## 第4章 分治策略

#### 练习 4.1-1



**问题分析**

返回数值最大的那个负数，即绝对值最小的负数。

#### 练习 4.1-2



**问题分析**

FIND\_MAXIMUM\_SUBARRAY(*A*, *low*, *high*)

*max\_sum* = -∞

**for** *i* = *low* 🡪 *high*

*sum* = 0

**for** *j* = *i* 🡪 *high*

*sum* += *A*[*j*]

**if** *sum* > *max\_sum*

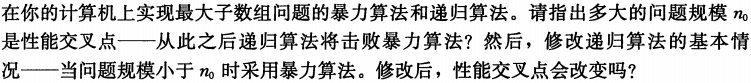
*max\_sum* = *sum*

*max\_left* = *i*

*max\_right* = *j*

**return** (*max\_left*, *max\_right*, *max\_sum*)

#### 练习 4.1-3



**问题分析**

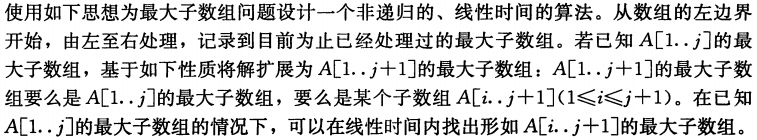
#### 练习 4.1-4



**问题分析**

先调用不允许结果为空子数组的算法，看看得到的结果。如果结果为负，那么就输出空子数组；如果结果不为负，那么将结果直接输出。

#### 练习 4.1-5



**问题分析**

形如*A*[*i* .. *j*+1]的最大子数组（包含*A*[*j*+1]）可能有2种情况：一种只包含*A*[*j*+1]，一种包含*A*[*j*+1]及之前的元素。下面分两种情况考虑：

① 如果形如*A*[*i* .. *j*]的最大子数组（包含*A*[*j*]）的和小于0，即*sum*[*i* .. *j*] < 0，那么形如*A*[*i* .. *j*+1]的最大子数组只能为*A*[*j*+1]。因为*sum*[*i* .. *j*] + *A*[*j*+1] < *A*[*j*+1]。

② 如果形如*A*[*i* .. *j*]的最大子数组的和不小于0，即*sum*[*i* .. *j*] ≥ 0，那么形如*A*[*i* .. *j*+1]的最大子数组只能为*A*[*i* .. *j*+1]。因为*sum*[*i* .. *j*] + *A*[*j*+1] ≥ *A*[*j*+1]。

由于*A*[1.. *j*+1]的最大子数组要么是*A*[1.. *j*]的最大子数组，要么是形如*A*[*i* .. *j*+1]的最大子数组。所以，*A*[1.. *j*+1]的最大子数组取二者中较大者。

**伪代码**

MAX\_SUBARRAY(*A*)

*max\_sum* = -∞ // *A*[1 .. *j*-1]的最大子数组的和

*cur\_sum* = -∞ // 形如*A*[*i* .. *j*]的最大子数组的和

**for** *i* = 1 🡪 *A*.*length*

**if** *cur\_sum* < 0

*cur\_sum* = *A*[*i*]

*cur\_low* = *i*

**else**

*cur\_sum* += *A*[*i*]

**if** *cur\_sum* > *max\_sum*

*max\_sum* = *cur\_sum*

*max\_low* = *cur\_low*

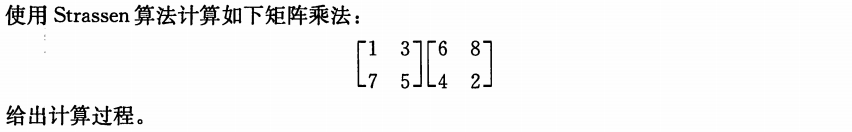
*max\_high* = *i*

**return** (*max\_low*, *max\_high*, *max\_sum*)

**时间分析**

上面的伪代码中，一共包含*n*次迭代，每次迭代花费Θ(1)时间。因此，算法的运行时间为Θ(*n*)。

#### 练习 4.2-1



**问题分析**

(1) 拆分原矩阵

*A*11 = 1 *A*12 = 3 *A*21 = 7 *A*22 = 5

*B*11 = 6 *B*12 = 8 *B*21 = 4 *B*22 = 2

(2) 计算矩阵*S*1、*S*2、…、*S*10

*S*1 = *B*12 – *B*22 = 8 – 2 = 6

*S*2 = *A*11 + *A*12 = 1 + 3 = 4

*S*3 = *A*21 + *A*22 = 7 + 5 = 12

*S*4 = *B*21 – *B*11 = 4 – 6 = –2

*S*5 = *A*11 + *A*22 = 1 + 5 = 6

*S*6 = *B*11 + *B*22 = 6 + 2 = 8

*S*7 = *A*12 – *A*22 = 3 – 5 = –2

*S*8 = *B*21 + *B*22 = 4 + 2 = 6

*S*9 = *A*11 – *A*21 = 1 – 7 = –6

*S*10 = *B*11 + *B*12 = 6 + 8 = 14

(3) 计算矩阵*P*1、*P*2、…、*P*7

*P*1 = *A*11∙*S*1 = 1∙6 = 6

*P*2 = *S*2∙*B*22 = 4∙2 = 8

*P*3 = *S*3∙*B*11 = 12∙6 = 72

*P*4 = *A*22∙*S*4 = 5∙(–2) = –10

*P*5 = *S*5∙*S*6 = 6∙8 = 48

*P*6 = *S*7∙*S*8 = (–2)∙6 = –12

*P*7 = *S*9∙*S*10 = (–6)∙14 = –84

(4) 计算矩阵*C*11、*C*12、*C*21和*C*22

*C*11 = *P*5 + *P*4 – *P*2 + *P*6 = 48 – 10 – 8 – 12 = 18

*C*12 = *P*1 + *P*2 = 6 + 8 = 14

*C*12 = *P*3 + *P*4 = 72 – 10 = 62

*C*22 = *P*5 + *P*1 – *P*3 – *P*7 = 48 + 6 – 72 + 84 = 66

最终的结果为



#### 练习 4.2-2



**问题分析**

这里考虑矩阵规模*n*不是2的幂的情况。需要考虑2种情况：

1. *n*是偶数

*n* x *n*矩阵可以划分为4个*n*/2 x *n*/2的子矩阵，可直接应用Strassen方法。为了计算矩阵乘积*C* = *A*•*B*，需要将矩阵划分为

   (1)

1. *n*是奇数

这种情况不能直接应用Strassen方法。为了计算矩阵乘积*C* = *A*•*B*，将矩阵做如下划分

   (2)

每个*n* x *n*矩阵都可以划分为一个(*n*−1) x (*n*−1)矩阵、一个(*n*−1) x 1矩阵、一个1 x (*n*−1)矩阵、一个1 x 1矩阵。并且有



