# 《计算机算法设计与分析》

## 第三章 分治法

马丙鹏

2020年09月29日

中国科学院大学
University of Chinese Academy of Sciences

# 第三章 分治法

- 3.1 一般方法
- 3.2 二分检索
- 3.3 找最大和最小元素
- 3.4 归并排序
- 3.5 快速排序
- 3.6 选择问题
- 3.7 斯特拉森矩阵乘法

# 3.6 选择问题

- 问题描述
  - 给出含有n个元素表A(1:n)，确定其中的第k小元素。
- 设计思路
  - 直接方法
    - 先排序，后查找。
    - 排序后第k位的元素即为待查找的元素。
    - 时间复杂度O(nlogn)。

# 3.6 选择问题

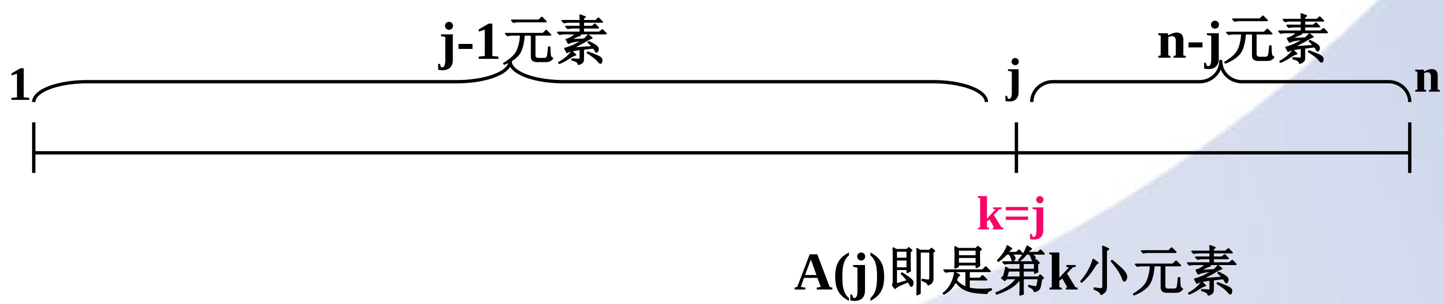
## ■ 设计思路

### □利用PARTITION过程

- 第一次划分后，划分元素v测定在A(j)的位置上，则有j-1个元素小于或等于A(j)，且有n-j个元素大于或等于A(j)。

① 若k=j，则A(j)即是第k小元素；

情况1:



# 3.6 选择问题

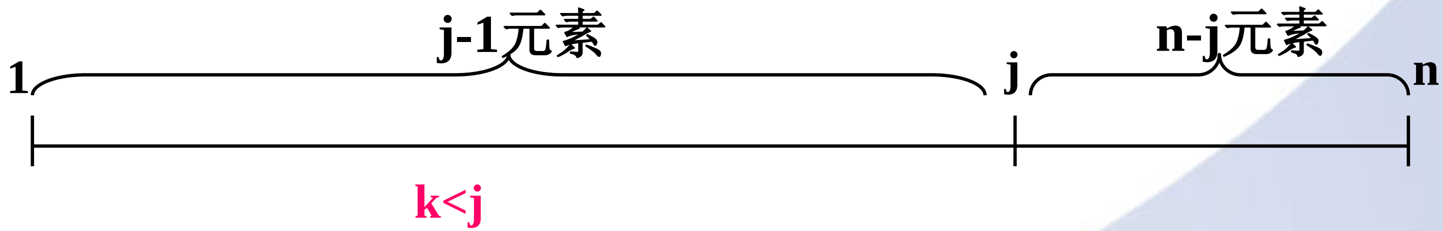
■ 设计思路

□利用PARTITION过程

➢第一次划分后，划分元素v测定在A(j)的位置上，则有j-1个元素小于或等于A(j)，且有n-j个元素大于或等于A(j)。

② 若k<j，则第k小元素将出现在A(1:j-1)中；

情况2:




