第五章 分治法

湖南大学信息科学与工程学院

提纲

4.1分治法的设计思想

4.2最大子段和

4.3求解递归式

4.4组合问题中的分治法

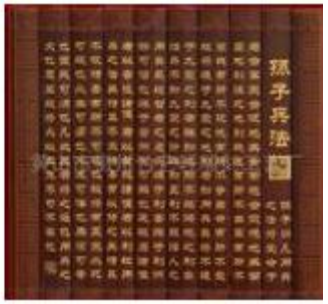
4.5几何问题中的分治法

4.1 分治法的设计思想

凡治众如治寡，分数是也；

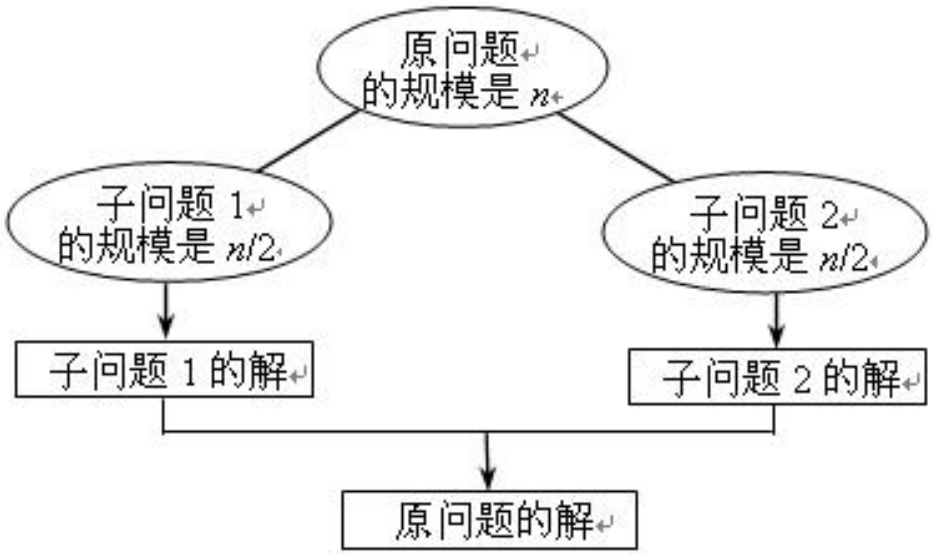
斗众如斗寡，形名是也。

Divide and conquer



4.1分治法的设计思想

将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。



4.1分治法的设计思想

分治法的求解过程是由以下三个阶段组成：

（1）Divide：划分

（2）Conquer：求解子问题

（3）Combine :合并

提纲

4.1分治法的设计思想

4.2最大子段和

4.3求解递归式

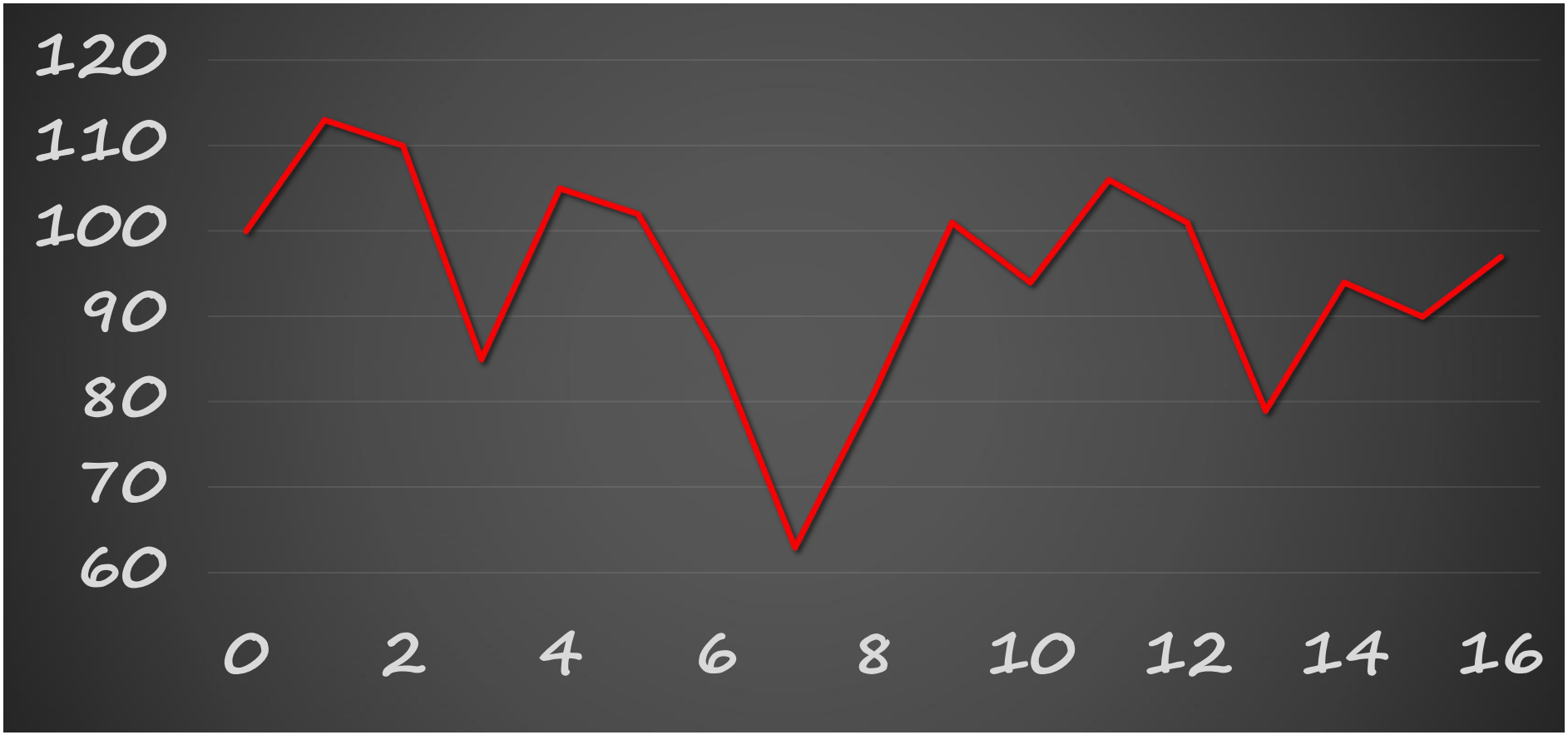
4.4组合问题中的分治法

4.5几何问题中的分治法

4.4组合问题中的分治法-最大子段和问题

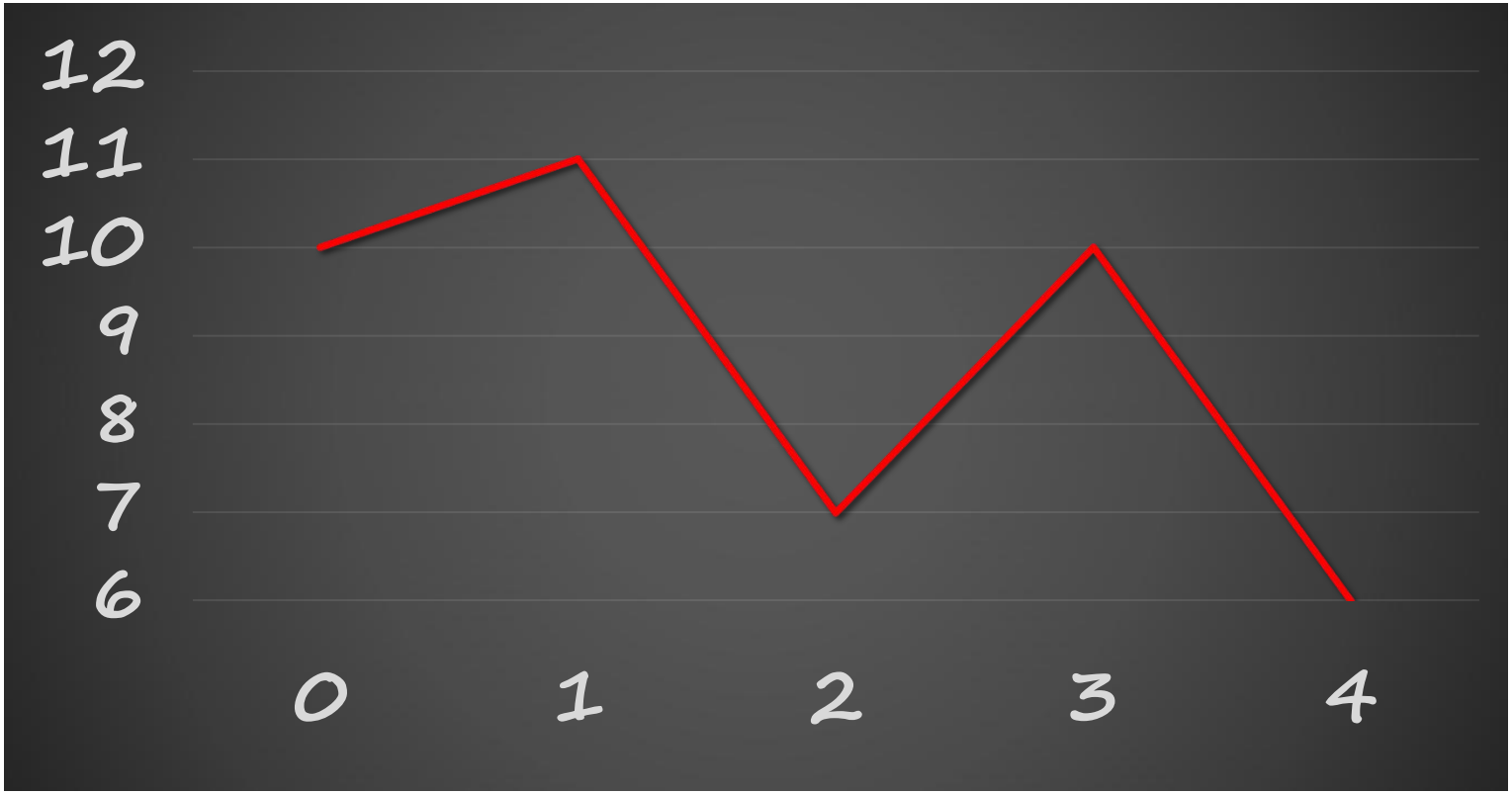
如何购买股票？

4.4组合问题中的分治法-最大子段和问题



天	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
价格	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
变化		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

