

算法设计与分析

主讲人：权丽君

Email: *ljquan@suda.edu.cn*

苏州大学 计算机学院



第6讲 中位数和顺序统计量

内容提要：

- 最小值和最大值
- 期望为线性时间的选择算法
- 最坏情况为线性时间的选择



基本概念：

1 **顺序统计量**: 在一个由n个元素组成的集合中，第i个顺序统计量(order statistic)是该集合中的第i小的元素。

如: 在一个元素集合中, 最小值是第1个顺序统计量 ($i=1$) ; 最大值是第n个顺序统计量 ($i=n$) .

2 **中位数**: 对一个有n个元素的集合, 将数据排序后, 位置在最中间的数称为该集合的中位数。

- 当元素数为奇数时, 中位数出现在 $i = (n+1)/2$ 处; 如: 1、2、3、6、7的中位数是3。
- 当元素数为偶数时, 中位数取作第 $n/2$ 个数据与第 $n/2+1$ 个数据的算术平均值。如: 1、2、3、5的中位数是2.5。



- 当元素数为偶数时，也可视为存在**两个中位数**，分别出现在 $i=n/2$ (称为下中位数)和 $i=n/2+1$ (称为上中位数) 处。

如： 1、2、3、5的下中位数是2，上中位数是3。

- 一般情况下，不管元素数是偶数或奇数，可以用下式计算

：

- 下中位数: $i = \left\lfloor \frac{n+1}{2} \right\rfloor$

- 上中位数: $i = \left\lceil \frac{n+1}{2} \right\rceil$

注：实际中多取下中位数

如： 1) 1、2、3、6、7的中位数是3。

$$\lfloor (5+1)/2 \rfloor = \lceil (5+1)/2 \rceil = 3$$

2) 1、2、3、5的下中位数是2，上中位数是3。

下中位数: $\lfloor (4+1)/2 \rfloor = 2$

上中位数: $\lceil (4+1)/2 \rceil = 3$



本章讨论首先从一个由n个元素构成的

集合中选择 第*i*个顺序统计量的选择问题，

然后再讨论中位数问题。

选择问题：从n个元素的集合中选择第*i*个顺序统计量的问题形式化地归结为“选择问题”。

- 假设集合中的元素是互异的（可推广至包含重复元素的情形）。

输入：一个包含n个（互异）元素的集合A和一个整数*i*， $1 \leq i \leq n$ 。

输出：元素 $x \in A$ ，且A中恰好有*i*-1个其他元素小于它。



How to do ?

1 排序

元素集合排序后，位于第 i 位的元素即为该集合的第 i 个顺序统计量。

时间复杂度： $O(n \log n)$

2 选择算法

设法找出元素集合里面的第 i 小元素，该元素为集合的第 i 个 顺序统计量。

时间复杂度： $O(n)$



第6讲 中位数和顺序统计量

内容提要：

- 最小值和最大值
- 期望为线性时间的选择算法
- 最坏情况为线性时间的选择



9.1 最小值和最大值

在一个有 n 个元素的集合中，需要做多少次比较才能确定其最小元素呢？

$n-1$ 次，时间： $O(n)$

```
MINIMUM( $A$ )
1    $min = A[1]$ 
2   for  $i = 2$  to  $A.length$ 
3       if  $min > A[i]$ 
4            $min = A[i]$ 
5   return  $min$ 
```

- 集合元素存放在数组A中
- **A.length**: 数组长度
这里， $A.length=n$ 。

最大值呢？ $n-1$ ，时间： $O(n)$

思考：这是求解上述问题的最好结果吗？ 是的！



锦标赛算法：

为了确定集合中的最小值，每一轮元素之间要进行两两的比较，每次比较都可看作“锦标赛”中的一场比赛，胜者参加下一轮的比赛，直到得到最后的胜出者。

- 为了得到最终的胜者（最小值），必须要做 $n-1$ 次比较。
 - 除了最终的获胜者，其他每个元素都至少要输掉一场比赛。
 - 求最小（最大）值算法是最优的。
-
- 若同时找集合中的最大值和最小值，共需要多少次比较呢？



同时找集合中的最大值和最小值

- 如果分别独立地找其中的最小值和最大值，则各需做 $n-1$ 次比较，共需 $2n-2$ 次比较。
- 能不能快一点？
 - 1) 记录比较过程中遇到的最小值和最大值；
 - 2) 成对处理元素，先比较两个输入元素，把较小者与当前最小值比较，较大者与当前最大值比较（每对元素需要3次比较）

MAX-MINIMUM (A)

```
1 if length[A] is odd
2   then min ← A[1]; max ← min;
3   else min ← MIN( A[1], A[2] ), max ← MAX( A[1], A[2] );
4 i++;
5 while i ≤ length[A]
6   min ← MIN( MIN(A[i], A[i+1]), min )
7   max ← MAX( MAX(A[i], A[i+1]), max )
8   i ← i+2;
9 end
10 return min, max
```



最小值和最大值

- 如何设定当前最小值和最大值的初始值：依赖于 n 的奇偶性
 - 如果 n 是奇数，将最小值和最大值都设为第一个元素的值；
 - 如果 n 是偶数，就对前两个元素做一次比较，以决定最小值和最大值的初始值。
- 总的比较次数：
 - 如果 n 是奇数，那么总共做了 $3\lfloor n/2 \rfloor$ 次比较
 - 如果 n 是偶数，总共做了 $3n/2 - 2$ 次比较。