第k小元素在A(1:j-1)中，

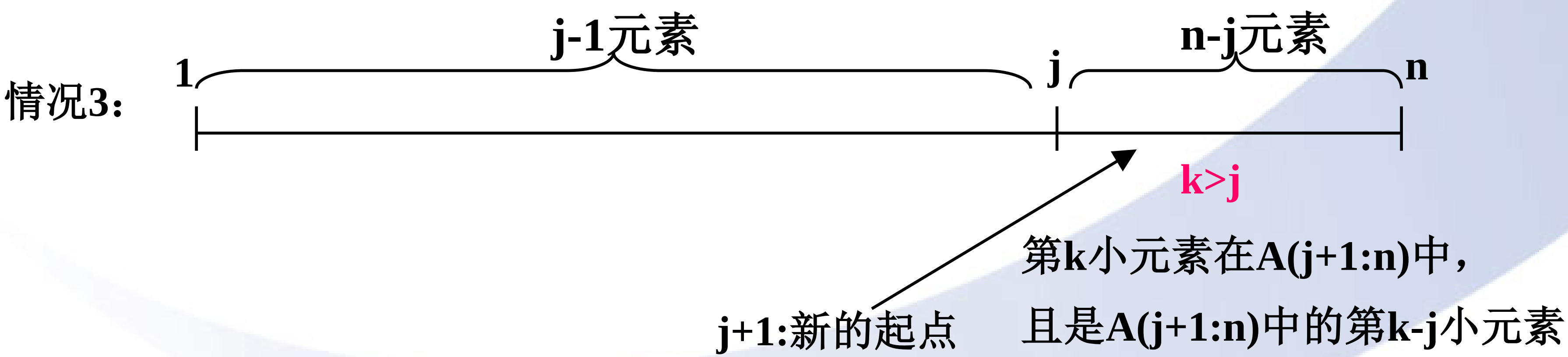
且是A(1:j-1)中的第k小元素

# 3.6 选择问题

## ■ 设计思路

- 利用PARTITION过程
  - 第一次划分后，划分元素v测定在A(j)的位置上，则有j-1个元素小于或等于A(j)，且有n-j个元素大于或等于A(j)。

③ 若k>j，则第k小元素将出现在A(j+1:n)中。



# 3.6 选择问题

## ■ 算法实现

算法3.15 找第k小元素

```
procedure SELECT(A, n, k)  //在数组A(1:n)中找第k小元素，并将之放在A(k)中。//
   integer n, k, m, r, j;
   m←1; r ←n+1; A(n+1) ←+∞  //A(n+1)被定义，并置为一大值，用于限界//
   loop  //在进入循环时，1≤m≤k≤r≤n+1 //
      j ←r  //将剩余元素的最大下标加1后置给j //
      call PARTITION(m, j)  //返回j,它使得A(j)是第j小的值//
      case
        :k=j:  return
        :k<j:  r←j      //j是新的上界//
        :else: m←j+1  //k>j, j+1是新的下界//
      endcase
   repeat
end SELECT
```

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| A[i] | 65 | 70 | 75 | 80 | 85 | 60 | 55 | 50 | 45 |

- K=7

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 65 | 70 | 75 | 80 | 85 | 60 | 55 | 50 | 45 |
| 2 | 60 | 45 | 50 | 55 | 65 | 85 | 80 | 75 | 70 |
| 3 | 60 | 45 | 50 | 55 | 65 | 70 | 80 | 75 | 85 |
| 4 | 60 | 45 | 50 | 55 | 65 | 70 | 80 | 75 | 85 |
| 5 | 60 | 45 | 50 | 55 | 65 | 70 | 75 | 80 | 85 |
| 6 | 60 | 45 | 50 | 55 | 65 | 70 | 75 | 80 | 85 |

以上过程分别执行了**PARTITION(1, 10), PARTITION(6, 10), PARTITION(6, 9), PARTITION(7, 9), PARTITION(7, 8)**

# 3.6 选择问题

- 算法分析
  - 两点假设
    - A中的元素互异。
    - 随机选取划分元素，且选择A中任一元素作为划分元素的概率相同。
  - 分析
    - 每次调用PARTITION(m, j)，所需的元素比较次数是O(j-m+1)。
    - 在执行一次PARTITION后，或者找到第k小元素，或者将在缩小的子集(A(m, k-1)或A(k+1, j))中继续查找。缩小的子集的元素数将至少比上一次划分的元素数少1。

# 3.6 选择问题

## ■ 算法分析

### □最坏情况

- ➢SELECT的最坏情况时间是$O(n^2)$
- ➢最坏情况特例:
  - ✓输入A(1:n)恰好使对PARTITION的第i次调用选用的划分元素是第i小元素，而k=n。

| i | 0 | 1 | 2 | 3 | 4 | … | n-1 | n | n+1 |
|---|---|---|---|---|---|---|---|---|---|
| a[i] | -- | 1 | 2 | 3 | … | n-2 | n-1 | n | ∞ |

