排序（Sorting）是计算机程序设计中的一个重要问题。几十年来，排序问题获得广泛关注，并成为计算机科学中研究得最多的问题之一。

8.1 问题定义

排序的功能是将一个数据元素（或记录）的任意序列重新排列成一个按关键字有序的序列。假设序列 $a_0,a_1,a_2,\ldots,a_{n-1}$ 的关键字的值依次为 $k_0,k_1,k_2,\ldots,k_{n-1}$，对该序列的排序过程就是确定一组下标排列 $p[0],p[1],\ldots,p[n-1]\ (0\le p[i]\le n-1, 0\le i\le n-1)$，使得当 $i\ne j$ 时，$p[i]\ne p[j]$，且相应的序列的关键字的值满足如下的关系：

$k_{p[0]}\le k_{p[1]}\le \ldots \le k_{p[n-1]}$（升序），或者 $k_{p[0]}\ge k_{p[1]}\ge \ldots \ge k_{p[n-1]}$（降序）。

如果没有特殊指明，本章所述的排序方法总是按关键字的值将序列排为升序。

若序列中关键字的值相等的结点经过某种排序方法进行排序之后，仍能保持它们在排序前的相对顺序；也就是说，在排序前，如果 $i<j \wedge k_i=k_j$，且排序后，a_i 仍在 a_j 之前，则称这种排序方法是**稳定的**；否则，称这种排序方法是**不稳定的**。

特别说明的是：在一些教材及资料中，排序也常被称为**分类**。

根据对内存的使用情况，排序方法分为内部排序和外部排序两种。内部排序是指排序期间全部结点都存储于内存，并在内存中调整等待排序结点的存放位置；外部排序是指排序期间大部分结点存储于外存中，排序过程借助内存，调整那些存放在外存、等待排序的结点的存放值。

根据排序实现的手段，排序方法分为基于“比较-交换”的排序和基于“分配”的排序。其中，基于“比较-交换”的排序包括：插入排序、冒泡排序、选择排序、快速排序、希尔排序、堆排序等；基于“分配”的排序包括：基数排序等。

根据实现的难易程度，排序方法又可分为基本排序方法和高级排序方法。通常，我们认为插入排序、冒泡排序和选择排序为基本排序方法。

8.2 基本排序方法

8.2.1 插入排序

插入排序（Insertion Sort）是一种基本的排序方法。本小节讨论的方法为直接插入排序（为描述方便，以下的插入排序方法除特殊说明均为直接插入排序），其基本思想是将一个记录插入到已排好顺序的序列中，形成一个新的、记录数增 1 的有序序列。

假设待排序的 n 个结点的序列为 $a_0,a_1,a_2,\ldots,a_{n-1}$，我们依次对 $i=1,2,\ldots,n-1$ 分别执行下面的插入步骤：

假设 $a_0,a_1,\ldots,a_{i-1}$ 已排序，故有 $a_0\le a_1\le\ldots\le a_{i-1}$。首先，让 $t=a_i$，然后将 t 依次与 a_{i-1}，a_{i-2}，… 进行比较，将比 t 大的结点依次右移一个位置，直到发现某个 $j\ (0\le j\le i-1)$，使得 $a_j\le t$，则令 $a_{j+1}=t$；如果这样的 a_j 不存在，那么在比较过程中，a_{i-1}，$a_{i-2},\ldots,a_0$ 都依次后移一个位置，此时令 $a_0=t$。

经过执行下面的函数实现上述的排序算法：

程序 8-1 插入排序方法

```
template <class Item>
```

```
void InsertionSort(Item a[], int l, int r)
{
  int i,j;  Item t;
  for (i=l+1; i<=r; ++i){
    for (j=i-1,t=a[i]; j>=0&&t<a[j]; j--)
      a[j+1] = a[j];
    a[j+1] = t;
  }
}
```

