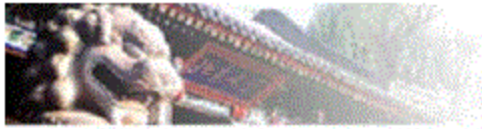
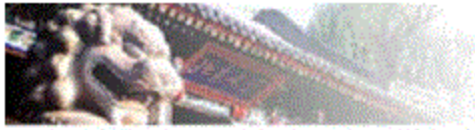


第2章 分治策略
（Divide and Conquer）



2.1.1 两个熟悉的例子

二分检索

算法2.1 BinarySearch(T, l, r, x)

输入：数组T，下标从l到r；数x

输出：j　// 如果x在T中，j为下标；否则为0

1. $l \leftarrow 1$; $r \leftarrow n$

2. while $l \le r$ do

3.　　$m \leftarrow \lfloor (l+r)/2 \rfloor$

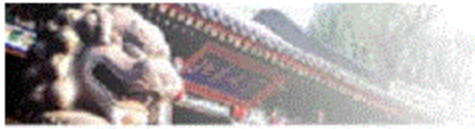
4.　　if $T[m]=x$ then return m　// x恰好等于中位元素

5.　　else if $T[m]>m$ then $r \leftarrow m-1$

6.　　else $l \leftarrow m+1$

7. return 0

二分归并排序 MergeSort（见第1章）



时间复杂度分析

二分检索最坏情况下时间复杂度$W(n)$满足

$$\begin{cases} W(n) = W(\left\lfloor \frac{n}{2} \right\rfloor) + 1 \\ W(1) = 1 \end{cases}$$

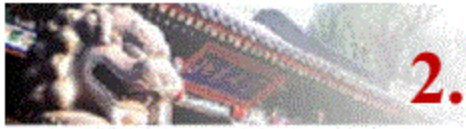
可以解出$W(n)=\lfloor \log n \rfloor + 1$.

二分归并排序最坏情况下时间复杂度$W(n)$满足

$$\begin{cases} W(n) = 2W(\frac{n}{2}) + n - 1 \\ W(1) = 0 \end{cases}$$

可以解出$W(n)=n\log n - n + 1$.

北京大学



2.1.2 分治算法的一般性描述

分治算法 Divide-and-Conquer(P)

1. if $|P| \le c$ then $S(P)$.
2. divide P into $P_1, P_2, \ldots, P_k$.
3. for $i = 1$ to k
4. y_i = Divide-and-Conquer(P_i)
5. Return Merge($y_1, y_2, \ldots, y_k$)

算法时间复杂度的递推方程

$$\begin{cases} W(n) = W(|P_1|) + W(|P_2|) + \ldots + W(|P_k|) + f(n) \\ W(c) = C \end{cases}$$

一般原则：子问题均匀划分、递归处理

2.2 分治算法的分析技术

分治策略的算法分析工具：递推方程

两类递推方程

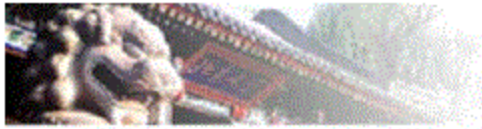
$$f(n)=\sum_{i=1}^{k}a_i f(n-i)+g(n)$$

$$f(n)=af(\frac{n}{b})+d(n)$$

求解方法

第一类方程：迭代法、换元法、递归树、尝试法

第二类方程：迭代法、递归树、主定理



递推方程的解

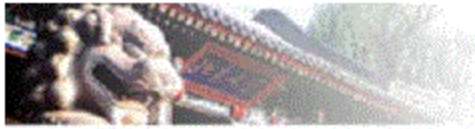
方程　$T(n) = aT(n/b) + d(n)$

$d(n)$为常数

$$T(n) = \begin{cases} O(n^{\log_b a}) & a \neq 1 \\ O(\log n) & a = 1 \end{cases}$$

$d(n) = cn$

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



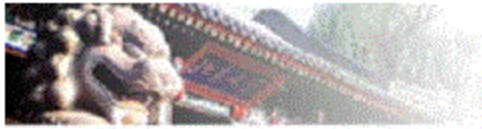
例2.1 芯片测试

条件：有n片芯片，(好芯片至少比坏芯片多1片).
问题：使用最少测试次数，从中挑出1片好芯片.

