10-1 什么是内排序？什么是外排序？什么排序方法是稳定的？什么排序方法是不稳定的？
【解答】参考教材

10-2 设待排序的关键码序列为{12, 2, 16, 30, 28, 10, 16*, 20, 6, 18}，试分别写出使用以下排序方法每趟排序后的结果。并说明做了多少次关键码比较。

(1) 直接插入排序　(2) 希尔排序(增量为 5,2,1)　(3) 起泡排序
(4) 快速排序　(5) 直接选择排序　(6) 锦标赛排序
(7) 堆排序　(8) 二路归并排序　(9) 基数排序

【解答】参考教材

10-3 在起泡排序过程中，什么情况下关键码会朝向与排序相反的方向移动，试举例说明。在快速排序过程中有这种现象吗？

【解答】

如果在待排序序列的后面的若干关键码比前面的关键码小，则在起泡排序的过程中，关键码可能向与最终它应移向的位置相反的方向移动。例如，

57 40 38 11 13 34 48 75 6 19 9 7　如 9 向相反方向移动
6 57 40 38 11 13 34 48 75 7 19 9　如 19 向相反方向移动
6 7 57 40 38 11 13 34 48 75 9 19　如 9 向最终方向移动
6 7 9 57 40 38 11 13 34 48 75 19　如 13 向相反方向移动
6 7 9 11 57 40 38 13 19 34 48 75　如 13 向最终方向移动
6 7 9 11 13 57 40 38 19 34 48 75　如 34 向相反方向移动
6 7 9 11 13 19 57 40 38 34 48 75
6 7 9 11 13 19 34 57 40 38 48 75
6 7 9 11 13 19 34

10-4 试修改起泡排序算法，在正反两个方向交替进行扫描，即第一趟把关键码最大的对象放到序列的最后，第二趟把关键码最小的对象放到序列的最前面。如此反复进行。

【解答】

```
template <class Type> void dataList<Type> :: shake_Sort ( ) {
//奇数趟对表 Vector 从前向后, 比较相邻的关键码, 遇到逆序即交换, 直到把参加比较关键码序列
//中最大的关键码移到序列尾部。偶数趟从后向前, 比较相邻的关键码, 遇到逆序即交换, 直到把
//参加比较关键码序列中最小的关键码移到序列前端。
    int i = 1, j;    int exchange;
    while ( i < CurrentSize ) {                    //起泡排序趟数不超过 n-1
        exchange = 0;                              //假定元素未交换
        for ( j = CurrentSize-i; j >= i; j-- )     //逆向起泡
            if ( Vector[j-1] > Vector[j] ) {       //发生逆序
                Swap ( Vector[j-1], Vector[j] );   //交换, 最小关键码放在 Vector[i−1]处
                exchange = 1;                      //做“发生了交换”标志
            }
        if ( exchange == 0 ) break;                ////当 exchange 为 0 则停止排序
        for ( j = i; j <= CurrentSize−i−1; j++ )   //正向起泡
            if ( Vector[j] > Vector[j+1] ) {       //发生逆序
                Swap ( Vector[j], Vector[j+1] );       //交换, 最大关键码放在 Vector[n−i]处
                exchange = 1;                      //做“发生了交换”标志
            }
        if ( exchange == 0 ) break;                ////当 exchange 为 0 则停止排序
```

```
            i++;
        }
    }
```

10-5 如果待排序的关键码序列已经按非递减次序有序排列，试证明函数 *QuickSort*( )的计算时间将下降到 O(n^2)。

【解答】参考教材

10-6 在实现快速排序的非递归算法时，可根据基准对象，将待排序关键码序列划分为两个子序列。若下一趟首先对较短的子序列进行排序，试证明在此做法下，快速排序所需要的栈的深度为 O($\log_2 n$)。

【解答】参考教材

10-7 在实现快速排序算法时，可先检查位于两端及中点的关键码，取三者之中的数值不是最大也不是最小的关键码作为基准对象。试编写基于这种思想的快速排序算法，并证明对于已排序的关键码序列，该算法的计算时间为 O($n\log_2 n$)。

【解答】参考教材

10-8 在使用非递归方法实现快速排序时, 通常要利用一个栈记忆待排序区间的两个端点。那么能否用队列来代替这个栈? 为什么?

