排序算法

1、**冒泡排序**
2、快速排序
3、直接插入排序
4、希尔排序

一.交换类排序法
所谓交换排序法是指借助数据元素之间互相交换进行排序的方法。冒泡排序与快速排序法都属于交换类排序方法。

1、**冒泡排序**（BubbleSort）
冒泡排序的基本概念：
依次比较相邻的两个数，将小数放在前面，大数放在后面。即在第一趟：首先比较第1个和第2个数，将小数放前，大数放后。然后比较第2个数和第3个数，将小数放前，大数放后，如此继续，直至比较最后两个数，将小数放前，大数放后。至此第一趟结束，将最大的数放到了最后。在第二趟：仍从第一对数开始比较（因为可能由于第2个数和第3个数的交换，使得第1个数不再小于第2个数），将小数放前，大数放后，一直比较到倒数第二个数（倒数第一的位置上已经是最大的），第二趟结束，在倒数第二的位置上得到一个新的最大数（其实在整个数列中是第二大的数）。如此下去，重复以上过程，直至最终完成排序。由于在排序过程中总是小数往前放，大数往后放，相当于气泡往上升，所以称作冒泡排序。
实现：
外循环变量设为i，内循环变量设为j。假如有10个数需要进行排序，则外循环重复9次，内循环依次重复9，8，...，1次。每次进行比较的两个元素都是与内循环j有关的，它们可以分别用a[j]和a[j+1]标识，i的值依次为1,2,...,9，对于每一个i,j的值依次为1,2,...10-i。
图示：

C语言实现：

```
1void Bublesort(int a[],int n)
2{
3int i,j,k;
4for(j=0;j<n;j++)  /* 气泡法要排序n次*/ 5    {
6for(i=0;i<n-j;i++)  /* 值比较大的元素沉下去后，只把剩下的元素中的最大值再沉下去就可以啦 */ 7        {
8if(a[i]>a[i+1])  /* 把值比较大的元素沉到底 */ 9            {
10                k=a[i];
11                a[i]=a[i+1];
12                a[i+1]=k;
13            }
14        }
15    }
16}
```

性能分析：

若记录序列的初始状态为"正序"，则冒泡排序过程只需进行一趟排序，在排序过程中只需进行n-1次比较，且不移动记录；反之，若记录序列的初始状态为"逆序"，则需进行n(n-1)/2次比较和记录移动。因此冒泡排序总的时间复杂度为O(n*n)。

冒泡排序是非常容易理解和实现，以从小到大排序举例：

设数组长度为N。

1．比较相邻的前后二个数据，如果前面数据大于后面的数据，就将二个数据交换。

2．这样对数组的第0个数据到N-1个数据进行一次遍历后，最大的一个数据就“沉”到数组第N-1个位置。

3．N=N-1，如果N不为0就重复前面二步，否则排序完成。

按照定义很容易写出代码：

```
//冒泡排序1
void BubbleSort1(int a[], int n)
{
    int i, j;

    for (i = 0; i < n; i++)
        for (j = 1; j < n - i; j++)
            if (a[j - 1] > a[j])
                Swap(a[j - 1], a[j]);
}
```

下面对其进行优化，设置一个标志，如果这一趟发生了交换，则为true，否则为false。明显如果有一趟没有发生交换，说明排序已经完成。

```
//冒泡排序2
void BubbleSort2(int a[], int n)
{
    int j, k;
    boolflag;

    k = n;
    flag = true;
    while (flag)
    {
        flag = false;
        for (j = 1; j < k; j++)
            if (a[j - 1] > a[j])
            {
                Swap(a[j - 1], a[j]);
                flag = true;
            }
        k--;
    }
}
```

再做进一步的优化。如果有100个数的数组，仅前面10个无序，后面90个都已排好序且都大于前面10个数字，那么在第一趟遍历后，最后发生交换的位置必定小于10，且这个位置之后的数据必定已经有序了，记录下这位置，第二次只要从数组头部遍历到这个位置就可以了。

```
//冒泡排序3
void BubbleSort3(int a[], int n)
{
```

```
    int j, k;
    int flag;

    flag = n;
    while (flag > 0)
    {
        k = flag;
        flag = 0;
        for (j = 1; j < k; j++)
            if (a[j - 1] > a[j])
            {
                Swap(a[j - 1], a[j]);
                flag = j;
            }
    }
}
```

