# 研究生算法课课堂笔记

**上课日期：2014.11.24　　第( 3 )节课**

**组长：安博　　组员：刘卢琛**

**组员：李睢　　组员：林泽琦**

**组员：施晨**

注意：请提交 Word 格式文档。

---

## 第一题

本题为“**寻找绝对众数**”问题。

注意题中只能测试两张卡是否等价，不能比较大小。如果允许比较大小，则将所有卡快排之后扫描一遍即可，复杂度为 O(n log n)。

题中要求给出 O(n log n)与 O(n)两种复杂度的解法。其中 O(n log n)的解法相当于自顶向下的归并，而 O(n)则对应自底向上的归并。以下分别给出这两种解法：

**O(n log n)解法：**

该解法核心基于以下事实：如果一个数组中存在绝对众数，则将该数组均分为两个子数组后，原来的绝对众数至少在其中一个子数组中依然是绝对众数。注意该命题反过来并不成立，即产生的两个子数组有绝对众数的情况下，原数组不一定有绝对众数，除非这两个子数组都有相同的绝对众数。

依据以上事实，采取分治思想，设当前层银行卡总数为 n，将 n 均分两半，分别在其中寻找绝对众数。若存在（设为 a, b），则若 n 中存在绝对众数一定是 a, b 之一。此时只要分别对 a, b 在 n 张卡中进行比较，看是否超过半数，即可确定其是否为 n 张卡中的绝对众数。

具体操作时，可以采取归并方法，即对原始卡集不断二分，递归寻找每个子集的绝对众数（到最底一层只剩两张卡时，相同则为该层绝对众数，不同则该层绝对众数不存在）。这样便可递归得到整个卡集的绝对众数。

时间复杂度：

由于二分过程中每层的比较代价为 2n（a, b 各比较 n 次），所以递推式为 T(n) <= 2*T(n/2) + O(n)。根据主定理，T(n) = O(n log n)。

这里需要注意的是，每次在子数组中得到绝对众数时，只需要在当前的数组中扫描一遍，不要每次都扫描整个原始数组。

**O(n)解法：**

在每一轮中将所有银行卡两两配对进行比较，如果两张卡等价则留下其中一张，不等价则全部丢弃。重复以上操作直到最终留下一张卡。将这张卡与其余所有卡比较，若与超过半数的卡一样则其为绝对众数。否则绝对众数不存在。

该解法证明如下：由于每次比较中或是一张绝对众数卡消耗掉一张非绝对众数卡，或是两张绝对众数卡中有一张能留下，故每轮操作后绝对众数卡所占的比例不会降至一半以下。故如果在原始卡集中绝对众数卡存在，则它一定能留到最后。这样只需要将最后一张卡与其余所有卡进行比较，看它是否超过了

半数即可。

**需要特别注意的是，如果当前轮卡数为奇数**，则剩余的一张卡需要与该轮中所有卡比较一次以判断其是否为该轮的绝对众数，以决定是否丢弃之。**直接将该卡保留或者丢弃都是错误的做法。**如果直接保留，则有以下反例：1, 1, 1, 1, 2, 2, 2，第一轮后变为 1, 1, 2, 2，第二轮后变为 1, 2，最底层不存在绝对众数，而原始数组中绝对众数为 1，矛盾；如果直接丢弃，则有以下反例：1, 1, 2, 2, 2，第一轮后变为 1, 2，最底层不存在绝对众数，但原始数组中绝对众数为 2，矛盾。

时间复杂度：

由于每轮操作中比较代价为 O(n)且卡集规模减半，故递推式为 T(n) <= T(n/2) + O(n)。根据主定理，T(n) = O(n)。

**另一个 O(n)算法：**

算法思路是：从数组中取不同的两个数，就把他们配对并移出数组，这样最后剩下很多相同的数不能配对。我们维护已处理过的元素中（即 A[1,i]中）还没有配对的元素值 v 和未配对的数量 count。

```
Count = 0
For i = 1 to n:
    If count = 0: //count = 0 说明没有未配对的元素
            V = A[i]
            count++
            continue
    If A[i] = v:
            count ++;
    Else
            count --;
//验证
Compare v with each element in A
if number > n/2:
    return v
else
            majority number does not exist.
```

