Course 1 Week 2

这周的内容主要是再介绍了几个Divide and Conquer的例子，最后给出了分析时间复杂度的Master Method。

Part 1:分治算法的例子

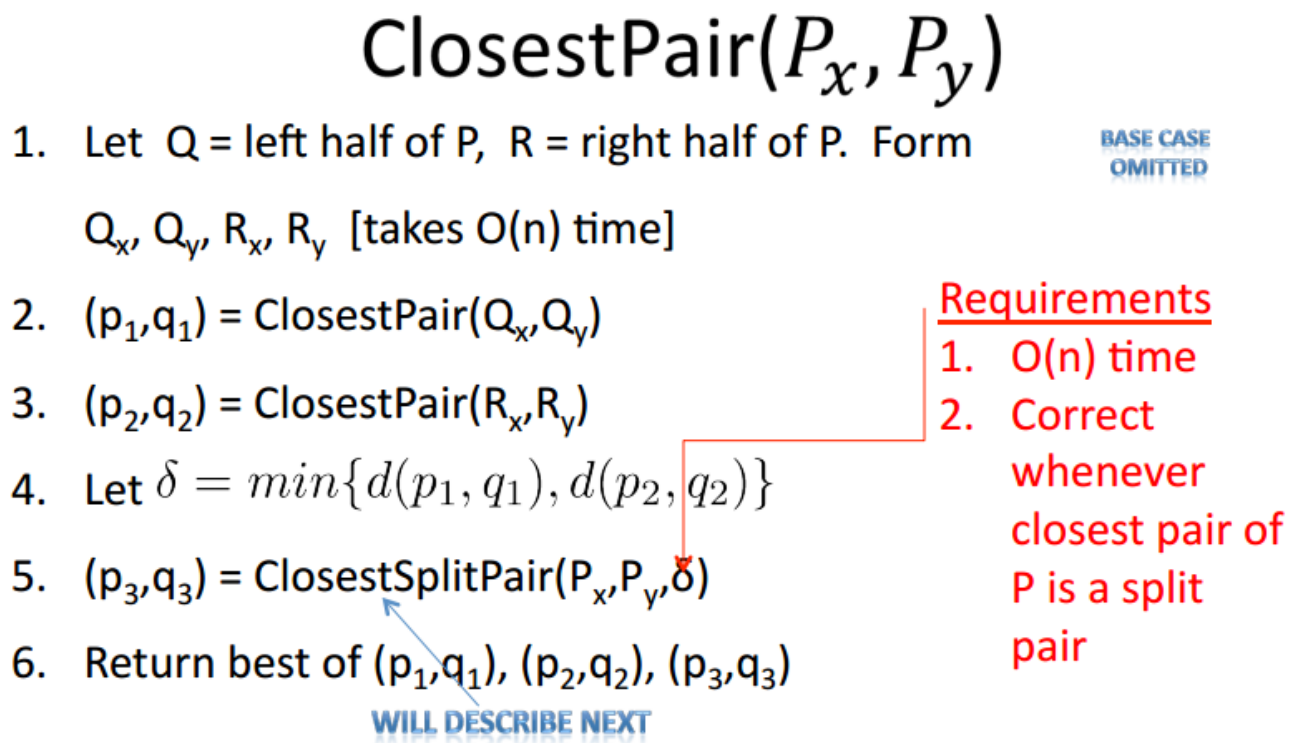
The Closest Pair Problem

- Input:R^2中n个点$P = \{p_1, \ldots, p_n\}$
- Notation:$d(x_i, x_j)$为欧几里得距离
- Output:P中两个距离最近的点$*p, *q$

这里我们假设每个点的横坐标，纵坐标均不相同。

这个问题的一维情形可以用排序解决，时间复杂度为$O(n \log n)$，二维情形如果暴力求解，那么时间复杂度为$\theta(n^2)$，现在的目标是把二维情形的时间复杂度降低到$O(n \log n)$。

我们先把这些点按x排序为P_x，按y排序为P_y，这一步需要$O(n \log n)$的时间，但这样做显然是不够的，后面还需要Divide+Conquer，伪代码如下：

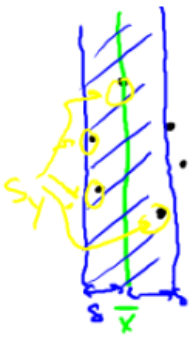


这个算法的关键是第5步，如果这个步骤只需要花$O(n)$的时间，那么由上一周的讨论我们知道，这个算法总共需要的时间为$O(n \log n)$，所以现在问题就转换为ClosestSplitPair的问题，这里老师先给出算法：

ClosestSplitPair(P_x, P_y, δ)

Let $\bar{x}$ = biggest x-coordinate in left of P. (O(1) time)

Let S_y = points of P with x-coordinate in
Sorted by y-coordinate (O(n) time)



Initialize best = δ, best pair = NULL
For i = 1 to $|S_y| - 7$
  For j = 1 to 7
    Let p,q = i^{th}, $(i+j)^{th}$ points of S_y
    If d(p,q) < best
      best pair = (p,q), best = d(p,q)

O(1) time

O(n) time

At end return best pair

补充一下，上图中的S_y的描述有误，应该为横坐标$x \in [\bar{x} - \delta, \bar{x} + \delta]$的点按照纵坐标$y$排序之后的点集。