冒泡排序毕竟是一种效率低下的排序方法，在数据规模很小时，可以采用。数据规模比较大时，最好用其它排序方法。

2、快速排序（Quicksort）

基本思想：

快速排序是对冒泡排序的一种改进。由C. A. R. Hoare在1962年提出。它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

实现：

设要排序的数组是A[0]……A[N-1]，首先任意选取一个数据（通常选用第一个数据）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序。值得注意的是，快速排序不是一种稳定的排序算法，也就是说，多个相同的值的相对位置也许会在算法结束时产生变动。

一趟快速排序的算法是：

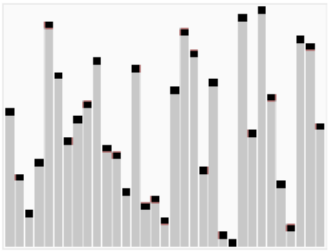
1）设置两个变量i、j，排序开始的时候：i=0，j=N-1；

2）以第一个数组元素作为关键数据，赋值给**key**，即 **key**=A[0]；

3）从j开始向前搜索（j -- ），找到第一个小于**key**的值A[j]，A[i]与A[j]交换；

4）从i开始向后搜索（i ++ ），找到第一个大于**key**的A[i]，A[i]与A[j]交换；

5）重复第3、4、5步，直到 I=J；（3,4步是在程序中没找到时候j=j-1，i=i+1，直至找到为止。找到并交换的时候i， j指针位置不变。另外当i=j这过程一定正好是i+或j-完成的最后令循环结束。）



举例说明：

如无序数组[6 2 4 1 5 9]

a),先把第一项[6]取出来,

用[6]依次与其余项进行比较,

如果比[6]小就放[6]前边,2 4 1 5都比[6]小,所以全部放到[6]前边

如果比[6]大就放[6]后边,9比[6]大,放到[6]后边,//6出列后大喝一声,比我小的站前边,比我大的站后边,行动吧!霸气十足~

一趟排完后变成下边这样:

排序前 6 2 4 1 5 9

排序后 2 4 1 5 6 9

b),对前半拉[2 4 1 5]继续进行快速排序
重复步骤a)后变成下边这样:
排序前 2 4 1 5
排序后 1 2 4 5
前半拉排序完成,总的排序也完成:
排序前:[6 2 4 1 5 9]
排序后:[1 2 4 5 6 9]
C语言实现 :

```
1int partition(int *data,int low,int high)
2 3     {
4int t = 0;
5 6      t = data[low];
7 8while(low < high)
910      { while(low < high && data[high] >= t)
1112       high--;
1314       data[low] = data[high];
1516while(low < high && data[low] <= t)
1718       low++;
1920       data[high] = data[low];
2122     }
2324       data[low] = t;
2526return low;
2728     }
2930void sort(int *data,int low,int high) //快排每趟进行时的枢轴要重新确定，由此进 //一步确定每个待排小记录的low及high的
值3132      { if(low >= high)
3334return ;
3536int pivotloc = 0;
3738       pivotloc = partition(data,low,high);
3940       sort(data,low,pivotloc-1);
4142       sort(data,pivotloc+1,high);
4344       }
```

性能分析
快速排序的时间主要耗费在划分操作上，对长度为k的区间进行划分，共需k-1次关键字的比较。
最坏情况是每次划分选取的基准都是当前无序区中关键字最小(或最大)的记录，划分的结果是基准左边的子区间为空(或右边的子区间为空)，而划分所得的另一个非空的子区间中记录数目，仅仅比划分前的无序区中记录个数减少一个。时间复杂度为O(n*n)
在最好情况下，每次划分所取的基准都是当前无序区的"中值"记录，划分的结果是基准的左、右两个无序子区间的长度大致相等。总的关键字比较次数：O(nlgn)
尽管快速排序的最坏时间为O(n2)，但就平均性能而言，它是基于关键字比较的内部排序算法中速度最快者，快速排序亦因此而得名。它的平均时间复杂度为O(nlgn)。