【解答】

可以用队列来代替栈。在快速排序的过程中，通过一趟划分，可以把一个待排序区间分为两个子区间，然后分别对这两个子区间施行同样的划分。栈的作用是在处理一个子区间时，保存另一个子区间的上界和下界，待该区间处理完成后再从栈中取出另一子区间的边界，对其进行处理。这个功能利用队列也可以实现，只不过是处理子区间的顺序有所变动而已。

10-9 试设计一个算法, 使得在 O(n)的时间内重排数组, 将所有取负值的关键码排在所有取正值(非负值)的关键码之前。

【解答】参考教材

10-10 奇偶交换排序是另一种交换排序。它的第一趟对序列中的所有奇数项 i 扫描，第二趟对序列中的所有偶数项 i 扫描。若 $A[i] > A[i+1]$，则交换它们。第三趟有对所有的奇数项，第四趟对所有的偶数项，…，如此反复，直到整个序列全部排好序为止。

(1) 这种排序方法结束的条件是什么？

(2) 写出奇偶交换排序的算法。

(3) 当待排序关键码序列的初始排列是从小到大有序，或从大到小有序时，在奇偶交换排序过程中的关键码比较次数是多少？

【解答】参考教材

10-11 请编写一个算法，在基于单链表表示的待排序关键码序列上进行简单选择排序。

【解答】参考教材

10-12 若参加锦标赛排序的关键码有 11 个，为了完成排序，至少需要多少次关键码比较？

【解答】参考教材

10-13 试给出适用于锦标赛排序的胜者树的类型声明。并写一个函数，对 n 个参加排序的对象，构造胜者树。设 n 是 2 的幂。

【解答】参考教材

10-14 手工跟踪对以下各序列进行堆排序的过程。给出形成初始堆及每选出一个关键码后堆

的变化。

(1) 按字母顺序排序：*Tim*, *Dot*, *Eva*, *Rom*, *Kim*, *guy*, *Ann*, *Jim*, *Kay*, *Ron*, *Jan*

(2) 按数值递增顺序排序：26, 33, 35, 29, 19, 12, 22

(3) 同样 7 个数字，换一个初始排列，再按数值的递增 顺序排序：12, 19, 33, 26, 29, 35, 22

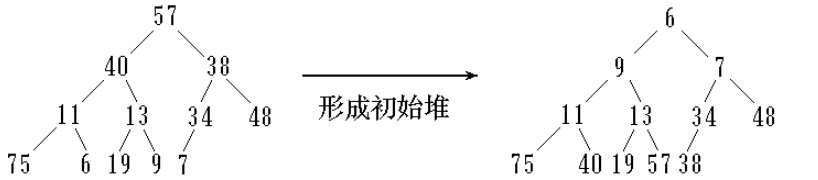
【解答】参考教材

10-15 如果只想在一个有 n 个元素的任意序列中得到其中最小的第 k ($k<<n$) 个元素之前的部分排序序列, 那么最好采用什么排序方法? 为什么? 例如有这样一个序列: {503, 017, 512, 908, 170, 897, 275, 653, 612, 154, 509, 612*, 677, 765, 094}, 要得到其第 4 个元素之前的部分有序序列: {017, 094, 154, 170}, 用所选择的算法实现时, 要执行多少次比较?

