


冒泡排序(交换排序)

排序思想

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. 对每一对相邻元素做同样的工作，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. 针对所有的元素重复以上的步骤，除了最后一个。
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。

第1轮排序: [1,7]

0	1	2	3	4	5	6	7
23	76	34	89	90	34	56	18




step 1:

$i = 1$

$\text{nums}[i]$ 与 $\text{nums}[i-1]$ 比较,
显然, $\text{nums}[i] > \text{nums}[i-1]$
 $i++$

0	1	2	3	4	5	6	7
23	76	34	89	90	34	56	18




0	1	2	3	4	5	6	7
23	34	76	89	90	34	56	18

step 2:

$i = 2$

$\text{nums}[i]$ 与 $\text{nums}[i-1]$ 比较,
显然, $\text{nums}[i] < \text{nums}[i-1]$
 $\text{nums}[i]$ 与 $\text{nums}[i-1]$ 位置交换
 $i++$

0	1	2	3	4	5	6	7
23	34	76	89	90	34	56	18




Step 3:

$i = 3$

$\text{nums}[i]$ 与 $\text{nums}[i-1]$ 比较,
显然, $\text{nums}[i] > \text{nums}[i-1]$
 $i++$

0	1	2	3	4	5	6	7
23	34	76	89	90	34	56	18




Step 4:

$i = 4$

$\text{nums}[i]$ 与 $\text{nums}[i-1]$ 比较,
显然, $\text{nums}[i] > \text{nums}[i-1]$
 $i++$

0	1	2	3	4	5	6	7
23	34	76	89	90	34	56	18




0	1	2	3	4	5	6	7
23	34	76	89	34	90	56	18

Step 5:

$i = 5$

$\text{nums}[i]$ 与 $\text{nums}[i-1]$ 比较,
显然, $\text{nums}[i] < \text{nums}[i-1]$
 $\text{nums}[i]$ 与 $\text{nums}[i-1]$ 位置交换
 $i++$

0	1	2	3	4	5	6	7
23	34	76	89	34	90	56	18




0	1	2	3	4	5	6	7
23	34	76	89	34	56	90	18

Step 6:

$i = 6$

$\text{nums}[i]$ 与 $\text{nums}[i-1]$ 比较,
显然, $\text{nums}[i] < \text{nums}[i-1]$
 $\text{nums}[i]$ 与 $\text{nums}[i-1]$ 位置交换
 $i++$

0	1	2	3	4	5	6	7
23	34	76	89	34	56	90	18



0	1	2	3	4	5	6	7
23	34	76	89	34	56	18	90

Step 7:

$i = 7$

$\text{nums}[i]$ 与 $\text{nums}[i-1]$ 比较,
显然, $\text{nums}[i] < \text{nums}[i-1]$
 $\text{nums}[i]$ 与 $\text{nums}[i-1]$ 位置交换
 $i++$

第2轮排序: [1,6]

第3轮排序: [1,5]

第4轮排序: [1,4]

第5轮排序: [1,3]

每一轮排序按上述排序进行, 终点依次减1,
每次排序都会有一个数的位置被定下来

第6轮排序: [1,2]

第7轮排序: [1,1]

冒泡排序运用时需要注意的

冒泡排序对于**最坏的情况**（严格递减/递增的数组），需要比较和移位的次数为 $n(n-1)/2$ ；

对于**最好的情况**（严格递增/递减的数组），需要比较的次数是 $n-1$ ，需要移位的次数是 0 。

冒泡排序算法分析

冒泡排序的时间复杂度是 $O(n^2)$ ，空间复杂度是 $O(1)$ ，同时也是**稳定排序**，**全局有序**（每次排序将会有有一个元素放在最终位置上）。

代码实现

```
1  class Solution{
2  public:
3      void bubbleSort(vector<int> &nums) {
4          for(int i = 0; i < nums.size() - 1; i++) {
5              for(int j = 1; j < nums.size() - i; j++) {
6                  if(nums[j] < nums[j-1]) {
7                      int tmp = nums[j-1];
8                      nums[j-1] = nums[j];
9                      nums[j] = tmp;
10                 }
11             }
12         }
13     }
14 };
```

加工后执行的结果

```
第0轮: 23, 76, 34, 89, 90, 34, 56, 18
第1轮: 23, 34, 76, 89, 34, 56, 18, 90
第2轮: 23, 34, 76, 34, 56, 18, 89, 90
第3轮: 23, 34, 34, 56, 18, 76, 89, 90
第4轮: 23, 34, 34, 18, 56, 76, 89, 90
第5轮: 23, 34, 18, 34, 56, 76, 89, 90
第6轮: 23, 18, 34, 34, 56, 76, 89, 90
第7轮: 18, 23, 34, 34, 56, 76, 89, 90
```

快速排序(交换排序)

排序思想

快速排序是一种划分交换排序。它采用了一种分治的策略，通常称其为分治法。

1. 先从数列中取出一个数作为基准数。
2. 分区过程，将比这个数大的数全放到它的右边，小于或等于它的数全放到它的左边。
3. 再对左右区间重复第二步，直到各区间只有一个数。

虽然快速排序称为分治法，但分治法这三个字显然无法很好的概括快速排序的全部步骤。因此我的对快速排序作了进一步的说明：挖坑填数+分治法

0	1	2	3	4	5	6	7
23	76	34	89	90	34	56	18
↑							↑

以nums[0] = 23作为基准，
此时蓝标的low++，红标的high--
此时nums[0]就是一个“坑”
先判断low是否小于high
填坑大法

0	1	2	3	4	5	6	7
18	76	34	89	90	34	56	18
	↑						↑

step 1:
先从后往前找比23小的值，放到nums[0]（蓝标）上
nums[7] < 23，所以nums[0] = 23，
nums[7]将nums[0]这个坑填满了，
于是，nums[7]就变成一个“坑”，
low++（那个标指向的坑填满，那个标移动）

