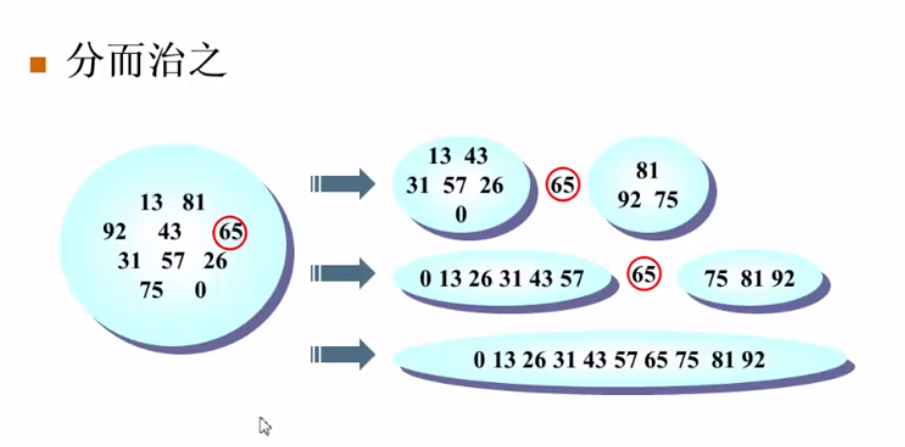
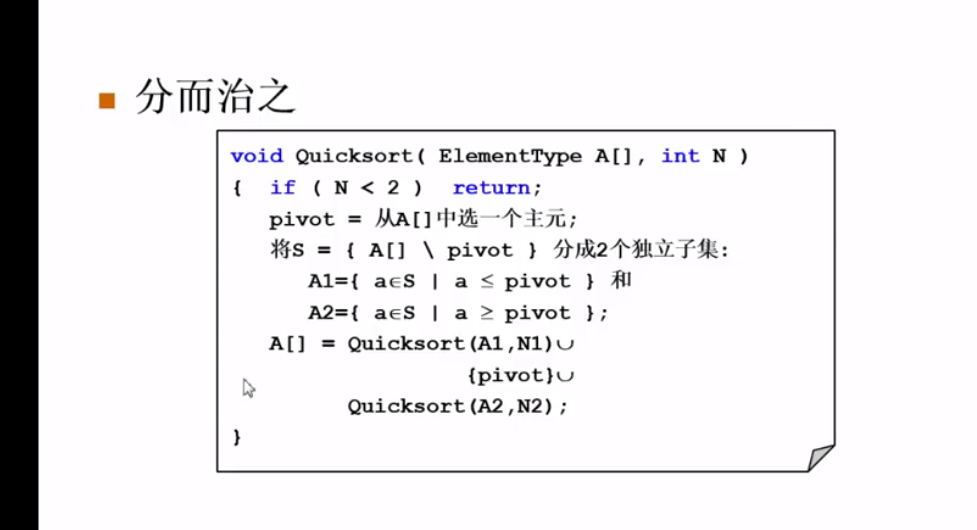
# 算法概述：



