

原 快速排序(三种算法实现和非递归实现)

2017年11月29日 18:41:44 清枫若待佳人醉 阅读数 116691 [更多](#)

版权声明：本文为博主原创文章，遵循 [CC 4.0 BY-SA](#) 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/qq_36528114/article/details/78667034

快速排序(Quick Sort)是对冒泡排序的一种改进，基本思想是选取一个记录作为枢轴，经过一趟排序，将整段序列分为两个部分，其中一部分的值都小于枢轴，另一部分都大于枢轴，继续对这两部分继续进行排序，从而使整个序列达到有序。

递归实现：

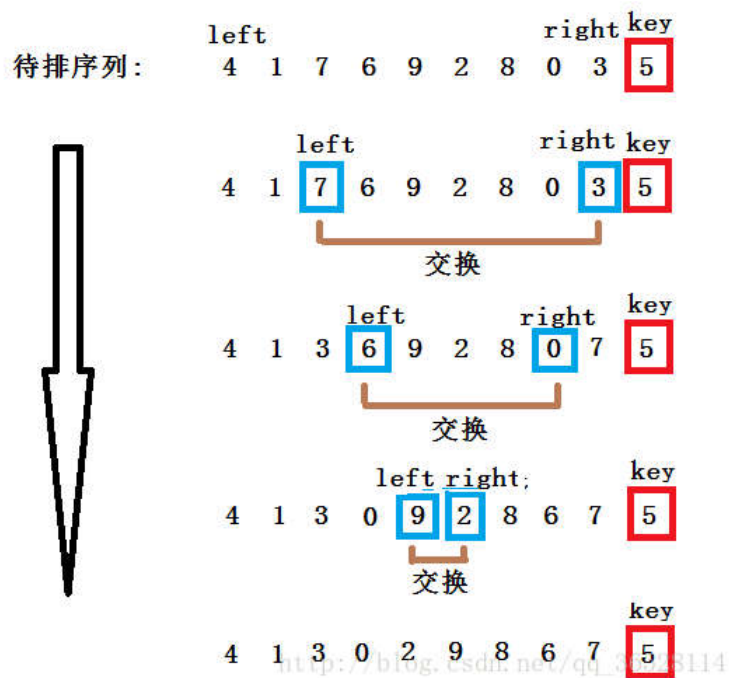
```
1 void QuickSort(int* array,int left,int right)
2 {
3     assert(array);
4     if(left >= right)//表示已经完成一个组
5     {
6         return;
7     }
8     int index = PartSort(array,left,right);//枢轴的位置
9     QuickSort(array,left,index - 1);
10    QuickSort(array,index + 1,right);
11 }
```

PartSort()函数是进行一次快排的算法。

对于快速排序的一次排序，有很多种算法，我这里列举三种。

左右指针法

1. 选取一个关键字(key)作为枢轴，一般取整组记录的第一个数/最后一个，这里采用选取序列最后一个数为枢轴。
2. 设置两个变量left = 0;right = N - 1;
3. 从left一直向后走，直到找到一个大于key的值，right从后至前，直至找到一个小于key的值，然后交换这两个数。
4. 重复第三步，一直往后找，直到left和right相遇，这时将key放置left的位置即可。



当 $left \geq right$ 时，一趟快速排序就完成了，这时将Key和array[left]的值进行一次交换。

一次快排的结果:4 1 3 0 2 5 9 8 6 7

基于这种思想，可以写出代码：

```

1 int PartSort(int* array,int left,int right)
2 {
3     int& key = array[right];
4     while(left < right)
5     {
6         while(left < right && array[left] <= key)
7         {
8             ++left;
9         }
10        while(left < right && array[right] >= key)
11        {
12            --right;
13        }
14        swap(array[left],array[right]);
15    }

```

```

16 swap(array[left],key);
17 return left;
18 }

```

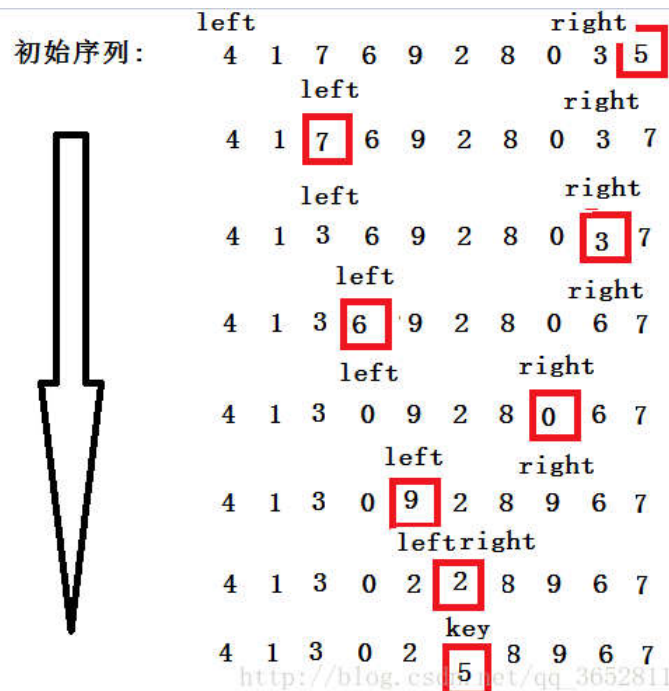
问题：下面的代码为什么还要判断left < right？

```
1 while(left < right && array[left] <= key)
```

key是整段序列最后一个，right是key前一个位置，如果array[right]这个位置的值和key相等，满足array[left] <= key，然后++left,这时候left会走到key的下标处。

