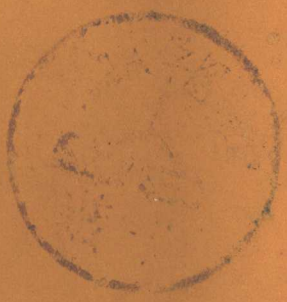
高级 Turbo C

——技巧与实例

【译】 叶常青 刘蓓敏
严学军 宋小军

【校】 宋小军 王 成



中科电脑技术公司
一九九一年三月

73·37221/602

译　者　序

在C语言编程世界中我们周游了多年，如获至宝似地发现了这本丰富而实用的Turbo C编程指南书籍。

本书作者S·希尔伯特是著名的Borland公司通用计算机实验室主任、著名C专家。他出版了大量有关C语言、人工智能等方面的书籍。

本书全面系统地阐述了TURBO C语言的各种编程问题，题材涉及到数据结构、系统资源的使用、内存的动态分配、与汇编语言的接口、图形学、加密解密、数据压缩、统计学、词法分析等许多有趣的课题。此外，还讨论了TURBO PASCAL到TURBO C的转换、程序的效率、移植、调试等程序开发实用技术。本书内容丰富，深入浅出，具体实用。每个问题均有详细程序示例以供参考。所有讨论的问题和程序实例均适合于各种TURBO C版本，同时又完全可供其他C语言的编程者参考，是C语言程序设计者很好的参考书。

本书经UNIX及C语言专家、上海交通大学计算机系尤晋元教授推荐，由上海市计算机开发公司和复旦大学应用数学中心联合CAD技术部刘蓓敏、叶常青，华东师范大学计算机系严学军，中科电脑技术公司宋小军等四位同志翻译，中科电脑技术公司宋小军、王成同志审阅、校核了全书，并修改了其中部分章节。

由于业务水平、翻译经验所限，同时又特着急于将此书推荐给读者的心情，错误肯定不少，诚望广大读者批评指示。

借此机会，我们向尤晋元教授及协助我们工作的各位同志表示感谢!

译　者

一九九零年十月

序　　言

我承认：我迷上了C。在C设计中，我发现了其与机器相结合之间的微妙关系。然而，Borlandls Turbo C又增加了一个特性，即速度。在过去许多年内，许多C编译器苦于速度太慢，编译是一个几乎难以忍受的见长过程。而今采用了Turbo C，在瞬间之内编译、连结、执行程序已成为可能。由于Turbo C的有效性，不再有何理由采用其它语言编制程序。本书将沿着应用性途径帮助你了解Turbo C的功能。每章讨论一个特定的编程题目，同时设计所讨论题目的程序。通过此过程，你可以明白Turbo C编程应用的诸多优点，而同时，又可提高你的编程技术。

第一章介始排序和搜索，第二章叙述队列，堆栈，链表和二叉树，第三章讨论动态分配和稀疏数组。第四章概述与操作系统的接口，第五章介绍Turbo C程序中如何调用汇编语言，第六章讨论图形学并设计了几个图形函数，其中包括画线，画圆函数。第七章讨论统计问题并且给出了几个完整的统计程序。第八章讨论编码、加密和数据压缩，而且还叙述了相当有趣的密码学简史。第九章详细讨论随机数发生器而且讨论了如何将其用于两个模拟模型中。第一个为商店中付账排队的模拟。第二个为随机产生的证券管理程序。

第二章是递归降序词法分析，这是我个人的爱好，许多年前，我有许多关于这类问题的编程工作。

第十一章讨论Turbo Pascal至Turbo C的转换，最后第十二章讨论效率、移值和调试问题。

本书有二个附录：附录A讨论Turbo C的内存模式选择。附录B综述了Turbo C语言。

本书中有许多有用而且有趣的函数和程序。如果你喜欢，可以采用，但可能厌烦于将它们敲入计算机中。我从某书中抄录程序时，也总是敲错而且花费很多时间去更正。由于这个原因，我提供了包含所在这本书中的函数和程序的源程序软盘售价为$ 24.95。若你想要这份软盘，请填写一张定单，随同付款一起邮寄来，地址见附页。若你急需，则请打电话至我的咨询办公室，通过电话预定，电话为（217）586-4021。

