# 算法设计和分析作业

# Chatper 1 Abstract Compute Model

## 1.2 找中位数

1. 设计算法找出中位数

```python
int FindMedian(int a,int b,int c){
    if a > b:
        swap(a,b) #让a,b,c形成升序, 现在 a一定小于b
    else if a>c:
        return a #此时已经知道c<a<b
    else:
        if c>b:
            return b
        else return c
}
```

2. 
```
最坏情况下:
    需三次比较。
平均意义:
    共 abc,  acb, bac, bca, cab, cba 六种升序排列可能。
    其中 2 次比较 return:  cab, cba
        3 次比较  return: abc, acb, bac, bca
```

$$Average = 2 \times \frac{2}{6} + 3 \times \frac{4}{6} = \frac{8}{3}$$

3. 最坏情况下至少需要 3 次比较。
   Proof:
   因为找到中位数，至少需要确定 3 个数的一种排列方式。而 3 个数的排列方式为 $3! = 6$ 个，那么至少需要比较 $3$ 次才能形成 $2^3 = 8$ 的答案空间，由鸽笼原理之想要的排列至少要 3 次比较才能确定下来从而确定中位数.

# 1.3 集合最小问题

## 1. 失败的例子

可以考虑：

$$U=\{1,2,3,4,5,6\},S_1=\{1,2,3,4,5\},S_2=\{1,2,3,4\},S_3=\{1,2,3\},\ldots,S_m=\{6\}$$

此时该算法会贪心地依次选择 $S_1,S_2,S_3\ldots$，但其实最优方案是 **选择** $S_1$ 和 $S_m$

## 2. 设计集合覆盖

PYTHON

```python
St={}
for all Si in S, do:
    if(St UNION Si != U):
        St= Si UNION St #add a new set
    else:#cover found
        return St
```

## 3.

并不是。它只是简单的从头遍历依次选择，（1）中失败的例子仍然可以使我的算法无效。

# 1.7 多项式计算

用数学归纳法。
Basis: 显然对于单项多项式 $P(x)$，算法正确
H: 假设 $k=n-1$，算法能正确计算 $P(x)$ 的值，即能成功计算
$P'(x)=a_nx^{n-1}+a_{n-1}x^{n-2}+\cdots+a_1$
I.S: 需要证明 $n=k$，也可以正确计算

$$\begin{aligned}P(x)&=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1+a_0\\&=P'(x)\cdot x+a_0\end{aligned}$$

根据算法过程，其完成的迭代运算正确，故该算法正确。

# 1.8 整数相乘

首先证明 **recursive equatation**

$$
\begin{aligned}
z = kc + z \bmod c \\
k = \left\lfloor \frac{z}{c} \right\rfloor \\
c \times \left\lfloor \frac{z}{c} \right\rfloor = z - z \bmod c
\end{aligned}
$$

移项即证算法中的递归式

证明算法终止：
该算法的传参规模的 $z$ 项是强递减的，且最终会递归到 $0$，故算法会终止。并给出正确答案。

## 1.9

$$
Pr(r = i) = \begin{cases} \frac{1}{n}\left(1 \le i \le \frac{n}{4}\right) \\ \frac{2}{n}\left(\frac{n}{4} < i \le \frac{n}{2}\right) \\ \frac{1}{2n}\left(\frac{n}{2} < i \le n\right) \end{cases}
$$

假设 n 是 4 的倍数。
算法的输入 r 为 1 到 n 之间的自然数。
求数学期望：denotes the counts of operatoins as $X$

$$
E(X) = \frac{1}{n} \cdot 10 \cdot \left(\frac{n}{4}\right) + \frac{2}{n} \cdot 20 \cdot \left(\frac{n}{2} - \frac{n}{4}\right) + \frac{1}{2n} \cdot 30 \cdot \left(\frac{3n}{4} - \frac{n}{2}\right) + \frac{1}{2n} n \cdot \left(n - \frac{3n}{4}\right)
$$