###挖坑法

1. 选取一个关键字(key)作为枢轴，一般取整组记录的第一个数/最后一个，这里采用选取序列最后一个数为枢轴，也是初始的坑位。
2. 设置两个变量left = 0;right = N - 1;
3. 从left一直向后走，直到找到一个大于key的值，然后将该数放入坑中，坑位变成了array[left]。
4. right一直向前走，直到找到一个小于key的值，然后将该数放入坑中，坑位变成了array[right]。
5. 重复3和4的步骤，直到left和right相遇，然后将key放入最后一个坑位。



http://blog.csdn.net/qq_36528114



78



48



当 $left \geq right$ 时，将key放入最后一个坑，就完成了排序。

注意，left走的时候right是不动的，反之亦然。因为left先走，所有最后一个坑肯定在array[right]。

写出代码：

```
1 int PartSort(int* array,int left,int right)
2 {
3     int key = array[right];
4     while(left < right)
5     {
6         while(left < right && array[left] <= key)
7         {
8             ++left;
9         }
10        array[right] = array[left];
11        while(left < right && array[right] >= key)
12        {
13            --right;
14        }
15        array[left] = array[right];
16    }
17    array[right] = key;
18    return right;
19 }
```

###前后指针法

1. 定义变量cur指向序列的开头，定义变量pre指向cur的前一个位置。
2. 当 $array[cur] < key$ 时，cur和pre同时往后走，如果 $array[cur] > key$ ，cur往后走，pre留在大于key的数值前一个位置。
3. 当 $array[cur]$ 再次 $< key$ 时，交换 $array[cur]$ 和 $array[pre]$ 。

通俗一点就是，在没找到大于key值前，pre永远紧跟cur，遇到大的两者之间机会拉开差距，中间差的肯定是连续的大于key的值，当再次遇到小于key的值时，交换两个下标对应的值就好了。

带着这种思想，看着图示应该就能理解了。

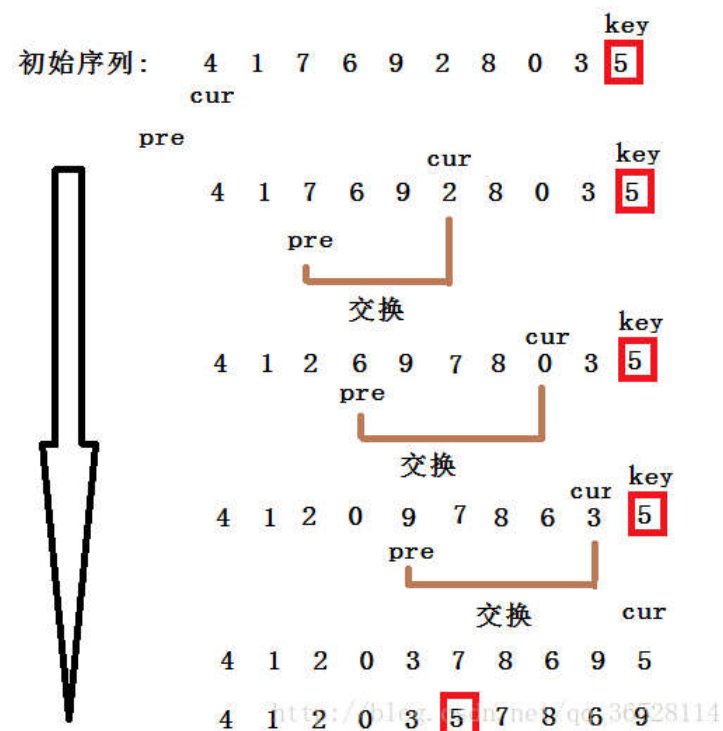


78



48





下面是实现代码：

```

1 int PartSort(int* array,int left,int right)
2 {
3     if(left < right){
4         int key = array[right];
5         int cur = left;
6         int pre = cur - 1;
7         while(cur < right)
8         {
9             while(array[cur] < key && ++pre != cur)//如果找到小于key的值，并且cur和pre之间有距离时则进行交换。注意两个条件的先后位置不能更换，可以参照评论中的解释
10            {
11                swap(array[cur],array[pre]);
12            }
13            ++cur;
14        }
15        swap(array[++pre],array[right]);
16        return pre;
17    }