这个算法最神奇的地方在于，第二重循环只要执行7次，所以保证了时间复杂度为$O(n)$，下面来解释下这个算法，有如下命题：

令$p \in Q, q \in R$为$d(p,q) < \delta$的split pair，那么

- (A) $p, q \in S_y$
- (B)p, q在S_y中最多距离7位

令$p = (x_1, y_1), q = (x_2, y_2)$

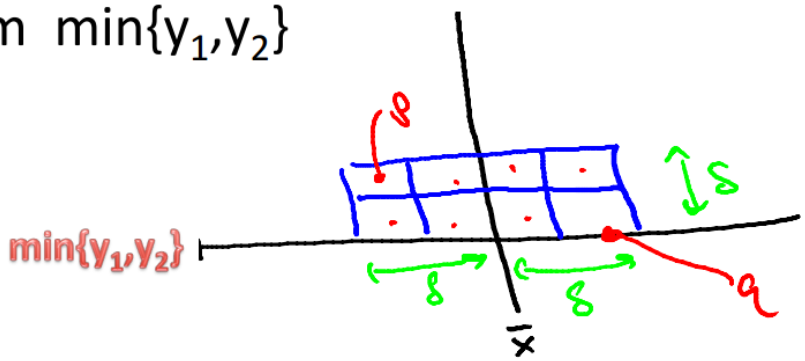
命题(A)的证明：

因为$p \in Q, q \in R$，所以$x_1 \le \bar{x}, x_2 \ge \bar{x}$，而$|x_1 - x_2| \le d(p,q) < \delta$，因此$x_1, x_2 \in [\bar{x} - \delta, \bar{x} + \delta]$

命题(B)的证明：

先画出以下关键图形：

Key Picture : draw $\delta/2 \times \delta/2$ boxes with center $\bar{x}$ and bottom min{y_1,y_2}



证明命题（B）需要以下两个引理：

引理1:S_y中纵坐标范围介于p, q纵坐标之间的点都落在这8个方格之内。

引理2:1个方格内最多有1个点。

引理1的证明：

首先$|y_1 - y_2| \leq d(p,q) < \delta$，说明满足条件的点的纵坐标落在方格的纵坐标范围内，其次S_y中的点横坐标的范围为$[\overline{x} - \delta, \overline{x} + \delta]$，结合以上两点，满足条件的点都落在这8个方格之内。

引理2的证明：

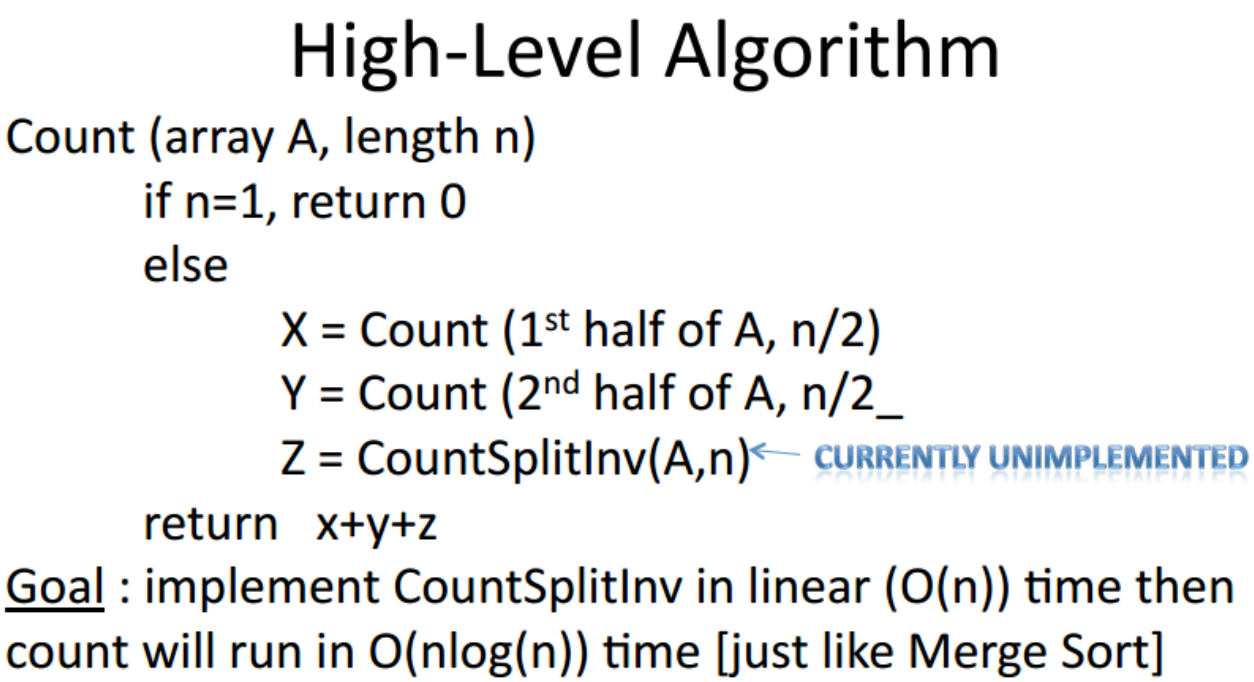
反证法，如果有两个点落在同1个方格内，或者这两个点都属于Q，或者都属于R，无论那种，这两个点的距离小于等与正方形的对角线的长度$\frac{\sqrt{2}\delta}{2} < \delta$，这与$\delta$的定义相矛盾，所以引理2成立。

