華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

# 算法设计与分析

刘渝
Liu_yu@hust.edu.cn
2025秋季-华科-计算机
24级CS

# 算法分析与设计

## 第七章

## 快速排序

# 快速排序

## 快速排序是一种基于**划分**的排序方法

通过对待排序集合反复划分达到排序目的的算法称为快速分类算法。

## 划分

在待排序集合A中选取某元素t，按照与t的大小关系重新整理A中元素，使得整理后t被置于序列的某个位置上，而在t以前出现的元素均小于等于t，在t以后的元素均大于等于t。这一元素的整理过程称为划分（Partitioning)

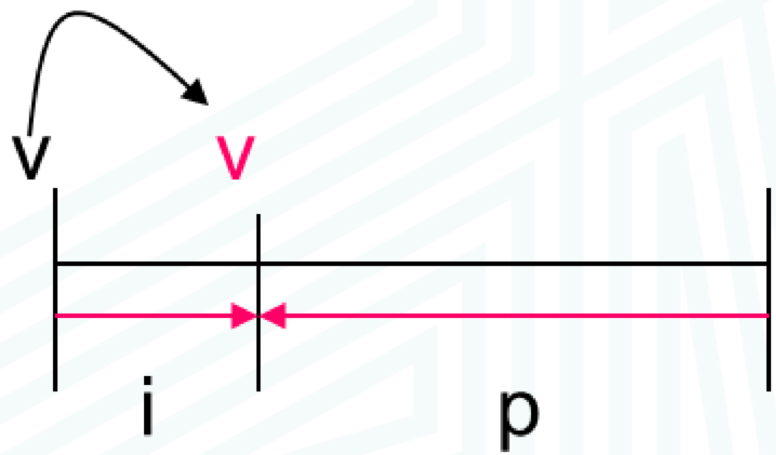
## 划分元素

元素t被称为“划分元素”（pivot，轴元素、主元素）

# 划分过程

```
procedure PARTITION(m,p)
    //用A(m)划分集合A(m:p-1)
    integer m,p,i;   global A(m:p-1)
    v←A(m); i←m //A(m)是划分元素//
    loop
        loop i←i+1 until A(i)≥v repeat     // i由左向右移//
        loop p←p-1 until A(p)≤v repeat  //p由右向左移//
        if i<p then
            call INTERCHANGE(A(i), A(p))
        else exit
      endif
    repeat
    A(m)←A(p)；  A(p)←v //划分元素在位置p//
end PARTITION
```

