Divide and conquer

这个方法逻辑很简单



可以把它和master theorem联系起来

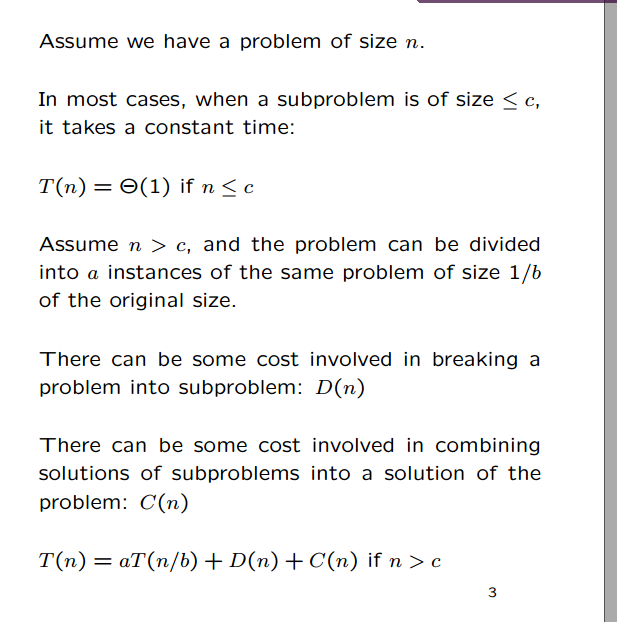
假设问题size为n

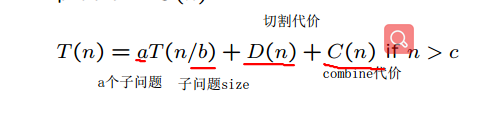
那么我们·将子问题的size切割到一个常数c以后，他需要的时间是固定的

假设每次问题都被切割成a份size为1/b原size的子问题（a个子问题，size为1/b）

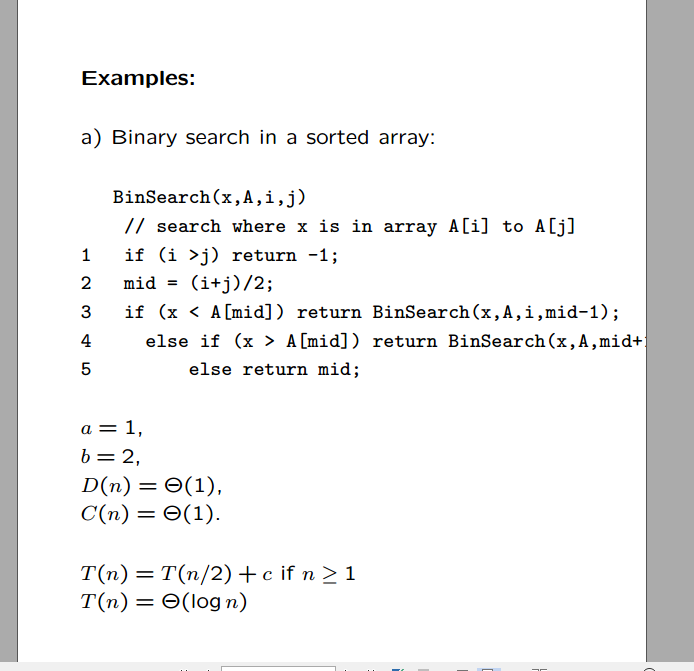
每次切割子问题所需要的代价为D（n）

把子问题合并成父问题所用代价为C（n）



那么你会

发现，这玩意儿的形式就是一个master theorem



前提这是一个sorted array

binary search，从i到j之间search x所在，