```

```
18     return -1;
19 }
```

最后的前后指针法思路有点绕，多思考一下就好了。它最大的特点就是，左右指针法和挖坑法只能针对顺序序列进行排序，如果是对一个链表进行排序，就无用武之地了。

所以记住了，前后指针这个特点！

###快速排序的优化

首先快排的思想是找一个枢轴，然后以枢轴为中介线，一遍都小于它，另一边都大于它，然后对两段区间继续划分，那么枢轴的选取就很关键。

1、三数取中法

上面的代码思想都是直接拿序列的最后一个值作为枢轴，如果最后这个值刚好是整段序列最大或者最小的值，那么这次划分就是没意义的。

所以当序列是正序或者逆序时，每次选到的枢轴都是没有起到划分的作用。快排的效率会极速退化。

所以可以每次在选枢轴时，在序列的第一，中间，最后三个值里面选一个中间值出来作为枢轴，保证每次划分接近均等。

2、直接插入

由于是递归程序，每一次递归都要开辟栈帧，当递归到序列里的值不是很多时，我们可以采用直接插入排序来完成，从而避免这些栈帧的消耗。

整个代码：

```
1 //三数取中
2 int GetMid(int* array,int left,int right)
3 {
4     assert(array);
5     int mid = left + ((right - left)>>1);
6     if(array[left] <= array[right])
7     {
8         if(array[mid] < array[left])
9             return left;
10        else if(array[mid] > array[right])
11            return right;
12        else
13            return mid;
14    }
15    else
16    {
17        if(array[mid] < array[right])
18            return right;
19        else if(array[mid] > array[left])
20            return left;
21        else
22            return mid;
```



78



48



```
23     }
24
25 }
26
27 //左右指针法
28 int PartSort1(int* array,int left,int right)
29 {
30     assert(array);
31     int mid = GetMid(array,left,right);
32     swap(array[mid],array[right]);
33
34     int& key = array[right];
35     while(left < right)
36     {
37         while(left < right && array[left] <= key)//因为有可能有相同的值，防止越界，所以加上left < right
38             ++left;
39         while(left < right && array[right] >= key)
40             --right;
41
42         swap(array[left],array[right]);
43     }
44
45     swap(array[left],key);
46     return left;
47 }
48
49 //挖坑法
50 int PartSort2(int* array,int left,int right)
51 {
52     assert(array);
53     int mid = GetMid(array,left,right);
54     swap(array[mid],array[right]);
55
56     int key = array[right];
57     while(left < right)
58     {
59         while(left < right && array[left] <= key)
60             ++left;
61         array[right] = array[left];
62
63         while(left < right && array[right] >= key)
64             --right;
65         array[left] = array[right];
66     }
67     array[right] = key;
68     return right;
```



78



48



```
69 }
70
71 //前后指针法
72 int PartSort3(int* array,int left,int right)
73 {
74     assert(array);
75     int mid = GetMid(array,left,right);
76     swap(array[mid],array[right]);
77     if(left < right){
78         int key = array[right];
79         int cur = left;
80         int pre = left - 1;
81         while(cur < right)
82         {
83             while(array[cur] < key && ++pre != cur)
84             {
85                 swap(array[cur],array[pre]);
86             }
87             ++cur;
88         }
89         swap(array[++pre],array[right]);
90         return pre;
91     }
92     return -1;
93 }
94
95 void QuickSort(int* array,int left,int right)
96 {
97     assert(array);
98     if(left >= right)
99         return;
100
101     //当序列较短时, 采用直接插入
102     if((right - left) <= 5)
103         InsertSort(array,right-left+1);
104
105     int index = PartSort3(array,left,right);
106     QuickSort(array,left,index-1);
107     QuickSort(array,index+1,right);
108 }
109
110 int main()
111 {
112     int array[] = {4,1,7,6,9,2,8,0,3,5};
113     QuickSort(array,0,sizeof(array)/sizeof(array[0]) - 1);//因为传的是区间, 所以这里要 - 1;
114 }
```



78



48



非递归实现

递归的算法主要是在划分子区间，如果要非递归实现快排，只要使用一个栈来保存区间就可以了。
一般将递归程序改成非递归首先想到的就是使用栈，因为递归本身就是一个压栈的过程。

```
1 void QuickSortNotR(int* array,int left,int right)
2 {
3     assert(array);
4     stack<int> s;
5     s.push(left);
6     s.push(right);//后入的right，所以要先拿right
7     while(!s.empty())//栈不为空
8     {
9         int right = s.top();
10        s.pop();
11        int left = s.top();
12        s.pop();
13
14        int index = PartSort(array,left,right);
15        if((index - 1) > left)//左子序列
16        {
17            s.push(left);
18            s.push(index - 1);
19        }
20        if((index + 1) < right)//右子序列
21        {
22            s.push(index + 1);
23            s.push(right);
24        }
25    }
26 }
```

上面就是关于快速排序的一些知识点，如果哪里有错误，还望指出。

我见过最通俗易懂的**快速排序**过程讲解，转自《坐在马桶上看算法：**快速排序**》

阅读数 5万+

如果以上C代码看不懂,请看下面java代码:public static int Partition(int[] a,int p,int r){ int x=a[r-1]; ... 博文 来自: [vayneXiao的...](#)



想对作者说点什么



代安：左右指针法while(left<right){swap(array[left],array[right]);后面是不是少了两行left++;right--;}我觉得 (3个月前 #28楼) 收起回复



Amin_Luckdog 回复 代安：不加也是对的 (1周前)



78



48