对芯片A与B测试，结果分析如下：

A报告	B报告	结论
B是好的	A是好的	A,B都好或A,B都坏
B是好的	A是坏的	至少一片是坏的
B是坏的	A是好的	至少一片是坏的
B是坏的	A是坏的	至少一片是坏的

算法思想：两两一组测试，淘汰后芯片进入下一轮.
如果测试结果是情况1，那么A、B中留1片，丢1片；
如果是后三种情况，则把A和B全部丢掉.



分治算法

命题2.1 当n是偶数时，在上述规则下，经过一轮淘汰，剩下的好芯片比坏芯片至少多1片.

证 设A与B都是好芯片有 i 组，A与B一好一坏有 j 组，A与B都坏有 k 组，淘汰后，好芯片数 i，坏芯片数 k

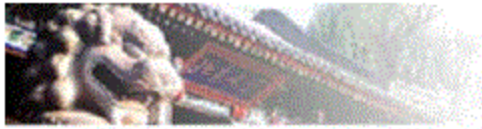
$$2i + 2j + 2k = n$$

$$2i+j > 2k+j \Rightarrow i > k$$

注：当n是奇数时，用其他芯片测试轮空芯片，如果轮空芯片是好的，算法结束；否则淘汰轮空芯片.

每轮淘汰后，芯片数至少减半，时间复杂度是：

$$\begin{cases} W(n) = W(\frac{n}{2}) + O(n) \\ W(3) = 1 \end{cases} \Rightarrow \quad W(n) = O(n)$$



伪码描述

算法2.3 Test(n)

1. $k \leftarrow n$
2. while $k>3$ do
3. 　将芯片分成$\lfloor k/2 \rfloor$组 // 如有轮空芯片，特殊处理
4. 　for $i=1$ to $\lfloor k/2 \rfloor$ do
　　if 2片好 then，则任取1片留下
　　else 2片同时丢掉
5. 　$k \leftarrow$剩下的芯片数
6. if $k=3$
　then 任取2片芯片测试
　　if 至少1坏 then 取没测的芯片
　　else 任取1片被测芯片
7. if $k=2$ or 1 then 任取1片

例2.2 幂乘计算

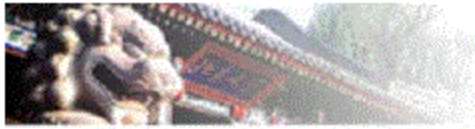
问题：设a是给定实数，计算 a^n, n为自然数

传统算法: $\Theta(n)$

分治法

$$a^n = \begin{cases} a^{n/2} \times a^{n/2} & n \text{ 为偶数} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a & n \text{ 为奇数} \end{cases}$$

$W(n) = W(n/2) + \Theta(1) \Rightarrow W(n) = \Theta(\log n)$.



计算 Fibonacci 数

Fibonacci 数

1, 1, 2, 3, 5, 8, 13, 21, …

满足 $F_n=F_{n-1}+F_{n-2}$

增加$F_0=0$，得到数列　0, 1, 1, 2, 3, 5, 8, 13, 21, …

通常算法：从 $F_0, F_1, \ldots$, 根据定义陆续相加，时间为$\Theta(n)$

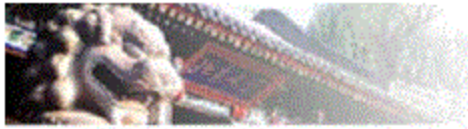
定理2.1 设 $\{F_n\}$为 Fibonacci 数构成的数列，那么

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

算法：令矩阵 $M=\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$，用分治法计算 M^n

时间 $T(n)=\Theta(\log n)$.

北京大学



2.3 改进分治算法的途径

2.3.1 通过代数变换 减少子问题个数

例2.3 位乘问题

设X,Y是n位二进制数，$n=2^k$, 求XY.

一般分治法　令$X=A2^{n/2}+B$, $Y=C2^{n/2}+D$.

$$XY=AC\,2^n+(AD+BC)\,2^{n/2}+BD$$

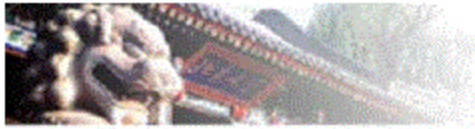
$$W(n)=4W(n/2)+cn,\quad W(1)=1$$

$$W(n)=O(n^2)$$

代数变换　$AD+BC=(A-B)(D-C)+AC+BD$

$$W(n)=3\,W(n/2)+cn,\quad W(1)=1$$

$$W(n)=O(n^{\log 3})=(n^{1.59})$$



矩阵乘法

例2.4 A,B 为两个n 阶矩阵，$n=2^k$, 计算 $C=AB$.