我们用图 8-1 的示例来说明插入排序的执行过程。

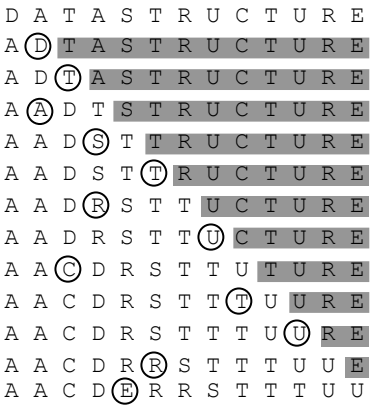

图 8-1　插入排序过程

现在，我们分析一下插入排序的比较次数和记录移动次数。当待排序的结点序列 $a_0,a_1,a_2,\ldots,a_{n-1}$ 按照关键字非递减的顺序排列时，对于自变量为 j 的循环，每执行一次，只要进行一次结点比较，故整个排序过程只进行(n-1)次比较，且不需要移动记录；当待排序结点序列 $a_0,a_1,a_2,\ldots,a_{n-1}$ 按关键字非递增的顺序排列时，对于 j 循环，每执行一次，需要进行 j 次比较，故整个排序过程需进行的比较次数和记录移动次数分别为

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

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

从上述分析可知，插入排序的运行时间和待排序记录的顺序密切相关。若记录出现在待排序序列中的概率相同，则可取上述最好情况和最坏情况的平均情况。此时，比较次数和记录移动次数约为 $n^2/4$。因此，插入排序的时间复杂度为 $O(n^2)$。

需要强调的是：**插入排序方法是稳定的，该方法适用于结点个数较少的场合。**

8.2.2　冒泡排序

冒泡排序（Bubble Sort）是另一种基本的排序方法，其基本思想是依次比较相邻的两个元素的顺序，如果顺序不对，则将两者交换，重复这样的操作直到整个序列被排好序。假设待排序的序列为 $a_0,a_1,a_2,\ldots,a_{n-1}$，起始时排序范围是从 a_0 至 a_{n-1}。冒泡排序是在当前排序范围内，自右向左对相邻的两个结点进行比较，让键值较大的结点向右移，让键值较小的结点向左移。当自左向右比较当前排序范围后，键值最小的元素被移动到序列的 a_0 位

置，故 a_0 无需再参加下一次比较，下一次比较的范围为 a_1 至 a_{n-1}。循环上述的比较过程，直到比较范围为 a_{n-2} 至 a_{n-1} 完成。在整个排序过程中，执行了 n-1 次比较过程。经过执行程序 8-2 所示的函数实现上述的排序算法：

程序 8-2　冒泡排序方法

```
template <class Item>
void swap(Item& A, Item& B)
{
  Item temp=A;  A=B;   B=temp;
}
template <class Item>
void BubbleSort(Item a[], int l, int r)
{
  for (int i=l; i<r; ++i)
      for (int j=r; j>i; --j)
          if (a[j-1]>a[j])
              swap(a[j-1],a[j]);
}
```

我们用图 8-2 的示例来说明冒泡排序的执行过程。

```
D A T A S T R U C T U R E
(A) D A T C S T R U E T U R
A (A) D C T E S T R U R T U
A A (C) D E T R S T R U T U
A A C (D) E R T R S T T U U
A A C D (E) R R T S T T U U
A A C D E (R) R S T T T U U
A A C D E R (R) S T T T U U
A A C D E R R (S) T T T U U
A A C D E R R S (T) T T U U
A A C D E R R S T (T) T U U
A A C D E R R S T T (T) U U
A A C D E R R S T T T (U) U
```

图 8-2　冒泡排序过程

假设待排序的结点有 n 个，那么冒泡排序最多执行 n-1 遍。第一遍最多执行 n-1 次比较和 n-1 次交换；第二遍最多执行 n-2 次比较和 n-2 次交换；…；第 n-1 遍最多执行一次比较和一次交换。因此，整个排序过程最多需要的比较和交换次数为：

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

需要强调的是：**冒泡排序是稳定的，因为相邻结点键值相同不会发生数据的交换。**

8.2.3　选择排序

选择排序（Selection Sort）也是一种基本的排序方法，其基本思想是：首先选出序列中键值最小的项，将它与序列的第一个项交换位置；然后选出键值次小的项，将其与序列的第二个项交换位置；…；直到整个序列完成排序。

假设待排序的序列为 $a_0, a_1, a_2, \ldots, a_{n-1}$，我们依次对 i=0,1,…,n-2 分别执行如下的选择步骤：在 $a_i, a_{i+1}, \ldots, a_{n-1}$ 中选择一个键值最小的项 a_k，然后将 a_k 与 a_i 交换。

