

第2章 分治策略

(Divide and Conquer)

2.1 分治策略的基本思想

2.1.1 两个熟悉的例子

2.1.2 分治算法的一般性描述

2.2 分治算法的分析技术

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

2.3.2 利用预处理减少递归内部的计算量

2.4 典型实例

2.4.1 快速排序算法

2.4.2 选择问题

2.4.3 $n-1$ 次多项式在全体 $2n$ 次方根上的求值

2.1.1 两个熟悉的例子

二分检索

算法2.1 BinarySearch(T, x)

输入：排好序数组 T ，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 $l \leq r$ 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 $T[m]>x$ 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

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

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

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

可以解出 $W(n) = n \log n - n + 1$.

2.1.2 分治算法的一般性描述

分治算法 Divide-and-Conquer(P)

P是待求解的问题，**|P|**是问题输入规模

1. **if** $|P| \leq c$ **then** $S(P)$. //若问题规模不超c，直接求解
2. **divide** P **into** P_1, P_2, \dots, P_k //否则将P归约为k个独立子问题
3. **for** $i \leftarrow 1$ **to** k //递归或迭代的归约
4. $y_i \leftarrow \text{Divide-and-Conquer}(P_i)$
5. **Return** $\text{Merge}(y_1, y_2, \dots, y_k)$

算法时间复杂度的递推方程

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

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：

$$T(n) = \sum_{i=1} a_i T(n-i) + f(n)$$

$$T(n) = aT\left(\frac{n}{b}\right) + d(n)$$

例子：

Hanoi塔, $W(n)=2W(n-1)+1$

二分检索, $W(n)=W(n/2)+1$

求解方法：

第一类方程：直接迭代法、换元法、递归树、尝试法

第二类方程：直接迭代法、递归树、主定理

利用主定理求解的常见形式

方程 $T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$ 为常数.

当 $f(n)$ 为常数时,

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

主定理第一种情况

主定理第二种情况

当 $f(n) = cn$ 时,

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

主定理第三种情况

主定理第二种情况

主定理第一种情况

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.

判定芯片A的好坏

问题：给定芯片A，判定A的好坏

方法：用其他 $n-1$ 片芯片对 A 测试。

$n=7$ ：好芯片数 ≥ 4 。

A 好，6个报告中至少 3 个报“好”

A 坏，6个报告中至少 4 个报“坏”

n 是**奇数**：好芯片数 $\geq (n+1)/2$ 。

A 好，至少有 $(n-1)/2$ 个报“好”

A 坏，至少有 $(n+1)/2$ 个报告“坏”

结论：至少一半报“好”，A是好芯片，超过一半报“坏”，A是坏芯片。

判定芯片A的好坏

$n=8$: 好芯片数 ≥ 5 .

A 好, 7个报告中至少 4 个报“好”

A 坏, 7个报告中至少 5 个报“坏”

n 是偶数: 好芯片数 $\geq n/2+1$.

A 好, 至少有 $n/2$ 个报告“好”

A 坏, 至少有 $n/2+1$ 个报告“坏”

结论: $n-1$ 份报告中, 至少一半报“好”, 则 A 为好芯片; 超过一半报“坏”, 则 A 为坏芯片

蛮力方法挑出1片好芯片: 最坏情况需要判断 $n/2$ 个芯片, $W(n)=(n-1)+(n-2)+\dots+(n/2-1)=O(n^2)$

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 ，坏芯片数至多 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：

$$\begin{cases} W(n) = W\left(\frac{n}{2}\right) + O(n) & n > 3 \\ W(n) = 1 & n \leq 3 \end{cases} \Rightarrow W(n) = O(n)$$

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, 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 1好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, \dots , 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

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

易用归纳法证明

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

时间复杂度 $T(n) = \Theta(\log n)$.

2.3 改进分治算法的途径

改进途径1：减少子问题个数

- 适用于：子问题个数多，划分和综合工作量不太大的算法。

$$W(n) = aW(n/b) + f(n),$$

当 a 较大, b 较小, $f(n)$ 不大:

$$W(n) = \Theta(n^{\log_b a})$$

- 减少 a 是降低函数 $W(n)$ 的阶的途径：利用子问题依赖关系，用某些子问题解的代数表达式表示另一些子问题的解，减少独立计算子问题个数。
- 综合解的工作量可能会增加，但增加的工作量不影响 $W(n)$ 的阶。

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 .