传统算法 $W(n)= O(n^3)$

分治法　将矩阵分块，得

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}=\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

其中

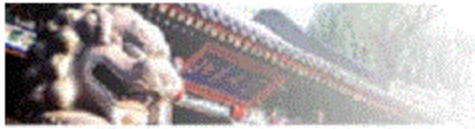
$$C_{11}=A_{11}B_{11}+A_{12}B_{21} \qquad C_{12}=A_{11}B_{12}+A_{12}B_{22}$$

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

递推方程 $W(n) = 8\ W(n/2) + cn^2$

$W(1) = 1$

解　$W(n) = O(n^3)$.



Strassen 矩阵乘法

变换方法：

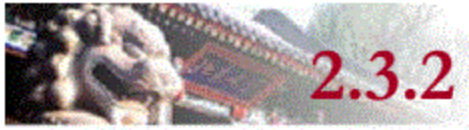
$$M_1 = A_{11}(B_{12} - B_{22})$$
$$M_2 = (A_{11} + A_{12})B_{22}$$
$$M_3 = (A_{21} + A_{22})B_{11}$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

$$C_{11} = M_5 + M_4 - M_2 + M_6$$
$$C_{12} = M_1 + M_2$$
$$C_{21} = M_3 + M_4$$
$$C_{22} = M_5 + M_1 - M_3 - M_7$$

时间复杂度：

$$W(n) = 7W(\tfrac{n}{2}) + 18(\tfrac{n}{2})^2$$
$$W(1) = 1$$

$$W(n) = O(n^{\log_2 7})$$
$$= O(n^{2.8075})$$



2.3.2 利用预处理减少递归操作

算法中的处理尽可能提到递归外面作为预处理

例2.5 平面点对问题

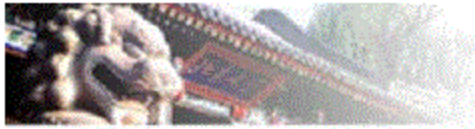
输入：集合S中有n个点，$n>1$，

输出：所有的点对之间的最小距离.

通常算法：$C(n,2)$个点对计算距离，比较最少需$O(n^2)$时间

分治策略：子集P中的点划分成两个子集P_L和P_R

$$|P_L|=\left\lceil\frac{|P|}{2}\right\rceil \qquad |P_R|=\left\lfloor\frac{|P|}{2}\right\rfloor$$



平面最邻近点对算法

MinDistance(*P*,*X*,*Y*)

输入：*n* 个点的集合*P*, *X* 和*Y* 分别为横、纵坐标数组

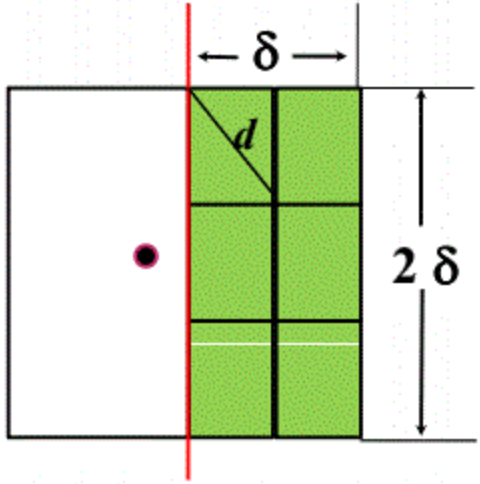
输出：最近的两个点及距离

1. 如果*P* 中点数小于等于3，则直接计算其中的最小距离
2. 排序*X*,*Y*
3. 做垂直线 *l* 将*P*划分为P_L和P_R，P_L的点在 *l* 左边，P_R的点在 *l* 右边
4. MinDidtance(P_L,X_L,Y_L); $\delta_L = P_L$中的最小距离
5. MinDistance(P_R,X_R,Y_R); $\delta_R = P_R$中的最小距离
6. $\delta = \min(\delta_L, \delta_R)$
7. 对于在垂直线两边距离δ范围内的每个点，检查是否有点与它的距离小于δ，如果存在则将δ修改为新值

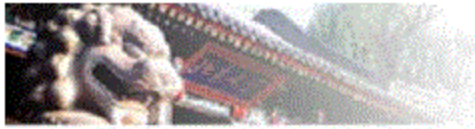
北京大学

跨边界的最邻近点

$$d = \sqrt{(\delta/2)^2 + (2\delta/3)^2}$$
$$= \sqrt{\delta^2/4 + 4\delta^2/9}$$
$$= \sqrt{25\delta^2/36} = 5\delta/6$$



