HOPE

Turbo系列算法
—实用编程指南

本书介绍排序技术、搜索技术、数学算法、字串处理、符号串处理、单向链表处理、双向链表和循环链表、栈和队列、二叉树、AVL一树等。

刘京
唐克 编译
晓兰

中国科学院希望高级电脑技术公司

Turbo系列算法
——实用编程指南

刘京
谐克 编译
晓兰

本书介绍排序技术、搜索技术、数学算法、字串处理、符号串处理、单向链表处理、双向链表和循环链表、栈和队列、二叉树、AVL-树等。

中国科学院希望高级电脑技术公司
一九九一年四月

编者的话

随着 Turbo 系列语言在我国的不断普及和应用，软件设计人员越来越感觉到有必要把它们用于一些经典问题的求解过程和数据结构的表达方法上，《Turbo 系列算法——实用编程指南》正是为这一目的而编写的。

本书首先概括了计算机科学中最基本的问题，包括排序、搜索、数学函数、串处理、表处理、椎栈和队列、二叉树、AVL 树的表达等基本知识，接着介绍了对这些基本数据结构的操作算法，最后分别用 Turbo BASIC、Turbo C、Turbo Pascal、Turbo Prolog 等加以实现。

无论是计算机程序设计的初学者还是高级程序员，都能从本书的算法分析和编程实现中获益匪浅。

本书虽经我们认真编译，但由于时间仓促，加上我们水平有限，书中难免会有些缺点和错误，望广大读者不吝赐教，以备再版时修订。

本书在出版过程中，得到了中国科学院希望高级电脑技术公司资料部秦人华经理、杨淑新老师的大力帮助和支持，在此表示衷心的感谢。

编译者

一九九一年三月

目　录

序　言

一旦你学会了怎样编写程序语句及如何放置逗号及分号，下一步该是什么？除非有一些稳固的数据结构和算法来支持，否则很多代码都是无甚价值的。试图解决一个实际的程序设计问题而没有正确的数据结构和算法就好比要想建造一所房子而没有锤子及钉子。

本书的目的就是为你提供编写实用程序所需的从排序、搜索到 AVL 树的数据表示这些主要算法的参考指南。由于我们用了每种 Borland 的 Turbo 语言来实现这些算法，所以你就能够就这些语言的相似及差异加以比较。

本书对象

如果你用过任何 Borland 的 Turbo 语言，而又想进一步发展你的程序设计技巧，就能从本书中获益非浅。需要将程序从一种语言转换成另一种语言的专业程序员会发现本书特别有价值。

如果你象大多数严肃的 Turbo 语言程序员，那么所需的就不只是另一个用户指南，而应是能帮助你迅速地找到关键信息的一本完整的参考书。为满足此需要，本书为每一种语言都提供了许多有用的实例，并为你提供了大量的程序设计建议与技巧。

基本要求

为使用本书，你需要一种或多种 Turbo 语言的编译器。你还需要有一 IBMPC，XT，AT，PS/2 或兼容的计算机系统。我们所给出的所有代码例子都是使用语言编译器的最新版本，包括最近的 Turbo BASIC，Turbo C 2.0，Turbo Pascal 5.0，及 Turbo Prolog 2.0。

本书内容

本书的编排使你能很方便而迅速地找到所需的信息。每节都以对所给出算法的概述而开始，接着是对以各种 Turbo 语言来实现算法的例程的描述。较复杂的算法被放在较简单的算法之后，主要的算法及数据结构在十个章节中给出。

第一章“排序技术”给出了用 Turbo 每种语言编写的各种主要排序例程。该章先以常用的插入排序开始，然后讨论 shell 排序，快速排序，基数排序及堆排序。

第二章“搜索技术”探索了用于搜索的各种主要算法。该章将数组和表作为主要的数据结构以实现用于启发式搜索，二分搜索，索引搜索及杂凑搜索这样的搜索算法。

第三章“数学算法”给出了从超越函数到积分的各种实用数学算法。

第四章“字串处理”描述了怎样以各种 Turbo 语言来设计与实现各种有用的串处理例程。在本章中，串以字序列的形式加以表示及处理。

第五章“符号串处理”扩充了第四章中的内容。这里串以通用的符号串加以表示与处理。

第六章“单向链表处理”包括了对单向链表所能进行的所有主要算法。我们探讨了从利用 Turbo BASIC 中的数组，到利用 Turbo C 和 Pascal 中的指针，记录及结构来表示单链表的各种方法。