【解答】

一般来讲，当 n 比较大且要选的数据 $k << n$ 时, 采用堆排序方法中的筛选(调整)算法最好。但当 n 比较小时，采用锦标赛排序方法更好。

例如，对于序列{ 57, 40, 38, 11, 13, 34, 48, 75, 6, 19, 9, 7 }，选最小的数据 6，需形成初始堆，进行 18 次数据比较；选次小数据 7 时，需进行 4 次数据比较；再选数据 9 时，需进行 6 次数据比较；选数据 11 时，需进行 4 次数据比较。



但如果选用锦标赛排序，对于有 n 个数据的序列，选最小数据需进行 $n-1$ 次数据比较，以后每选一个数据，进行数据比较的次数，均需 $\lfloor \log_2 n \rfloor$ 次。例如，同样 12 个数据，第一次选最小的数据 6，需进行 11 次数据比较，以后选 7、9、11 时，都是 $\lfloor \log_2 12 \rfloor = 3$ 次数据比较。

10-16 希尔排序、简单选择排序、快速排序和堆排序是不稳定的排序方法, 试举例说明。

【解答】

(1) 希尔排序
{ 512 275 275* 061 } 增量为 2
{ 275* 061 512 275 } 增量为 1
{ 061 275* 275 512 }

(2) 直接选择排序
{ 275 275* 512 061 } $i = 1$
{ **061** 275* 512 275 } $i = 2$
{ **061** **275*** 512 275 } $i = 3$
{ **061** **275*** **275** 512 }

(3) 快速排序
{ 512 275 275* }
{ 275* 275 **512** }

(4) 堆排序
{ 275 275* 061 170 } 已经是最大堆，交换 275 与 170
{ 170 275* 061 **275** } 对前 3 个调整
{ 275* 170 061 **275** } 前 3 个最大堆，交换 275*与 061
{ 061 170 **275*** **275** } 对前 2 个调整
{ 170 061 **275*** **275** } 前 2 个最大堆，交换 170 与 061
{ 061 **170** **275*** **275** }

10-17 设有 *n* 个待排序元素存放在一个不带表头结点的单链表中, 每个链表结点只存放一个元素, 头指针为 *r*。试设计一个算法, 对其进行二路归并排序, 要求不移动结点中的元素, 只改各链结点中的指针, 排序后 r 仍指示结果链表的第一个结点。(提示：先对待排序的单链表进行一次扫描, 将它划分为若干有序的子链表, 其表头指针存放在一个指针队列中。当队列不空时重复执行, 从队列中退出两个有序子链表, 对它们进行二路归并, 结果链表的表头指针存放到队列中。如果队列中退出一个有序子链表后变成空队列, 则算法结束。这个有序子链表即为所求。)

【解答】

(1) 两路归并算法

```
procedure merge ( ha, hb : pointer;   var hc : pointer );
var pa, pb, pc : pointer;
  begin
    if ha↑.data <= hb↑.data
      then begin   hc := ha;   pa := ha↑.next;   pb := hb;   end
      else begin   hc := hb;   pb := hb↑.next;   pa := ha;   end;
    pc := hc;
    while ( pa <> nil ) and ( pb <> nil ) do
      if pa↑.data <= pb↑.data
          then begin
                  pc↑.next := pa;   pc := pa;   pa := pa↑.next;
               end
          else begin
                  pc↑.next := pb;   pc := pb;   pb := pb↑.next;
               end;
    if pa <> nil then   pc↑.next := pa
                  else   pc↑.next := pb;
  end;
```

(2) 归并排序主程序

```
procedure mergesort( var r : pointer );
  var s, t : pointer;   var Q : Queue;
  begin
    if r = nil then return;
    SetEmpty ( Q );   s := r;   Enqueue( Q, r );
    while s <> nil do
        begin
          t := s↑.next;
          while ( t <> nil ) and ( s↑.data <= t↑.data ) do
              begin   s := t;   t := t↑.next;   end;
          if t <> nil then
              begin   s↑.next := nil;   s := t;   EnQueue( Q, s );   end;
        end;
    while not IsEmpty( Q ) do
        begin
          r := DlQueue( Q );
          if IsEmpty( Q ) then break;
          s := DlQueue( Q );
```

merge(*r*, *s*, *t*); *EnQueue*(*Q*, *t*);
end
end;