右边每个小方格至多1个点，每个点至多比较对面的6个点，距离≤δ的2个点(左右各1个)其纵坐标位置相差不超过12，检查1个点是常数时间，$O(n)$ 个点需要$O(n)$时间



算法分析

分析：步1 $O(1)$

步2 $O(n\log n)$

步3 $O(1)$

步4-5 $2T(n/2)$

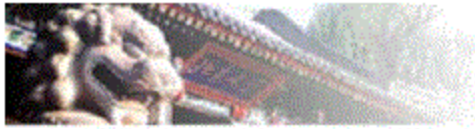
步6 $O(1)$

步7 $O(n)$

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

$$T(n) = O(1) \qquad n \le 3$$

由递归树估计 $T(n) = O(n\log^2 n)$



预排序的处理方法

在每次调用时将已经排好的数组分成两个排序的子集，每次调用这个过程的时间为$O(n)$

$W(n)$总时间，$T(n)$算法递归过程，$O(n\log n)$预处理排序

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

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

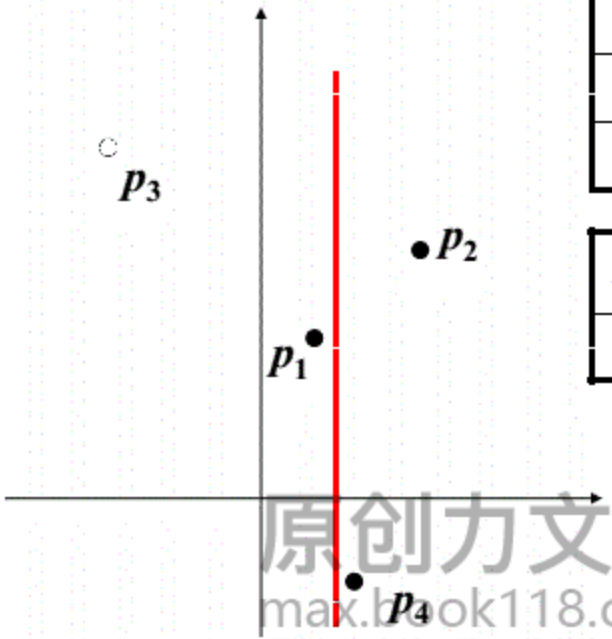
$$T(n) = O(1) \qquad n \le 3$$

解得

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

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

实例：递归中的拆分

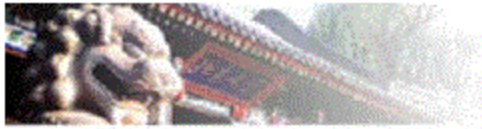


P	1	2	3	4
X	0.5	2	-2	1
Y	2	3	4	-1

X	-2(3)	0.5(1)	1(4)	2(2)
Y	-1(4)	2(1)	3(2)	4(3)

X_L	-2(3)	0.5(1)
X_R	1(4)	2(2)
Y_L	2(1)	4(3)
Y_R	-1(4)	3(2)

北京大学



典型实例分析

2.4.1 快速排序

算法 Quicksort(*A*,*p*,*r*)

输入：数组*A*[*p*..*r*]

输出：排好序的数组 *A*

```
1. if p < r
2.   then q←Partition(A, p, r)
3.        A[p]↔A[q]
4.        Quicksort(A,p,q-1)
5.        Quicksort(A,q+1,r)
```

初始置*p*=1, *r*=*n*，然后调用上述算法

划分过程

Partition(*A*,*p*,*r*)

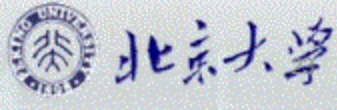
输入：数组*A*[*p*,*r*]

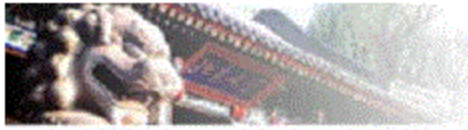
输出：*j*，*A*的首元素在排好序的数组中的位置

1. $x \leftarrow A[p]$
2. $i \leftarrow p$
3. $j \leftarrow r+1$
4. while true do
5. repeat $j \leftarrow j-1$
6. until $A[j] \leq x$
7. repeat $i \leftarrow i+1$
8. until $A[i] \geq x$
9. if $i < j$
10. then $A[i] \leftrightarrow A[j]$
11. else return j