时间复杂度为： $O(n)$



第6讲 中位数和顺序统计量

内容提要：

- 最小值和最大值
- 期望为线性时间的选择算法
- 最坏情况为线性时间的选择



9.2 期望为线性时间的选择算法

- 借助QUICKSORT的PARTITION过程

PARTITION(A, p, r)

```

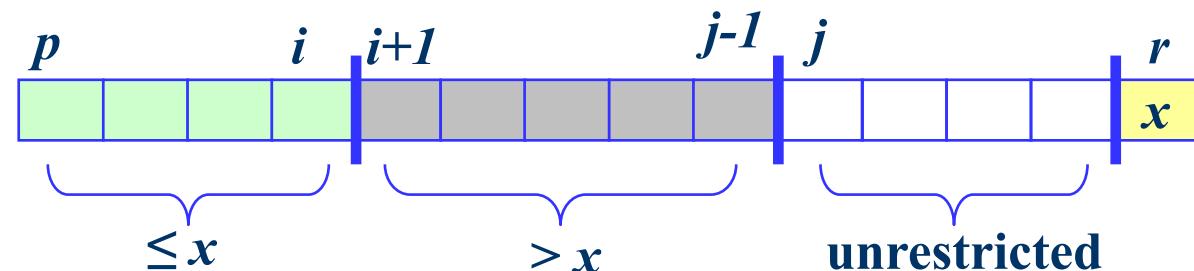
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4    if  $A[j] \leq x$ 
5       $i = i + 1$ 
6      exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 
```

随机化的PARTITION过程

RANDOMIZED-PARTITION(A, p, r)

```

1   $i = \text{RANDOM}(p, r)$  ← 随机选择划分元素
2  exchange  $A[r]$  with  $A[i]$ 
3  return PARTITION( $A, p, r$ )
```



QUICKSORT(A, p, r)

```

1  if  $p < r$ 
2     $q = \text{PARTITION}(A, p, r)$ 
3    QUICKSORT( $A, p, q - 1$ )
4    QUICKSORT( $A, q + 1, r$ )
```

RANDOMIZED-QUICKSORT(A, p, r)

```

1  if  $p < r$ 
2     $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
3    RANDOMIZED-QUICKSORT( $A, p, q - 1$ )
4    RANDOMIZED-QUICKSORT( $A, q + 1, r$ )
```



2 利用RANDOMIZED-PARTITION设计一个较低时间复杂度的 算法找集合中的第*i*小元素

PARTITION(1,n): 设在第一次划分后，主元素v被放在位置 $A(j)$ 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 $A(j)$ 。

此时，

- 若 $i = j$ ，则 $A(j)$ 即是第 i 小元素；否则，
- 若 $i < j$ ，则 $A(1:n)$ 中的第 i 小元素将出现在 $A(1:j-1)$ 中；
- 若 $i > j$ ，则 $A(1:n)$ 中的第 i 小元素将出现在 $A(j+1:n)$ 中。

j-1元
素

n-j元
素

1

j

n

A(j)即是第i小元
素

j-1元
素

n-j元
素

1

j

n

i < j

第i小元素在A(1:j-1)中，
且是A(1:j-1)中的第i小元
素

j-1元
素

n-j元
素

1

j

n

i > j

j+1:新的起
点

第i小元素在A(j+1:n)中，
但是A(j+1:n)中的第i-j小元素

情况1:

情况2:

情况3:



利用RANDOMIZED-PARTITION 实现选择算法

在 $A[p, r]$ 中找第 i 小元素的算法：

RANDOMIZED-SELECT(A, p, r, i)

```
1  if  $p == r$ 
2      return  $A[p]$ 
3   $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
4   $k = q - p + 1$ 
5  if  $i == k$           // the pivot value is the answer
6      return  $A[q]$ 
7  elseif  $i < k$ 
8      return RANDOMIZED-SELECT( $A, p, q - 1, i$ )
9  else return RANDOMIZED-SELECT( $A, q + 1, r, i - k$ )
```

■ RANDOMIZED-SELECT的最坏情况运行时间是 $O(n^2)$

➤ 最坏情况下的特例：输入 A 恰好使对RANDOMIZED-PARTITION 的第 j 次调用选中的主元素是第 j 小元素，而 $i=n$ 。



利用RANDOMIZED-PARTITION 实现选择算法

时间复杂度分析：

- 幸运的例子：每次都能去除十分之一以上。

$$T(n) = T(\frac{9n}{10}) + \Theta(n) = O(n)$$

可利用主方法计算出。

$$n^{\log_{10} \frac{1}{9}} = n^0 = 1.$$

- 运气不好的例子：每次都只能去除一个元素。

$$T(n) = T(n-1) + \Theta(n) = O(n^2)$$



RANDOMIZED-SELECT的期望运行时间是O(n)。

证明：

设算法的运行时间是一个随机变量，记为 $T(n)$ 。

设RANDOMIZED-PARTITION(A, p, r) 可以等概率地返回任何元素作为主元。即，对每个 k ($1 \leq k \leq n$)，划分后区间 $A[p, q]$ 恰好有 k 个元素（全部小于或等于主元）的概率是 $1/(r-p+1)$ 。