A(p)被定义，但为一限界值，不包含在实际的分类区间内。

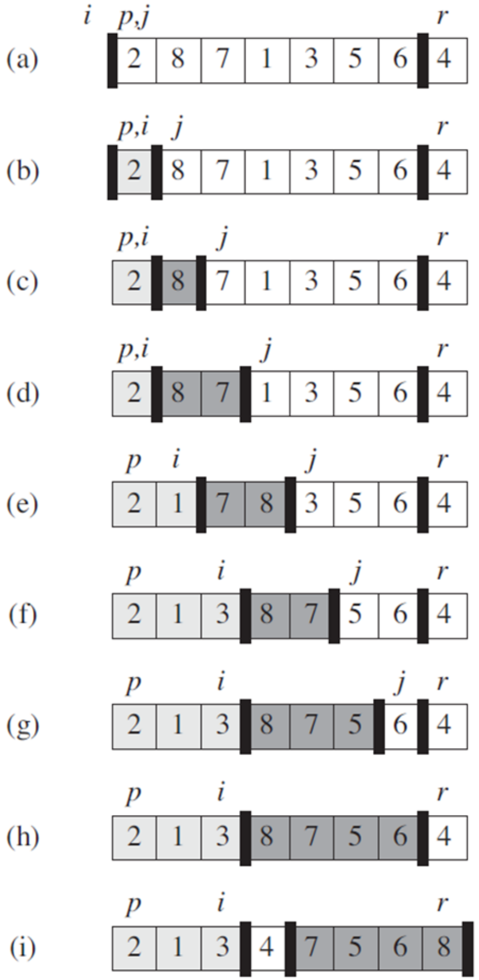


# 划分过程

```
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
```

- A[i+1~j-1]是大于x的元素区间；
- A[j]是紧挨其后且小于x的元素

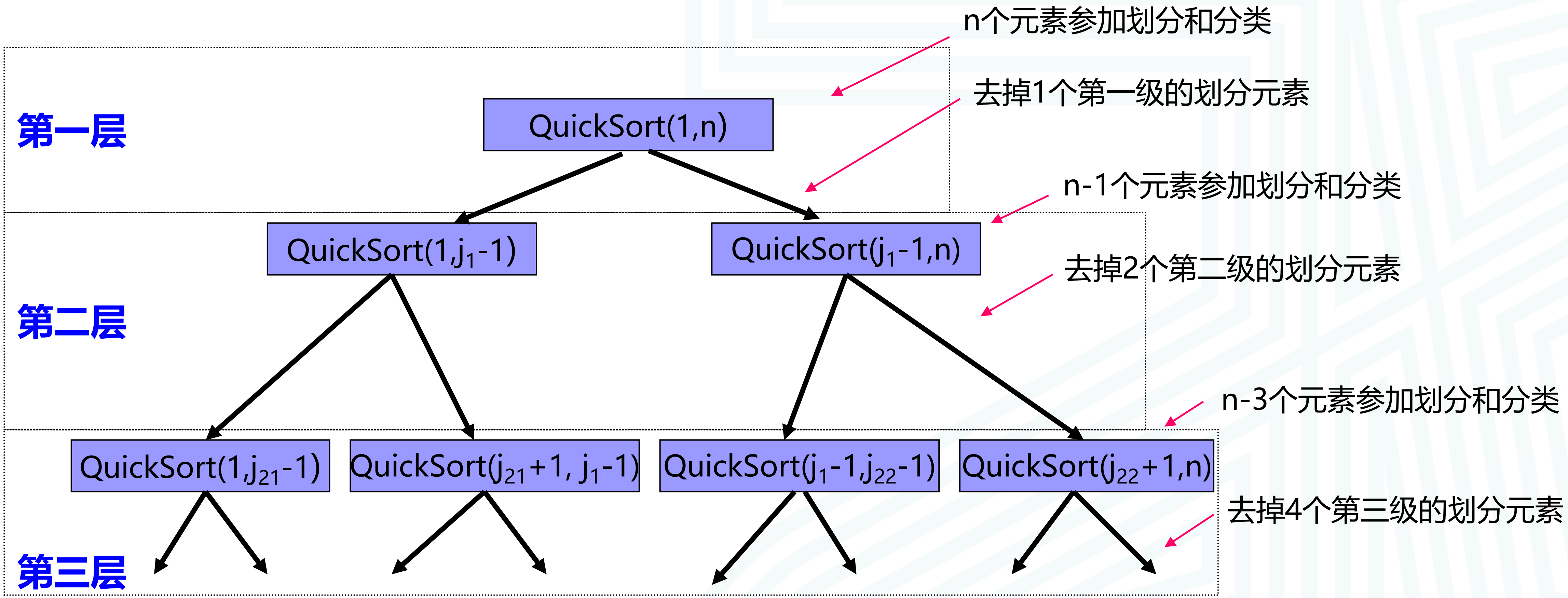
- i加1后，A[i]是前方大于x的第一个元素；
- 交换后，A[i+1~j]是前方大于x的元素区间；



# 完整过程

```
procedure QUICKSORT(p,q)
    //将数组A(1:n)中的元素A(p),…A(q)按递增的方式分类。
      A(n+1)有定义，且假定A(n+1)←+∞//
    integer p,q; global n,A(1:n)
    if p<q then  j←q+1                  //进入时，A(j)定义了划分区间[p,q]的上界，首次调用时j=n+1
      call PARTITION(p,j)                //出口时，j带出此次划分后划分元素所在的下标位置//
      call QUICKSORT(p,j-1)              //对前一子集合递归调用
      call QUICKSORT(j+1,q)              //对后一子集合递归调用
    endif
  end QUICKSORT
```