10-18 若设待排序关键码序列有 *n* 个关键码，*n* 是一个完全平方数。将它们划分为$\sqrt{n}$块，每块有$\sqrt{n}$个关键码。这些块分属于两个有序表。下面给出一种 O(1)空间的非递归归并算法：

step1: 在两个待归并的有序表中从右向左总共选出$\sqrt{n}$个具有最大值的关键码；

step2: 若设在 **step1** 选出的第 2 个有序表中的关键码有 *s* 个，则从第 1 个有序表选出的关键码有$\sqrt{n}$- *s* 个。将第 2 个有序表选出的 *s* 个关键码与第 1 个有序表选出的关键码左边的同样数目的关键码对调；

step3: 交换具有最大$\sqrt{n}$个关键码的块与最左块(除非最左块就是具有最大$\sqrt{n}$个关键码的块)。对最右块进行排序；

step4: 除去具有最大$\sqrt{n}$个关键码的块以外，对其它的块根据其最后的关键码按非递减顺序排序；

step5: 设置 3 个指针，分别位于第 1 块、第 2 块和第 3 块的起始位置，执行多次 **substep**，直到 3 个指针都走到第$\sqrt{n}$块为止。此时前$\sqrt{n}-1$块已经排好序。

☞ **subStep** 所做的工作是比较第 2 个指针与第 3 个指针所指关键码，将值小的与第 1 个指针所指关键码对调，相应指针前进 1 个关键码位置。

step6: 对最后第$\sqrt{n}$块中最大的$\sqrt{n}$个关键码进行排序。

(1) 设有 16 个关键码，分别存放于两个有序表{10, 12, 14, 16, 18, 20, 22, 25}和{11, 13, 15, 17, 19, 21, 23, 24}中，试根据上面的描述，写出排序的全过程，并说明它具有时间复杂度 O(*n*)和空间复杂度 O(1)。

(2) 编写相应的算法。要求两个待排序有序表的长度可以不同，但每一个表的长度都是$\sqrt{n}$的倍数。

(3) 假设两个有序表分别为(x_1, …, x_m)和(x_{m+1}, …, x_n)，编写一个算法归并这两个有序表，得到(x_1, …, x_n)。设 s = $\sqrt{n}$ 。

10-19 试编写一个算法，将对象序列(x_1, x_2, …, x_n)循环右移 *p* 个位置，$0 \le p \le n$。要求该算法的时间复杂度为 O(*n*)而空间复杂度为 O(1)。

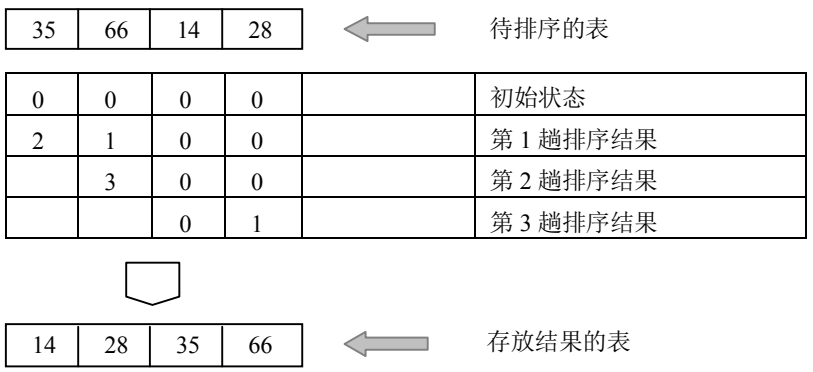
【解答】参考教材

10-20 在什么条件下，MSD 基数排序比 LSD 基数排序效率更高？

【解答】参考教材

10-21 在已排好序的序列中，一个对象所处的位置取决于具有更小关键码的对象的个数。基于这个思想，可得计数排序方法。该方法在声明对象时为每个对象增加一个计数域 *count*，用于存放在已排好序的序列中该对象前面的对象数目，最后依 *count* 域的值，将序列重新排列，就可完成排序。试编写一个算法，实现计数排序。并说明对于一个有 *n* 个对象的序列，为确定所有对象的 *count* 值，最多需要做 *n*(*n*-1)/2 次关键码比较。