对所有 $k=1, 2, \dots, n$ ，定义指示器随机变量 X_k ：

划分后主
元在位置 q

$$X_k = I\{\text{子数组 } A[p..q] \text{ 正好包含 } k \text{ 个元素}\}$$

假设 A 中元素是互异的，则有 $E[X_k] = 1/n$

■ 期望上界分析

- **RANDOMIZED-SELECT** 当前处理中， $A[q]$ 是主元。若*i=q*，则得到正确答案，结束过程。否则在 $A[p,q-1]$ 或 $A[q+1,r]$ 上递归。
- 对一次给定的**RANDOMIZED-SELECT**调用，若主元素恰好落在给定的*k*值，则指示器随机变量 X_k 值为1，对其它值都为0。
- 设 $T(n)$ 是单调递增的。
 - 为了分析递归调用所需时间的上界，我们设每次划分都有：(很不幸地)第*i*个元素总落在元素数较多的一边。
 - 当 $X_k=1$ 时，若需递归，两个子数组的大小分别为*k-1*和*n-k*，算法只在其中之一、并设是在较大的子数组上递归执行。

则有以下递归式：



$$\begin{aligned} T(n) &\leq \sum_{k=1}^n X_k \cdot (T(\max(k-1, n-k)) + O(n)) \\ &= \sum_{k=1}^n X_k \cdot T(\max(k-1, n-k)) + O(n) . \end{aligned}$$

■ 两边取期望：

$$\begin{aligned} \mathbb{E}[T(n)] &\leq \mathbb{E}\left[\sum_{k=1}^n X_k \cdot T(\max(k-1, n-k)) + O(n)\right] \\ &= \sum_{k=1}^n \mathbb{E}[X_k \cdot T(\max(k-1, n-k))] + O(n) \quad (\text{by linearity of expectation}) \\ &= \sum_{k=1}^n \mathbb{E}[X_k] \cdot \mathbb{E}[T(\max(k-1, n-k))] + O(n) \quad (\text{by equation (C.24)}) \\ &= \sum_{k=1}^n \frac{1}{n} \cdot \mathbb{E}[T(\max(k-1, n-k))] + O(n) \quad (\text{by equation (9.1)}) . \end{aligned}$$

注：公式C.24的应用依赖于 X_k 和 $T(\max(k-1, n-k))$ 是独立的随机变量。见习题9.2-2

$$\begin{aligned}
& \sum_{k=1}^n E[T(\max(k-1, n-k))] \\
&= E(T(\max(0, n-1))) + E(T(\max(1, n-2))) + \cdots + E(T(\max(n-1, 0))) \\
&= E(T(n-1)) + E(T(n-2)) + \dots + E(T(1))
\end{aligned}$$

在 $k=1 \sim n$ 的区间里，表达式 $T(\max(k-1, n-k))$ 有：

- 如果 n 是偶数，则从 $T(\lceil n/2 \rceil)$ 到 $T(n-1)$ 的每一项在总和中恰好出现两次；
- 如果 n 是奇数，则 $T(\lceil n/2 \rceil)$ 出现一次，从 $T(\lceil \frac{n}{2} \rceil + 1)$ 到 $T(n-1)$ 各项在总和中出现两次；

则有：

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n).$$



代换法证明：

$E[T(n)] = O(n)$.

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n).$$

■ 即证明：存在常数c，使得 $E[T(n)] \leq cn$ 。

■ 将上述猜测代入推论证明阶段有：

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an$$

注：a是为去掉 $E[T(n)]$ 中的 $O(n)$ 而引入的常数

$$= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an$$

$$= \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an$$

$$\leq \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2} \right) + an$$

$$= \frac{2c}{n} \left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an$$

$$= \frac{c}{n} \left(\frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an$$

$$\begin{aligned} &= c \left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \\ &\leq \frac{3cn}{4} + \frac{c}{2} + an \\ &= cn - \left(\frac{cn}{4} - \frac{c}{2} - an \right). \end{aligned}$$

■ 这里，为了证明 $E[T(n)] \leq cn$ ，须有 $cn/4 - c/2 - an \geq 0$.

■ 什么样的c能满足？



■ (续 : $cn/4 - c/2 - an \geq 0$ 何时成立 ?)

即要求有 : $n(c/4 - a) \geq c/2$

选取常数 c , 使得 $(c/4 - a) > 0$, 两边同除 $(c/4 - a)$, 则有

$$n \geq \frac{c/2}{c/4 - a} = \frac{2c}{c - 4a}.$$

因此, 当 $n \geq 2c/(c-4a)$ 时, 对任意的 n 有 $E[T(n)] \leq cn$, 即

$E[T(n)] = O(n)$ 成立。

➤ $n < 2c/(c-4a)$ 时, 可假设 $T(n) = O(1)$ 。

结论: 若所有元素互异, 则可在线性期望时间内, 找到任意顺序统计量。



第6讲 中位数和顺序统计量

内容提要：

- 最小值和最大值
- 期望为线性时间的选择算法
- 最坏情况为线性时间的选择



9.3 最坏情况是O(n)的选择算法

- 造成最坏情况是 $O(n^2)$ 的原因分析：类似快速排序的最坏情况
- 采用两次取中间值的规则精心选取划分元素

目标：精心选择划分元素，避免随机选取可能出现的极端情况。

分三步：

首先，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 $n - r\lfloor n/r \rfloor$ 个元素可以忽略不计)

