快速排序（转载）

作者：940140976
原文链接：https://ld246.com/article/1581468244617
来源网站：链滴

原博客：https://blog.csdn.net/qq_41765114/article/details/86435110

分治代码

```
/**
 *从中间划分
 * 子问题：左边排好序
 *        右边排好序
 */
public static void quickSort(int[] arr,int p,int r) {
    if(p<r){
        int q = partition(arr,p,r);
        quickSort(arr,p,q-1);
        quickSort(arr,q+1,r);
    }
}
```

1. 单向扫描

主元(X)+ 左指针(p1)+ 右指针(p2)

从左往右，若 p1 所指元素 <= X，则 p1 右移；

若 p1 所指元素 > X，则 p1 与 p2 的元素交换，p2 左移，p1 不动。

边界：当 p1 > p2 时，p2 左边的元素均 <= X，把 X 和 p2 所指元素交换。

```
public static int partition(int[] arr, int p, int r) {
    int pivot = arr[p];//定主元
    int sp = p+1;
    int bigger = r;
    //先做后右
    while(sp<=bigger){
        if(arr[sp]<pivot){
            sp++;
        }else{
            swap(arr,sp,bigger);//交换两个扫描指针的元素
            bigger--;
        }
    }
    swap(arr,p,bigger);
    return bigger;
}
```

2. 双向扫描

主元(X)+ 左指针(p1)+ 右指针(p2)

先从左往右，p1 移到第一个 > X 的元素；

再从右往左，p2 移到第一个 <= X 的元素；

p1 和 p2 的元素进行交换，重复以上过程直至 p1 与 p2 交错，进行交换

```
public static int partition2(int[] arr, int p, int r) {
    int pivot = arr[p];//定主元
```

```
    int left = p+1;
    int right = r;
    //先做后右
    while(left<=right){
        //加上判断条件防止交叉
        while(left<=right&&arr[left]<=pivot) left++;
        while(left<=right&&arr[r]>pivot) right--;
        if(left<right)
            swap(arr, left, right);
    }
    swap(arr,p,right);
    return right;
}
```

无论是单向扫描还是双向扫描，最后的左指针总会指向从左边数第一个大于主元的数，有指针会指向右边数第一个不大于主元的数，右指针指向的位置就是主元排序后的正确位置

3. 三分法

主元(X)+ 左指针(p1)+ 右指针(p2)+“等于”指针(e)

从左往右，若 p1 所指元素 < X，则 p1 与 e 的元素进行交换，均右移；

若 p1 所指元素 = X，则 p1 右移；

若 p1 所指元素 > X，则 p1 与 p2 的元素交换，p2 左移，p1 不动。

边界：当 p1 > p2 时，e 左移，X 与 e 的元素交换。

e 指向相等区间的第一个元素，s 指向等于区间的后一个元素，b 为移动元素。

指针停止移动，e 不变，s 指向等于区间的后一个元素，也是大于主元的第一个元素指向等于区间的后一个元素，也是等于区间的边界

与原博客的单向扫描不同，本篇的三分用的是双向扫描，所以代码略长

```
/**
 * 有相同排序值的排序法：三分法
 * 提升效率
 * 增加一个指针保存等于区间的起始位置
 *
 * 从左向右移动
 * < s++,e++  => = s++ < e和s元素交换，e++，s++ > 与b交换 如果b<主元，e和b交换，e+
 b=主元 b和s交换
 * 边界：s>b
 * 返回数组等于区间的边界
 */
public static int[] partition3(int[] arr, int p, int r) {
    int pivot = arr[p];//定主元
    int e = p;
    int left = p;
    int right = r;
        while(left<=right){
//          从左向右找，如果left<主元
            if(arr[left]<pivot){
                swap(arr, left,e);
```

```
                left++;
                e++;
            }
//          如果left=主元
            if(arr[left]==pivot){
                left++;
            }
//          如果left>主元
            if(arr[left]>pivot){
                while(true){
                    if(arr[right]>pivot) right--;
                    if(arr[right]==pivot) {
                        swap(arr,left,right);
                        left++;
                        right--;
                        break;
                    }
                    if(arr[right]<pivot){
                        swap(arr,e,right);
                        e++;
                        swap(arr,left,right);
                        right--;
                        left++;
                        break;
                    }
                }
            }
        }
        return new int[]{e,right};
    }
    public static void quickSort2(int[] arr,int p,int r) {
        if(p<r){
            int q = partition3(arr,p,r)[0];
            int l = partition3(arr,p,r)[1];
            quickSort(arr,p,q);
            quickSort(arr,l,r);
        }
    }
```

快速排序的算法的性能取决于划分的对称性，最坏情况下划分的结果为 n-1 和 1，因此可以引入随机略提高其性能。

4.随机选取主元