结合这两个引理可知如果split pair中两点距离小于δ，那么这两个点都属于S_y，并且S_y中的下标差不会超过7，从而算法是正确的。

Counting Inversions

- Input: 含有数字1,2,...,n任意顺序的数组
- 逆序数(inversions)的数量，即满足$(i,j), i < j, a[i] > a[j]$的二元组的数量

这个问题还是用Divide and Conquer，有如下算法



这个算法最重要的是解决CountSplitInv部分，即计算$i \leq \frac{n}{2} \leq j$，但是$a[i] > a[j]$的数量，和之前的算法一样，我们的目标是希望在$O(n)$的时间内解决这个问题，解决之后就可以使得该算法的运行时间为$O(n \log n)$。

但是上述算法可能无法很好地解决该问题，所以这里老师给出一个更强的算法，在计算数量的同时给数组排序，如下：

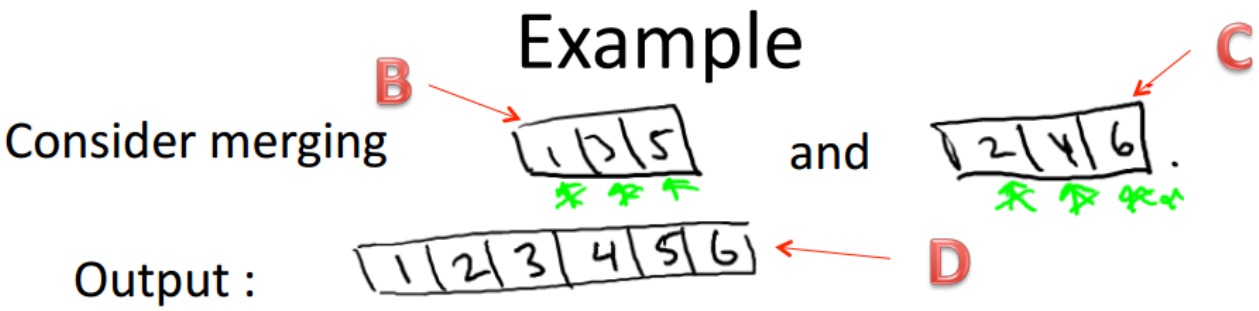
High-Level Algorithm (revised)

Sort-and-Count (array A, length n)
if n=1, return 0
else
(B,X) = Sort-and-Count(1st half of A, n/2)
(C,Y) = Sort-and-Count(2nd half of A, n/2)
(D,Z) = CountSplitInv(A,n)
return X+Y+Z

Sorted version of 1st half
Sorted version of 2nd half
Sorted version of A
CURRENTLY UNIMPLEMENTED

Goal : implement CountSplitInv in linear (O(n)) time
=> then Count will run in O(nlog(n)) time [just like Merge Sort]

由B,C推出D的过程为merge，操作和归并排序里一样，以一个具体例子来观察merge操作和逆序数数量的关系



⇒When 2 copied to output, discover the split inversions (3,2) and (5,2)
⇒ when 4 copied to output, discover (5,4)