可以得到

$$
E(X) = 2.5 + 10 + \frac{15}{4} + \frac{1}{8}n = 16.25 + \frac{1}{8}n
$$

## 1.10

判断：用于判断数组中是否有两个位置不同的元素相等

1. 最坏时间复杂度：
    1. 假设数组没有相同元素，则将完整遍历数组，时间复杂度为 $O(n^2)$
2. 平均复杂度：
    1. 数组中有且仅有两个元素相等，该遍历构成这样一个 $n-1 \times n-1$ 遍历矩阵图：

$$
\begin{aligned}
0, 1, 2, 3, \ldots, n-1 \\
1, 2, 3, \ldots n-1 \\
2, 3, \ldots, n-1 \\
\ldots
\end{aligned}
$$

若输入中 $A[i] == A[j]$，则算法在第 $i$ 行第 $j$ 列终止。矩阵共有 $\sum_{i=0}^{n-2}(i+1)$ 个可能终止的位置，知道在这 $\frac{n(n-1)}{2}$ 个位置终止概率相等
比较次数：

$$\frac{2}{n(n-1)} \times \sum_{k=1}^{n(n-1)/2} k = \frac{1}{2} + \frac{n(n-1)}{4}$$

复杂度 $O(n^2)$

3. 考虑在 (i, j) 算法没有终止：那么说明前 i 个数字各不相同，且第 i 个数字和 i+1 到 j 的数字没有重复。这个事件的概率为（不太会写）

# Chapter 2 Math and Algorithm

## 2.2

请证明：对于任意整数 $n \geq 1$，$\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$
Proof：
假设 $2^k \leq n \leq 2^{k+1} - 1$, 则 $k < \log(n+1) \leq k+1$
则: $\lceil \log(n+1) \rceil = k+1$
同理，$k \leq \log(n) < k+1$
则：$\lfloor \log n \rfloor + 1 = k+1$
即证得

## 2.5

我们越过第一问直接证明第二问，则第一问也是显然的
Proof:
用数学归纳法.
当 T 的节点数 $n = 1$ 时，显然成立 $n_0 = n_2 + 1$
假设 $n = k - 1$ 时成立 $n_0 = n_2 + 1$，证明 $n = k$ 时也成立：
分两种情况讨论：
若新加的节点，作为左节点加入：
则 $n_0$ 数目不变，$n_2$ 数目不变，仍然成立
若新加的节点，作为右节点接入：
则 $n_2 + = 1$，$n_0 + = 1$，仍然成立

故递归步骤可以完成，故得证。

## 2.7

1. 传递性证明
   先证明 $O$
   假设存在函数 $f, g, h$，且 $f \in O(g)$，$g \in O(h)$
   Then

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=c_1<\infty$$

且

$$\lim_{n\to\infty}\frac{g(n)}{h(n)}=c_2<\infty$$

则

$$\lim_{n\to\infty}\frac{f(n)}{h(n)}=\lim_{n\to\infty}\frac{f(n)}{g(n)}\times\frac{g(n)}{h(n)}=\lim_{n\to\infty}\frac{f(n)}{g(n)}\times\lim_{n\to\infty}\frac{g(n)}{h(n)}=c_1c_2<\infty$$

则有 $f\in O(h)$，符合传递性（利用了极限的四则运算）

对于 $\Omega$ 的证明完全类似，略去。
对于 $o$ 的证明：只需要认识到两个极限都趋于 0 时，其有限个乘积的极限也是 0 立得
对于 $\omega$ 的证明与 $o$ 完全类似，略去
对于 $\Theta$ 的证明： 我们利用已经证明的 $\Omega$ 和 $O$ 的结论
To Prove: if $f\in\Theta(g)\wedge g\in\Theta(h)$，then $f\in\Theta(h)$
Proof:
$f\in\Theta(g)\to f\in O(g)\wedge f\in\Omega(g), g\in\Theta(h)\to g\in O(h)\wedge g\in\Theta(h)$
由前结论，知道 $f\in O(h)\wedge f\in\Omega(h)$
故 $f\in\Theta(h)$
证明完成。