X	<table border="1"><tr><td>A</td><td>B</td></tr></table>	A	B
A	B		
Y	<table border="1"><tr><td>C</td><td>D</td></tr></table>	C	D
C	D		

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 $AD + BC = (A - B)(D - C) + AC + BD$

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

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

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

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

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:

$$W(n) = 7W\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$

$$W(1) = 1$$

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

$$= O(n^{2.8075})$$

目前已知矩阵乘法最好上界:

Coppersmith-Winograd 算法: $O(n^{2.376})$

2.3.2 利用预处理减少递归内部操作

改进途径2：增加预处理，减小 $f(n)$.

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

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

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

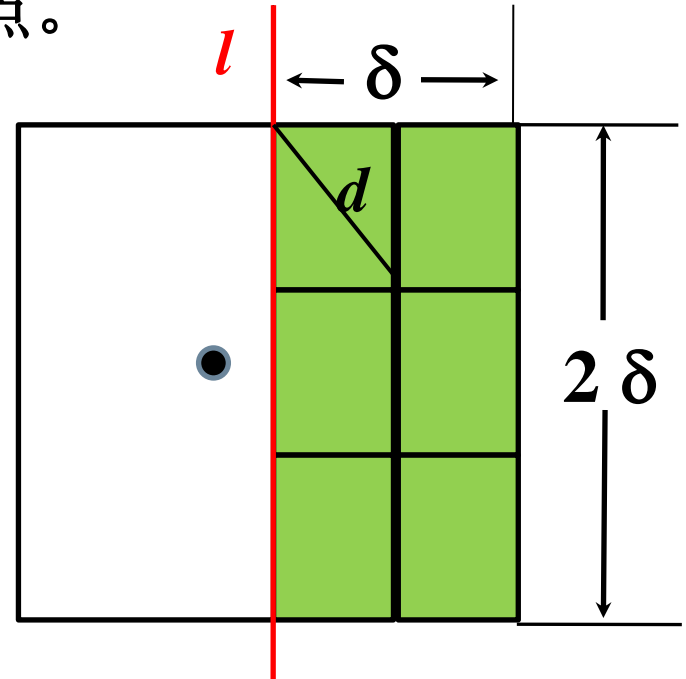
输出： 最近的两个点及距离

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. MinDistance(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 l 线左边距离 δ 内每个点, 检查右边是否有与之距离小于 δ 的点, 如果存在则将 δ 修改为新值

跨边界的最邻近点

第7步分析：先考虑左侧距离 l 小于 δ 的一个点， l 右侧最多需比较几个点。

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



右边每个方格至多1个点，左侧每个点至多比较对面的6个点。
检查1个点是常数时间，至多 $O(n)$ 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 $O(n)$