由于(*n*−1)是偶数，所以矩阵乘法可以直接应用Strassen算法，这一步的时间为



矩阵乘法采用朴素算法，需要做(*n*−1)2次乘法，其运行时间为Θ(*n*2)。

矩阵乘法采用朴素算法，需要做(*n*−1)2次乘法，以及(*n*−1)(*n*−2)次加法，其运行时间为Θ(*n*2)。

矩阵乘法采用朴素算法，需要做(*n*−1)次乘法，其运行时间为Θ(*n*)。

矩阵乘法采用朴素算法，需要做(*n*−1)2次乘法，以及(*n*−1)(*n*−2)次加法，其运行时间为Θ(*n*2)。

矩阵乘法采用朴素算法，需要做(*n*−1)次乘法，其运行时间为Θ(*n*)。

矩阵乘法采用朴素算法，需要做(*n*−1)次乘法，以及(*n*−2)次加法，其运行时间为Θ(*n*)。

矩阵乘法采用朴素算法，只需要做1次乘法，其运行时间为Θ(1)。

计算时，矩阵乘积与矩阵乘积相加需要做(*n*−1)2次加法，其运行时间为Θ(*n*2)。

计算时，矩阵乘积与矩阵乘积相加需要做(*n*−1)次加法，其运行时间为Θ(*n*)。

计算时，矩阵乘积与矩阵乘积相加需要做(*n*−1)次加法，其运行时间为Θ(*n*)。

计算时，矩阵乘积与矩阵乘积相加只需要做1次加法，其运行时间为Θ(1)。

除去计算之外，其他部分的运行时为Θ(*n*2)。因此，当*n*是奇数时，计算*n* x *n*矩阵乘积的运行时间为



求解这个递归式得到。

NAIVE\_MATRIX\_MULTIPLY(*A*, *B*)

*m* = *A*.*rows*

*p* = *A*.*columns*

*n* = *B*.*columns*

let *C* be a new *m*x*n* matrix

**for** *i* = 1 🡪 *m*

**for** *j* = 1 🡪 *n*

*C*[*i*, *j*] = 0

**for** *k* = 1 🡪 *p*

*C*[*i*, *j*] += *A*[*i*, *k*]·*B*[*k*, *j*]

**return** *C*

MATRIX\_ADD(*A*, *B*)

*m* = *A*.*rows*

*n* = *A*.*columns*

let *C* be a new *m*x*n* matrix

**for** *i* = 1 🡪 *m*

**for** *j* = 1 🡪 *n*

*C*[*i*, *j*] += *A*[*i*, *j*] + *B*[*i*, *j*]

**return** *C*

MATRIX\_SUB(*A*, *B*)

*m* = *A*.*rows*

*n* = *A*.*columns*

let *C* be a new *m*x*n* matrix

**for** *i* = 1 🡪 *m*

**for** *j* = 1 🡪 *n*

*C*[*i*, *j*] += *A*[*i*, *j*] - *B*[*i*, *j*]

**return** *C*

STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*A*, *B*)

*n* = *A*.*rows*

let *C* be a new *n*x*n* matrix

**if** *n* == 1

*C*[1, 1] = *A*[1, 1]·*B*[1, 1]

**elseif** *n* is odd

partition *A*, *B* and *C* as in equations (2)

*C*1 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*A*1, *B*1)

+ NAIVE\_MATRIX\_MULTIPLY(*A*2, *B*3)

*C*2 = NAIVE\_MATRIX\_MULTIPLY (*A*1, *B*2) + NAIVE\_MATRIX\_MULTIPLY(*A*2, *B*4)

*C*3 = NAIVE\_MATRIX\_MULTIPLY (*A*3, *B*1) + NAIVE\_MATRIX\_MULTIPLY(*A*4, *B*3)

*C*4 = NAIVE\_MATRIX\_MULTIPLY (*A*3, *B*2) + NAIVE\_MATRIX\_MULTIPLY(*A*4, *B*4)

**else**

partition *A*, *B* and *C* as in equations (1)

*S*1 = MATRIX\_SUB(*B*12, *B*22)

*S*2 = MATRIX\_ADD(*A*11, *A*12)

*S*3 = MATRIX\_ADD(*A*21, *A*22)

*S*4 = MATRIX\_SUB(*B*21, *B*11)

*S*5 = MATRIX\_ADD(*A*11, *A*22)

*S*6 = MATRIX\_ADD(*B*11, *B*22)

*S*7 = MATRIX\_SUB(*A*12, *A*22)

*S*8 = MATRIX\_ADD(*B*21, *B*22)

*S*9 = MATRIX\_SUB(*A*11, *A*21)

*S*10 = MATRIX\_ADD(*B*11, *B*12)

*P*1 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*A*11, *S*1)

*P*2 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*S*2, *B*22)

*P*3 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*S*3, *B*11)

*P*4 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*A*22, *S*4)

*P*5 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*S*5, *S*6)

*P*6 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*S*7, *S*8)

*P*7 = STRASSEN\_SQUARE\_MATRIX\_MULTIPLY(*S*9, *S*10)

*C*11 = MATRIX\_ADD(MATRIX\_SUB(MATRIX\_ADD(*P*5, *P*4), *P*2), *P*6)

*C*12 = MATRIX\_ADD(*P*1, *P*2)

*C*21 = MATRIX\_ADD(*P*3, *P*4)

*C*22 = MATRIX\_SUB(MATRIX\_SUB(MATRIX\_ADD(*P*5, *P*1), *P*3), *P*7)

**return** C

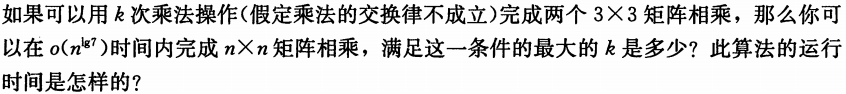
#### 练习 4.2-3



**问题分析**

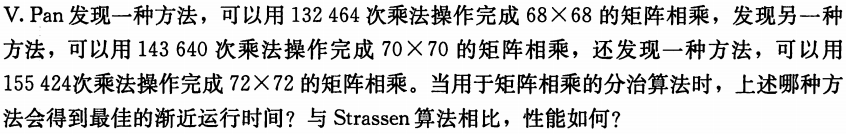
见练习4.2-2。

#### 练习 4.2-4



**问题分析**

#### 练习 4.2-5



**问题分析**

#### 练习 4.2-6



**问题分析**

根据题意，假设两个矩阵*A*(*kn*)x*n*和*Bn*x(*kn*)相乘，得到矩阵*C*(*kn*)x(*kn*)。如果要利用Strassen算法，则需要将矩阵*A*、*B*和*C*按下面的方式划分

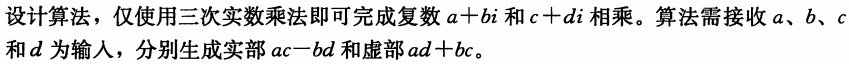
很明显有*Cij* = *Ai*·*Bj*。*Ai*·*Bj*是一个*n* x *n*矩阵乘法，采用Strassen算法，运行时间为Θ(*n*lg7)。一共有*k*2个这样的*n* x *n*矩阵乘法，所以总的运行时间为Θ(*k*2*n*lg7)。

