算法设计与分析

Computer Algorithm Design & Analysis

吕志鹏
zhipeng.lv@hust.edu.cn
群名称：算法设计与分析

关于排序

排序主要参见数据结构课程和教材第6~8章相关内容。

第6章 堆排序

1、堆的定义及堆的操作：HEAPIFY、建堆

2、堆排序的基本思想和分析

3、优先队列（6.5，★）

优先队列（Priority Queue）：是一种用来维护由一组元素构成的集合S的**数据结构**，其中的每一个元素都有一个相关的值，称为**关键字**(key)。优先队列有最大优先队列和最小优先队列。

第8章 线性时间的排序算法

- 计数排序
- 基数排序
- 桶排序

- 以比较为基础的排序算法的时间下界

以比较为基础的排序：只使用比较运算来决定元素之间的大小关系并调整其位置，不做改变元素值大小等其它操作的排序算法

Chapter 9
Medians and Order Statistics

中位数和顺序统计量

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： $i=\lfloor (n+1)/2 \rfloor$,
  - 上中位数： $i=\lceil (n+1)/2 \rceil$

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： $\lceil (4+1)/2 \rceil = 3$

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≤i≤n。

输出：元素x∈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：$O(n)$

9.1 最小值和最大值

- **O(n)求最大值、最小值**

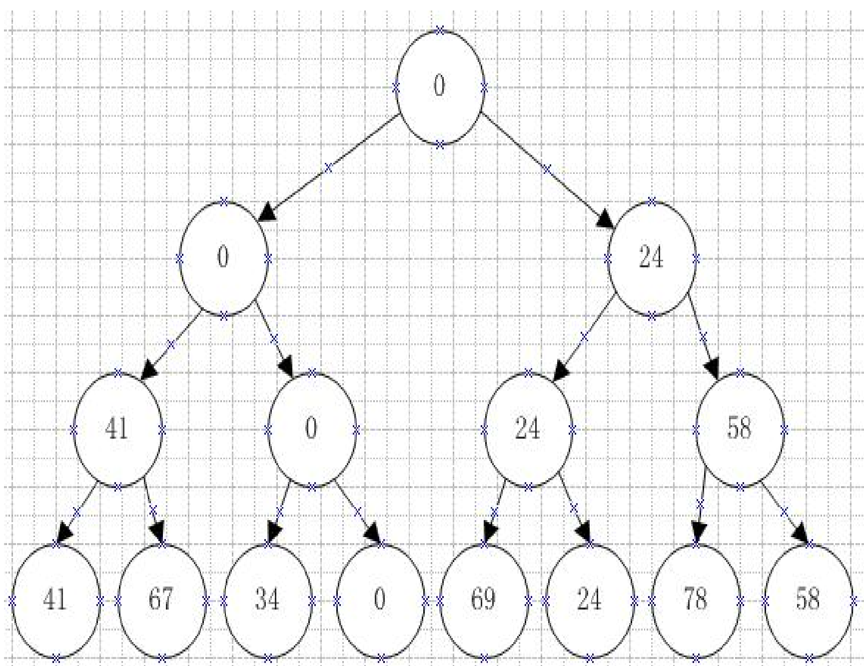
　　这个采用最直观朴素的解法就能解决，我们取个名字吧，叫做“锦标赛法”。就是一个个比较，时间复杂度O(n)