目　　录

第一章　排序和搜索

在计算机领域中，排序和搜索可以说是最基础，分析最透彻的算法，这些算法实际应用于所有的数据库系统，编译器，解释器及操作系统中．本章介绍排序和搜索的基础．因为数据排序通常可使数据搜索更方便、快速，所以首先讨论排序问题．

一．排序

排序是把一组相关信息（数据）组成具有递增顺序或递减顺序的过程．特别地，给出一个 N 个元素的排序表 I，则有

$$I_1 <= I_2 <= \ldots <= I_N.$$

虽然 Turbo C 提供标准的 qsort ( ) 函数作为标准库的组成部分．对排序的研究和掌握仍很重要．有三个道理，第一，类似 qsort ( ) 的通用函数不能用于所有情况；第二，qsort ( ) 是带参数的，为了对各种类形的数据进行操作，需要开销额外的时间进行类型转换，增加了运行时间．第三，快速排序算法（用于 qsort ( )）虽然在一般情况下是相当好的，但在某些特殊情况下并非最佳的排序方法．

两类基本的排序操作是数组（在内存或随机存取磁盘文件中）排序和顺序磁盘文件排序．本章主要集中于第一类，因为它是大多数程序员最关心的，同时也讨论顺序文件排序的一般方法．

数组排序和顺序文件排序的主要区别在于数组的每个元素都是直接可访问的．也就是说，任何时侯，任意元素都可与其他元素相比较或交换．而对一个顺序文件，任何一段时间，只有一部分元素是可直接存取的．由于这个差别，两者排序技术有很大区别．

通常，当信息（例如，一个邮件表）被排序时，只有部分信息用作排序关键字，该关键字用于比较，但在交换时，整个数据结构需进行调动，例如，在一个邮件表中，只有 ZIP 码域被用作关键字，但整个地址要是排序的．为简便起见，我们仅研究字符数组的排序，随后，再学习如何使这些方法适用于任何类型的数据结构．

一）排序算法分类

用于数组排序有三种通用方法：

- 交换法
- 选择法
- 插入法

为理解这三种方法设想有一盒卡片，用交换法排序，就是将卡片摊在桌子上，然后交换错序的卡，直到所有的卡片排好序．

用选择法排序卡片，就是将卡片盒放于桌上，选择最小值的卡，从盒中取出，握在手里，再从余下的卡中取出最小值的卡，放在手中卡的最后，因为总是众余下的卡中取出最小值的卡．当此过程结束，手中的卡片即为排好序的．

用插入法排序是将卡握在手中，一次取一张，当你将卡插入盒中时，总是插在正确的位置，当你手中的卡片插光时，盒中的卡将是排好序的．

二）判决排序算法

三种排序法中的每一种都有许多不同的算法，每种算法都有其价值，但一般都基于以下几个标准来评价其好坏：

- 在平均的情况下，它排序数据是否快？
- 在最好、最坏情况下它是否快？
- 它是否具有自然属性？

· 它是否重排关键字相同的元素?

一个特定算法排序数据的速度是至关重要的，数组排序的速度直接与比较的次数和交换的次数（交换次数占时更长）有直接关系。所谓比较即为一个数组元素与另一元素相比，所谓交换即为数组中两个元素互换位置。在本章后面的部份你将看到有些排序需要指数级时间，有些则只需对数级时间。

在最佳、最坏情况下的运行时间也很重要，如果你估计将在这两个情况之一频繁计数，则其运行时间很重要。有时，一个排序算法在平均情况下是良好的，但在最差情况却可能是最糟糕的。

一个排序是有自然特性的是指，当表是有序时，排序工作量最少，当表少量有序时，排序工作量多些，当表是倒序时工作量最大。排序工作量多少是基于排序所必需的比较和移动次数。

为了理解重排具有相同关键字的元素的重要性，设想有一个按主关键字和副关键字排序的数据库，例如，一个以 ZIP 码为其主关键字，ZIP 码字段中最后名为副关键字的邮件表，当一个新地址加到表中，并且对表重新排序时，你并不希望副关键字被重排，为了保证这点排序操作不交换相同值的主关键字。