如果将输入矩阵的规模互换，即矩阵*An*x(*kn*)和*B*(*kn*)x*n*相乘，得到矩阵*Cn*x*n*，那么需要将矩阵*A*和*B*按下面的方式划分

很明显有。一共有*k*个*n* x *n*矩阵乘法，并且还有(*k*-1)个*n* x *n*矩阵加法，所以运行时间为Θ(*kn*lg7)。

#### 练习 4.2-7



**问题分析**

借鉴Strassen算法的思想，该问题可以按以下步骤解决。

(1) 计算*P*1、*P*2和*P*3

*P*1 = *ad*

*P*2 = *bc*

*P*3 = (*a* – *b*)(*c* + *d*) = *ac* – *bd* + *ad* – *bc*

(2) 计算*ac* – *bd*和*ad* + *bc*

*ac* – *bd* = *P*3 – *P*1 + *P*2

*ad* + *bc* = *P*1 + *P*2

该算法只需要3次乘法即可。

#### 练习 4.3-1



**问题分析**

取边界条件*T*(1) = 1，只要取任何*c* ≥ 1，就满足*T*(1) ≤ *c*∙12。因此命题对*n* = 1成立。

当n ≥ 2时，假设命题对所有*m* < *n*都成立。考虑*T*(*n*)，有



要使不等式成立*cn*2 – (2*n*–1)*c* + *n* ≤ *cn*2成立，只需要让(2*n*–1)*c* ≥ *n*成立，即*c* ≥ *n*/(2*n*–1) = 1/(2–1/*n*)。当n ≥ 2时，1/(2–1/*n*)的最大值为2/3(当*n* = 2时)。因此，只要取*c* ≥ 2/3，就可使不等式*cn*2 – (2*n*–1)*c* + *n* ≤ *cn*2成立，此时*T*(*n*) ≤ *cn*2成立。考虑到边界条件*n* = 1，取*c* ≥ 1。

因此，*T*(*n*) = *T*(*n*–1) + *n*的解为O(*n*2)。

#### 练习 4.3-2



**问题分析**

令*T*(1) = 1，则*T*(2) = 2。对于*n* = 1，无论*c*取何值，都不能满足*T*(1) ≤ *c*∙lg1。但只要取*c* ≥ 2，就能满足*T*(2) ≤ *c*∙lg2。因此命题对*n* = 2成立，以*n* = 2为边界条件。

当*n* ≥ 3时，假设命题对所有*m* < *n*都成立。考虑*T*(*n*)，有



要使不等式成立，只需要让成立，于是有



当*n* ≥ 3时，的最大值为1/(lg3–1)(当*n* = 3时)。因此，只要取*c* ≥ 2，就可使不等式成立，此时*T*(*n*) ≤ *c*lg*n*成立。

因此的解为O(lg*n*)。

#### 练习 4.3-3



**问题分析**

取边界条件*T*(1) = 1。无论*c*取何值，都能满足*T*(1) ≥ *c*∙1∙lg1。

当*n* ≥ 2时，假设命题对所有*m* < *n*都成立。考虑*T*(*n*)，有



我们的目标是要使不等式成立。









显然，当*n* ≥ 2时，有2(*n*-1) ≥ *n*，所以有。如果能够选取合适的*c*，使得不等式成立，那么就能使不等式成立。









当*n* ≥ 2时，显然成立，所以有。所以只要取*c* ≤ 1/3，就能使不等式成立，此时不等式也成立，所以*T*(*n*) ≥ *cn*lg*n*也成立，即*T*(*n*) = Ω(*n*lg*n*)。

#### 练习 4.3-4





**问题分析**

为O(*n*lg*n*)加上一个低阶项，得O(*n*lg*n*+*dn*)。

取边界条件*T*(1) = 1。只要取*c* ≥ 1和*d* ≥ 1，都能满足*T*(1) ≤ *c*(1∙lg1+*d*∙1)。

当*n* ≥ 2时，假设命题对所有*m* < *n*都成立。考虑*T*(*n*)，有



我们的目标是要使不等式成立。

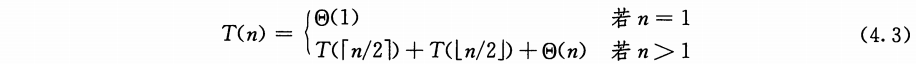




显然，只要取*c* ≥ 1， 不等式恒成立，所以成立。所以*T*(*n*) ≤ *n*lg*n*+*dn*，于是有*T*(*n*) = O(*n*lg*n*+*dn*) = O(*n*lg*n*)。

#### 练习 4.3-5





**问题分析**

(1) 先证*T*(*n*) = O(*n*lg*n*)

令*T*(1) = 1，则*T*(2) = 2+Θ(2) ≤ 2 + 2*c*2*’*，其中*c*2*’* > 0。对于*n* = 1，无论*c*2取何值，都不能满足*T*(1) ≤ *c*2∙1∙lg1。但只要取*c*2 ≥ *c*2*’* + 1，就能满足*T*(2) ≤ *c*2∙2∙lg2。因此命题对*n* = 2成立，以*n* = 2为边界条件。

当*n* ≥ 3时，假设*T*(*m*) = O(*m*lg*m*)对所有*m* < *n*都成立。考虑*T*(*n*)，有



我们的目标是使不等式成立。









（归纳步骤从*n* = 3开始，所以，这一步转换成立）

当*n* ≥ 3时，的最大值为*c*2*’*/(1–lg(4/3)) < 2 *c*2*’*(当*n* = 3时)。因此，只要取*c*2 ≥ 2*c*2*’*，就可使不等式成立，此时不等式也成立。综合边界条件，只要取*c*2 ≥ max{2*c*2*’*, *c*2*’*+1}，就可保证*T*(*n*) ≤ *c*2*n*lg*n*。故*T*(*n*) = O(*n*lg*n*)。

1. 再证*T*(*n*) = Ω(*n*lg*n*)

取边界条件*T*(1) = 1，无论*c*1取任何大于0的值，都能满足*T*(1) ≥ *c*1∙1∙lg1。

当*n* ≥ 2时，假设*T*(*m*) = Ω(*m*lg*m*)对所有*m* < *n*都成立。考虑*T*(*n*)，有

 （*c*1*’* > 0）

我们的目标是使不等式成立。











（归纳步骤从*n* = 2开始，所以，这一步转换成立）

当*n* ≥ 2时，的最小值为*c*1*’*/2(当*n* = 2时)。因此，只要取*c*1 ≤ *c*1*’*/2，就可使不等式成立，此时不等式也成立。故只要取*c*1 ≤ *c*1*’*/2，就可保证*T*(*n*) ≥ *c*1*n*lg*n*。故*T*(*n*) = Ω(*n*lg*n*)。

综合(1)和(2)，*T*(*n*) = Θ(*n*lg*n*)。

#### 练习 4.3-6



**问题分析**

初始情况必须覆盖*n* = 1, 2, … , 34，取*T*(1) = *T*(2) = … = *T*(34) = 1。

显然，对于*n* = 2, 3, … 34，只要取*c* ≥ 1，就满足*T*(*n*) ≤ *cn*lg*n*。

当*n* ≥ 35时，假设*T*(*m*) = O(*m*lg*m*)对所有*m* < *n*都成立。考虑*T*(*n*)，有



我们的目标是要证明。





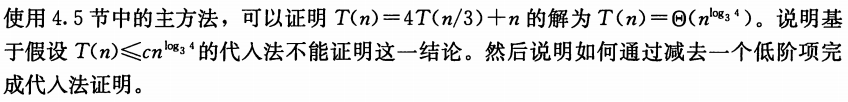




（由于，所以）

由于，所以只要取*c* ≥ 1，就可保证不等式成立，此时不等式也成立。故只要取*c* ≥ 1，就可保证*T*(*n*) ≤ *cn*lg*n*。故*T*(*n*) = O(*n*lg*n*)。