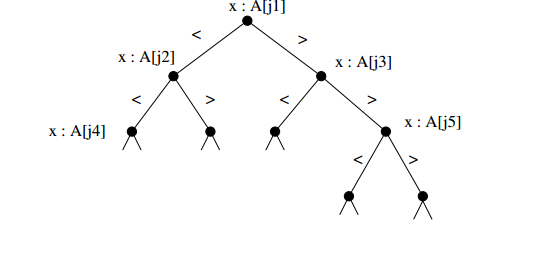
每次对半分，但只分成了一个子问题，另外一半我们不care

T（n）=T(n/2)+c

n^logba=1=f(n),第二种情况

Tn=lgn

Decision tree



每一个node代表array的一个值，如果小于，就往左边，如果大于就往右边，如果等于，那么就是你了

任意比较算法都可以用decision tree表示出来，例如刚刚那个二分法，，每一个node都代表一个中点

这棵树的depth代表了算法的最坏情况，

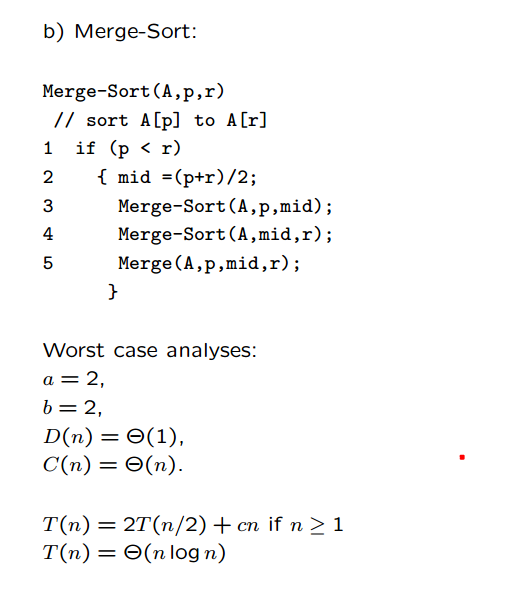
但是不管怎么比较，depth至少是Logn

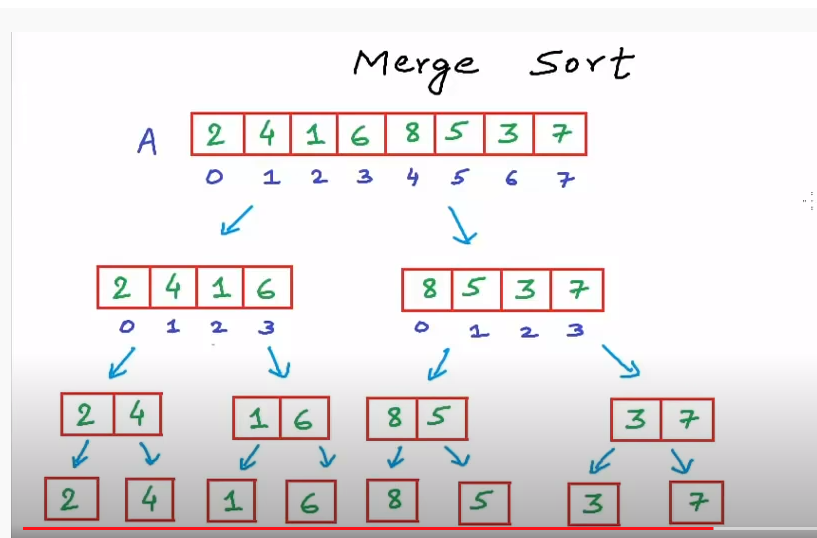
Binary search是一个optimal comparison-based search algorithm //optimal 优化的

