第三章

(a). 输入：实数 a，自然数 n.

输出：a^n

```
FunA(a, n)
if (n==1) return a
else if (n mod 2 == 0)
    return FunA(a, n/2) * FunA(a, n/2)
else
    return FunA(a, (n-1)/2) * FunA(a, (n-1)/2) * a
```

算法时间复杂度：$T(n) = T(\frac{n}{2}) + \Theta(1)$

由 master 定理：$T(n) = \Theta(\log n)$

(b). 输入：实数矩阵 A，自然数 n.

输出：A^n

```
FunB(A, n)
if (n==1) return A
else if (n mod 2 == 0)
    return FunB(A, n/2) * FunB(A, n/2)
else
    return FunB(A, (n-1)/2) * FunB(A, (n-1)/2) * A
```

时间复杂度：$T(n) = T(\frac{n}{2}) + \Theta(1)$

由 master 定理：$T(n) = \Theta(\log n)$

3.2 (a) (1) 证明 $F(n+2) = 1 + \sum_{i=0}^{n} F(i)$.

证：

$F(n+2) - F(n+1) = F(n)$

$F(n+1) - F(n) = F(n-1)$

$F(n) - F(n-1) = F(n-2)$

$\vdots$

$F(2) - F(1) = F(0)$

累加可得：

$F(n+2) - F(1) = F(n) + F(n-1) \cdots + F(0)$

故 $F(n+2) = 1 + \sum_{i=0}^{n} F(i)$.

(2) 证明 $F(n+2) \geq \left(\frac{1+\sqrt{5}}{2}\right)^n$

由例 2-19 结论可得：$F(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$

数学归纳法：

当 $n=0$ 时 $F(2) = F(1) + F(0) = 1$

$= \left(\frac{1+\sqrt{5}}{2}\right)^0$. 成立

假设 $n=k$ 时成立 则 $F(k+1) = \left(\frac{1+\sqrt{5}}{2}\right)^{k-1}$

当 $n=k$ 时 $F(k+2) = F(k+1) + F(k)$

$= \left(\frac{1+\sqrt{5}}{2}\right)^{k-1} + \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k\right]$

$= \left(\frac{1+\sqrt{5}}{2}\right)^{k-1} + \left(\frac{1+\sqrt{5}}{2}\right)^{k-2} = \left(\frac{1+\sqrt{5}}{2}\right)^{k-2}\left(\frac{1+\sqrt{5}}{2} + 1\right)$

$= \left(\frac{1+\sqrt{5}}{2}\right)^{k-2} \cdot \left(\frac{3+\sqrt{5}}{2}\right) = \left(\frac{1+\sqrt{5}}{2}\right)^{k-2} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^2$

$= \left(\frac{1+\sqrt{5}}{2}\right)^k$ 命题成立.

$\left(\frac{1+\sqrt{5}}{2}\right)^2$

$= \frac{1+2\sqrt{5}+5}{4}$

$= \frac{6+2\sqrt{5}}{4} = \frac{3+\sqrt{5}}{2}$

). F(n).

```
if (n=0 || n=1)
    return 1
else
    return F(n-1) + F(n-2).
```

$F(n) = F(n-1) + F(n-2) + 1$

特征方程为 $x^2 - x - 1 = 0$.

通解为 $F(n) = c_1(a_1)^n + c_2(a_2)^n$

故 $F(n) = O(2^n)$.

(c)2. F(n)

```
if (n=0 || n=1)
    return 1
else
    return 1/√5 [ ((1+√5)/2)^n - ((1-√5)/2)^n ]
```

由3.1(a)可知，利用分治法.

$F(n) = O(\log n)$.

问题：浮点数运算时精准度不高.