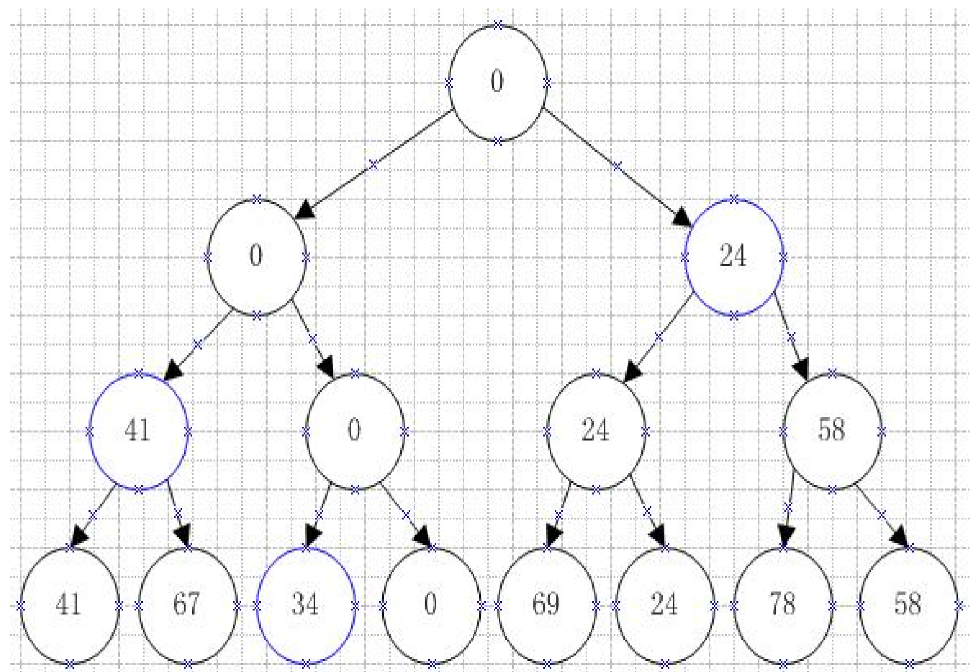
- **3/2n次比较同时求最大最小值**

按照锦标赛法，同时求最大最小值，需要2(n-1)次比较，但是换一种思路，我们没必要一个元素比较两次，而是两个元素比较一次，然后得出大小关系，再分别和最大、最小值比较，这样两个元素就只用比较3次，总共就是3/2n次。

求第二小的元素

- **最坏情况下，n+lgn-2次比较求第二小的元素**
- 本处要求的时间复杂度中含有lgn，我们自然想到这恰好是由n个元素组成的二叉树中的树的高度。有了这个提示之后，我们把思考点放在如何将n个元素的比较转化成一棵二叉树来求。





