algorithm

4.b.分治策略

本节内容

- 分治算法改进
  - 4.6 改进分治算法的途径1：减少子问题数
  - 4.7 改进分治算法的途径2：增加预处理
- 分治算法实例：
  - 4.8 选最大与最小
  - 4.9 选第二大
  - 4.10 一般选择问题的算法设计
  - 4.11 选择问题的算法分析

改进分治算法的途径1:减少子问题数

- 减少子问题个数的依据
- 分治算法的时间复杂度方程

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

a：子问题数，n/b：子问题规模，$d(n)$：划分与综合工作量

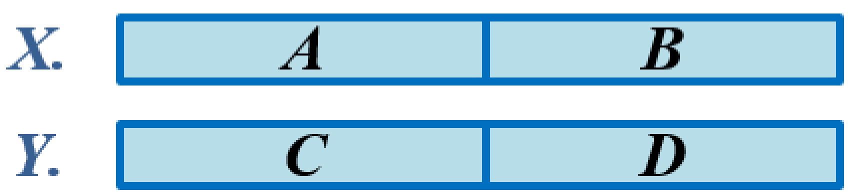
- 当 a 较大, b较小, $d(n)$不大时,方程的解：

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

  - 减少a是降低函数$W(n)$的阶的途径
- 利用子问题的依赖关系，使某些子问题的解通过组合其他子问题的解而得到

例1：整数位乘问题

- 输入：X ,Y 是 n 位二进制数， $n = 2^k$
- 输出：XY
- 普通乘法：需要$O(n^2)$次位乘运算
- 简单划分：令
  - $X = A2^{n/2} + B$, $Y = C2^{n/2} + D$.
  - $XY = AC\ 2^n + (AD + BC)\ 2^{n/2} + BD$



- $W(n) = 4W(n/2)+O(n) \Rightarrow W(n)=O(n^2)$

减少子问题个数

- 子问题间的依赖关系：代数变换

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

- 算法复杂度

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

- 方程的解

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

例2：矩阵相乘问题

- 输入：A, B 为 n 阶矩阵，$\boldsymbol{n} = 2^k$
- 输出：$\boldsymbol{C} = \boldsymbol{AB}$

- 通常矩阵乘法：
  - C 中有 n^2个元素
  - 每个元素需要做 n 次乘法 以元素相乘为基本运算

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

简单分治算法

- 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

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

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

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：
- 设计 $M_1, M_2, \ldots, M_7$，对应7个子问题

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

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

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

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

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

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

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

Strassen 矩阵乘法

- 利用中间矩阵，得到结果矩阵

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

$$C_{12} = M_1 + M_2$$

$$C_{21} = M_3 + M_4$$

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

- 时间复杂度函数：

$$W(n) = 7\,W(n/2) + 18(n/2)^2$$

$$W(1) = 1$$

- 解 $W(n) = O(n^{\log 7}) = O(n^{2.8075})$

矩阵乘法的研究及应用

- 矩阵乘法问题的难度:
- 目前为止最好的上界：Coppersmith–Winograd算法：$O(n^{2.376})$
- 目前为止最好的下界：$\Omega(n^2)$
- 应用:
- 科学计算、图像处理、数据挖掘等
- 回归、聚类、主成分分析、决策树等挖掘算法常涉及大规模矩阵运算