下面几节分析各类排序算法中具有代表性的几种，并判断其有效性，然后研究重要的排序方法。

1、冒泡排序法

最为出名（其实也是最差）的排序法要数冒泡法。它的出名是因为其易记的名称和简单性。通过下面的分析也很容易看出，它是最差的排序法之一。

冒泡排序法使用交换法排序，冒泡算法中的一般概念是重复比较，并在必要时，交换相邻元素。该名起源于与水桶中的冒泡过程类似，每个水泡都在寻找自己的位置。

下面是冒泡排序的简单形式:

```
void bubble(item,count)  / * bubble sort * /
char * item;
int count;
{
  register int a,b;
  register char t;
  for(a = 1; a < count; ++a)
    for(b = count-1; b > a; --b){
      if(item[b-1] > item[b]){
        / * exchange elements * /
        t = item[b-1];
        item[b-1] = item[b];
        item[b] = t;
      }
    }
}
```

其中 item 是一个指向排序字符数组的指针．count 是数组中的元素个数。

冒泡算法是由两个循环实现的，假设在数组中有 count 个元素，外循环使数组扫描 count-1 次，以保证在最坏情况下，当函数终结时，每个元素都在其相应的位置。内循环执行实际的比较和交换。稍加优化的冒泡排序法在无交换时就终结了。但这需每次通过内循环时加一次比较。

冒泡排序法可用来将字符数组排列为升序。例如，用下面的小程序对一个从键盘输入的字符串进行排序。

```
void bubble( );
```

```
/ * sort drives * /
main( ) / * sort a string from the keyboard * /
{
  char s[80];
  printf("enter a string:");
  gets(s);
  bubble(s, strlen(s));
  printf("the sorted string is: %s\n", s);
}
```

为说明冒泡排序如何工作，下面是用冒泡法来排序 dcab 的过程（从最后两个元素比起）：

起　始	d c a b
pass 1	a d c b
pass 2	a b d c
pass 3	a b c d

分析任何一种排序时，都必须决定有多少个比较和交换。在最好、平均、最差情况下，冒泡排序的比较次数是固定的。因为两个 for 循环需循环固定的次数，不论表在起始是否排序。这意味着冒泡排序总是执行 $(n^2-n)/2$ 次比较。其中，n 为排序元素的个数，该公式基于这样的事实：外循环执行 n−1 次，内循环执行 n/2 次，两者相乘而得此公式。

在最佳状况下是 0 次交换——对已排好的表而言，平均情况下为 3 $(n^2-n)/4$ 次，最坏请况下为 3 $(n^2-n)/2$ 次，其推导超出了本书范围，但可看到，当表趋向无序时，无序的元素个数接近于比较的次数（注意，在冒泡排序中每个无序元素有三次交换），冒泡排序被称作 n^2 的算法，因为它的执行时间是元素个数的平方关系。冒泡排序在大数目元素的情况下是不理想的。因为，执行时间直接与比较和交换次数有关。

例如，若忽视了交换错位元素所需的时间，每次比较需 0.001 秒。则排序 10 个元素需 0.05 秒，排序 100 个元素需 5 秒,排序 1000 个元素需 500 秒，100,000 个元素（象一电话簿大小）需 5,000,000 秒，即 1400 小时，约两个月的连续排序！图 1−1 表示了执行时间随数组大小增加而增大的关系。

```
void  shaker(item, count) / * shaker sort, an improved bubble sort * /
char  * item;
int  count;
{
  register int  a, b, c, d;
  char  t;
  c = 1;
  b = count−1;  d = count−1;
  do {
    for(a = d; a > = c; −−a) {
      if(item[a−1] > item[a]) {
        t = item[a−1];
        item[a−1] = item[a];
        item[a] = t;
        b = a;
      }
    }
    c = b+1;
```

```
    for(a=c; a<d+1; ++a) {
      if(item[a-1]>item[a]) {
        t=item[a-1];
        item[a-1]=item[a];
        item[a]=t;
        b=a;
      }
    }
    d=b-1;
  } while {c<=d);
}
```

