快速排序

和归并排序一样，也是采用分治(Divide and Conquer)思想。分为三步：

分解：将数组A[p...q]划分成两个数组A[p..r-1]和A[r+1..q]，使得A[p..r-1]中的每个元素都小于等于A[r]，并且A[r+1..q]中所有元素大于等于A[r]，A[r]称为主元。

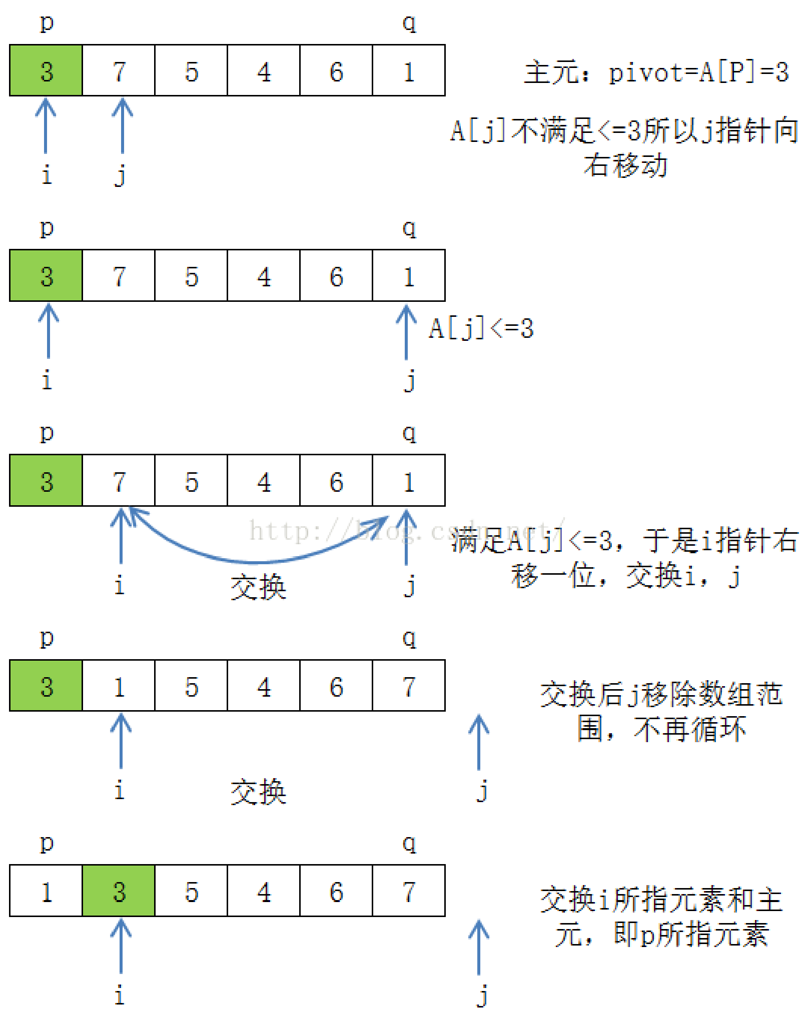
解决：递归调用快速排序，对两个子数组进行排序

合并：不需要合并操作，子数组采用原址排序，已经有序。

和归并排序相比，归并排序主要工作在于合并操作，而快速排序在于划分。划分数组的过程如下描述

[plain]
```
PARTITION(A,p,q)
    x = A[p]
    i = p;
    for j = p+1 to q
        if A[j] <= x
            i = i+1
            exchange A[i] A[j]
    exchange A[p] A[i]
```

详细的划分过程如图所示



经过一次partition划分，分成两个数组A[p..r-1]和A[r+1..q]，使得A[p..r-1]中的每个元素都小于等于A[r]，并且A[r+1..q]中所有元素大于等于A[r]，r为partition的返回值