然后，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(中位数)——一次取中。

再后，对这 $\lfloor n/r \rfloor$ 个中间值查找，再找出其中间值 mm (中位数)。

——二次取中。

最后，将 mm 作为划分元素执行划分。

if $i = k$ then return x ;

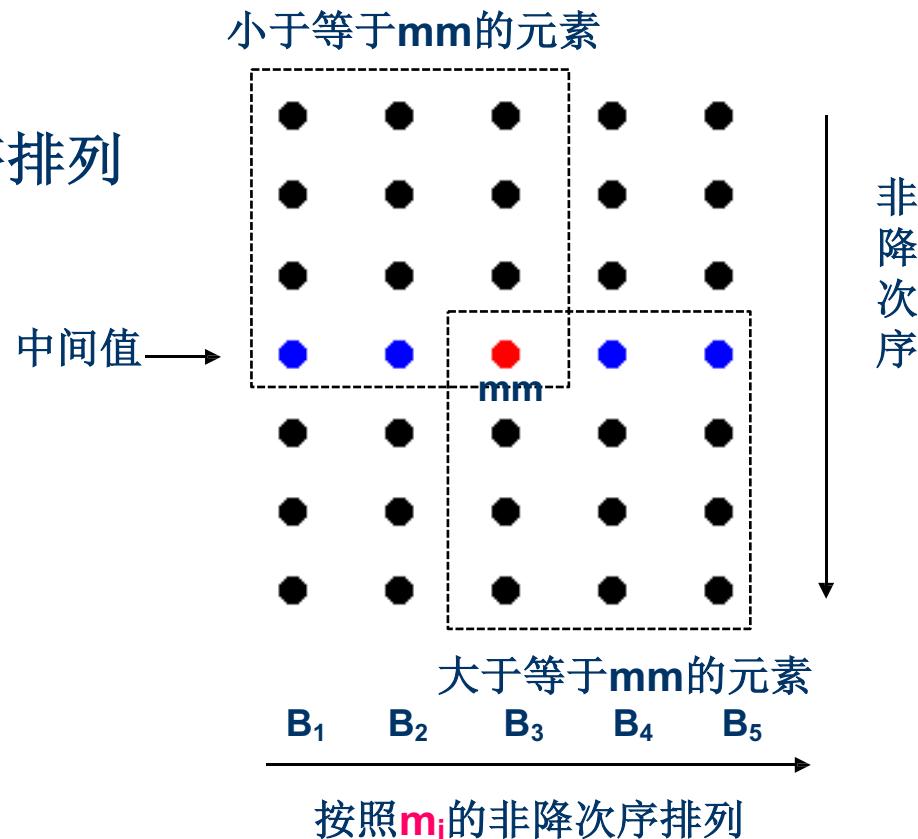
else if $i < k$ then 找左区间的第 i 个最小元;
else 找右区间的第 $i-k$ 个最小元



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
。
- $m_m = m_i$ 的中间值, $1 \leq i \leq 5$

由图所示有:





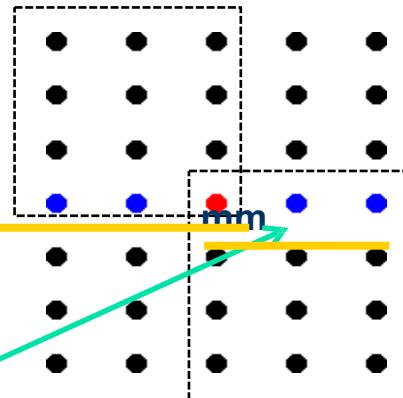
r 个元素的中间值是第 $\lceil r/2 \rceil$ 小元素;

至少有 $\lfloor n/r \rfloor/2$ 个 m_i 小于或等于 mm ;

也至多有 $\lfloor n/r \rfloor - \lfloor n/r \rfloor/2 + 1 \geq \lfloor n/r \rfloor/2$ 个 m_i 大于或等于 mm 。