第七章“双向链表处理及循环链表处理”通过引入双向链表及循环表来继续讨论表处理算法。这些表结构能够有助于改进第六章中给出的算法性能。

第八章“栈与队列”描述了怎样实现及处理基本栈及队列数据结构。所给出的算法既包括了基于数组的，也包括了基于链接表的栈及队列。

第九章“二叉树”给出了表示与处理二叉树的技术。你会学到怎样利用诸如递归及动态存储分配这样的程序设计技术来实现二叉树处理算法。

第十章“AVL 树”探讨了二叉树的一种很有用的变异 AVL 树。该章描述了对元素的加、删及搜索算法，还描述了用于平衡树的算法。

本书用法

本书既包括了用语言进行程序设计的技巧，也包括了熟练运用各种算法的技巧。在使用本书时，你应当尝试尽可能多的例子。由于每种语言都是高度交互式的，因此你能通过运行例子及对其加以修改来很快地学到程序设计的精妙之处。

表示约定

每当引入一般的程序设计语句时，我们都使用下列格式：

```
<函数名>（[<变元 1,><变元 2,> ... ]）
{
 [<声明>]
 <语句集>
 return（返回值）；
}
```

诸如 return 和{}这样的语言的保留字及符号都以正常形式而放置，必须由程序员提供的标识符被放于尖括号中。符号[]用来说明可选项。

第一章　排序技术

§1.1　介绍

考虑用 Turbo 语言实现算法的技术时，得着重于排序方法。因为，对从数据库到操作系统大多数应用来说，排序操作是很关键的。你将会发现在这里讨论的许多算法都会在你的许多程序中得到应用。这些年来，计算机领域的科学家们已探讨了许多复杂的排序算法。遗憾的是我们不可能在这里都把它们加以讨论。我们仅限于讨论那些比较普遍和实用的算法。本章以较为简单的算法，如插入排序等开始，然后再讨论较复杂的算法。

下面是本章要讨论的算法

- 插入排序
- Shell-Metzner 排序
- 快速排序
- 基数排序
- 堆排序

对于每一个排序方法，我们将以实例给出一个完整的、全面的描述，以帮助你了解它们是如何工作的，在每一个算法之后，还给出用四种 Turbo 语言写的对排序子程序的调用，以及在本章末尾给出一组每一种 Turbo 语言的源程序。你可以在程序中直接使用本书所提供的排序子程序，也可以对它们作一些修改以适于你的程序。我们建议你比较排序实现语言以了解排序算法是如何编码的。

§1.2　排序概述

若没有数据的排序，我们就不可能完成数据库、空白表格程序以及操作系统这些应用系统，幸运的是，排序是一个易于理解的概念，要执行一项任务，在计算机中程序需要内部数据如数组，外部数据，如文件。在大多数情况下，要用的数据必须以特定的顺序安排，这种技术就称为排序（sort）。

一般来说，数据以两种顺序进行排序：升序（递增）或降序（递减）。对于升序，下列的式子定义了一组数据中每一个元素是如何排列的：

e1 < = e2 < = ... < = en

这里的 e1，e2 等代表被排序的数据元素。另一方面，对于降序，下列的式子定义了一组数据中每一个元素是如何排列的：

e1 > = e2 > = ... > = en

我们在这里用 Turbo BASIC 、C 和 Pascal 所给出的排序算法，在大多数情况下是针对串组的操作，这是因为串通常涉及到排序问题。Turbo Prolog 的排序算法把表作为主要的数据结构。而对于其它类型数据结构的排序与它们相似，而且只需作很少的修改。

你可以修改本章中的任何排序子程序，使它们适于非串的数据类型。

§1.3　排序算法

§ 1.3.1 插入排序

插入排序是一个最简单的排序算法之一，它可以用任何程序设计语言加以实现。顾名思义，插入排序即为通过把表中的数据元素插入到适当的位置来进行排序。该算法基本上包括如下三个步骤：

(1) 将表中的头两个元素按排序顺序排列（升序或降序）。

(2) 把下一个（第三个）插入到其对应于已排序元素的排序位置。

(3) 对于表中的每一个元素重复第(2)步。即把第四个元素插入到适当位置，然后是第五个元素，等等。

要搞清楚该排序算法是如何工作的，让我们来看一个例子。考虑一个未经排序的州名缩写表，它有如下的初始顺序：

WA VA MI MA AL AZ CA OR NY NJ

我们将把该表按升序排序。第一个元素，将被放在一个位置直到有另一个元素来代替它。该部分排序表如下：

WA| VA MI MA AL AZ CA OR NY NJ

这里的竖杠把已排序的子表与未排序的子表分开。把第二个元素与已排序的子表中的每一个元素（这里是 WA）进行比较。结果使得 VA 置于第一个位置，WA 置于第二个位置。现在该表变成：