(d). $f(2) = F(1) + F(0) = 2$.

$$\begin{pmatrix} F(n+2) & F(n+1) \\ F(n+1) & F(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2\begin{pmatrix} F(n) & F(n-1) \\ F(n-1) & F(n-2) \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F(2) & F(1) \\ F(1) & F(0) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \times \begin{pmatrix} 2 & -1 \\ 1 & 1 \end{pmatrix} = \cancel{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n}$$

可利用3.1(b)的方法计算 $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ 值. 设 $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

```
F(A, n)
    if (n = 0 || n = 1)
        return 1
    else if (n = 2)
        return 2.
    else
        B = A^(n-2)
        return 2B11 + B12.
```

由3.1(b)时间复杂度为 ~~$\Theta(n\log n)$~~ $O(\log n)$.

(e). b：由递归生成，计算量大，复杂度高.
精确，简单，易于实现.

c. 分治思想，降低了复杂度.
精度降低，n很大时易出误差.

d. 矩阵相乘便于计算
降低了复杂度，同时保证了计算精度.

3.3. 输入：平面n个点集合S，数据经预处理，A[0:n-1]存储S(按x坐标有序)
输出：S中友谊点对个数 count

~~B[0:n-1]存储S(按y坐标有序)~~

A_L 表示 A在B中

```
F_Point(A)
~~if n=2 return 1~~
k = n/2
A_L[0:k-1] = A[0:k-1]
A_R[0:n-k-1] = A[k:n-1]          Divide
-----------------------------------
countA = F_Point(A_L)
countB = F_Point(A_R)            conquer
-----------------------------------
countM = Friend(A_L, A_R)        Merge.
```

```
Friend(A_L[0:n], A_R[0:m])   // 合并过程
~~对y排序，得到B[0:m-1]~~   ~~B_R[0:m]~~   count =
for i=0 to n
    for j=0 to m.
        ~~if A_L[i] 对所有 x<A_R[j] 且 x>AL~~
        ~~对y坐标排序得到 B~~
        ~~B[A[j].z]~~
        if |A_L[i].z - A_R[j].z| = ≤
```

Divide：依据 x 坐标取中位线将点集一分为二 S'_L 和 S'_R

Coaquer：递归求出 S'_L 和 S'_R 中的友谊点对，当划分出的子集只含两个或十个点时终止

Merge：寻找友谊点对 (P,q)，其中 P 是左半部分点，q 是右半部分点

对于任一左半部点，分别考察右半部分点是否存在友谊点对

例：考察 (P',q')，在 S 中找到 P'，q' 以及 x 坐标介于 P'.x 和 q'.x 之间的点，

按照 y 坐标排序，如果排序后 P'，q' 相邻，则 (P',q') 是友谊点对。

$T(n)=2T(\frac{n}{2})+\theta(n^2)$　　所以 $T(n)=\theta(n^2)$.

---

4. 首先将点集 S 按照 x 坐标排序得到点集 S'

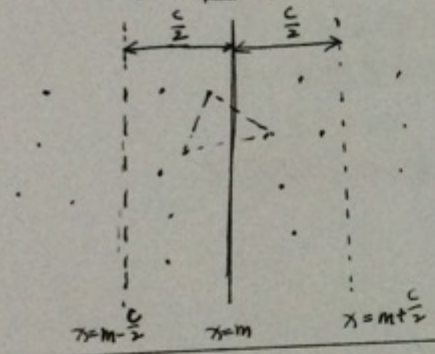
① Divide：依据 x 坐标定找到一条中位线将 S' 一分为二 S'_L 和 S'_R

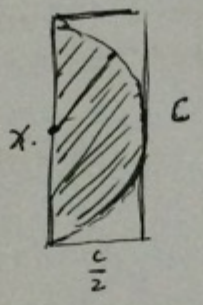
② conquer：递归在 S'_L 和 S'_R 中找到周长最小三角形 $(P_1P_2P_3)\in S'_L$，$(q_1q_2q_3)\in S'_R$

周长分别为 C_L，C_R。设 $C=\min\{C_L,C_R\}$

③ Merge：三角形两边之和大于第三边，故在~~宽度为~~ C 的中位线两侧寻找最小周长三角形

当固定一点时只需考察该点右侧 $C\cdot\frac{C}{2}$ 的矩形即可（常数时间 $*n$）





$$T(n)\begin{cases}2T(\frac{n}{2})+\theta(n) & n>3\\ \theta(1) & n\le 3\end{cases}$$