最差情况是logn

hash并不是基于比较的，平均是O（1）,最差是O（n）

b/merge-sort

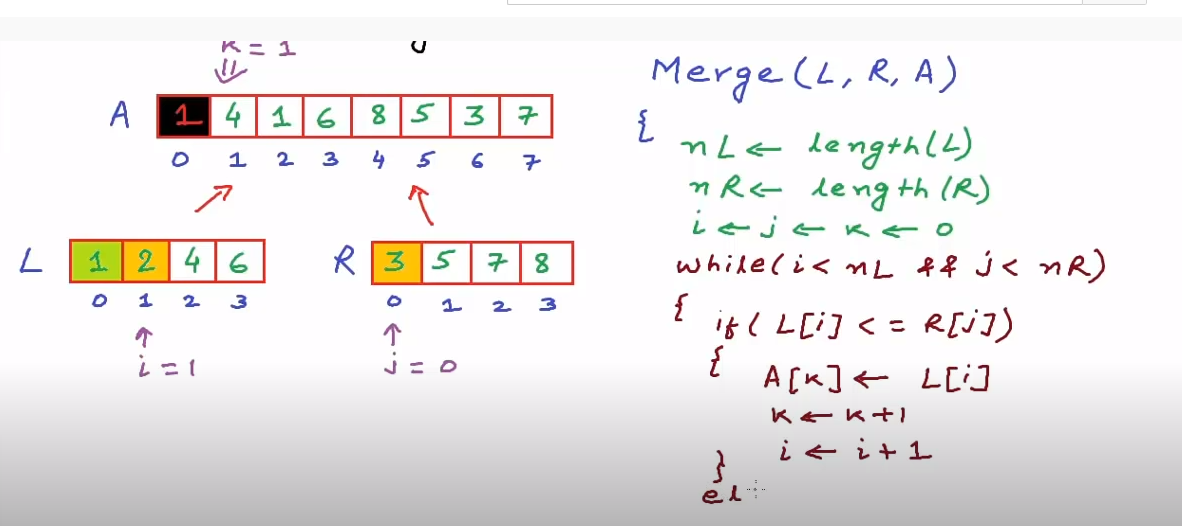


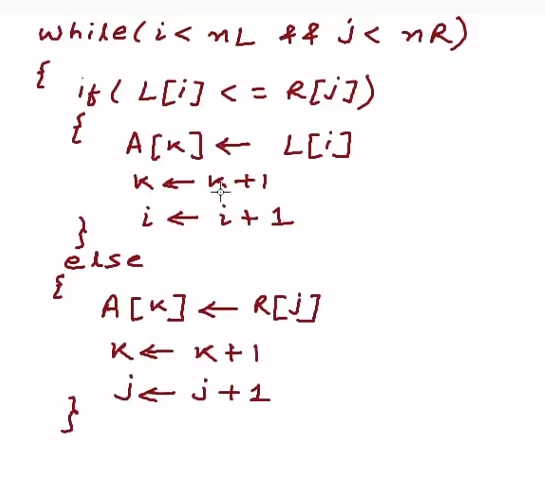


第一步对半分，一只分到只有一个

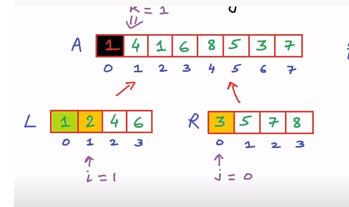
只有一个的时候自然是sorted

然后开始merge

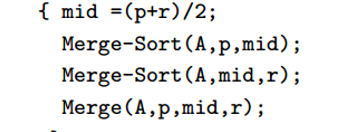




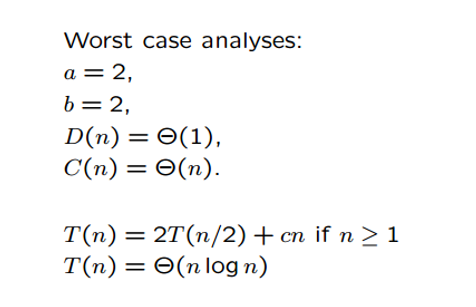
Merge需要三个参数，一个Left,一个Right,一个Array



Left Right各有一个pointer,当前pointer比较大小，小的那个右移一格并且填充到A当前pointer,就一直这样移

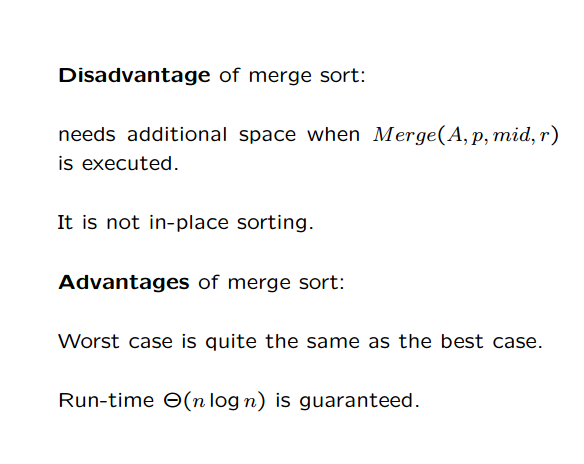


反正就是经过两个Merge-Sort,再Merge回来



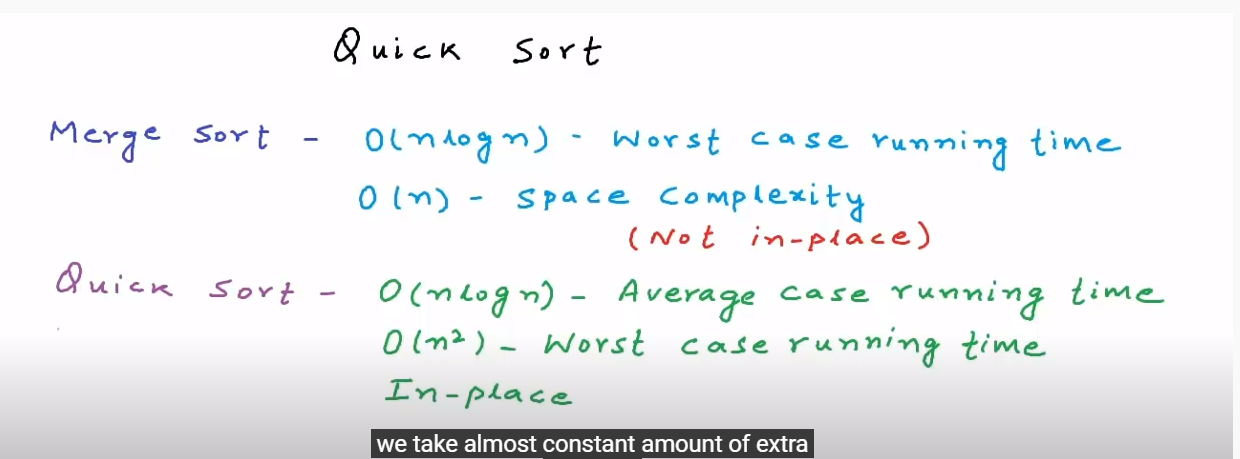
复杂度为nLogn， 分为两部分，然后merge部分复杂度也就cn

因此T(n)=nlogn



