**冒泡排序及其优化算法**

1、排序方法

     将被排序的记录数组R[1..n]垂直排列，每个记录R[i]看作是重量为R[i].key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"飘浮"。如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。

（1）初始

　    R[1..n]为无序区。

（2）第一趟扫描

　    从无序区底部向上依次比较相邻的两个气泡的重量，若发现轻者在下、重者在上，则交换二者的位置。即依次比较(R[n]，R[n-1])，(R[n-1]，R[n-2])，…，(R[2]，R[1])；对于每对气泡(R[j+1]，R[j])，若R[j+1].key<R[j].key，则交换R[j+1]和R[j]的内容。

    　第一趟扫描完毕时，"最轻"的气泡就飘浮到该区间的顶部，即关键字最小的记录被放在最高位置R[1]上。

（3）第二趟扫描

　    扫描R[2..n]。扫描完毕时，"次轻"的气泡飘浮到R[2]的位置上……

    　最后，经过n-1 趟扫描可得到有序区R[1..n]

  注意：

　    第i趟扫描时，R[1..i-1]和R[i..n]分别为当前的有序区和无序区。扫描仍是从无序区底部向上直至该区顶部。扫描完毕时，该区中最轻气泡飘浮到顶部位置R[i]上，结果是R[1..i]变为新的有序区。

3、冒泡排序算法

（1）分析

    　因为每一趟排序都使有序区增加了一个气泡，在经过n-1趟排序之后，有序区中就有n-1个气泡，所以整个冒泡排序过程至多需要进行n-1趟排序。

具体算法：

1. //冒泡排序
2. **void** BubbleSort1(**int**\* arr, **size\_t** size)
3. {
4. assert(arr);
5. **int** i = 0, j = 0;
7. **for** (i = 0; i < size - 1; i++)//一共要排序size-1次
8. {
9. **for** (j = 0; j < size - 1 - i; j++)//选出该趟排序的最大值往后移动
10. {
11. **if** (arr[j] > arr[j + 1])
12. {
13. **int** tmp = arr[j];
14. arr[j] = arr[j + 1];
15. arr[j + 1] = tmp;
16. }
17. }
18. }
20. }

（2）优化1（优化外层循环）：

    　若在某一趟排序中未发现气泡位置的交换，则说明待排序的无序区中所有气泡均满足轻者在上，重者在下的原则，因此，冒泡排序过程可在此趟排序后终止。为此，在下面给出的算法中，引入一个标签flag，在每趟排序开始前，先将其置为0。若排序过程中发生了交换，则将其置为1。各趟排序结束时检查flag，若未曾发生过交换则终止算法，不再进行下一趟排序。

具体算法：

1. //冒泡排序优化1
2. **void** BubbleSort2(**int**\* arr, **size\_t** size)
3. {
4. assert(arr);
5. **int** i = 0, j = 0;
7. **for** (i = 0; i < size - 1; i++)//一共要排序size-1次
8. {
9. //每次遍历标志位都要先置为0，才能判断后面的元素是否发生了交换
10. **int** flag = 0;
12. **for** (j = 0; j < size - 1 - i; j++)//选出该趟排序的最大值往后移动
13. {
14. **if** (arr[j] > arr[j + 1])
15. {
16. **int** tmp = arr[j];
17. arr[j] = arr[j + 1];
18. arr[j + 1] = tmp;
19. flag = 1;//只要有发生了交换，flag就置为1
20. }
21. }
23. //判断标志位是否为0，如果为0，说明后面的元素已经有序，就直接return
24. **if** (flag == 0)
25. {
26. **return**;
27. }
28. }
30. }

4、算法分析

（1）算法的最好时间复杂度

    　若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数C和记录移动次数M均达到最小值：

        C(min)=n-1

        M(min)=0。

    　冒泡排序最好的时间复杂度为O(n)。

（2）算法的最坏时间复杂度

    　若初始文件是反序的，需要进行n-1趟排序。每趟排序要进行n-i次关键字的比较(1≤i≤n-1)，且每次比较都必须移动记录三次来达到交换记录位置。在这种情况下，比较和移动次数均达到最大值：

        C(max)=n(n-1)/2=O(n^2)

        M(max)=3n(n-1)/2=O(n^2)

    　冒泡排序的最坏时间复杂度为O(n^2)。

（3）算法的平均时间复杂度为O(n^2)

    　虽然冒泡排序不一定要进行n-1趟，但由于它的记录移动次数较多，故平均时间性能比直接插入排序要差得多。

（4）算法稳定性

    　冒泡排序是就地排序，且它是稳定的。

5、算法优化2（优化内层循环）

   (1)记住最后一次交换发生位置lastExchange的冒泡排序

　　在每趟扫描中，记住最后一次交换发生的位置lastExchange，（该位置之后的相邻记录均已有序）。下一趟排序开始时，R[1..lastExchange-1]是无序区，R[lastExchange..n]是有序区。这样，一趟排序可能使当前无序区扩充多个记录，因此记住最后一次交换发生的位置lastExchange，从而减少排序的趟数。

具体算法：

1. //冒泡排序优化2
2. **void** BubbleSort3(**int**\* arr, **size\_t** size)
3. {
4. assert(arr);
5. **int** i = 0, j = 0;
6. **int** k = size - 1,pos = 0;//pos变量用来标记循环里最后一次交换的位置
8. **for** (i = 0; i < size - 1; i++)//一共要排序size-1次
9. {
10. //每次遍历标志位都要先置为0，才能判断后面的元素是否发生了交换
11. **int** flag = 0;
13. **for** (j = 0; j <k; j++)//选出该趟排序的最大值往后移动
14. {
15. **if** (arr[j] > arr[j + 1])
16. {
17. **int** tmp = arr[j];
18. arr[j] = arr[j + 1];
19. arr[j + 1] = tmp;
20. flag = 1;//只要有发生了交换，flag就置为1
21. pos = j;//循环里最后一次交换的位置 j赋给pos
22. }
23. }
25. k = pos;
26. //判断标志位是否为0，如果为0，说明后面的元素已经有序，就直接return
27. **if** (flag == 0)
28. {
29. **return**;
30. }
31. }
33. }