将这个过程总结一下，有如下命题：

split inversions的逆序数量为将第二段数组C中元素y复制到D时B中剩余元素之和。

证明：

令x为第一段数组B中的元素

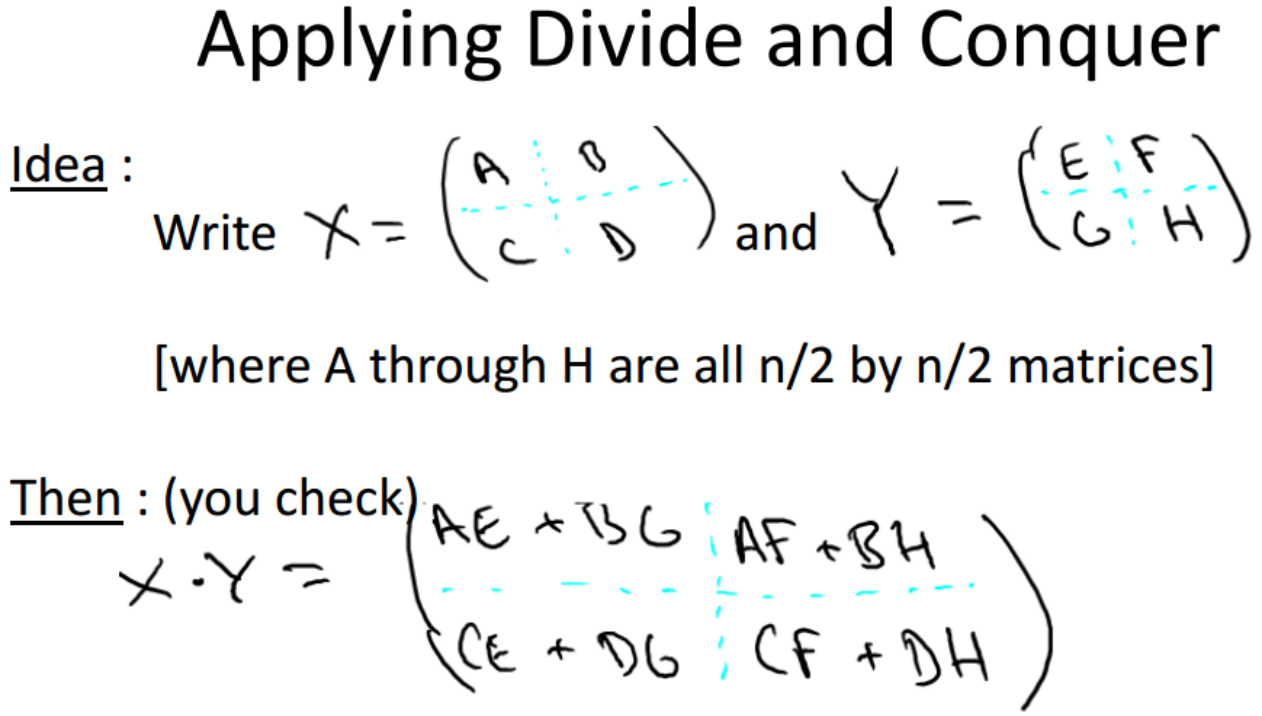
1.如果x在y之前被复制到D，那么$x < y$，说明(x, y)不是逆序对。

2.如果y在x之前被复制到D，那么$y < x$，说明(x, y)是逆序对。

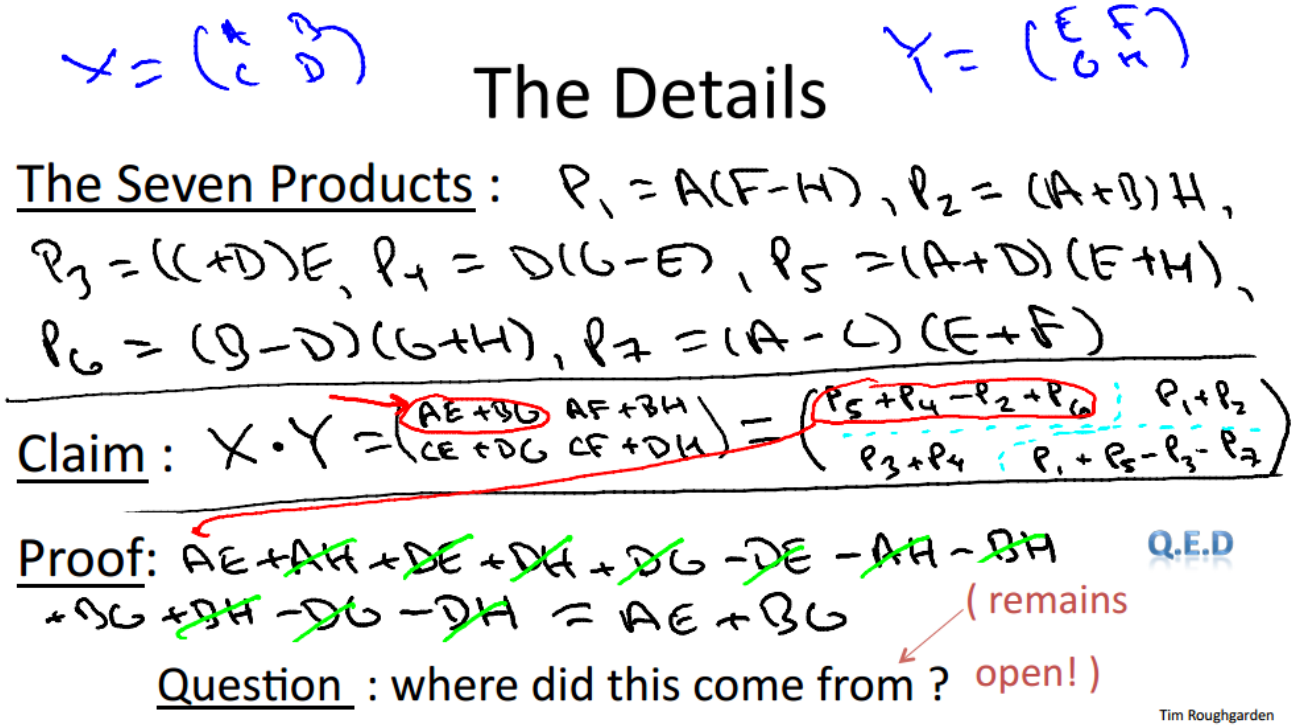
因为merge的时间为$O(n)$，所以这个算法的时间为$O(n \log n)$

Matrix Multiplication

如果将每个$n \times n$矩阵拆为4个$\frac{n}{2} \times \frac{n}{2}$，利用Divide and Conquer，有如下算法



这个算法将原问题拆成8个子问题，时间复杂度为$\theta(n^3)$，而直接计算的方法也为$\theta(n^3)$，说明这个算法没有提升速度，这时老师介绍了一个神奇的算法：