# 3.6 选择问题

## ■ 算法分析

### □最坏情况

- ➢SELECT的最坏情况时间是$O(n^2)$
- ➢最坏情况特例:
  - ✓此时(区间下界)m随着PARTITION的每一次调用而仅增加1，j保持不变。PARTITION最终需要调用n次。
  - ✓则n次调用的时间总量是

$$O\left(\sum_1^n (i+1)\right) = O(n^2)$$

中国科学院大学
University of Chinese Academy of Sciences

# 3.6 选择问题

对n个不同的元素，问题实例可能的n!种不同情况，综合考查所得的平均值

某个特定的k

- 算法分析
  - 平均情况
    - 设$T_A^k(n)$是找A(1:n)中第k小元素的平均时间。$T_A(n)$是SELECT的平均计算时间，则有

在所有可能的情况下，找所有可能的k小元素

$$T_A(n) = \frac{1}{n}\sum_{1\le k\le n} T_A^k(n)$$

并定义

$$R(n) = \max_k \{T_A^k(n)\}$$

有：T(n)≤R(n)。

# 3.6 选择问题

- 算法分析
  - 定理3.4 SELECT的平均计算时间$T_A(n)$是O(n)
    (对比快速排序平均计算时间O(nlogn))
  - 证明: (课下阅读)

PARTITION和SELECT中，case语句的执行时间是O(n)。在随机等概率选择划分元素时，首次调用PARTITION中划分元素v刚好是A中第i小元素的概率为1/n，1≤i≤n。

则，存在正常数c，c>0,有，

$$T_A^k(n) \le cn + \frac{1}{n}\left(\sum_{1\le i<k} T_A^{k-i}(n-i) + \sum_{k<i\le n} T_A^k(i-1)\right) \quad n\ge 2$$

划分元素i<k,将在i的后半部分求解

划分元素i>k,将在i的前半部分求解

# 3.6 选择问题

## ■ 算法分析

### □证明:

且有，

$$R(n) \le cn + \tfrac{1}{n}\max_{k}\{\sum_{1\le i<k} R(n-i) + \sum_{k<i\le n} R(i-1)\}$$

$$= cn + \tfrac{1}{n}\max_{k}\{\sum_{n-k+1}^{n-1} R(i) + \sum_{k}^{n-1} R(i)\} \quad n\ge 2$$

令**c≥R(1)**。利用数学归纳法证明，对所有**n≥2**，有**R(n)≤4cn.**

①归纳基础 当**n=2**时，由上式得： $R(n) \le 2c + \frac{1}{2}\max\{R(1),R(1)\}$

$$\le 2.5c < 4cn$$

②归纳假设 假设对所有得**n, 2≤n<m**，有**R(n)≤4cn**

# 3.6 选择问题

■ 算法分析

□证明:

③归纳步骤 当n = m时，有，

由于R(n)是n的非降函数，故在当m为偶数而k=m/2，或当m为奇数而k=(m+1)/2时，$\sum_{n-k+1}^{n-1} R(i) + \sum_{k}^{n-1} R(i)$ 取得极大值。因此，

若m为偶数，则 $R(m) \le cm + \frac{2}{m}\sum_{m/2}^{m-1} R(i) \le cm + \frac{8c}{m}\sum_{m/2}^{m-1} i < 4cm$

若m为奇数，则 $R(m) \le cm + \frac{2}{m}\sum_{(m+1)/2}^{m-1} R(i) \le cm + \frac{8c}{m}\sum_{(m+1)/2}^{m-1} i < 4cm$

由于TA(n)≤R(n)，所以TA(n)≤4cn。 故$T_A(n)=O(n)$

# 3.6 选择问题

- 最坏情况是O(n)的选择算法
  - 基本思想：
    - 精心挑选划分元素v
  - 方法：
    - 二次取中间值
  - 目的：
    - 使v比一部分元素小，比另一部分元素大