#### 练习 4.3-7



**问题分析**

取边界条件*T*(1) = 1，*T*(2) = 1。只要取*c* ≥ 1，就可使得和成立。

当*n* ≥ 3时，假设对所有*m* < *n*都成立。考虑*T*(*n*)，有



无论c取何值，都不能使成立。

重新做归纳假设，减去一个低阶项*dn*(其中*d* > 0)，于是假设。

取边界条件*T*(1) = 1，*T*(2) = 1。只要取*d* ≤ 1/2并且*c* ≥ 2，就可使得和成立。

当*n* ≥ 3时，假设这个不等式所有*m* < *n*都成立。考虑*T*(*n*)，有



我们的目标是使不等式成立。



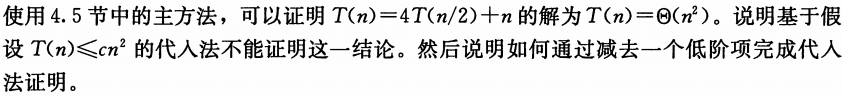




所以，只要让，就可使不等式成立，此时也成立。由于我们选择*d* ≤ 1/2，故，故边界条件中选取*c* ≥ 2也隐含在中。

所以。

#### 练习 4.3-8



**问题分析**

取边界条件*T*(1) = 1。只要取*c* ≥ 1，就可使得成立。

当*n* ≥ 2时，假设对所有*m* < *n*都成立。考虑*T*(*n*)，有



无论*c*取何值，都不能使成立。

重新做归纳假设，减去一个低阶项*dn*(其中*d* > 0)，于是假设。

取边界条件*T*(1) = 1。只要取*d* ≤ 1/2并且*c* ≥ 2，就可使得成立。

当*n* ≥ 2时，假设这个不等式所有*m* < *n*都成立。考虑*T*(*n*)，有



我们的目标是使不等式成立。







所以，只要让，就可使不等式成立，此时也成立。由于我们选择*d* ≤ 1/2，故，故边界条件中选取*c* ≥ 2也隐含在中。

所以。

#### 练习 4.3-9



**问题分析**

令*m* = lg*n*，所以*n* = 2*m*，代入递归式，得到

*T*(2*m*) = 3*T*(2*m*/2) + *m*

令*S*(*m*) = *T*(2*m*)，得到

*S*(*m*) = 3*S*(*m*/2) + *m*

求解递归式*S*(*m*) = 3*S*(*m*/2) + *m*，得到*S*(*m*) = Θ(*m*lg3)。

所以*T*(*n*) = *T*(2*m*) = *S*(*m*) = Θ(*m*lg3) = Θ((lg*n*)lg3)。

#### 练习 4.4-1



**问题分析**

*n*

*n*/2

*n*/2

*n*/2

*n*/4

*n*/4

*n*/4

*n*/4

*n*/4

*n*/4

**∙ ∙ ∙**

*T*(1)

*T*(1)

*T*(1)

*T*(1)

*T*(1)

*T*(1)

**∙ ∙ ∙**

*T*(1)

*T*(1)

*n*

(3/2)*n*

(3/2)2*n*

Θ(*n*lg3)



#### 练习 4.4-2



**问题分析**

*n*

*n*/2

*n*2

(*n*/2)2

(*n*/4)2

Θ(1)

*n*/4

*T*(1)



#### 练习 4.4-3



**问题分析**

初始情况应当为*n* =1~ 5，取*T*(1) = *T*(2) = *T*(3) = *T*(4) = *T*(5) = Θ(1)。

假设递归树的深度为*m*，则有(*n*/2+2)(*m*) = 5。



得到*m* = lg(*n*-4)。

下面分析递归树每层的情况。

|  |  |
| --- | --- |
| 0 | *n* |
| 1 | (*n*/2+2)∙4 = 2*n*+8 |
| 2 | (*n*/4+2+1)∙42 = 4*n*+48 |
| 3 | (*n*/8+2+1+1/2)∙43 = 8*n*+224 |
| 4 | (*n*/16+2+1+1/2+1/4)∙44 = 16*n*+960 |
| … … | … … |
| lg(*n*-4) | 4lg(*n*-4)∙Θ(1) = (*n*-4)2∙Θ(1) = Θ(*n*2) |

从以上分析可以看出，除最后一层外，第*i*层的时间为



由此可计算*T*(*n*)，



#### 练习 4.4-4



**问题分析**

*n*

*n*-1

1

1

1

Θ(1)

*n*-2

*T*(1)



#### 练习 4.4-5



**问题分析**

#### 练习 4.4-6



**问题分析**

*cn*

*cn*/3

2*cn*/3

*cn*/9

2*cn*/9

*cn*

c*n*

*cn*

2*cn*/9

4*cn*/9

这棵树不是一棵完全二叉树。假设树从第*i*层开始往上都是满的，从第*i*+1层开始往下是满的，则有*n*(1/3)*i* = 1，求解得到*i* = log3*n*。于是有



所以*T*(*n*) = Ω(*n*lg*n*)。

#### 练习 4.4-7



**问题分析**

*cn*

*cn*/2

*cn*/2

*cn*/2

*cn*/4

*cn*/4

*cn*/4

**∙ ∙ ∙**

*T*(1)

*T*(1)

*T*(1)

*T*(1)

*T*(1)

*T*(1)

**∙ ∙ ∙**

*T*(1)

*T*(1)

*cn*

2c*n*

22*cn*

Θ(*n*2)

*cn*/2

*cn*/4

*cn*/4

*cn*/4

*cn*/4

*cn*/4



#### 练习 4.4-8



**问题分析**

*cn*

*c*(*n*−*a*)

*cn*

*cn*−2*ca*

*cn*−3*ca*

*cn*−*c*(*n*/*a*−2)*a*

*ca*

*c*(*n*−2*a*)

*ca*

*c*(*n*−(*n*/*a*−1)*a*)

*ca*



#### 练习 4.4-9



**问题分析**

*cn*

*cαn*

*c*(1−*α*)*n*

*cα*2*n*

*cα*(1−*α*)*n*

*cn*

c*n*

*cn*

*cα*(1−*α*)*n*

*c*(1−*α*)2*n*

可以发现树的每一层的代价均是*cn*，然而这棵树并非是一棵完全二叉树（只有当*α* = 1/2时才是完全二叉树），树的底部有若干层不是满的，这些层的代价小于*cn*。假如最底层为*n* = 1，并且树的高度为*i*，则有*n*(max{*α*, 1−*α*})*i* = 1，求解得到*i* = −lg*n*/lg(max{*α*, 1−*α*})。于是有



所以*T*(*n*) = O(*n*lg*n*)。

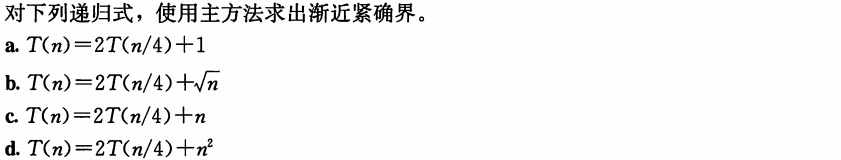
假设树从第*j*层开始往上都是满的，从第*j*+1层开始往下是满的，则有*n*(min{*α*, 1−*α*})*j* = 1，求解得到*j* = −lg*n*/lg(min{*α*, 1−*α*})。于是有



所以*T*(*n*) = Ω(*n*lg*n*)。

综上所述，得到*T*(*n*) = Θ(*n*lg*n*)。

#### 练习 4.5-1



**问题分析**

a. 