将数组A进行划分的关键代码

[java]
```java
private int partition(int[] a, int p, int q) {
        int i = p, j = p + 1;
        while (j <= q) {
            if (a[j] <= a[p]) {
                i++;
                swap(a, i, j);
```

```java
            }
            j++;
        }
        swap(a, i, p);
        return i;
    }
```

于是，快速排序的算法描述为

```java
[java]
public static void quickSort(int[] a, int p, int q) {
        if (p < q) {
            int r = partition(a, p, q);
            quickSort(a, p, r - 1);
            quickSort(a, r + 1, q);
        }
    }
```

快速排序的性能：

快速排序在再坏情况下，运气实在糟透了，每次都出现不平等的划分，将其划分成了1个元素和n-1个元素，于是有$T(n)=T(n-1)+T(1)+\theta(n)$，则：$T(n)=O(n^2)$

最好情况下,我们假设每次划分都是平等划分，即$T(n)=2T(n/2)+\theta(n)$，于是$T(n)=O(n\lg n)$.

可以证明，只要是常数比例的划分，算法的运行时间都是$T(n)=O(n\lg n)$。因此，快速排序的平均运行时间更接近于其最好情况而不是最坏情况。

为了保证每次划分出现常数比例的划分，在算法中引入随机性。

```java
[java]
private int randomPartition(int[] a, int p, int q) {
        int i = (int) (Math.random() * (q - p) + p);
        swap(a, p, i);
        return partition(a, p, q);
    }
```

可以证明，快速排序的期望运行时间为$T(n)=O(n\lg n)$，最坏运行时间为$T(n)=O(n^2)$

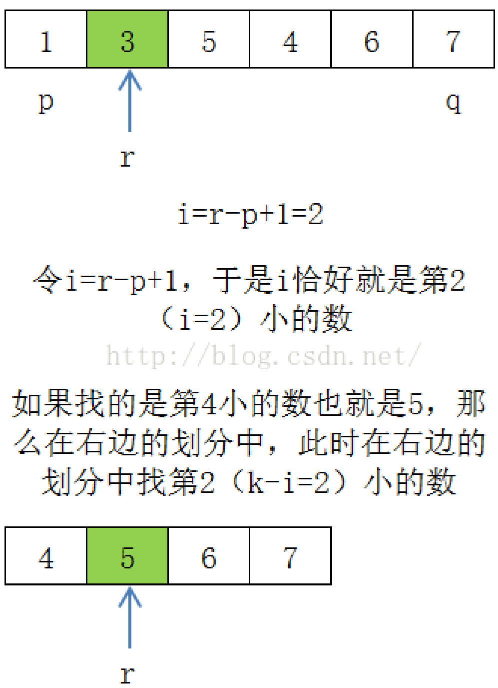
同时，快速排序也是不稳定的排序算法，例如A=[2，3，1，2]，经过第一次划分后，划分为[2，1，2，3]，这时的r位于下标2处，且与A中的两个2的顺序相比已经交换过一次了。两个2逆序。

关于找数组第**k**小的数

其实没看算法的时候我觉得需要排序，然后可以在$O(1)$的时间里面找到第k小的数，学习了算法就不能犯这样的错误啦，常用的好的排序算法比如快速排序，它的时间复杂度为$O(n\log n)$。事实上我们可以在$O(n)$的时间内找到数组的第k小的元素。

答案就在上面讲的快速排序中，事实上，在上面的快速排序的划分过程中其实已经产生了第K小的数。

可以参考下图所示



获得数组第k小的数的关键代码

[java]

```java
public static int getKthValue(int[] a, int p, int q, int k) {
        if (p == q) return a[p];
        int r = randomPartition(a, p, q);
        int i = r - p + 1;// r是当前序列里面第i小的数字
        if (i == k) {
            return a[r];
        } else if (k < i) {
            return getKthValue(a,p,r-1,k);
        } else return getKthValue(a,r+1,q,k - i);
    }
```

这种算法的时间复杂度可以达到期望为线性。E[T(n)]=O(n)

来源： http://blog.csdn.net/cauchyweierstrass/article/details/49766715