【解答】



template <class Type> void datalist**<Type> ::** count_sort ( ) **{**

```
//initList是待排序表，resultList是结果表
    int i, j;
    int *c = new datalist <Type>;                          // c是存放计数排序结果的临时表
    for ( i = 0; i < CurrentSize; i++ ) Vector[i].count = 0;        //初始化，计数值都为0
    for ( i = 0; i < CurrentSize−1; i++ )
        for ( j = i+1; j < CurrentSize; j++ )
            if ( Vector[j].key < Vector[i].key ) Vector[i].count++;
            else Vector[j].count++;                                 //统计
    for ( i = 0; i < CurrentSize; i++ )                             //在c->Vector[ ]中各就各位
        c->Vector[ Vector[i].count ] = Vector[i];
    for ( i = 0; i < CurrentSize; i++ ) Vector[i] = c->Vector[i];   //结果复制回当前表对象中
    delete c;
}
```

10-22 试证明对一个有 n 个对象的序列进行基于比较的排序，最少需要执行 $n\log_2 n$ 次关键码比较。

【解答】参考教材

10-23 如果某个文件经内排序得到 80 个初始归并段，试问

(1) 若使用多路归并执行 3 趟完成排序，那么应取的归并路数至少应为多少？

(2) 如果操作系统要求一个程序同时可用的输入/输出文件的总数不超过 15 个，则按多路归并至少需要几趟可以完成排序？如果限定这个趟数，可取的最低路数是多少？

【解答】

(1) 设归并路数为 k，初始归并段个数 $m = 80$，根据归并趟数计算公式 $S = \lceil \log_k m \rceil = \lceil \log_k 80 \rceil = 3$ 得：$k^3 \geq 80$。由此解得 $k \geq 3$，即应取的归并路数至少为 3。

(2) 设多路归并的归并路数为 k，需要 k 个输入缓冲区和 1 个输出缓冲区。1 个缓冲区对应 1 个文件，有 k +1 = 15，因此 k = 14，可做 14 路归并。由 $S = \lceil \log_k m \rceil = \lceil \log_{14} 80 \rceil = 2$。即至少需 2 趟归并可完成排序。

若限定这个趟数，由 $S = \lceil \log_k 80 \rceil = 2$，有 $80 \leqslant k^2$，可取的最低路数为 9。即要在 2 趟内完成排序，进行 9 路排序即可。

10-24 假设文件有 4500 个记录，在磁盘上每个页块可放 75 个记录。计算机中用于排序的内存区可容纳 450 个记录。试问：

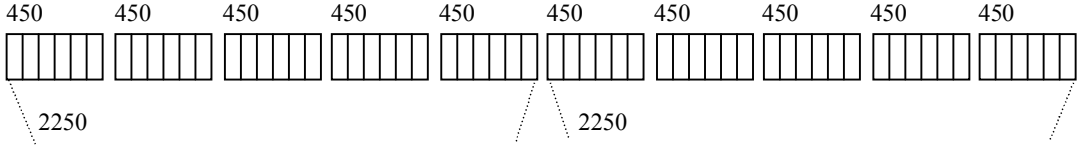
(1) 可以建立多少个初始归并段？每个初始归并段有多少个记录？存放于多少个页块中？

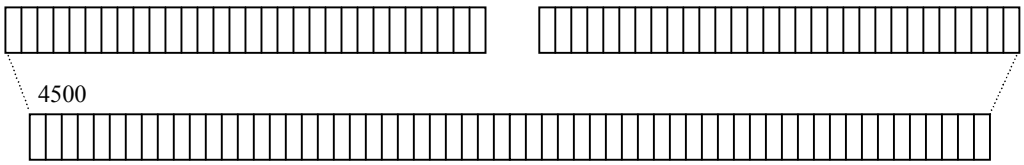
(2) 应采用几路归并？请写出归并过程及每趟需要读写磁盘的页块数。

【解答】

(1) 文件有 4500 个记录，计算机中用于排序的内存区可容纳 450 个记录，可建立的初始归并段有 4500 / 450 = 10 个。每个初始归并段中有 450 个记录，存于 450 / 75 = 6 个页块中。