# 递归层次


第一层
QuickSort(1,n)
n个元素参加划分和分类
去掉1个第一级的划分元素
第二层
QuickSort(1,$j_1$-1)
QuickSort($j_1$-1,n)
n-1个元素参加划分和分类
去掉2个第二级的划分元素
第三层
QuickSort(1,$j_{21}$-1)
QuickSort($j_{21}$+1, $j_1$-1)
QuickSort($j_1$-1,$j_{22}$-1)
QuickSort($j_{22}$+1,n)
n-3个元素参加划分和分类
去掉4个第三级的划分元素

## 最坏情况

- 记**最坏情况**下的元素比较次数是$C_w(n)$;
- PARTITION一次调用中的元素比较数是p-m+1，若一级递归调用上处理的元素总数为r，则 PARTITION的比较总数为O(r)。

最坏情况下，每级递归调用的元素总数仅比上一级少1（如：第i次调用Partition所得的划分元素恰好是第i小元素），故$C_w(n)$是r由n到2的累加和。

即：$C_w(n) = \sum_{1<r\le n} r = O(n^2)$

## 平均情况

设调用PARTITION(m,p)时，所选取划分元素v恰好是A(m:p-1)中的第i小元素（1≤i≤p-m）的概率相等。则经过一次划分，所留下的待分类的两个子文件恰好是A(m:j-1)和A(j+1:p-1)的概率是：1/(p-m), m≤j<p。

记平均情况下的元素比较次数是$C_A(n)$；则有，

$$C_A(n) = n+1+\frac{1}{n}\sum_{1\le k\le n}(C_A(k-1)+C_A(n-k))$$

其中，n+1是第一次调用PARTITION时所需的元素比较次数。

$C_A(0) = C_A(1) = 0$

## 平均情况

化简上式可得：

$$
\begin{aligned}
C_A(n)/(n+1) &= C_A(n-1)/n + 2/(n+1) \\
&= C_A(n-2)/(n-1) + 2/n + 2/(n+1) \\
&= C_A(n-3)/(n-2)+2/(n-1)+2/n+2/(n+1) \\
&\dots \\
&= C_A(1)/2+ 2\sum_{3\le k\le n+1} 1/k
\end{aligned}
$$

由于 $\sum_{3\le k\le n+1} 1/k \le \int_2^{n+1}\frac{dx}{x} < \log_e(n+1)$

所以得，$C_A(n)<2(n+1)\log_e(n+1) = O(n\log n)$

华中科技大学计算机科学与技术学院
School of Computer Science &Technology,HUST

# 算法分析与设计
# 第九章
# 中位数和顺序统计量

# 目　　录

# 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）.*

# 基本概念：

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。

# 基本概念：

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。*

# 基本概念：

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**

## 基本概念：

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：$\lceil (4+1)/2 \rceil = 3$

## 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）

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

**输出**：元素$x \in A$，且A中恰好有$i-1$个其他元素小于它。


# 讨论：

## 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(nlogn)

## 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)

# 最大最小值

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?

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

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

n-1次，时间：O(n)

思考：这是求解上述问题的**最好结果**吗?

**是的!**

## Example：锦标赛算法

为了确定集合中的最小值，分多轮进行。每一轮中，元素之间两两一组，然后进行比较，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。
- **该求最小（最大）值算法是最优的。**

## Example：锦标赛算法

若**同时**找集合中的最大值和最小值，共需要多少次比较呢?

- 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。

**能不能更快一点?**

## Example:锦标赛算法

```
MAXMIN(A)
    max←min←A(1)    //设n为奇数
    for i=2 to A.length-1 step by 2
    do
        if(A[i] > A[i+1])
            max1= A[i];
            min1 = A[i+1];
        else
            max1= A[i+1];
            min1 = A[i];
        if max1 > max
            max=max1
        if min1 < min
            min=min1
     return max,min
```

如果n为偶数，用A[1]、A[2]对max和min进行初始化

- 成对比较。除了max和min的初始化，其余每对元素元素需3次比较即可。
  - 如果n为奇数，共需 $3\lfloor n/2 \rfloor$ 次比较；
  - 如果n是偶数，共需 3n/2-2 次比较。
- 总的比较次数至多是 $3\lfloor n/2 \rfloor$