VA WA| MI MA AL AZ CA OR NY NJ

选择第三个元素并继续作比较，结果使 MI 移至第一个位置，使 VA 和 WA 往后移一个位置。选择 MA，重复上面过程，结果使之成为第一个元素并把已排序子表的各元素往右移一个位置。该表变成：

MA MI VA WA| AL AZ CA OR NY NJ

重复上述过程直至表中的所有元素都被考虑过。完全被排序的表为：

AL AZ CA MA MI NJ NY OR VA WA

插入排序的特点是，对于一张已排序表，当增加一个新的元素时，不需要重新排序该表便可以使之仍是一张排好序的表。

§ 1.3.1.1 插入排序的实现

用 Turbo BASIC、C 和 Pascal 实现的排序算法彼此是很相似的（参见源程序 1.1，1.2 和 1.3）。每个程序都使用一个嵌套的 WHILE 循环对串组进行升序排序。第一个 WHILE 循环重复到串组中所有的元素都被测试过。下面是 Turbo Pascal 的循环初始状态：

```
WHILE (i<=NumData) DO BEGIN
  ...
```

这里，变量 NumData 为表中元素的个数。

另一方面，内部的 WHILE 循环完成排序的各步。只要下面两个条件为真，下面的循环将不断重复：

```
WHILE (j<i) AND nomatels DO BEGIN
```

这里变量 i 为被排序元素在表中的当前位置，j 作为一个计数器，它的值从 1 增到被排序元素的当前位置。在内部 WHILE 循环开始之前，j 的值被赋为 1。循环中的每次重复，该变量都增加 1。该循环一直重复到下面的两个条件有一个满足：

(1) A[i]<A[j]，即找到一个其值大于被排序元素的元素。

(2) A[i]=A[j]，即找到一个其值等于被排序元素的元素。

若 A[i]<A[j]条件满足，如图 1.1 所示，该算法将把元素 A[i]放到 A[j]所在的位置，并且将 A[j]到 A[i-1]的所有元素往右移一个位置，如图 1.2 所示。完成此功能的程序代码如下：

```
tempo := A[i];
FOR k  := i-1 DOWNTO j TO j DO
  A[k+1] := A[k];
  A[j] := tempo;
```

注意在重新调整位置之前需保存 A[i]的值。这些简单的操作仅为使该表保持已排序状态。

在 Turbo Prolog 中需要不同的策略实现插入排序算法，这是因为 Turbo Prolog 不支持串组。数据被放在一张表中，该表采用递归方式进行排序。我们不采用对表从头到尾进行排序的方式，而是使用递归子句，直到到达表尾，然后从表尾到表头对每一个元素进行排序和插入。参阅源程序 1.4，可以看到需要两条子句。第一条为：insertsort，负责控制递归；第二条为：insert，执行排序和插入。