9.2 期望为线性时间的选择算法

- 借助QUICKSORT的PARTITION过程

随机化的PARTITION过程

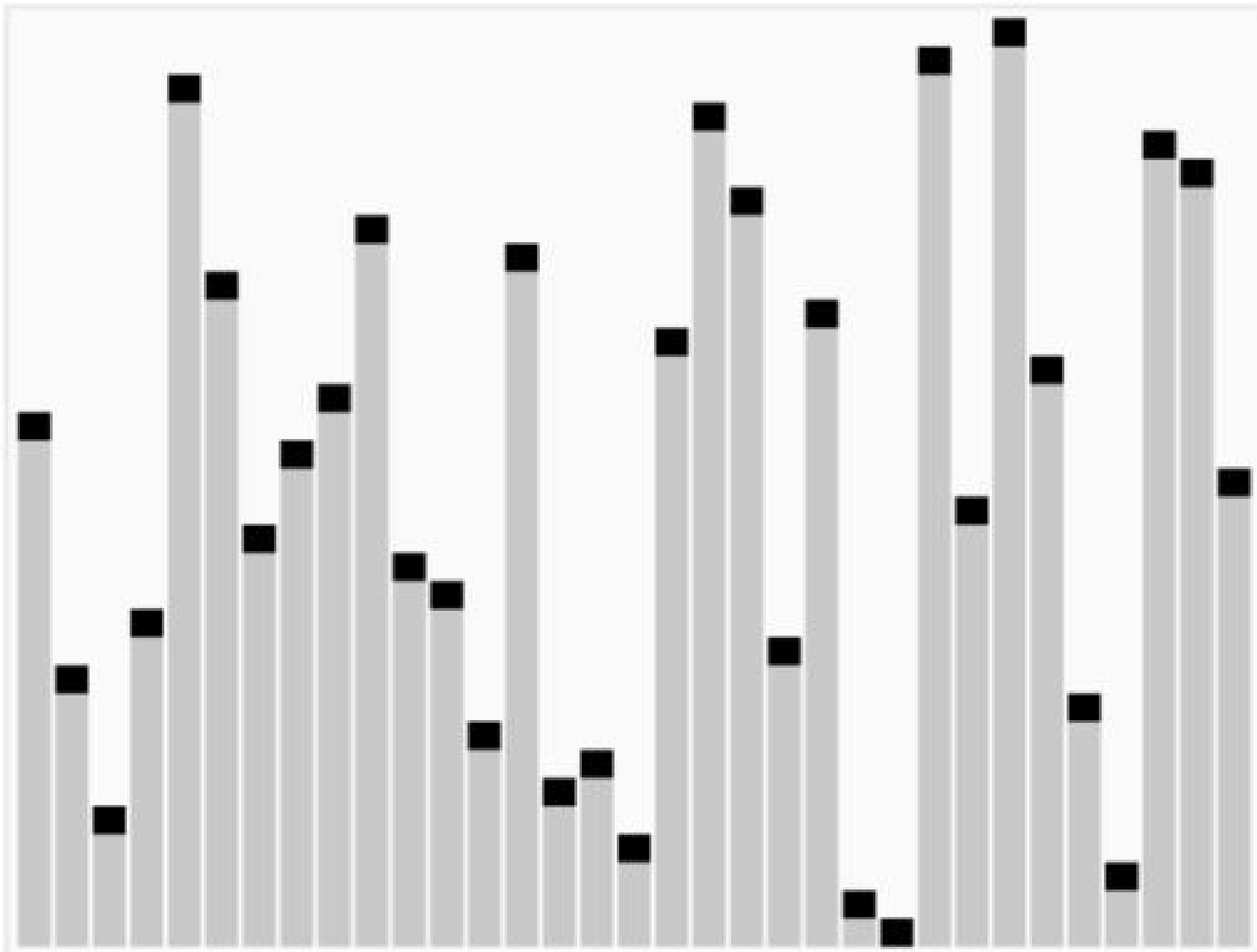
```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q − 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

快速排序算法示意图



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)：设主元素v被放在位置A(j)上。

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。

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)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1
if i == k            // the pivot value is the answer
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT(A, p, q − 1, i)
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。（而我们想找的是最大的元素）

- 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≤k≤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, …, n，定义指示器随机变量X_k:

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

假设A中元素是互异的，则有 $\mathrm{E}[X_k] = 1/n$

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，否则为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:

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

- 两边取期望：

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