2. 自反性
   证明：
   先证明 $O$
   假设有一个函数 $f$，则显然

$$\lim_{n\to\infty}\frac{f(n)}{f(n)}=1<\infty$$

   满足 $f\in O(f)$
   同理，

$$\lim_{n\to\infty}\frac{f(n)}{f(n)}=1>0$$

   满足 $f\in\Omega(f)$
   由 $f\in O(f)\wedge f\in\Omega(f)\to f\in\Theta(f)$
   满足 $f\in\Theta(f)$

3. 证明 $\Theta$ 是等价关系
   Proof: 由 2 已经证明 $\Theta$ 满足自反性，由 1 已经证明传递性，只需证明对称性即可。
   To Prove: if $f \in \Theta(g)$，then $g \in \Theta(f)$
   $f \in \Theta(g)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c, c \in (0, \infty)$$

   故

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{1}{\frac{f(n)}{g(n)}} = \frac{1}{\lim_{n\to\infty} \frac{f(n)}{g(n)}} = \frac{1}{c} \in (0, \infty)$$

   故 $g \in \Theta(f)$

4. We know

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c, c \in (0, \infty)$$

   则知道我们满足

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \ \wedge \ \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

   由 $O, \Omega$ 定义，知道 $f \in O(g) \wedge f \in \Omega(g)$

反向推类似，故这两种表述是等价的。

5. 证明 $f \in O(g) \iff g \in \Omega(f)$
   If $f \in O(g)$, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$$

   则由极限的四则运算性质，有

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{1}{\frac{f(n)}{g(n)}} = \frac{1}{c} > 0$$

   故 $g \in \Omega(f)$
   $o$ 的证明完全类似，只需要注意在算法复杂度领域我们认为 c 恒不负，即可得到

$$f \in o(g) \iff g \in \omega(f)$$

6. Prove $o(g(n)) \cap \omega(g(n)) = \emptyset$
   $\forall f \in o(g(n))$, we hold that

$$\forall c > 0, \exists n_0 > 0, 0 \le f(n) < cg(n) \text{ when } n \ge n_0$$

$\forall h \in \omega(g(n))$, we hold that

$$\forall c > 0, \exists n_0 > 0, 0 \le cg(n) < f(n) \text{ when } n \ge n_0$$

取 $N$ 为 $f$ 和 $h$ 对应的 $n_0$ 中较大的一个，当 $n > N$, 发现

$$f(n) < cg(n) \wedge cg(n) < f(n) = False$$

故二者交集为空。

## 2.8 渐进增长率排序

1. $$\log n < n < \ n \log n < \ n^2 = \ n^2 + \log n < n^3 < n - n^3 + 7n^5 < 2^n,$$

2. $$\begin{aligned} \log \log n < \log n = \ \ln n < (\log n)^2 < \ \sqrt{n} \\ < \ n < \ n^{1+\varepsilon} < \ n \log n < \ n^2 \\ = \ n^2 + \log n < n^3 < n - n^3 + 7n^5 < 2^{n-1} < \ 2^n < e^n < \ n! \end{aligned}$$

## 2.16 计算渐进增长率

引用 master 定理在这里：
Divide-and-conquer:
$T(n) = bT\left(\dfrac{n}{c}\right) + f(n)$
主定理：
Define $E = \frac{\log b}{\log c}$

- $f(n) \in O(n^{E-\varepsilon})$, then $T(n) \in \Theta(n^E)$
- $f(n) \in \Theta(n^E)$, then $T(n) \in \Theta(f(n) \log n)$
- $f(n) \in \Omega(n^{E+\varepsilon})$, and $f(n) \in O(n^{E+\delta}), \delta \ge \varepsilon$ then $T(n) \in \Theta(f(n))$

做题：(为防止与我习惯的 c 概念混淆，如果是用 MasterTheorm 解决的题，f(n) 的常数 c我用 a 替代)

1. $T(n) = 2T\left(\frac{n}{3}\right) + 1$
   $b = 2, c = 3, f(n) = 1, E = \log_3 2, 1 \in O(n^E)$

$$T(n) \in \Theta(n^{\log_3 2})$$

2.