这个算法的神奇之处为将原问题拆成7个子问题，利用主定理，可以计算出时间复杂度小于$\theta(n^3)$。

Part 2:主定理

Master Method

我们之前讨论的很多算法的时间复杂度都可以写为如下形式

$$T(n) \leq aT(\frac{n}{b}) + O(n^d)$$

例如Merge Sort

$$T(n) \leq 2T(\frac{n}{2}) + O(n)$$

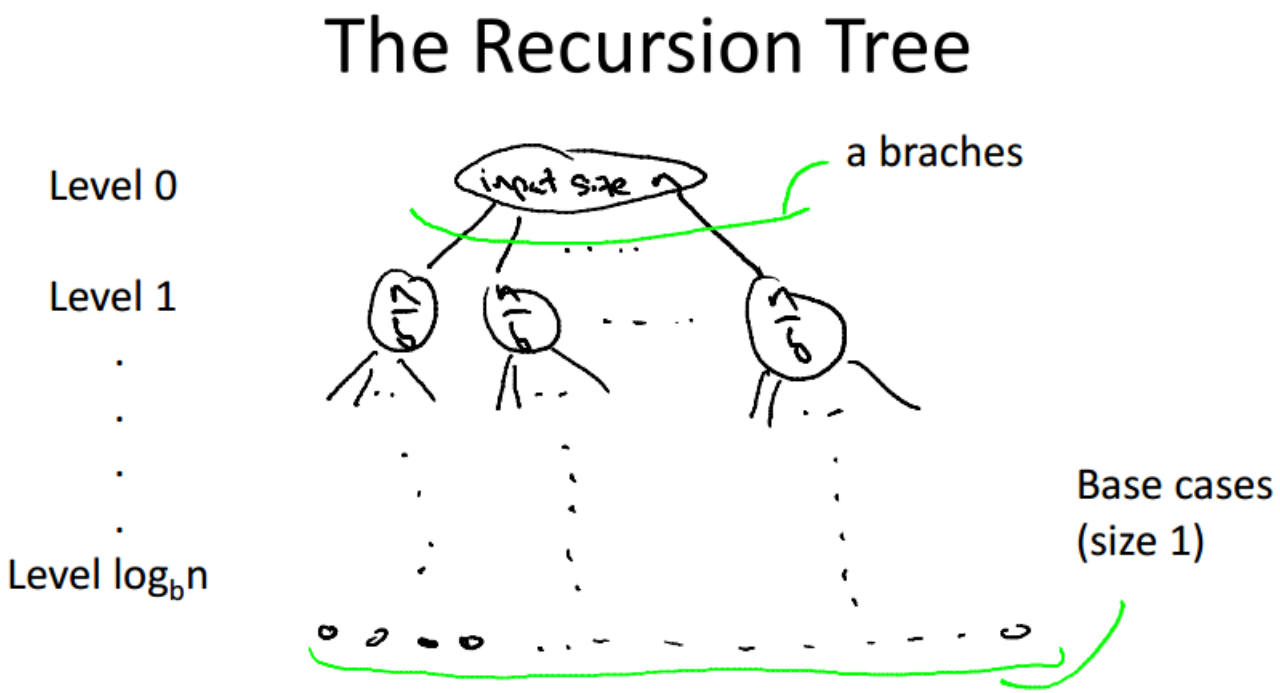
Karatsuba Multiplication

$$T(n) \leq 3T(\frac{n}{2}) + O(n)$$

Master Method描述的就是$T(n) \leq aT(\frac{n}{b}) + O(n^d)$可以推出怎样的时间复杂度，我们对原条件变形一下

$$T(1) \leq c$$
$$T(n) \leq aT(\frac{n}{b}) + cn^d$$

那么同Merge Sort的算法时间推导过程，可以画出如下树状图



那么第j层的时间复杂度

$$\leq a^j c(\frac{n}{b^j})^d = cn^d(\frac{a}{b^d})^j$$

一共的时间复杂度

$$\leq cn^d \sum_{j=0}^{\log_b n}(\frac{a}{b^d})^j$$

现在讨论这个式子，可以看出，这个式子和a, b^d的大小关系有关，分三种情况讨论

(1) $a = b^d$

那么时间复杂度

$$\leq cn^d \sum_{j=0}^{\log_b n} (\frac{a}{b^d})^j = cn^d \sum_{j=0}^{\log_b n} 1 = cn^d \log_b n$$

从而时间复杂度为

$$O(n^d \log n)$$

讨论其余两种情况之前，先考虑等比级数

$$1 + r + \ldots + r^k = \frac{r^{k+1} - 1}{r - 1} (r \neq 1)$$

如果$r < 1$，那么

$$1 + r + \ldots + r^k = \frac{r^{k+1} - 1}{r - 1} = \frac{1 - r^{k+1}}{1 - r} \leq \frac{1}{1 - r}$$

如果$r > 1$，那么

$$1 + r + \ldots + r^k = \frac{r^{k+1} - 1}{r - 1} \leq r^k (\frac{r}{r + 1})$$

利用这两个结论

(2) $a > b^d$

此时$\frac{a}{b^d} > 1$，由等比级数的性质，我们知道

$$cn^d \sum_{j=0}^{\log_b n} (\frac{a}{b^d})^j \leq cn^d D(\frac{a}{b^d})^{\log_b n} = cDa^{\log_b n} = cDn^{\log_b a}$$