# 3.6 选择问题

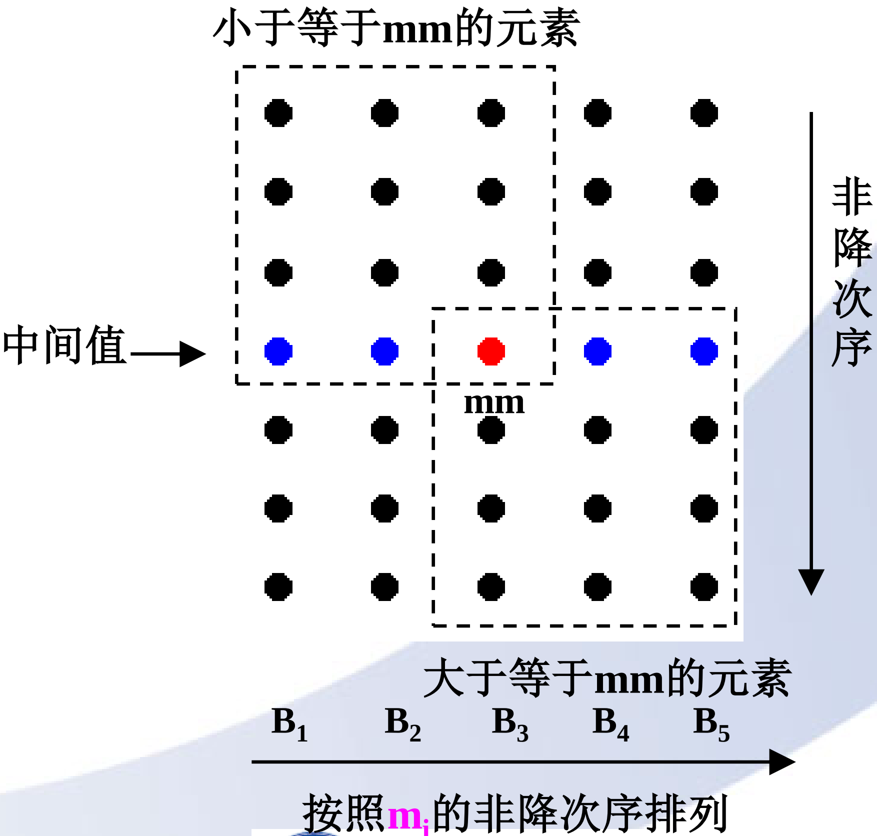
- 最坏情况是O(n)的选择算法
  - 采用两次取中间值的规则精心选取划分元素
    - ① 将参加划分的n个元素分成$\lfloor n/r \rfloor$组，每组有r个元素($r \geq 1$)。(多余的$n - r\lfloor n/r \rfloor$个元素忽略不计)
    - ② 对这$\lfloor n/r \rfloor$组每组的r个元素进行排序并找出其中间元素$m_i$, $1 \leq i \leq \lfloor n/r \rfloor$，共得$\lfloor n/r \rfloor$个中间值。
    - ③ 对这$\lfloor n/r \rfloor$个中间值排序，并找出其中间值mm。
    - ④ 将mm作为划分元素执行划分。

# 3.6 选择问题

■ 最坏情况是O(n)的选择算法

□采用两次取中间值的规则精心选取划分元素

- 例：设 n=35, r=7。
- 分为n/r = 5个元素组：$B_1$，$B_2$，$B_3$，$B_4$，$B_5$；每组有7个元素。
- $B_1$-$B_5$按照各组的$m_i$的非降次序排列。$mm = m_i$的中间值，$1 \le i \le 5$
- 由图所示有：

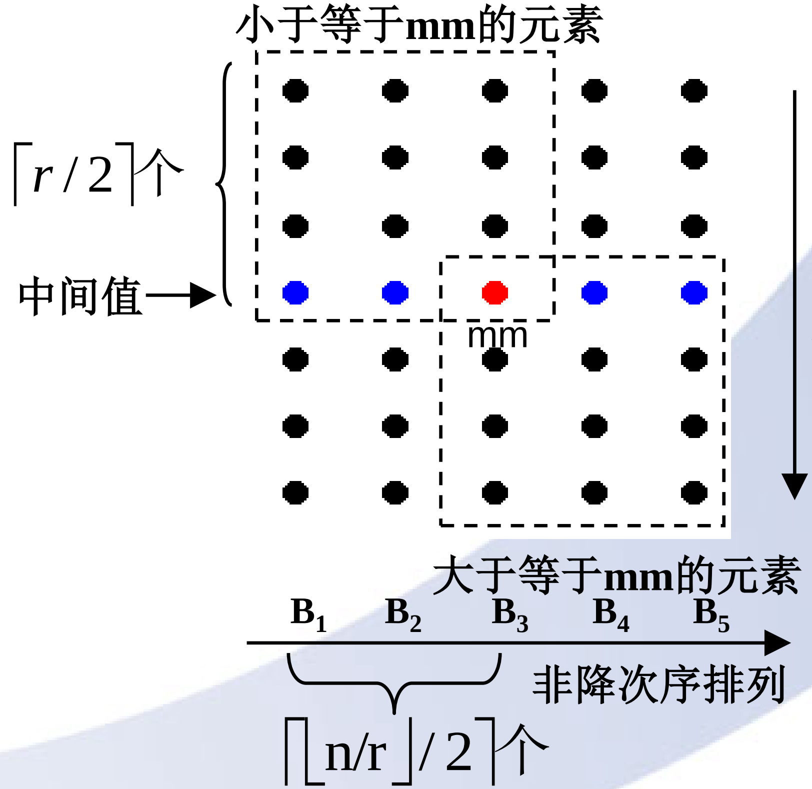


# 3.6 选择问题

■ 最坏情况是O(n)的选择算法

□采用两次取中间值的规则精心选取划分元素

- r个元素的中间值是第 $\lceil r/2 \rceil$小元素；
- 至少有$\lceil \lfloor n/r \rfloor /2 \rceil$个$m_i$小于或等于mm；
- 至少有$\lfloor n/r \rfloor - \lceil \lfloor n/r \rfloor /2 \rceil + 1 \geq \lceil \lfloor n/r \rfloor /2 \rceil$个$m_i$大于或等于mm。
- 故，至少有$\lceil r/2 \rceil \lceil \lfloor n/r \rfloor /2 \rceil$个元素小于或等于(或大于或等于)mm。