正确性：

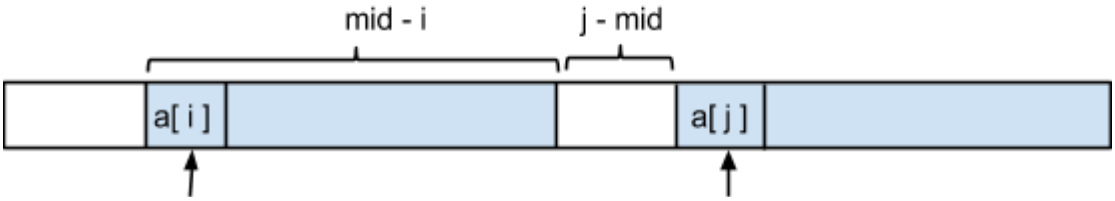
如果大众数 v 存在，每次配对只能消耗 1 个或 0 个 v，而配对次数最多只能有 n/2 次，所以一共最多消耗 n/2 个 v。剩下的不能配对的数就一定是 v。之后用 v 和数组其他元素比较一次，确认出现次数是否超过 n/2 即可。（这是由于，如果大众数不存在，算法也可以找到某个 v，算法找到的数不是大众数的充要条件，需要验证）

算法配对和验证都只需要线性扫描，总时间复杂度为 O(n)。

算法只需要常数的辅助空间，空间复杂度为 O(1 )。

## 第二题

1. 首先考虑普通的逆序对算法




在如图所示的归并的过程中,

(1) 若 a[i] < a[j], 输出 a[i], i++, a[i]与剩余的元素(图中蓝色)不形成任何逆序对,

(2) 若 a[i] > a[j], 输出 a[j], j++, a[j]与剩余的元素构成多少逆序对呢?
是 j - i 个吗? 不是.
后面的 j - mid 个元素已经被输出了, 在输出的时候他们的逆序对已经被算过, 因此这里不必再考虑. 所以只计算 mid - i 个

2. 作业中的显著逆序对算法

老师认为在每次归并过程中需要进行两遍扫描, 一遍扫描的几乎没人做对.

第一遍扫描用上面的方法统计重要逆序对个数, 即

若 a[i] < 2 * a[j], i++, 没有重要逆序对

若 a[i] > 2 * a[j], j++, 重要逆序对数量增加 mid - i 个

第二遍扫描再进行正常的归并.

同学发言给出的方法:

扫描的过程中在左边的一半数组里同时保留两个指针,

一个归并指针 i, 一个重要逆序对的指针 i'.

在扫描的过程中

若 a[i] < a[j], i++, 输出 a[i], 没有重要逆序对

若 a[i] > a[j], 首先找到 2*a[j]的位置,

即 while ((a[i'] < 2 * a[j] ) and (i' < mid)) i'++

然后重要逆序对数量增加 mid - i'个

最后 j++, 输出 a[j]

另外顺便提一下我自己作业中用的一遍扫描的方法:

维护一个队列来存储待与 a[i]比较的所有 2*a[j]的值,

每次 i++之后将新的 a[i]与队首元素比较, 如果队首元素小于 a[i]则重要逆序个数 mid - i 并队首元素出队, 重复比较直至队首元素大于 a[i]为止.