改进途径小结

- 适用于：子问题个数多，划分和综合工作量不太大，时间复杂度函数

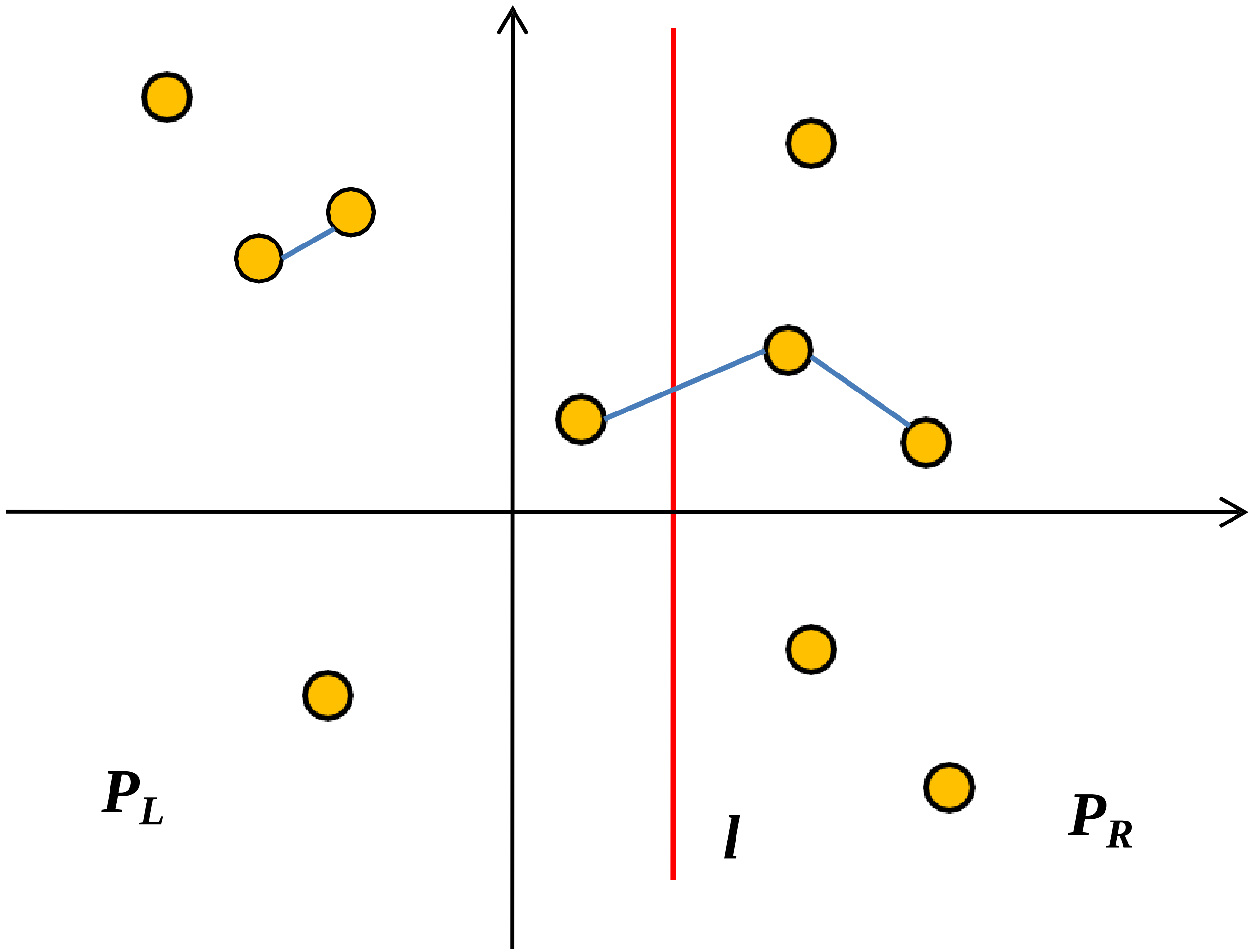
$$W(n) = \Theta(n^{\log_b a})$$

- 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

改进分治算法的途径2：增加预处理

- 例子：平面点对问题
- 输入：平面点集 P 中有n 个点, $n > 1$
- 输出：P中的两个点，其距离最小
- 蛮力算法：$C(n,2)$个点对,计算最小距离，$O(n^2)$
- 分治策略：P 划为大小相等的P_L和P_R
  - 分别计算 P_L、P_R中最近点对
  - 计算 P_L与P_R中各一个点的最近点对
  - 上述情况下的最近点对是解

划分实例：n=10



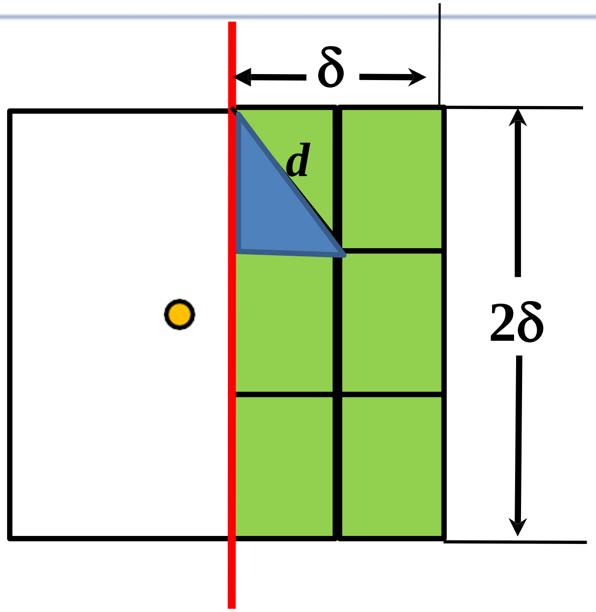
算法伪码

- MinDistance (P, X, Y)
- 输入: 点集P, X和Y为横、纵坐标数组
- 输出: 最近的两个点及距离

1. 若$|P|\leq 3$，直接计算其最小距离
2. 排序X,Y
3. 做中垂线 l 将P划分为P_L和P_R
4. MinDidtance (P_L,X_L,Y_L)
5. MinDistance (P_R,X_R,Y_R)
6. $\delta=\min(\delta_L,\delta_R)$//$\delta_L$,$\delta_R$为子问题的距离
7. 检查距 l 不超过δ两侧各1个点的距离．若小于δ，修改δ为这个值

跨边界处理

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



- 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，
- 检查1个点是常数时间，$O(n)$ 个点需要$O(n)$时间

算法分析

- 步1 递归边界处理：　$\boldsymbol{O(1)}$
- 步2 排序：　$\boldsymbol{O(n\log n)}$
- 步3 划分：　$\boldsymbol{O(1)}$
- 步4-5子问题：　$\boldsymbol{2T(n/2)}$
- 步6确定δ：　$\boldsymbol{O(1)}$
- 步7检查跨边界点对：　$\boldsymbol{O(n)}$

