第十章　　排　　序

10.1　概述
10.2　插入排序
10.3　快速排序
10.4　选择排序
10.5　归并排序

学习要点

1　理解排序的定义和各种排序方法的特点；

2　了解各种方法的排序过程及其依据的原则；

3　理解排序方法“稳定”或“不稳定”的含义；

10.1 概 述

1. 排序的定义

排序也是数据处理中经常使用的一种操作。例高考考生信息管理系统提供了将考生按总分排序、按单科排序的功能。

排序是将一组“无序”的记录序列调整为“有序”的记录序列。

例如：将下列关键字序列

52，49，80，36，14，58，61，23，97，75

调整为

14，23，36，49，52，58，61 ，75，80，97

设R1 R2 R3 … Rn 是n个记录，k1,k2, k3 … kn为它们的关键字，排序就是将记录按关键字递增（或递减）的次序排列起来。

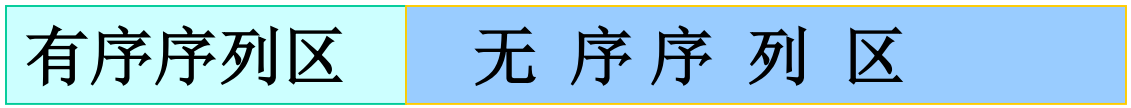
2. 内部排序和外部排序

内部排序：若整个排序过程不需要访问外存便能完成.

外部排序：若参加排序的记录数量很大，整个序列的排序过程不可能在内存中完成.

3 . 内部排序的方法

在排序的过程中，参与排序的记录序列中存在两个区域：有序区和无序区。内部排序的过程是一个逐步扩大记录的有序序列长度的过程。



4. 排序的稳定性：

在待排记录序列中，任何两个**关键字相同**的记录，用某种排序方法排序后**相对位置不变**，则称这种排序方法是**稳定的**，否则称为**不稳定的**。

例　设 49, 38, 65, 97, 76, 13, 27, 49 是待排序列

排序后：13, 27, 38, 49, 49, 65, 76, 97 —— *稳定*

排序后：13, 27, 38, 49, 49, 65, 76, 97 —— *不稳定*

5.约定

在本章中待排序记录用顺序表存储

顺序表类型说明:

```
#define MAXSIZE 20      //一个用作示例的顺序表的最大长度
typedef   int   KeyType;     //定义关键字类型为整数类型
typedef   struct{
    KeyType  key;            //关键字项
    InfoType  otherinfo;     //其它数据项
}RedType;                    //记录类型

typedef struct{
    RedType  r[MAXSIZE+1];   //r[0]闲置或用作哨兵单元
     int  length;              //顺序表长度
}SqList;                     //顺序表类型
```

10.2　插入排序

插入排序的基本思想：

每一趟将一个待排序的记录，按其关键字的大小，插入到前面已经排好序的一组记录的适当位置上，直到所有待排序记录全部插入为止。

10.2.1 直接插入排序

直接插入排序的基本操作是将一条记录插入到已排好序的有序表中，从而得到一个新的、记录数量增1的有序表。

算法步骤：

(1) 设待排序的记录存放在数组r[1..n]中，r[1]是一个有序序列；

(2)循环n-1次，每次使用顺序查找法，查找r[i](i=2,…,n)在已排好序的序列r[1…i-1]中的插入位置，然后将r[i]插入有序序列r[1…i-1]，直到将r[n]插入有序序列r[1…n-1]，最后得到一个表长为n的有序序列。

例　　49　38　65　97　76　13　27　49

排序结果：　(13　27　38　49　49　65　76　97)

插入排序的关键：如何查找插入位置。

直接插入排序（顺序插入排序）：

用**顺序查找**法定位插入位置。

折半插入排序：

采用**折半查找**法定位插入位置.

算法描述如下：

```
void InsertiSort ( SqList &L)
{   // 对顺序表L作直接插入排序
    for ( i=2; i<=L.length; ++i ) {
        if (L.r[i].key<L.r[i-1].key){
            L.r[0] = L.r[i];     // 将待插入的记录暂存到监视哨中
            for ( j=i-1;   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
        }
}
```

实现排序的基本操作有两个：

（1）“比较”序列中两个关键字的大小；

（2）“移动”记录。

对于直接插入排序：

最好的情况（关键字在记录序列中顺序有序）：

“比较”的次数：

$$\sum_{i=2}^{n} 1 = n - 1$$

“移动”的次数：

$$0$$

最坏的情况（关键字在记录序列中逆序有序）：

“比较”的次数：

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

“移动”的次数：

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

直接插入排序特点：

1） 算法简单

2） 时间复杂度为0（n^2 ）

3）稳定排序

该方法适用于记录基本（正向）有序或n较少的情况

折半插入排序

折半插入排序在寻找插入位置时，不是逐个比较而是利用折半查找的原理寻找插入位置。待排序元素越多，改进效果越明显。

算法描述如下：

```
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=mid-1; //插入点在前一子表
        else   low=mid+1;          //插入点在后一子表
     }
      for( j=i-1; j>=high+1; – –j )   L[j+1]=L[j];   //  记录后移
      L.r[high+1]=L.r[0];   //  插入
   }
 }
```

