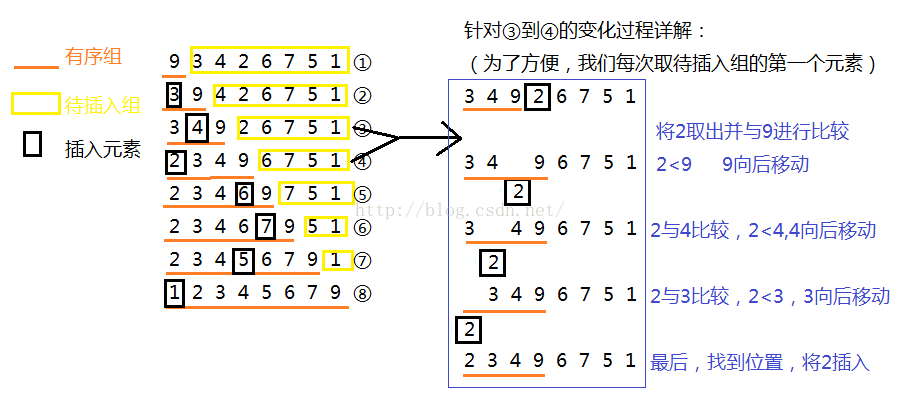
插入排序相对冒泡排序而言是一种较为快捷方便的排序算法。

冒泡排序：<http://blog.csdn.net/llzk_/article/details/51547923>

插入排序原理很简单，讲一组数据分成两组，我分别将其称为有序组与待插入组。每次从待插入组中取出一个元素，与有序组的元素进行比较，并找到合适的位置，将该元素插到有序组当中。就这样，每次插入一个元素，有序组增加，待插入组减少。直到待插入组元素个数为0。当然，插入过程中涉及到了元素的移动。

为了排序方便，我们一般将数据第一个元素视为有序组，其他均为待插入组。

下面以升序为例进行一次图解：



代码为：

**[cpp]** [view plain](https://blog.csdn.net/llzk_/article/details/51628574) [copy](https://blog.csdn.net/llzk_/article/details/51628574)

1. #include<stdio.h>
2. **void** InsertionSort(**int** \*num,**int** n)
3. {
4. **int** i = 0;
5. **int** j = 0;
6. **int** tmp = 0;
7. **for**(i = 1;i<n;i++)
8. {
9. tmp = num[i];//从待插入组取出第一个元素。
10. j = i-1; //i-1即为有序组最后一个元素（与待插入元素相邻）的下标
11. **while**(j>=0&&tmp<num[j])  //注意判断条件为两个，j>=0对其进行边界限制。第二个为插入判断条件
12. {
13. num[j+1] = num[j];//若不是合适位置，有序组元素向后移动
14. j--;
15. }
16. num[j+1] = tmp;//找到合适位置，将元素插入。
17. }
18. }
19. **int** main()
20. {
21. **int** i = 0;
22. **int** num[8]={9,3,4,2,6,7,5,1};
23. InsertionSort(num,8);
24. /\*这个函数必须知道元素的个数，所以将元素个数传入。
25. 有心者可以在函数内部用sizeof求出元素个数 \*/
26. **for**(i=0;i<8;i++)
27. {
28. printf("%d ",num[i]);
29. }
30. **return** 0;
31. }

执行结果：



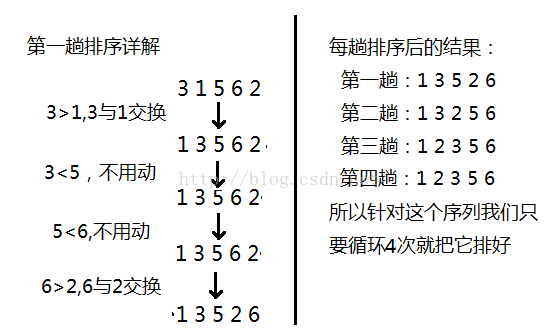
冒泡排序算是排序算法里面的一种较为简单的算法，也是我接触的第一种排序算法，有升序与降序之分，如果面试的时候面试官问道这个题目，一定要问清楚是升序还是降序，这样会给你加分。

下面，我以升序来讲一下它的运作。

1. 1、比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 2、对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
3. 3、针对所有的元素重复以上的步骤，除了最后一个。
4. 4、持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