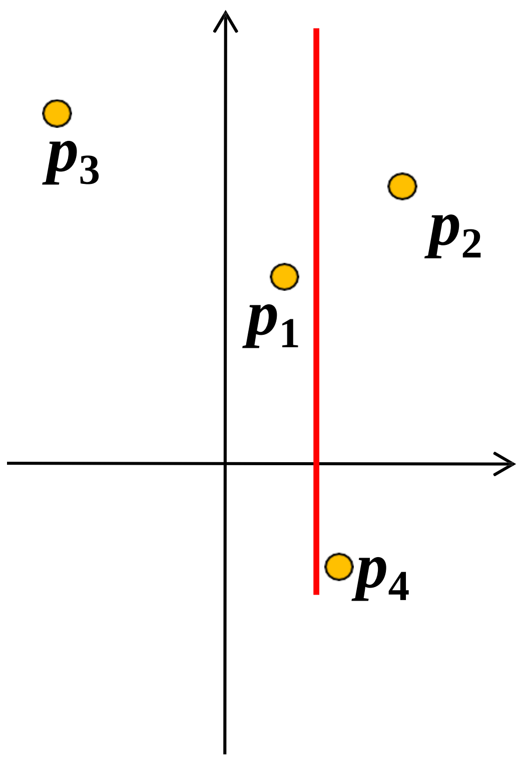
  $T(n)=2T(n/2)+O(n\log n)$

  $T(n)=O(1),\ n\leq 3$
- 递归树求解　$T(n)=O(n\log^2 n)$

增加预处理

- 原算法：
- 在每次划分时对子问题数组重新排序
- 改进算法：
  - 在递归前对 X,Y 排序，作为预处理
  - 划分时对排序的数组 X,Y 进行拆分，得到针对子问题 P_L的数组 X_L,Y_L及针对子问题 P_R的数组 X_R,Y_R
- 原问题规模为n，拆分的时间为$O(n)$

实例：递归中的拆分



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

输入

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

预处理

X_L	2(3)	0.5(1)
Y_L	2(1)	4(3)

X_R	1(4)	2(2)
Y_R	-1(4)	3(2)

拆分后

改进算法时间复杂度

- $W(n)$为算法时间复杂度
- 递归过程：$T(n)$，预处理：$O(n\log n)$

$$W(n) = T(n) + O(n\log n)$$
$$T(n) = 2T(n/2) + O(n)$$
$$T(n) = O(1) \qquad n \le 3$$

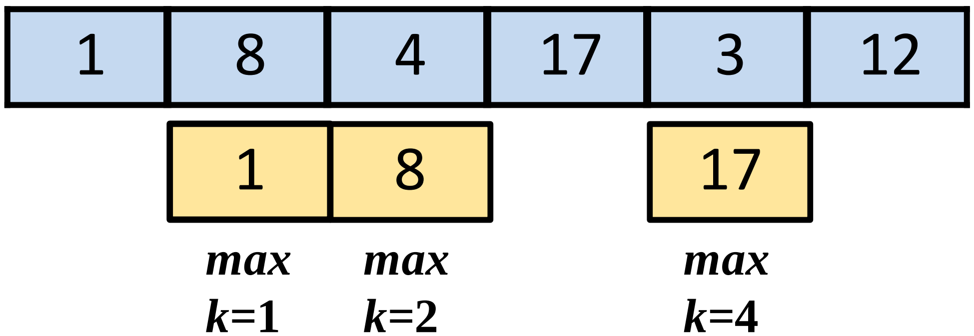
- 解得 $T(n) = O(n\log n)$
- 于是 $W(n) = O(n\log n)$

选最大与最小

- 选择问题
- 输入：集合 L (含n个不等的实数)
- 输出：L中第 i 小元素
- 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
- n为奇数，中位数唯一，$i = (n+1)/2$
- n为偶数，可指定 $i = n/2+1$

选最大

- 算法：顺序比较



- 输出：$max = 17$，$k=4$
- 算法最坏情况下的时间$W(n)=n-1$

伪码

- 算法 Findmax
- 输入：n 个数的数组 L
- 输出：max, k

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. $k \leftarrow i$
6. **return** max, k