$$
\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + c\log n \\
&= T\left(\frac{n}{4}\right) + c\log n + c\log\frac{n}{2} \\
&= \dots \\
&= T(1) + \sum_{k=0}^{\log n} c\log\frac{n}{2^k} \\
&= 1 + c\log^2 n - \frac{c\log^2 n}{2} \\
&= 1 + \frac{c\log^2 n}{2}
\end{aligned}
$$

3. 满足 $f(n) = cn \in \Omega(n^{E+\varepsilon}) \wedge f(n) \in O(n^{E+\delta})$, 其中 $\epsilon$ 可取 $0.8$, $\delta$ 可取 $1.2$
   So

$$T(n) \in \Theta(cn)$$

4. 满足

$$f(n) \in \Theta(n^E) = \Theta(n)$$

   So

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

5. Can' t not be solved by Master Theorm

$$T(n) = 2\left(\frac{n}{2}\right) + cn\log n$$

   Draw recursive Tree, we can get

$$T(n) = nT(1) + c\sum_{k=0}^{\log n} 2^k\frac{n}{2^k}\log\frac{n}{2^k} = n + c\sum_{k=0}^{\log n} n\log\frac{n}{2^k}$$

   Finally,

$$T(n) = n + cn\log^2 n - cn\frac{\log^2 n}{2} = n + \frac{cn\log^2 n}{2} = O(n\log^2 n)$$

6. Similarily

$$T(n) = nT(1) + \sum_{k=0}^{\log_3 n} 3^k\frac{n}{3^k} \times \log^3\frac{n}{3^k}$$

   So

$$
\begin{aligned}
T(n) &= n + \sum_{k=0}^{\log_3 n} n \times (\log n - k\log 3)^3 \\
&= n\sum_{k=0}^{\log_3 n}\left[(\log n)^3 - 3(\log n)^2(k\log 3) + 3(\log n)(k\log 3)^2 - (k\log 3)^3\right] \\
&= O(n\log^4 n)
\end{aligned}
$$

7. By MasterTheorm.
   $f(n) = cn^2 \in \Omega(n^{E+0.5})$, and $f(n) \in O(n^{E+2})$
   故 $T(n) \in \Theta(n^2)$

8. By MasterTheorm
   $f(n) = n^{3/2}\log n$, define $E = \frac{\log 49}{\log 25} = \frac{\log 7}{\log 5}$
   符合第三个条件

$$
T(n) \in \Theta(n^{3/2}\log n)
$$

9. $$
T(n) = T(n-1) + 2
$$

   Then

$$
\begin{aligned}
T(n) &= T(n-1) + 2 \\
&= T(n-2) + 2 + 2 \\
&= T(n-3) + 2 + 2 + 2 \\
&= \dots \\
&= T(1) + 2\times(n-1) = \Theta(n)
\end{aligned}
$$

10. $$
\begin{aligned}
T(n) &= T(n-1) + n^c \\
&= T(n-2) + n^c + (n-1)^c \\
&= \dots \\
&= T(1) + \sum_{k=2}^{n} k^c \\
&= \Theta(n^{c+1})
\end{aligned}
$$

11. $$
\begin{aligned}
T(n) &= T(n-1) + c^n \\
&= T(n-2) + c^n + c^{n-1} \\
&= \dots \\
&= T(1) + c^n + c^{n-1} + \dots + c^2 \\
&= \Theta(c^{n+1})
\end{aligned}
$$

12.

T(n)=T(n-2)+2n^{3}-3n^{2}+3n

主导项为$2n^3$故$T(n) \leq T(n-2) + 2n^3$ $$T(n) \leq \sum_{i=0}^{k-1} 2(n-2i)^3 \leq \int_0^k 2(n-2x)^3 dx = \int 2u^3 du =$$

增长率 $\Theta(n^4)$

13. ... 以后再来挑战没做出

## 2.18

计算 $T(n) = \sqrt{n}T(\sqrt{n}) + O(n)$ 复杂度
代换，Let $n = e^t$
则 $T(e^t) = e^{t/2}T(e^{t/2}) + O(e^t))$
则

$$\frac{T(e^t)}{e^t} = \frac{T(e^{t/2})}{e^{t/2}} + O(1)$$

令 $S(t) = \frac{T(e^t)}{e^t}$

$$S(t) = S\left(\frac{t}{2}\right) + O(1)$$

则 $S(t) \in \Theta(\log t) = \Theta(\log \log n)$
则 $T(n) = nS(t) \in n\Theta(S(t))$

