几种常用c语言的排序方法

冒泡排序

基本概念

冒泡排序（BubbleSort）的基本概念是：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。

用二重循环实现，外循环变量设为i，内循环变量设为j。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。

产生

在许多程序设计中，我们需要将一个数列进行排序，以方便统计，而冒泡排序一直由于其简洁的思想方法而倍受青睐。

排序过程

设想被排序的数组R［1..N］垂直竖立，将每个数据元素看作有重量的气泡，根据轻气泡不能在重气泡之下的原则，从下往上扫描数组R，凡扫描到违反本原则的轻气泡，就使其向上"漂浮"，如此反复进行，直至最后任何两个气泡都是轻者在上，重者在下为止。

算法示例

A[0]、A[1]、A[2]、A[3]、A[4]、A[5]、A[6]：

49 38 65 97 76 13 27

第一趟冒泡排序过程

38 49 65 97 76 13 27

38 49 65 97 76 13 27

38 49 65 97 76 13 27

38 49 65 76 97 13 27

38 49 65 76 13 97 27

38 49 65 76 13 27 97 – 这是第一趟冒泡排序完的结果

第二趟也是重复上面的过程，只不过不需要比较最后那个数97，因为它已经是最大的

38 49 65 13 27 76 97 – 这是结果

第三趟继续重复，但是不需要比较倒数2个数了

38 49 13 27 65 76 97

....

```
// 经典冒泡排序
void BubbleSort(int arr[], int n)
{
   int i = 0, j =0;
   for(i = 0; i < n; i++)
    for(j = 0; j < n - 1 - i; j++)
    {
        if(arr[j] > arr[j + 1])
        {
                arr[j] = arr[j] ^ arr[j+1];
                arr[j+1] = arr[j] ^ arr[j+1];
                arr[j] = arr[j] ^ arr[j+1];
        }
    }
```

选择排序

基本思想

n个记录的文件的直接选择排序可经过n-1趟直接选择排序得到有序结果：

①初始状态：无序区为R[1..n]，有序区为空。

②第1趟排序

在无序区R[1..n]中选出关键字最小的记录R[k]，将它与无序区的第1个记录R[1]交换，使R[1..1]和R[2..n]分别变为记录个数增加1个的新有序区和记录个数减少1个的新无序区。

……

③第i趟排序

第i趟排序开始时，当前有序区和无序区分别为R[1..i-1]和R(1≤i≤n-1)。该趟排序从当前无序区中选出关键字最小的记录 R[k]，将它与无序区的第1个记录R交换，使R[1..i]和R分别变为记录个数增加1个的新有序区和记录个数减少1个的新无序区。

这样，n个记录的文件的直接选择排序可经过n-1趟直接选择排序得到有序结果。

常见的选择排序细分为简单选择排序、树形选择排序（锦标赛排序）、堆排序。上述算法仅是简单选择排序的步骤。

排序过程

A[0] 、 A[1]、 A[2]、 A[3]、 A[4]、 A[5]、 A[6]：

49 38 65 97 76 13 27

第一趟排序后 13 ［38 65 97 76 49 27]

第二趟排序后 13 27 ［65 97 76 49 38]

第三趟排序后 13 27 38 [97 76 49 65]

第四趟排序后 13 27 38 49 [76 97 65]

第五趟排序后 13 27 38 49 65 [97 76]

第六趟排序后 13 27 38 49 65 76 [97]

最后排序结果 13 27 38 49 49 65 76 97

```
// 选择排序
void SelectSort(int arr[], int n)
{
   int i, j;
   int min;

   for(i = 0; i < n - 1; i++)
   {
       int index = 0;
       min = arr[i];
       for(j = i + 1; j < n; j++) //找出 i+1 - n 无序区的最小者与arr[i]交换
       {
           if(arr[j] < min)
           {
             min = arr[j];
             index = j;
           }
       }
       if(index != 0) //表明无序区有比arr[i]小的元素
       {
          arr[i] = arr[i]^arr[index];
          arr[index] = arr[i]^arr[index];
          arr[i] = arr[i]^arr[index];
       }
   }
```

快速排序算法

算法过程

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。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，即由后开始向前搜索（J=J-1），找到第一个小于key的值A[J]，并与A[I]交换；

4）从I开始向后搜索，即由前开始向后搜索（I=I+1），找到第一个大于key的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+完成的最后另循环结束)

例如：待排序的数组A的值分别是：（初始关键数据：X=49） 注意关键X永远不变，永远是和X进行比较，无论在什么位子，最后的目的就是把X放在中间，小的放前面大的放后面。

A[0] 、 A[1]、 A[2]、 A[3]、 A[4]、 A[5]、 A[6]：

49 38 65 97 76 13 27

进行第一次交换后： 27 38 65 97 76 13 49

(按照算法的第三步从后面开始找)

进行第二次交换后： 27 38 49 97 76 13 65

(按照算法的第四步从前面开始找>X的值，65>49,两者交换，此时：I=3)

进行第三次交换后： 27 38 13 97 76 49 65

(按照算法的第五步将又一次执行算法的第三步从后开始找

进行第四次交换后： 27 38 13 49 76 97 65

(按照算法的第四步从前面开始找大于X的值，97>49,两者交换，此时：I=4,J=6)

此时再执行第三步的时候就发现I=J，从而结束一趟快速排序，那么经过一趟快速排序之后的结果是：27 38 13 49 76 97 65，即所以大于49的数全部在49的后面，所以小于49的数全部在49的前面。

快速排序就是递归调用此过程——在以49为中点分割这个数据序列，分别对前面一部分和后面一部分进行类似的快速排序，从而完成全部数据序列的快速排序，最后把此数据序列变成一个有序的序列，根据这种思想对于上述数组A的快速排序的全过程如图6所示：

初始状态 {49 38 65 97 76 13 27}

进行一次快速排序之后划分为 {27 38 13} 49 {76 97 65}

分别对前后两部分进行快速排序 {27 38 13} 经第三步和第四步交换后变成 {13 27 38} 完成排序。

{76 97 65} 经第三步和第四步交换后变成 {65 76 97} 完成排序。

```
// 快速排序的递归实现
void QuickSort(int arr[], int n)
{
    if(n <= 1)
    return;

    int i =0 , j = n - 1;
    int key = arr[0];
    int index = 0;

    while(i < j)
    {
        // 从后向前搜索
        while(j > i && arr[j] > key)
        j--;
        if(j == i)
        break;
        else
        {
           //交换 a[j] a[i]
           arr[j] = arr[j] ^arr[i];
           arr[i] = arr[j] ^arr[i];
           arr[j] = arr[j] ^arr[i];
           index = j;
        }

        // 从前向后搜索
        while(i < j && arr[i] <key)
        i++;
```

```
            if(i == j)
            break;
            else
            {
               // 交换 a[i] a[j]
               arr[j] = arr[j] ^arr[i];
               arr[i] = arr[j] ^arr[i];
               arr[j] = arr[j] ^arr[i];
               index = i;
            }
      }
      QuickSort(arr, index);
      QuickSort(arr + index + 1, n - 1 - index);
}
```