选最大最小

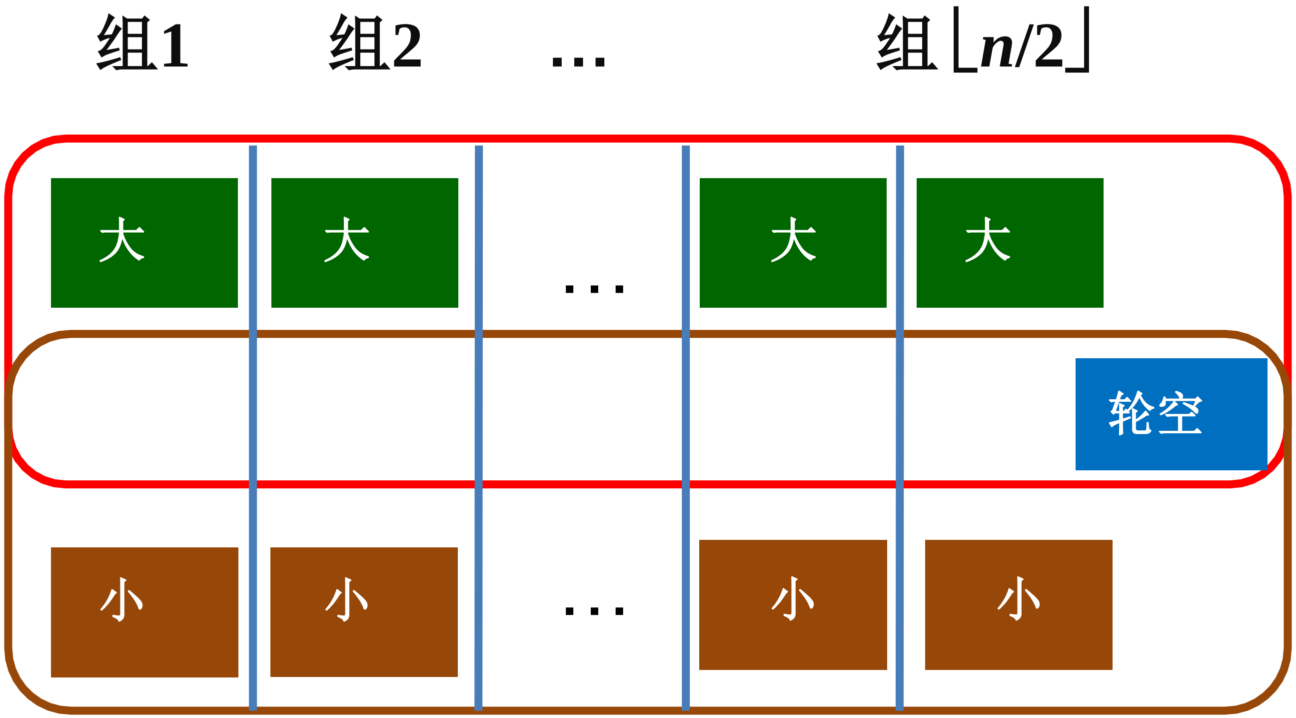
- 通常算法:

1. 顺序比较，先选最大 max
2. 顺序比较，在剩余数组中选最小 min，类似于选最大算法，但比较时保留较小的数

- 时间复杂性:

$$W(n) = n\text{-}1 + n\text{-}2 = 2n–3$$

分组算法



伪码

- 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$\lceil n/2 \rceil$个较大（含轮空元素）中找最大 max
- 4．在$\lceil n/2 \rceil$个较小（含轮空元素）中找最小 min

最坏情况时间复杂度

- 行2 的组内比较：$\lfloor n/2 \rfloor$ 次
- 行3--4 求 max 和 min 比较：
- 至多 $2\lceil n/2 \rceil - 2$ 次

$$
\begin{aligned}
W(n) &= \lfloor n/2 \rfloor + 2\lceil n/2 \rceil - 2 \\
&= n + \lceil n/2 \rceil - 2 \\
&= \lceil 3n/2 \rceil - 2
\end{aligned}
$$

分治算法

1. 将数组 L 从中间划分为两个 子数组 L_1 和 L_2
2. 递归地在 L_1 中求最大 max_1 和 min_1
3. 递归地在 L_2 中求最大 max_2 和 min_2
4. $max \leftarrow \max\{max_1, max_2\}$
5. $min \leftarrow \min\{min_1, min_2\}$

最坏情况时间复杂度

假设 $n = 2^k$,

$$W(n) = 2W(n/2) + 2$$

$$W(2) = 1$$

解 $W(2^k) = 2W(2^{k-1}) + 2$

$= 2[2W(2^{k-2}) + 2] + 2$

$= 2^2W(2^{k-2}) + 2^2+2 = \dots$

$= 2^{k-1} + 2^{k-1} + \dots + 2^2 + 2$

$= 3 \cdot 2^{k-1} – 2 = 3n/2 - 2$

选择算法小结

- 选最大：顺序比较, 比较次数 n-1
- 选最大最小
- 选最大+ 选最小, 比较次数 2n-3
- 分组： 比较次数 $\lceil 3n/2 \rceil - 2$
- 分治：$n=2^k$，比较次数 3n/2-2

选第二大

- 输入：n个数的数组 L
- 输出：第二大的数 $second$
- 通常算法：顺序比较

1．顺序比较找到最大 max

2．从剩下 $n-1$个数中找最大，就是第二大$second$

- 时间复杂度：

$$W(n) = n-1 + n-2 = 2n-3$$

提高效率的途径

- 成为第二大数的条件：仅在与最大 数的比较中被淘汰
- 要确定第二大数，必须知道最大数
- 在确定最大数的过程中记录下被最大数直接淘汰的元素
- 在上述范围（被最大数直接淘汰的数）内的最大数就是第二大数
- 设计思想： 用空间换时间

锦标赛算法

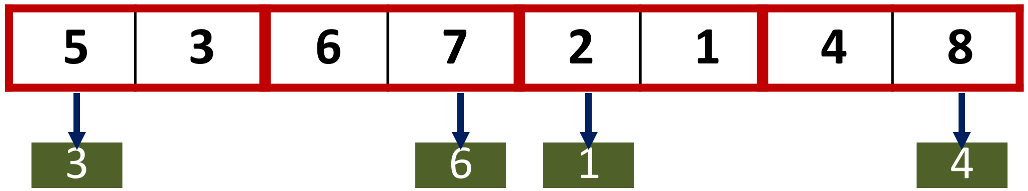
- 两两分组比较，大者进入下一轮，直到剩下 1个元素 max 为止
- 在每次比较中淘汰较小元素，将被淘汰元素记录在淘汰它的元素的链表上
- 检查 max 的链表，从中找到最大元，即$second$