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

## 2.19

$$a = 1,\ b = 2,\ f(n) = \log n$$

## 2.22

ALG 1: 输出 1
ALG 2：输出 1

时间复杂度:
ALG 1 $\Theta(n)$
ALG 2 $\Theta(n \log n)$

## 2.24

MYSTERY (n)

$$
\begin{aligned}
r &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j}1 \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} j \\
&= \sum_{i=1}^{n-1}\left(\frac{(i+1+n)(n-i)}{2}\right) \\
&= \frac{(n-1)n(n+1)}{3}
\end{aligned}
$$

返回结果为 $\frac{(n-1)n(n+1)}{3}$，由于算法每次操作只能使得 r:=r+1 且 r 初值为 0，所以时间复杂度和返回结果的大小同阶，为 $O(n^3)$

PERSKY (n)

$$
\begin{aligned}
r &= \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}1 \\
&= \sum_{i=1}^{n}\sum_{j=1}^{i} i \\
&= \sum_{i=1}^{n} i^2 \\
&= \frac{n(n+1)(2n+1)}{6}
\end{aligned}
$$

返回结果为 $\frac{n(n+1)(2n+1)}{6}$, 时间复杂度为 $O(n^3)$

PRESTIFEROUS (n)

$$
\begin{aligned}
r &= \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}\sum_{l=1}^{i+j-k}1 \\
&= \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}(i+j-k) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{i}\frac{(2i+j)(i+1)}{2} \\
&= \sum_{i=1}^{n}\left(\frac{(1+i)^2 i}{4}+i^3+i^2\right) \\
&= \frac{9}{4}\left(\frac{n(n+1)}{2}\right)^2+\frac{5}{2}\times\frac{n(n+1)(2n+1)}{6}+\frac{1}{4}\times\frac{(1+n)n}{2}
\end{aligned}
$$

CONUNDRUM (n)

$$
\begin{aligned}
r &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=i+j-1}^{n} 1 \\
&= \sum_{i=1}^{n} \sum_{j=i+1}^{n} (n-(i+j-1)+1) \\
&= \sum_{i=1}^{n} (n-3i+3)(n-i) \\
&= n^3 - 4n \times \frac{(n+1)n}{2} + 3n^2 - 3 \times \frac{n(n+1)}{2} + \frac{n(n+1)(2n+1)}{6}
\end{aligned}
$$

时间复杂度 $O(n^3)$

# Chapter 3 Brute Force

## 3.2 (Bubble Sort)

1. 证明冒泡排序的正确性
   Proof:
   先证明排序的每层 $i$ 循环，会保证 $[i-1, n-1]$ 是排好序的，即每次将 $[0, i-1]$ 中最大的元素放到 $i-1$ 位置

我们假设一个最大的元素 $m$, 其 index 为 $k$, 则根据循环算法，其必然会满足
$m > A[j], j \in [0, i-1] \wedge j \neq m$
则 $m$ 元素会依次和 $k+1, k+2 \ldots$ 交换，故最后其会落在 $i-1$

则循环每次能保证 $[i-1, n-1]$ 是排好序的
现在考虑 i down to 2，用反证法证明 $[i-1, n-1]$ is sorted then $[i-2, n-1]$ is sorted
假设 $[i-2, n-1]$ 是没有排好序，则由循环算法处理方式可知，只可能是 $A[i-2] > A[i-1]$ ，而这与

```python
if A[j]>A[j+1] then
    SWAP(A[j],A[j+1])
```

矛盾，故假设不成立

故递归下去，可知 $[0, n-1]$ 都会排好序。

2. 对于题中算法，我们每次遍历都不可避免的对相邻元素进行比较，与 input 无关

$$
T(n) = \sum_{i=2}^{n} \sum_{j=1}^{i-1} = \sum_{i=2}^{n} (i-1) = \frac{(1+n)(n-1)}{2}
$$