# 3.6 选择问题

- 最坏情况是O(n)的选择算法
  - 采用两次取中间值的规则精心选取划分元素
    - 当r=5，则使用两次取中间值规则来选择v=mm
    - 至少有$1.5\lfloor n/5 \rfloor$个元素小于等于划分元素v。

$$\lceil r/2 \rceil \lceil \lfloor n/r \rfloor /2 \rceil = \lceil 5/2 \rceil \lceil \lfloor n/5 \rfloor /2 \rceil \ge 3\lfloor n/5 \rfloor /2$$
$$= 1.5\lfloor n/5 \rfloor$$

    - 至多有 $n - 1.5\lfloor n/5 \rfloor \le 0.7n + 1.2$ 个元素大于等于v。

$$n - 1.5\lfloor n/5 \rfloor \le n - 1.5(n-4)/5 = 0.7n + 1.2$$

注：$\lfloor n/5 \rfloor \ge (n-4)/5$

    - 同理，至多有 0.7n+1.2个元素小于等于v。

# 3.6 选择问题

- ■ 最坏情况是O(n)的选择算法
  - □采用两次取中间值的规则精心选取划分元素
    - ➢故，这样的v可较好地划分A中的n个元素：
    - ➢比足够多的元素大，也比足够多的元素小，
    - ➢不论落在那个区域，总可以在下一步查找前舍去足够多的元素，
    - ➢而在剩下的“较小”范围内继续查找。

# 算法3.16 使用二次取中规则得选择算法的说明性描述

**procedure** SELECT2(A, k, n)　　//在集合A中找第k小元素，使用两次取中规则//

① 若n≤r，则采用插入法直接对A排序并返回第k小元素

② 把A分成大小为r的$\lfloor n/r \rfloor$个子集合，忽略多余的元素

③ 设M={$m_1, m_2, \ldots, m_{\lfloor n/r \rfloor}$}是$\lfloor n/r \rfloor$子集合的中间值集合

④ v←SELECT2(M, $\lceil \lfloor n/r \rfloor /2 \rceil$, $\lfloor n/r \rfloor$)

⑤ j←PARTITION(A, v) //v作为划分元素，划分后j等于划分元素所在位置的下标//

⑥ **case**

　　:k=j: **return**(v)

　　:k<j: 设S是A(1:j-1)中元素的集合

　　　　**return**(SELECT2(S, k, j-1))

　　:**else**: 设R是A(j+1:n)中元素的集合

　　　　**return**(SELECT2(R, k-j, n-j))

　**endcase**

**end** SELECT2

# 3.6 选择问题

- 最坏情况是O(n)的选择算法
  - 算法分析
    - 记T(n)是SELECT2所需的最坏情况时间
      对特定的r分析SELECT2：选取r=5。
    - 假定A中的元素各不相同，则有

# 算法3.16 使用二次取中规则得选择算法的说明性描述

**procedure** SELECT2(A, k, n) //在集合A中找第k小元素，使用两次取中规则//

① 若n≤r，则采用插入法直接对A排序并返回第k小元素 →O(1)

② 把A分成大小为r的$\lfloor n/r \rfloor$个子集合，忽略多余的元素 →O(n)

③ 设$M=\{m_1, m_2, \dots m_{\lfloor n/r \rfloor}\}$是$\lfloor n/r \rfloor$子集合的中间值集合 →O(n)

④ v←SELECT2(M, $\lceil \lfloor n/r \rfloor /2 \rceil$, $\lfloor n/r \rfloor$) →T(n/5)

⑤ j←PARTITION(A, v) →O(n)

⑥ **case** →T(3n/4), n≥24

:k=j: **return**(v)

:k<j: 设S是A(1:j-1)中元素的集合

**return**(SELECT2(S, k, j-1))

:**else**: 设R是A(j+1:n)中元素的集合

**return**(SELECT2(R, k-j, n-j))

**endcase**

**end** SELECT2

中国科学院大学 University of Chinese Academy of Sciences

# 3.6 选择问题

- 最坏情况是O(n)的选择算法
  - 算法分析

  r为定值，记对r个元素的直接排序的时间为“定值”