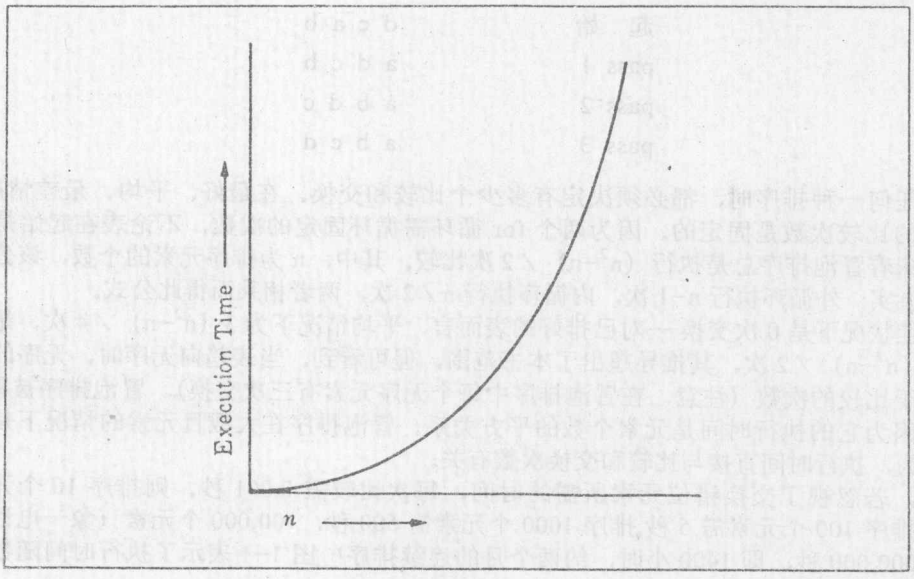


图 1-1

对冒泡排序稍加改进，可以加快其运行速度。例如，冒泡排序有一特点：在最后的错序元素（如在 decab 中 a 元素）在第一遍执行时就排到了正确的位置上，而在开始位置的错序元素（如上例中 d 元素），将很晚才放到其正确位置上，这就意味着对冒泡算法可作改进。

代替总是从一个方向读取数组，改为后继的操作从相反的方向读取数组，错位最大的元素将很快地排列到其正确位置上。下面列出的是根据这一思想设计的名为"shaker"的一种冒泡排序版本。

虽然 shaker 排序法可改进冒泡排序的速度，但它仍需 n^2 次级数。因为比较次数不变，交换次数只减少了相当小的常数。

2、选择排序法：

选择排序法是选取最小值元素，并与第一元素交换，然后在余下的 n-1 个元素中取最小值与第二元素交换。如此继续直到最后两个元素。例如：用选择法对 bdac 数组排序，每次操作后的情况如下：

初　始	b d a c
pass 1	a d b c
pass 2	a b d c
pass 3	a b c d

基本选择排序法程序如下:

```
void  select(item,count)  / * selection sort * /
char  * item;
int  count;
{
  register int  a, b, c;
  char  t;
  for(a = 0; a < count-1; ++a) {
    c = a;
    t = item[a];
    for(b = a+1; b < count; ++b) {
      if(item[b] < t) {
        c = b;
        t = item[b];
      }
    }
    item[c] = item[a];
    item[a] = t;
  }
}
```

不幸的是类似冒泡排序，选择法外循环执行 n-1 次比较，内循环执行 n / 2 次比较，即选择排序法需 (n^2-n) / 2 次比较，这对大数量元素排序而言就显得很慢．交换次数在最好情况下是 3(n-1)次，在最坏的情况下，为 $n^2/4+3$ (n+1) 次．

在最佳的情况，即表是有序的，则只需移动 n-1 个元素．每次移动需三次交换（赋值）．在最坏情况下，比较次数相同．在平均情况下，虽然超出了本书研究范围，它有 $n(\ln^n+y)$次．其中 y 是 Euler 常数 (0.577216)．由此可见，虽然冒泡排序和选择排序的比较次数相同，但平均的交换次数、选择法小得多．

3、插入排序法:

插入排序法是最后一种简单算法，插入法先对数组中两个元素进行排序，然后算法把第三个元素插到前两排好序元素组成的表中，正确位置，再将第四个元素插到前三个元素的表中，此过程继续，直到所有元素均已排好序．例如，给出数组 dcab．每次插入排序的情况如下:

初　始	d c a b
pass 1	c d a b
pass 3	a c b d
pass 4	a b c d

插入排序法显示如下:

```
void  insert(item,count)  / * sorting by straight insertion * /
char  * item;
int  count;
{
  register int  a, b, c;
  char  t;
  for(a = 1; a < count; ++a) {
    t = item[a];
```

```
    b=a-1;
    while(b>=0; &&  t<item[b]) {
      item[b+1]=item[b];
      b--;
    }
    item[b+1]=t;
  }
}
```

与冒泡排序和选择排序不同，插入排序中的比较次数将与表的初始排序顺序有关，若表是有序的，则比较次数为 n-1 次，若表是乱序的，则比较次数为 $(n^2+n)/2-1$ 次，而平均比较次数为 $(n^2+n-2)/4$ 次．交换次数情况如下：

最佳情况下：　　2(n-1)

平均情况下：　　$(n^2+9n-10)/4$

最差情况下：　　$(n^2+3n-4)/2$

可见在最坏情况下的次数与冒泡排序、选择排序一样，而在平均情况稍好些．

插入排序有两优点：第一，自然性，当数组有序时，执行次数最少，这对几乎已排好序数组很有用．第二，保持关键字相同元素原来顺序，若表按两个关键字排序，则用插入排序法可保持按每个关键字排序的顺序．

虽然比较操作对某类数据相当好，但每当一个元素放于相应位置时数组需移动．这意味着移动次数将会相当可观的．然而插入排序法总的来说是比较自然的，对几乎排好序的表交换次数最少，对于完全乱序交换次数最多．

三）改进的排序法

上述所有算法都有执行时间为 n^2 的致命缺陷，对于大量数据进行排序将会变得很慢．事实上，在某些意义上是太慢以致不实用．每个计算机程序员都听说过"三整天排序"的故事．不幸的是，这样的故事常有发生．

排序时间过长常常是由算法引起的．一种悲观论点提出用汇编代码编写程序．虽然汇编代码可提高子程序的速度，但若其算法是差的，无论如何优化代码都不会提高排序速度．请注意：当运行子程序时间与 n^2 相关时，则提高编码速度或计算机速度，只有很少的改进．因为运行时间是以指数级增加的（图 1-1 上的曲线稍向右移动，但曲线形状未变）．可以确信，用 Turbo C 编写的速度不快，用汇编语言编写的也不会快．提高速度的办法，只好用更好的排序算法．

本节研究两个优秀的排序法．第一，希尔（谢尔）排序法；第二，快速排序算法．这些排序法都是相当快的．

1.希尔排序法（Shell Sort）

希尔排序法是以发明者 D.L.Shell 命名的.但是，从其名称上似乎也可看出些名堂．因为这种操作方法，很象海贝一个堆一个．

通用方法是由插值排序法衍生的．基于缩减增值的思想．图 1-2 给出了对数组 fdacbc 的 Shell 排序图解.

开始，对所有相隔三个位置的元素进行排序，然后，对所有相隔两个位置的元素进行排序，最后，对所有相邻的元素进行排序．

这种方法的效果以及它是否排序了整个数组看上去都不明显，然而这种方法的确是有效的．因为每个排序过程涉及的元素相当少或者元素已放在适当的位置．因此，每个过程都增强了数据的有序性．

间隔位置的确切序列可改变，唯一原则是最后增量必须为 1．例如，序列 9，5，3，2，1 也可工作．用于 shell 排序程序如下所示，避免用 2 的指数值序列，因为由于数学上复杂性，他们降低了排序算法的效率（但若用这个序列，排序仍可进行）．

```
void shell(item,count)  / * a shell sort * /
char  * item;
int  count;
{
  register int  i,j,k,s,w;
  int  x, a[5];
  a[0]=9; a[1]=5; a[2]=3; a[3]=2; a[4]=1;
  for(w=0; w<5; w++) {
    k=a[w]; s=-k;
    for(i=k; i<count; ++i) {
      x=item[i];
      j=i-k;
      if(s==0){
        s=-k;
        s++;
        item[s]=x;
      }
      while(x<item[j] &&  j>=0 && j<=count) {
        item[j+k]=item[j];
        j=j-k;
      }
      item[j+k]=x;
    }
  }
}
```