(2) 内存区可容纳 6 个页块，可建立 6 个缓冲区，其中 5 个缓冲区用于输入，1 个缓冲区用于输出，因此，可采用 5 路归并。归并过程如下：

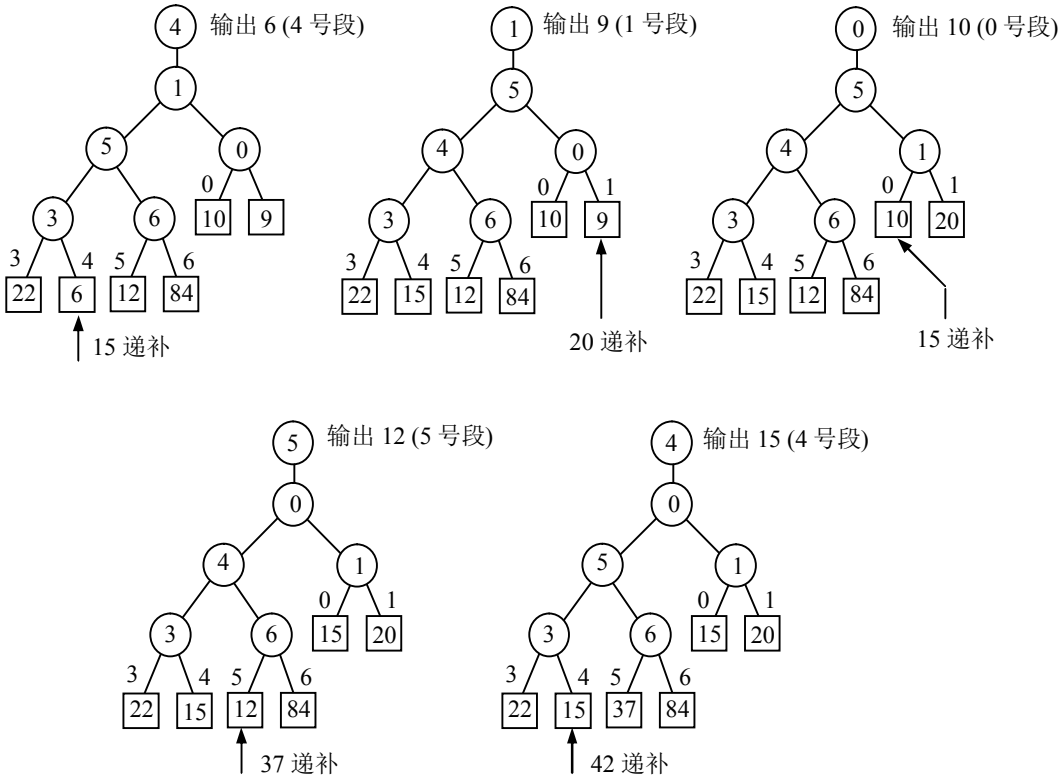




共做了 2 趟归并，每趟需要读 60 个磁盘页块，写出 60 个磁盘页块。

10-25 设初始归并段为(10, 15, 31, ∞), (9, 20, ∞), (22, 34, 37, ∞), (6, 15, 42, ∞), (12, 37, ∞), (84, 95, ∞)，试利用败者树进行 k 路归并，手工执行选择最小的 5 个关键码的过程。

【解答】做 6 路归并排序，选择最小的 5 个关键码的败者树如下图所示。



10-26 设输入文件包含以下记录：14, 22, 7, 24, 15, 16, 11, 100, 10, 9, 20, 12, 90, 17, 13, 19, 26, 38, 30, 25, 50, 28, 110, 21, 40。现采用败者树生成初始归并段，请画出选择的过程。

10-27 给出 12 个初始归并段，其长度分别为 30, 44, 8, 6, 3, 20, 60, 18, 9, 62, 68, 85。现要做 4 路外归并排序，试画出表示归并过程的最佳归并树，并计算该归并树的带权路径长度 *WPL*。

13	9.6 9.9 9.14 9.19	9.11 9.17	9.8 9.12 9.15
14	9.23 9.25 9.27	9.19 9.21	9.24 9.26