故有，

$$T(n)=\begin{cases} cn & n<24, \\ T(n/5)+T(3n/4)+cn & n\geq 24 \end{cases}$$

用归纳法可证：

$T(n)\leq 20cn$

故，在r=5的情况下，求解n个不同元素选择问题的算法SELECT2的最坏情况时间是O(n)。

中国科学院大学 University of Chinese Academy of Sciences

# 3.6 选择问题

- 最坏情况是O(n)的选择算法
  - 当A中的元素可能相同
    - 步骤⑤经PARTITION调用所产生的S和R两个子集合中可能存在一些元素等于划分元素v，可能导致|S|或|R|大于0.7n+1.2。影响到算法的效率。

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 2 | 2 |
| 1 | 1 | 2 | 2 | 2 |
| 1 | 1 | 2 | 2 | 2 |

0.7*25+1.2=18.7

# 3.6 选择问题

- 最坏情况是O(n)的选择算法
  - 当A中的元素可能相同
    - 特例：当r=5，且A中的元素不全相同。假设其中有0.7n+1.2个元素比v小而其余的元素都等于v的情况。
      则经过PARTITION，在这些等于v的元素中至多有一半可能在S中，故$|S| \le 0.7n+1.2+(0.3n-1.2)/2=0.85n+0.6$
      同理，$|R| \le 0.85n+0.6$
      可得，步骤④和⑥此时所处理的元素总数将是
      $T(n/5)+T(0.85n+0.6) \approx 1.05n+0.6>n$
      不再是线性关系。故有$T(n) \neq O(n)$

# 3.6 选择问题

- 最坏情况是O(n)的选择算法
  - 改进方法一：将A集合分成3个子集合U，S和R，其中U是A中所有与v相同的元素组成，S是A中所有比v小的元素组成，R则是A中所有比v大的元素组成。
  - 同时步骤⑥更改：

```
case
    :|S|≥k: return(SELECT2(S, k, |S|))
    :|S|+|U|≥k: return(v)
    :else: return(SELECT2(R, k-|S|-|U|, |R|))
endcase
```

  - 从而保证 |S|和|R|≤0.7n+1.2成立，故关于T(n)的分析仍然成立。T(n) = O(n)

中国科学院大学
University of Chinese Academy of Sciences

# 3.6 选择问题

- ■ 最坏情况是O(n)的选择算法
  - □改进方法二：选取其他的r值进行计算：

取r=9。经计算可得，此时将至少有$2.5\lfloor n/9\rfloor$个元素小于或等于v，同时至少有同样多的元素大于或等于v。

$$\lceil r/2\rceil\ \lceil\lfloor n/r\rfloor/2\rceil = \lceil 9/2\rceil\ \lceil\lfloor n/9\rfloor/2\rceil \ge 5\lfloor n/9\rfloor/2 = 2.5\lfloor n/9\rfloor$$

则当$n\ge 90$时，$|S|$和$|R|$都至多为

相等元素的一半

$$n - 2.5\lfloor n/9\rfloor + \boxed{\frac{1}{2}(2.5\lfloor n/9\rfloor)} = n - 1.25\lfloor n/9\rfloor$$

$$\le 31n/36 + 1.25 \le 63n/72$$

基于上述分析，有新的递推式：

# 3.6 选择问题

■ 最坏情况是O(n)的选择算法

□改进：

故有，

$$T(n) = \begin{cases} c_1 n & n < 90 \\ T(n/9) + T(63n/72) + c_1 n & n \geq 90 \end{cases}$$

用归纳法可证：

$$T(n) \leq 72 c_1 n$$

即，$T(n) = O(n)$

# 3.6 选择问题

- **SELECT2的实现**
  - 算法中需要解决的两个问题
  - 1) 如何确定子集合的中间值
    - 当r较小时，采用INSERTIONSORT(A, i, j)直接对每组的r个元素排序，在排序好的序列中，中间元素即为当前r个元素中的中间位置下标对应的元素。
  - 2) 如何保存⌊n/r⌋个子集合的中间值
    - 注：各组找到的中间元素值将调整到数组A的前部，连续保存，从而可方便用递归调用的方式对这些中间值进行排序并找出中间值的中间值。