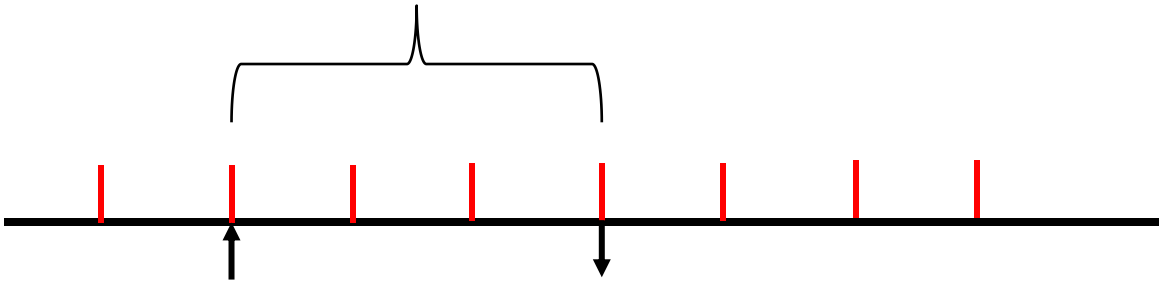
4.4组合问题中的分治法-最大子段和问题



天	0	1	2	3	4
价格变化	10	11	7	10	6
		1	-4	3	-4

4.4组合问题中的分治法-最大子段和问题

最简单方法：暴力法解决



代码及时间复杂度分析

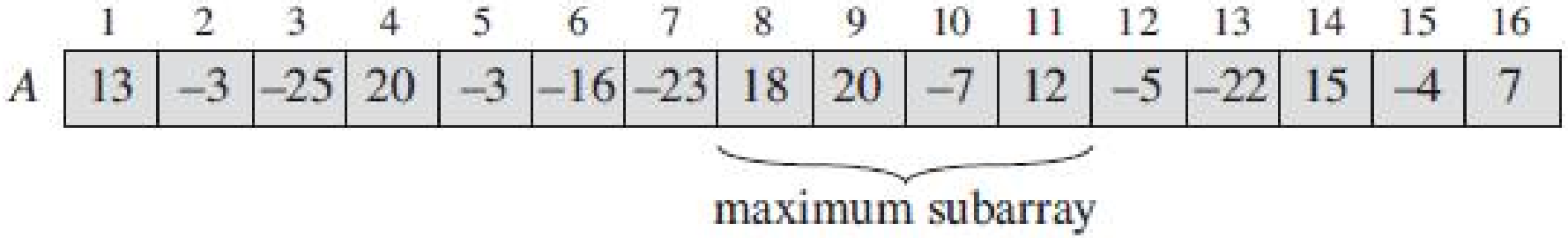
4.4组合问题中的分治法-最大子段和问题

问题变换(最大字段和问题或最大子数组问题)

给定由n个整数（可能有负整数）组成的序列$(a_1, a_2, \ldots, a_n)$，最大子段和问题要求该序列形如$\sum_{k=i}^{j} a_k$的最大值（$1 \leqslant i \leqslant j \leqslant n$），当序列中所有整数均为负整数时，其最大子段和为0。例如，序列(−20, 11, −4, 13, −5, −2)的最大子段和为$\sum_{k=2}^{4} a_k = 20$。

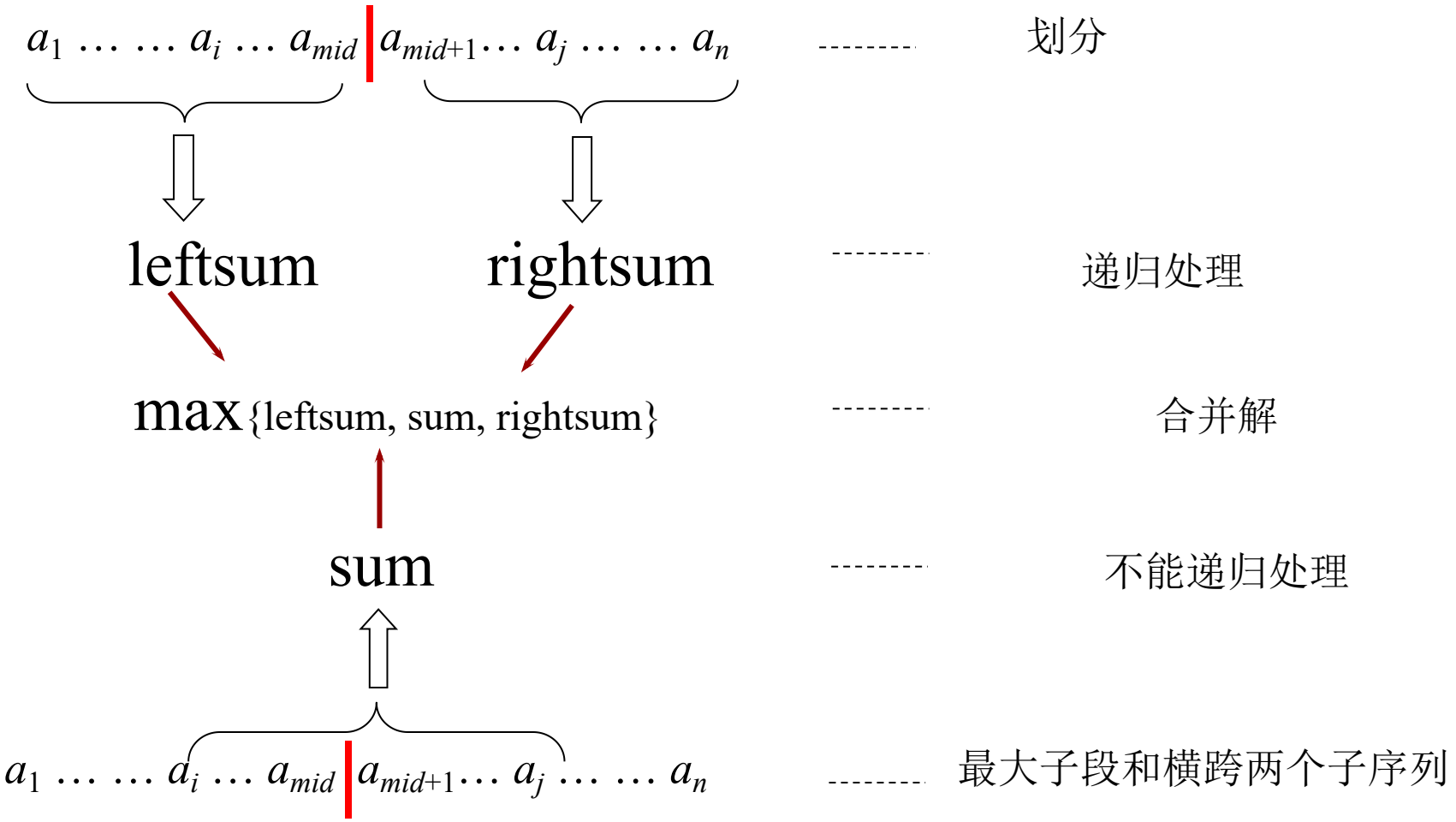
天	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
价格	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
变化		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