---

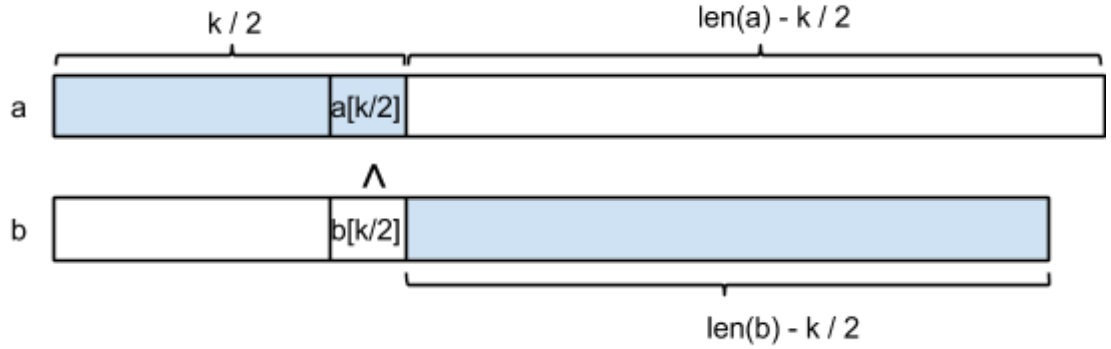
## 第三题

考虑更一般的问题:

(1) 两个数组的长度不相等?

(2) 需要寻找的不是中位数而是第 k 个的元素?

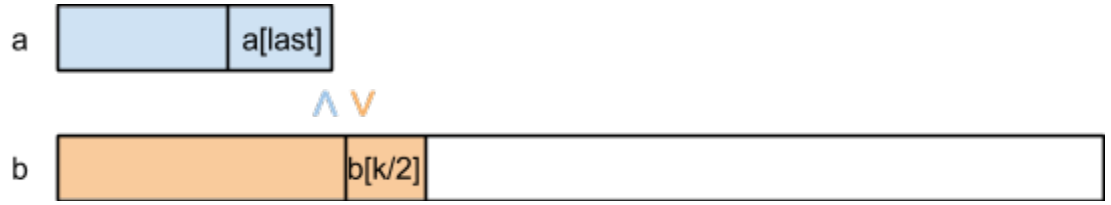
由于问题的规模主要是和 k 相关, 因此两个数组不等长是无所谓的.




寻找两个数组中的第 k 个的元素的算法:

首先比较两个数组的第 k / 2 个元素, 如图所示, 不妨设 b 数组的第 k / 2 个元素更大, 则此时 a 数组的前 k / 2 个元素都至少小于 (len(a) - k/2) + (len(b) - k / 2) = len(a) + len(b) - k 个元素, 因此必在最小的 k 个元素之中. 类似的, b 数组的后 len(b) - k / 2 个元素都至少大于 2 * (k / 2) = k 个元素, 因此必然不在最小的 k 个元素中. 所以我们可以在数组中去掉图中蓝色区域的元素, 然后再递归地考虑两个数组的剩下元素(白色区域)的第 k / 2 大的元素即可.

特殊情形 : 一个数组的不够长, 取不到第 k / 2 个元素的情况?




(1) 这个问题只有 k > n 时才会出现. (n 为数组的初始长度)
(2) 如果出现了,
    (a) 较大数组应该仍然能够取到所需的数
    (b) 较小的数组取最后一个即可
(3) 数组不够长的问题会不会频繁出现而影响复杂度呢? - 不会.
    (a) 如果较小数组的最后一个元素小于大数组的第 k / 2 个元素的话, 小数组全取, 此时递归已结束. 这种情况最多出现一次.
    (b) 如果较小数组的最后一个元素大于大数组的第 k / 2 个元素的话, 大数组的前 k / 2 个元素全取, 问题规模仍减半, 不影响复杂度.
(4) 考虑 k > n 的情形是不是无意义的? 即在这种情形下 2n - k < n 我们可以考虑倒数第 2n - k 个元素, 这个是不是说我们不需要考虑 k > n 的情形呢?
    不是, 我们不愿意写两套代码一个找第 k 小, 一个找第 k 大, 太麻烦.