10.2.3 希尔排序

直接插入排序法简单，适用于记录较少，或待排记录基本（正向）有序的情况。 基于直接插入排序上述特点，希尔提出了另一种插入排序算法。

1. 基本思想：

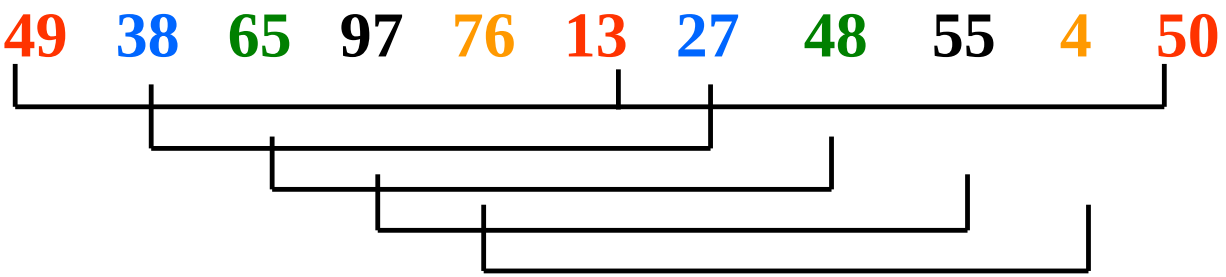
先将整个待排记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时。再对全体记录进行一次直接插入排序。

关键： 这种子序列不是由相邻的记录构成的，而是由相隔某个“增量”的记录组成一个子序列。

例如：已知待排序记录的关键字序列为{49, 38, 65, 97, 76, 13, 27, 48, 55, 4, 50}，用希尔排序法进行排序的过程如下：

取d1=5

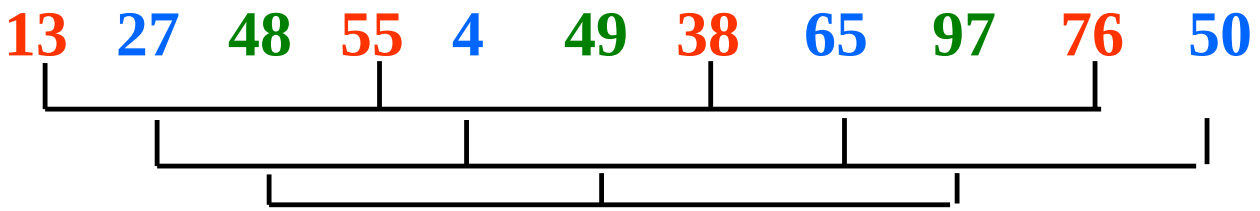
一趟分组：



第一趟排序结果：13 27 48 55 4 49 38 65 97 76 50

取d2=3

二趟分组：



第二趟排序结果：13 4 48 38 27 49 55 50 97 76 65

取d3=1

三趟分组：13 4 48 38 27 49 55 50 97 76 65

第三趟排序结果：4 13 27 38 48 49 50 55 65 76 97

希尔排序算法的实现：

```
void ShellInsert(SqList &L, int dk)
{   // 对顺序表L作一趟希尔插入排序。
   for (i=dk+1; i<=L.length; ++i)
      if (L.r[i].key<L.r[i-dk].key){  // 需将L.r[i]插入有序增量子表
         L.r[0] = L.r[i];              // 暂存在L.r[0]
         for (j=i-dk; j>0&&L.r[0].key< L.r[j].key; j-=dk)
            L.r[j+dk] = L.r[j];        // 记录后移，查找插入位置
         L.r[j+dk] = L.r[0];           // 插入
      }
```

取d1=5

1	2	3	4	5	6	7	8	9	10	11
49	38	65	97	76	13	27	48	55	4	50

```
void ShellSort(SqList &L, int dlta[], int t)
{   // 按增量序列dt[0..t-1]对顺序表L作希尔排序。
    for (k=0; k<t; ++k)
        ShellInsert(L, dlta[k]);    // 一趟增量为dt[k]的插入排序
}
```