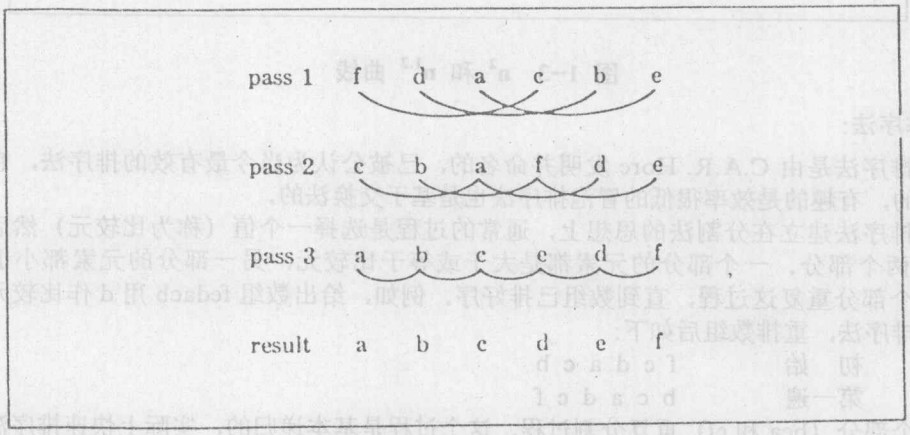


图 1-2　希尔排序法

你也许已注意到内层 while 循环有三个测试条件.

x<item[j]是排序过程的必要比较. j>=0 和<=count 测试用于避免排序超出数组 item 的边界，这些额外的检查在某种程度上降低了 shell 排序操作的效率. 版本稍有区别的 shell 排序法使用了称作标志的特殊数组元素. 这些元素不在排序数组中，仅标记具有特殊终止值或为最小可能值或为最大可能值. 此时，边界检查就不必要了. 但设立标志需预先知道数据，它限制了排序函数的通用性.

Shell 排序的分析涉及了一些非常困难的数学问题．这些问题已超出了本书范围．但对几个元素排序的执行时间可能为 $n^{1.2}$ 级数，这相对于以前所述的 n^2 级数的排序法是个可观的改进．为了说明这种改进的程序，图 1-3 作出了 n^2 和 $n^{1.2}$ 的曲线．不过，在决定使用 shell 排序法之前你应该知道快速排序法是更好的一种方法．

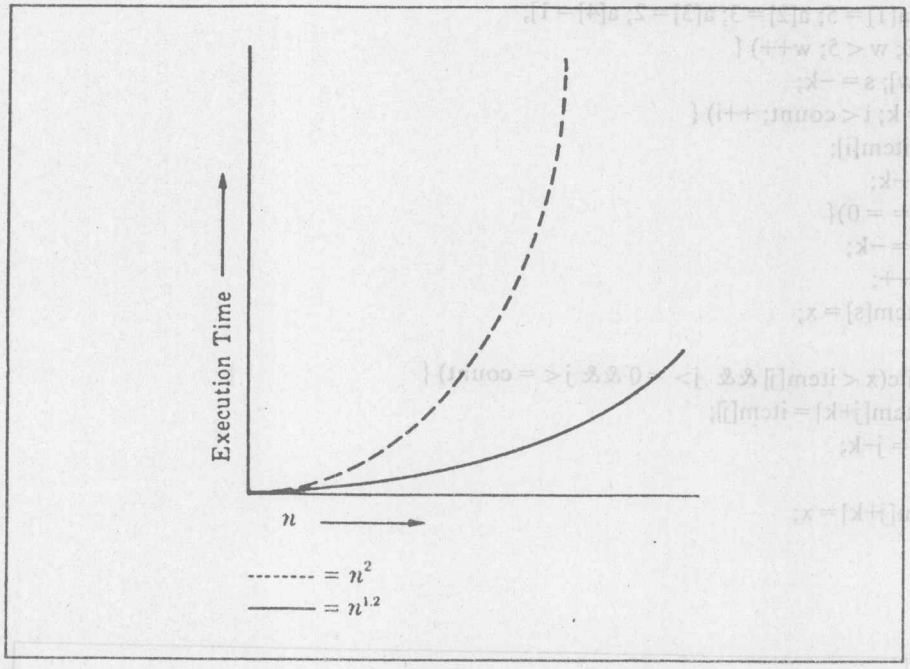


图 1-3 n^2 和 $n^{1.2}$ 曲线

2、快速排序法:

快速排序法是由 C.A.R. Hore 发明并命名的，已被公认为当今最有效的排序法，它是基于交换法的，有趣的是效率很低的冒泡排序法也是基于交换法的．

快速排序法建立在分割法的思想上，通常的过程是选择一个值（称为比较元）然后将数组划分成两个部分．一个部分的元素都是大于或等于比较元，另一部分的元素都小于比较元，对每个部分重复这过程，直到数组已排好序．例如，给出数组 fedacb 用 d 作比较元，第一遍快速排序法，重排数组后如下:

初 始　　f e d a c b
第一遍　　b c a d e f

对两个部分（bca 和 ef）重复分割过程，这个过程是基本递归的，实际上快速排序法中最清晰的运行部分就是递归算法．

中间比较元的选择可由两种方法实现：随机选取或数组中一小组数据值的平均值．为优化排序法若选取的元素正好处于数组的中央那是最理想的，这对大多数数据集来说并非容易之事．但即使在最坏的情况下——选择的值在端，快速排序法仍可很好的执行．

下面的快速排序法中选取数组中间位置的元素作为比较元．虽然这并非总是很好选择，但排序法可正确实现．

```
void quick(item,count) / * quicksort set up * /
char * item;
int count;
{
```

```
  qs(item,0,count-1);
}
void  qs(item,left,right)  / * quicksort * /
char   * item;
int  left, right;
{
  register int  i, j;
  char  x, y;
  i=left;  j=right;
  do {
    while(item[i]<x && i<right)  i++;
    while(x<item[j] && j>left)  j--;
    if(i<=j) {
      y=item[i];
      item[i]=item[j];
      item[j]=t;
      i++;  j--;
    }
  } while(i<=j);
  if{left<j)  qs(item,left,j);
  if(i<right)  qs(item,i,right);
}
```

在此排序法中，函数 quick（ ）调用排序主体函数 qs（ ），item 和 count 作为通用的接口。但这并非必需的，因为 qs（ ）可用三个参量直接调用。

快速排序法中比较次数和交换次数的推导超出了本书范围，你只需知道如下结果，快速排序法的比较次数为 n $\log^n$，交换次数约为 n $\log^n$ / 6．这比至今我们讨论过的任何一种排序法都好．

等式 $N=a^x$ 可写作 $X=log_a N$

这意味着，若有 100 个元素进行排序，快速排序法将需求 100×2 即 200 次比较，因为 $log\ 100=2$．与冒泡排序法需 900 次相比，这个数字是相当好的．

应该看到，快速排序法也有不好的方面，若比较元的值是每次划分的最大值，则快速算法将变得很慢，运行时间为 n^2 级，但一般情况下，这种情况不会发生．

我们必须仔细选择一种决定比较元值的选择方法．通常，这个值是排序中的实际数据所决定的．在大型邮件表中，通常按 ZIP 邮码排序．此时，选取是容易的．因为，邮码分布是相对平均的，用简单的代数函数即可决定一个较适合的比较元，但在某些数据库中，排序关键字与数值关系密切（许多具有相同值），以致随机选择常常是最佳的（有效方法）一通用而较有效的方法是取三个样本元素，然后取其中间值元素．

四）排序法的选择

通常，由于快速排序法的速度快而被选择，但只有少量数据排序时（小于 100 个），由快速排序法中的递归调用产生的额外开销有损于高级算法的效率，在这种特殊情况下，用简单的排序法（冒泡排序法）将会更快．