算法思想图：

  假设有个数列为 3 1 5 6 2，我们以升序对它进行排序。



冒泡排序第一版：

**[cpp]** [view plain](https://blog.csdn.net/llzk_/article/details/51547923) [copy](https://blog.csdn.net/llzk_/article/details/51547923)

1. #include<stdio.h>
2. #include<stdlib.h>
3. **void** bubblesort(**int** \*p,**int** len)
4. {
5. **int** i = 0;
6. **int** j = 0;
7. **for**(i = 0;i<len-1;i++)
8. {
9. /\*每排序一趟，则至少有一个元素已经有序，
10. 用 j<len-i-1 可以缩小排序范围 \*/
11. **for**(j = 0;j<len -1-i;j++)
12. {
13. /\*当前面的元素大于后面的元素时，交换位置\*/
14. **if**(p[j]>p[j+1])
15. {
16. **int** tmp = p[j];
17. p[j] = p[j+1];
18. p[j+1] = tmp;
19. }
20. }
21. }
22. }
23. **int** main()
24. {
25. **int** num[5]={3,1,5,6,2};
26. **int** i=0;
27. bubblesort(num,5);
28. **for**(i=0;i<5;i++)
29. {
30. printf("%d ",num[i]);
31. }
32. **return** 0;
33. }
34. printf("%d ",num[i]);
35. }
36. **return** 0;
37. }

当数列在基本有序的情况下时，例如数列：

                                                   1 2 3 5 4 6 7 8 9 10

在第一趟排序 5 和 4 交换之后数列就已经有序了，所以剩下的8趟排序就是多余的。再用上面的算法处理这个数列，效率就显得低了。所以，我们对它添加一个检视flag。

冒泡排序优化版：

**[cpp]** [view plain](https://blog.csdn.net/llzk_/article/details/51547923) [copy](https://blog.csdn.net/llzk_/article/details/51547923)

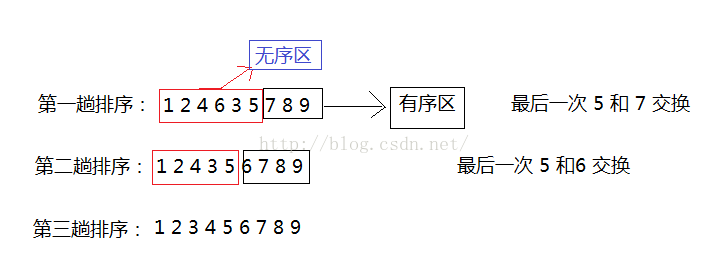
1. #include<stdio.h>
2. #include<stdlib.h>
3. **void** bubblesort(**int** \*p,**int** len)
4. {
5. **int** i = 0;
6. **int** j = 0;
7. **int** flag;
8. **for**(i = 0;i<len-1;i++)//
9. {
10. /\*flag初始化为0\*/
11. flag = 0;
12. /\*每排序一趟，则至少有一个元素已经有序，
13. 用 j<len-i-1 可以缩小排序范围 \*/
14. **for**(j = 0;j<len -1-i;j++)
15. {
16. /\*当前面的元素大于后面的元素时，交换位置\*/
17. **if**(p[j]>p[j+1])
18. {
19. **int** tmp = p[j];
20. p[j] = p[j+1];
21. p[j+1] = tmp;
22. /\*若程序进入if语句,则必然会发生交换，
23. 当发生交换时把flag置成 1 \*/
24. flag = 1;
25. }
26. }
27. /\*判断,如果flag=0，说明上一趟排序没有发生交换，
28. 数列已经有序，break\*/
29. **if**(flag == 0)
30. {
31. **break**;
32. }
33. }
34. }
35. **int** main()
36. {
37. **int** num[10]={1,2,3,5,4,6,7,8,9,10};
38. **int** i=0;
39. bubblesort(num,10);
40. **for**(i=0;i<10;i++)
41. { printf("%d ",num[i]);
42. }
43. **return** 0;
44. }

其实上面的算法还可以进行优化，来看下面一组数据：1 2 7 4 6 3 5 8 9

首先我们要知道：



    当它每完成一趟排序的适合，我们记住它最后一次交换的位置，也就是flag最后一次置1的位置，我们假设这个位置为P，P之后的元素已经有序，P之前的元素还可能无序。所以我们只需要对P之前的元素再进行排序就可以了