应用主定理情况1，得到*T*(*n*) = Θ(*n*1/2)。

b. 



应用主定理情况2，得到*T*(*n*) = Θ(*n*1/2lg*n*)。

c. 



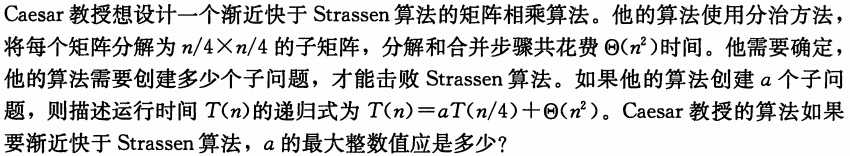
应用主定理情况3，得到*T*(*n*) = Θ(*n*)。

d. 



应用主定理情况3，得到*T*(*n*) = Θ(*n*2)。

#### 练习 4.5-2



**问题分析**

用主方法求解递归式*T*(*n*) = *aT*(*n*/4) + Θ(*n*2)。



Strassen算法的时间为*n*lg7。要让新算法的时间小于*n*lg7，必须满足，得到*a* < 49。因此，*a*的最大值为48。

#### 练习 4.5-3



**问题分析**



应用主定理情况2，得到*T*(*n*) = Θ(lg*n*)。

#### 练习 4.5-4



**问题分析**



显然，*n*2lg*n*不是Ω(*n*2+*ε*)，故该递归式不能运用主方法求解。

运用递归树方法求解。

*n*2lg*n*

(*n*2/4)lg(*n*/2)

*T*(1)

*T*(1)

*T*(1)

*T*(1)

*T*(1)

*T*(1)

**∙ ∙ ∙**

*T*(1)

*T*(1)

*n*2lg*n*

*n*2lg(*n/*2)

*n*2lg(*n/*4)

*n*2

(*n*2/4)lg(*n*/2)

**∙ ∙ ∙**

(*n*2/16)lg(*n*/4)

(*n*2/16)lg(*n*/4)

(*n*2/16)lg(*n*/4)

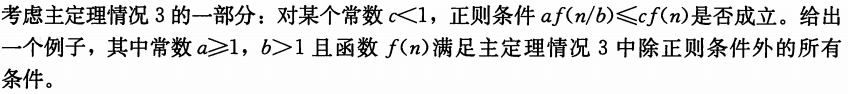
(*n*2/16)lg(*n*/4)

**∙ ∙ ∙**

**∙ ∙ ∙**

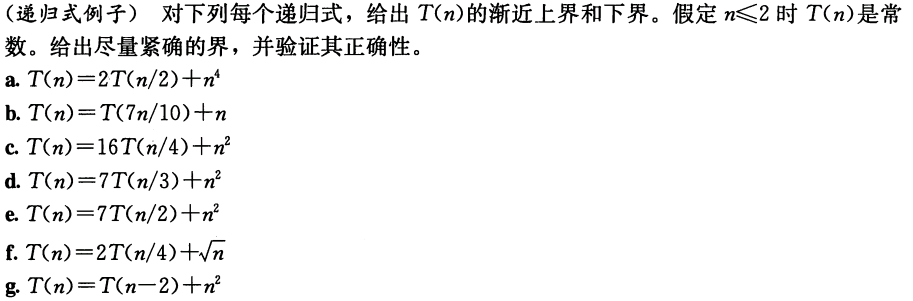


#### 练习 4.5-5\*



**问题分析**

#### 思考题 4-1



**问题分析**

a. 

因为*a* = 2, *b* = 2，所以。于是有*f*(*n*) = *n*4 = Ω(*n*1+ε)，并且*af*(*n*/*b*) = 2*f*(*n*/2) = 2∙(*n*/2)4 = *n*4/8 ≤ *cf*(*n*) = *cn*4（当1/8 ≤ c < 1时）。

应用主定理情况3，得到*T*(*n*) = Θ(*f*(*n*)) = Θ(*n*4)。

b. 

因为*a* = 1, *b* = 7/10，所以。于是有*f*(*n*) = *n* = Ω(*n*0+ε)，并且*af*(*n*/*b*) = *f*(7*n*/10) = 7*n*/10 ≤ *cf*(*n*) = *cn*（当7/10 ≤ c < 1时）。

应用主定理情况3，得到*T*(*n*) = Θ(*f*(*n*)) = Θ(*n*)。

c. 

因为*a* = 16, *b* = 4，所以。于是有*f*(*n*) = *n*2 = Θ(*n*2)。

应用主定理情况2，得到*T*(*n*) = Θ(*n*2lg*n*)。

d. 

因为*a* = 7, *b* = 3，所以。于是有*f*(*n*) = *n*2 = Ω()，并且*af*(*n*/*b*) = 7*f*(*n*/3) = 7∙(*n*/3)2 = 7*n*2/9 ≤ *cf*(*n*) = *cn*2（当7/9 ≤ c < 1时）。

应用主定理情况3，得到*T*(*n*) = Θ(*f*(*n*)) = Θ(*n*2)。

e. 

因为*a* = 7, *b* = 2，所以。于是有*f*(*n*) = *n*2 = O()。

应用主定理情况1，得到*T*(*n*) = Θ(*n*lg7)。

f. 

因为*a* = 2, *b* = 4，所以。于是有*f*(*n*) == Θ(*n*1/2)。

应用主定理情况2，得到Θ(*n*1/2lg*n*)。

g. 

应用递归树方法求解。

*n*2

(*n*−2)2

*n*2

(*n*−2)2

(*n*−4)2

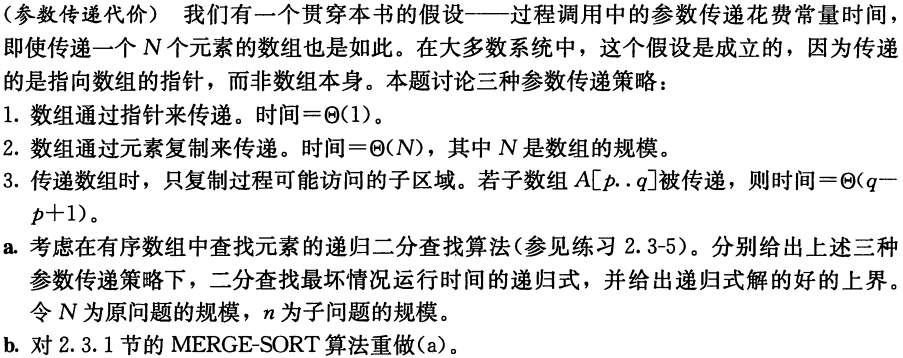
Θ(1)

(*n*−4)2

*T*(2)