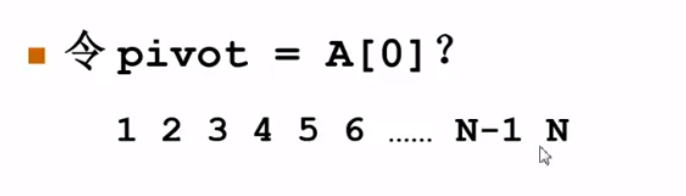
## 伪码描述：



# 如何选主元【会影响算法的时间复杂度】

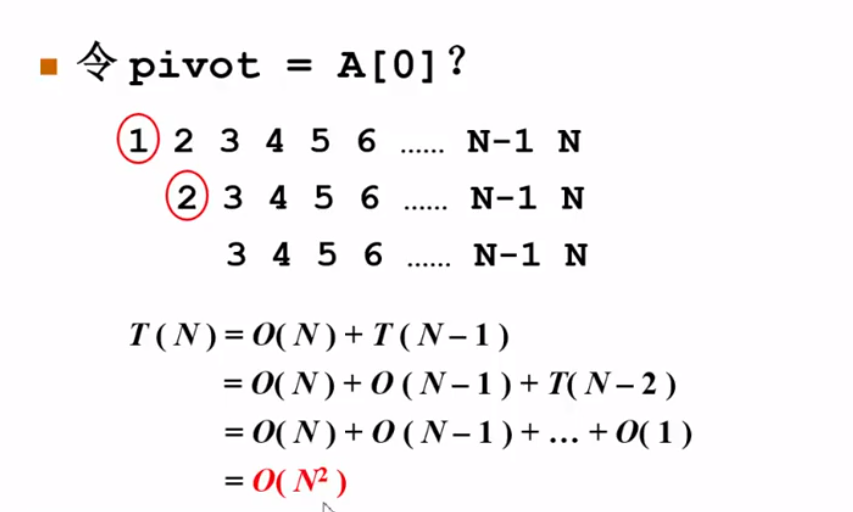
## 例如：直接选第一个元素作为主元：

万一，是一组按顺序排好的数据：



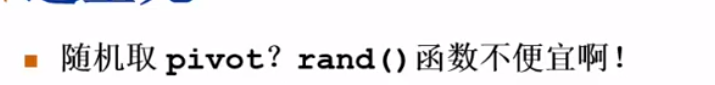
那么它的时间复杂度就是：

O[n] 是完成子集划分，T[] 就是要递归处理的元素个数

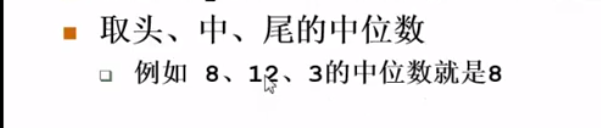


## 又或者是可以随机取：

但也不太好：



## 比较常用的取法是：取头、中、尾的中位数：



8/12/3按大小排好后的中位数是：8

### 取中位数的实现代码：

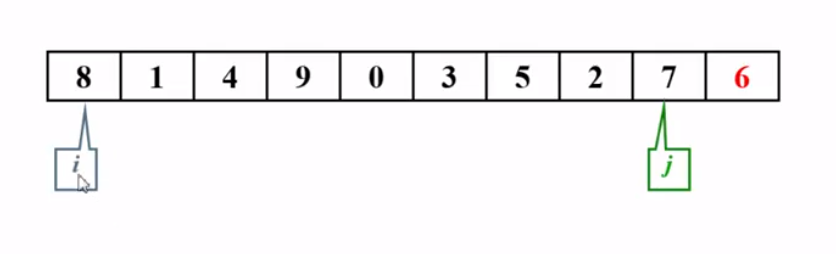
Left 是头元素、Right 是尾元素。

