分治算法的设计

(1) 三个阶段：

divide（分）

conquer（治）

merge（合）

(2) 归并排序（严格的对半分）

(3) 快速排序：大子区，小于区。

(4) 比较算法的下界 $n\log n$

（这个用决策树分析）

n个元素有 n! 种排列方程，

基于比较的话，

一半排列方式是 $<a,b>$，且 $a\le b$

另一半是：$<b,a>$ 且 $a\le b$

上述只有一种成立，

每次决策只选取其中一半 不严格

决策树高 $h = \log n! \leq n\log n$

对于 $n!$ 的放缩，Stirling 近似

$$n! \simeq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

(5) 找 max - min，最小比较次数

可用分治：

$ans = \min k(arr[0, \frac{n}{2}], k)$

或 $\min k(arr[\frac{n}{2}, n], k-\frac{n}{2})$

这里也用到了快排的划分策略。

分治：左半边找 max, min

1) base case：high - low = 1 窗口只覆盖了两个元素 → 返回其中的 max 与 min

2) merge：左右两边比较，找 max, min

过程:

```
1.  IF high-low = 1 THEN
2.        IF  A[low] < A[high] THEN  return (A[low], A[high])
3.      ELSE  return (A[high], A[low])
4.  ELSE
5.        mid ← (low+high)/2
6.        (x1,y1) ← Max-min(A, low,mid)
7.        (x2,y2) ← Max-min(A, mid+1,high)
8.        x ← min{x1,x2}
9.        y ← max{y1,y2}
10.       return (x,y)
```

conquer

divide

divide and-conquer

merge

(b) 大数乘法,

master定理 $T(n) = aT(\frac{n}{b}) + f(n)$

这里是通过减少重复计算, 降低 $n^{\log_b a}$

$X = A << \frac{n}{2} + B$

$Y = C << \frac{n}{2} + D$

$XY = AC << n +$

$((A-B)(D-C)+AC+BD) << \frac{n}{2} + BD$

(7) 模型覆盖问题：

分治：要处理相同的子问题

⇒ 构造出来

(8) 中位数问题 bfprt

主要解决 pivot 的选取.

非严格的快排.

找 left 第 x 小，找 right 第 k-len(left) 小

pivot 选取步骤：

每5个一组，取中位数得 mediums = [...]

在 mediums 中找中位数（递归3）

$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + O(n)$

在left找x，在right找k-len(k)

⇒ $T(n) = O(n)$

① 分组 ② 排序 ③ 中之王者 ④ 分 ⑤ 递归

快排的partition

引申题：

(1) 递归和分析归并排序

(2) 二分搜索

(3) 最中间的 k 个数

1) bfprt 算法，找到中位数 mid

2) distance_arr 是 arr[i] 与 mid 距离

3) bfprt 算法，找到 distance_arr 中第 k 小的数，表示符合要求的最长距离 pivot

4) 遍历 arr，如果 $|arr[i] - mid| \le k$

⇒ ans.append(arr[i])

(4) 两个数组的中位数

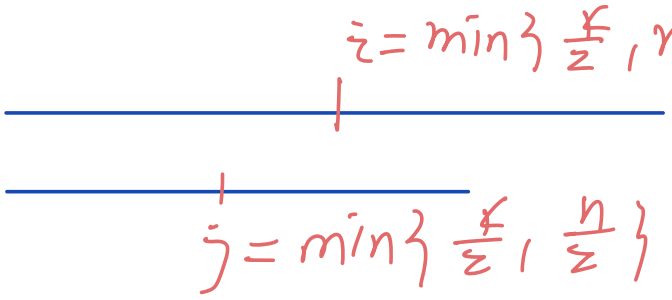
arr1, arr2 都是有序的

m n

找第 k 个数

这题主要难点在划分，

假设 arr1 比 arr2 长，



如果 arr1[i] < arr[j]

⇒ arr1 = arr1[i:]

return findk(arr1, arr2, k-i)

else

....