#### 思考题 4-2



**a-问题分析**

1. 通过指针

递归式为*T*(*n*) = *T*(*n*/2) + Θ(1)，解为*T*(*n*) = Θ(lg*n*)。

2. 通过复制所有元素

递归式为*T*(*n*) = *T*(*n*/2) + Θ(*N*)，*N*为原数组规模，解为*T*(*n*) = Θ(*n*lg*n*)。

3. 通过复制可能访问的元素

递归式为*T*(*n*) = *T*(*n*/2) + Θ(*n*/2) = *T*(*n*/2) + Θ(*n*)，解为*T*(*n*) = Θ(*n*)。

**b-问题分析**

1．通过指针

递归式为*T*(*n*) = 2*T*(*n*/2) + Θ(*n*) + Θ(1) = 2*T*(*n*/2) + Θ(*n*)，解为*T*(*n*) = Θ(*n*lg*n*)。

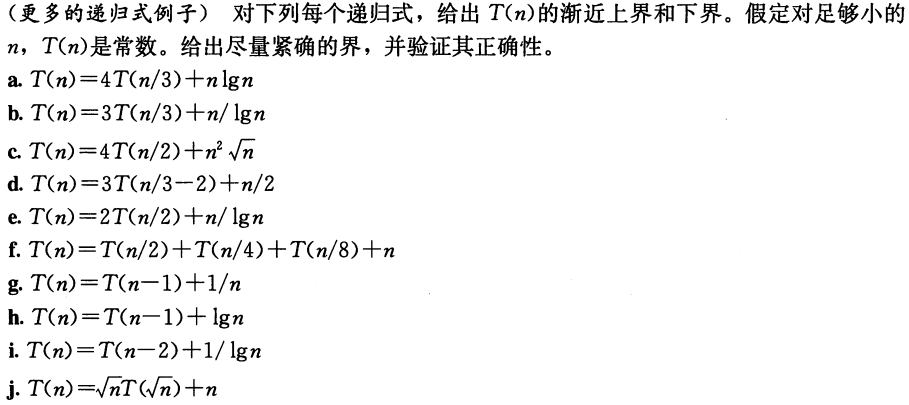
2. 通过复制所有元素

递归式为*T*(*n*) = 2*T*(*n*/2) + Θ(*n*) + Θ(*N*)，*N*为原数组规模，解为*T*(*n*) = Θ(*n*2)。

3. 通过复制可能访问的元素

递归式为*T*(*n*) = 2*T*(*n*/2) = 2*T*(*n*/2) + Θ(*n*) + Θ(*n*) = 2*T*(*n*/2) + Θ(*n*)，解为*T*(*n*) = Θ(*n*lg*n*)。

#### 思考题 4-3



**问题分析**

a. 

因为*a* = 4, *b* = 3，所以。于是有*f*(*n*) = *n*lg*n* = O()。

应用主定理情况1，得到*T*(*n*) = Θ()。

b. 

用递归树方法求解。

*n*/lg*n*

(*n*/3)/lg(*n*/3)

(*n*/3)/lg(*n*/3)

(*n*/9)/lg(*n*/9)

(*n*/9)/lg(*n*/9)

*n*/lg*n*

*n*/(lg*n*−lg3)

*n*/(lg*n*−lg9)

(*n*/3)/lg(*n*/3)

(*n*/9)/lg(*n*/9)

**∙ ∙ ∙**

*T*(1)

*T*(1)

*T*(1)

**∙ ∙ ∙**

**∙ ∙ ∙**

Θ(*n*)



c. 

因为*a* = 4, *b* = 2，所以。于是有*f*(*n*) == *n*2.5 = Ω()，，并且*af*(*n*/*b*) = 4*f*(*n*/2) = 4∙(*n*/2)2.5 = ≤ *cf*(*n*) = *cn*2.5（当时）。

应用主定理情况3，得到*T*(*n*) = Θ(*f*(*n*)) = Θ(*n*2.5)。

d. 

猜想这个递归式的解应该和递归式的解一致，用主方法求解这个新的递归式得到*T*(*n*) = Θ(*n*lg*n*)。再用代入法进行验证，这里省去验证过程。

e. 

*T*(*n*) = Θ(*n*lglg*n*)。证明过程与递归式b类似。

f. 

采用递归树方法求解。

*n*

*n*/2

*n*/8

*n*/4

*n*/16

*n*

(7/8)*n*

(7/8)2*n*

*n*/4

*n*/8

**∙ ∙ ∙**

**∙ ∙ ∙**

*n*/8

*n*/32

*n*/16

*n*/16

*n*/64

*n*/32

这不是一棵完全二叉树。从第0层到第层是满的，从第层到最底层是不满的。考虑从第0层到第层的代价，有



所以可以得到*T*(*n*)的下界，*T*(*n*) = Ω(*n*)。下面用代入法验证。

以*n* = 1为边界条件，*T*(1) = 1。很显然，只要取*c* ≤ 1，就能得到*T*(*n*) ≥ *c*∙1。

当*n* ≥ 2时，假设*T*(*m*) ≥ *cm*对所有*m* < *n*都成立，于是有



很显然，只要取*c* ≤ 8，就能得到，即*T*(*n*) ≥ *cn*。考虑边界条件，取*c* ≤ 1。所以*T*(*n*) = Ω(*n*)。

上面得到*T*(*n*)的下界为*T*(*n*) = Ω(*n*)，很有可能*T*(*n*)的上界也为*T*(*n*) = O(*n*)。下面用代入法验证。

以*n* = 1为边界条件，*T*(1) = 1。很显然，只要取*c* ≥ 1，就能得到*T*(*n*) ≤ *c*∙1。

当*n* ≥ 2时，假设*T*(*m*) ≤ *cm*对所有*m* < *n*都成立，于是有



很显然，只要取*c* ≥ 8，就能得到，即*T*(*n*) ≤ *cn*。所以*T*(*n*) = O(*n*)。

综上所述，*T*(*n*) = Θ(*n*)。

g. 

采用递归树方法求解。

1/*n*

1/(*n*−1)

1/*n*

1/(*n*−1)

1/(*n*−2)

Θ(1)

1/(*n*−2)

*T*(1)



h. 

采用递归树方法求解。

lg*n*

lg(*n*−1)

lg*n*

lg(*n*−1)

lg(*n*−2)

Θ(1)

lg(*n*−2)

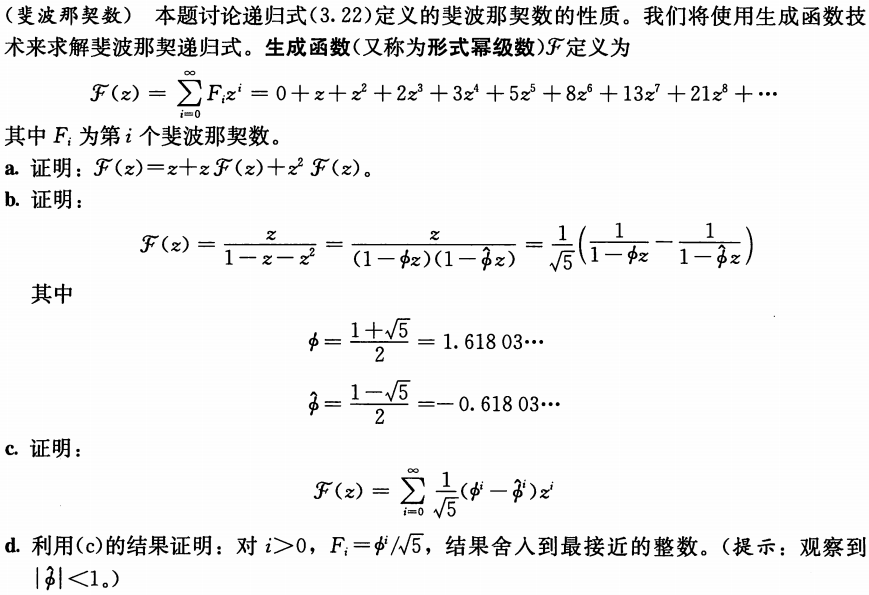
*T*(1)



i. 

j. 