## 算法3.17 SELECT2的SPARKS的描述

```
procedure SEL(A, m, p, k)
  //返回一个i，使得i∈[m, p]，且A(i)是A(m: p)中第k小元素，r是一个全程变量，其取值为大于1的整数
global r;  integer n, i, j;
loop
  if p-m+1≤r then call  INSERTIONSORT(A, m, p);  return (m+k-1); endif
  n←p-m+1 //元素数//
  for i←1 to  ⌊n/r⌋  do  //计算中间值//
    call  INSERTIONSORT(A, m+(i-1)*r, m+i*r-1)  //将中间值收集到A(m:p)的前部//
    call  INTERCHANGE(A(m+i-1), A(m+(i-1)r + ⌊r/2⌋-1))
  repeat
  j←SEL(A, m, m+ ⌊n/r⌋-1,  ⌈⌊n/r⌋/2⌉)//mm//
  call  INTERCHANGE (A(m), A(j)) //产生划分元素，将之调整到第一个元素//
  j←p+1
  call PARTITION(m, j)
  case
    :j-m+1=k: return(j)
    :j-m+1>k: p←j-1
    :else: k←k-(j-m+1); m←j+1
  endcase
 repeat
end SEL
```

# 第三章 分治法

- 3.1 一般方法
- 3.2 二分检索
- 3.3 找最大和最小元素
- 3.4 归并排序
- 3.5 快速排序
- 3.6 选择问题
- 3.7 斯特拉森矩阵乘法

# 3.7 斯特拉森矩阵乘法

## ■ 问题描述

- 矩阵的加法
  - 若A和B是2个n×n的矩阵，则它们的加法C=A+B同样是一个n×n的矩阵。
  - A和B的和矩阵C中的元素C[i, j]定义为:

$$c_{ij} = a_{ij} + b_{ij}, \qquad i, j = 1, 2, \ldots, n$$

  - 则每计算C的一个元素C[i, j]，需要做1次加法。
  - 因此求出矩阵C的$n^2$个元素所需的计算时间为$O(n^2)$。

中国科学院大学
University of Chinese Academy of Sciences

# 3.7 斯特拉森矩阵乘法

## ■ 问题描述

### □矩阵的乘法

- 若A和B是n×n的矩阵，则A和B的乘积矩阵C=AB同样是n×n的矩阵。C中的元素C[i, j]定义为:

$$C(i,j) = \sum_{1\le k\le n} A(i,k)B(k,j) \qquad 1 \le i,j \le n$$

- 计算C的一个元素C[i, j]，需要做n个乘法和n-1次加法。
- 因此求矩阵C的$n^2$个元素所需的计算时间为$O(n^3)$。
- 问：是否可以用少于$n^3$次乘法完成C的计算?
- 60年代末，Strassen采用了分治技术，将计算2个n阶矩阵乘积所需的计算时间改进到$O(n\log 7)=O(n^{2.81})$。

# 3.7 斯特拉森矩阵乘法

- 解法介绍
  - 假设n是2的幂。将矩阵A，B和C中每一矩阵都分块成为4个大小相等的子矩阵，每个子矩阵都是n/2×n/2的方阵。由此可将方程C=AB重写为:

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

  - 由此可得:

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

$C_{12}=A_{11}B_{12}+A_{12}B_{22}$

$C_{21}=A_{21}B_{11}+A_{22}B_{21}$

$C_{22}=A_{21}B_{12}+A_{22}B_{22}$

# 3.7 斯特拉森矩阵乘法

## ■ 解法介绍

### □复杂度分析

- 如果n=2，则2阶方阵的乘积可以直接用上式计算出来，共需8次乘法和4次加法。
- 当子矩阵的阶大于2时，为求2个子矩阵的积，可以继续将子矩阵分块，直到子矩阵的阶降为2。这样，就产生了一个分治降阶的递归算法。

# 3.7 斯特拉森矩阵乘法

■ 解法介绍

□复杂度分析

➢令T(n)表示两个n×n矩阵相乘的计算时间。

① **8**次**(n/2)** ×**(n/2)** 矩阵乘　——————>**8T(n/2)**

② **4**次**(n/2)** ×**(n/2)** 矩阵加　——————>**dn²**

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

$$\Big\Downarrow \ \text{渐近复杂度}$$

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

➢其中，**b**，**d**是常数。

# 3.7 斯特拉森矩阵乘法

- 解法介绍
  - 复杂度分析
    - 该方法并不比用原始定义直接计算更有效。
    - 原因
      - 由于没有减少矩阵的乘法次数。
    - 观察：
      - 矩阵乘法的花费比矩阵加法大
      - $O(n^3)$对$O(n^2)$
    - 要想改进矩阵乘法的计算时间复杂性，必须减少子矩阵乘法运算的次数。