4.4组合问题中的分治法-最大子段和问题

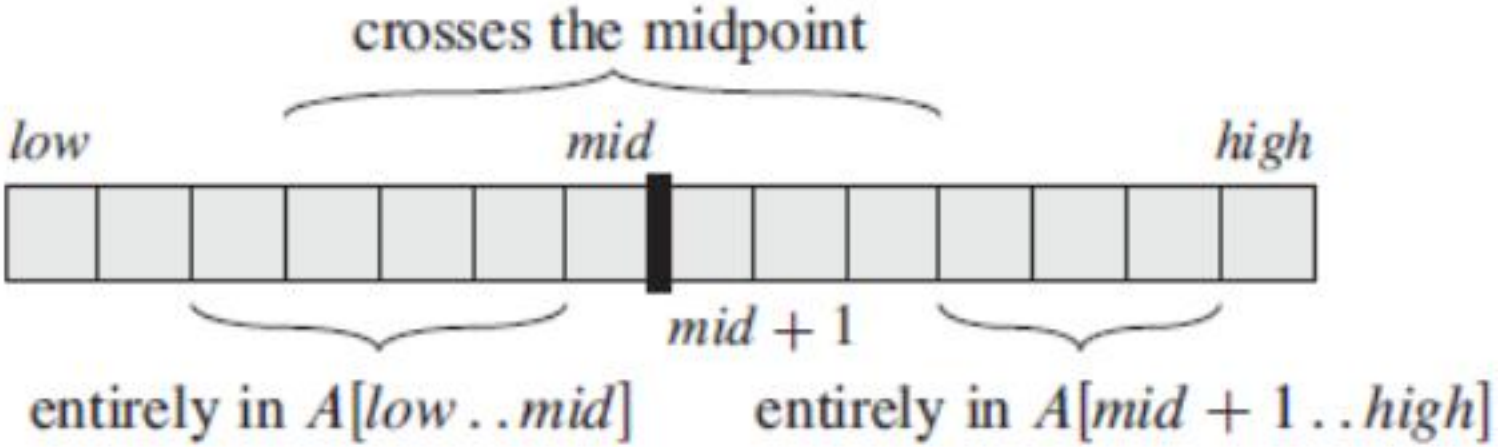


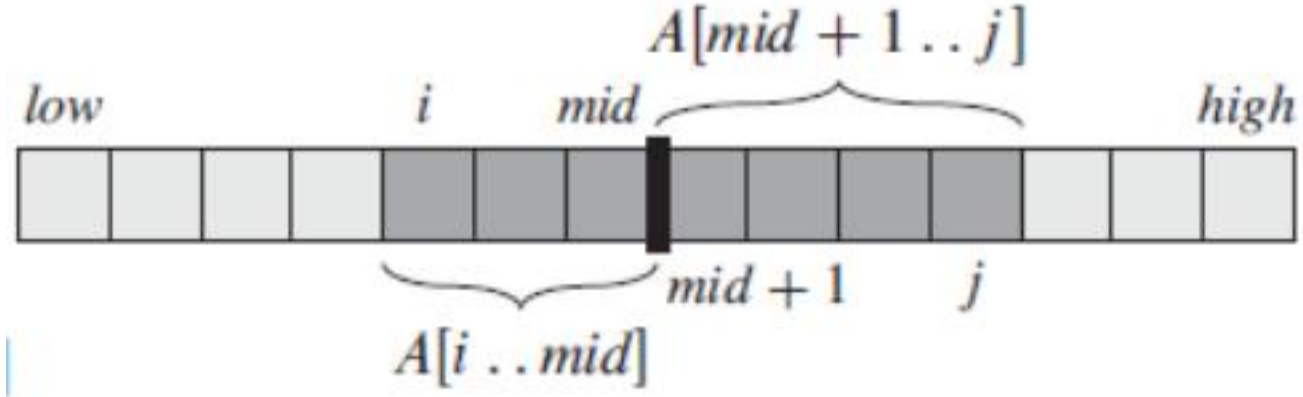
4.4组合问题中的分治法-最大子段和问题

使用分治策略的求解方法



使用分治策略的求解方法





4.4组合问题中的分治法-最大子段和问题

伪代码

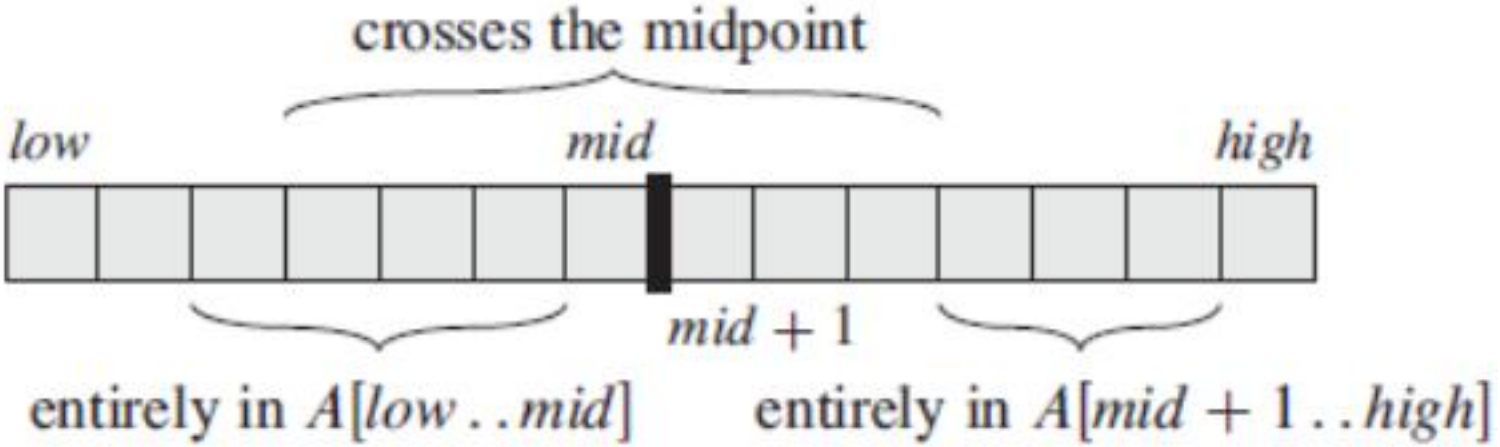
```
FIND-MAX-CROSSING-SUBARRAY (A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

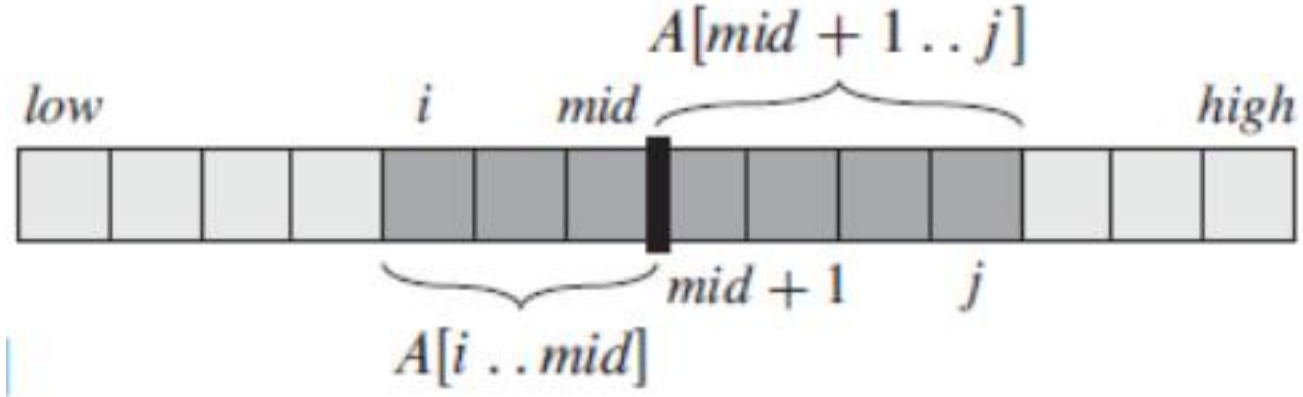
4.4组合问题中的分治法-最大子段和问题

伪代码

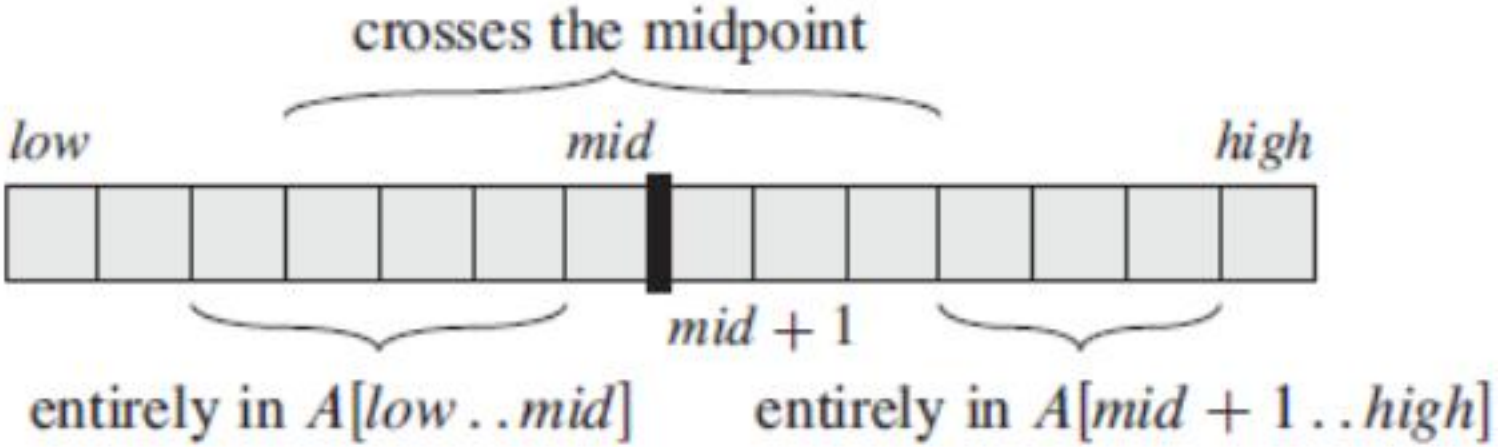
```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2      return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4      (left-low, left-high, left-sum) =
            FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5      (right-low, right-high, right-sum) =
            FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6      (cross-low, cross-high, cross-sum) =
            FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8          return (left-low, left-high, left-sum)
 9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10          return (right-low, right-high, right-sum)
11      else return (cross-low, cross-high, cross-sum)
```

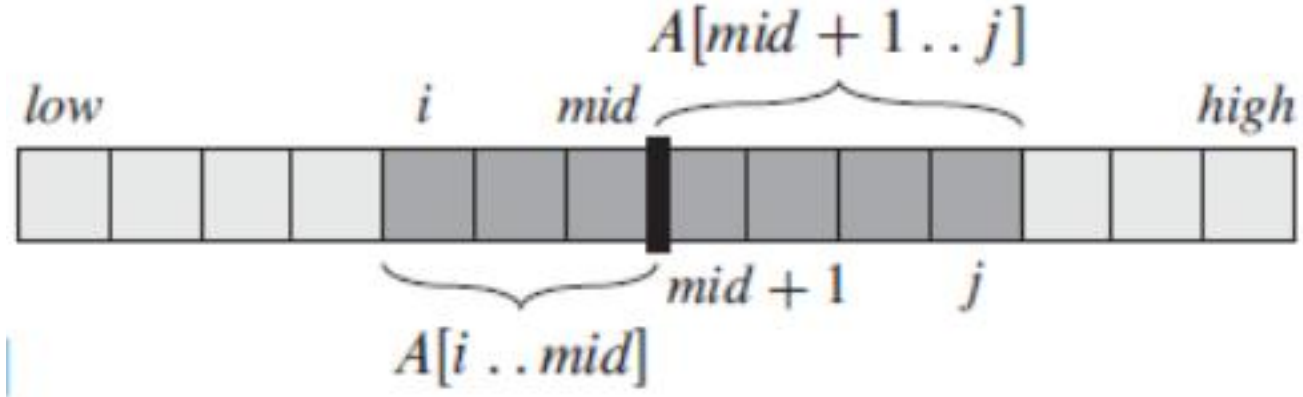
分治法时间复杂度分析


crosses the midpoint
low
mid
high
mid + 1
entirely in A[low . . mid]
entirely in A[mid + 1 . . high]


A[mid + 1 . . j]
low
i
mid
high
mid + 1
j
A[i . . mid]

分治法时间复杂度分析


crosses the midpoint
low
mid
high
mid + 1
entirely in A[low . . mid]
entirely in A[mid + 1 . . high]


A[mid + 1 . . j]
low
i
mid
high
mid + 1
j
A[i . . mid]

4.2最大子段和

时间复杂度：$T(n)=2T(n/2)+\Theta(n)=\Theta(nlogn)$

提纲

4.1分治法的设计思想

4.2最大子段和

4.3求解递归式

4.4组合问题中的分治法

4.5几何问题中的分治法

4.3 求解递归式

三种求解递归式的方法：

1、代入法
2、递归树法
3、主方法

代入法

求解：$T(n)=4T(n/2)+n$

We first guess $T(n)=O(n^3)$

(Proof by induction)Assume $T(k)<=ck^3$, for $k<n$

Then $T(n)=4T(n/2)+n<=4c(n/2)^3+n=cn^3/2+n=cn^3-(cn^3/2-n)$