#### 思考题 4-4



**a-问题分析**



**b-问题分析**

根据问题a的结论，*F*(*z*) = *z* + *zF*(*z*) + *z*2*F*(*z*)，得到



**c-问题分析**



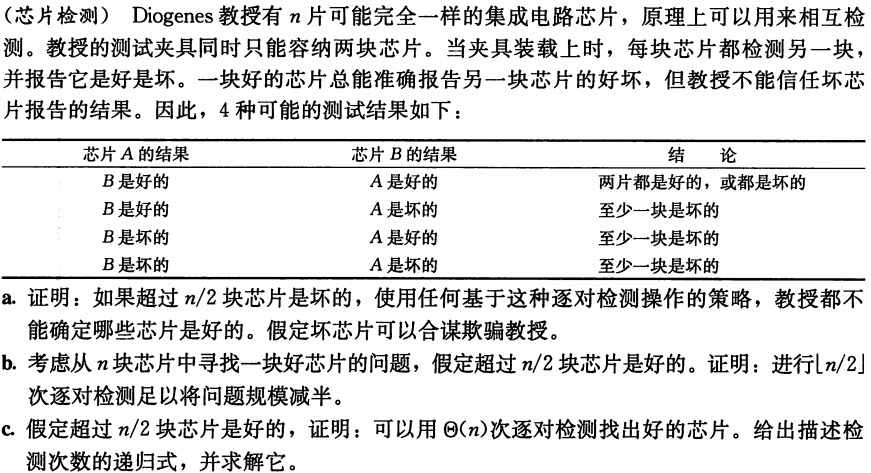
**d-问题分析**

根据问题c的结论，有。

当*i* = 1时，*Fi* = 1，此时，所以，将舍入到最接近的整数恰好是1，命题成立。

当*i* ≥ 2时，可以得到，所以将舍入到最接近的整数相当于，命题也成立。

#### 思考题 4-5



**a-问题分析**

**b-问题分析**

一对芯片的检测结果可以分为两种情况：

(1) 至少一个坏的

一对芯片中至少有一个是坏的，要么两个都坏，要么一坏一好。针对这种情况，可以把两个芯片都除去。如果两个芯片都是坏的，那么在除去两个坏芯片后，剩余的芯片中好芯片仍然比坏芯片多。如果两个芯片一坏一好，那么好芯片和坏芯片各被除去一个，剩余的芯片中好芯片仍然比坏芯片多。于是得到了一个规模更小的子问题。

(2) 全为好

一对芯片要么都是好的，要么都是坏的。

假设在两两检测的芯片对中，坏坏组合的对数为*pb*，好好组合的对数为*pg*，而好坏组合的对数为*pc*。如果芯片总数*n*为偶数，有；如果*n*为奇数，有，还有一片落单的芯片。

如果*n*为偶数，没有落单芯片，好芯片个数*ng* = 2*pg* + *pc*，坏芯片个数*nb* = 2*pb* + *pc*。*n*为偶数时，好芯片至少比坏芯片多2个，即*ng* ≥ *nb* + 2，即2*pg* + *pc* ≥ 2*pb* + *pc* + 2，可得*pg* ≥ *pb* + 1，这说明好好组合至少比坏坏组合多1个。所以，在最坏情况下，所有坏坏组合的检测结果为全好，如果将检测结果为全好的组合中去掉一个，保留一个，那么剩下的芯片中好芯片也比坏芯片多。这样就可得到一个规模更小的子问题。

如果*n*为奇数，会有一块芯片落单。针对除落单芯片之外的其他芯片，采用与前面相前的处理方法，即将检测结果为全好的组合中去掉一个。下面讨论针对落单芯片的处理方法：

① 如果采用上面的处理方法后，剩余芯片个数为奇数，那么说明好芯片至少比坏芯片多1个。因为如果剩余芯片个数为奇数，则好芯片要么比坏芯片多，要么比坏芯片少，好芯片不可能与坏芯片个数相等。而如果好芯片比坏芯片少，即使落单芯片是好芯片，那么原来的*n*个芯片中好芯片也不可能多于坏芯片，这与超过*n*/2的芯片是好的假设是矛盾的，所以不可能出现这种情况。这时落单芯片可以直接排除。

② 如果采用上面的处理方法后，剩余芯片个数为偶数，那么说明好芯片不少于坏芯片。最坏情况下，好芯片与坏芯片一样多，这种结果一定发生在落单芯片为好的情况下，这时将落单芯片保留下来，那么最后剩下的芯片中，好芯片仍然比坏芯片多。如果不是最坏情况，那么好芯片至少比坏芯片多2个，这时无论落单芯片是好是坏，将其保留下来，在最后剩下的芯片中，好芯片仍然比坏芯片多。因此，综上所述，如果剩余芯片个数为偶数，不排除落单芯片。

**c-问题分析**

根据问题b的分析，不断缩小子问题规模，最后剩下一块芯片，一定是好芯片。在算法的每一层递中，算法需要比较组芯片，即比较Θ(*n*)次，所以该算法的时间递归式可以写为，求解得到*T*(*n*) = Θ(*n*)。

**c-伪代码**

假设比较两块芯片的函数为COMPARE\_CHIP(*c*1, *c*2)。

CHECK\_CHIP(*A*, *n*)

{

**if** *n* < 1

return “error: no chip”

**else if** *n* == 1

return “chip *A*[1] is good”