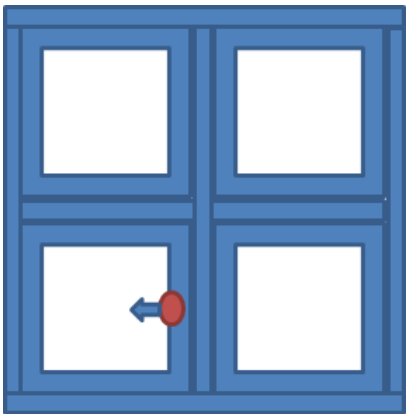
---

**第四题**

1. 典型解法和错误

按照习题解答的标准答案。在如图所示的“田”字框找最小值，如果它是

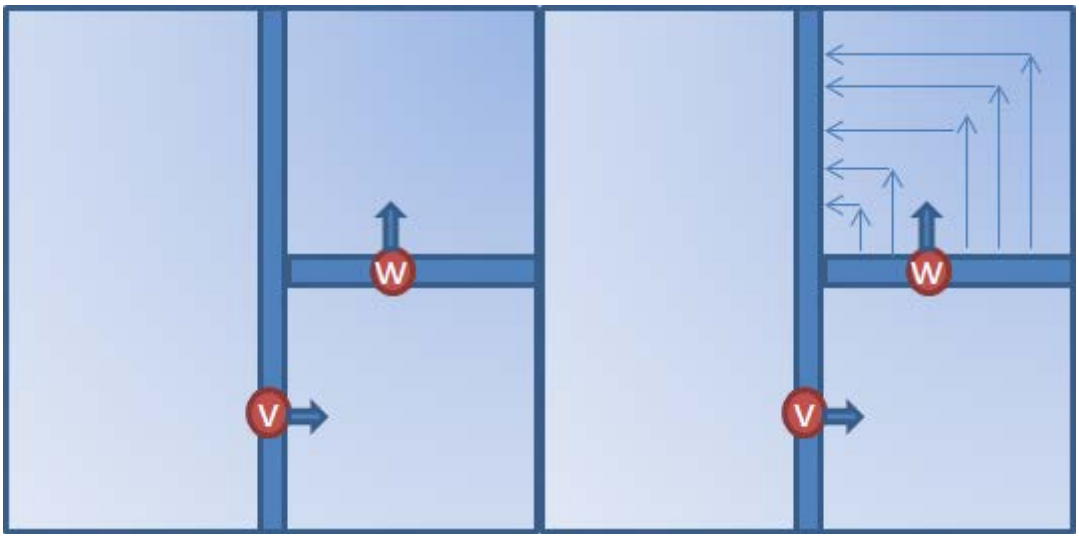
极小值则返回，否则跳到他相邻格子的小区域中递归寻找极小值。



典型错误是：在四个区域都递归寻找极小值 T(n)=4T(n/2)+O(n)，或在两个地方递归寻找极小值 T(n)=2T(n/2)+O(n)，由主定理，复杂度分别是 O(n^2)和 O(n log n)，不符合题目要求。

2. 一个简单但错误的解法

如图，先竖切一刀，由最小值 v，再在右边矩形中，横切一刀，找到最小值 w，跳到比 w 小的上面矩形中。两次递归，问题规模缩小一半，3n/2 向下取整次比较，比原来 14n 次比较，在常数上好得多。




但是 w 所在的边框的值可能都很大，v 所在的边框值都很小，这样右上角的矩形中可能没有极小值。比如，数据分布如图所示，箭头表示相邻数据由大到小的方向。

3. 关键性质分析

（回顾）图中如果允许数据相等，那必须要 O(n^2)的时间复杂度。因为如果除了某个点其余点值都相等，在没有额外信息时，为了找到这个点，必须把所有点都扫描一遍才能被找到。

分治过程要保证子问题有解：

（1）原问题有解。因为图中的最小值，就是极小值。

（2）为了保证分治子问题有解，子区域的最小值不能在人为分割开的边上。这样即使这个边上的点是子区域的极小值，不能保证它比人为分割

的边外相邻的点要小。

（3）从算法设计的角度，我们维护这样一个性质：**所选的子区域中包含一个点，这个点比这个区域，由人为分割产生的边框外相邻的点都要小。**这样，这个区域的最小值一定比边框外的相邻点要小，它即使在边上，也是全图的极小值。