$T(n) \in \Theta(n^2)$
最坏和平均情况都是如此

3. 最坏情况下不影响：假设数组完全逆序，那么每次 k 出现在 i-1 处，和不加优化的情况下是一样的
平均情况：假设元素随机排列，那么这个优化使得每次 $k$ 出现的位置的数学期望为 $\frac{i-1}{2}$

$$T(n) = \sum_{i=2}^{n} \sum_{j=1}^{i-1} = \sum_{i=2}^{n} \left(\frac{i-1}{2}\right) = \frac{(1+n)(n-2)}{4}$$

$T(n) \in \Theta(n^2)$

## 3.5 PREVIIOUS-LARGER

$P[i]$ 记录了 $i$ 序列中位于 $a_i$ 左边且值比 $a_i$ 大的 index 最大的元素，我们通过利用 $P[i]$ 的记录可以降低复杂度水平。即：$arr[P[i+k]]$ 一定大于等于 $arr[P[i]]$，利用这一点可以避免不必要的查找。即我们对数组的遍历是不重复的，这样就达到的 $O(n)$ 级别。Code 实现如下

```python
def PREVIOUSLARGER(list arr)
    P[]=range(0,len(arr))#初始化为0,1,2,...,len-1
    for i:= 1 to n do:
        if arr[i]>=arr[P[i-1]]:#如果arr[i]大于arr[0:i-1]的最大元素
            P[i]=i-1 #则此位最大元素是本身，返回自身下标
        else:
            P[i]=P[i-1]
```

正确性证明：
首先显而易见：

$$arr[P[i+k]] \geq arr[P[i]], i, j \in Z^+$$

根据这一点可以推导出 $P[i]$ 是数组前 i 个最大的元素
用数学归纳法：
Base case: 当数组只有一个元素，显而易见算法返回正确值即 0（自己的下标）
I.H: 假设 $P[]$ 没有正确计算，则存在最小的不正确项之前都是正确的，假设 k 之前都是正确的。
若 $arr[k+1] \geq arr[P[k]]$，k+1 是最大元素下标，$P[k+1] = k$ 正确计算
否则，数组前 k+1 位的最大元素应该是数组前 k 位的最大元素，$P[k+1] = P[k]$，由于 $P[k]$ 已经正确计算，所以 $P[k+1]$ 也可以正确计算，q.e.d

# 3.6 数组左右交换位置

## 1 O (n^2), O (1)

```python
Def SwapPart(arr[],int k):
    for(int i=0;i<k;i++):
        for(int j=0;j<n;j++):
            if(i+k==j):
                SWAP(arr[i],arr[j])
```

## 2 O (n), O (n)

```python
Def SwapPart2(arr[],int k):
    arr1=[i for i in arr, 0<=i<k]
    arr2=[j for j in arr, k<=j<n]
    #cost time and space both in O(n)
    #compose two part into one
    for i in range(len(arr)):
        if(0<=i<k):
            arr[i]=arr1[i]
        else:
            arr[i]=arr2[i-k]
    #cost time O(n)
```

## 3.O (n), O (1)

```python
Def SwapPart3(arr[],int k):
    int left=0;
    int right=k;
    while(right<n&&left<k):
        SWAP(arr[left],arr[right])
        left+=1
        right+=1
```

## 3.8 Celebrity Problem

1. Max celebrity amount in n people.
   SOLUTION: 最多只有 1 个名人。
   反证法：假设有大于等于两个名人，任意取其中两个名人，记为 A 和 B.
   因为 $A$ 是名人，所以 $A$ 被所有人关注，因为 $B \in$ 所有人，所以 $B$ 关注 $A$. 而 $B$ 是名人，所以 $B$ 不关注任何人，矛盾。故最多只有 1 个名人。

2. 设计找出名人算法
   思路：
   Create two pointer: i and j. i is initialized as 0 (beginning), and j as n-1 (ended)
   In each loop (ended when i >= j), we check if j knows i.
   If j knows i, then j can' t be celebrity, we moved it back "1", that is j-=1. Obviously, this operatoins don' t affect the answer. We just move n-1 out of the candidates
   Else, then i can' t be celebrity. We process it similarily to the first case.

