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

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，然后继续对这两部分继续进行排序，从而使整个序列达到有序。

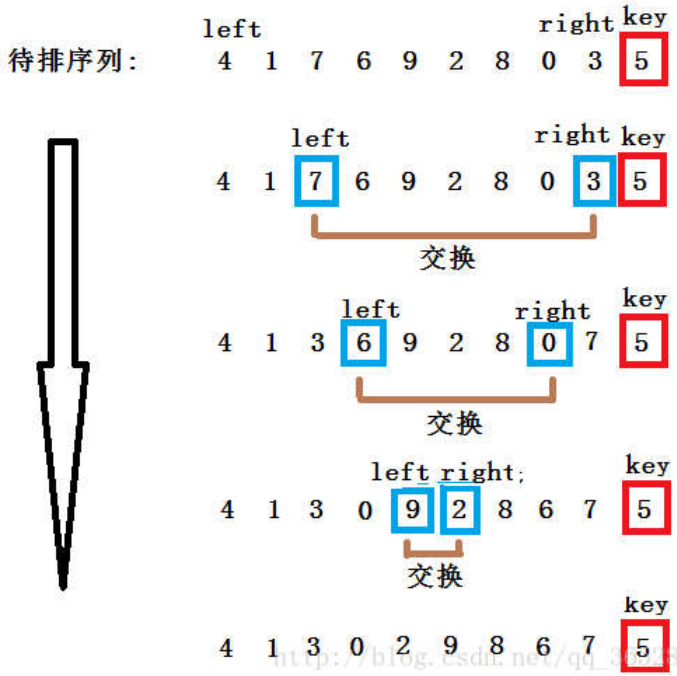
递归实现：

```
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 >= 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：

```
int PartSort(int* array,int left,int right)
{
    int& key = array[right];
    while(left < right)
    {
        while(left < right && array[left] <= key)
        {
            ++left;
        }
        while(left < right && array[right] >= key)
        {
            --right;
        }
        swap(array[left],array[right]);
    }
```

```
    swap(array[left],key);
    return left;
}
```

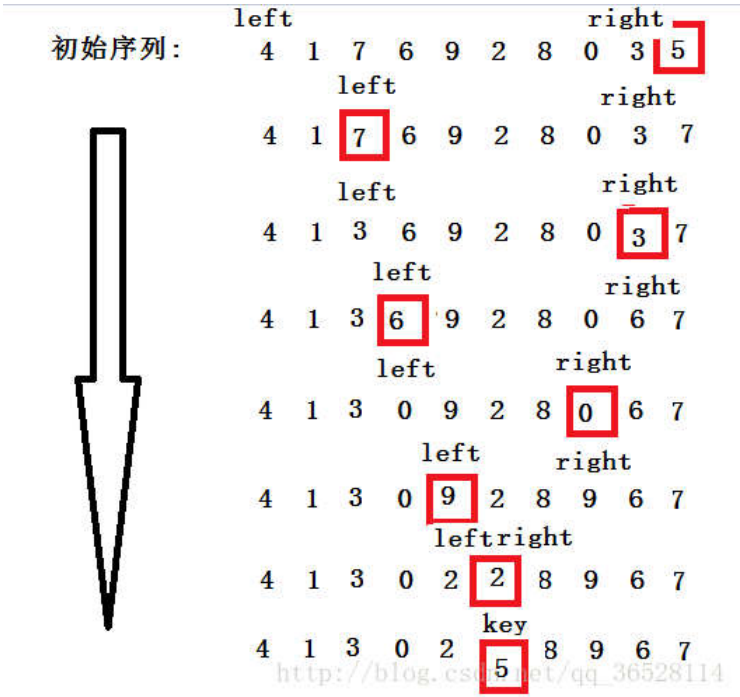
问题：下面的代码为什么还要判断left < right ?

```
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。



当left >= right时，将key放入最后一个坑，就完成了一次排序。
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：