解得 $T(n)=\theta(n\log n)$

3.5 算法过程：

```
1  If  n=3  Then 逐对判断顺序输出S顶点. return.
2  For ∀P1.P2.P3.P4∈S  do.   // 四层循环.
3      If 任意一点位于其他点构成三角形内或边上 Then
4          删除该点.
5  A ← S中横坐标最小点
6  B ← S中横坐标最大点
7  SU ← {P | P∈S且 g(A.B.P)>0}
8  SL ← {P | P∈S且 g(A.B.P)≤0}
9  排序 SL.  按横坐标增序   SU按横坐标逆序.
10 输出: SL. SU
```

说明：

初始：给定一个点集，凸包必在其中

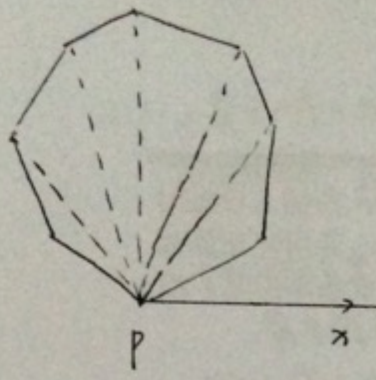
循环：根据性质 3-2，如果 $P_1P_2P_3P_4\in S$ 是4个不同的点，且 P_1 位于 $\triangle P_2P_3P_4$ 内部或边界则 P_1 不是凸包顶点

将这个点删除后，得到 S'，则原点集 S 的凸包仍在 S' 中。

终止：去除所有非顶点后

剩余的点即是点集 S 的最大凸包

在多边形上取1坐标点P



① 计算出多边形其他点相对P极角并排序 ⟶ $\Theta(n)+\Theta(n\log n)$.

② 算出 $q_1 q_2 \cdots q_n$ 相对P极角 ⟶ $\Theta(n)$

③、采用二分查找 找出q所在的极角范围 ⟶ $n\times\Theta(\log n)$

若 q_i 不在凸多边形点构成的极角范围内

则 q_i 肯定在多边形外.

若 q_i 在某一极角范围内，再计算 $g(A.B.q)$ 确定是否在多边形中.

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

3.8 可以从根结点开始划分，每次对一棵子树递归求解 算出距离 求和

① 计算当前根结点所有下属子结点到该根结点距离

并存在数组D中.

② 相对于当前根结点，距离小于 τ 的点v有两种情况.

1. v到该根结点距离小于 τ，由D数组可直接得到

2. 跨越该根结点 两子树顶点距离根结点距离和小于 τ 则

③ 递归在各子树进行①②.

开始时 输入 t = root(T)

```
Find-Dis (t, v).
count=0 .  D={ }
for t的所有下属结点a do
    D[a] = a到t距离.
for v in D.
    if D[v] < τ      count+=1
    for t in D:
        if D[v]+D[t] < τ.  count+=1
    Find-Dis(v, τ).
return count.
```

3.9. ① 分别计算两数组的中位数. $a=X[\frac{n}{2}]$ $b=Y[\frac{n}{2}]$ 设2n个数中位数为m

② 若 $a=b$，则2n个数的中位数就是a. $m=a=b$.

2. 若 $a>b$，则至少有n个数比a小. $b<m<a$ 保留 $Y[\frac{n}{2}, n]$. $X[0, \frac{n}{2}]$

3. 若 $a<b$ 则至少有n个数比b小. $a<m<b$. 保留 $Y[0, \frac{n}{2}]$, $X[\frac{n}{2}, n-1]$

③ 递归以上步骤. 当两数组长度为1时结束.

利用剪枝二分搜索. $T(n)=T(\frac{n}{2})+O(1)$. $T(n)=O(\log n)$.