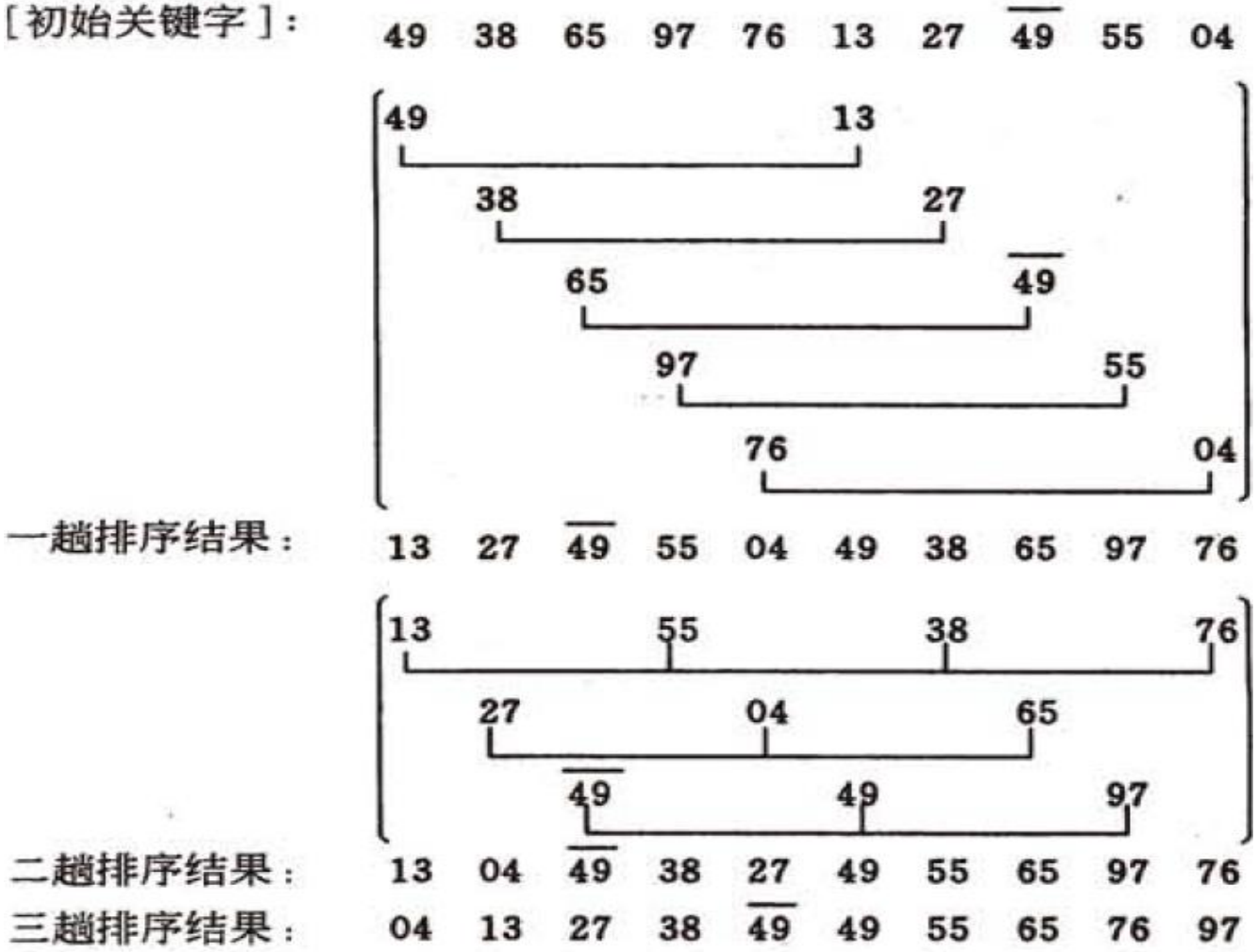

[初始关键字]：
49 38 65 97 76 13 27 49 55 04
一趟排序结果：
13 27 49 55 04 49 38 65 97 76
二趟排序结果：
13 04 49 38 27 49 55 65 97 76
三趟排序结果：
04 13 27 38 49 49 55 65 76 97

图 10.5　希尔排序示例

希尔排序的特点:

1）时间复杂度，取决于增量序列的选择，选择的好，效率优于直接插入排序;

2）不稳定排序方法;

3）适用于n 较大情况.

10.3 快速(交换)排序

基本思想：

两两比较待排序记录的关键字，一旦发现两个记录不满足次序要求时则进行交换，直到整个序列全部满足要求为止。

一、冒泡排序

冒泡排序是一种最简单的交换排序方法。它通过两两比较相邻记录的关键字，如果发生逆序，则进行交换，从而使关键字小的记录如气泡一样逐渐往上“漂浮”（左移），或者使关键字大的记录如石块一样逐渐向下“坠落”（右移）。

首先将第一个记录的关键字和第二个记录的关键字进行比较，若为逆序，则交换两个记录。然后比较第二个记录和第三个记录的关键字。依此类推，直至第n-1个记录和第n个记录的关键字进行过比较为止。上述过程称作第一趟冒泡排序，其结果使得关键字最大的记录被安置到最后一个记录的位置上。