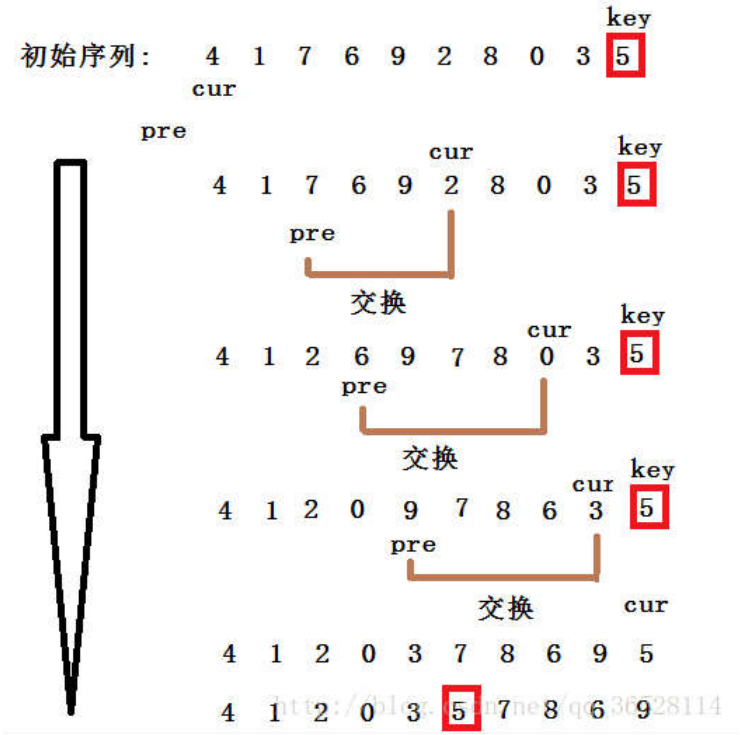
```
int PartSort(int* array,int left,int right)
{
    int key = array[right];
    while(left < right)
    {
        while(left < right && array[left] <= key)
        {
            ++left;
        }
        array[right] = array[left];
        while(left < right && array[right] >= key)
        {
            --right;
        }
        array[left] = array[right];
    }
    array[right] = key;
    return right;
}
```

###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。



下面是实现代码：

```
int PartSort(int* array,int left,int right)
{
    if(left < right){
        int key = array[right];
        int cur = left;
        int pre = cur - 1;
        while(cur < right)
        {
            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
            {
                swap(array[cur],array[pre]);
            }
            ++cur;
        }
        swap(array[++pre],array[right]);
        return pre;
    }
```

```
    return -1;
}
```

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：

```
//三数取中
int GetMid(int* array,int left,int right)
{
    assert(array);
    int mid = left + ((right - left)>>1);
    if(array[left] <= array[right])
    {
        if(array[mid] <  array[left])
            return left;
        else if(array[mid] > array[right])
            return right;
        else
            return mid;
    }
    else
    {
        if(array[mid] < array[right])
            return right;
        else if(array[mid] > array[left])
            return left;
        else
            return mid;
```

```
    }

}

//左右指针法
int PartSort1(int* array,int left,int right)
{
    assert(array);
    int mid = GetMid(array,left,right);
    swap(array[mid],array[right]);

    int& key = array[right];
    while(left < right)
    {
        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
            ++left;
        while(left < right && array[right] >= key)
            --right;

        swap(array[left],array[right]);
    }

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

//挖坑法
int PartSort2(int* array,int left,int right)
{
    assert(array);
    int mid = GetMid(array,left,right);
    swap(array[mid],array[right]);

    int key = array[right];
    while(left < right)
    {
        while(left < right && array[left] <= key)
            ++left;
        array[right] = array[left];

        while(left < right && array[right] >= key)
            --right;
        array[left] = array[right];
    }
    array[right] = key;
    return right;
```

```
}

//前后指针法
int PartSort3(int* array,int left,int right)
{
    assert(array);
    int mid = GetMid(array,left,right);
    swap(array[mid],array[right]);
    if(left < right){
        int key = array[right];
        int cur = left;
        int pre = left - 1;
        while(cur < right)
        {
             while(array[cur] < key && ++pre != cur)
             {
                 swap(array[cur],array[pre]);
             }
                ++cur;
        }
            swap(array[++pre],array[right]);
            return pre;
    }
    return -1;
}

void QuickSort(int* array,int left,int right)
{
    assert(array);
    if(left >= right)
        return;

    //当序列较短时，采用直接插入
    if((right - left) <= 5)
    InsertSort(array,right-left+1);

    int index = PartSort3(array,left,right);
    QuickSort(array,left,index-1);
    QuickSort(array,index+1,right);
}

int main()
{
    int array[] = {4,1,7,6,9,2,8,0,3,5};
    QuickSort(array,0,sizeof(array)/sizeof(array[0]) -1);//因为传的是区间，所以这里要 - 1;
}
```

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。

```
void QuickSortNotR(int* array,int left,int right)
{
    assert(array);
    stack<int> s;
    s.push(left);
    s.push(right);//后入的right，所以要先拿right
    while(!s.empty)//栈不为空
    {
        int right = s.top();
        s.pop();
        int left = s.top();
        s.pop();

        int index = PartSort(array,left,right);
        if((index - 1) > left)//左子序列
        {
            s.push(left);
            s.push(index - 1);
        }
        if((index + 1) < right)//右子序列
        {
            s.push(index + 1);
            s.push(right);
        }
    }
}
```

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。