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

$$T(n) = O(1) \quad n \leq 3$$

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

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

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

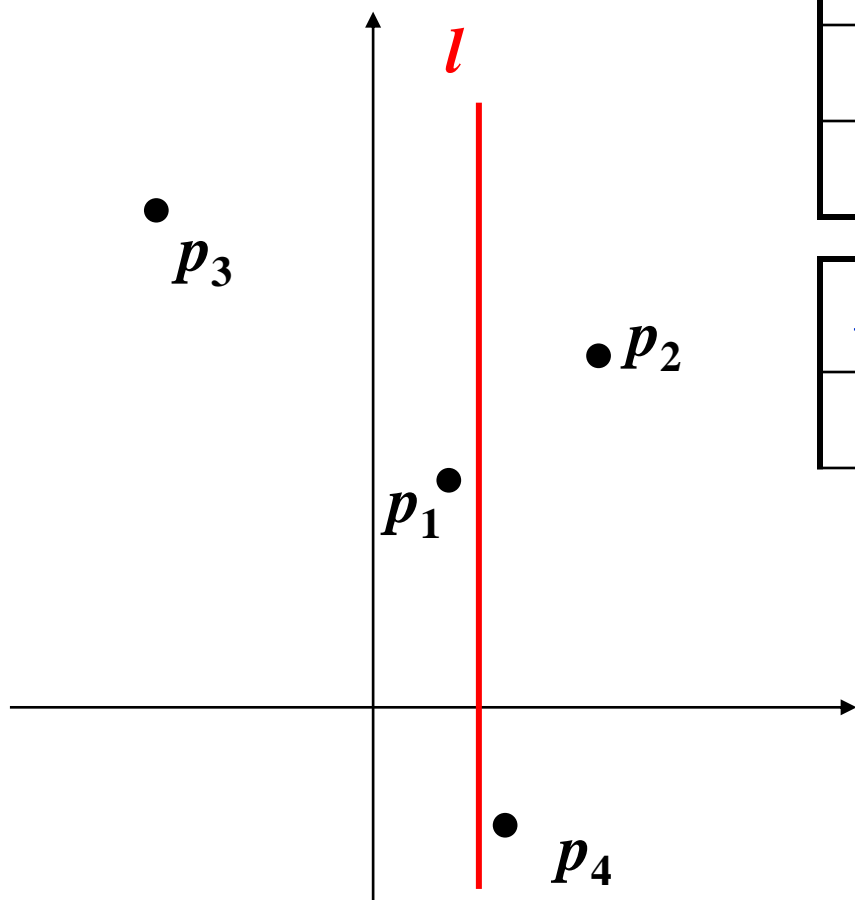
$$T(n) = O(1) \quad n \leq 3$$

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)

小结

- 依据

$$W(n) = aW(n/b) + f(n)$$

- 提高算法效率的方法:

- 减少子问题个数 a :

$$W(n) = O(n^{\log_b a})$$

- 增加预处理, 减少 $f(n)$

2.4 典型实例分析

2.4.1 快速排序

算法2.5 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 $q \leftarrow \text{Partition}(A, p, r)$
3. $A[p] \leftrightarrow A[q]$
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

划分实例

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

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

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

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

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

划分过程

算法2.6 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 $A[i] > x$
 9. if $i < j$
 10. then $A[i] \leftrightarrow A[j]$
 11. if $i = j - 1$ then return j
 12. else return j