$cn^3-(cn^3/2-n)<=cn^3$, if $(cn^3/2-n)>=0$, e.g. $c>1$ and $n>=1$

代入法

求解：$T(n)=4T(n/2)+n$

Then, we prove $T(n)=O(n^2)$.

(Proof by induction)Assume $T(k)<=ck^2$, for $k<n$

Then $T(n)<=4c(n/2)^2+n=cn^2+n$.

代入法

求解：$T(n)=4T(n/2)+n$

Assume $T(k)<=c_1k^2-c_2k$, for $k<n$

$T(n)<=4(c_1(n/2)^2-c_2n/2)+n<= c_1n^2-2\ c_2n+n= c_1n^2-c_2n-(c_2-1)n$

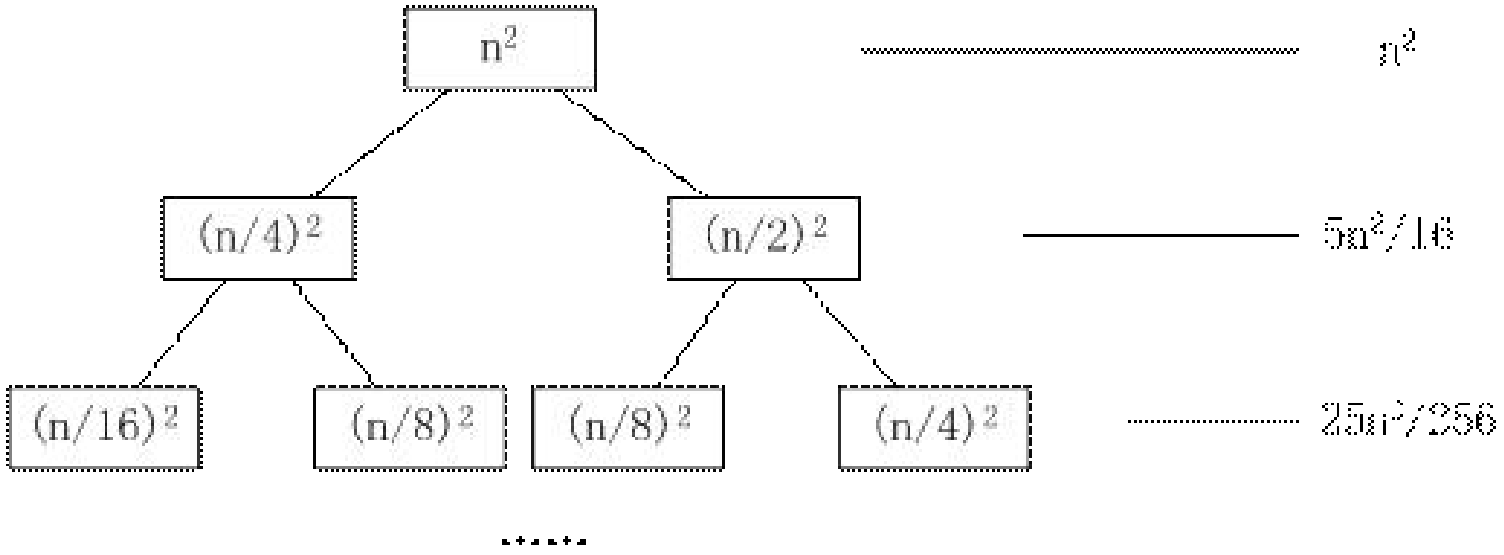
If $c_2>=1$, $(c_2-1)n>=0$. Then $T(n)<= c_1n^2-c_2n$

4.3 求解递归式

$$T(n) = T(n/4) + T(n/2) + n^2:$$

递归树法

Example: $T(n)=T(n/4)+T(n/2)+n^2$



$T(n)<=n^2(1+5/16+(5/16)^2+...+(5/16)^k+.....)<2n^2=O(n^2)$

4.3 求解递归式-递归树法

（1）归并排序关键代码分析

（2）递归调用过程分析（图例）

（3）时间复杂度分析？

4.3 求解递归式-主方法

主方法适用于下面的递归形式

$$T(n) = aT(n/b) + f(n)$$

$a \geq 1, b > 1, f$为正的渐进函数

4.3 求解递归式-主方法

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

4.3 求解递归式-主方法

1. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = \mathrm{O}(n^{\log_b a - \varepsilon})$， 则 $\mathrm{T(n)} = \Theta(n^{\log_b a})$。

2. 若 $f(n) = \Theta(n^{\log_b a})$， 则 $\mathrm{T(n)} = \Theta(n^{\log_b a} \lg n)$。

3. 若对某个常数 $\varepsilon > 0$， 有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$， 且对某个常数 $c < 1$ 和足够大的 n 有 $af(n/b) \le cf(n)$， 则 $\mathrm{T(n)} = \Theta(f(n))$。

4.3 求解递归式-主方法

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

$$T(n) = 3T\left(\frac{n}{4}\right) + nlogn$$

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

4.3 求解递归式-主定理

自学，算法导论：P55-60

提纲

4.1分治法的设计思想

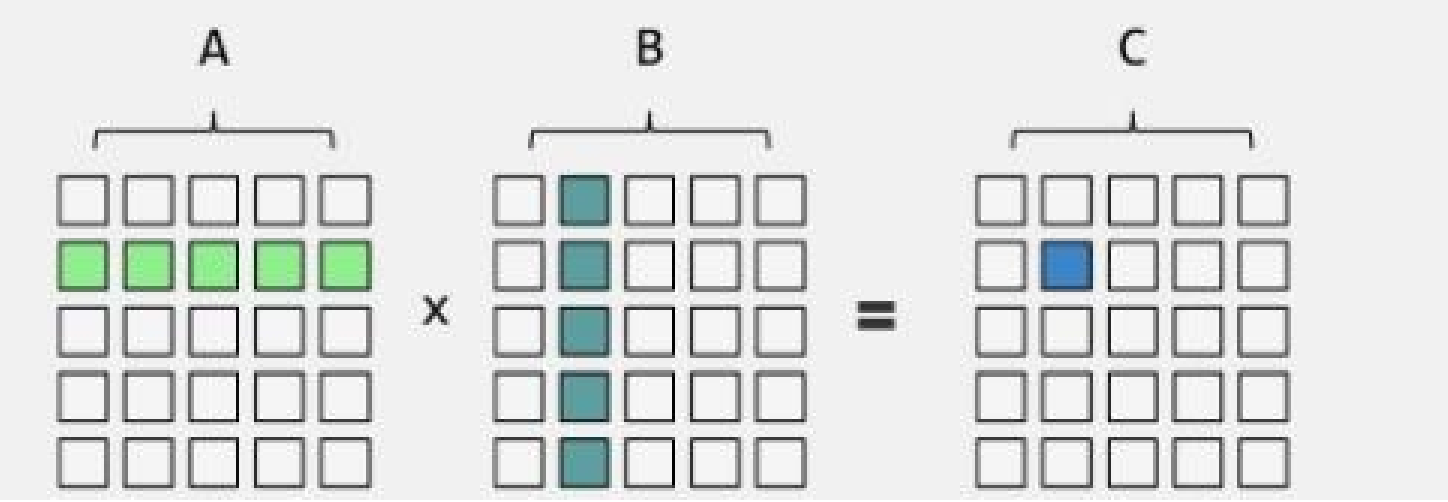
4.2最大子段和

4.3求解递归式

4.4组合问题中的分治法

4.5几何问题中的分治法

4.4组合问题中的分治法-矩阵乘法



矩阵加法运算的时间复杂度是：$\Theta(n^2)$

而矩阵乘法的时间复杂度是：$\Theta(n^3)$

求每个元素C(i, j)都需要n次乘法和n-1次加法运算，且C共有n^2个元素。

4.4组合问题中的分治法-矩阵乘法

用分治法解决矩阵的乘法问题，先假定n是2的k次幂，即$n=2^k$。

1）第一步：递归维度半分。

如果n=2，则递归结束。

如果n>2，将A 和B 两个n 维方阵各分解为4份，每份n/2 维。当子矩阵的阶还是大于2时，继续将子矩阵分块，直到子矩阵的阶降为2。

第一步将会产出很多个2阶的方阵！

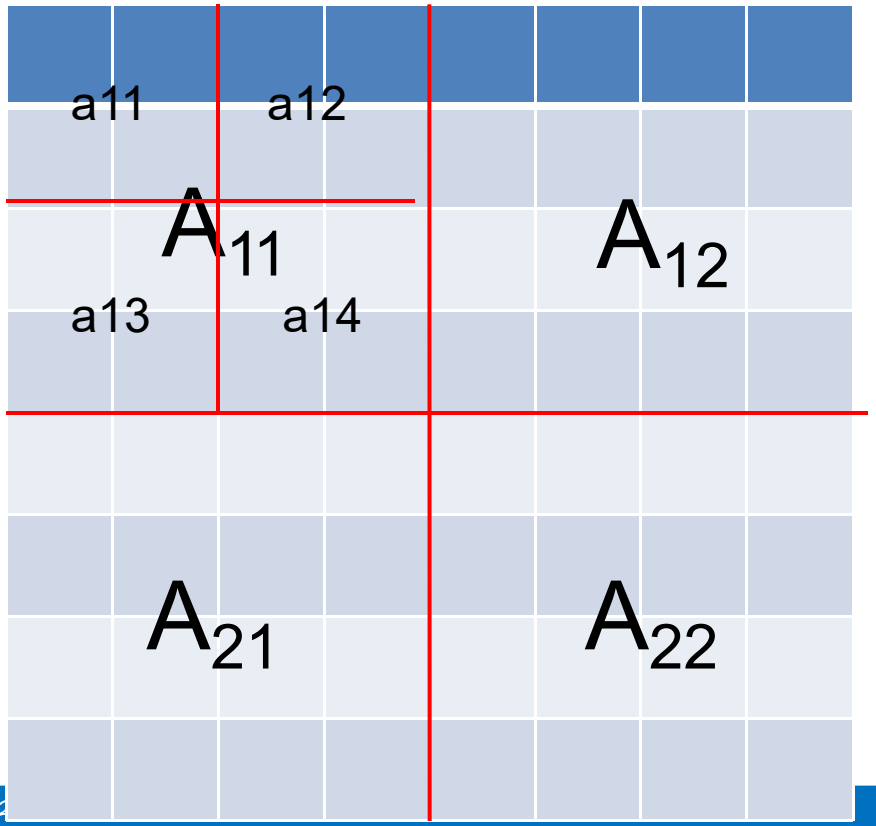