所以时间复杂度为

$$O(n^{\log_b a})$$

(3) $a < b^d$

此时$\frac{a}{b^d} < 1$，由等比级数的性质，我们知道

$$cn^d \sum_{j=0}^{\log_b n} (\frac{a}{b^d})^j \leq cn^d D = cDn^d$$

所以时间复杂度为

$$O(n^d)$$

综合以上三种情形，可以得到Master Method

$$\text{如果 } T(n) \leq aT(\frac{n}{b}) + O(n^d)$$

$$\text{那么 } T(n) = \begin{cases} O(n^d), & a < b^d \\ O(n^d \log n), & a = b^d \\ O(n^{\log_b a}), & a > b^d \end{cases}$$

Part 3:习题

选择题

选择题1

$T(n) = 7T(n/3) + n^2$，那么$T(n)$的近似时间为

- $\theta(n^2)$
- $\theta(n^2 \log n)$
- $\theta(n^{2.81})$
- $\theta(n \log n)$

应用主定理，$a = 7, b = 3, d = 2$，$a = 7 < 9 = b^d$，所以时间复杂度为

$$\theta(n^d) = \theta(n^2)$$

选择题2

$T(n) = 9T(n/3) + n^2$，那么$T(n)$的近似时间为

- $\theta(n^2)$
- $\theta(n \log n)$
- $\theta(n^{3.17})$
- $\theta(n^2 \log n)$

应用主定理，$a = 9, b = 3, d = 2$，$a = 9 = b^d$，所以时间复杂度为

$$\theta(n^d \log n) = \theta(n^2 \log n)$$

选择题3

$T(n) = 5T(n/3) + 4n$，那么$T(n)$的近似时间为

- $\theta(n^{\log_3 5})$
- $\theta(n^{2.59})$
- $\theta(n^{\frac{\log_3}{\log_5}})$
- $\theta(n^2)$
- $\theta(n \log n)$

应用主定理，$a = 5, b = 3, d = 1$，$a = 5 > 3 = b^d$，所以时间复杂度为

$$\theta(n^{\log_b a}) = \theta(n^{\log_3 5})$$

选择题4

考虑如下计算a^b的伪代码

```
FastPower(a,b) :
  if b = 1
    return a
  else
    c := a*a
    ans := FastPower(c,[b/2])
  if b is odd
    return a*ans
  else return ans
end
```

这里$[x]$表示向下取整函数，即小于等于x的最大整数。

现在假设您使用支持乘法和除法的计算器（即，您可以在恒定时间内进行乘法和除法），上述算法的整体渐近运行时间（作为b的函数）是什么？

- $\theta(b\log b)$
- $\theta(b)$
- $\theta(\sqrt{b})$
- $\theta(\log b)$

设运行时间为$T(b)$，根据算法，有以下递推式

$$T(b)=T(\frac{b}{2})+\theta(1)$$

利用主定理，可得

$$T(b)=\log b$$

选择题5

选择下述问题的最小上界：$T(1)=1, T(n)\leq T([\sqrt{n}])+1$，这里$[x]$表示向下取整函数，即小于等于$x$的最大整数。注意主定理无法使用。

- $O(\log\log n)$
- $O(\sqrt{n})$
- $O(\log n)$
- $O(1)$

假设$2^{s-1}\leq n<2^s$，那么

$$T(n)\geq T(2^{s-1})=g(s-1)$$
$$T([\sqrt{n}])+1\leq T([2^{\frac{s}{2}}])+1\leq T(2^{\frac{s}{2}})+1=g(\frac{s}{2})+1$$
$$g(s-1)\leq g(\frac{s}{2})+1$$

那么问题可以改写为

$$g(s)\leq g(\frac{s}{2})+\theta(1)$$

对这个递推式利用主定理

$$a = 1, b = 2, d = 0$$
$$a = b^d = 1$$
$$T(n) = g(s) = O(\log s)$$

因为$2^{s-1} \leq n < 2^s$，所以

$$s - 1 \leq \log_2 n$$
$$\log_2 n < s$$
$$\log_2 n < s \leq \log_2 n + 1$$
$$s = \theta(\log n)$$