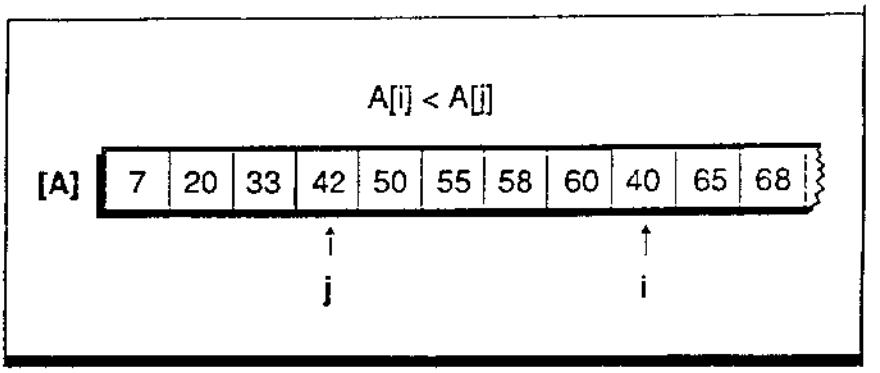

图 1.1　比较两个数组元素

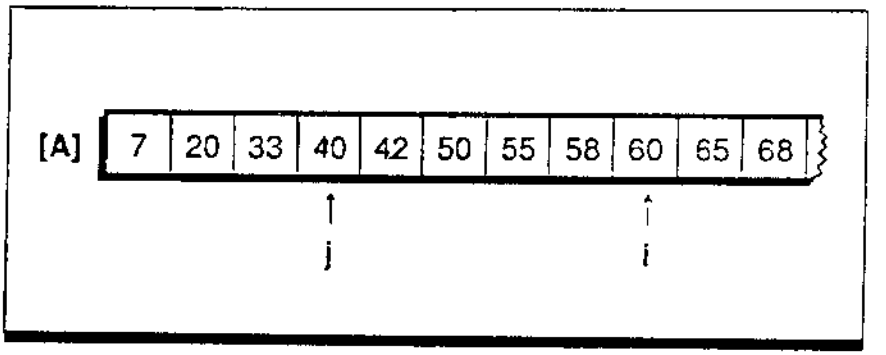

图 1.2　移动数组元素

§ 1.3.2　Shell_Metzner 排序

Shell_Metzner 排序，通常称为 shell 排序法，是简单插入排序的扩展。不是仅仅比较和交换相邻的元素，shell 排序法还比较和交换相距较远的元素。由于这一特性，该排序方法对于那些较长的数据表的排序更为有效。该排序技术就仿佛象是把海贝一个一个地迭放

起来似的。Shell 排序算法包括如下的基本步骤：

(1) 把待排序表分为长度相同的两部分；

(2) 把第一个表中的各个元素与第二个元素表中相对应的元素进行比较和交换。即，将第一个表中的元素 1 与第二个表中的元素 1 进行比较，将第一个表中的元素 2 与第二个表中的元素 2 进行比较，等等。

(3) 为进一步进行排序，较近的元素之间进行比较和排序。

(4) 重复第(3)被完全排序为止。每重复第(3)步时，两比较元素之间的距离都要缩短。

正如前面所述，该算法比插入分入算法复杂。我们在这里同样使用前面用来说明插入排序方法的州名缩写表。

WA	VA	MI	MA	AL	AZ	CA	OR	NY	NJ

由于本表含有 10 个元素，因而一开始进行比较的元素之间相距 5 个元素。把表分为两个子表：

WA	VA	MI	MA	AL	AZ	CA	OR	NY	NJ
1	2	3	4	5	1	2	3	4	5

把具有相同序号的元素进行比较，必要时，还要进行交换。因此，元素 WA 和 AZ 相比较，并进行交换，得到：

AZ	VA	MI	MA	AL	WA	CA	OR	NY	NJ
1	2	3	4	5	1	2	3	4	5

类似地，VA 和 CA 进行比较并交换。对于子表中的其它元素重复该过程，然后我们得到下列串排序表：

AZ	CA	MI	MA	AL	WA	VA	OR	NY	NJ
1	2	3	4	5	1	2	3	4	5

尽管该表没有完全排好序，但是剩下的工作不多了。用算法中的第 1 步和第 2 步把相隔 5 个元素的元素排好序。下一步便是减少一倍两比较元素之间的距离。用整数除，把原距离除以 2(5/2)。新的子表为：

AZ	CA	MI	MA	AL	WA	VA	OR	NY	NJ
1	2	1	2	1	2	1	2	1	2

现在，元素 AZ 与 MI 比较但不交换。下一对元素的比较是 CA 和 MA，它们按正确次序排列，不需交换，而后是 MI 和 AL 比较并交换，因为 MI 大于 AL。象这样比较一遍后（最后为 OR 和 NJ 的比较），得到的表为：

AZ	CA	AL	MA	MI	OR	NY	NJ	VA	WA
1	2	1	2	1	2	1	2	1	2

可以看到，该表还没有按顺序排列好。因而，还要进行第二遍比较。首先 AZ 和 AL 进行比较并交换。类似地，OR 和 NJ 两元素要进行交换，这一遍比较完之后，表将为：

AL	CA	AZ	MA	MI	NJ	NY	OR	VA	WA
1	2	1	2	1	2	1	2	1	2

由于在第二遍中有元素的交换，因而还需要作第三遍比较。第三遍比较之后，相距两个元素的元素之间已按正确次序排列。下一步，算法将两比较元素之间的距离从 2 缩小到 1。你可以看出，Shell 排序方法是把表从外向内进行排序的。这就是说这一步将要进行相邻元素之间的比较。这一步中，一开始比较 AL 和 CA，然后比较 CA 和 AZ，并将 CA 和 AZ 进行交换，对表中的其它元素重复这一过程。象这样一遍一遍地重复直至在某一遍中不再有元素需要交换为此。排序好的表为：