3.10. ①. Divide：将 $A[1:n]$ 由中位数划分为两个数组 A_1, A_2 ~~$A_1[1:\frac{n}{2}]$ $A_2[\frac{n}{2}+1:n]$~~ 当n=2，统计反序对并排序

②. Conquer：递归计数 A_1 和 A_2 中的反序对个数 ~~先将 $A_1 A_2$ 归并排序，得到 $A'[1. m]$~~

③. Merge： ~~$A_1 A_2$ 均有序 将 $A_1 A_2$ 归并排序 $O(n)$. 得到 A' 然后按以下合并~~

仿照归并排序. 按以下方式合并 ~~$A_1[1, m]$. $A_2[1, n]$~~

```
count=0 .  j=m/2 +1
for( i=1; i≤ m/2; i++ )
    while ( j ≤ m 且 A'[i] > A'[j] )
        j++
        count += 1
```

时间复杂性： $T(n)=2T(\frac{n}{2})+O(n)$

~~for double~~：然后对S中的元素y计算z=x-y.

② 再从S中使用二分查找z. 复杂度 O(logn) 找到说明存在.

最多循环n次. 复杂度O(nlogn)

所以算法时间复杂度 T(n)=O(nlogn).

---

2. (1). 若K已知. 数组A为 A[1:n]. 则令 x= kmod n. 若 x=0. 则最大元素为A[n].

否则 A中最大元素为 A[x]. 复杂度为 O(1).

(2) 若K未知. 数组为 A[1:n]. 则令 x= A[m] 其中 m=⌊n/2⌋

则大将A分成左右大致等长两个部分 比较 x. A[1]. A[n] 大小关系.

① 若 A[1] ≤ X ≤ A[n]. 例: 1 2 3 4 5 X=3. 此时A中最大元素为A[n]

② 若 X≤A[1] 且 X≤A[n] 例: 5 1 2 3 4 X=2 此时A最大元素在A[1:m]中.

可对 A[1:m-1] 递归求解

③ 若 X≥A[1] 且 X≥A[n] 例: 2 3 4 5 1 X=4. 此时A最大元素在A[m:n]中.

可对 A[m:n] 递归求解.

T(n) = T(n/2) + O(1). 由master定理 T(n)= O(logn)

此算法与二分查找方法类似.

3.13. 首先将A[1:n]和B[1:n]排序. 得到A'[1:n]和 B'[1:n].

设 L=∅. R= A'[n]·B'[n].

①令 key = (L+R)/2. 统计集合{A[i]·B'[j] | L≤i≤R. L≤j≤R}中小于key的个数num.

②. 若 num=k. 则 key就是所求结果. return.

若 num>k. 则 R= key-1 从①开始递归调用.

若 num<k 则 L= key+1. 从①开始递归调用.

3.14. ① 若 M[0][0] = X 则 输出 M[0][0] 算 return

若 M[n-1][n-1] = X 输出 M[n-1][n-1] return.

若 M[0][0]>X 或 M[n-1][n-1]<X. 则 X不在M中. return

② 将M划分为四个子矩阵:

A: M[0: ⌊m/2⌋][0:⌊n/2⌋] C: M[⌊m/2⌋+1:m][0:⌊n/2⌋]

B: M[0:⌊m/2⌋][⌊n/2⌋+1:n] D. M[⌊m/2⌋+1:m][⌊n/2⌋+1:n]