五）其他数据结构的排序

至今，我们只是对字符数组进行排序，这使得每个排序程序得以简化．显然，任何数据类型数组的排序，可以通过简单地交换排序函数中参量和变量的数据类型实现．然而，常常

需要对一些复杂的数据类型如串、结构等进行排序，为了使算法适合于其他数据结构的排序，需对比较段或交换段进行转换，但算法本身保持不变。

因为快速排序法是至今最佳的通用程序，它将用于下面的例子中。对于以前所述的任何一种排序法都可使用相同的技术进行类似的处理。

1、串组排序

串排序的最简单方法是建立一个指向这些串的字符指针数组，这为你提供简单的索引，并保持基本快速排序算法不变。下面列出的快速串排序法将处理指向排序串的字符指针数组，排序法重排指向串组的指针，而非内存中的实际串组，该排序法把串组排成字典顺序。

```
void  quick__string(item,count)  / * quick sort for string setup * /
char  * item[ ];
int  count;
{
  qs__string(item,0,count-1);
}
void  qs__string(item,left,right)  / * quick sort for string * /
char  * item[ ];
int  left, right;
{
  register int  i, j;
  char  * x, * y;
  i = left;  j = right;
  x = item[(left+right) / 2];
  do {
    while(strcmp(item[i],x) < 0 && i < right)  i++;
    while(strcmp(item[j],x) > 0 && j > left)  j--;
    if(i < = j) {
      y = item[i];
      item[i] = item[j];
      item[j] = t;
      i++;  j--;
    }
  } while(i < = j);
  if(left < j)  qs__string(item,left,j);
  if(i < right)  qs__string(item,i,right);
}
```

基本程序中的比较步骤由于使用了 strcmp() 函数而用所改变。strcmp() 函数的功能为：若第一个字串按字典规则小于第二字串，则 strcmp 返回负值。若两字串相等则返回 0。若第一字串大于第二字串则返回正值。子程序的交换部份未改变，因为仅仅是指针交换而非实际串交换。若要交换实际字串，则需调用函数 strcpy()。

使用 strcmp() 将减慢排序速度。因为第一，涉及化费时间的函数调用；第二，strcmp() 函数本身执行时需若干次比较方能决定两字串的关系。若速度是至关重要的，则程序中可直接写入 strcmp() 的代码，以取代对 strcmp() 函数调用，但串比较是不可避免的，因为这是由涉及的任务所决定的。

2、结构排序

大多数需要排序的应用程序需要排序一组信息，邮件表就是一个极好的例子，因为姓名、街道、城市、州名和 ZIP 码都是连接在一起的，当排序这些数据的混合单元对，使用了

一个排序关键字，但整个结构需进行交换。为了说明其工作情况，首先需建立一个结构，对于邮件表这个例子，一个方便的结构是：

```
struc address {
        char  name[40];
        char  street[40];
        char  city[20];
        char  state[3];
        char  zip[10];
};
```

其中州为三字符长，ZIP 是十字符长，因为一个串组总是需要比最大数组长度大一，以便存储空终止符。

因为用结构数组安排一个邮件表是合适的，就这个例子，结构类型 address 数组的排序子程序，列出如下：

```
void  quick__struct(item,count)  / * quick sort for structures setup * /
struct  address  item[ ];
int  count;
{
  qs__struct(item,0,count-1);
}

void  qs__struct(item,left,right)  / * quick sort for structures * /
struct  address  item[ ];
int  left, right;
{
  register int  i, j;
  char  * x, * y;
  i = left;  j = right;
  x = item[(left+right) / 2].zip;
  do {
    while(strcmp(item[i].zip,x) < 0 && i < right)  i++;
    while(strcmp(item[j].zip,x) > 0 && j > left)  j--;
    if(i < = j) {
      swap__all__field(item,i,j);
      i++;  j--;
    }
  } while(i < = j);
  if(left < j)  qs__struct(item,left,j);
  if(i < right)  qs__struct(item,i,right);
}
```

请注意，比较步骤和交换步骤都需修改，因为太多区域需交换，为此建立一个独立的函数 swap__al__fields () 来做这件事，创建的 swap__all__field () 需与欲排序的结构的特性相一致。

六）磁盘文件的排序

磁盘文件有两种：顺序文件和随机文件，若磁盘文件很小，它们可读入内存，并且，前述的数组排序程序对它们将是最有效的，但许多磁盘文件都很大以致不能简单地在内存排序，因而需要一些特殊的技术。大多数微机数据库应用软件都使用随机存取文件，因此首先在此进行讨论。