# 问题：



若同时找集合中的最大值和次大值，又共需要多少次比较呢？

# 线性期望选择算法

## 借助QUICKSORT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
```

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

随机化的PARTITION过程

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

随机选择划分元素

```
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)
```

## 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)。

- 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。

## 找集合中的第i小元素

情况1：


A(j)即是第i小元素

# 找集合中的第i小元素

j-1元素　　　　n-j元素

情况2：1　　　　　　j　　　　n

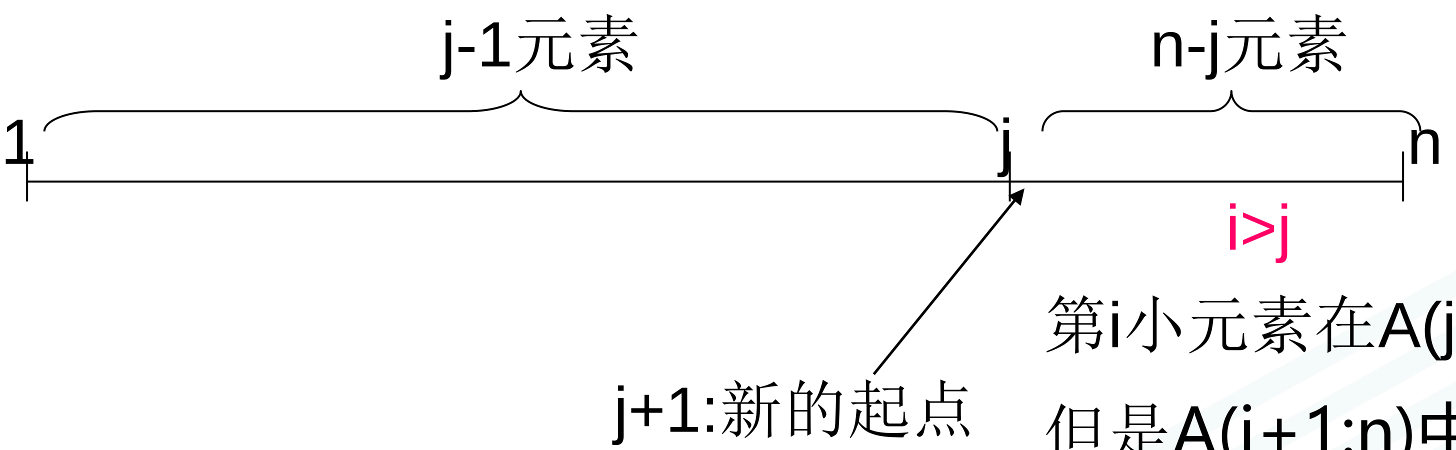
i<j

第i小元素在A(1:j-1)中,

且是A(1:j-1)中的第i小元素

# 找集合中的第i小元素

情况3：




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

## 找集合中的第i小元素

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。

## 期望时间证明

证明随机选择的期望运行时间是O(n)