不同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

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

$$T(1) = 0$$

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

均衡划分

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + n$$

$$T(1) = 0$$

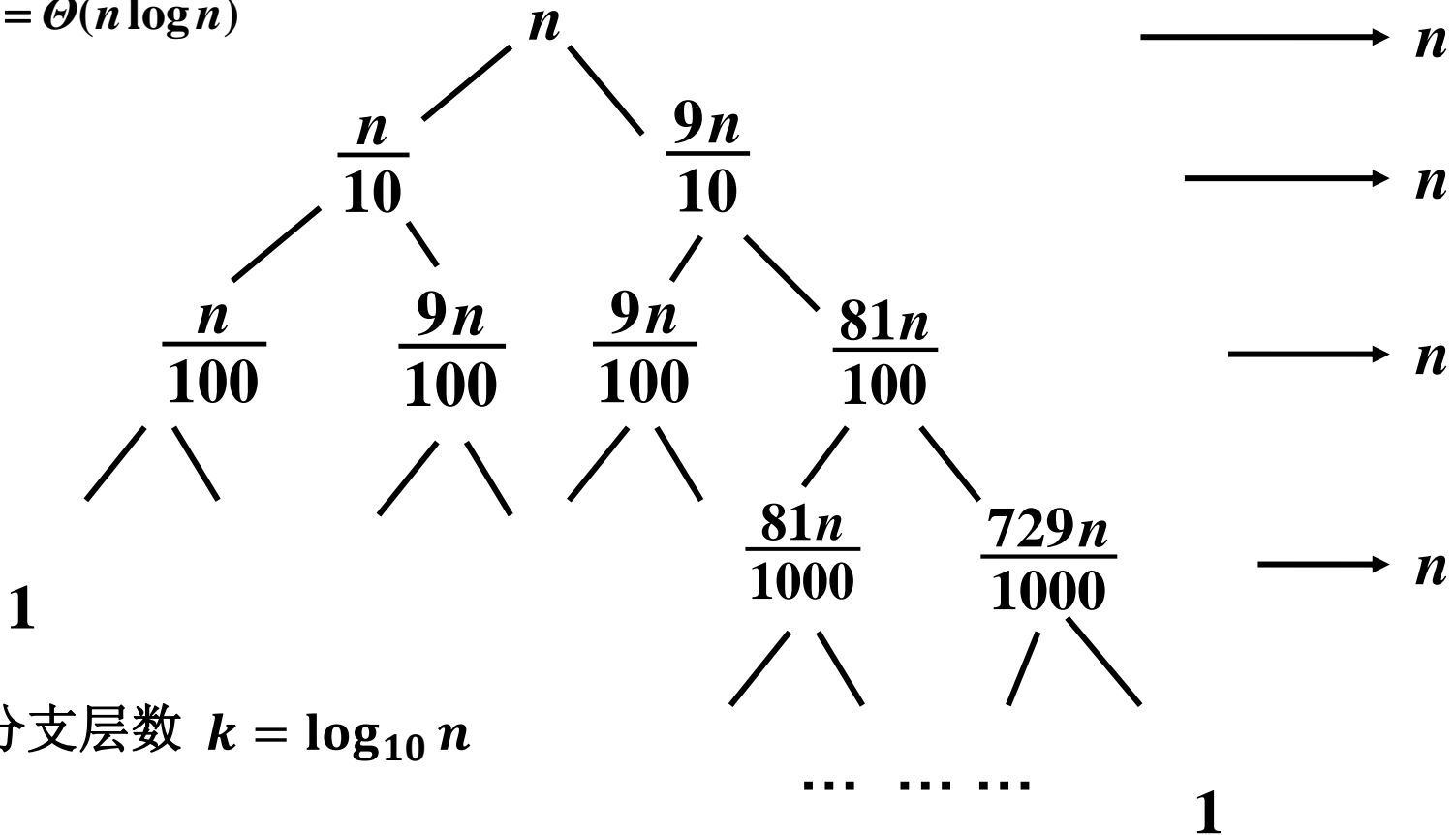
$$T(n) = \Theta(n \log n)$$

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + n$$

$$T(1) = 0$$

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

均衡划分



最左分支层数 $k = \log_{10} n$

最右分支层数 $k = \log_{\frac{10}{9}} n$

$$n \log_{10} n = O(n \log n) \leq T(n) \leq \log_{\frac{10}{9}} n = O(n \log n) \quad 29$$

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 $1, 2, \dots, n$ 各种情况是等可能的，概率为 $1/n$.

$$T(n) = \frac{1}{n} \{ [T(0)+T(n-1)] + [T(1)+T(n-2)] + \dots + [T(n-1)+T(0)] \} + O(n)$$

$$= \frac{2}{n} \sum_{i=0}^{n-1} T(i) + O(n)$$

例1.14

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

$$T(1) = 0$$

利用差消法求得 $T(n) = \Theta(n \log n)$

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;

$i=n-1$, 称为第二大元素;

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, $i=n/2, n/2+1$. 也可以规定其中的一个

选最大

算法2.7 Findmax

输入： n 个数的数组 L

输出： max, k

1. $max \leftarrow L[1]; k \leftarrow 1$
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, k

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2.9 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 $second$

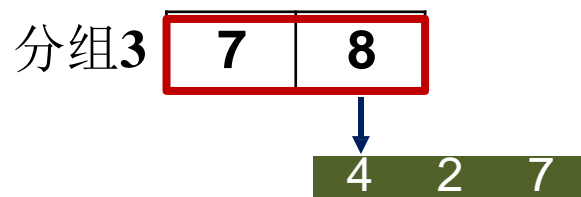
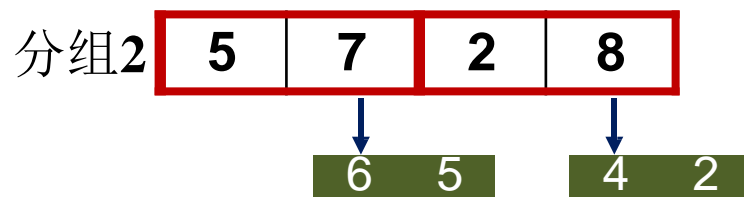
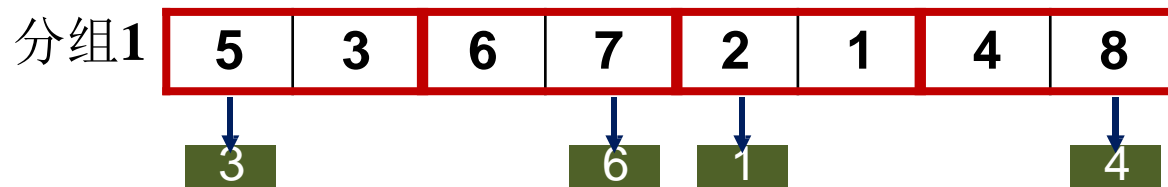
复杂性： $W(n)=n-1+n-2=2n-3$

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

实例



锦标赛算法

算法2.10 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. $max \leftarrow$ 最大数
9. $Second \leftarrow max$ 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

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

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 , $1 \leq k \leq n$.