绿色部分是减少了工作量



# 子集划分：【详见视频】

像下面这样：6为主元。



设置变量 i，j分别存放头和尾的下标。充当指针（并不是真的指针）。

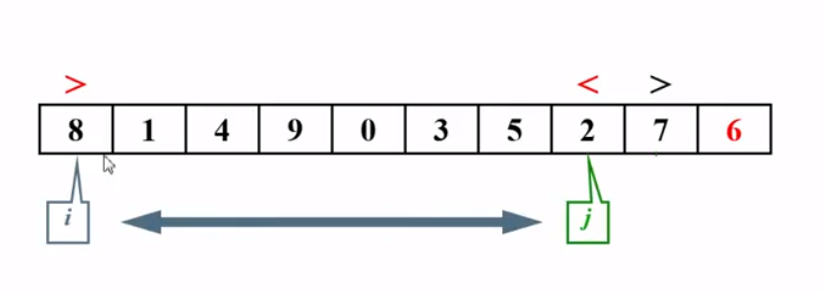
我们认为i是指向 主元左边的元素。j是指向主元右边的元素。

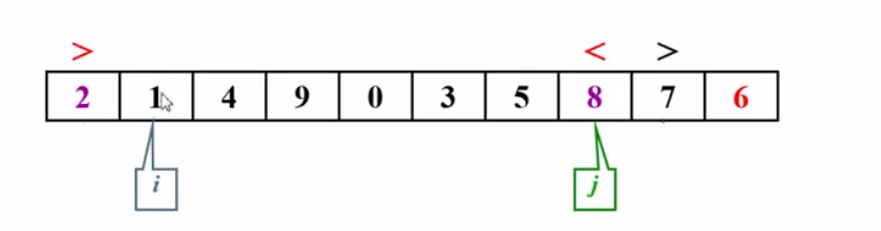
**主元 左边的元素是要比主元小，而主元右边的元素是要比主元大。**

先对i指向的元素 与 主元比较，如果比主元大（不合理）就停下，开始j指向元素的比较。否则i++指向到下一个，继续比较。

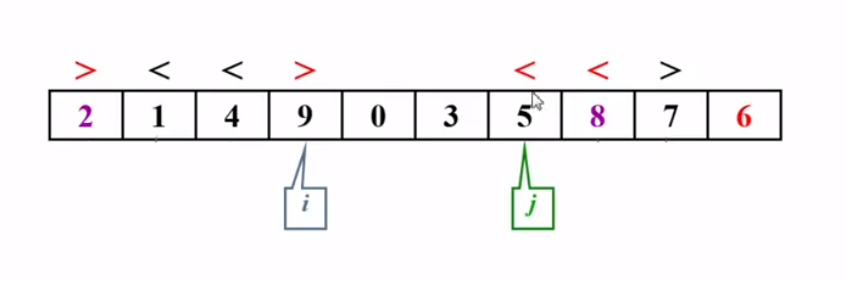
然后对j指向的元素 与 主元进行比较，如果比主元小就停下来。否则继续比较。

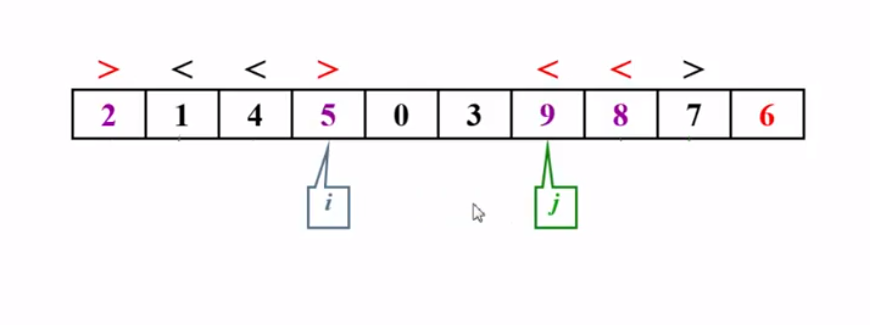
当发现两边都有不对的元素就进行交换：