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素。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素）的概率是1/(r-p+1)

划分后主元在位置q

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]是主元素。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。**

## 期望上界分析

- 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。

## 期望上界分析

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

- 两边取期望：

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

## 期望上界分析

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$T(\lfloor \frac{n}{2} \rfloor + 1)$ 到T(n-1)各项在总和中出现两次；

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

## 期望上界分析

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

代换法证明：E[T(n)]=O(n).

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

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

注：a是为去掉E[T(n)]中的O(n)而引入的常数

## 期望上界分析

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

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

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

- 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？

## 期望上界分析

- （续：cn/4-c/2-an≥0何时成立？）

即要求有：n(c/4-a)≥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≥2c/(c-4a)时，对任意的n有E[T(n)]≤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。**

## O(n)选择算法

## 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。

## 二次取中间值

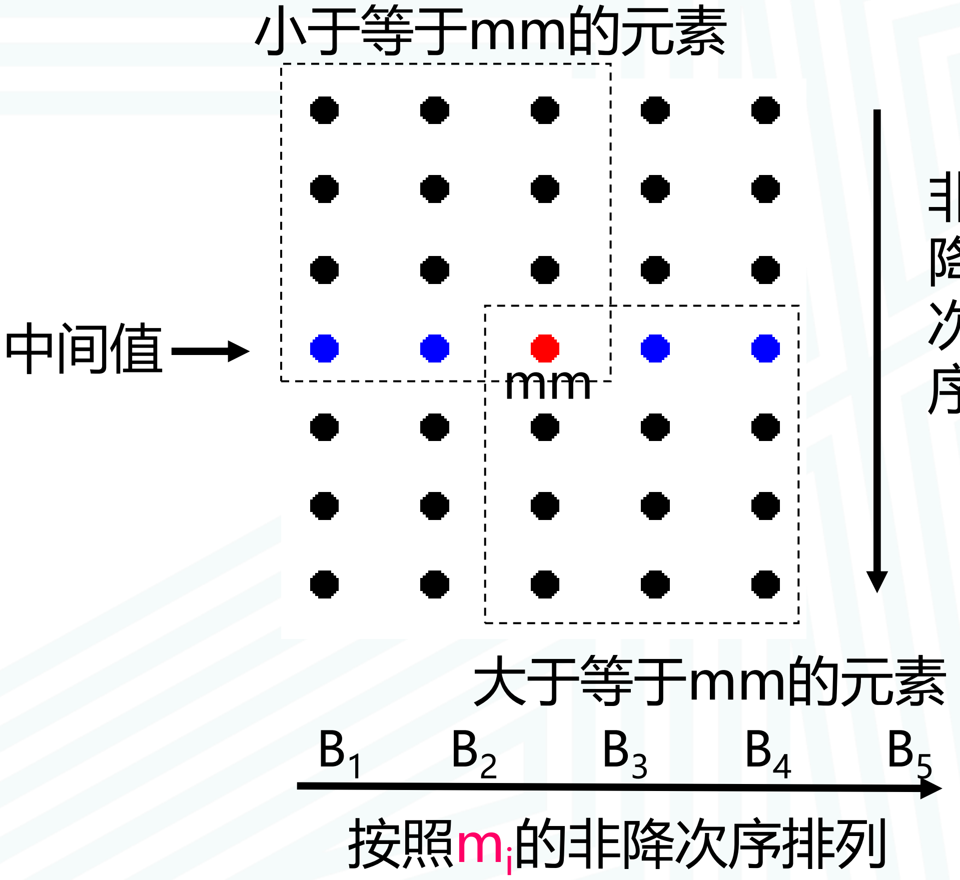
- 首先，将参加划分的n个元素分成$\lfloor n/r \rfloor$组，每组有r个元素(r≥1)。(多余的 $n-r\lfloor n/r \rfloor$ 个元素忽略不计)
- 然后，对这$\lfloor n/r \rfloor$组每组的r个元素进行排序并找出其中间元素$m_i$，$1 \le i \le \lfloor n/r \rfloor$，共得$\lfloor n/r \rfloor$个中间值（中位数）—— 一次取中。