在上述步骤执行完成后，序列 $a_0, a_1, a_2, \ldots, a_{n-1}$ 完成排序。

经过执行程序 8-3 所示的函数实现上述的排序算法：

程序 8-3　选择排序方法

```
template <class Item>
void SelectionSort(Item a[], int l, int r)
{
  int i,j,min;
  for (i=l; i<r; ++i){
    for (min = i, j=i+1; j<=r; ++j)
      if (a[j]<a[min])
            min = j;
      swap(a[i],a[min]);
  }
}
```

我们用图 8-3 的示例来说明选择排序的执行过程。

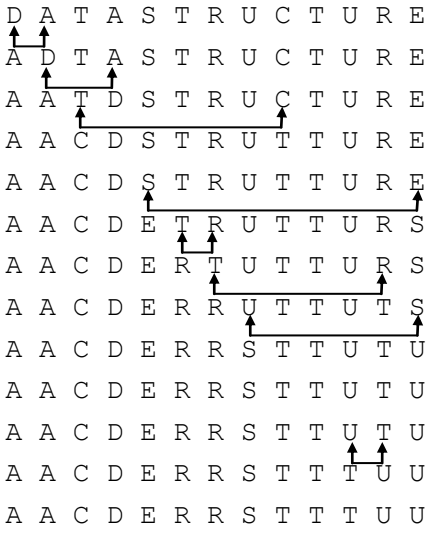

图 8-3　选择排序过程

类似地，我们可得到最坏情况下选择排序的比较次数和数据交换次数，分别为：n(n-1)/2 和(n-1)。

选择排序有两个特点：

(1) **它的运行时间和待排序中记录的顺序关系很小**，因为每次从序列中选出最小键值的项需要依次比较序列中剩余所有项的键值。

(2) **需要很少的数据交换次数**。因此，**选择排序对数据项较大、键较小的序列进行排序时效果较好。**

需要强调的是：**选择排序方法是不稳定的**。例如，对于序列 2,2,1，经过选择排序后，两个整数 2 的位置颠倒了。**选择排序也适用于数据项较少的场合。**

8.3　内部排序方法的比较

经典的排序方法包括：插入排序、冒泡排序、选择排序、归并排序、快速排序、堆排序、

希尔排序和基数排序。

在上述方法中，我们通常认为前三种排序方法为基本排序方法。它们具有的共同特点是实现方法比较简单，只需要一个辅助单元即可，时间复杂性相对较高（$O(n^2)$）。

归并排序、快速排序和堆排序是三种平均复杂性为$O(n*logn)$的高效排序方法。

快速排序是一种高效、不稳定的排序方法。在某些情况下（如待排序序列已经接近排序完成时），其时间复杂性会变为$O(n^2)$，并占用$O(n)$的存储空间。

堆排序是对插入排序的改进，利用堆结构快速实现堆内最大元素的查找，其排序过程不需要额外的存储开销，且排序所需时间比较稳定。其排序是不稳定的。

归并排序是一种稳定的排序方法，且排序所需时间与待排序序列的顺序无关。该方法也常用于外排序过程。

希尔排序是一种介于基本排序方法和高效排序方法之间的方法，其时间复杂性依赖于增量序列的选取，还有待研究。

基数排序是一类特殊的排序方法，并具有线性的时间复杂性。对于整数和字符串排序，基数排序具有很高的效率。由于基数排序实现时需要很多内部循环，且关键字的抽取取决于具体数据类型，使得其实际效率和通用性不如其它基于“比较-交换”的排序方法。

在表 8-1 中，我们给出了各种排序方法的比较（由于希尔排序的复杂性还有待研究，在表 8-1 中并未给出）。

表 8-1　排序方法比较

排序方法	平均时间	最坏时间	辅助空间	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
归并排序	$O(n*logn)$	$O(n*logn)$	$O(n)$	稳定
快速排序	$O(n*logn)$	$O(n^2)$	$O(logn)$	不稳定
堆排序	$O(n*logn)$	$O(n*logn)$	$O(1)$	不稳定
基数排序	$O(d(n+rd))$	$O(d(n+rd))$	$O(rd)$	稳定