带入上式可得

$$T(n) = O(\log \log n)$$

思考题

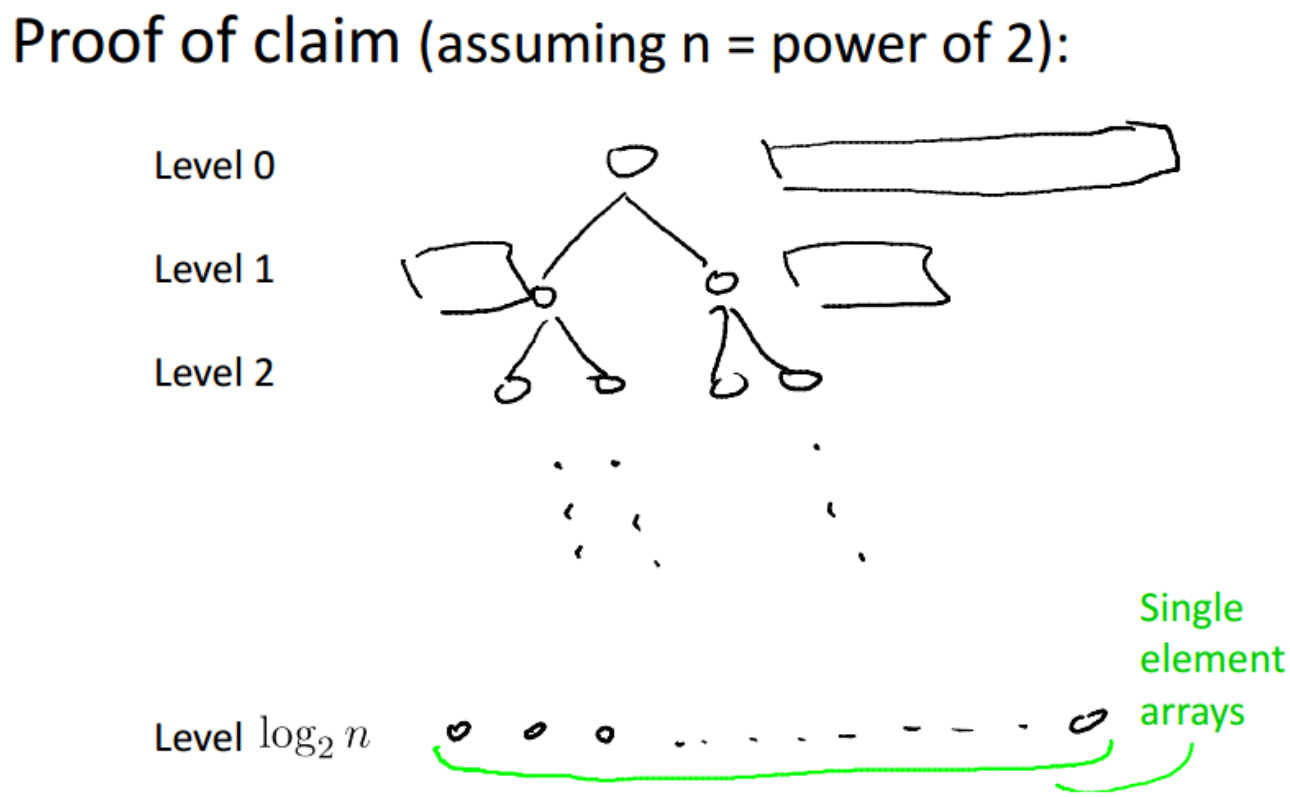
思考题1

现在有n个元素且大小不相同的无序数组，n为2的幂，给出一个最多用$n + \log_2 n - 2$次比较的算法来找出第二大的元素。

设$n = 2^s$

利用淘汰赛赛制的方法，先找出最大的元素，再在和最大元素“交手”过的元素中找到最大的元素，这个元素就是第二大的元素，下面具体说明一下。

将数组中元素两两一组进行比较，大的元素获胜，进行下一轮比较，持续这个操作，最后会剩下一个元素，这个元素就是最大值，这个过程可以用以下树状图来说明。下面借助树状图来说明一下这个结论为什么是正确的：



首先由比赛的规则我们知道，每个节点都大于等于其子节点，由这个性质，最后剩下的这个元素必然大于等于全部的元素，从而为最大值。

现在说明了这个结论的正确性，我们来计算一共需要几步。对于2^k个元素，需要$2^k/2 = 2^{k-1}$次比较来进行一轮淘汰赛，所以对于$n = 2^k$，完成全部淘汰赛一共需要的比较为

$$\sum_{i=0}^{k-1} 2^i = 2^k - 1 = n - 1$$

现在考虑和最大元素交手过的k个元素，下面说明，这个k个元素中的最大元素即为全部元素中第二大的元素。

还是借助上述树状图，和最大元素交手的元素必然大于等于其左子树以及右子树的全部元素，而我们的目标是找到除了最大元素以外的最大元素，所以只要考虑这些和最大元素交过手的元素即可。

所以现在只要找到k个元素的最大值即可，这个只要进行$k-1$次比较即可，所以总共的次数为

$$n - 1 + k - 1 = n + \log_2 n - 2$$

思考题2

给定一个由n个不同元素组成的单峰数组，这意味着它的元素按递增顺序排列直到其最大元素，之后其元素按递减顺序排列。给出一个算法在$O(\log n)$时间内找到最大元素。

这个问题关键在于找到波峰在哪，可以结合二次函数的图像来考虑，假设数组的元素为$a_1, \ldots, a_n$，将数组分为三个部分：

$$\text{第一部分：}\ a_1, \ldots, a_{[\frac{n}{3}]},\ \text{第二部分：}\ a_{[\frac{n}{3}]}, \ldots, a_{[\frac{2n}{3}]},\ \text{第三部分：}\ a_{[\frac{2n}{3}]}, \ldots, a_n$$

波峰必然在这三个部分中，利用很熟悉的二次函数的图像，根据比较$a_{[\frac{n}{3}]}, a_{[\frac{2n}{3}]}, a_n$的大小关系（一共6种，这里略过），可以确定波峰在哪段数组中，所以

$$T(n) \leq T(\frac{n}{3}) + O(1)$$

利用主定理，这个算法的时间复杂度为

$$O(\log n)$$

思考题3

现在有一个从最小到最大排好序的n个不同整数的数组A，它们可以是正数，负数或零。现在要判断是否存在索引i，使得$A[i] = i$。 设计最快的算法来解决这个问题。