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

这里,

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil, \\ n-k & \text{if } k \le \lceil n/2 \rceil. \end{cases}$$

在k=1~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($\left\lceil \frac{n}{2} \right\rceil + 1$) 到T(n-1)各项在总和中出现两次；

则有：

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

代换法证明：E[T(n)]=O(n).

- 即证明：存在常数c，使得E[T(n)]≤cn。
- 将上述猜测代入推论证明阶段有：

$$
\begin{aligned}
\mathrm{E}[T(n)] &\le \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} ck + an \\
&= \frac{2c}{n}\left(\sum_{k=1}^{n-1}k - \sum_{k=1}^{\lfloor n/2\rfloor-1} k\right) + an \\
&= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2\rfloor-1)\lfloor n/2\rfloor}{2}\right) + an \\
&\le \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an \\
&= \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{n^2/4-3n/2+2}{2}\right) + an \\
&= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an \\
&= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \\
&\le \frac{3cn}{4} + \frac{c}{2} + an \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right) .
\end{aligned}
$$

- 这里，为了证明E[T(n)]≤cn，须有cn/4-c/2-an≥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。

9.3 最坏情况是O(n)的选择算法

1）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

2）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$(r\geqslant 1)$。

（多余的$n-r\lfloor n/r\rfloor$个元素忽略不计）

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，

$1\leqslant i\leqslant\lfloor n/r \rfloor$，共得$\lfloor n/r \rfloor$个中间值（中位数）。

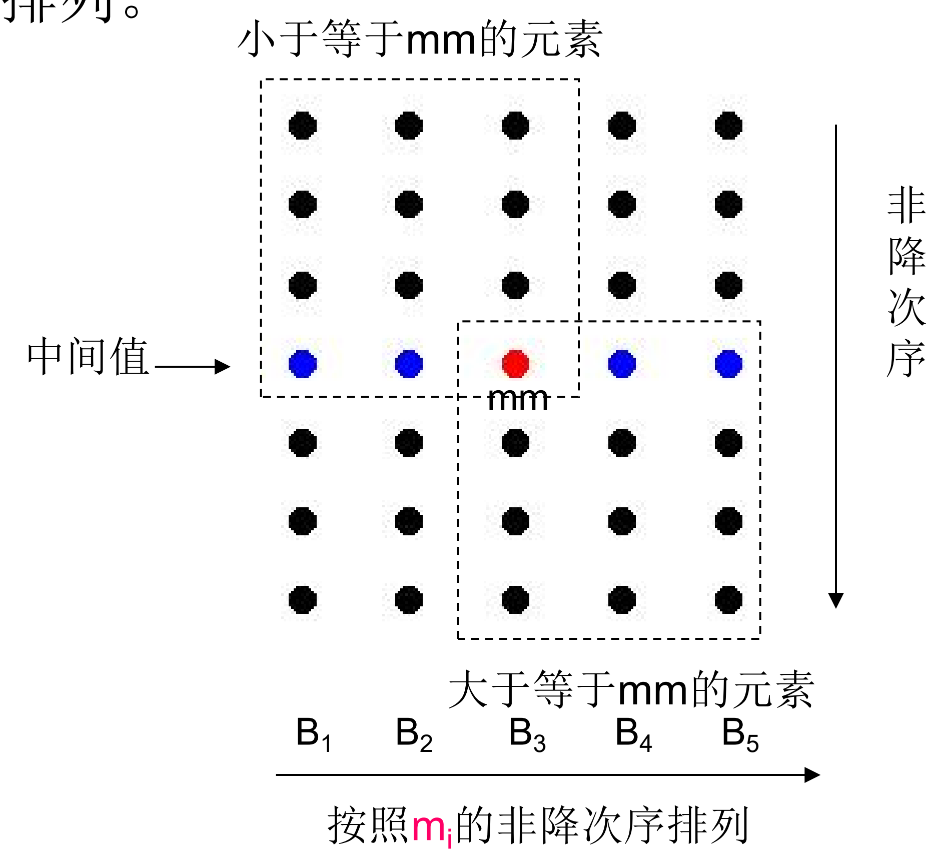
再后，对这$\lfloor n/r \rfloor$个中间值查找，再找出其中间值mm（中位数）。

最后，将mm作为划分元素执行划分。

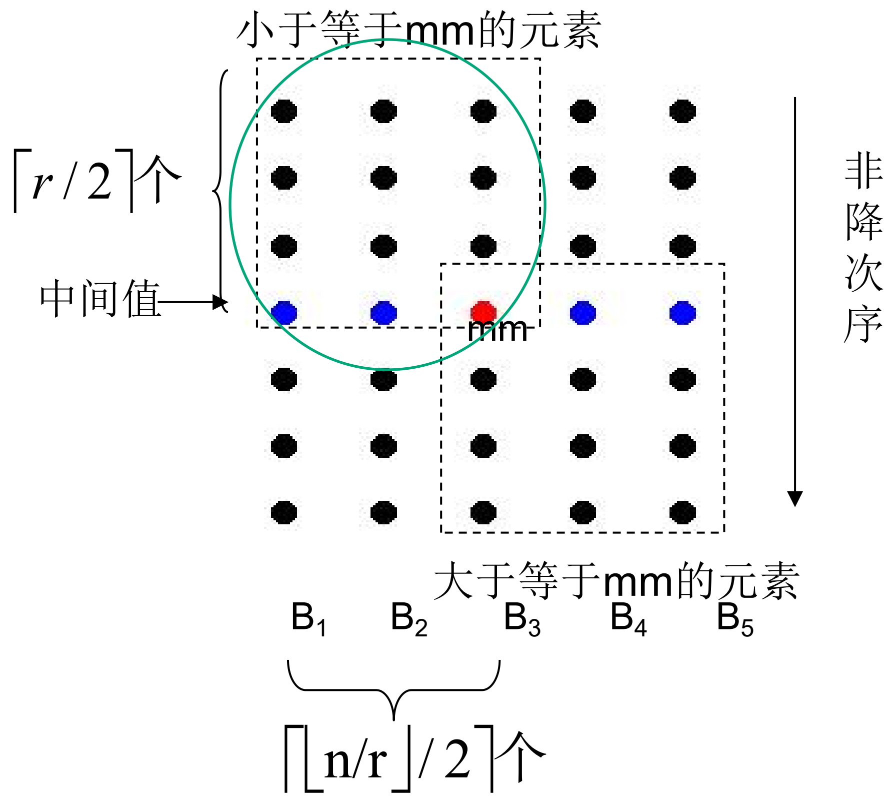
例：设 n=35，r=7。

- 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，$1 \leqslant i \leqslant 5$

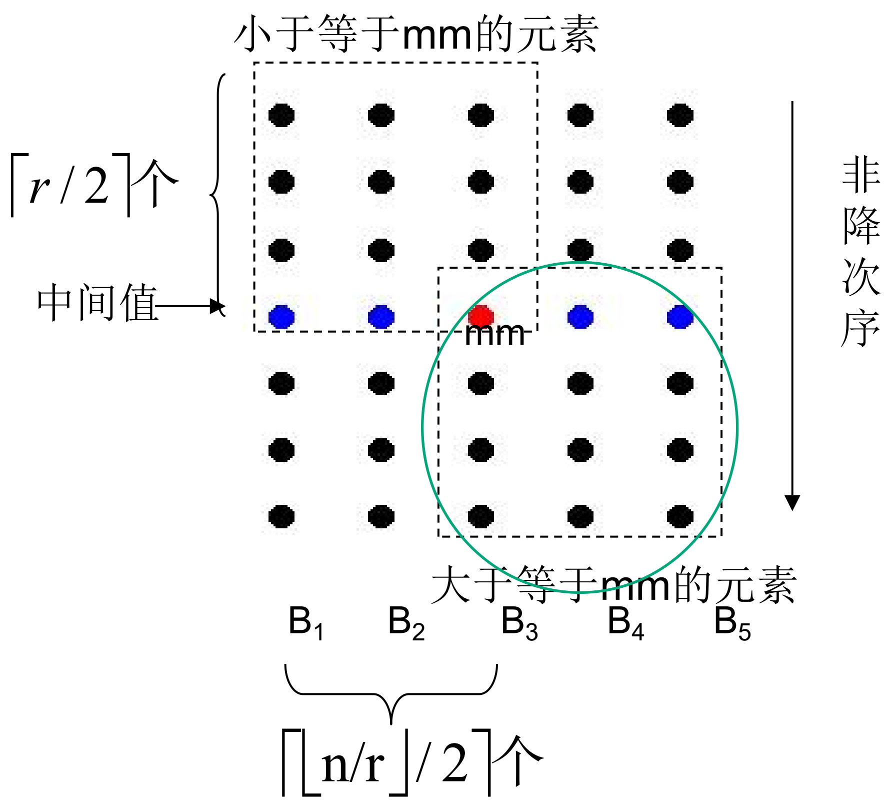
由图所示有：



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



同理，也至少有⌈r/2⌉⌈⌊n/r⌋/2⌉个元素大于或等于mm。



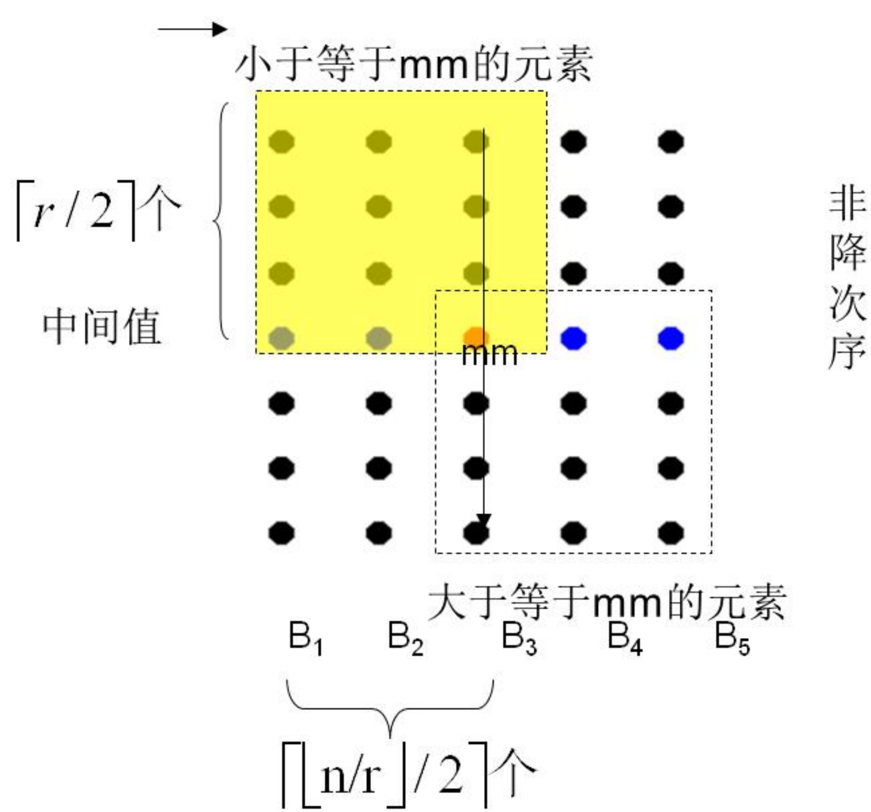
以r=5为例。使用两次取中间值规则来选择划分元素v(即mm)。

可得到，

- 至少有 $1.5\lfloor n/5 \rfloor$=0.3n-1.2个元素小于或等于选择元素v
- 且至多有 $n-1.5\lfloor n/5 \rfloor \le 0.7n+1.2$ 个元素大于等于v

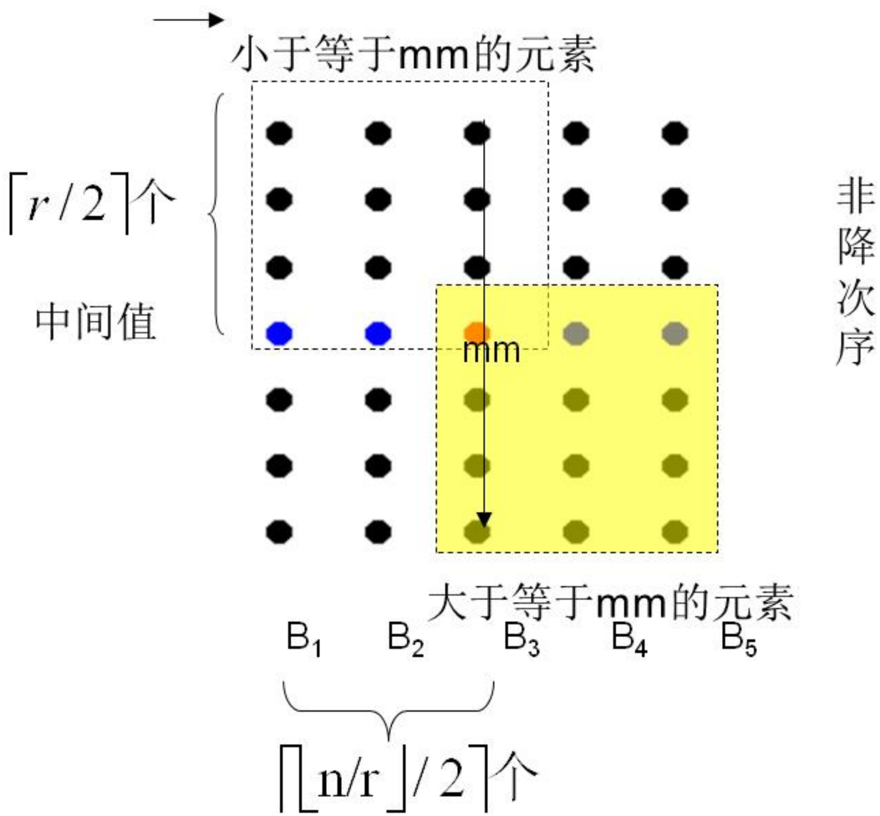
$$\boxed{\begin{aligned}&\lceil r/2 \rceil \lceil \lfloor n/r \rfloor /2 \rceil \\ &= \lceil 5/2 \rceil \lceil \lfloor n/5 \rfloor /2 \rceil \\ &\ge 3\lfloor n/5 \rfloor /2 = 1.5\lfloor n/5 \rfloor\end{aligned}}$$