4.4组合问题中的分治法-矩阵乘法

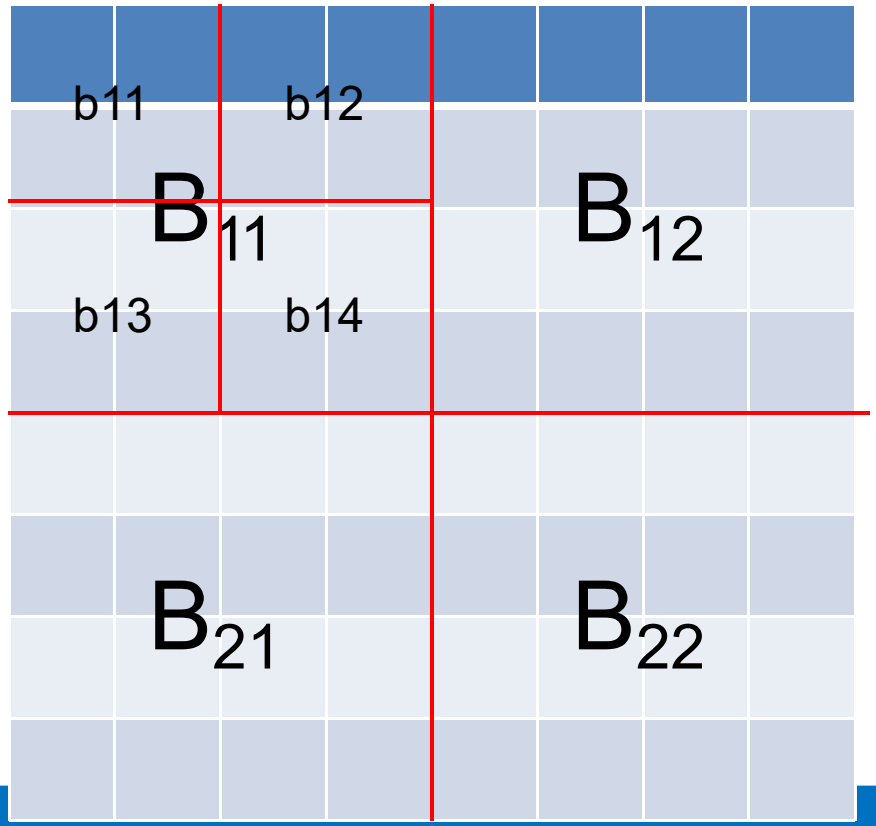
$$C_{11}=A_{11}*B_{11}+A_{12}*B_{21}$$

下面用一个动画来展示分矩阵的过程

$$A_{11}*B_{11}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}=\begin{bmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{bmatrix}$$

A矩阵　　B矩阵





4.4组合问题中的分治法-矩阵乘法

2）第二步：计算两个2阶方阵的乘积之后再合并。

假如A，B两个矩阵是由第一步得出的维数为2的方阵，则：

$$A*B=\begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix}\begin{bmatrix}B_{11} & B_{12}\\ B_{21} & B_{22}\end{bmatrix}=\begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}$$

$$C_{11}=A_{11}B_{11}+A_{12}B_{21} \qquad C_{12}=A_{11}B_{12}+A_{12}B_{22}$$

$$C_{21}=A_{21}B_{11}+A_{22}B_{21} \qquad C_{22}=A_{21}B_{12}+A_{22}B_{22}$$

通过上面4条式可以将A*B的值存到C矩阵中。

分治法求矩阵乘法的效率分析

如果用T(n)记两个n阶矩阵相乘所用的时间，则有如下递归关系式：

$$T(n)=\begin{cases} b & n \le 2 \\ 8T(n/2)+dn^2 & n>2 \end{cases}$$

$T(n)=\Theta(n^3)$

4.4组合问题中的分治法-矩阵乘法

Strassen矩阵乘法

Strassen提出了一种新的算法来计算2个2阶方阵的乘积。他的算法只用了7次乘法运算，但增加了加、减法的运算次数。这7次乘法是：

$M_1=A_{11}(B_{12}-B_{22})$

$M_2=(A_{11}+A_{12})B_{22}$

$M_3=(A_{21}+A_{22})B_{11}$

$M_4=A_{22}(B_{21}-B_{11})$

$M_5=(A_{11}+A_{22})(B_{11}+B_{22})$

$M_6=(A_{12}-A_{22})(B_{21}+B_{22})$

$M_7=(A_{11}-A_{21})(B_{11}+B_{12})$

4.4组合问题中的分治法-矩阵乘法

Strassen矩阵乘法

做了7次乘法后，再做若干次加、减法：

$C_{11}=M_5+M_4-M_2+M_6$

$C_{12}=M_1+M_2$

$C_{21}=M_3+M_4$

$C_{22}=M_5+M_1-M_3-M_7$

Strassen算法效率分析

Strassen矩阵乘积分治算法中，用了7次对于n/2阶矩阵乘积的递归调用和18次n/2阶矩阵的加减运算。由此可知，该算法的所需的计算时间T(n)满足如下的递归方程:

$$T(n)=\begin{cases} b & n \le 2 \\ 7T(n/2)+an^2 & n>2 \end{cases}$$

Strassen算法效率分析

Strassen矩阵乘积分治算法中，用了7次对于n/2阶矩阵乘积的递归调用和18次n/2阶矩阵的加减运算。由此可知，该算法的所需的计算时间T(n)满足如下的递归方程：

$$T(n)=\begin{cases} b & n \le 2 \\ 7T(n/2)+an^2 & n>2 \end{cases}$$

$T(n)=O(n^{\log_2 7})$

提纲

4.1分治法的设计思想

4.2最大子段和

4.3求解递归式

4.4组合问题中的分治法

4.5几何问题中的分治法

题

设$p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$, …, $p_n=(x_1,y_1)$，是平面上n个点构成的集合S，最近对问题就是找出集合S中距离最近的点对。