划分实例

27	99 *i*	0	8	13	64	86	16	7	10	88	25 *j*	90
27	25	0	8	13	64 *i*	86	16	7	10 *j*	88	99	90
27	25	0	8	13	10	86 *i*	16	7 *j*	64	88	99	90
27	25	0	8	13	10	7	16 *j*	86 *i*	64	88	99	90
16	25	0	8	13	10	7	27	86	64	88	99	90





最坏情况下的时间复杂度

最坏情况

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

$$W(1) = 0$$

$$W(n) = \frac{1}{2} n(n-1) = \Theta(n^2)$$

最好划分

$$T(n) = 2T(\tfrac{n}{2}) + n - 1$$

$$T(1) = 0$$

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

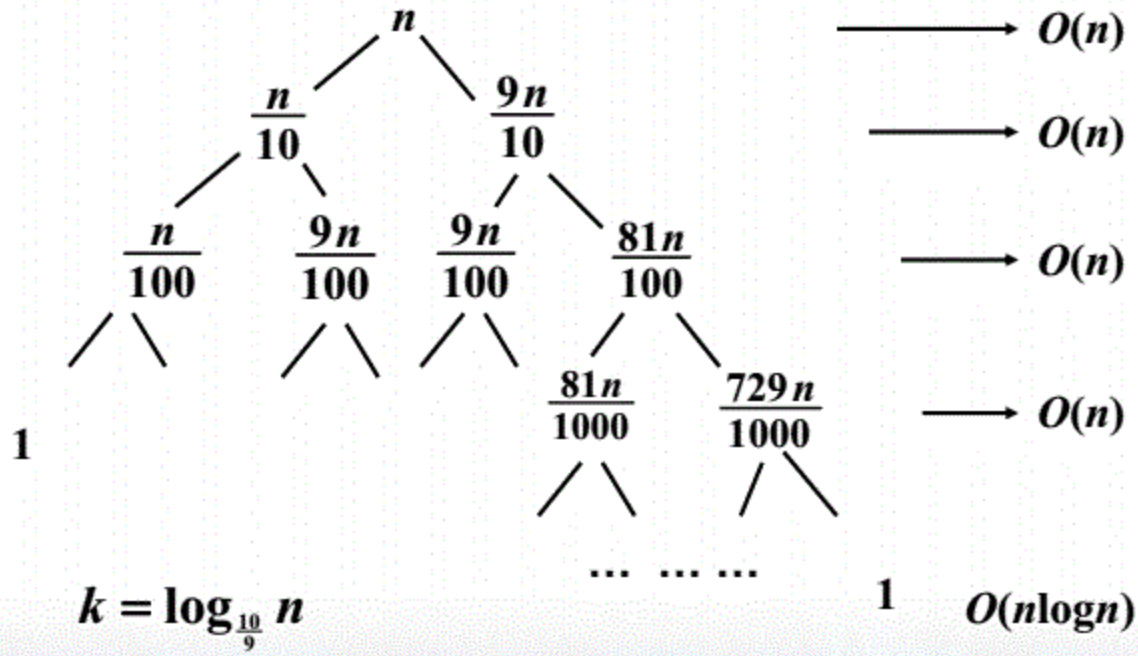
均衡划分

$$T(n) = T(\tfrac{9n}{10}) + T(\tfrac{n}{10}) + n$$

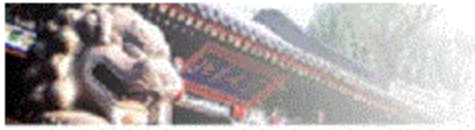
$$T(1) = 0$$

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

均衡划分



$k = \log_{\frac{10}{9}} n$



平均情况下时间复杂度

假设输入数组首元素排好序后的正确位置处在1,2,…,n 各种情况是等可能的，概率为1/n.

$$T(n) = \frac{1}{n}\sum_{k=1}^{n-1}(T(k) + T(n-k)) + n - 1$$

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

$$T(1) = 0$$

利用差消法求得 $T(n)=O(n\log n)$

2.4.2 选择问题

问题：从给定的集合 L 中选择第 i 小的元素

不妨设 L 为 n 个不等的实数

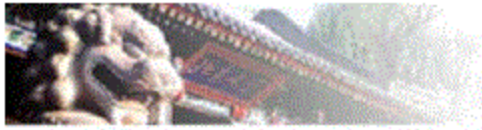
i=1, 称为最小元素；

i=n，称为最大元素；

位置处在中间的元素，称为中位元素

当n为奇数时，中位数只有1个，i=(n+1)/2；

当n为偶数时，中位数有2个，i=n/2, n/2+1.