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



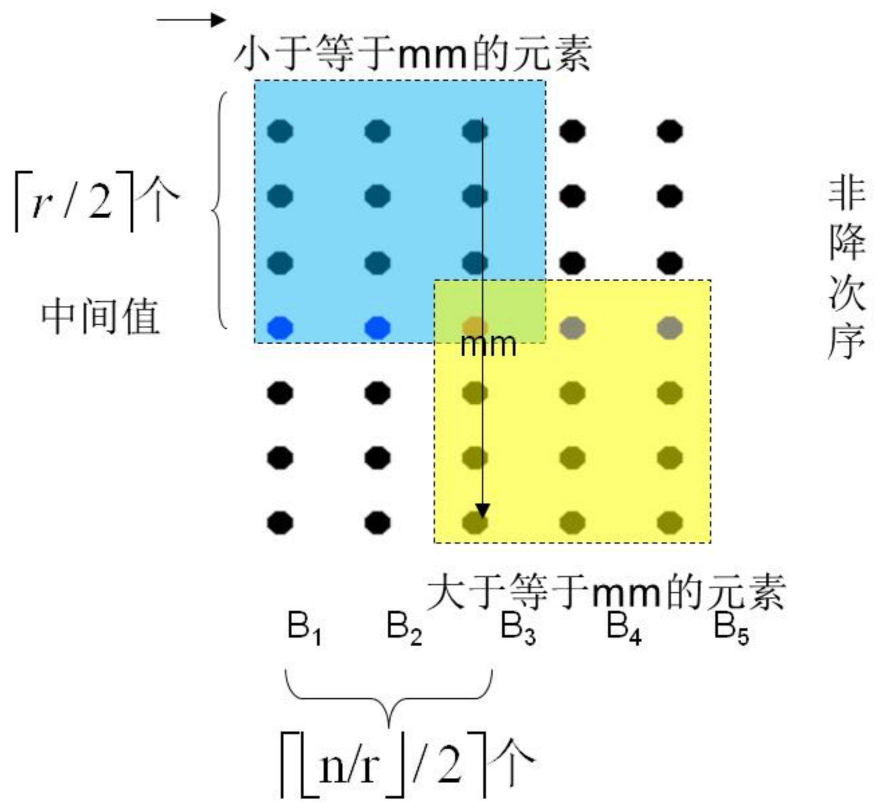
同理,

- 至少有$1.5\lfloor n/5 \rfloor = 0.3n-1.2$个元素大于或等于选择元素v
- 且至多有$n-1.5\lfloor n/5 \rfloor \le 0.7n+1.2$个元素小于等于v



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。则，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。



2）算法描述

算法　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≤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, …$m_{\lfloor n/r \rfloor}$ }是 $\lfloor n/r \rfloor$个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)

⑥ case

:i=j: return(v)

:i<j: 设S是A(1:j-1)中元素的集合; return(SELECT2(S, i, j-1))

:else: 设R是A(j+1:n)中元素的集合; return(SELECT2(R, i-j, n-j))

endcase

end SELECT2

SELECT2的时间分析：注，由于r为定值，所以这里视对r个元素的直接排序的时间为“定值”O(1)。

故有，

$$T(n) = \begin{cases} cn & n < 24, \\ T(n/5) + T(0.7n+1.2) + cn & n \geq 24 \end{cases}$$

（第4步，中位数）（第6步，递归调用）

用归纳法(代入法)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)。

进一步分析：

若A中有相同的元素时，上述结论T(n)=O(n)可能不成立。原因：

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，从而影响到算法的效率。