然后进行第二趟冒泡排序，对前n-1个记录进行同样操作，其结果是使关键字次大的记录被安置到第n-1个记录的位置上。

重复上述比较和交换过程，直到在某一趟排序过程中没有进行过交换记录的操作为止，完成排序。

冒泡排序例：

初始关键字：8　11　25　49　25　16

8	11	25	49	25	16

第1趟：8　11　25　25　16　49

第2趟：8　11　25　16　25　49

第3趟：8　11　16　25　25　49

第4趟：8　11　16　25　25　49

冒泡排序算法：

```
void BubbleSort(SqList &L)
{ //  冒泡排序
   for(i=1; i<=L.length-1;i++)  //排序趟数
   {    flag=0;
        for(j=1; j<=L.length-i; j++)
            if(L.r[j].key>L.r[j+1].key)
             {   flag=1;
                 L.r[j] <--> L.r[j+1] //交换前后两个记录
             }
         if (flag==0) return;
   }
}
```

时间分析:

最好的情况（关键字在记录序列中顺序有序）：

只需进行一趟起泡

“比较”的次数：

n-1

“移动”的次数：

0

最坏的情况（关键字在记录序列中逆序有序）：

需进行n-1趟起泡

“比较”的次数：

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

“移动”的次数：

$$3\sum_{i=n}^{2}(i-1)=\frac{3n(n-1)}{2}$$

冒泡排序特点：

(1)平均时间复杂度为$O(n^2)$

(2)稳定排序

二、快速排序

基本思想：

- 在待排序的n个记录中任取一个记录(通常取第一个)作为支点，设其关键字为pivotkey。经过一趟排序后，把所有关键字小于pivotkey的记录交换到前面，把所有关键字大于pivotkey的记录交换到后面，结果将待排序记录分成两个子表，最后将支点放置在分界处的位置。
- 然后，分别对左、右子表重复上述过程，直至每一子表只有一个记录时，排序完成。

如：　**49　38　65　97　76　13　27　49**

{ 27　38　13 }　49 { 76　97　65　49 }

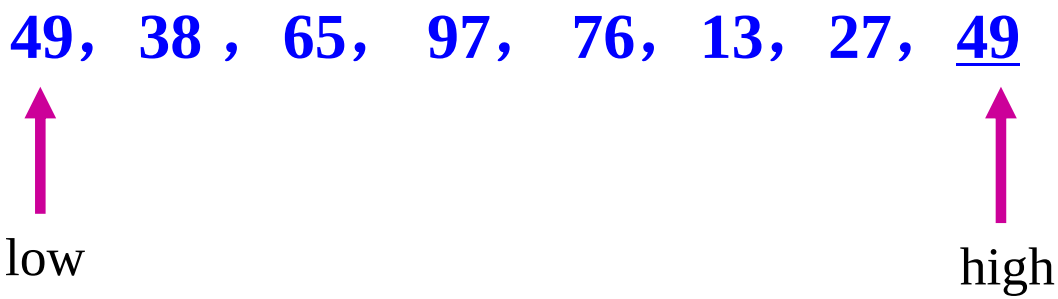
其中，一趟快速排序的过程如下：

(1) 选择待排序表中的第一个记录作为支点，将支点记录暂存在r[0]的位置上。附设两个指针low和high，初始时分别指向表的下界和上界。

(2)从指针high指向的位置依次向左搜索，找到第一个关键字小于支点关键字pivotkey的记录，将其移到low处。

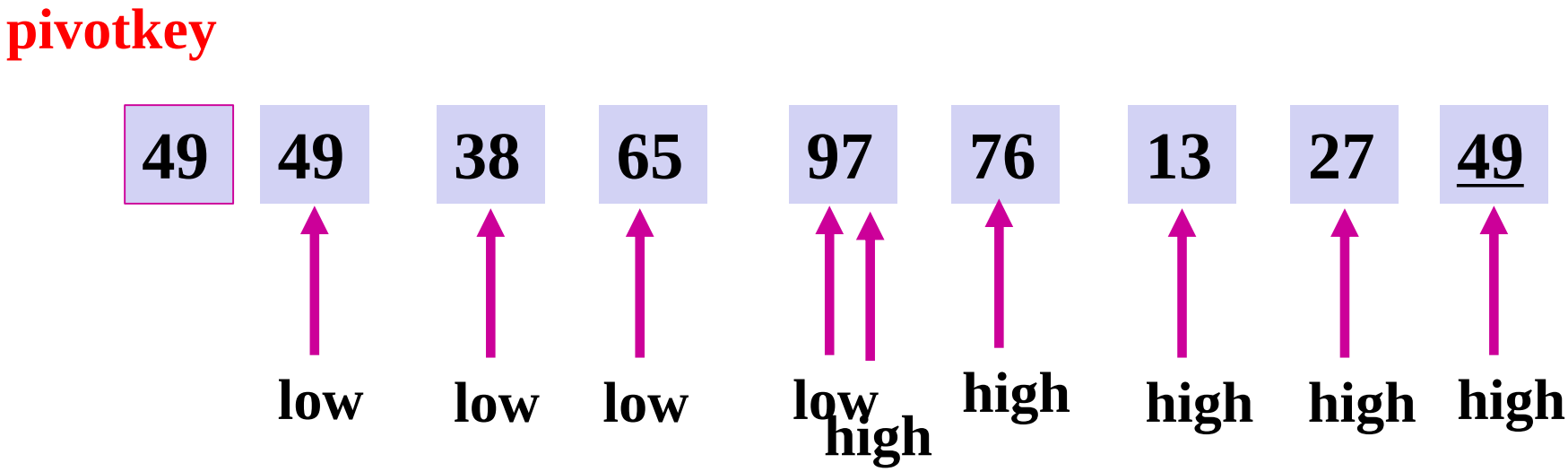
(3) 然后再从指针low指向的位置，依次向右搜索找到第一个关键字大于pivotkey的记录，将其移到high处。