你想到些什么？

题

最近对问题的分治策略是：

(1)划分：将集合S分成两个子集S_1和S_2，设集合S的最近点对是p_i和p_j（$1 \le i, j \le n$），则会出现以下三种情况：

①p_i和p_j在S_1

②p_i和p_j在S_2

③p_i和p_j分别在S_1和S_2

(2)求解子问题：对于划分阶段的情况①和②可递归求解，如果最近点对分别在集合S_1和S_2中，问题就比较复杂了。

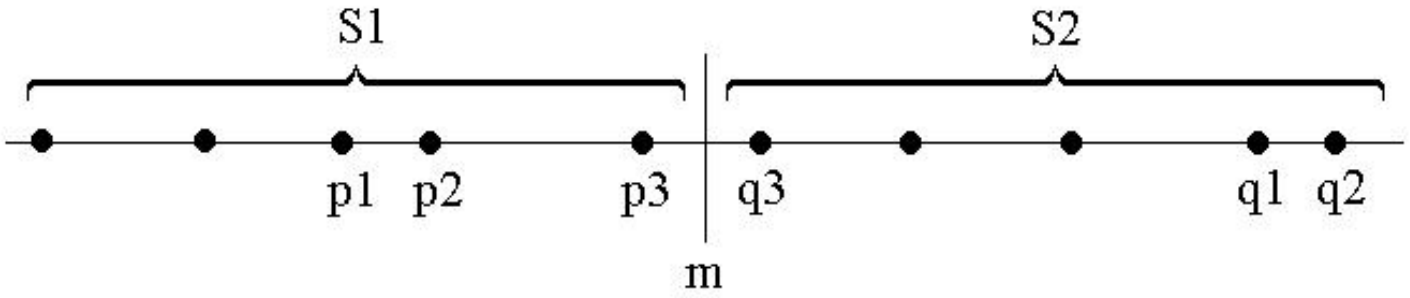
(3)合并：比较在划分阶段三种情况下最近点对，取三者之中较小者为原问题的解。

题

- 一维空间
- 二维空间（平面)

4.5几何问题中的分治法-最近点对问题

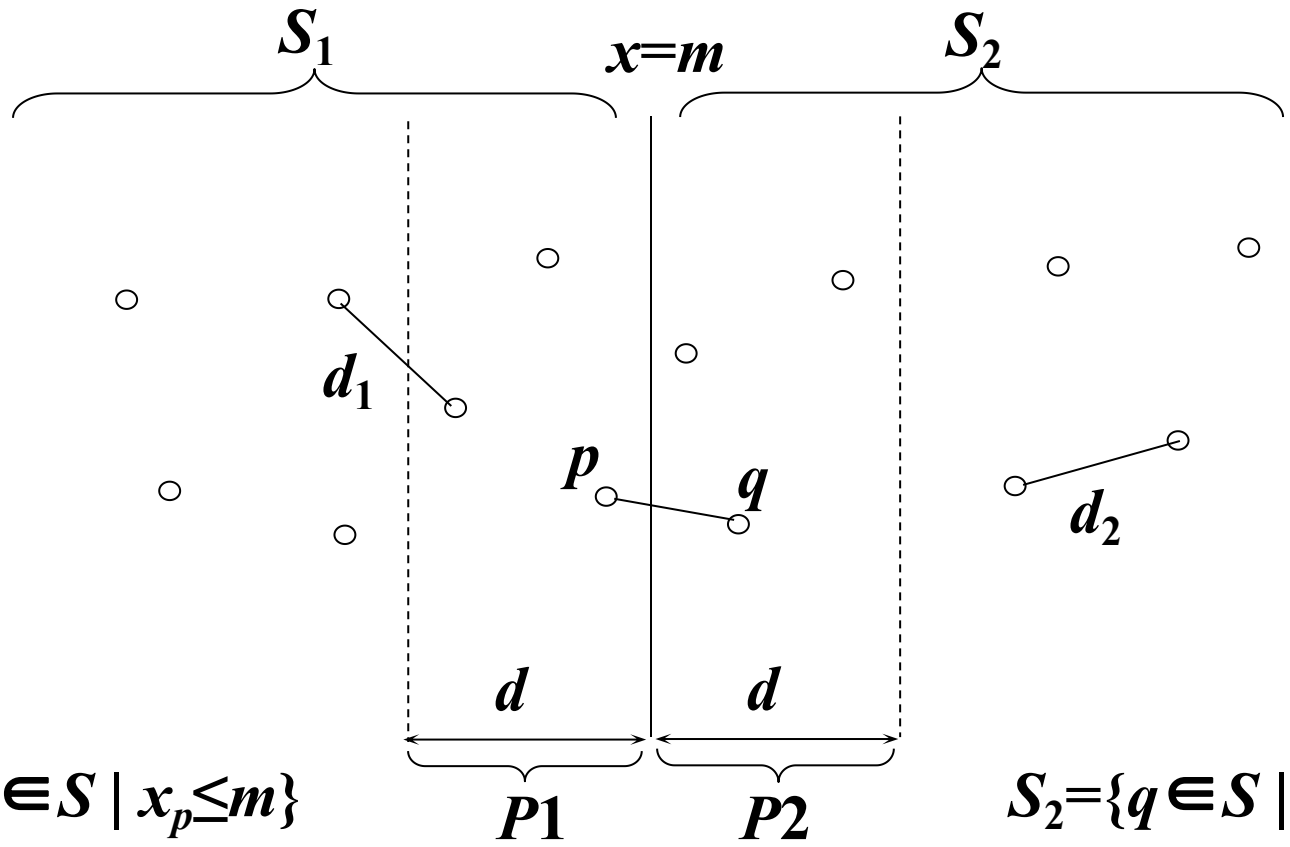
一维空间



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

4.5几何问题中的分治法-最近点对问题

二维空间

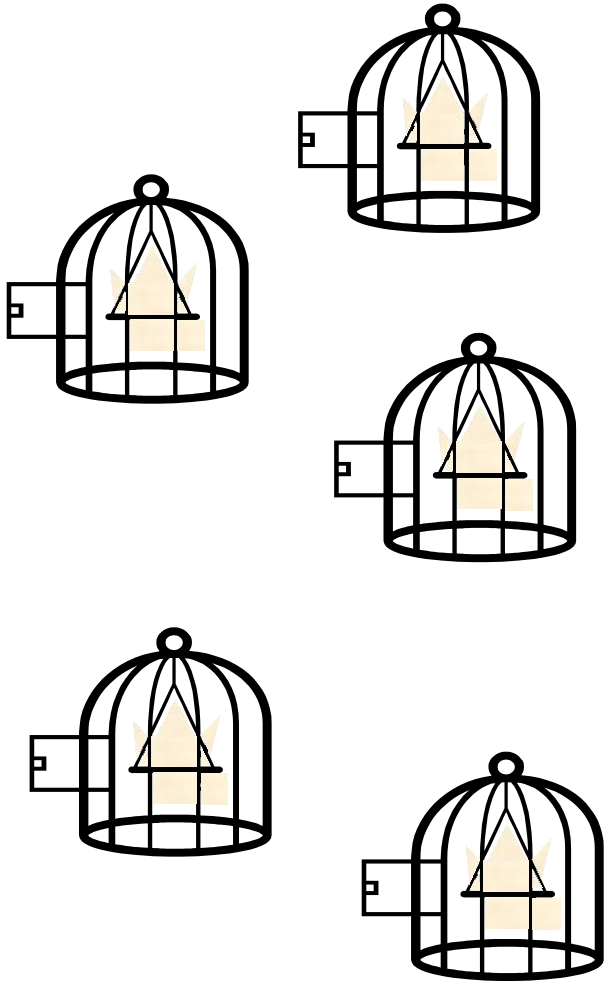


$S_1=\{p \in S \mid x_p \leq m\}$

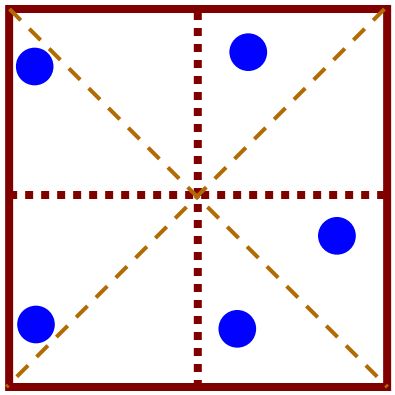
$S_2=\{q \in S \mid x_q > m\}$

4.5几何问题中的分治法-最近点对问题

鸽笼原理



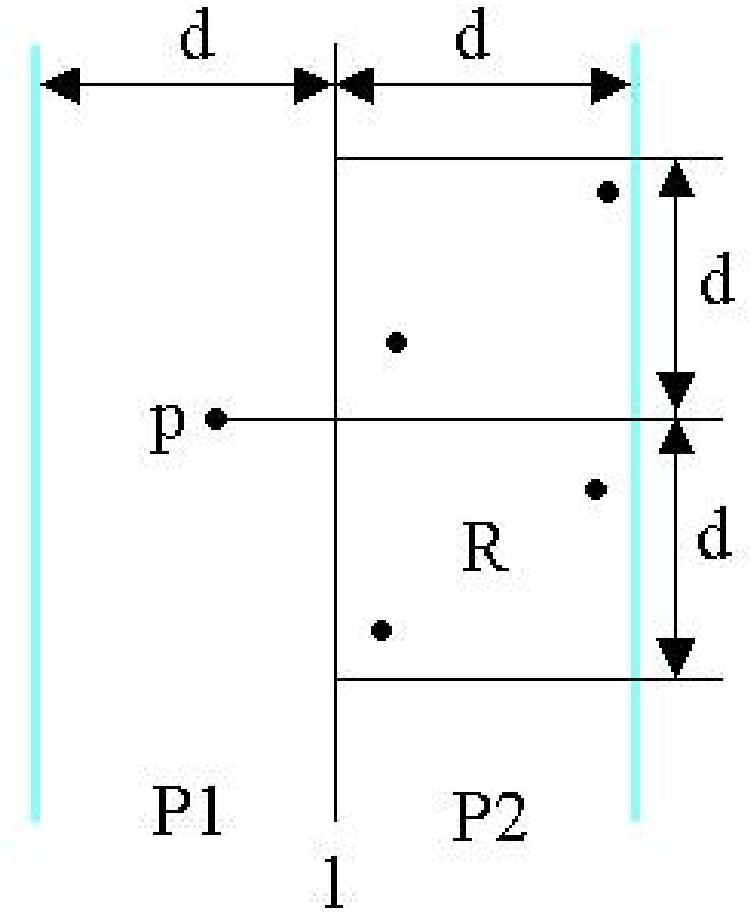
4.5几何问题中的分治法-最近点对问题



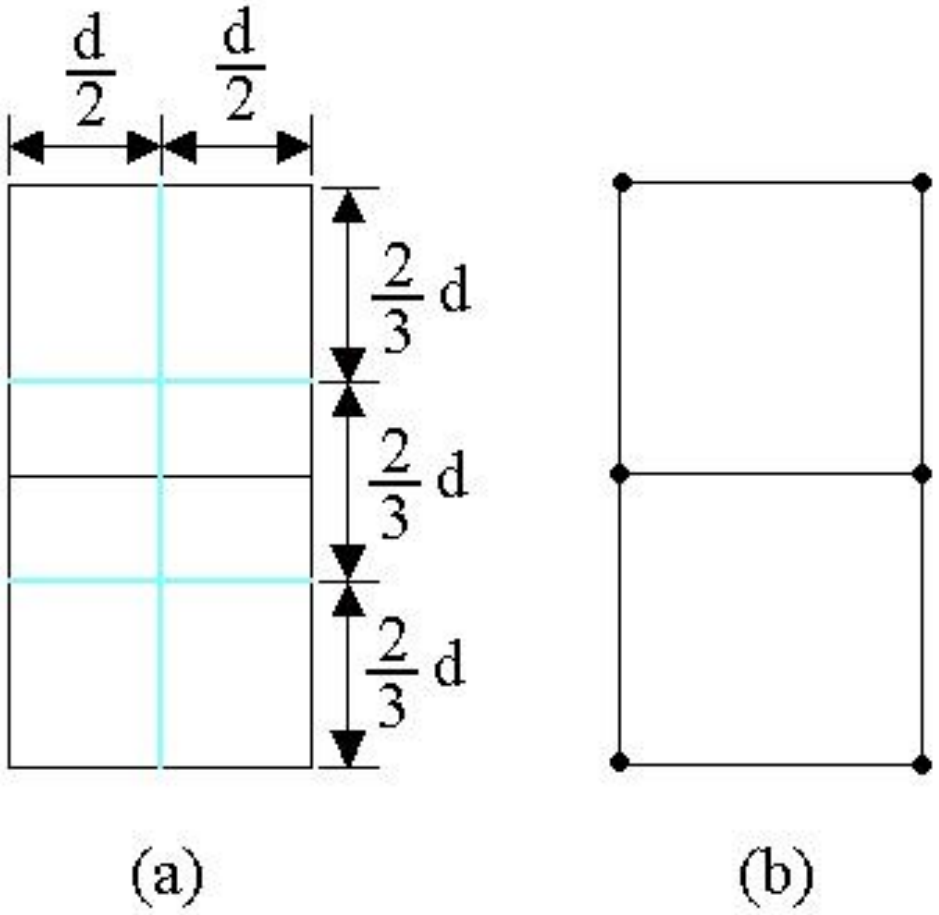
证明：把5个顶点放到边长为2的正方形中，至少存在两个顶点，它们之间的距离小于或等于$\sqrt{2}$。

4.5几何问题中的分治法-最近点对问题

二维空间



二维空间



二维空间（伪代码）

输入：按x坐标排列的n(n>=2)个点的集合S={(x1,y1),(x2,y2),…,(xn,yn)}
输出：最近点的距离
1.如果n==2，则返回（x1,y1）和（x2,y2）之间的距离，算法结束；
2.如果n==3，则返回（x1,y1）、（x2,y2）和（x3,y3）之间的最小距离，算法结束；（此步必要在于，若n==3划分之后必然有一半为n==1，导致无法正确执行递归）
3.划分：m==S中各点x坐标的中位数；
4.d1 = 计算{(x1,y1),…,(xm,ym)}的最近对距离；
5.d2 = 计算{(xm,ym),…,(xn,yn)}的最近对距离；
6.d = min(d1,d2);
7.依次考察集合S中的点p（x，y），如果（x<=xm 并且x>=xm-d），则将点p放入集合P1中；如果（x>xm 并且x<=xm+d），则将点p放入集合P2中;
8.将集合P1和P2按y坐标升序排列；
9.对集合P1和P2中的每个点p（x，y），在y坐标区间[y,y+d]内最对取出6个候选点，计算与点p的最近距离d3；
10.返回min{d，d3}；

4.5几何问题中的分治法-最近点对问题

（1）代码分析

（2）算法时间复杂度分析

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

Finding Peak…
IN SPACE

- Problem: Find a local minimum or maximum in a terrain by sampling

1D Peak Finding

- Given an array $A[0..n-1]$:

- $A[i]$ is a **peak** if it is not smaller than its neighbor(s):

$$A[i-1] \leq A[i] \geq A[i+1]$$

where we imagine

$$A[-1] = A[n] = -\infty$$

- Goal: Find *any* peak

“Brute Force” Algorithm

- Test all elements for peakyness