伪码

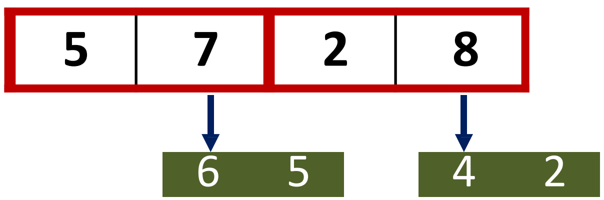
- 算法 FindSecond
- 输入: n个数的数组 L, 输出: *second*
- 1． $k \leftarrow n$ // 参与淘汰的元素数
- 2． 将k个元素两两1组, 分成$\lfloor k/2 \rfloor$组
- 3． 每组的2个数比较，找到较大数
- 4． 将被淘汰数记入较大数的链表
- 5． if k 为奇数 then $k \leftarrow \lfloor k/2 \rfloor +1$
- 6． else $k \leftarrow \lfloor k/2 \rfloor$
- 7． if $k>1$ then goto 2
- 8． $max \leftarrow$最大数
- 9． $second \leftarrow max$ 的链表中的最大

实例

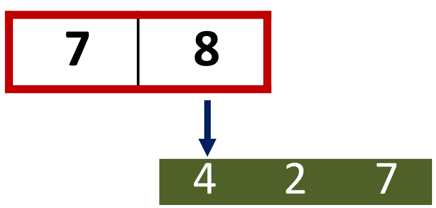
- 分组1



- 分组2



- 分组3



时间复杂度分析

- 命题1：设参与比较的有 t 个元素，经过 i 轮淘汰后元素数至多为 $\lceil t/2^i \rceil$
- 证 对 i 归纳.
  - $i=1$,分 $\lfloor t/2 \rfloor$ 组，淘汰 $\lfloor t/2 \rfloor$ 个元素，进入下一轮元素数是

    $$t - \lfloor t/2 \rfloor = \lceil t/2 \rceil$$
  - 假设 i 轮分组淘汰后元素数至多为 $\lceil t/2^i \rceil$,
  - 那么 $i+1$ 轮分组淘汰后元素数为

    $$\lceil \lceil t/2^i \rceil /2 \rceil = \lceil t/2^{i+1} \rceil$$

时间复杂度分析

- 命题2 max 在第一阶段分组比较中总计进行了$\lceil \log n \rceil$次比较.
- 证： 假设到产生 max 时总计进行k 轮淘汰, 根据命题 1有

$\lceil n/2^k \rceil=1$

- 若 $n=2^d$，那么有

$$k = d = \log n = \lceil \log n \rceil$$

- 若 $2^d < n < 2^{d+1}$， 那么

$$k = d + 1 = \lceil \log n \rceil$$

第一阶段元素数：n, 比较次数: $n–1$，淘汰了 $n–1$个元素

第二阶段：元素数$\lceil \log n \rceil$

比较次数:$\lceil \log n \rceil–1$，淘汰元素数为$\lceil \log n \rceil–1$

时间复杂度:

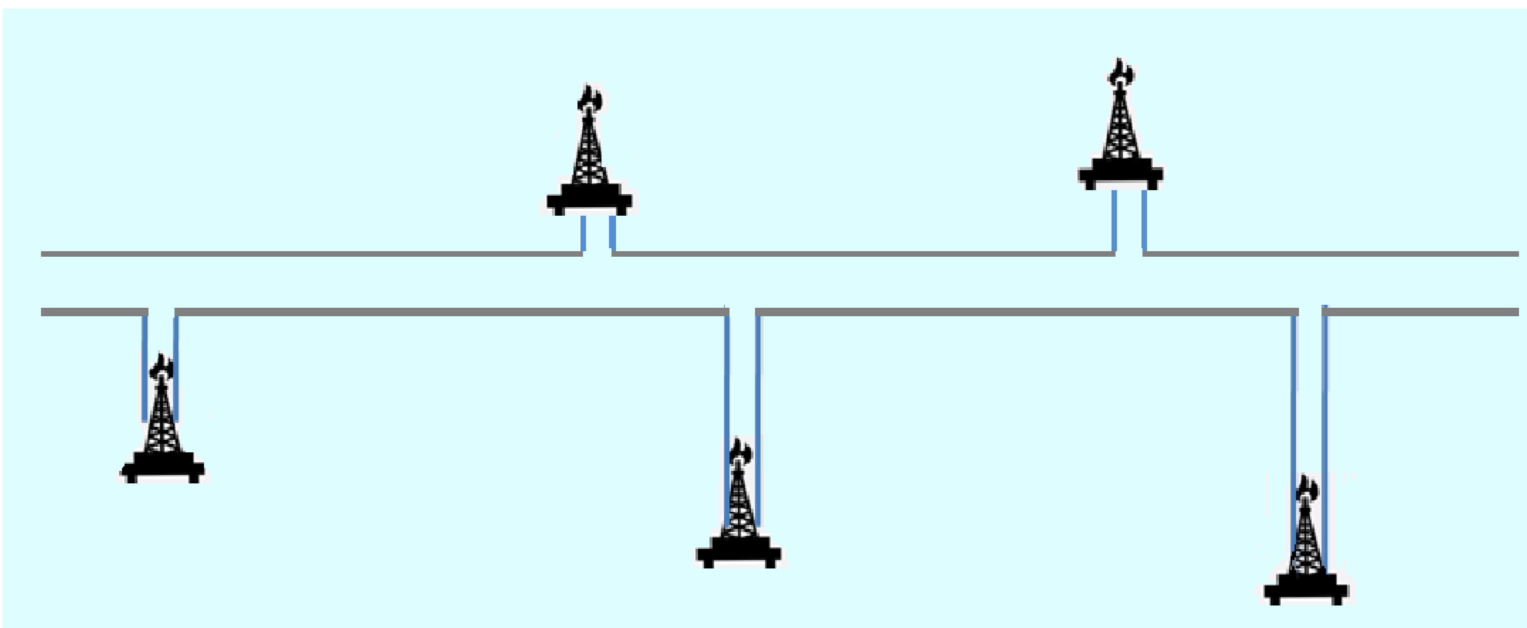
$$W(n) = n –1 + \lceil \log n \rceil –1 = n + \lceil \log n \rceil –2$$