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： $O(n \log n)$

实例: $n=15, k=6$

8	2	3	5	7	6	11	14	1	9	13	10	4	12	15
---	---	---	---	---	---	----	----	---	---	----	----	---	----	----

	8	14	15
	7	11	13
M	5	9	12
	3	6	10
	2	1	4

$m^* = 9$

	8	14	15
A	7	11	13
	5	9	12
C	3	6	10
	2	1	4

B

D

- C 中都比 m^* 小, B 中都比 m^* 大
- A 和 D 中的数 (8, 7, 10, 4) 需要与 m^* 比较
- 比 m^* 小的构成 S_1 , 大的构成 S_2 , 判断 k 落在哪个子集, 淘汰另一个

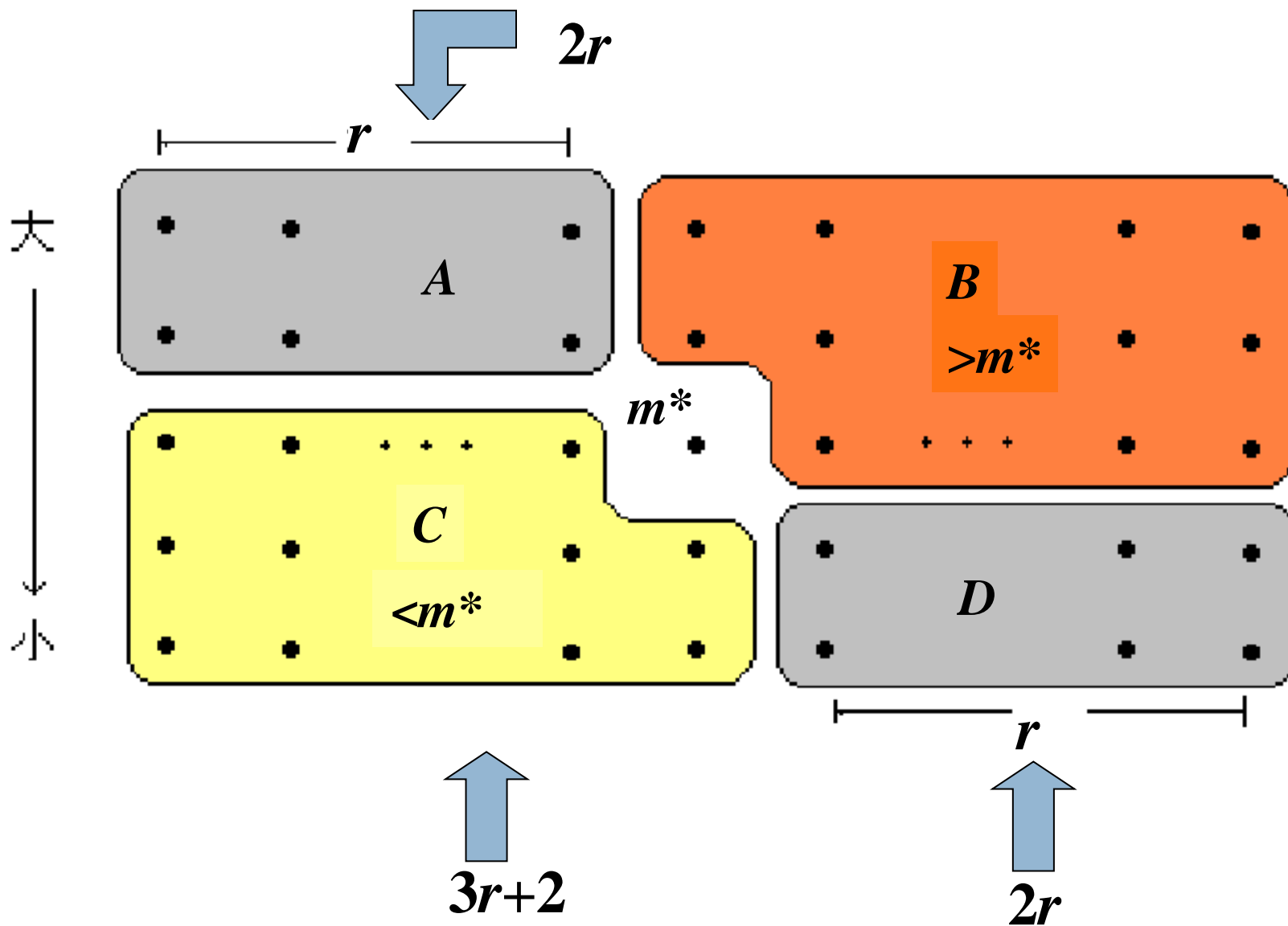
分治选择算法

算法2.11 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 $\lceil n/5 \rceil$ 个组
2. 每组找一个中位数，所有个中位数放到集合 M