```
for i in range(n):
    if A[i − 1] ≤ A[i] ≥ A[i + 1]:
        return i
```

O(1) O(n)

Algorithm 1½

- $\max(A)$
  - Global maximum is a local maximum

```
m = 0
for i in range(1, n):
    if A[i] > A[m]:
        m = i
return m
```

$\Theta(1)$ $\Theta(n)$

Clever Idea

Find a subarray containing at least one elements larger than the boundary

- Find the largest element of the first, last and central elements
- If it's a peak, return it; otherwise, find the subarray containing its larger neighbor

- Recurse in the subarray

1D Peak Finding Analysis

- Divide problem into 1 problem of size $\sim \frac{n}{2}$
- Divide cost: $O(1)$
- Combine cost: $O(1)$
- Recurrence:

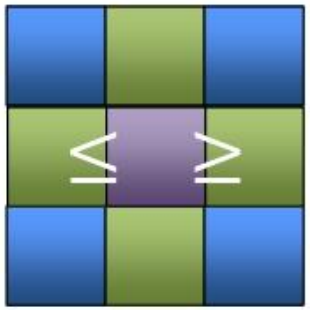
$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

Cleverer Idea

- Look at any element $A[i]$ and its neighbors $A[i-1]$ & $A[i+1]$
  - If peak: return i
  - Otherwise: locally rising on some side
    - Must be a peak in that direction
    - So can throw away rest of array, leaving $A[:i]$ or $A[i+1:]$

2D Peak Finding

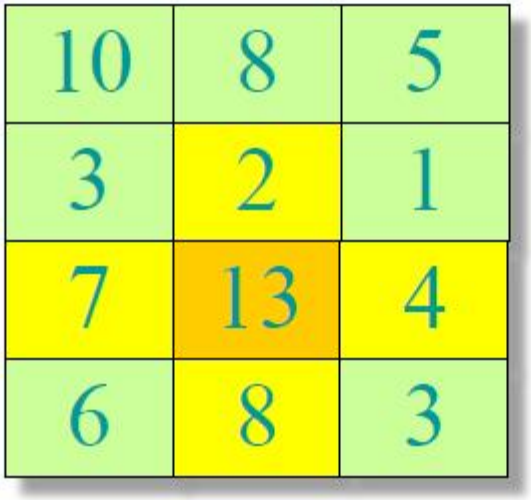
- Given $n \times n$ matrix of numbers
- Want an entry not smaller than its (up to) 4 neighbors:



9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

Peak Finding: 2D

- Consider a 2D array A[1…n, 1…m] :



- An element A[i] is a *2D peak* if it is not smaller than its (at most 4) neighbors.
- Problem: find any 2D peak.

Algorithm I: use the 1D algorithm

- Algorithm I:
  - For each column j, find its *global* maximum B[j]
  - Apply 1D peak finder to find a peak (say B[j]) of B[1...m]
- Running time ?

  …is $\Theta(n \cdot m)$
- Correctness:
  - B[j] not smaller than B[j-1], B[j+1]
  - For any k, B[k] not smaller than any element from the k-th column of A
  - Therefore, B[j] not smaller than any element from the columns j-1, j and j+1 of A
  - But this includes all neighbors of B[j] in A, so B[j] is a peak in A

12	8	5
11	3	6
10	9	2
8	4	1

12	9	6

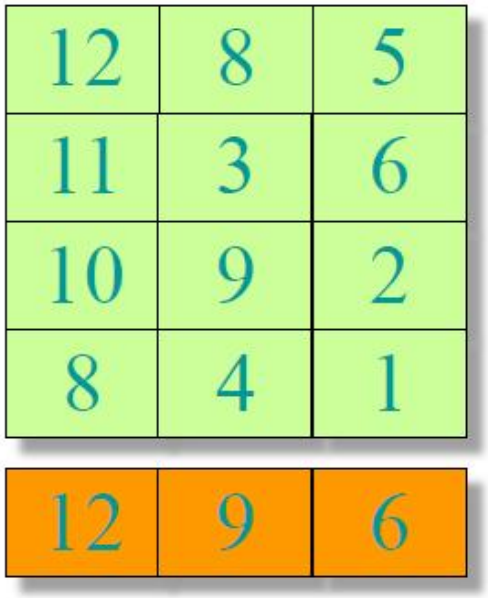
Algorithm I': use the 1D algorithm

- Observation: 1D peak finder uses only O(log m) entries of B
- We can modify Algorithm I so that it only computes B[j] *when needed* !
- Total time ?

  …only O(n log m) !

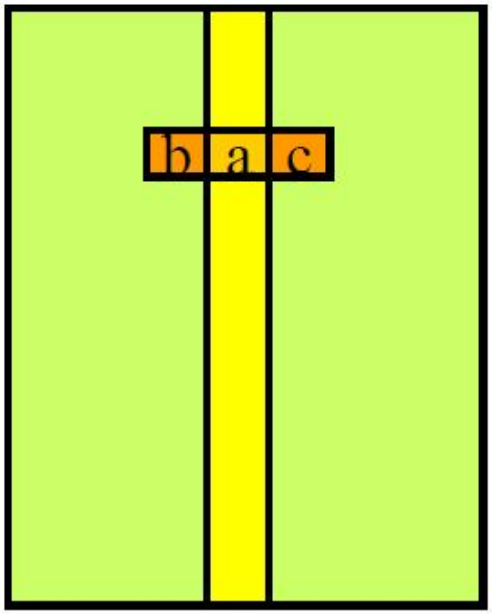
  – Need O(log m) entries B[j]

  – Each computed in O(n) time



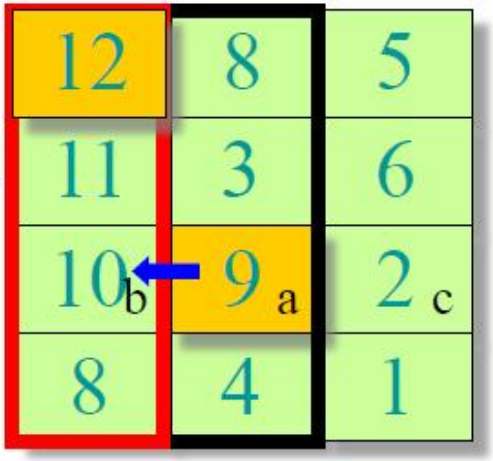
Algorithm II

- Pick middle column (j=m/2)
- Find *global* maximum a=A[i,m/2] in that column (and quit if m=1)
- Compare a to b=A[i,m/2-1] and c=A[i,m/2+1]
- If b>a
  then recurse on left columns
- Else, if c>a
  then recurse on right columns
- Else a is a 2D peak!



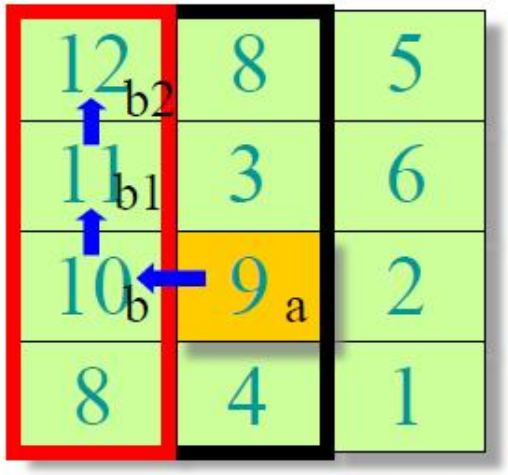
Algorithm II: Example

- Pick middle column (j=m/2)
- Find *global* maximum a=A[i,m/2] in that column (and quit if m=1)
- Compare a to b=A[i,m/2-1] and c=A[i,m/2+1]
- If b>a
  then recurse on left columns
- Else, if c>a
  then recurse on right columns
- Else a is a 2D peak!



Algorithm II: Correctness

- Claim: If b>a, then there is a peak among the left columns
- Proof (by contradiction):
  - Assume no peak on the left
  - Then b must have a neighbor b1 with higher value
  - And b1 must have a neighbor b2 with higher value
  - …
  - We have to stay on the left side – why?