我们考虑$A[[\frac{n}{2}]]$这个元素：

- 如果$A[[\frac{n}{2}]] = [\frac{n}{2}]$，那么直接返回结果True；
- 如果$A[[\frac{n}{2}]] > [\frac{n}{2}]$，那么由$A$为整数数组可知，任取$k \in N$，$A[[\frac{n}{2}] + k] \geq A[[\frac{n}{2}]] + k > [\frac{n}{2}] + k$，这说明$A[[\frac{n}{2}] + 1], \ldots, A[n]$这些元素都不可能满足$A[i] = i$，所以只要递归地考虑$A[1], \ldots, A[[\frac{n}{2}] - 1]$这些元素即可。
- 如果$A[[\frac{n}{2}]] < [\frac{n}{2}]$，那么由$A$为整数数组可知，任取$k \in N$，$A[[\frac{n}{2}] - k] \leq A[[\frac{n}{2}]] - k < [\frac{n}{2}] - k$，这说明$A[1], \ldots, A[[\frac{n}{2}] - 1]$这些元素都不可能满足$A[i] = i$，所以只要递归地考虑$A[1], \ldots, A[[\frac{n}{2}] - 1]$这些元素即可。

由上述叙述知，这个算法的时间复杂度满足以下递推式

$$T(n) \leq T(\frac{n}{2}) + O(1)$$

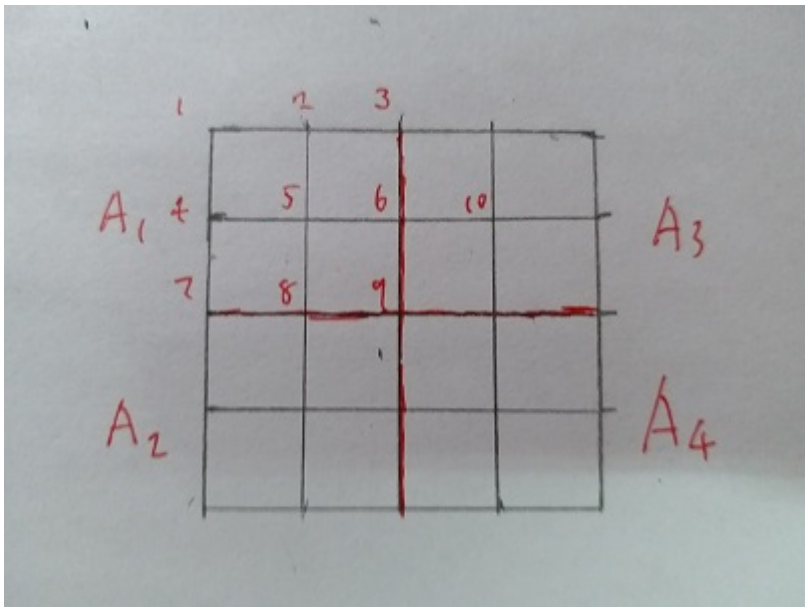
由主定理可知，时间复杂度为

$$T(n) = \log n$$

思考题4

给你一个$n \times n$个不同数字的网格。 如果数字小于其所有邻居，则该数字是局部最小值。 （一个数字的邻居是上方，下方，左侧或右侧的一个。大多数数字有四个邻居;侧面的数字有三个;四个角有两个。）使用分而治之的算法计算局部最小值，仅对数字对进行$O(n)$次比较。（注意：因为有n^2个元素，所以不能查看全部元素。 提示：考虑哪种类型的递归会给你所需的上限。）

结合下图进行说明



以左下角为原点，右手方向为x轴，竖直方向由y轴，那么直线$x = [\frac{n}{2}]$以及直线$y = [\frac{n}{2}]$将格子分为四个部分，按上图分别记为A_1, A_2, A_3, A_4。

我们考虑位于直线$x = 0, x = [\frac{n}{2}], x = n, y = 0, y = [\frac{n}{2}], y = n$这6条直线上的点，设这些点中的最小值为$a$，考虑以下几种情形：

情形1：$a \in A_i, a \notin A_j$，这种情形表示a不属于边界处，此时对A_i递归地调用该算法。（对应上图$1, 2, 4$）

情形2：$a \in A_i \cap A_j \cap A_k$，这种情形表示$a$属于三个区域交界处，由定义，此时$a$即为局部最小值。（对应上图$9$）

情形2：$a \in A_i \cap A_j, a \notin A_k$，这种情形表示$a$属于两个区域的交界处，但不属于三个区域的交界处，此时将a和其上下或者左右两个未比较过的点c, d进行比较，如果$a < b, a < c$，那么a为局部最小值，否则对最小值所在区域递归地调用该算法（对应上图$3, 6, 7, 8$）

计算这6条直线上的点需要$O(n)$的时间，而每次递归，问题的规模缩小一半，所以有以下递推式

$$T(n) = T(\frac{n}{2}) + O(n)$$

由主定理，$a=1, b=2, d=1$，时间复杂度为

$$O(n)$$