3. $m^* \leftarrow \text{Select}(M, \lceil |M|/2 \rceil)$ //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 $k \leq |S_1|$
8. then $\text{Select}(S_1, k)$
9. else $\text{Select}(S_2, k - |S_1| - 1)$



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: $O(n)$

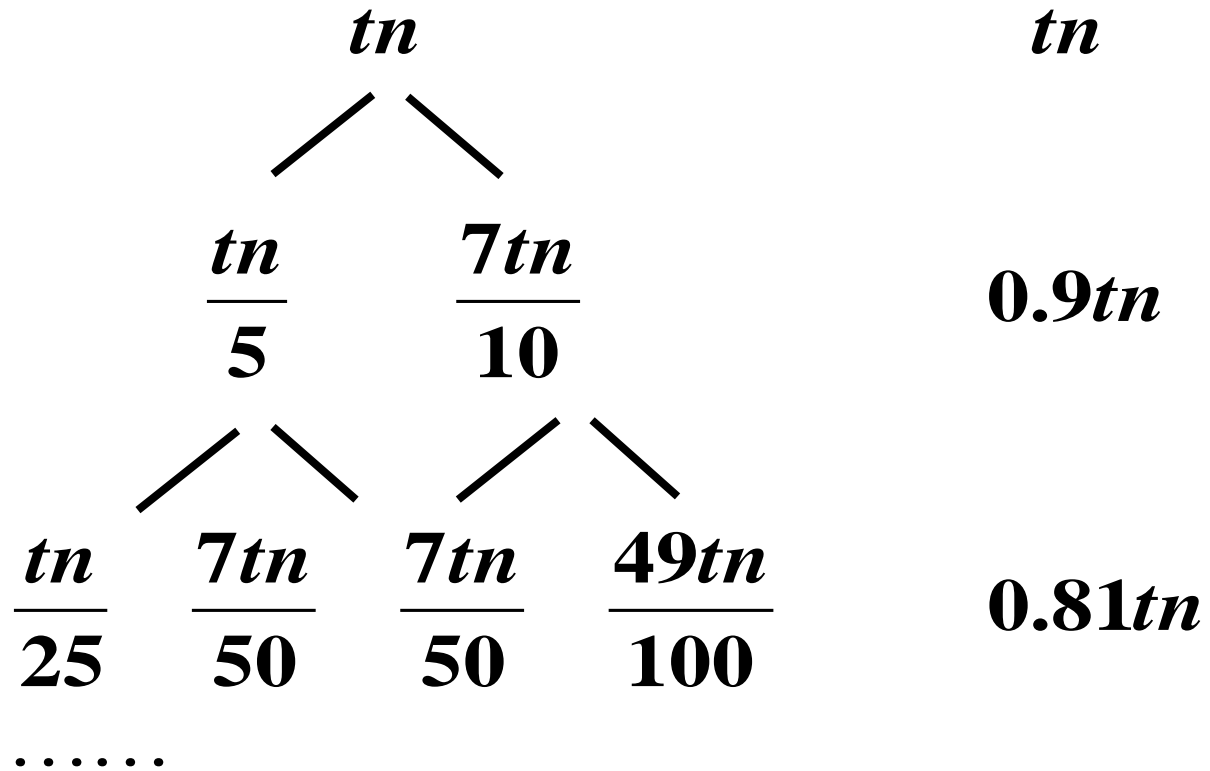
行8-9: $W(7r+2)$

$$\begin{aligned} W(7r+2) &= W\left(7\left(\frac{n}{10} - \frac{1}{2}\right) + 2\right) \\ &= W\left(\frac{7n}{10} - \frac{3}{2}\right) \leq W\left(\frac{7n}{10}\right) \end{aligned}$$

用递归树做复杂度估计

$$W(n) \leq W\left(\frac{n}{5}\right) + W\left(\frac{7n}{10}\right) + tn \leq tn + \frac{9}{10}tn + \frac{81}{100}tn + \dots = O(n)$$