小于等于 mm 的元素

$\lceil r/2 \rceil$
个中间值



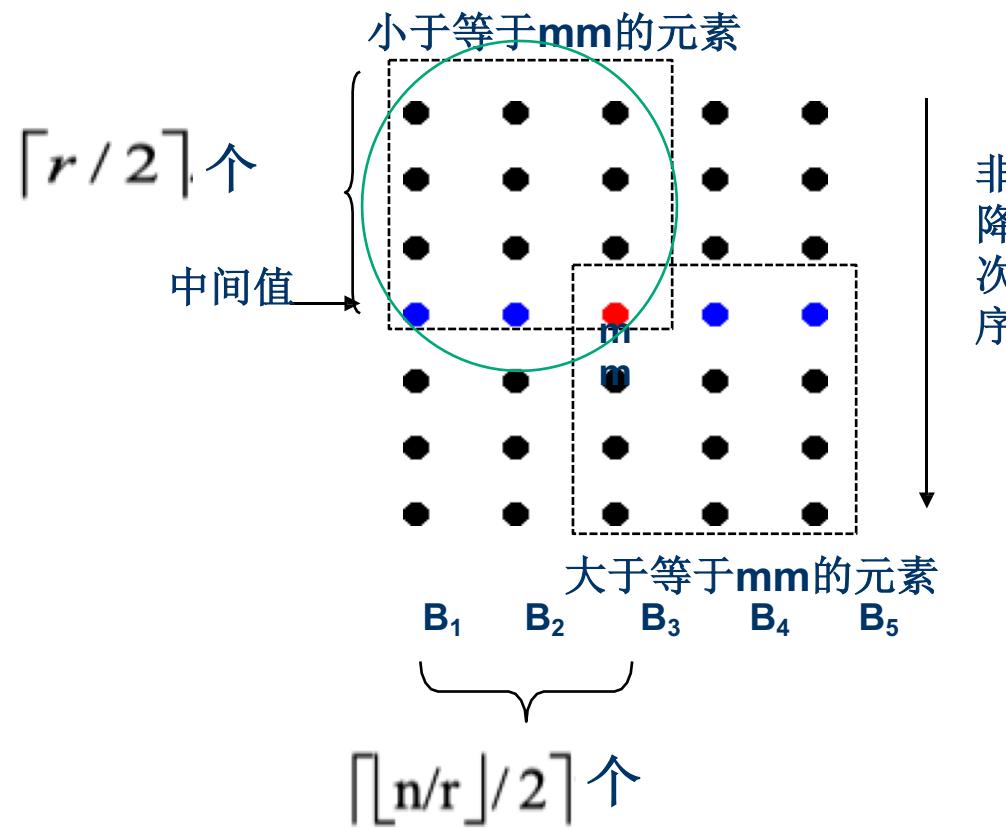
大于等于 mm 的元素

$B_1 \quad B_2 \quad B_3 \quad B_4 \quad B_5$

$\lfloor n/r \rfloor/2$ 个

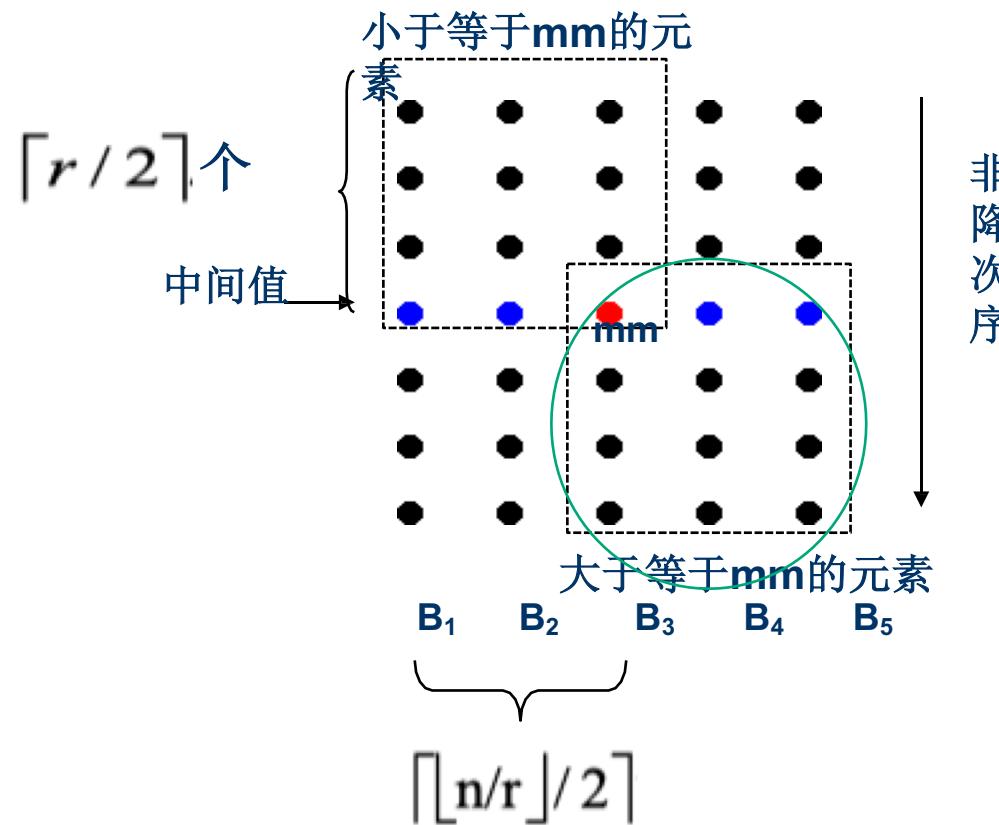


故，至少有 $\lceil r/2 \rceil \lceil n/r \rfloor / 2$ 个元素小于或等于 m





同理，至少有 $\lceil r/2 \rceil \lfloor n/r \rfloor / 2$ 个元素大于或等于 m_m





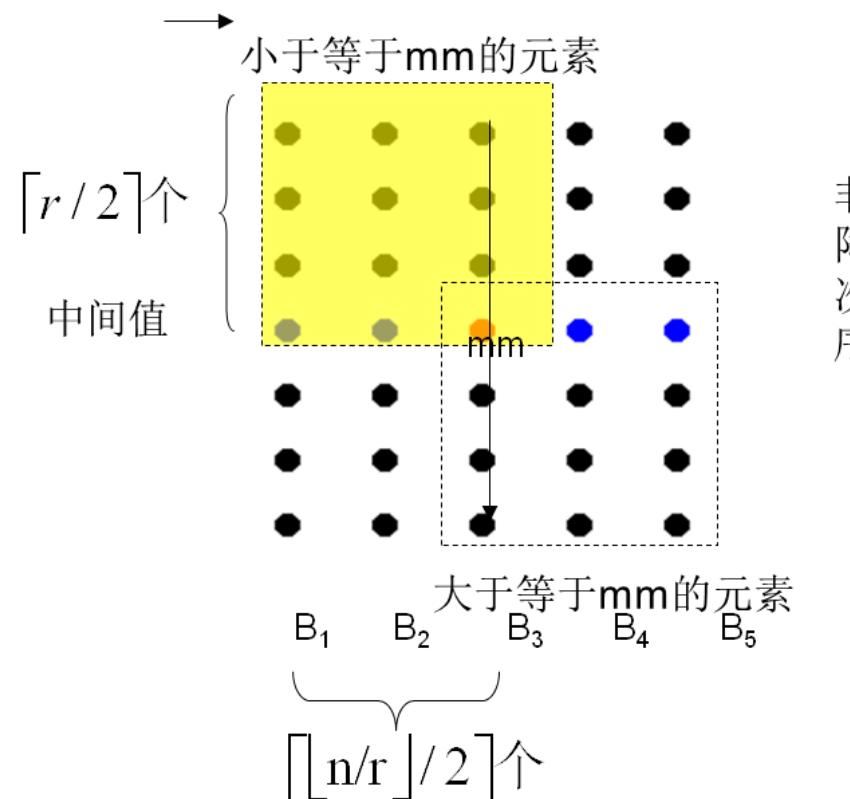
以 $r=5$ 为例。使用两次取中间值规则来选择划分元素 mm。可得到，

- ◆ 至少有 $1.5\lfloor n/5 \rfloor$ 个元素小于或等于选择元素 mm