例如：设r=5,且A中有相同元素。不妨假设其中有0.7n+1.2个元素比v小，而其余的元素都等于v。

则，经过PARTITION，这些等于v的元素中除了v有**可能全部在落在S中**。

可得，步骤④和⑥所处理元素总数是

$$T(n/5)+T(n)>n$$

不再是线性关系。故有$T(n) \neq O(n)$

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	**0**	**0**	**0**	**1**	**1**	**1**	**1**
4	0	0	0	1	1	1	1
5	0	0	0	1	1	1	1

改进:

方法一：将A集合分成3个子集合U, S和R，其中U是由A中所有与v相同的元素组成，S是由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$|R| \le 0.7n+1.2$成立，故关于T(n)的分析仍然成立。即 $T(n) = O(n)$

方法二：选取其它r值进行计算

取r=9。重新计算可得，此时将有$2.5\lfloor n/9 \rfloor$个元素小于或等于v，同时至少有$2.5\lfloor n/9 \rfloor$大于或等于v。

则 当n≥90时，|S|和|R|都至多为

$$n-2.5\lfloor n/9 \rfloor+\tfrac{1}{2}(2.5\lfloor n/9 \rfloor)=n-1.25\lfloor n/9 \rfloor \leq 31n/36+1.25 \leq 63n/72$$

基于上述分析，有新的递推式：

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

用归纳法可证：

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

即，T(n)=O(n)成立

4）SELECT2的实现

算法中需要解决的两个问题

1）如何求子集合的中间值？

当r较小时，采用INSERTIONSORT直接对每组的r个元素排序，在排序好的序列中，中间下标位置所对应的元素即为本组中间元素。

2）如何保存$\left\lfloor \frac{n}{r} \right\rfloor$个子集合的中间值？

在各组找到中间元素后，将其调整到数组A的前部，按子集合的顺序关系连续保存。从而可方便用递归调用的方式对这些中间值进行二次取中，找出中间值的中间值。

算法3.11　SELECT2算法的实现