Merge Sort的缺点，需要额外的space，不是in-place sorting

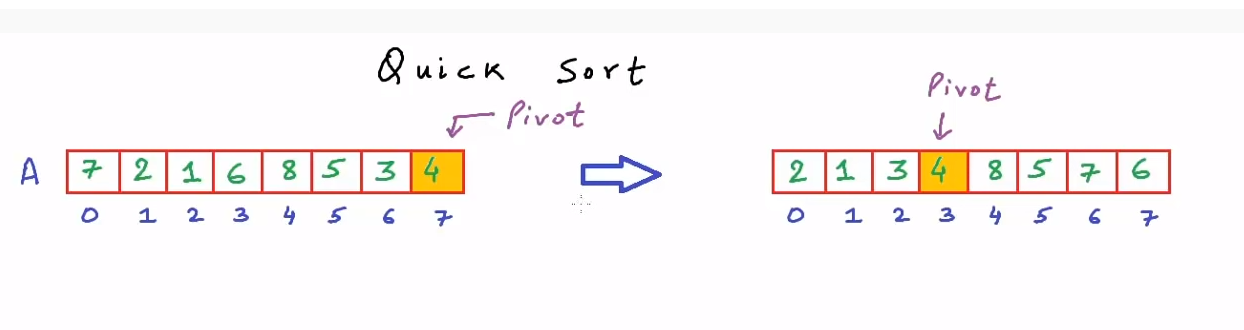
优势：nlogn稳定的运行速度



Quick sort worst case 是On^2，但是我们可以使用randomized quick sort, 几乎可以完全避免，

优势在于in-place，不需要额外的空间

randomized quick sort有很高几率做到nlogn



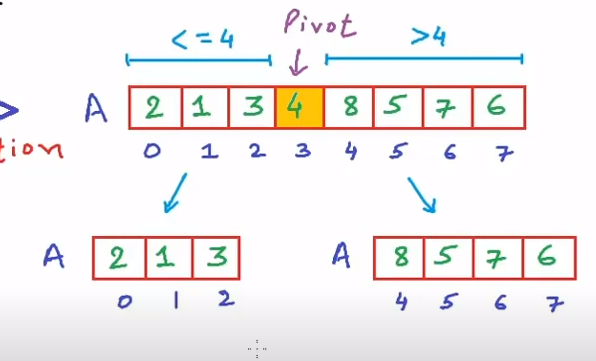
pivot,首先任选一个点作为pivot 轴，

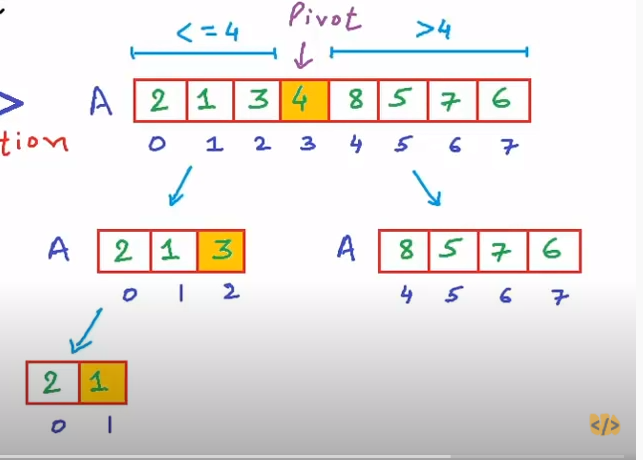
partition: 选好Pivot以后，小的放左边，大的放右边

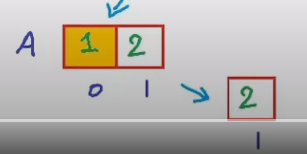
可以有多种partition，例如这里213可以改成231

然后开始recursion

注意：recursion的时候需要记住原index下标







注意partition这一部分是在进行下一步recursion之前做的

下一步recursion会挑选除了pivot以外的index

