第七讲 标准模版库

一、冒泡排序

二、选择排序

三、插入排序

四、快速排序

一、冒泡排序

冒泡排序：依次比较相邻的两个数，将小数放在前面，大数放在后面(构成从小到大排列)，或者将大叔放在前面，小数放在后面(构成从大到小排列)。

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， 再然后比较第3个数和第4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。

以下面5个无序的数据为例：

初始值： 65 27 59 64 58

第一次： 27 59 64 58 [65]

第二次： 27 59 58 [64 65]

第三次： 27 58 [59 64 65]

第四次： 27 [58 59 64 65]

【示例1-1】冒泡排序

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    const int N = 9;
    int array[N] = {65,27,59,64,58,30,49,2,19};

    //比较次数(因为是和当前位后面元素比较，所以不用比较到最后一位)
    for(int i=0;i<N-1;i++) {
        //通过下标依次和后面元素比较(比较到确定元素的前2位就能实现)
        for(int j = 0;j<N-1-i;j++) {
            if(array[j]<array[j+1]) {
                //换位
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }
    for(int m=0;m<N;m++) {
        printf("%4d",array[m]);
    }
    return 0;
}
```

程序运行结果如下：

65 64 59 58 49 30 27 19 2

二、选择排序

选择排序（Selection sort）是一种简单直观的排序算法。它的工作原理是每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，直到全部待排序的数据元素排完。

即在第一趟：从第一位开始和后面元素比较找出最小(大)的元素，和第一位互换位置，至此，能确定一个数的位置。

在第二趟：从第二位开始和后面元素比较找出最小(大)的元素，和第二位互换位置，至此，能确定一个新数的位置(就是此数列中第二小(大)的数排在了第二位)。

在第三趟：从第三位开始和后面元素比较找出最小(大)的元素，和第三位互换位置，至此，能确定一个新数的位置(就是此数列中第三小(大)的数排在了第三位)。

如此下去，重复以上过程，直至最终完成排序。

初始关键字　49 38 65 97 76 13 27 49

第一趟排序后 [13] 38 65 97 76 49 27 49

第二趟排序后 [13 27] 65 97 76 49 38 49

第三趟排序后 [13 27 38] 97 76 49 65 49

第四趟排序后 [13 27 38 49] 76 97 65 49

第五趟排序后 [13 27 38 49 49] 97 65 76

第六趟排序后 [13 27 38 49 49 65] 97 76

第七趟排序后 [13 27 38 49 49 65 76] 97

```
//数值交换
void DataSwap(int* data1, int* data2) {
    int temp = *data1;
    *data1 = *data2;
    *data2 = temp;
}
//进行排序
void SelectionSort(int* pDataArray, int iDataNum) {
    for (int i = 0; i < iDataNum - 1; i++) { //从第一个位置开始

        int index = i;//确定下标

        //从确定开始向后面元素进行比较
        for (int j = i + 1; j < iDataNum; j++) {
            if (pDataArray[j] < pDataArray[index]) {
                index = j;
            }
        }
        //最小(大)位置不为初始化位置，进行值互换
        if (index != i) {
            DataSwap(&pDataArray[index], &pDataArray[i]);
        }
    }
}
```

```
int main(int argc, const char * argv[]) {


    int array[] = {14,3,5,7,9,2,4,6,8,10};
    SelectionSort(array, 10);

    for(int i=0;i<10;i++) {
        printf("%3d",array[i]);
    }
    printf("\n");

    return 0;
}
```

程序运行结果如下：

2 3 4 5 6 7 8 9 10 14

三、插入排序

插入即表示将一个新的数据插入到一个有序数组中，并继续保持有序。

例如有一个长度为N的无序数组，进行N-1次的插入即能完成排序:

第一次，数组第1个数认为是有序的数组，将数组第二个元素插入仅有1个有序的数组中。

第二次，数组前两个元素组成有序的数组，将数组第三个元素插入由两个元素构成的有序数组中。

第三次，数组前三个元素组成有序的数组，将数组第四个元素插入由三个元素构成的有序数组中。

第N-1次，数组前N-1个元素组成有序的数组，将数组的第N个元素插入由N-1个元素构成的有序数组中，则完成了整个插入排序。

以下面5个无序的数据为例：

初始化：　　65　27　59　64　58

第1次插入：　[27　65] 59　64　58

第2次插入：[27　59　65]　64　58

第3次插入：　[27　59　64　65]　58

第4次插入：　[27　58　59　64　65]

```
void sort(int* pDataArray,int iDataNum) {
    //第一位确定有序数组，所以从第二位开始
    for (int i = 1; i < iDataNum; i++) {
        int j = 0;
        //取出数字后开始与有序数组进行比较
        while (j < i && pDataArray[j] <= pDataArray[i]) {
            j++;
        }
        //判断取出位置是否和比较位置相同
        if (j < i) {
            int k = i;
            int temp = pDataArray[i];
            while (k > j) {//位置向后移动一位
                pDataArray[k] = pDataArray[k-1];
                k--;
            }
            pDataArray[k] = temp;//有序数组插入元素
        }
    }
}
```

```
int main(int argc, const char * argv[]) {

    const int iDataNum = 6;
    int pDataArray[iDataNum] = {12,34,14,76,44,50};

    sort(pDataArray, iDataNum);

    for (int i = 1; i < iDataNum; i++) {
        printf("%4d",pDataArray[i]);
    }
    printf("\n");

    return 0;
}
```

程序运行结果如下：

14 34 44 50 76

四、快速排序

快速排序的算法是:

⑴ 设置两个变量I、J，排序开始的时候：I=0，J=N-1。

⑵ 以第一个数组元素作为关键数据，赋值给key，即 key=A[0]。

⑶ 从J开始向前搜索，即由后开始向前搜索（J=J-1即J--），找到第一个小于key的值A[j]，A[j]与A[i]交换。

⑷ 从I开始向后搜索，即由前开始向后搜索（I=I+1即I++），找到第一个大于key的A[i]，A[i]与A[j]交换。

⑸ 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)。

待排序的数组A的值分别是：（初始关键数据：key=49） 注意关键key永远不变，永远是和key进行比较，无论在什么位置，最后的目的就是把key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，此时:J=6)

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>key的值，65>49,两者交换，此时：I=2)

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key的值，97>49，两者交换，此时：I=3, J=5)

此时再执行第三步的时候就发现I=J=3，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，即所有大于key（49）的数全部在key（49）的后面，所有小于key（49）的数全部在key（49）的前面。

```
void quick_sort(int *x, int low, int high) {
    if(low>=high) {
        return;
    }
    int first=low;
    int last=high;
    int key=x[first];
    while(first<last) {
        while(first<last&&x[last]>=key)
            --last;
        x[first]=x[last];/*将比第一个小的移到低端*/
        while(first<last&&x[first]<=key)
            ++first;
        x[last]=x[first];/*将比第一个大的移到高端*/
    }
    x[first]=key;/*枢轴记录到位*/
    quick_sort(x,low,first-1);
    quick_sort(x,first+1,high);
}
```

```
int main(int argc, const char * argv[]) {

    int m[] = {1,2,4,9,3,5,6};
    int low = 0;
    int high = 6;
    quick_sort(m,low,high);

    for(int i = 0; i<7;i++) {
        printf("%2d",m[i]);
    }

    printf("\n");

    return 0;
}
```

55555555

程序运行结果如下：

1 2 3 4 5 6 9