选最大

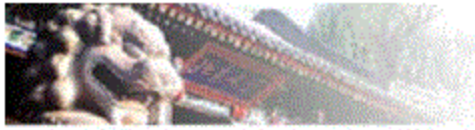
算法 Findmax

输入： n 个数的数组L

输出： max

1. $max \leftarrow L[1]$;
2. for $i \leftarrow 2$ to n do
3. if $max < L[i]$
4. then $max \leftarrow L[i]$
5. $k \leftarrow i$
5. return max

算法最坏情况下的时间复杂度 $W(n)=n-1$



选最大和最小

通常算法：顺序比较

复杂性：$W(n)=2n-3$

算法 FindMaxMin

输入：n个数的数组L

输出：max，min

1. 将n个元素两两一组分成$\lfloor n/2 \rfloor$组
2. 每组比较，得到$\lfloor n/2 \rfloor$个较小和$\lfloor n/2 \rfloor$个较大
3. 在$\lceil n/2 \rceil$个(n为奇数，是$\lfloor n/2 \rfloor+1$)较小中找最小min
4. 在$\lceil n/2 \rceil$个(n为奇数，是$\lfloor n/2 \rfloor+1$)较大中找最大max

复杂性：行2 比较$\lfloor n/2 \rfloor$次，行3--4 比较至多$2\lceil n/2 \rceil-2$次，

$$W(n)=\lfloor n/2 \rfloor+2\lceil n/2 \rceil-2=n+\lceil n/2 \rceil-2=\lceil 3n/2 \rceil-2$$

找第二大

通常算法：顺序比较

1．顺序比较找到最大*max*;

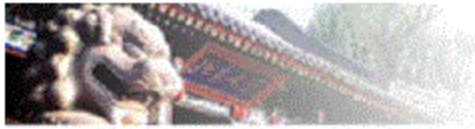
2．从剩下的$n-1$个数中找最大，就是第二大*second*

复杂性：$W(n)=n-1+n-2=2n-3$

锦标赛算法：

两两分组比较，大者进入下一轮

每个元素用数表记录每次比较时小于自己的元素



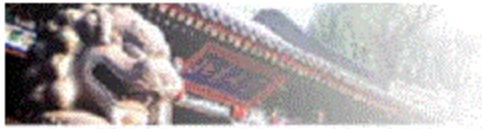
锦标赛算法

算法 FindSecond

输入：n个数的数组L

输出：*Second*

1. $k \leftarrow n$
2. 将 k 个元素两两一组，分成$\lfloor k/2 \rfloor$组
3. 每组的2个数比较，找到较大的数
4. 将被淘汰的较小的数在淘汰它的数所指向的链表中做记录
5. if k 为奇数 then $k \leftarrow \lfloor k/2 \rfloor + 1$
6. else $k \leftarrow \lfloor k/2 \rfloor$
7. if $k>1$ then goto 2
8. *max* ←最大数
9. *second* ← *max* 的链表中的最大



时间复杂度分析

命题2.2 *max*在第一阶段的分组比较中总计进行了$\lceil \log n \rceil$次比较.

证 设本轮参与比较的有 t 个元素，经过分组淘汰后进入下一轮的元素数至多是$\lceil t/2 \rceil$. 假设 k 轮淘汰后只剩下一个元素 *max*，利用

$$\lceil \lceil t/2 \rceil /2 \rceil = \lceil t/2^2 \rceil$$

的结果并对 k 归纳，可得到 $\lceil n/2^k \rceil = 1$.

若 $n=2^d$，那么有 $k=d=\log n=\lceil \log n \rceil$

若 $2^d<n<2^{d+1}$，那么 $k=d+1=\lceil \log n \rceil$

算法时间复杂度是

$$W(n)=n-1+\lceil \log n \rceil -1= n+\lceil \log n \rceil -2.$$

北京大学

一般性选择问题

问题：选第 k 小.

输入：数组 S, S的长度 n, 正整数 k, $1 \le k \le n$.