# 3.7 斯特拉森矩阵乘法

## ■ 解法描述

- 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
  - $P=(A_{11}+A_{22})(B_{11}+B_{22})$
  - $Q=(A_{21}+A_{22})B_{11}$
  - $R=A_{11}(B_{12}-B_{22})$
  - $S=A_{22}(B_{21}-B_{11})$
  - $T=(A_{11}+A_{12})B_{22}$
  - $U=(A_{21}-A_{11})(B_{11}+B_{12})$
  - $V=(A_{12}-A_{22})(B_{21}+B_{22})$

7个乘法和10个加(减)法

# 3.7 斯特拉森矩阵乘法

8个加(减)法

## ■ 解法描述

- 用八个加减法计算$C_{ij}$
  - $C_{11}=P+S-T+V$
  - $C_{12}=R+T$
  - $C_{21}=Q+S$
  - $C_{22}=P+R-Q+U$
  - 共用7次乘法和18次加减法
  - $C_{11}=P+S-T+V =(A_{11}+A_{22})(B_{11}+B_{22}) + A_{22}(B_{21}-B_{11}) - (A_{11}+A_{12})B_{22} + (A_{12}-A_{22}) (B_{21}+ B_{22})=A_{11}B_{11}+A_{12}B_{21}$
  - $C_{12}=R+T = A_{11}(B_{12}-B_{22}) + (A_{11}+A_{12})B_{22}= A_{11}B_{12}-A_{11}B_{22} + A_{11}B_{22}+A_{12}B_{22} = A_{11}B_{12}+A_{12}B_{22}$

# 3.7 斯特拉森矩阵乘法

## ■ 解法描述

□斯特拉森时间复杂度

$$T(2) = b$$

$$T(n) = 7T(n/2) + an^2 \qquad n > 2$$

□按照解递归方程的套用公式法，其解为

$$
\begin{aligned}
T(n) &= an^2(1+7/4+(7/4)^2+\cdots+(7/4)^{k-1})+7^k T(1) \\
&\le cn^2(7/4)^{\log n}+7^{\log n} \\
&= cn^{\log 4+\log 7-\log 4}+n^{\log 7} \\
&= (c+1)n^{\log 7} = O(n^{\log 7}) \approx O(n^{2.81})
\end{aligned}
$$

# 3.7 斯特拉森矩阵乘法

- 其他矩阵乘法
  - 有人曾列举了计算2个2阶矩阵乘法的36种不同方法。但所有的方法都要做7次乘法。
  - 除非能找到一种计算2阶方阵乘积的算法，使乘法的计算次数少于7次，按上述思路才有可能进一步改进矩阵乘积的计算时间的上界。
  - 但是Hopcroft 和 Kerr(197l) 已经证明，计算2个2×2矩阵的乘积，7次乘法是必要的。

# 3.7 斯特拉森矩阵乘法

- 其他矩阵乘法
  - 因此要想进一步改进矩阵乘法的时间复杂性，就不能再寄希望于计算2×2矩阵的乘法次数的减少。或许应当研究3×3或5×5矩阵的更好算法。
  - 在Strassen之后又有许多算法改进了矩阵乘法的计算时间复杂性。目前最好的计算时间上界是$O(n^{2.367})$。而目前所知道的矩阵乘法的最好下界仍是它的平凡下界$\Omega(n^2)$。

# 3.7 斯特拉森矩阵乘法

- 实际性能分析
  - 斯特拉森矩阵乘法目前还只具有理论意义，因为只有当n相当大时他才优于通常的矩阵乘法。
  - 经验表明，当n=120时，斯特拉森矩阵乘法与通常的矩阵乘法在计算上无显著差别。
  - 有益的启示
    - 由定义出发所直接给出的明显算法并非总是最好的。

# 作业-算法实现2

- ■ 棋盘覆盖问题
  - □在一个$2^k \times 2^k$个方格组成的棋盘中，恰有一个方格与其他方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。如图1所示，蓝色的为特殊方格：

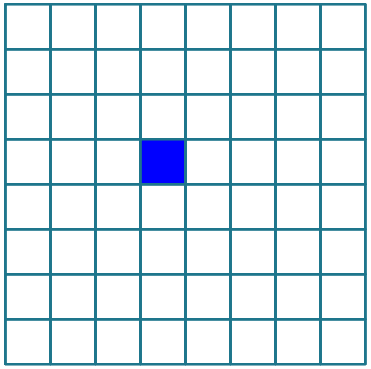


图1 特殊棋盘，蓝色的为特殊方格

图2 四种L形骨牌

# 作业-算法实现2

- 棋盘覆盖问题
  - 棋盘覆盖问题是指，要用图2中的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。
- 作业
  - 用分治法设计一个求解棋盘覆盖问题的算法。
  - 用C(C++)或者Matlab语言实现。
  - 有求解思路的简单说明。
  - 上载到课程网站上。

# End