AL AZ CA MA MI NJ NY OR VA WA

由于 Shell 算法的复杂性以及所含步骤的数量，你一开始可能会认为它并不是十分有效的。但实际上它是很有效的，因这每一遍所需要的是最少的比较次数和交换次数。由于数据排列顺序的不断改进，算法比较的元素距离越近时，交换的次数也将减少。

记住，当你使用 Shell 排序算法时，你可以修改比较元素的距离系列。在我们的例子中，距离系列为 5-2-1，然而，我们还可以采用 5-3-2-1。唯一的要求是最后一步的比较距离须为 1。可以尝试不同的系列为你要排序的数据找出最有效的系列。

§1.3.2.1 Shell 排序的实现

象插入排序法一样，Turbo BASIC、C 和 Pascal 的实现过程是相似的。仍使用嵌套式循环来完成搜索。然而，Shell 排序法比插入排序方法稍微复杂一些，因而需要三层循环嵌套。外层循环重复待排序表长的次数；中间层循环在每次重复中把表分成两半；最里层循环完成实际的元素比较以及在必要时进行元素交换。完成这些操作的程序代码与插入排序方法的程序代码非常相似。这里用 Turbo Pascal 描述如下：

```
IF A[i] < A[j] THEN BEGIN
   { swap members }
   tempo := A[j];
   A[j] := tempo;
   inorder := FALSE;
END;
```

在 Turbo Prolog，shell 排序算法的效率不高，这是因为把表分开的速度不快。在一个象 Turbo Pascal 这样的过程化设计语言中，只要我们能够确定表的中间元素的序号便可以很容易地到达该中间元素。而在 Prolog 中，一张表只能被顺序地进行访问。

另一个可以考虑的算法是用得很广的冒泡排序法。该算法引入了一个表分类的元素交换技术。其基本策略是比较两个相邻的元素，并当这两个元素次序不对时把它们进行交换。在 Turbo Prolog，该算法较容易实现。在源程序 1.4 中，我们使用谓词 append 从表中选择元素。主要的谓词 bubblisort，是递归的并不断调用 appende 直到表被排序好为止。

§1.3.3 快速排序

就象 Shell 排序法是插入排序法的改进一样，快速排序是 Shell 排序法的改进。许多设计员认为快速排序法是所有排序算法中最好的算法之一，因为对于一个算法来说，它满足两个重要的条件：速度快，并在大多数语言中易于实现。

快速排序算法引入了一个灵巧的 divide-and-conquer 策略，从而使之效率得以提高。快速排序算法的基本策略是从表中选择一个中间的分隔元素。该分隔元素把表分成两个子表，一个子表中的所有元素小于该分隔元素，而另一个子表中的所有元素等于或大于该分隔元素。要完成整个表的排序，这种分隔方法将在每个子表中重复。即算法将从每个子表中选出一个分隔元素并把各子表分成更小的子表。从这个描述中你可以看出，该算法具有递归的特性。

下面是快速排序法进行排序所需步骤的概括：

(1) 从表中选择一个分隔元素并把该表分成两个子表。（把所有小于分隔元素的元素放在第一张表中，把所有其它元素放在第二张表中）。

(2) 从每一个子表中选择一个分隔元素并把每个子表分成更小的子表。

(3) 重复第(2)步直到每个子表只含一个或零个元素。

(4) 用递归方法，把子表中的元素联接起来以形成一张排序表。

下面我们来看一个例子，看看快速排序法是如何工作的。我们从下面这张未排序的表开始：

WA VA MI CA MA OR NY AZ AL NJ

首先，选择分隔元素。我们选择 MA，它是一个中间元素或者是一个真正的中间元素。然后把表分成两个子表。左边的子表包含所有那些小于 MA 的元素，右边的子表包含所有那些大于 MA 的元素，如下所示：

CA AZ AL <-- 左子表

MA <-- 中间元素

WA VA MI OR NY NJ <-- 右子表

对于每个子表递归地重复上述过程。例如，选择 OR 作为中间元素，把右子表分开，得到的左子表有三个元素，而右子表只有两个元素，如：

MI NY NJ <-- 左子表

OR <-- 中间元素

WA VA <-- 右子表

重复上述过程直至每一个子表只含一个或零个元素为止。此时，算法将建立已排序子表并最后生成一张完整的排序表。

在快速排序法中，递归扮演了非常重要的角色。但是要记住，该算法可以不用递归来实现。实际上，在源程序 1.1，1.2 和 1.3 中，我们提供了用 Turbo BASIC 、C 和 Pascal 书写的采用递归和不采用递归来实现的快速排序算法，以便于你比较这两种方法。