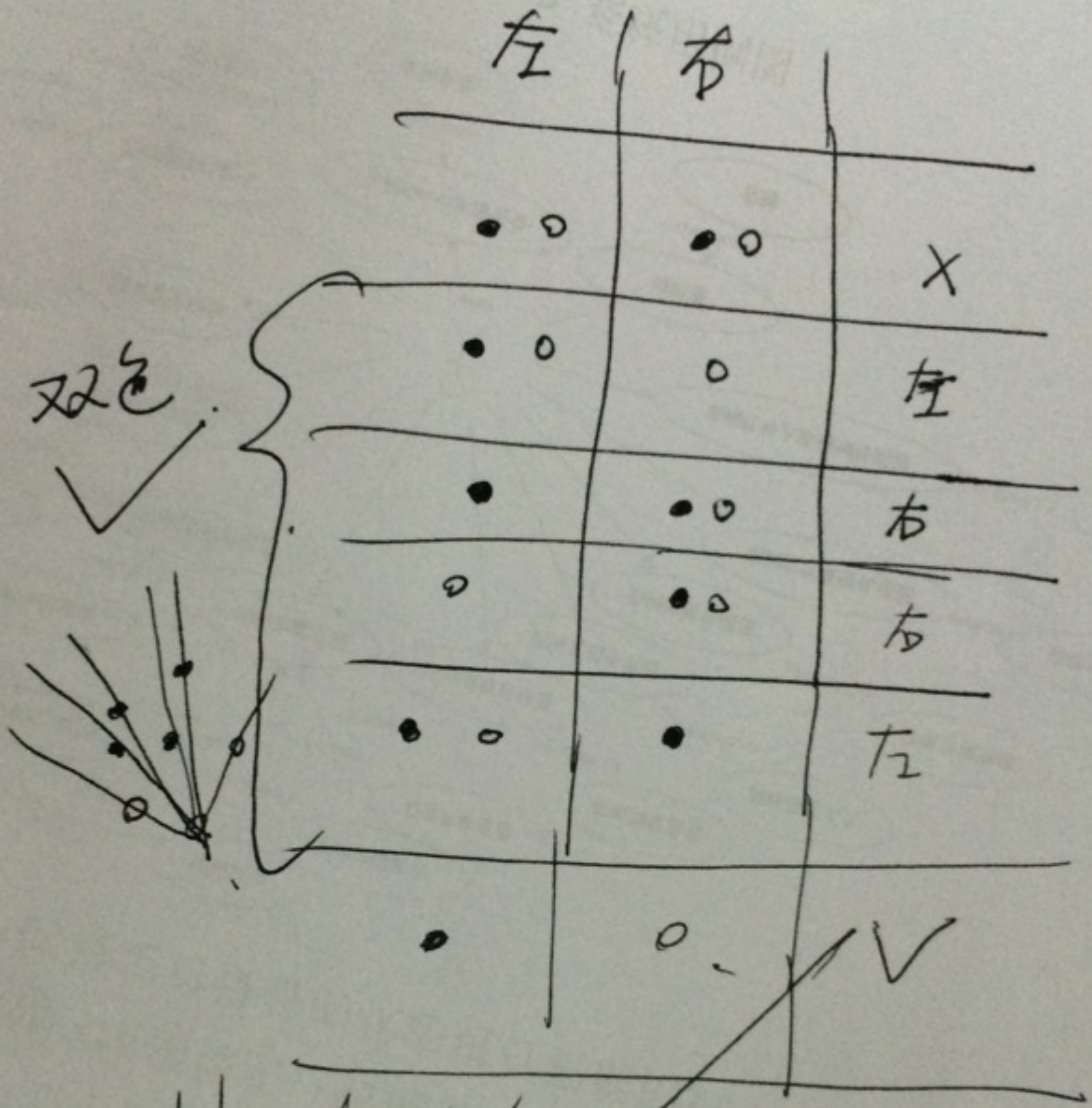
③ 递归进行①.②对于选择的矩阵:

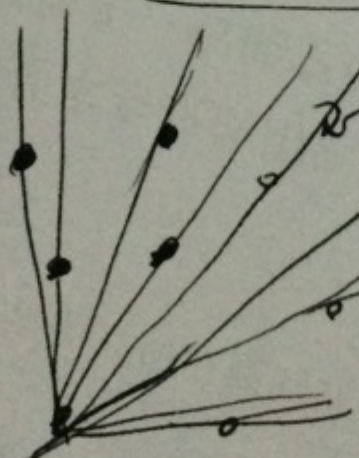
若 M[⌊m/2⌋][⌊n/2⌋] < X. 则 BCD.

若 M[⌊m/2⌋][⌊n/2⌋] ≥ X 则 ABC

每次最坏情况递归3个矩阵. 则 T(mn) = 3T(mn/4) + θ(1)

3.15





左指针 每次去掉一半

$T(n) = T(\frac{n}{2}) + \Theta(1)$

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

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

$2T(\frac{n}{2}) + \Theta(1)$ A