一般选择问题的算法设计

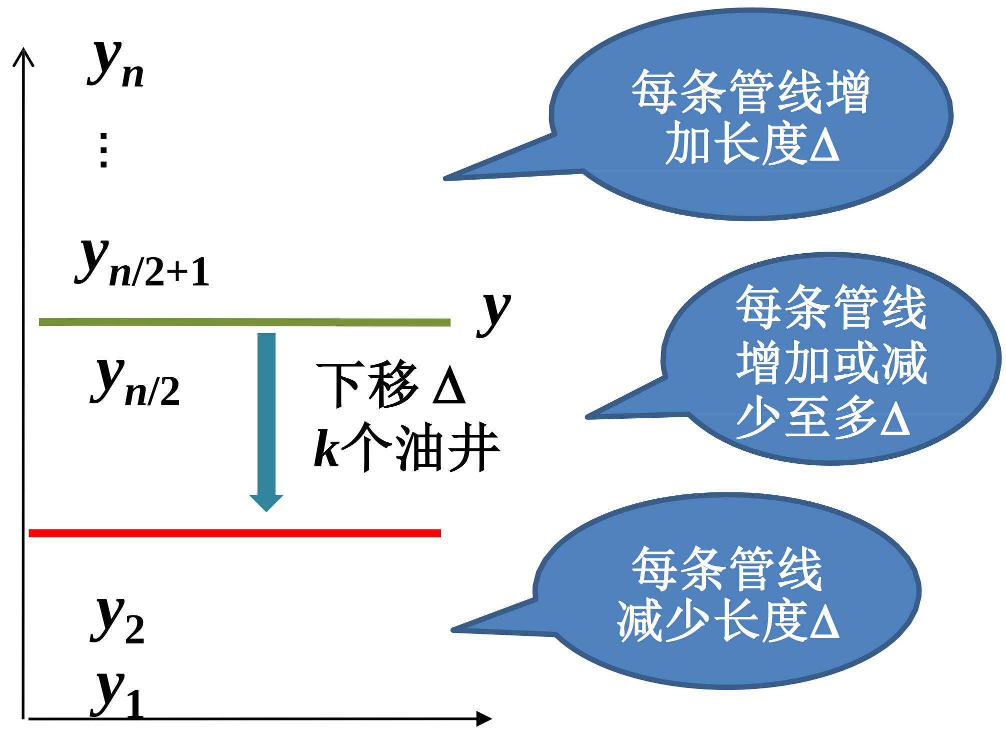
- 问题：选第 k 小.
- 输入：数组 S, S 的长度 n, 正整数 k, $1\leq k\leq n$.
- 输出：第 k 小的数
- 实例 1
  - $S=\{3,4,8,2,5,9,18\}$，$k=4$，解：5
- 实例 2
- 统计数据的集合S，$|S|=n$，
- 选中位数，$k=\lceil n/2\rceil$

一个应用：管道位置

- 问题：某区域有n口油井，需要修建输油管道. 根据设计要求，水平方向有一条主管道，每口油井修一条垂直方向的支管道通向主管道
- 如何选择主管道的位置，以使得支管道长度的总和最小？