- ◆ 且至多有 $n - 1.5\lfloor n/5 \rfloor \leq 0.7n + 1.2$ 个元素大于等于 mm

$$\begin{aligned} & \lceil r/2 \rceil \lceil \lfloor n/r \rfloor / 2 \rceil \\ &= \lceil 5/2 \rceil \lceil \lfloor n/5 \rfloor / 2 \rceil \\ &\geq 3\lfloor n/5 \rfloor / 2 = 1.5\lfloor n/5 \rfloor \end{aligned}$$

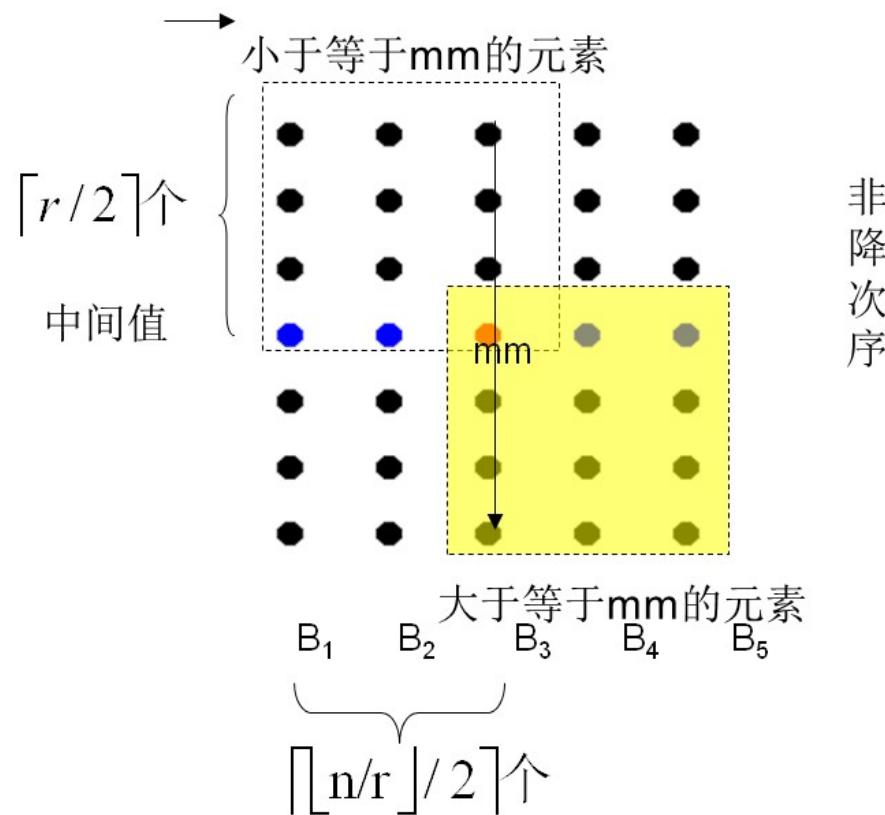
$$\begin{aligned} & n - 1.5\lfloor n/5 \rfloor \\ &\leq n - 1.5(n - 4)/5 \\ &= 0.7n + 1.2 \\ &\text{注: } \lfloor n/5 \rfloor \geq (n - 4)/5 \end{aligned}$$