   When it ended, then c=i. c is the final candidate. Then we do a range (n) loop to check if c really know anyone else.

This algorithm made 2 loop over size n array, running in $O(n)$ time and $O(1)$ space

```python
def celebrity(mat):
    n = len(mat)
    i = 0
    j = n - 1
    while i < j:

        # j knows i, thus j can't be celebrity
        if mat[j][i] == 1:
            j -= 1

        # else i can't be celebrity
        else:
            i += 1

    # i points to our celebrity candidate
    c = i

    # Check if c is actually
    # a celebrity or not
    for i in range(n):
        if i == c:
            continue

        # If any person doesn't
        # know 'c' or 'c' doesn't
        # know any person, return -1
        if mat[c][i] or not mat[i][c]:
            return -1

    return c

if __name__ == "__main__":
    mat = [[0, 1, 0],
           [0, 0, 0],
           [0, 1, 0]]
    print(celebrity(mat))
```

## 3.9 最大和连续子序列

设计 O (n^3) 算法

```python
Def maxSeqSum(arr []):
    maxSum=-inf;
    for(int i=0;i<N;i++):#loop the start of seq
        for(int j=i;j<N;j++):#loop the end of seq
            thisSum=0;
            for(int k=i;k<j;k++):#loop SUM
                thisSum+=arr[k]
            if(thisSum>maxSum):
                maxSum=thisSum
    return maxSum;
```

设计 O (n^2) 基于遍历的算法

```python
Def maxSeqSum2 (arr[]) :
    MaxSum=-inf
    for(int i=0; i<N; i++):
        ThisSum=0
        for(int j=i;j<N;j++):
            ThisSum+=arr[j]
            if(ThisSum>MaxSum)
                MaxSum=ThisSum

    return MaxSum
```

设计分治的 O (nlogn) 算法

```python
Def maxSeqSum3(arr[],left,right):
    #base case
    if(right-left<=1):
        return arr[left]
    int center=(left+right)/2
    int leftMax=maxSeqSum3(arr,left,center-1)
    int rightMax=maxSeqSum3(arr,center+1,right)

    #calculate the center part max sum
    int leftBorderSum=0
    int maxLeftBorderSum=arr[center-1]
    int p=center-1
    for (int i = center - 1; i >= left; i--):
        leftBonderSum += center[i]
        if (maxLeftBonderSum < leftBonderSum):
            maxLeftBonderSum = leftBonderSum


    int rightBonderSum = 0
    int maxRightBonderSum = arr[center]
    for (int i = center; i < right; i++):
        rightBonderSum += arr[i]
        if (maxRightBonderSum < rightBonderSum):
            maxRightBonderSum = rightBonderSum
    return MAX(maxLeftBonderSum + maxRightBonderSum, maxLeftSum,
maxRightSum);
}
```

设计规避重复计算的线性算法 O (n)

```python
Def maxSeqSum4(arr[]):
    thisSum=maxSum=0;
    for(j=0;j<N;j++){
        thisSum+=arr[j];
        if(thisSum>maxSum):
            maxSum=thisSum
        else if(thisSum<0):
            #这段不可能是最优序列的前缀，所以重新选择
            ThisSum=0
    }
```

动态规划策略的线性算法
首先找到动态规划的状态转移方程：
用 $dp\left[i\right]$ 存储第 $i$ 位之前的最优解-index i 指向元素为结尾的所有连续子序列的最大和

$$dp[i]=\begin{cases}arr[0],\ i=0\\ \max\left\{(dp[i-1]+arr[i]),arr[i]\right\}\end{cases}$$

故算法为

```python
Def maxSeqSum5(arr[]):
    int dp[N]
    dp[0]=arr[0]
    ans=-inf
    for(i=1;i<len(arr);i++):
        dp[i]=MAX(dp[i-1]+arr[i],arr[i])
        ans=max(ans,dp[i])
    #ans 是 dp[i]中最大的一个，也就是所有不同结尾的最优子数组中最优的。
    return ans
```