(4)重复步骤2、3，直到low=high，再将r[0]中的支点记录移至low或high的位置。



例如：已知待排序记录的关键字序列为：

{49，38，65，97，76，13，27，49}，用快速排序法进行排序的第一趟排序过程如下演示：



一趟排序结果为：

{ 27　38　13 }　49　{ 76　97　65　49 }

课堂习题：

已知待排序记录的关键字序列为：{ 50，49 ，55， 65， 76，60，49 }，

用快速排序法进行排序的第一趟结果为：

快速排序的算法如下所示：

```
int Partition(SqList &L, int low, int high)
{  //对顺序表L中的子表 r[low..high]进行一趟排序，返回支点位置
    L.r[0] = L.r[low];        // 用子表的第一个记录作为支点记录
   pivotkey = L.r[low].key;   // 支点记录关键字
   while (low<high){          // 从表的两端交替地向中间扫描
        while (low<high && L.r[high].key>=pivotkey) --high;
        L.r[low] = L.r[high];  // 将比支点记录小的记录移到低端
        while (low<high && L.r[low].key<=pivotkey) ++low;
        L.r[high] = L.r[low];  // 将比支点记录大的记录移到高端
   }
   L.r[low] = L.r[0];         // 支点记录到位
   return low;                // 返回枢轴位置
}
```

```
void QSort(SqList &L, int low, int high)
{  // 对顺序表L中的子序列L.r[low..high]作快速排序
   if (low<high){                    // 长度大于1
      pivotloc = Partition(L, low, high); // 将L.r[low..high]一分为二
      QSort(L, low, pivotloc-1);  //对低子表递归排序，pivotloc是支点位置
      QSort(L, pivotloc+1, high);    // 对高子表递归排序
   }
}


void QuickSort(SqList &L)
{  //对顺序表作快速排序
     Qsort(L, 1, L.length);
}
```

快速排序特点：

1） 时间复杂度为0（$nlog_2n$）

2）**不稳定**

10.4　选择排序

选择排序的基本思想：

每一趟从待排序的记录中选出关键字最小的记录，按顺序放在已排序的记录序列的最后，直到全部排完为止。

一、简单选择排序

排序过程:

- 首先通过n-1次关键字比较，从n个记录中找出关键字最小的记录，将它与第一个记录交换.
- 再通过n-2次比较，从剩余的n-1个记录中找出关键字最小的记录，将它与第二个记录交换.
- 重复上述操作，共进行n-1趟排序后，排序结束

例如，已知待排序记录的关键字序列为{49, 38, 65, 97, 49 ,13, 27,76}，用简单选择排序法进行排序的过程如下：

初始关键字	49 38 65 97 49 13 27 76
一趟排序结果	(13) 38 65 97 49 49 27 76
二趟排序结果	(13 27) 65 97 49 49 38 76
三趟排序结果	(13 27 38) 97 49 49 65 76
四趟排序结果	(13 27 38 49) 97 49 65 76
五趟排序结果	(13 27 38 49 49) 97 65 76
六趟排序结果	(13 27 38 49 49 65) 97 76
七趟排序结果	(13 27 38 49 49 65 76) 97

算法描述如下:

```
void SelectSort(SqList &K)
{   //对顺序表L作简单选择排序
  for (i=1; i<L.length; ++i){
      j=SelectMinKey(L, i);
            //在L.r[i..L.length] 中选择关键字最小记录
      if(i!=j)   L.r[i]←→L.r[j];          //与第i个记录交换
  }
}
```

算法特点:

1) 时间复杂度为$O(n^2)$

2)不稳定排序

二. 堆排序

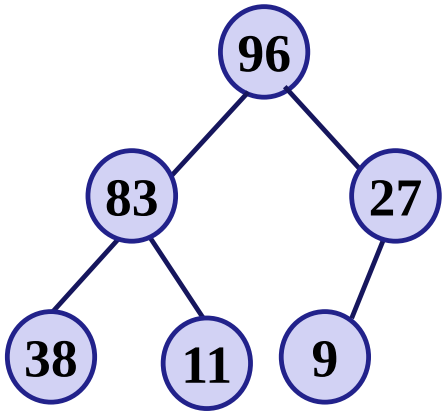
堆的定义：

n个元素的序列(k1, k2, …, kn)，当且仅当满足下列关系时，称之为堆:

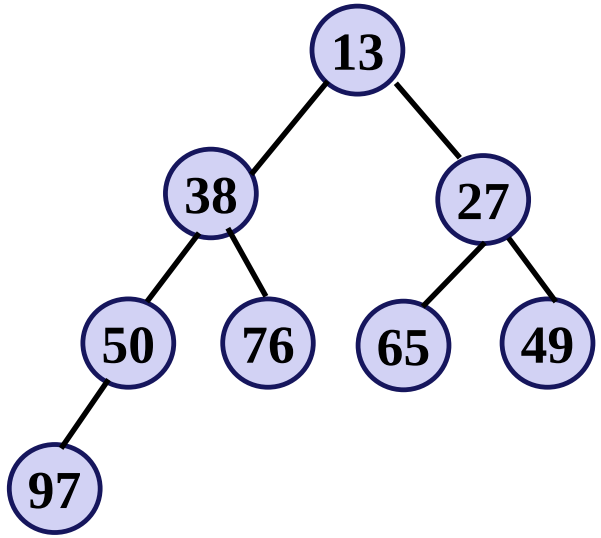
(1) $k_i \geq k_{2i}$且$k_i \geq k_{2i+1}$　或　(2) $k_i \leq k_{2i}$且$k_i \leq k_{2i+1}$ $(i=1,2,\ldots\lfloor n/2 \rfloor)$

若将和此序列对应的一维数组看成一个完全二叉树，则堆实质上是满足如下性质的二叉树：树中所有非终端结点的值均不大于(或不小于)其左、右孩子结点的值。

例如关键字序列（96，83，27，38，11，9）和（13，38，27，50，76，65，49，97）分别满足条件1和条件2，故它们均为堆，对应的完全二叉树分别如图a和b所示。显然，在这两种堆中，堆顶元素必为序列中n个元素的最大值(或最小值)，分别称之为大根堆和小根堆。



a. 大根堆



b. 小根堆

以小根堆进行排序，堆排序的过程如下：

按堆的定义将待排序序列调整为小根堆(这个过程称为建初堆)，若在输出堆顶的最小值之后，使得其余n-1个元素的序列重又建成一个堆，则得到n个元素中的次小值。如此反复，便能得到一个有序序列，这个过程称之为堆排序。