算法导论: $0.7n+6$



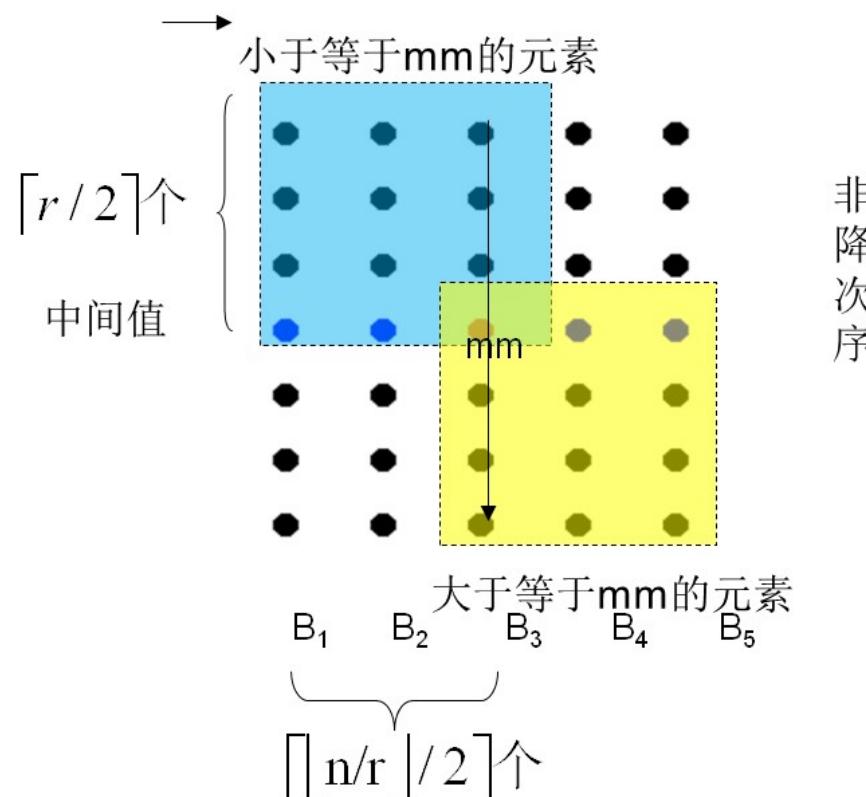
同理，

- ◆ 至少有 $1.5\lfloor n/5 \rfloor$ 个元素大于或等于选择元素 mm
- ◆ 且至多有 $n - 1.5\lfloor n/5 \rfloor \leq 0.7n + 1.2$ 个元素小于等于 mm



故，

这样的 m 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。则，不论落在哪个区域，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。





2) 算法描述

使用二次取中规则的选择算法的说明性描述

Procedure SELECT2(A,i,n)

// 在集合A中找第i 小元素

- ① 若 $n \leq r$, 则采用插入排序法直接对A分类并返回第i 小元素。否则
- 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 $M=\{m_1, m_2, \dots, m_{\lfloor n/r \rfloor}\}$ 是 $\lfloor n/r \rfloor$ 个子集合的中间值集合
- ④ $v \leftarrow \text{SELECT2}(M, \lceil \lfloor n/r \rfloor / 2 \rceil, \lfloor n/r \rfloor)$ □
- ⑤ $j \leftarrow \text{PARTITION}(A, v)$
- ⑥ case
 - : $i=j$: return(v)
 - : $i < j$: 设 S 是 $A(1:j-1)$ 中元素的集合; return($\text{SELECT2}(S, i, j-1)$)
 - :else: 设 R 是 $A(j+1:n)$ 中元素的集合; return($\text{SELECT2}(R, i-j, n-j)$)

