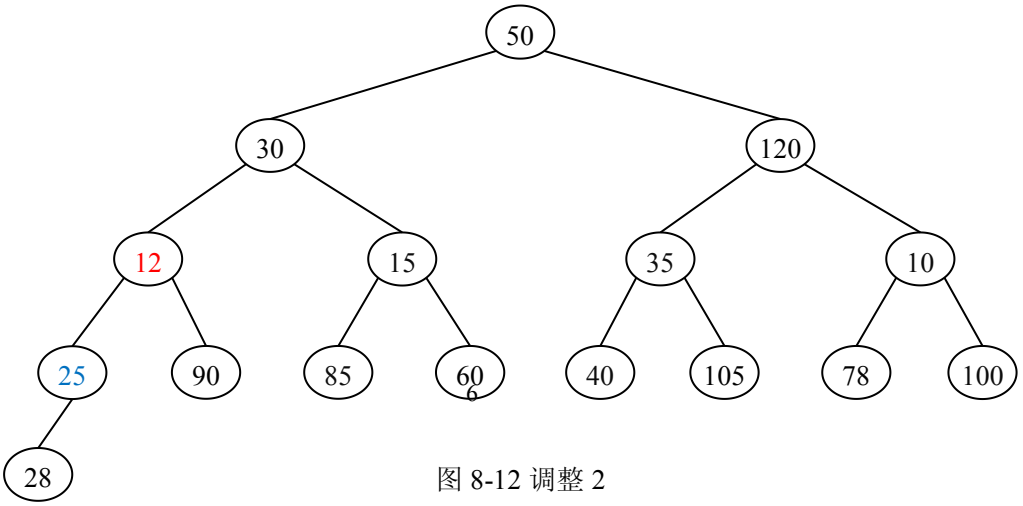

图 8-12 调整 2

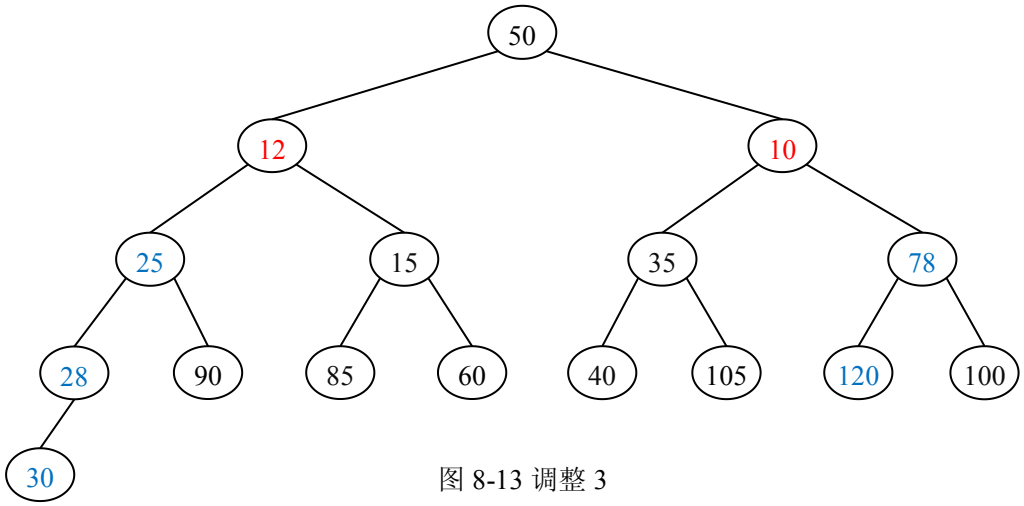

图 8-13 调整 3

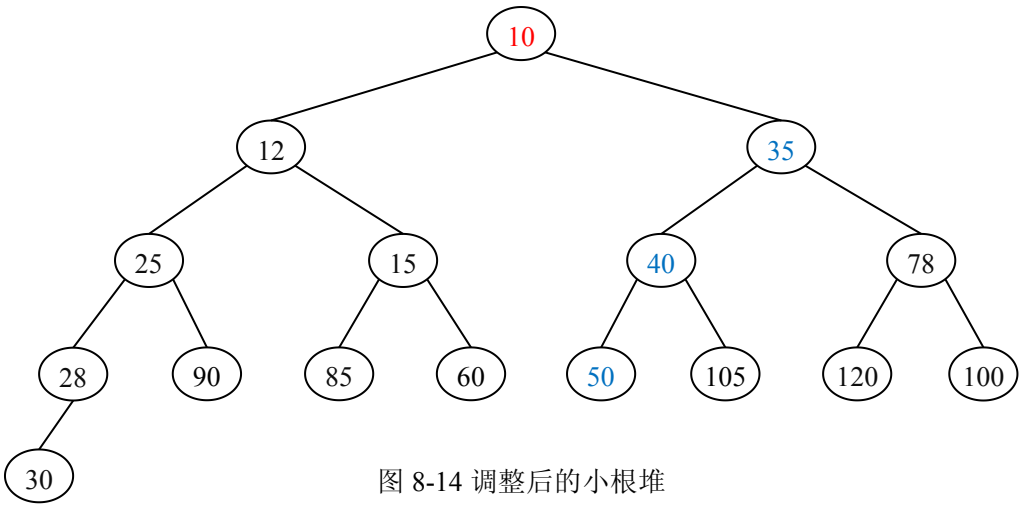

图 8-14 调整后的小根堆

8.11 由初始建堆过程的讨论可知，调整过程可以是递归形式的，请写出这一递归形式的算法。

【解】

建初堆的递归实现

```
void CreateHeap(elementType A[], int i, int n)
{
    //n 为结点个数（元素个数）
    //i 为当前子树结点编号
            //初建堆--由初始序列产生堆（此处为大根堆）
            //从第 n/2 结点开始往上筛，直到 1 号结点（根、堆顶）
    if(i>=1)
    {
        Sift(A, i, n);  //每次调用此函数，都将以 i 为根结点的子树调整为堆。
```

```
            i--;
            CreateHeap(A[],i, n);  //递归调用
        }
    }
```

初始调用为：CreateHeap(A, n/2, n);

8.12 对长度为 10000 的数据表，如果在不用完整排序的情况下，要求找出其中最大的 10 个数，应选用何种排序算法最节省时间？

【解】

堆排序。思想同 8.8 题。

8.13 如果递增有序的数据表 A、B 分别代表一个集合，设计算法以求解此类集合的交、并差等运算

【解】见线性表一章相关习题求解方法。