我们将前面无序区的数列当做一个新的数列，把它的新长度传给第一层循环，这样就可以大大减少循环次数，提升效率。

冒泡排序最终优化版：

**[cpp]** [view plain](https://blog.csdn.net/llzk_/article/details/51547923) [copy](https://blog.csdn.net/llzk_/article/details/51547923)

1. #include<stdio.h>
2. #include<stdlib.h>
3. **void** bubblesort(**int** \*p,**int** len)
4. {
5. **int** i = 0;
6. **int** j = 0;
7. **int** k = len-1;
8. **int** m = 0;
9. **int** flag;
10. /\*用k来保存新的判断条件\*/
11. **for**(i = 0;i<k;i++)
12. {
13. /\*flag初始化为0\*/
14. flag = 0;
15. m = 0;
16. /\*每排序一趟，则至少有一个元素已经有序，
17. 用 j<len-i-1 可以缩小排序范围 \*/
18. **for**(j = 0;j<len-1-i;j++)
19. {
20. /\*当前面的元素大于后面的元素时，交换位置\*/
21. **if**(p[j]>p[j+1])
22. {
23. **int** tmp = p[j];
24. p[j] = p[j+1];
25. p[j+1] = tmp;
26. /\*若程序进入if语句,则必然会发生交换，
27. 当发生交换时把flag置成 1 \*/
28. flag = 1;
29. /\*记住最后一次交换的位置\*/
30. m = j;
31. }
32. }
33. /\*判断,如果flag=0，说明上一趟排序没有发生交换，
34. 数列已经有序，break\*/
35. **if**(flag == 0)
36. {
37. **break**;
38. }
39. /\*将新的长度赋值给k\*/
40. k = m;
41. }
42. }
43. **int** main()
44. {
45. **int** num[10]={1,2,7,4,6,3,5,8,9};
46. **int** i=0;
47. bubblesort(num,9);
48. **for**(i=0;i<9;i++)
49. {
50. printf("%d ",num[i]);
51. }
52. **return** 0;
53. }

这篇文章是对[上一篇文章](http://blog.csdn.net/cbs612537/article/details/8294960" \t "_blank)中的冒泡排序进行优化

先来说说，冒泡排序哪些地方需要优化：

根据[上一篇文章](http://blog.csdn.net/cbs612537/article/details/8294960)的内容，可以知道冒泡排序的核心是两两对比进行交换。如果有一个**无序数列（2，1，3，4，5，6，7，8，9，10）**

按照[上一篇文章](http://blog.csdn.net/cbs612537/article/details/8294960" \t "_blank)的代码，从第一次循环交换后的操作，可以说都是没必要的。所以，这些操作就是我们需要优化的地方。

**那么如何优化？**

通过观察可以看到，造成没必要的操作主要原因是后面8个数的顺序都已经是有序。所以，我们可以通过**设置一个标记变量，标记数列中的数是否在循环结束前就已经排好序**

代码：

**[cpp]** [view plain](https://blog.csdn.net/cbs612537/article/details/8513723) [copy](https://blog.csdn.net/cbs612537/article/details/8513723)

1. #include <stdio.h>
2. **void** swap(**int** \*a, **int** \*b);
3. **int** main()
4. {
5. **int**    array[10] = {2, 1, 3, 4, 5, 6, 7, 8, 9, 10};
6. **int**    i, j;
7. **int**    flag = 1;   //设置标记变量
8. **for** (i = 0; i < 10 && flag; i++)
9. {
10. flag = 0;      //只要flag在下一次外循环条件检测的时候值为0，就说明已经排好序，不用继续循环
11. **for** (j = 9; j > i; j--)
12. {
13. **if** (array[j] < array[j-1])
14. {
15. swap(&array[j], &array[j-1]);
16. flag = 1;   //如果有交换，就将标记变量赋1
17. }
18. }
19. }
21. **for** (i = 0; i < 10; i++)
22. {
23. printf("%d\n", array[i]);
24. }
25. **return**    0;
26. }
27. **void** swap(**int** \*a, **int** \*b)
28. {
29. **int**    temp;
30. temp = \*a;
31. \*a = \*b;
32. \*b = temp;
33. }

**根据优化过的代码，当最好情况的时候，冒泡排序的时间复杂度是O(n)**