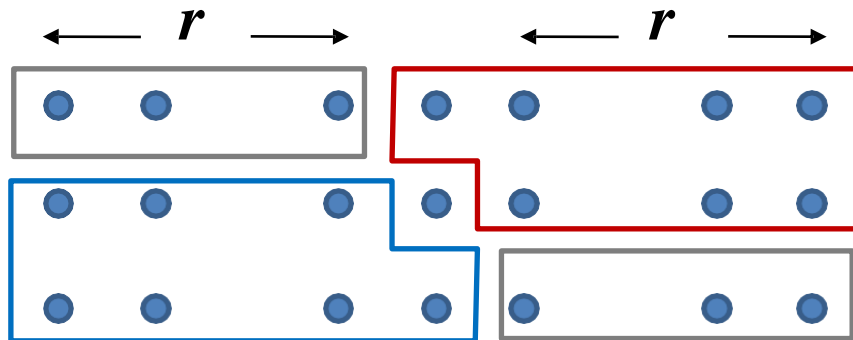
递归树



讨论

分组时为什么5个元素一组？3个一组或7个一组是否可行？

假设 $t=3$ ，3个一组：



$$n = 3(2r + 1)$$

$$r = (n/3 - 1)/2 = n/6 - 1/2$$

子问题规模最多为

$$4r + 1 = 4n/6 - 1$$

算法的时间复杂度满足方程 $W(n) = W(n/3) + W(4n/6) + cn$

由递归树得 $W(n) = \Theta(n \log n)$

关键：递归树每行的工作量构成公比小于 1 的等比级数，
算法复杂度才是 $O(n)$ 。

2.4.3 $n-1$ 次多项式在全体 $2n$ 次方根上的求值

$$\text{欧拉公式 } e^{ix} = \cos x + i \sin x$$

1的 $2n$ 次根

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

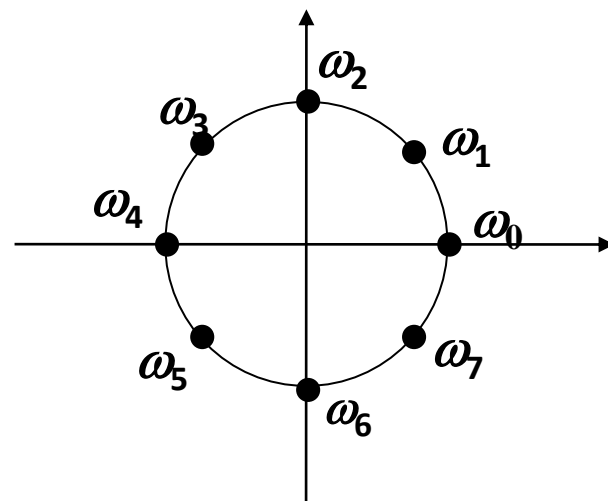
例如 $n=4$, 1的8次方根是:

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

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

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

$$\omega_6 = e^{\frac{6\pi i}{4}} = -i, \quad \omega_7 = e^{\frac{7\pi i}{4}} = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i$$



多项式求值

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

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 $a_i x^i$, 对 i 求和得到 $A(x)$,

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

算法2: $A_1(x) = a_{n-1}$

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

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

...

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

对每个 x 按照上述顺序求值

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

分治算法

原理:

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

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

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

算法3:

1. 计算1 的所有的 $2n$ 次根
2. 分别计算 $A_{\text{even}}(x^2)$ 与 $A_{\text{odd}}(x^2)$
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

$$T_1(n) = 2T_1(n/2) + g(n), \quad g(n) = O(n),$$

$$T_1(1) = O(1)$$

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

第二章小结

2.1 分治策略的基本思想

2.1.1 两个熟悉的例子（二分检索、二分归并排序）

2.1.2 分治算法的一般性描述

2.2 分治算法的分析技术

• 两种形式： $W(n)=2W(n-1)+1$ ； $W(n)=W(n/2)+1$

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

2.3.2 利用预处理减少递归内部的计算量

2.4 典型实例

2.4.1 快速排序算法

2.4.2 选择问题

2.4.3 $n-1$ 次多项式在全体 $2n$ 次方根上的求值

注意：

- 子问题必须与原问题的类型一样，且相互独立
- 一般尽量均匀划分、递归处理
- 最常见的是二分法