- 再后，对这$\lfloor n/r \rfloor$个中间值查找，再找出其中间值mm（中位数）。—— 二次取中。

最后，将mm作为划分元素执行划分。

## *Example*

**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，$1 \le i \le 5$

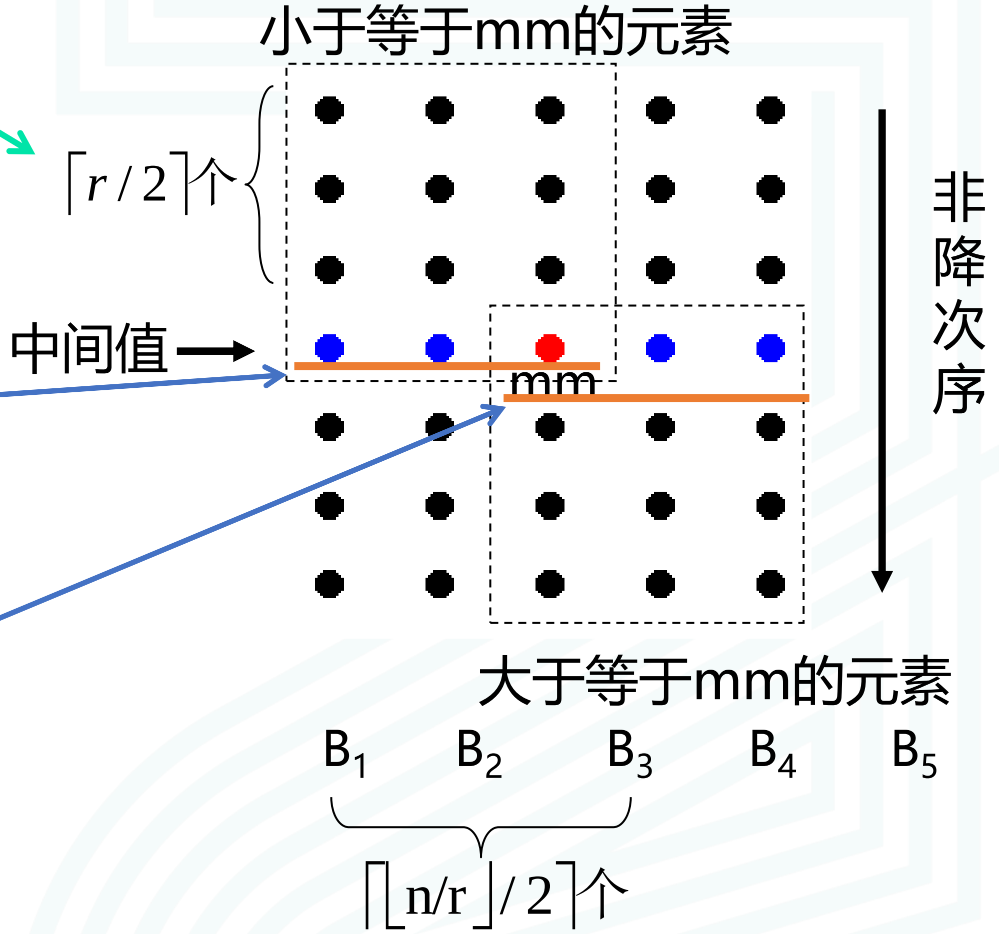


# Example

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;

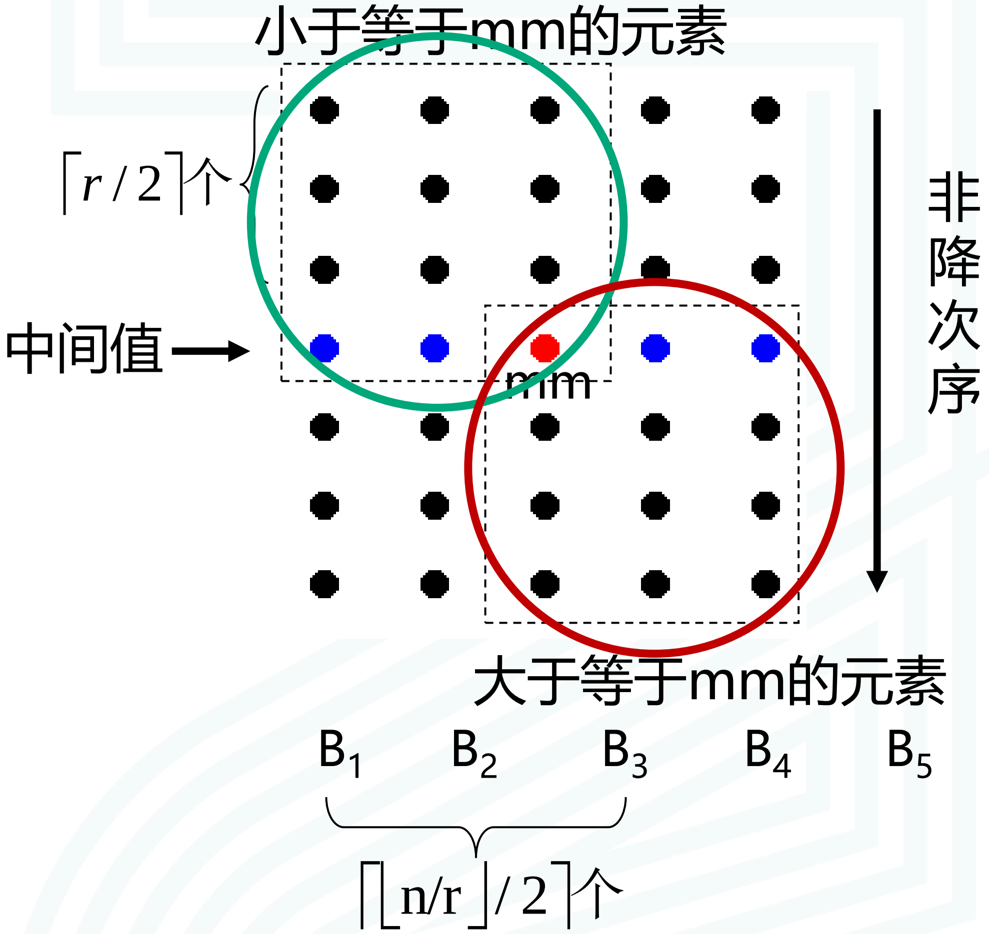
也至少有$\lfloor n/r \rfloor - \lceil \lfloor n/r \rfloor /2 \rceil + 1 \geq \lceil \lfloor n/r \rfloor /2 \rceil$个$m_i$大于或等于mm。



## *Example*

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 $\lceil r/2 \rceil \lceil \lfloor n/r \rfloor /2 \rceil$ 个元素大于或等于mm。



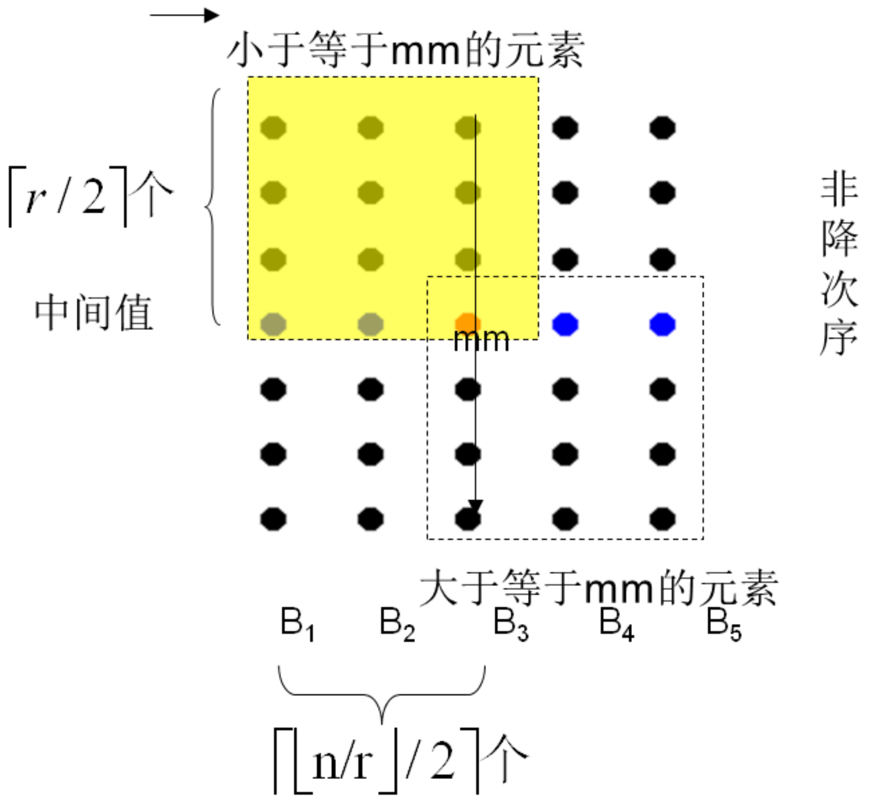
## Example

**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)

- 至少有 $1.5\lfloor n/5 \rfloor$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