（4）上面性质保证子问题有解，还有一种理解方式：

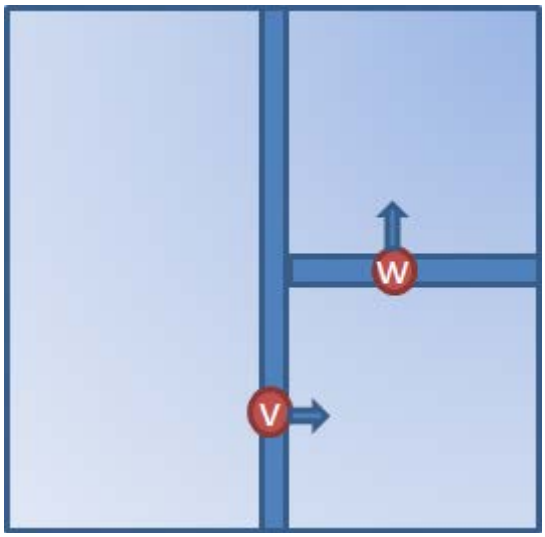
设 v 比这个区域人为分割产生的边界点都要小，从点 v 开始，循环进行如下操作：跳到比他小的邻域网格上。由于 v 的值严格地比边界小，这个跳转过程严格限制在该区域内，而由于不会走相同的点，这个过程在有限步就会完成。而最终停止的点一定是这个区域中的极小值点。

4. 正确的更简单的解法

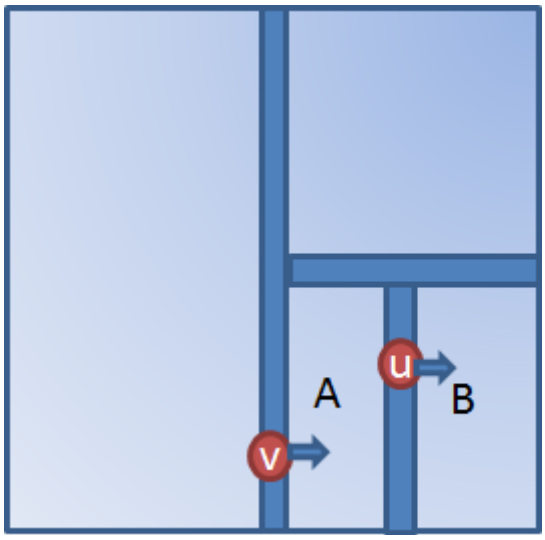
维护一个所有比较过的元素的最小值 globle_min 和其位置，每次在进行划分时，找出新的人为切割的边界上的点和 globle_min 中，选择一个最小值，跳到它相邻的点所在的区域内，递归求解子问题。

可以发现，globle_min 就是当前考察区域边界上的最小值，它的位置一定处在当前考察区域的边界，这样，上述解法满足了 3 中的关键性质。

举例说明：




比如在上面描述的情况下，现在横着切一刀，因为 w 比 globel_min（即 v）大，应当对右下角方格递归求解，此时的 globel_min 还是 v。




在右下角区域中，竖着切一刀的最小值为 u。如果 v 依然比 u 小，应当跳入 A 区域，globel_min 还是 v；否则根据 u 的邻域，跳入 B 区域，这时 globel_min 是 u

---

**第五题**

1. 递推方程式：$T(n) = T(3)T(n/3) = kT(n/3)$。含义如下：我们将规模为 n 的矩阵自底向上划分成 n/3*n/3 个 3*3 的小矩阵，把一个小矩阵看为一个元素,此时矩阵规模变为 n/3，每个小矩阵间计算的时间代价为 T(3)，因此总的时间代价为$T(3)T(n/3)$。由主定理知：复杂度为$T(n) = O(n^{log_3 k})$
2. $\log_3 \mathrm{k} < \log_2 7$，则$\mathrm{k} < 3^{\log_2 7} = 21.8499$，则 k 最大为 21。