堆排序需要解决两个问题：

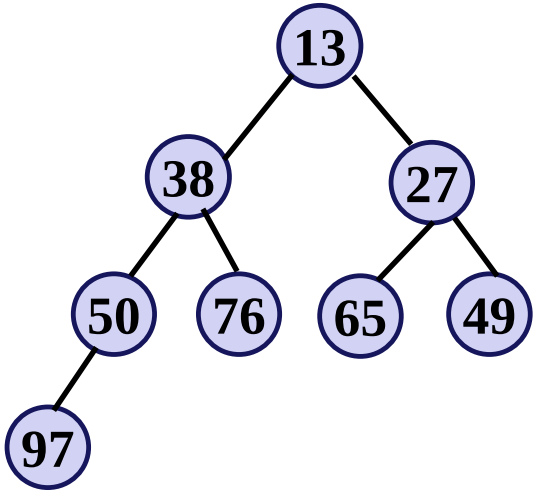
1. 建初堆：如何将 一个无序序列建成一个堆？

2. 调整堆：如何在输出堆顶元素后，调整剩余元素成为一个新的堆？

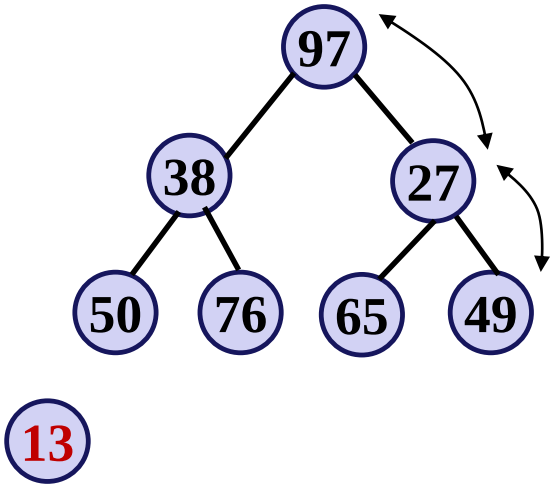
第二个问题解决方法——筛选

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”。

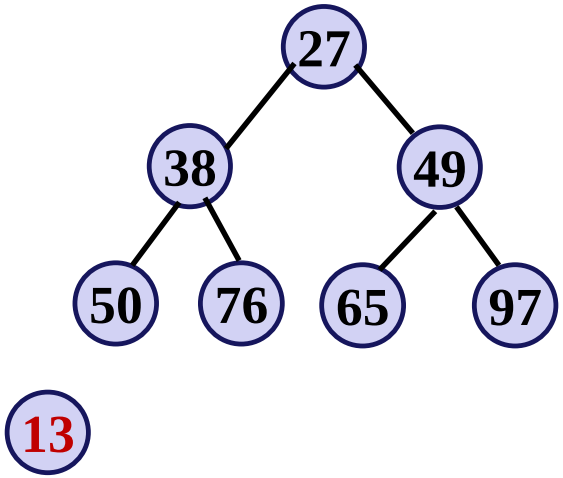
例如：以下是一个小根堆的筛选过程。



a. 堆



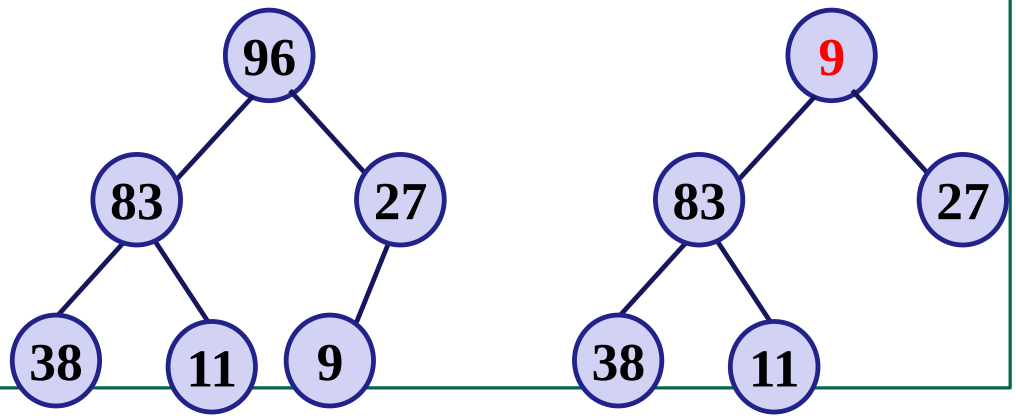
b. 输出13，用97替代后



c. 调整后的新堆

假设r[s+1..m]已经是堆，按“筛选法”将r[s..m]调整为以r[s]为根的大根堆，算法描述如下：

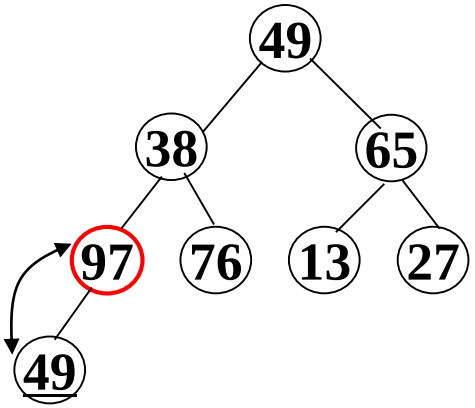
```
void HeapAdjust (HeapType &H, int s, int m)
{  // 假设r[s+1..m]已经是堆，将r[s..m]调整为以r[s]为根的大根堆
   rc = H.r[s];
   for (j=2*s; j<=m; j*=2){   // 沿key较大的孩子结点向下筛选
      if (j<m && H.r[j].key< H.r[j+1].key)
         ++j;                // j为key较大的记录的下标
      if ( rc.key>= H.r[j].key)  break;// rc应插入在位置s上
      H.r[s] = H.r[j];
      s = j;
   }
   H.r[s] = rc;              // 插入
}
```



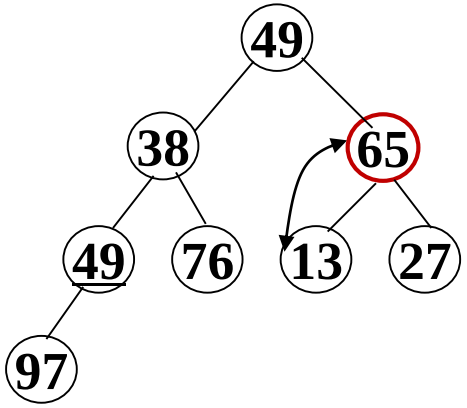
第一个问题解决方法

方法：从无序序列的第⌊n/2⌋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。

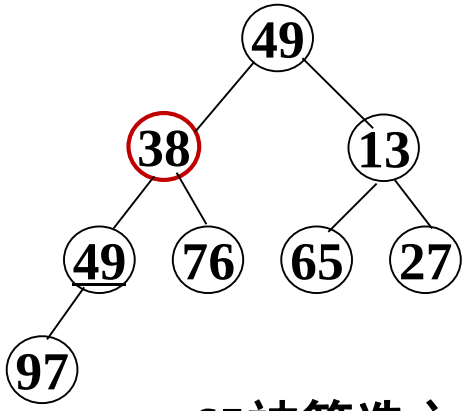
例 图a中的二叉树表示一个有8个元素的无序序列（49，38，65，97，76，13，27，49），建小根堆的过程如下：