```
    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//
       v = A(j)
       call j ← PARTITION(A,v)//以v为主元进行划分
       case
         :j-m+1=k: return(j)
         :j-m+1>k: p←j-1  //左边
         :else: k←k-(j-m+1);m←j+1 //右边
      endcase
   repeat
  end SEL
```

同理，我们可以分析当划分为7个元素一组时（习题9.3-1），递归式为：

$$T(n) \le T(\lceil n/7 \rceil) + T(5n/7 + 8) + O(n)$$

当划分为3个元素一组时，递归式为：

$$T(n) \le T(\lceil n/3 \rceil) + T(2n/3 + 4) + O(n) ,$$

其时间复杂度变为O(nlgn)，所以3不是好的划分，7相对5来说划分元素太多，不太适合应用，所以，最终定为5个元素划分为一组，是较好的选择！

第三次作业：

习题：9.1-1、9.3-5、9-2

思考题

（1）9.3-1

（2）9.3-9（见课件）

（3）分金币（Spreading the Wealth, UVa 11300）

圆桌旁坐着n个人，每人有一定数量的金币，金币总数能被n整除。每个人可以给他左右相邻的人一些金币，最终使得每个人的金币数目相等。你的任务是求出被转手的金币数量的最小值。比如，n=4，且4个人的金币数量分别为1,2,5,4时，只需转移4枚金币（第3个人给第2个人两枚金币，第2个人和第4个人分别给第1个人1枚金币）即可实现每人手中的金币数目相等。

OJ题目（中位数）：POJ 1723、POJ 3579