**else**

create array *B* to save reserved chips

*reserved\_num* = 0

**for** *i* = 1 🡪 

**if** COMPARE\_CHIP(*A*[2*i*], *A*[2*i*+1]) == BOTH\_GOOD

add *A*[2*i*] to *B*

*reserved\_num*++

**if** *n*%2 != 0 && *reserved\_num*%2 == 0

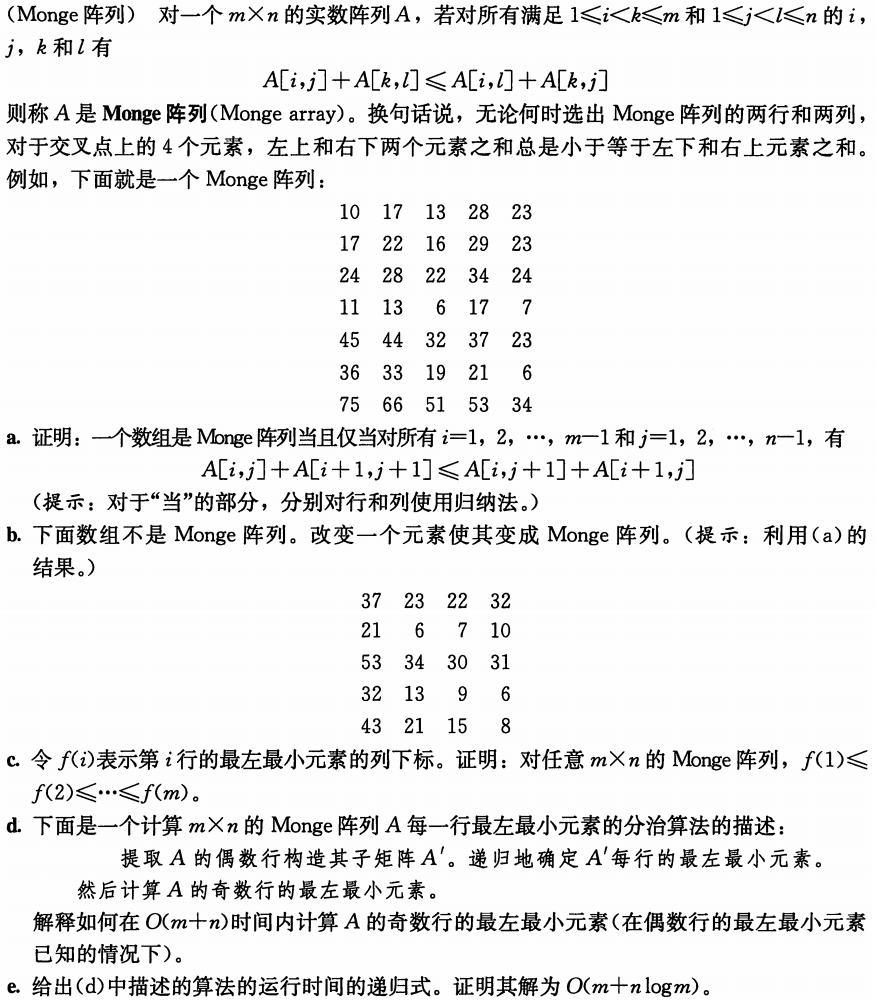
add *A*[*n*] to *B*

*reserved\_num*++

**return** CHECK\_CHIP(*B*, *reserved\_num*)

}

#### 思考题 4-6



**a-问题分析**

(1) Monge阵列 🡺 *A*[*i*, *j*] + *A*[*i* + 1, *j* + 1] ≤ *A*[*i*, *j* + 1] + *A*[*i* + 1, *j*]

根据Monge阵列的定义，对所有满足1 ≤ *i* < *k* ≤ *m*和1 ≤ *j* < *l* ≤ *n*的*i*, *j*, *k*, *l*，有*A*[*i*, *j*] + *A*[*k*, *l*] ≤ *A*[*i*, *l*] + *A*[*k*, *j*]。

令*k* = *i*+1并且*l* = *j*+1，显然满足1 ≤ *i* < *i*+1 ≤ *m*和1 ≤ *j* < *j*+1 ≤ *n*，故*A*[*i*, *j*] + *A*[*i* + 1, *j* + 1] ≤ *A*[*i*, *j* + 1] + *A*[*i* + 1, *j*]成立。

(2) *A*[*i*, *j*] + *A*[*i* + 1, *j* + 1] ≤ *A*[*i*, *j* + 1] + *A*[*i* + 1, *j*] 🡺 Monge阵列

采用数学归纳法来证明。

1) 先固定*k* = *i*+1, 对*l*进行归纳

初始情况为*l* = *j*+1，由于*k* = *i*+1，显然有*A*[*i*, *j*] + *A*[*k*, *l*] ≤ *A*[*i*, *l*] + *A*[*k*, *j*]。

假设*A*[*i*, *j*] + *A*[*k*, *l*] ≤ *A*[*i*, *l*] + *A*[*k*, *j*]对*l*取*j*+1, *j*+2, … , *n*-1都成立，有

*A*[*i*, *j*] + *A*[*k*, *n*-1] ≤ *A*[*i*, *n*-1] + *A*[*k*, *j*] ①

考虑*l* = *n*的情况。取*j* = *n*-1，有

*A*[*i*, *n*-1] + *A*[*k*, *n*] ≤ *A*[*i*, *n*] + *A*[*k*, *n*-1] ②

联立不等式①和②，得到

*A*[*i*, *j*] + *A*[*k*, *n*-1] + *A*[*i*, *n*-1] + *A*[*k*, *n*] ≤ *A*[*i*, *n*-1] + *A*[*k*, *j*] + *A*[*i*, *n*] + *A*[*k*, *n*-1]

消去不等式两边相同的项，得到

*A*[*i*, *j*] + *A*[*k*, *n*] ≤ *A*[*i*, *n*] + *A*[*k*, *j*]

根据以上的归纳，可以得到，有所有满足1 ≤ *i* < *m*和1 ≤ *j* < *l* ≤ *n*的*i*, *j*和*l*，有

*A*[*i*, *j*] + *A*[*i*+1, *l*] ≤ *A*[*i*, *l*] + *A*[*i*+1, *j*]

2) 对k进行归纳

可用类似的归纳法证明，对所有满足1 ≤ *i* < *k* ≤ *m*和1 ≤ *j* < *l* ≤ *n*的*i*, *j*, *k*, *l*，有*A*[*i*, *j*] + *A*[*k*, *l*] ≤ *A*[*i*, *l*] + *A*[*k*, *j*]。因此，矩阵为Monge阵列。证明过程略。

**b-问题分析**

37 23 24 32

21 6 7 10

53 34 30 31

32 13 9 6

43 21 15 8

**c-问题分析**

该问题可以转换为对任意1 ≤ *i* < *j* ≤ *m*，证明*f*(*i*) ≤ *f*(*j*)。

采用反证法，假设*f*(*i*) > *f*(*j*)，即第*i*行的最左最小值出现在第*j*行的左最小值的右边。

由于*A*[*i*, *f*(*i*)]为第*i*行的最左最小值，并且*f*(*i*) > *f*(*j*)，所以*A*[*i*, *f*(*i*)] < *A*[*i*, *f*(*j*)]。

由于*A*[*j*, *f*(*j*)]为第*j*行的最左最小值，并且*f*(*i*) > *f*(*j*)，所以*A*[*i*, *f*(*j*)] ≤ *A*[*j*, *f*(*i*)]。

所以*A*[*i*, *f*(*j*)] + *A*[*j*, *f*(*i*)] > *A*[*i*, *f*(*i*)] + *A*[*i*, *f*(*j*)]。这与Monge阵列的定义矛盾。所以*f*(*i*) > *f*(*j*)的假设不成立，故*f*(*i*) ≤ *f*(*j*)。

因此，*f*(1) ≤ *f*(*2*) ≤ … ≤ *f*(*m*)。

**d-问题分析**

如果偶数行的最左最小元素已知，根据问题*c*的结论，寻找一个奇数行的最左最小元素只需查找跟它相邻的两个偶数行的最左最小元素之间的元素。因此，每个奇数行只需查找一段区间内的元素，这些区间的元素个数总和等于矩阵的列数*n*，由于区间在偶数行的最左最小元素位置上重合，所以还要加上所有偶数行的行数*m*/2。故查找的运行时间为O(*m*+*n*)。

**e-问题分析**

该算法运行时间的递归式为*T*(*m*) = *T*(*m*/2) + O(*m*+*n*)。采用递归树方法求得，得到*T*(*m*) = O(*m*+*nlgm*)。