0	1	2	3	4	5	6	7
18	76	34	89	90	34	56	76
	↑					↑	

step 2:
从前往后找比23大的值，放到nums[7]（红标）上
nums[1] > 23，所以nums[7] = 76，
nums[1]将nums[7]这个坑填满了，
于是，nums[1]就变成一个“坑”，
high--（那个标指向的坑填满，那个标移动）

0	1	2	3	4	5	6	7
18	23	34	89	90	34	56	76
	↑↑						

step 3:
先从后往前找比23小的值，放到nums[1]（红标）上
随着high--，当low与high相等时，未出现比23小的值，
于是在low=high这个地方，也就是nums[1]还有个坑，
将基准23填入，于是23在1这个位置就被固定下来了

0	1	2	3	4	5	6	7
18	23	34	89	90	34	56	76
↑↑		↑					↑

step 4:
最后就是递归，posl=0，posr=7
quickSort(nums, posl, low-1);
quickSort(nums, low+1, posr);

快速排序运用时需要注意的

快速排序对于**最坏的情况**（严格递减/递增的数组），需要比较和移位的次数为 $n(n-1)/2$ ；

对于**最好的情况**（严格递增/递减的数组），需要比较的次数是 $n-1$ ，需要移位的次数是0。

快速排序算法分析

快速排序最坏的时间复杂度是 $O(n^2)$ ，平均时间复杂度为 $O(n\log n)$ ，空间复杂度最好是 $O(\log n)$ ，最坏是 $O(n)$ ，

但是快速排序是一种**不稳定排序**，**全局有序**。

代码实现

```
1  class Solution{
2  public:
3      int counter=0;
4      void quickSort(vector<int> &nums, int low, int high) {
5          if(low<high) {
6              int num = nums[low];
7              int posl = low;
8              int posr = high;
9              while(low < high) {
10                 //从后往前找比num小的值
```

```

11         while(nums[high] >= num && low < high) {
12             high--;
13         }
14         if(low < high){
15             nums[low++] = nums[high];
16         }
17         while(nums[low] <= num && low < high) {
18             low++;
19         }
20         if(low < high){
21             nums[high--] = nums[low];
22         }
23     }
24     nums[low] = num;
25     quickSort(nums, posl, low-1);
26     quickSort(nums, low+1, posr);
27 }
28 }
29 };

```

加工后执行的结果

```

第0轮: 23, 76, 34, 89, 90, 34, 56, 18
第1轮: 18, 23, 34, 89, 90, 34, 56, 76
第2轮: 18, 23, 34, 89, 90, 34, 56, 76
第3轮: 18, 23, 34, 76, 56, 34, 89, 90
第4轮: 18, 23, 34, 34, 56, 76, 89, 90

```

测试代码

冒泡排序测试代码:

```

1  #include <stdio.h>
2  #include <vector>
3  using namespace std;
4
5  class Solution{
6  public:
7      void bubbleSort(vector<int> &nums) {
8          // 查看
9          printf("第0轮: ");
10         for(int j = 0; j < nums.size(); j++) {
11             printf("%d",nums[j]);
12             if(j!=nums.size()-1) printf(",");
13         }
14         printf("\n");
15         for(int i = 0; i < nums.size() - 1; i++) {
16             for(int j = 1; j < nums.size() - i; j++) {
17                 if(nums[j] < nums[j-1]) {
18                     int tmp = nums[j-1];
19                     nums[j-1] = nums[j];
20                     nums[j] = tmp;

```

```

21         }
22     }
23     // 查看
24     printf("第%d轮: ", i+1);
25     for(int j = 0; j < nums.size(); j++) {
26         printf("%d", nums[j]);
27         if(j!=nums.size()-1) printf(",");
28     }
29     printf("\n");
30 }
31 }
32 };
33
34 int main() {
35     vector<int> v;
36     v.push_back(23);
37     v.push_back(76);
38     v.push_back(34);
39     v.push_back(89);
40     v.push_back(90);
41     v.push_back(34);
42     v.push_back(56);
43     v.push_back(18);
44     Solution solution;
45     solution.bubbleSort(v);
46     return 0;
47 }

```

快速排序测试代码

```

1  #include <stdio.h>
2  #include <vector>
3  using namespace std;
4
5  class Solution{
6  public:
7      int counter=0;
8      void quickSort(vector<int> &nums, int low, int high) {
9          if(low<high) {
10             // 查看
11             printf("第%d轮: ", counter);
12             for(int j = 0; j < nums.size(); j++) {
13                 printf("%d", nums[j]);
14                 if(j!=nums.size()-1) printf(",");
15             }
16             printf("\n");
17             counter++;
18             int num = nums[low];
19             int posl = low;
20             int posr = high;
21             while(low < high) {

```

```

22         //从后往前找比num小的值
23         while(nums[high] >= num && low < high) {
24             high--;
25         }
26         if(low < high){
27             nums[low++] = nums[high];
28         }
29         while(nums[low] <= num && low < high) {
30             low++;
31         }
32         if(low < high){
33             nums[high--] = nums[low];
34         }
35     }
36     nums[low] = num;
37     quickSort(nums, posl, low-1);
38     quickSort(nums, low+1, posr);
39 }
40 }
41 };
42
43 int main() {
44     vector<int> v;
45     v.push_back(23);
46     v.push_back(76);
47     v.push_back(34);
48     v.push_back(89);
49     v.push_back(90);
50     v.push_back(34);
51     v.push_back(56);
52     v.push_back(18);
53     solution solution;
54     solution.quickSort(v,0,7);
55     return 0;
56 }

```