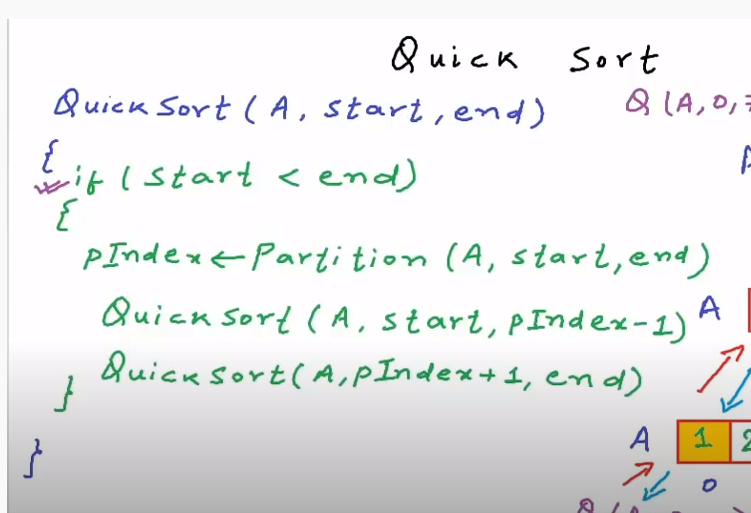
只有一个的时候就是base case

然后index=1就是2，

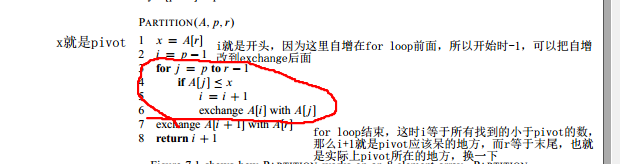
index 01就是12

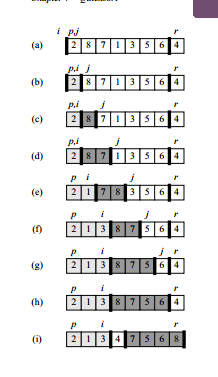
index 012 就是123

sort完成



第一步先partitioon,A代表原array,会return pivot的index





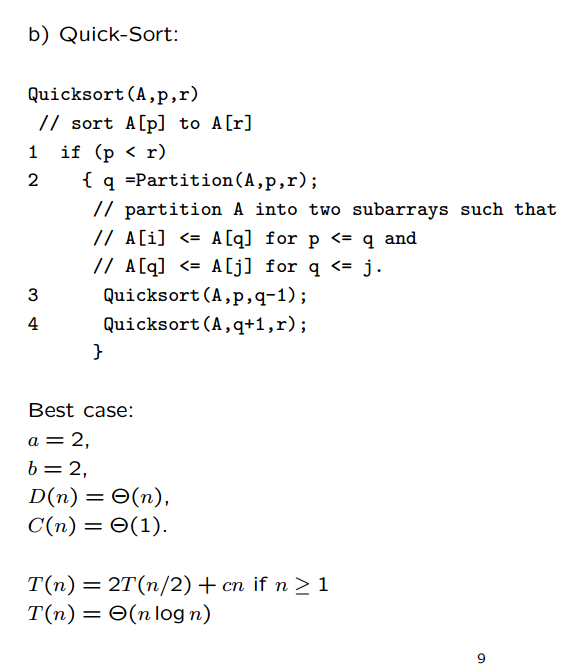
它将i与j换时，这时j的作用是当前index，i的作用是当前小于他的index

**因为j跑得比i快，i指的是所有小于r的，而j是自增的，那么r-i 自然就是大于r的，这也是这个算法精妙之处**

**最后把r到中间**

这里Pivot默认选取最后一个，如果小于pivot，那么把他换到第一个，第二个......

最后大的会排到pivot右边，小的排到左边

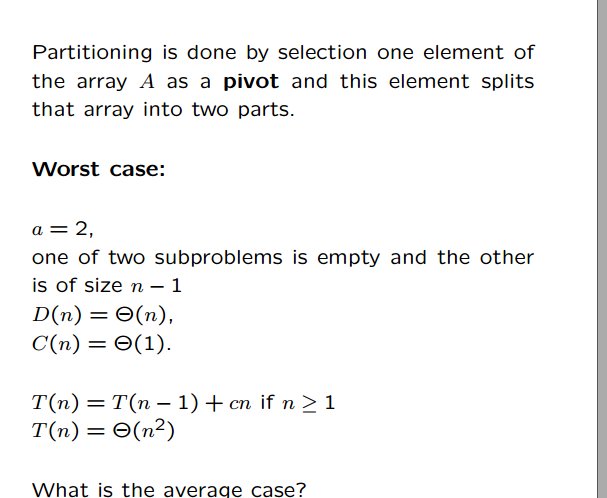


quicksort，就很标准，

分为两部分，size为一般

Partition部分复杂度显然为n

那他就是标准的nlogn