二、插入类排序
插入排序(Insertion Sort)的基本思想是：每次将一个待排序的记录，按其关键字大小插入到前面已经排好序的子文件中的适当位置，直到全部记录插入完成为止。
插入排序一般意义上有两种：直接插入排序和希尔排序，下面分别介绍。
3、直接插入排序
基本思想 :
最基本的操作是将第i个记录插入到前面i-1个以排好序列的记录中。具体过程是：将第i个记录的关键字K依次与其前面的i-1个已经拍好序列的记录进行比较。将所有大于K的记录依次向后移动一个位置，直到遇到一个关键字小于或等于K的记录，此时它后面的位置必定为空，则将K插入。
图示 :

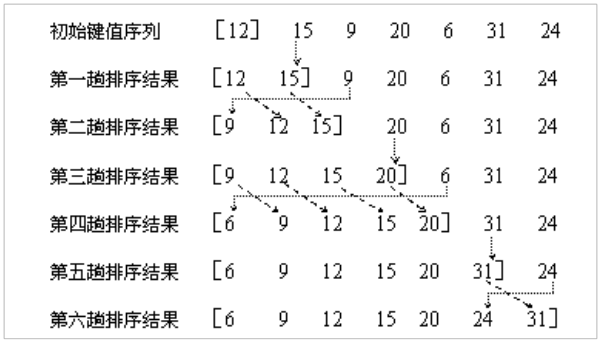
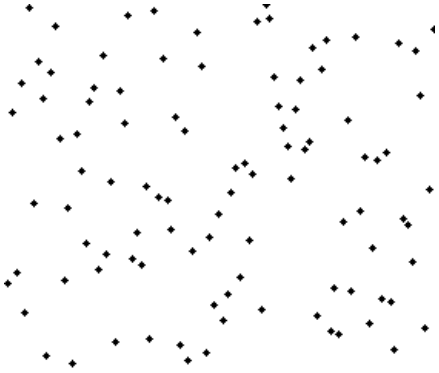


图 8-2 直接插入排序过程示例



C语言实现：

```
void InsertSort(int arr[], int n)
{
int temp;
int i,j;
for (int i = 1; i < arr.Length; i++)
      {
         int temp = arr[i];
         int j = i;
         while ((j > 0) && (arr[j - 1] > t))
         {
            arr[j] = arr[j - 1];//交换顺序
            --j;
         }
         arr[j] = temp;
      }

}
```

算法分析：

1．算法的时间性能分析

对于具有n个记录的文件，要进行n-1趟排序。

各种状态下的时间复杂度：

初始文件状态	正序	反序	无序(平均)
字比较次数	1	i+1	(i-2) /2
总关键字比较次数	n-1	(n+2)(n-1)/2	≈n2/4
第i趟记录移动次数	0	i+2	(i-2) /2
总的记录移动次数	0	(n-1)(n+4)/2	≈n2/4
时间复杂度	0 (n)	O (n2)	O (n2)

注意：

初始文件按关键字递增有序，简称"正序"。

初始文件按关键字递减有序，简称"反序"。

2．算法的空间复杂度分析

算法所需的辅助空间是一个监视哨，辅助空间复杂度S(n)=O(1)。是一个就地排序。

3．直接插入排序的稳定性

直接插入排序是稳定的排序方法。

直接插入排序法，针对少量的数据项排序，速度比较快，数据越大，这中方法的劣势也就越明显了。

改进方案：**折半插入排序（binary insertion sort）**

思路：折半插入排序（binary insertion sort）是对插入排序算法的一种改进，由于排序算法过程中，就是不断的依次将元素插入前面已排好序的序列中。由于前半部分为已排好序的数列，这样我们不用按顺序依次寻找插入点，可以采用折半查找的方法来加快寻找插入点的速度。

具体操作：在将一个新元素插入已排好序的数组的过程中，寻找插入点时，将待插入区域的首元素设置为a[low],末元素设置为a[high]，则轮比较时将待插入元素与a[m],其中m=(low+high)/2相比较,如果比参考元素小，则选择a[low]到a[m-1]为新的插入区域(即high=m-1)，否则选择a[m+1]到a[high]为新的插入区域（即low=m+1），如此直至low<=high不成立，即将此位置之后所有元素后移一位，并将新元素插入a[high+1]。