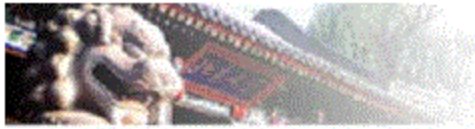
输出：第 k 小的数

通常算法

1．排序

2．找第 k 小的数

时间复杂性：$O(n\log n)$



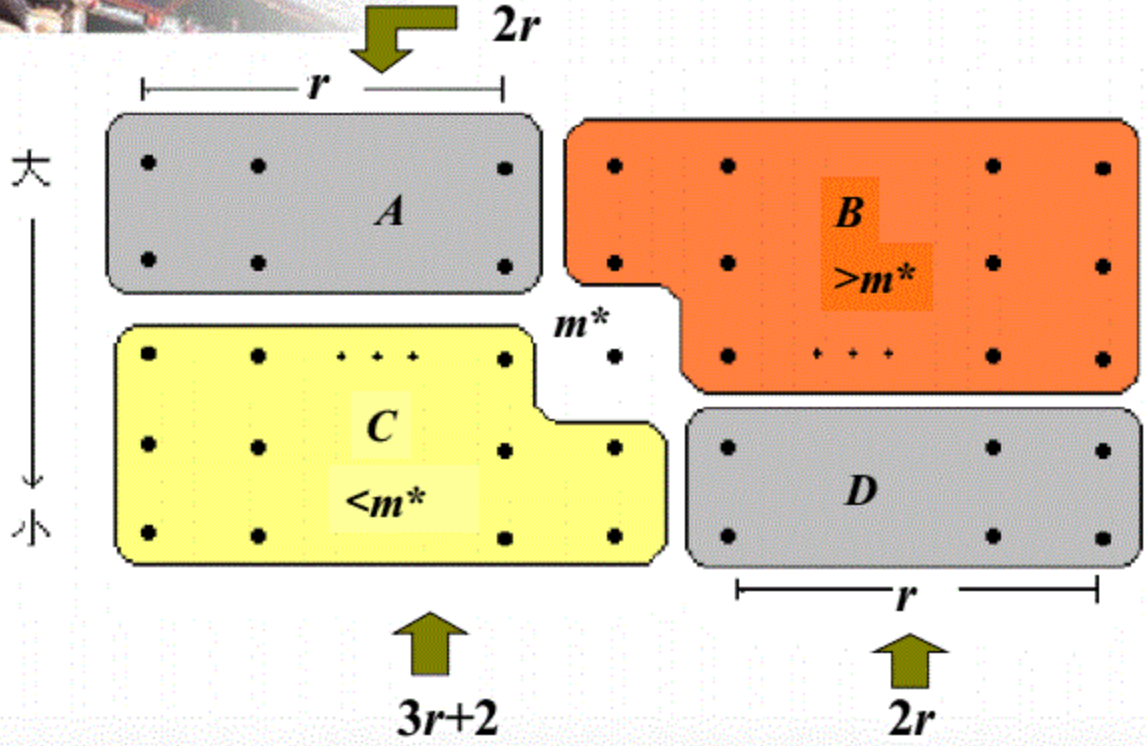
分治选择算法

算法 Select(S,k)

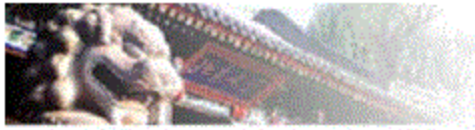
输入：数组 S，正整数 k

输出：S中的第 k 小元素

1. 将S划分成 5 个一组，共 $n_M=\lceil n/5\rceil$个组
2. 每组找中位数，n_M个中位数放到集合 M
3. $m^*\leftarrow$Select($M,\lceil |M|/2\rceil$)　//将S划分成A,B,C,D四个集合
4. 把A和D的每个元素与m^*比较，小的构成S_1, 大的构成S_2
5. $S_1\leftarrow S_1\cup C$; $S_2\leftarrow S_2\cup B$
6. if $k=|S_1|+1$ then 输出m^*
7. else if $k\le|S_1|$
8. 　　then Select(S_1,k)
9. 　　else Select($S_2,k-|S_1|-1$)



最坏情况：子问题大小为 $2r + 2r + 3r + 2 = 7r + 2$



复杂度估计 $W(n)=O(n)$

不妨设 $n=5(2r+1)$, $|A|=|D|=2r$, $r=\frac{\frac{n}{5}-1}{2}=\frac{n}{10}-\frac{1}{2}$