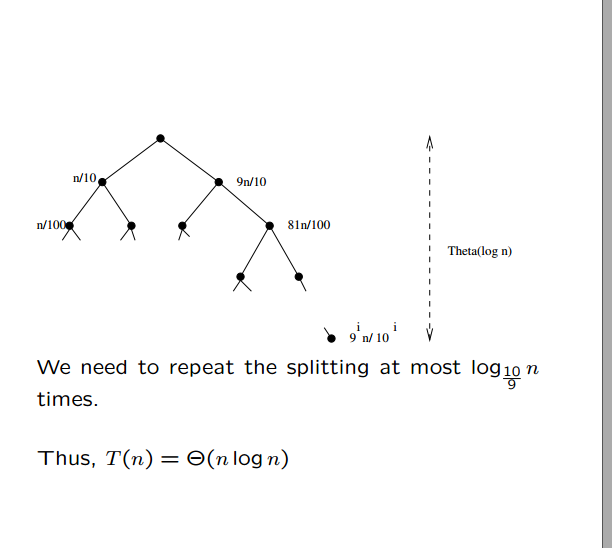


worst case

每次都选取最大值或最小值，那么他就没有把size切割成1/2，也没有切成两份

//it is already sorted or reverse sorted// 因为我们partition是取的结尾

//但是如果说左边切了9个，右边只剩一个，我们还认为他是nlogn



他是一个recursion tree， 每层的代价和都是n，直到左边的已经到底部（因为左边明显下降的更快，每次剩1/10,所以depth低） 右边高度为log10/9 n

那么总Tn小于cn\*log10/9 n

在算法中，也就是nlogn

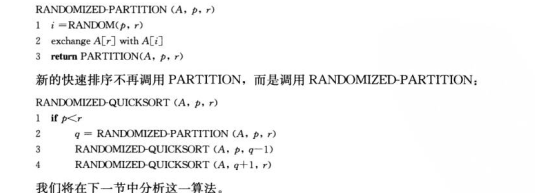
7.3

randomized quick sort

如同上文所说，如果我们总取第一个或最后一个为pivot，那么面对sorted array,效率最差，是n^2

那么，我们可以通过随机选取一个Pivot，或者打乱array，来尽量达到nlogn

可能运气特别特别差，每次都选到最大值，那还是n^2,但即使是这样，问题也是出在随机，我们不必再依赖输入的array



我们先随机p到r之间一个数，把他换到末尾，再进行原来的partition