证明：

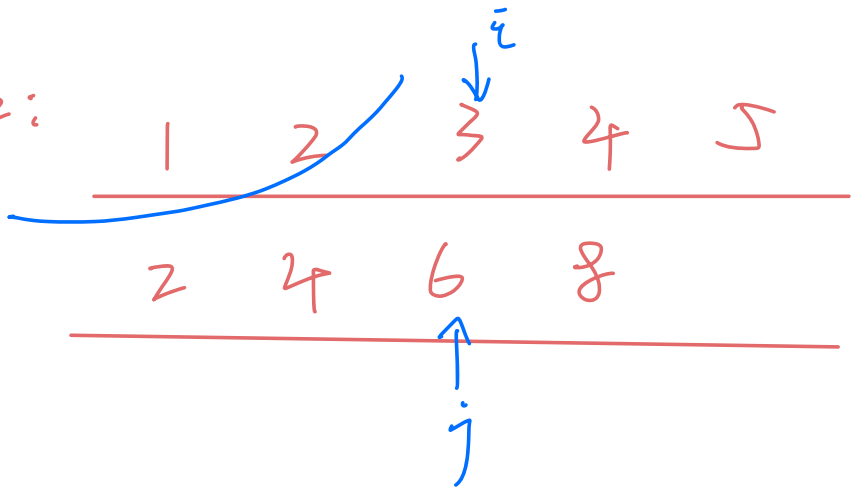
1) base case ⇒

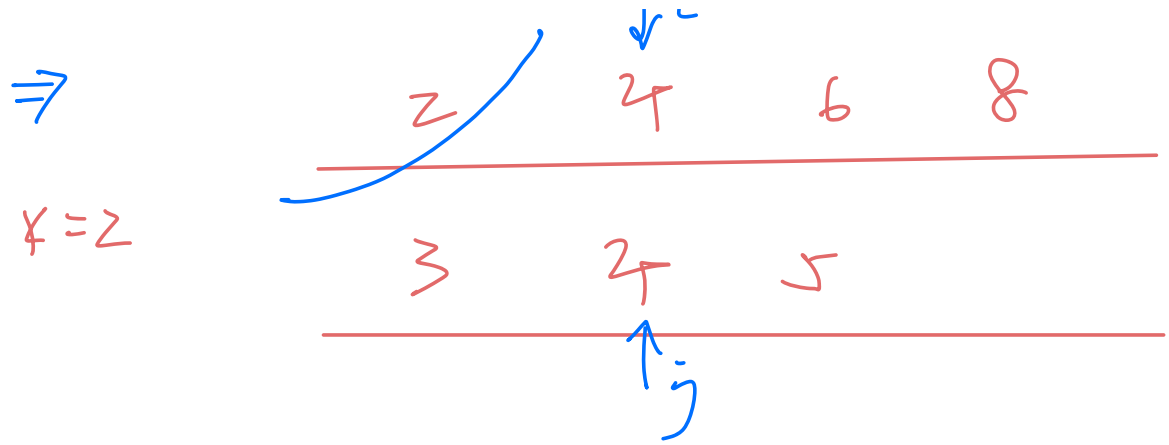
k==1, return min{arr1[0]
arr2[0]}

2) 如果砍掉一半，不是第k小的？

例如：

k=4





$\Rightarrow$ $k=2$

$\Rightarrow$ 4 6 8

3 4 5

$k=1$

$\Rightarrow$ ③ 然而3也正是所求

(2) 找到频数第k大的元素：

先用一个 hash表(字典)/counter

对元素进行频数统计

(key, value) = (key, frequence)

然后对 key 进行快速排序，

排序使用二元谓词 (return a.fre <= b.fre)

我们知道，快速排序，每次可以确定一个元素的位置，也就是pivot的位置。

并且我们这里，位置先后与顺序相关

如果某个pivot，最后落在K位置，

那么锁定结果

```
index ← partition( l, r, pivot )
if   index - start > k : 砍掉右边
    qsort( v, start, index-1, k)
```

1) index落在K右边

2) index落在K左边

分治与砍掉