endcase

end SELECT2



P123 思路

- 基本思想：类似 RandomizedSelect 算法，通过对输入数组进行递归划分来找出所求元素，但是算法保证每次对数组的划分是个好划分
- 主要步骤：

Step 1. 将 n 个元素分成 5 个 1 组，共 $\text{ceiling}(n/5)$ 组。其中，最后 1 组有 $n \bmod 5$ 个元素；

Step 2. 用插入排序对每组排序，取其中值。若最后 1 组有偶数个元素，取较小的中值；

Step 3. 递归地使用本算法寻找 $\text{ceiling}(n/5)$ 个中位数的中值 x ；

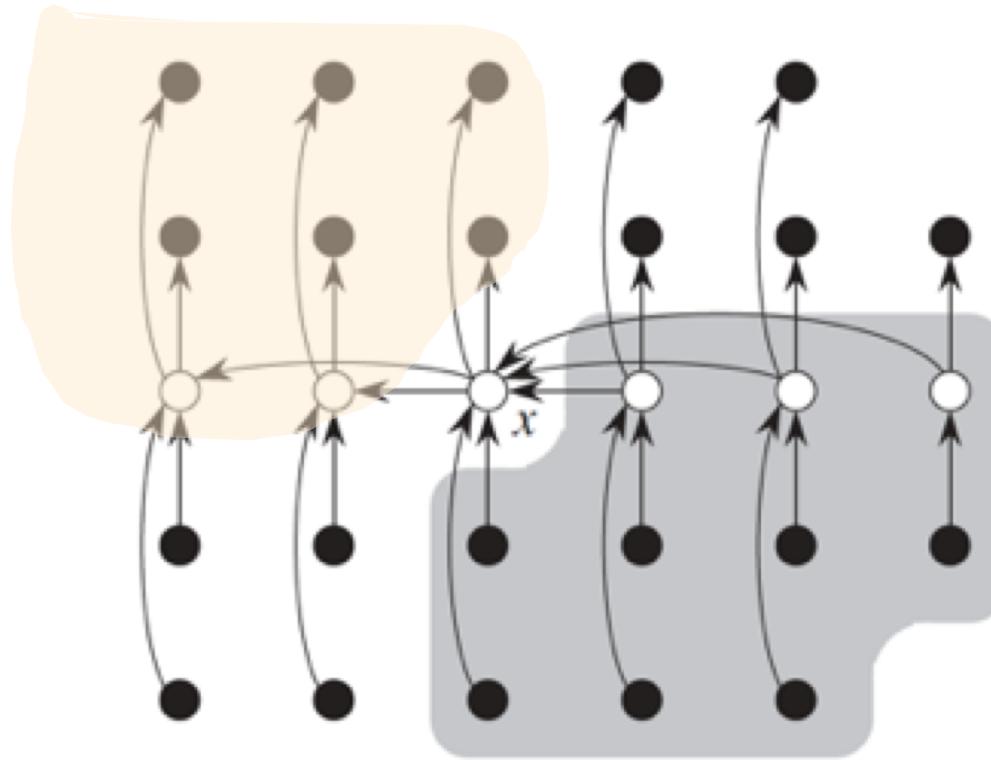
// 第一次递归调用本身

Step 4. 用 x 作为划分元对数组 A 进行划分，并设 x 是第 k 个最小元；

Step 5. if $i = k$ then return x ;

else if $i < k$ then 找左区间的第 i 个最小元； // 第二次递归调用本身

else 找右区间的第 $i-k$ 个最小元





时间复杂度分析：

- 由上页图示，可知至少有 $3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$ 的元素较x来得大。
- 同理，至少有 $3n/10 - 6$ 的元素较x来得小。

- 如果**Partition**过， $i \neq k$ ，则至多只要在 $7n/10 + 6$ 个元素的情况下递归调用**Select**。
- 而先前找出 $\lceil n/5 \rceil$ 小组中位数的中位数时，只在 $n/5$ 个元素的情况下递归调用 **Select**。

- 故 $T(n)=T(\lceil n/5 \rceil)+T(7n/10+6)+\Theta(n)$, for $n \geq 140$.



□ 利用替换法，令 $T(n) = O(cn)$

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(7n/10 + 6) + an \\ &\leq cn/5 + c + c7n/10 + 6c + an \\ &= 9cn/10 + 7c + an \\ &= cn + (-cn/10 + 7c + an) \\ &\leq cn, \quad \text{if } -cn/10 + 7c + an \leq 0! \end{aligned}$$

假设 $n > 140$ 时， $c \geq 20a$ ，上式就可以成立！

当 $n > 70$ 时，不等式(9.2)等价于不等式 $c \geq 10a(n/(n-70))$ 。因为假设 $n > 140$ ，所以有 $n/(n-70) \leq 2$ 。因此，选择 $c \geq 20a$ 就能够满足不等式(9.2)。（注意，这里常数 140 并没有什么特别之处，我们可以用任何严格大于 70 的整数来替换它，然后再相应地选择 c 即可。）因此，最坏情况下 SELECT 的运行时间是线性的。



```
Select( A, p, r, i){  
    if ( r-p <= 140 ) {  
        用简单的排序算法对数组A[p..r]进行排序；  
        return A[p+k-1];  
    }  
    n = r - p + 1;  
    for i ← 0 to floor(n/5) //寻找每组的中位数  
        将A[p + 5 * i] 至A[p+5*i+4] 的第3小元素与A[p+i] 交换位置；  
    x = Select( A, p, p + floor(n/5), floor(n/10) ); //找中位数的中位数  
    j = Partition(A, p, r, x ), k = j - p + 1;  
    if( k <= j ) return Select ( A, p, i, k );  
    else return Select( A, i + 1, r, k - j );  
}
```



```
Select(A, p, r, i){
```

```
    if (r-p <= 140) {
```

用简单的排序算法对数组A[p..r]进行排序；

```
        return A[P + i - k]
```

```
}
```

```
n = r - p + 1;
```

插入排序，需要注意不满5个元素的

```
for i ← 0 to floor(n/5) { //寻找每组的中位数
```

将A[p + 5 * i] 至A[p+5*i+4] 的第3小元素与A[p+i] 交换位置； }

```
x = Select(A, p, p + floor(n/5), floor(n/10)); //找中位数的中位数
```

```
j = Partition(A, p, r, x), k = j - p + 1;
```

```
if( i == k) return x;  
if( i < k) return Select(A, p, j-1, i);  
return Select(A, j+1, r, i-k);
```

```
}
```