C语言实现：

```
void BInsertSort(int data[],int n)
 {
     int low,high,mid;
     int temp,i,j;
     for(i=1;i<n;i++)
     {
                  low=0;
                  temp=data[i];// 保存要插入的元素
                 high=i-1;
                while(low<=high) //折半查找到要插入的位置                {
               mid=(low+high)/2;
                 if(data[mid]>temp)
                   high=mid-1;
                 else
                  low=mid+1;
                }
            int j = i;
         while ((j > low) && (arr[j - 1] > t))
         {
            arr[j] = arr[j - 1];//交换顺序
            --j;
         }
         arr[low] = temp;

}

}
```

算法分析：折半插入排序算法是一种稳定的排序算法，比直接插入算法明显减少了关键字之间比较的次数，因此速度比直接插入排序算法快，但记录移动的次数没有变，所以折半插入排序算法的时间复杂度仍然为O(n^2)，与直接插入排序算法相同。附加空间O(1)。

4、希尔排序

希尔排序(Shell Sort)是插入排序的一种。是针对直接插入排序算法的改进。该方法又称缩小增量排序，因DL．Shell于1959年提出而得名。

基本思想：

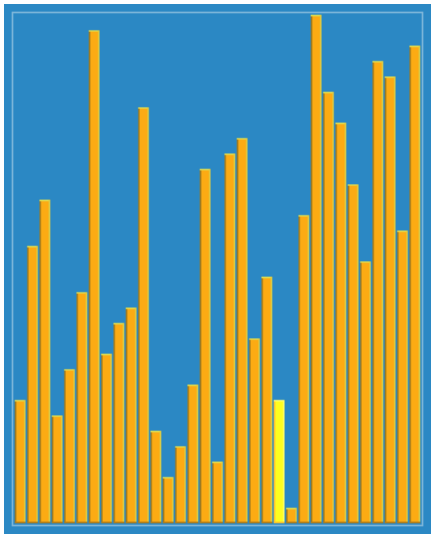
先取一个小于n的整数d1作为第一个增量，把文件的全部记录分成d1个组。所有距离为dl的倍数的记录放在同一个组中。先在各组内进行直接插人排序；然后，取第二个增量d2<d1重复上述的分组和排序，直至所取的增量dt=1(dt<dt-l<…<d2<d1)，即所有记录放在同一组中进行直接插入排序为止。

该方法实质上是一种分组插入方法。

举例阐述：

例如，假设有这样一组数[13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10]，如果我们以步长为5开始进行排序，我们可以通过将这

列表放在有5列的表中来更好地描述算法，这样他们就应该看起来是这样：
13 14 94 33 82
25 59 94 65 23
45 27 73 25 39
10
然后我们对每列进行排序：
10 14 73 25 23
13 27 94 33 39
25 59 94 65 82
45
将上述四行数字，依序接在一起时我们得到：[10 14 73 25 23 13 27 94 33 39 25 59 94 65 82 45].这时10已经移至正确位置了，然后再以3为步长进行排序：
10 14 73
25 23 13
27 94 33
39 25 59
94 65 82
45
排序之后变为：
10 14 13
25 23 33
27 25 59
39 65 73
45 94 82
94
最后以1步长进行排序（此时就是简单的插入排序了）。
图示：



C++代码实现：

```
1void shellsort(int *data, size_t size)
2 {
3for (int gap = size / 2; gap > 0; gap /= 2)
4for (int i = gap; i < size; ++i)
5       {
6 7int key = data[i];
8int j = 0;
9for( j = i -gap; j >= 0 && data[j] > key; j -=gap)
10            {
11               data[j+gap] = data[j];
12             }
13            data[j+gap] = key;
14        }
```

15 }

性能分析：

希尔排序是按照不同步长对元素进行插入排序，当刚开始元素很无序的时候，步长最大，所以插入排序的元素个数很少，速度很快；当元素基本有序了，步长很小，插入排序对于有序的序列效率很高。所以，希尔排序的时间复杂度会比o(n^2)好一些。由于多次插入排序，我们知道一次插入排序是稳定的，不会改变相同元素的相对顺序，但在不同的插入排序过程中，相同的元素可能在各自的插入排序中移动，最后其稳定性就会被打乱，所以shell排序是不稳定的。

最差时间复杂度	根据步长序列的不同而不同。已知最好的:
最优时间复杂度	O(n)
平均时间复杂度	根据步长序列的不同而不同。

$O(n \log^2 n)$