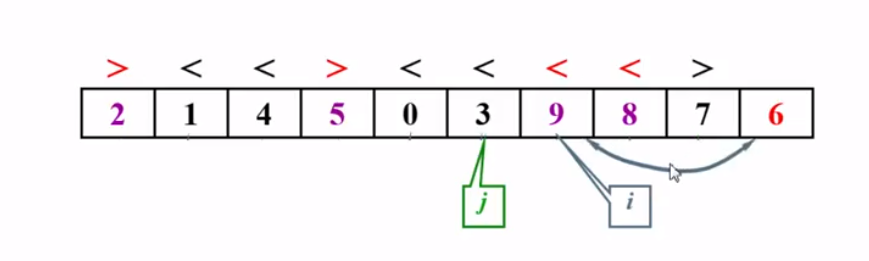


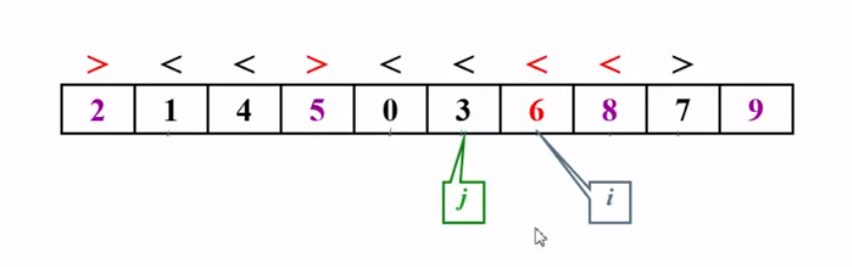
然后进行下一轮比较：





当i>j,就是结束的时候了。主元与i所在的元素调换：





然后就是递归，对子集进行快速排序。

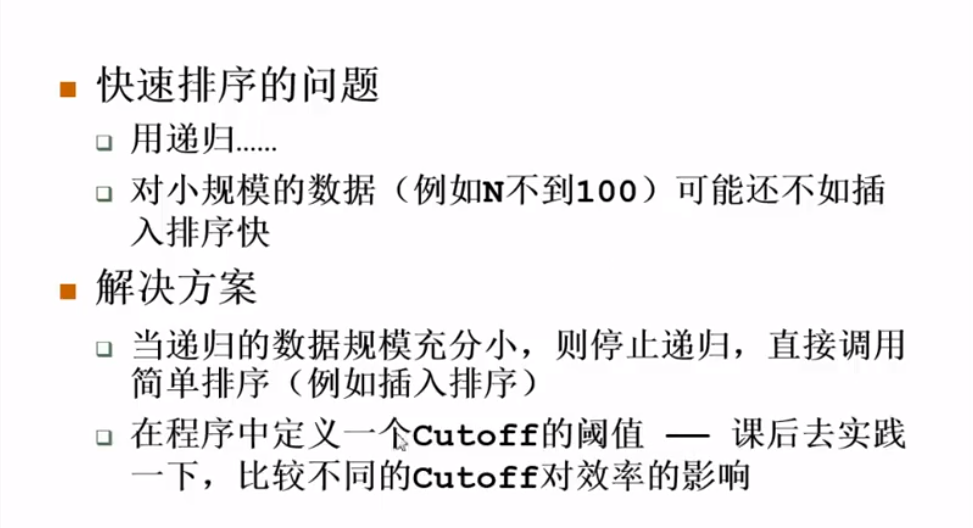
## 万一有元素与主元相同：

建议是；停下来做交换。

最后i、j会停在比较中间的部分。

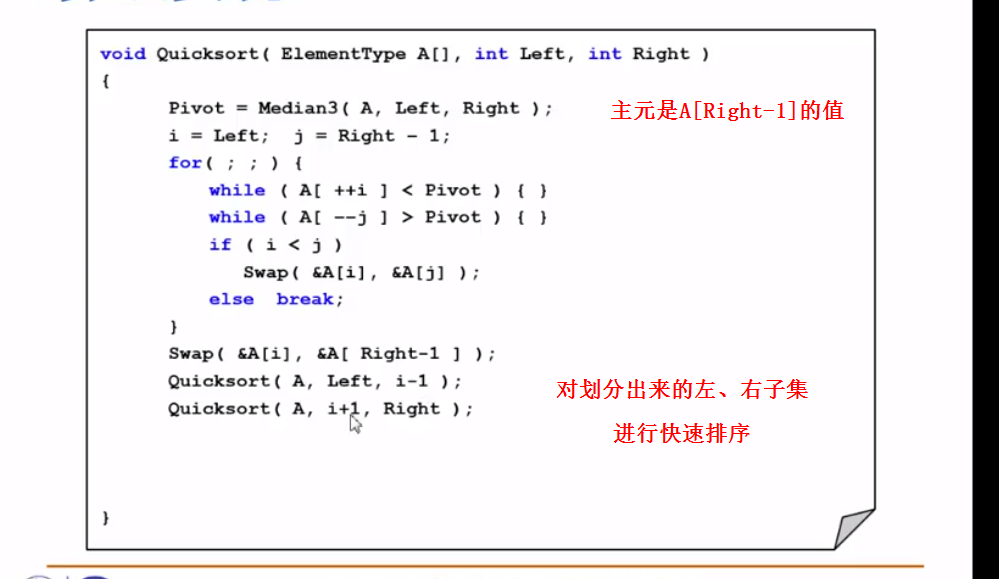
如果不理，继续移动指针：变成n平方的时间复杂度。

# 使用建议：

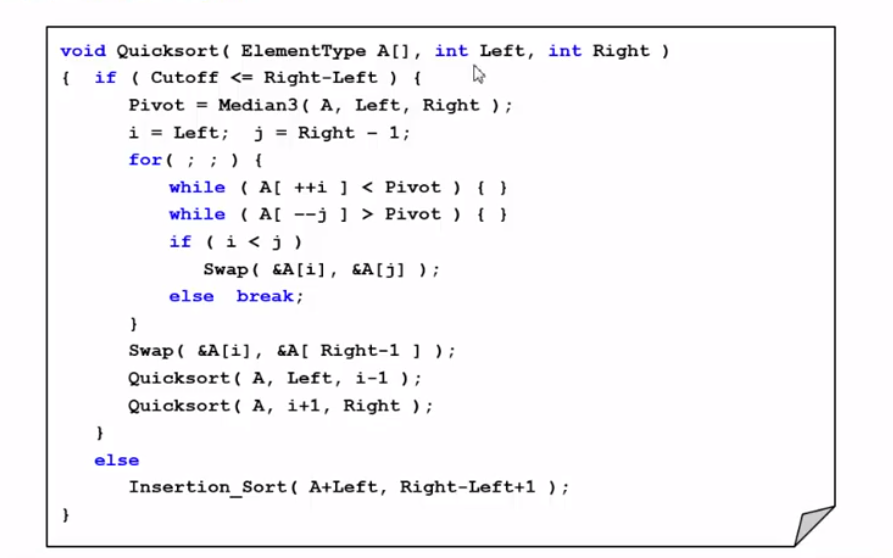


# 算法实现：【代码】

这段代码是少了一个判断，照抄肯定是得不出你想要的结果的。



## 优化：如果数据是很少的不如用插入排序



## 快速排序的标准接口写法：