算法工作量

行2： $O(n)$

行3： $W(n/5)$

行4： $O(n)$

行8-9： $W(7r+2)$

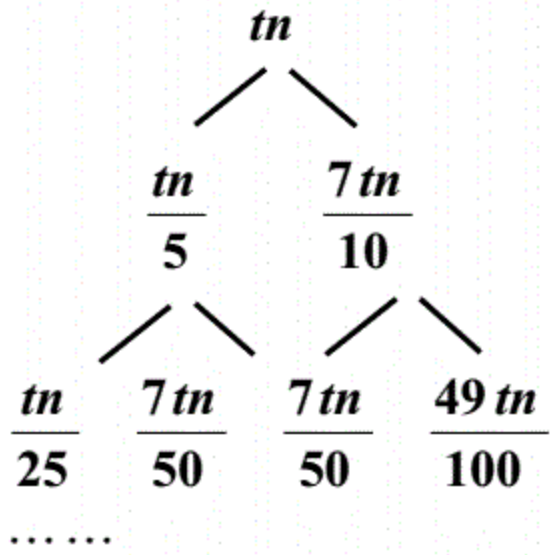
$$W(7r+2)=W(7(\frac{n}{10}-\frac{1}{2})+2)$$

$$=W(\frac{7n}{10}-\frac{3}{2})\le W(\frac{7n}{10})$$

用递归树做复杂度估计

$$w(n)\le W(\frac{n}{5})+W(\frac{7n}{10})+cn\le cn+\frac{9}{10}cn+\frac{81}{100}cn+\ldots=O(n)$$

递归树



tn

$0.9tn$

$0.81tn$

1 的2n次根

$$\omega_j = e^{\frac{2\pi j}{2n}i} = e^{\frac{\pi j}{n}i} = \cos\frac{\pi j}{n} + i\sin\frac{\pi j}{n} \qquad j=0,1,\ldots,2n-1$$

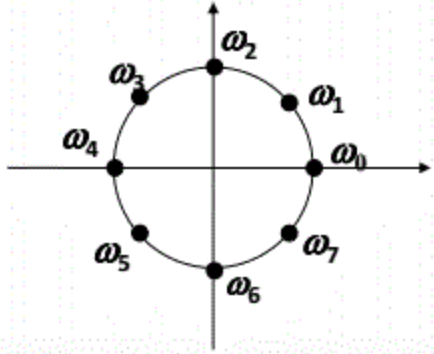
例如n=4, 1的8次方根是：

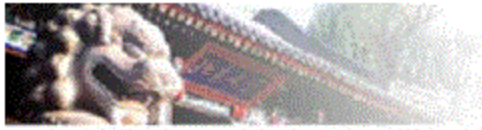
$$\omega_0 = 1, \qquad \omega_1 = e^{\frac{\pi}{4}i} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i,$$

$$\omega_2 = e^{\frac{\pi}{2}i} = i, \quad \omega_3 = e^{\frac{3\pi}{4}i} = -\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i,$$

$$\omega_4 = e^{\pi i} = -1, \ \omega_5 = e^{\frac{5\pi}{4}i} = -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i,$$

$$\omega_6 = e^{\frac{3\pi}{2}i} = -i, \ \omega_7 = e^{\frac{7\pi}{4}i} = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i$$





多项式求值

给定多项式： $A(x)=a_0+a_1x+a_2x^2+\ldots+a_{n-1}x^{n-1}$

设 x 为1 的 $2n$ 次方根，对所有的 x 计算 $A(x)$ 的值.

算法1：对每个 x 做下述运算：

依次计算每个项 a_ix^i，对 i 求和得到 $A(x)$，

$T_1(n)=O(n^3)$

算法2：$A_1(x)=a_{n-1}$

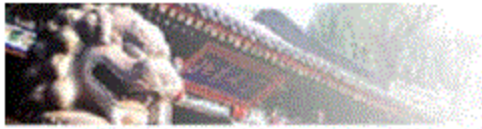
$A_2(x)=a_{n-2}+xA_1(x)$

$A_3(x)=a_{n-3}+xA_2(x)$

…

$A_n(x)=a_0+xA_{n-1}(x)=A(x)$

$T_2(n)=O(n^2)$



分治算法

原理：

$A_{even}(x)=a_0+a_2x+a_4x^2+\ldots+a_{n-2}x^{(n-2)/2}$

$A_{odd}(x)=a_1+a_3x+a_5x^2+\ldots+a_{n-1}x^{(n-2)/2}$

$A(x)=A_{even}(x^2)+xA_{odd}(x^2)$，$x^2$ 为1 的 n 次根

算法3：

1. 计算1 的所有的 $2n$ 次根
2. 分别计算 $A_{even}(x^2)$ 与 $A_{odd}(x^2)$
3. 利用步2 的结果计算 $A(x)$

复杂度分析：$T(n)=T_1(n)+f(n)$，$f(n)=O(n)$计算$2n$次根时间

$T_1(n)=2T_1(n/2)+g(n)$，$g(n)=O(n)$，

$T_1(1)=O(1)$

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