当你使用快速排序算法时，可以尝试不同的分隔元素选择方法。在我们给出的例子及源程序中，我们把中间元素选作分隔元素。另一种方法是随机地选择一个值或者指定一个值作为分隔元素，而不是用中间元素作分隔元素。

§ 1.3.3.1 快速排序的实现

在 BASIC、C 和 Pascal 中，用子程序 RecQuickSort 和 Sort 来实现递归的快速排序算法，RecQuickSort 子程序是调用 Sort 执行递归排序算法的高层子程序。每次调用 Sort，便选择一个中间元素并把表或子表分成两半。Sort 根据被分成的子表之一递归地调用自身，使排序不断地进行。为记住正在被排序的子表的左、右位置，使用两个下标指针 i 和 j。排序实际是在 Repeat 循环中完成的，该循环一直重复执行到左下标指针超过了右下标指

针时才结束。

非递归的实现要稍微复杂些。它需要使用一个类似于堆栈的结构以记录各个子表。该结构如图 1.3 所示。注意，它包含有两列：左和右。它们存放子表的左下标和右下标。每当一个子表被分时，新的左下标和右下标将被推入栈中。要记住一个表是通过把它分成越来越小的子表来实现排序的。当一个最内层的子表排序结束时，我们则需要返回到较外一层的子表。因而，把每个子表的下标存放在栈中，使我们能够容易地找到每个子表的位置。

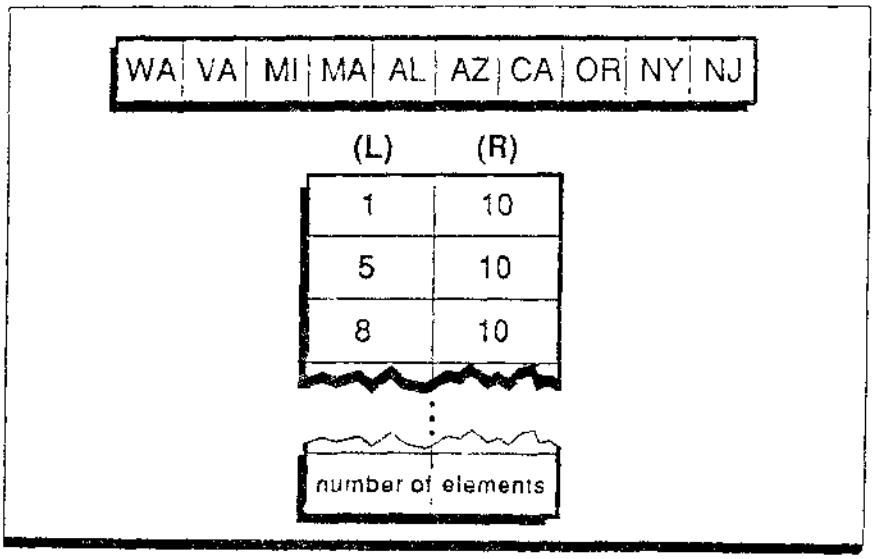

图 1.3　实现快速排序的数据结构

§ 1.3.4　基数排序

基数排序方法对数排序很有用。实际上，它特别适用于二进制数排序。它还可以用于非数数据。如串，只要首先把该数据转换成数的表达形式。

基数排序方法不同于其它的排序方法，因为它在进行数据的排序处理时，把一个数据元素分成几个部分，然后通过比较这些部分而不是比较整个数据元素来进行数据排序。例如，若按常规的排序技术排序如下元素：

Saul　Mary　John

我们可能会比较每一个完整的串以决定适当的排序次序。从另一个方面，若我们打算用其基数方法排序如下数：

100　500　201　309

则我们可能要将每一个数分成 3 个数字的系列，并分别比较每一个数字。即我们可能要将 100 中的 1 与 500 中的 5 进行比较，将 500 中的 5 与 201 中的 2 进行比较，等等。

让我们来看一个例子，看看基数方法是如何排序一组数的，考虑如下 3 数字数表：

132　213　321　231　111　232　112　221　123

第一步是为每个数的最低有效位建立 bins。通常情况下需要 10 个 bin；但这里我们只需要 3 个 bin，因为没有一个数含有大于 3 的数字。把数安排在 1s，2s 和 3s 中，如：

1s-->321　231　111　221