  - (because we cannot enter the middle column)
  - But at some point, we would run out the elements of the left columns
  - Hence, we have to find a peak at some point



Algorithm II: Complexity

- We have

Recursion

$$T(n,m) = T(n,m/2) + \Theta(n)$$

Scanning middle column

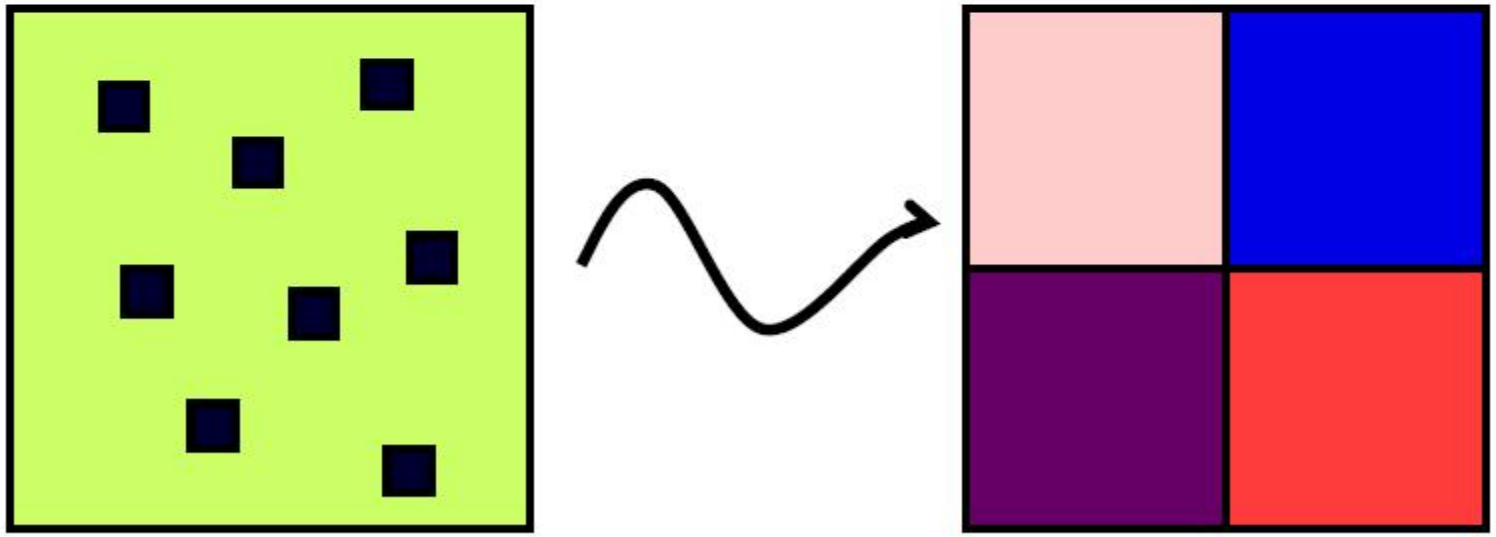
- Hence:
  - $T(n,n) = \underbrace{\Theta(n) + \Theta(n) + \ldots + \Theta(n)}_{\log_2 m} = \mathbf{\Theta(n \log m)}$

Faster than O(n log n) ?

- Idea:

  Reading only O($n + m$) elements, reduce an array of $n \times m$ candidates to an array of $n/2 \times m/2$ candidates

- Pictorially:



read only O($n + m$) elements

Faster than O(n log n) ?

- Hypothetical algorithm has recursion:

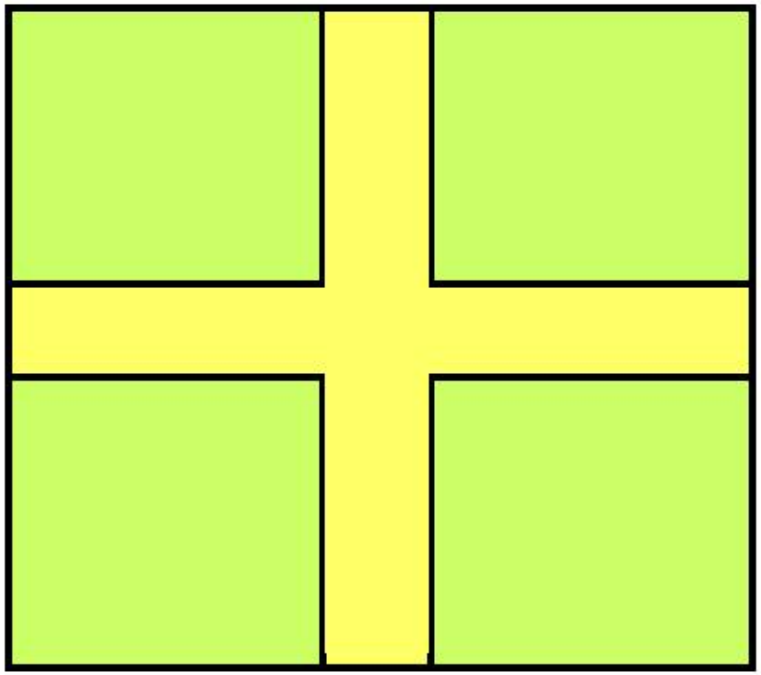
$$T(n,m) = T\left(\frac{n}{2}, \frac{m}{2}\right) + \Theta(n+m)$$

- Hence:

$$\begin{aligned} T(n,m) &= \Theta(n+m) + \Theta\left(\frac{n+m}{2}\right) \\ &\quad + \Theta\left(\frac{n+m}{4}\right) \\ &\quad + \ldots + \Theta(1) \\ &= \Theta(n+m) \quad \text{!} \end{aligned}$$

Towards a linear-time algorithm

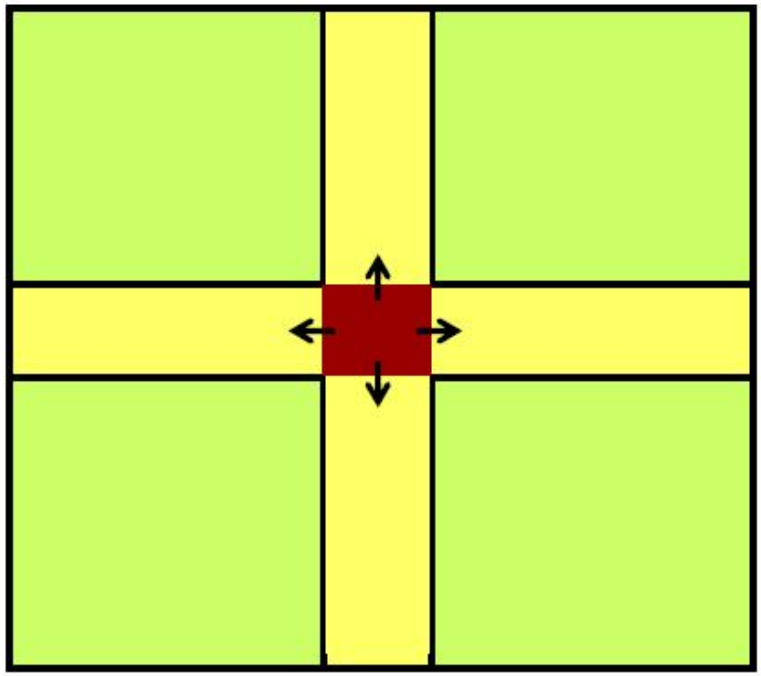
What elements are useful to check?



- suppose we find global max on the cross

Towards a linear-time algorithm

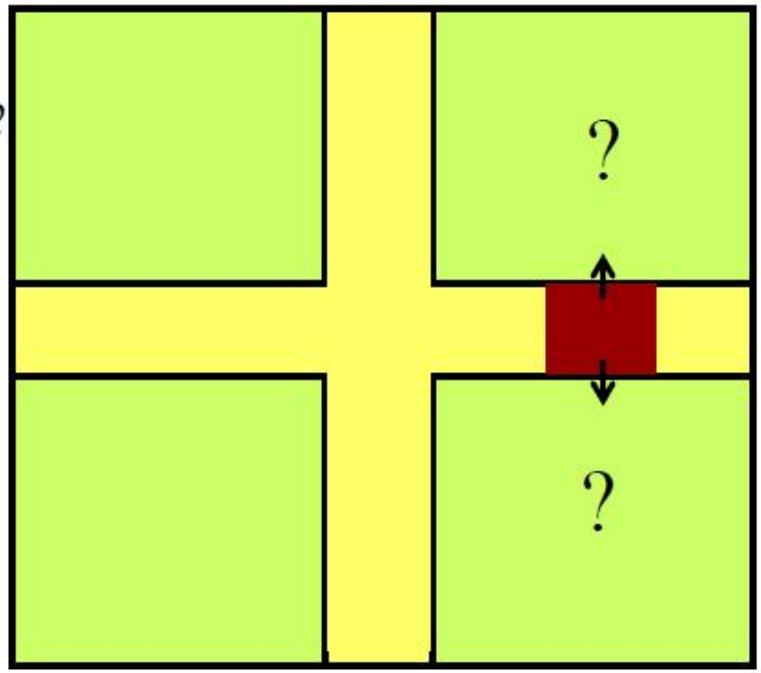
What elements are useful to check?



- suppose we find global max on the cross
- if middle element done!

Towards a linear-time algorithm

What elements are useful to check?



- find global max on the cross
- if middle element done!
- o.w. two candidate sub-squares
- determine which one to pick by looking at its neighbors not on the cross (as in Algorithm II)

proof of claim 2 and fix to this algorithm provided in recitation

Claim: The sub-square chosen by the above procedure (if any), always contains a peak of the large square.

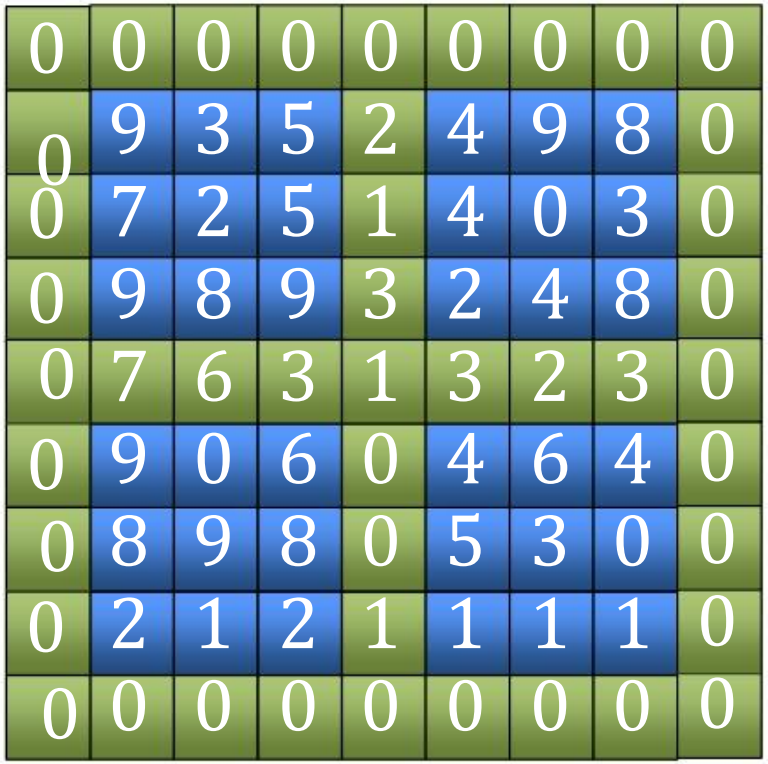
BUT: **Claim 2:** Not every peak of the chosen sub-square is necessarily a peak of the large square. Hence, it is hard to recurse…

Example

5	4	5	6	4	9	8
3	2	3	1	4	0	3
9	3	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

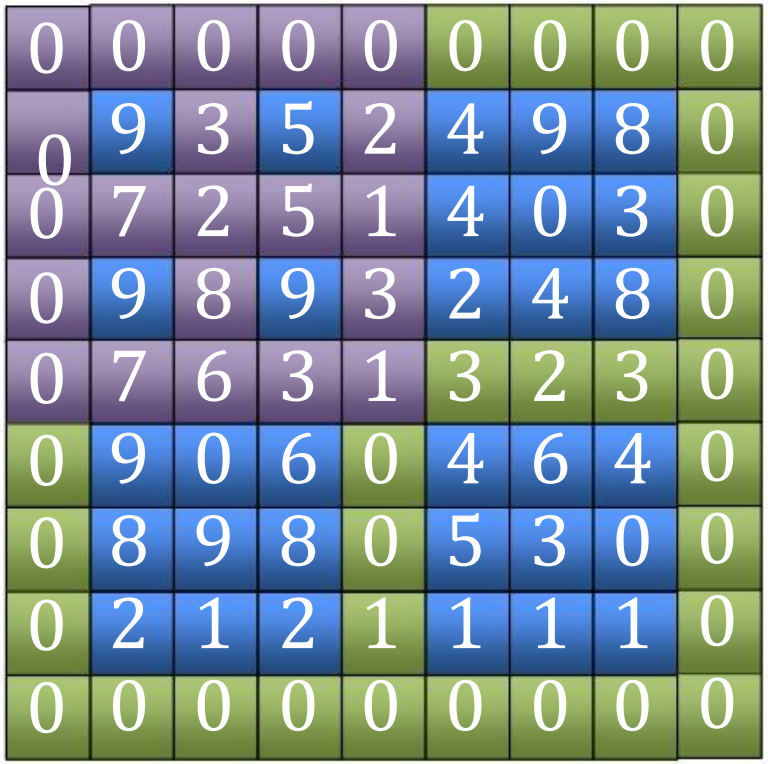
Divide & Conquer #2

- Look at boundary, center row, and center column (**window**)
- Find global max within
- If it's a peak: return it
- Else:
  - Find larger neighbor
  - Can't be in window
  - Recurse in quadrant, including green boundary



Correctness

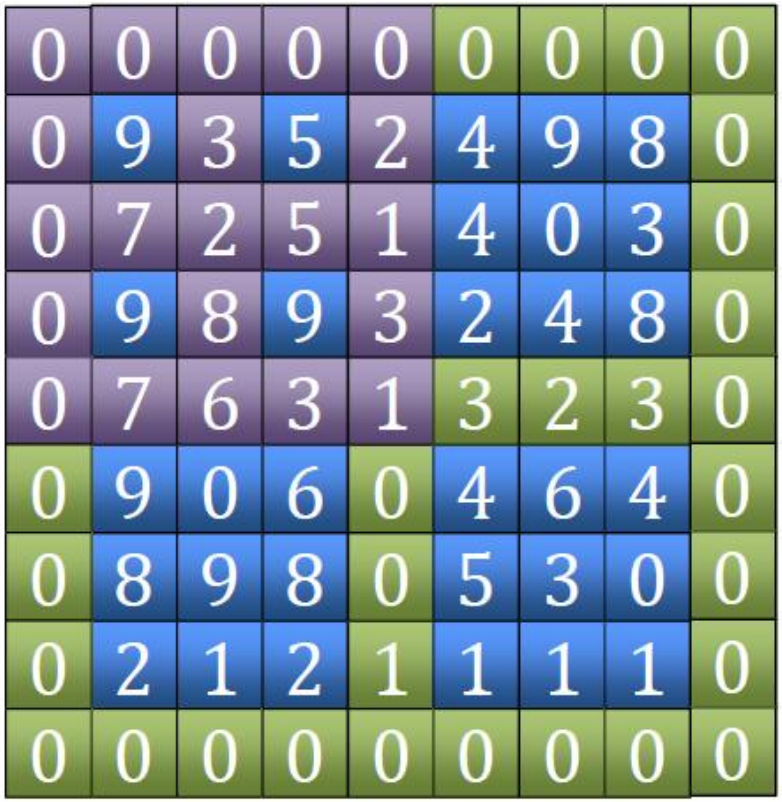
- Lemma: If you enter a quadrant, it contains a peak of the overall array [climb up]
- Invariant: Maximum element of window never decreases as we descend in recursion
- Theorem: Peak in visited quadrant is also peak in overall array



→ proofs in recitation

Analysis #2

- Reduce $n \times n$ matrix to $\sim \frac{n}{2} \times \frac{n}{2}$ submatrix in $O(n)$ time (|window|)



$$T(n) = T\left(\frac{n}{2}\right) + c\,n$$

$$T(n) = T\left(\frac{n}{4}\right) + c\frac{n}{2} + c\,n$$

$$T(n) = T\left(\frac{n}{8}\right) + c\frac{n}{4} + c\frac{n}{2} + c\,n$$

$\Theta(n)$

$$T(n) = T(1) + c\left(1 + 2 + 4 + \cdots + \frac{n}{4} + \frac{n}{2} + n\right)$$

第四章：小结

4.1分治法的设计思想

4.2最大子段和

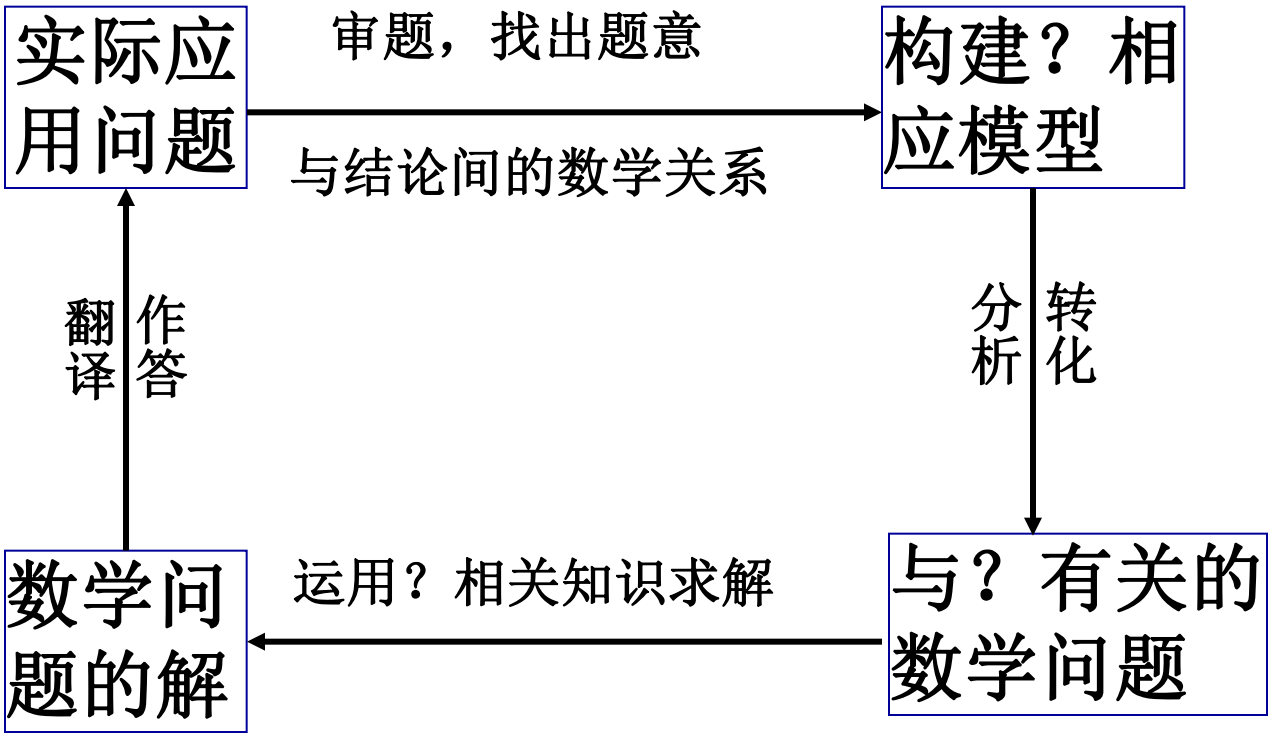
4.3求解递归式

4.4排序问题中的分治法-快速排序

4.4组合问题中的分治法

4.5几何问题中的分治法

第四章：小结


实际应用问题
审题，找出题意
与结论间的数学关系
构建？相应模型
翻译
作答
分析
转化
数学问题的解
运用？相关知识求解
与？有关的数学问题