最优解：Y坐标的中位数



y_n
⋮
$y_{n/2+1}$
y
$y_{n/2}$
下移 Δ
k个油井
y_2
y_1
每条管线增加长度Δ
每条管线增加或减少至多Δ
每条管线减少长度Δ

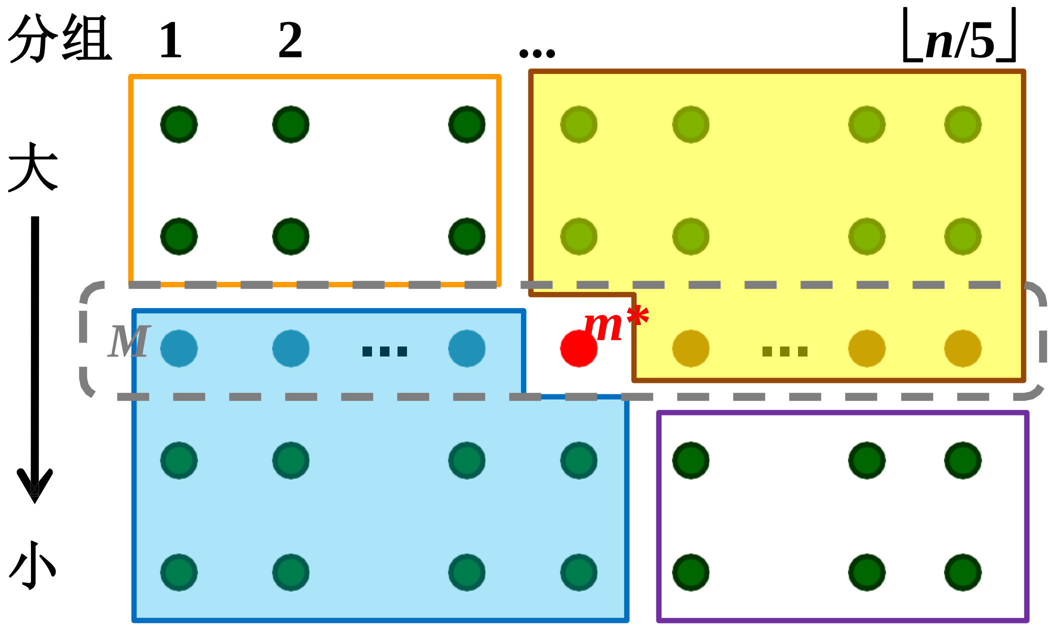
简单的算法

- 算法一：
  - 调用 k 次选最小算法 时间复杂度为 $O\ (k\ n)$
- 算法二：
  - 先排序，然后输出第 k 小的数
  - 时间复杂度为 $O(n\ \log n)$

分治算法

- 假设元素彼此不等，设计思想：
- 用某个元素 m^*作为标准将 S 划分 成 S_1 与 S_2，其中 S_1的元素小于 m^*, S_2 的元素大于等于 m^*.
- 如果 $k \leq |S_1|$，则在 S_1中找第 k 小. 如果 $k = |S_1|+1$，则 m^*是第 k 小 如果 $k > |S_1|+1$，则在 S_2中找第 $k-|S_1| - 1$小

m^*的选择与划分过程



A：数需要与m^*比大小，
B：数大于m^*
C：数小于m^*，
D：数需要与m^*比大小

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	B
	5	9	12	
C	3	6	10	D
	2	1	4	

8, 7, 10, 4 需要与9比较

归约为子问题

	8	14	15	
	7	11	13	
S_1	5	9	12	S_2
	3	6	10	
	2	1	4	

子问题

8 7 5 3 2 6 1 4

子问题规模 = 8， k=6

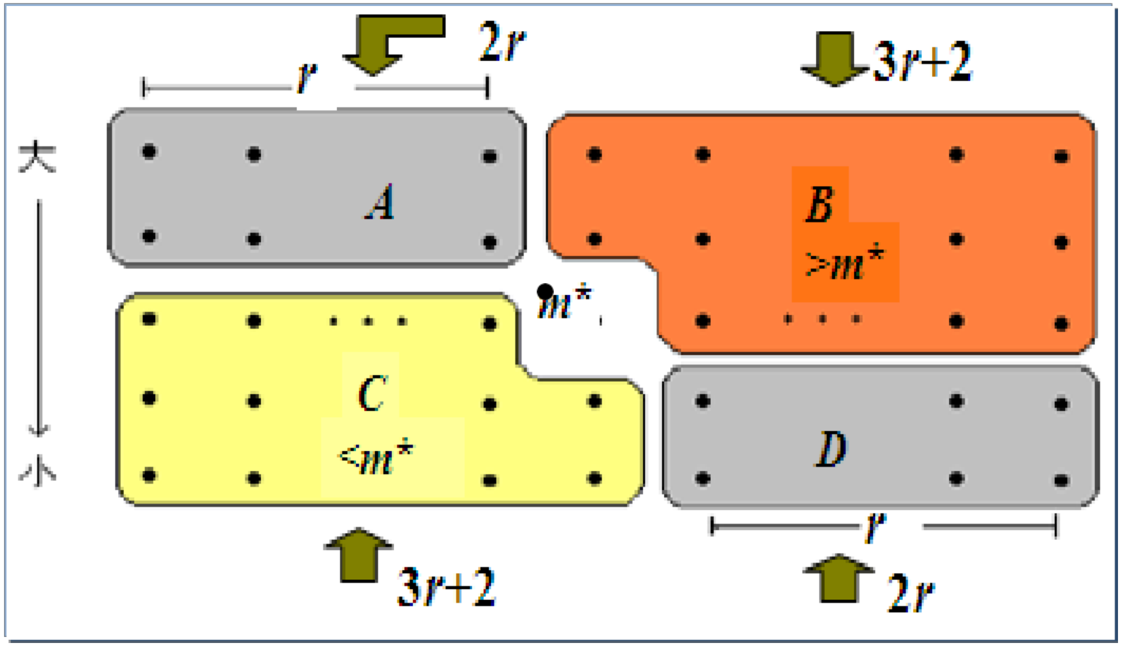
伪码

- 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分5个一组, 共 $n_M = \lceil n/5 \rceil$组
2．每组排序,中位数放到集合 M
3．$m^* \leftarrow \text{Select}(M, \lceil |M|/2 \rceil)$ //S分A,B,C,D
4．A,D元素小于m^*放 S_1,大于m^*放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 $(S_2, k - |S_1| - 1)$