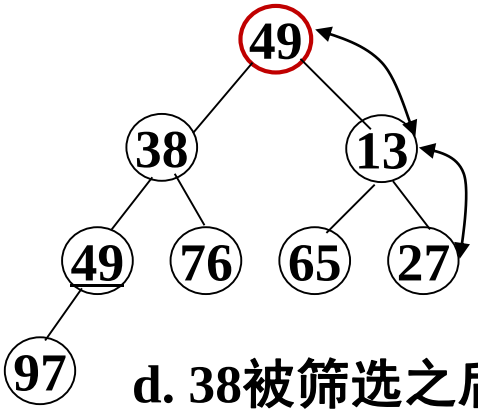
a.无序序列



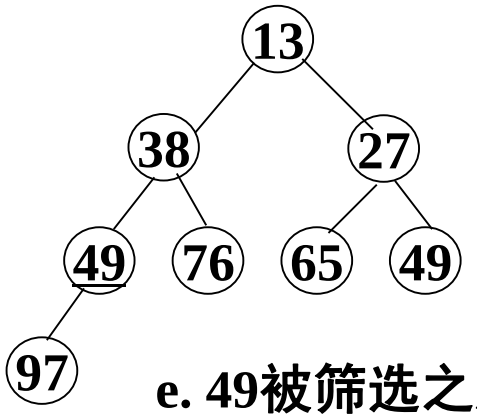
b.97被筛选之后的状态



c. 65被筛选之后的状态



d. 38被筛选之后的状态



e. 49被筛选之后建成的堆

对顺序表H进行堆排序的算法描述如下：

```
void HeapSort(HeapType &H)
{   // 对顺序表H进行堆排序
   for (i=H.length/2; i>0; --i)      // 把r[1..H.length]建成大顶堆
      HeapAdjust(H, i, H.length);
   for (i=H.length; i>1; --i){
      Swap(H.r[1], H.r[i]);    // 将堆顶记录和当前未经排序子序列
                               // H.r[1..i]中最后一个记录相互交换
      HeapAdjust(H, 1, i-1);   // 将r[1..i-1]重新调整为大顶堆
   }
}
```

课堂练习：

1. 下列（　　）是一个堆。

A. 19, 75, 34, 26, 97, 56　　B. 97, 26, 34, 75, 19, 56

C. 19, 56, 26, 97, 34, 75　　D. 19, 34, 26, 97, 56, 75

2. 有一组数据(15，9，7，8，20，−1，7，4)，用堆排序的筛选方法建立的初始小根堆为（　　）。

A. -1, 4, 8, 9, 20, 7, 15, 7　　B. 97, 26, 34, 75, 19, 56

C. -1, 4, 7, 8, 20, 15, 7, 9　　D. A、B、C均不对

3. 对关键码序列(23，17，72，60，25，8，68，71，52)进行堆排序，输出两个最小关键码后的剩余堆是（　　）。

A. 23, 72, 60, 25, 68, 71, 52

B. 23, 25, 52, 60, 71, 72, 68

C. 71, 25, 23, 52, 60, 72, 68

D. 23, 25, 68, 52, 60, 72, 71

10.5 归并排序

一 基本思想：

将两个或多个有序表归并成一个有序表。

二 2 路归并

1）设有n个待排序记录，初始时将它们分为n个长度为 1有序子表；

2）两两归并相邻有序子表，得到若干个长度2为的有序子表；

3）重复2）直至得到一个长度为n的有序表；

例 待排记录 [49] [38] [65] [97] [76] [13] [27] [49]

[38 49] [65 97] [13 76] [27 49]

[38 49 65 97] [13 27 76 49]

13 27 38 49 49 65 76 97

一趟2路归并排序结果

整个归并排序需 $\lceil \log_2 n \rceil$ 趟

特点：

1）时间复杂度为0（$n\log_2 n$）

2）稳定排序

- 插入排序
  - 直接插入排序
  - 希尔排序
- 快速排序
  - 冒泡排序
  - 快速排序
- 选择排序
  - 简单选择排序
  - 堆排序
- 归并排序 → 2-路归并排序

课堂作业：

给定一个关键字序列{24，19，32，43，38，6，13，22}，请分别写出应用快速排序、2-路归并排序、希尔排序（增量d1=5），排序第一趟的结果。