选择问题的算法分析

- 用m^*划分



$n = 5\ (2r + 1)$, $|A| = |D| = 2r$

- 子问题规模至多： $2r+2r+3r+2 = 7r+2$

子问题规模估计

- 不妨设 $n = 5(2r+1)$，$|A|=|D|=2r$,

$$r = \frac{n/5-1}{2} = \frac{n}{10} - \frac{1}{2}$$

- 划分后子问题规模至多为

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

$$= \frac{7n}{10} - \frac{3}{2} < \frac{7n}{10}$$

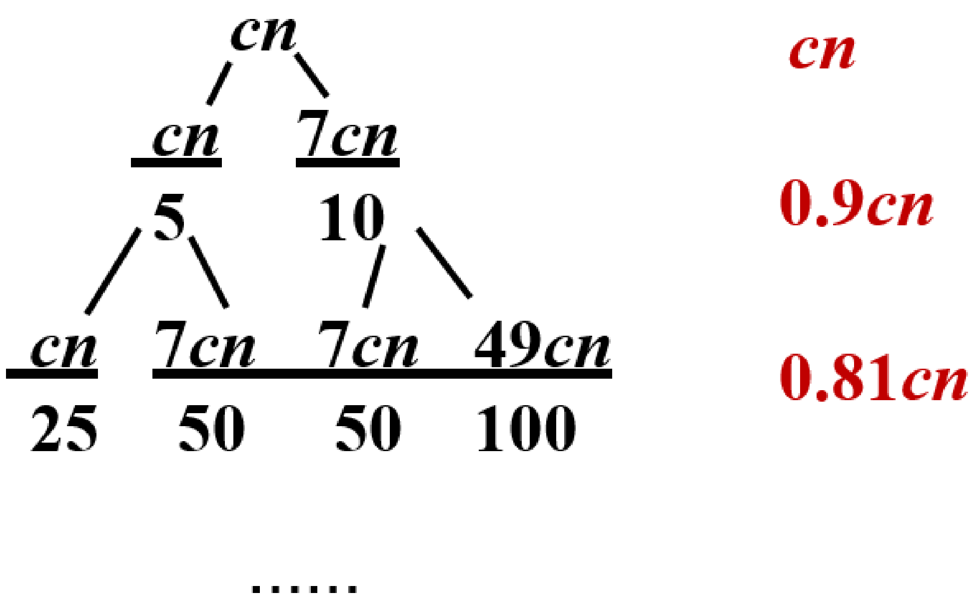
时间复杂度递推方程

- 算法工作量 $W(n)$
- 行2: $O(n)$//每5个数找中位数,构成M
- 行3: $W(n/5)$// M 中找中位数 m^*
- 行4: $O(n)$ // 用m^*划分集合 S
- 行8-9：$W(7n/10)$//递归

$$W(n) \leq W(n/5) + W(7n/10) + O(n)$$

递归树

- $W(n)=W(n/5)+W(7n/10)+cn$



- $W(n) \leq cn\ (1+0.9+0.9^2+\ldots)=O(n)$

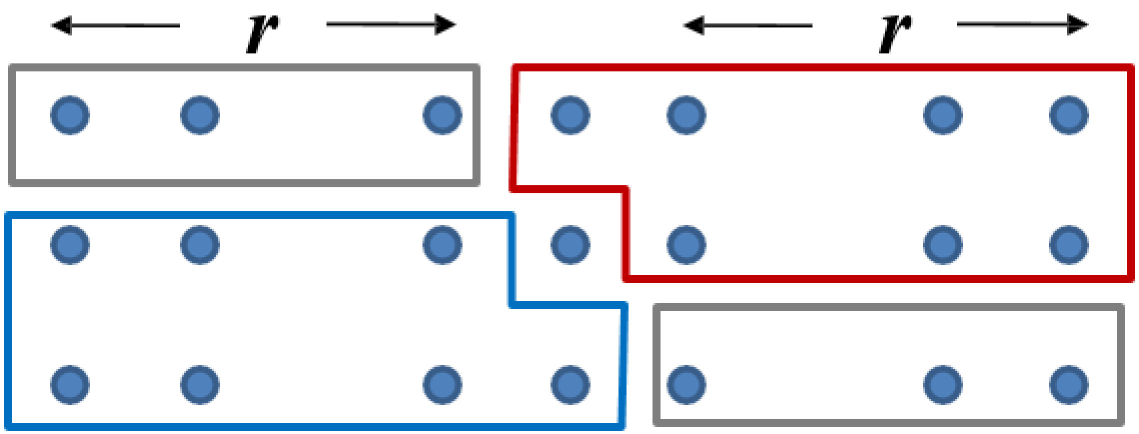
思考

- 分组时为什么5个元素一组?
- 3个一组或 7个一组行不行?
- 分析：递归调用

1. 求 m^*的工作量与 $|M| = n/t$ 相关，t 为每组元素数. t大, $|M|$小
2. 归约后子问题大小与分组元素数t 有关 . t 大，子问题规模大

3分组时的子问题规模

- 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

- 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:
- $|M|$与归约后子问题规模之和小于 n,
- 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$O(n)$.