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

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

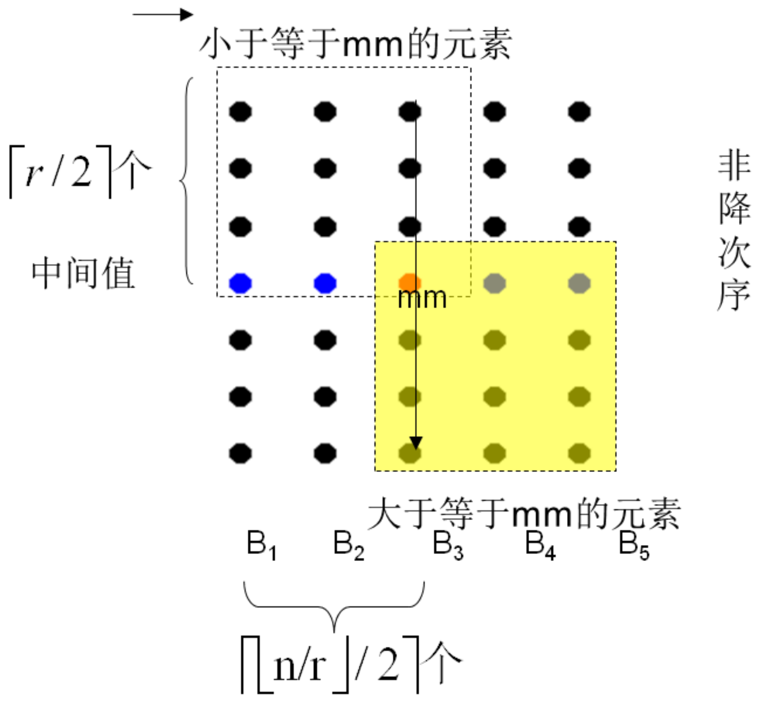


## Example

**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)

**同理：**

- 至少有 $1.5\lfloor n/5 \rfloor$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

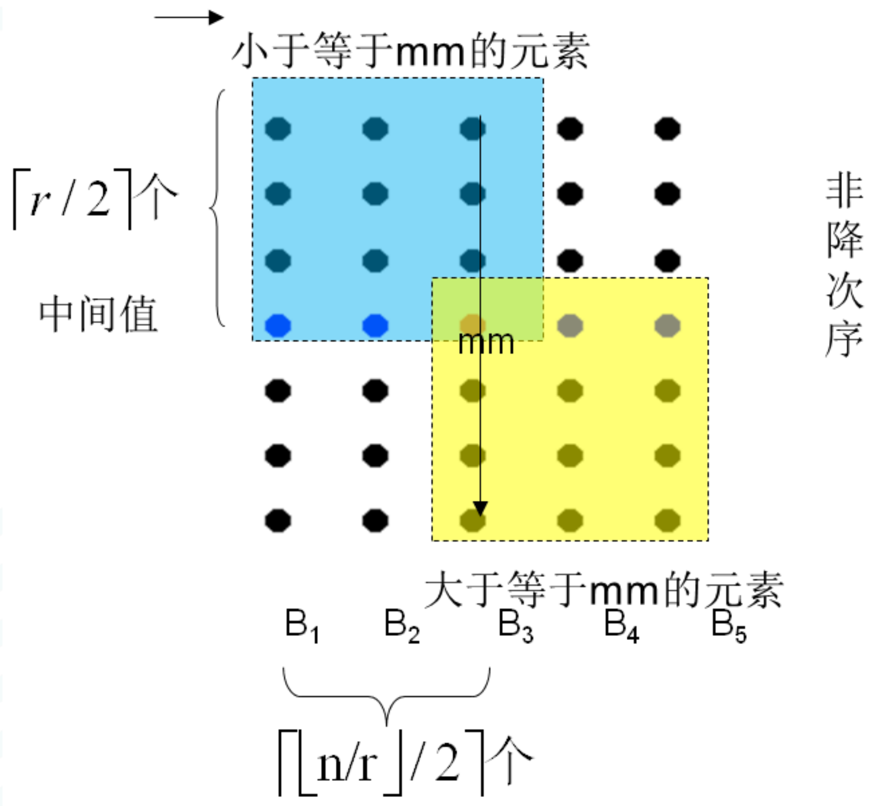


# O(n)选择算法

## *Example*

**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)

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



# 算法描述

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

# 时间分析

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　→ O（1）→ 将个数固定的r个元素的排序时间视为常数时间

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={$m_1$,$m_2$,...$m_{\lfloor n/r \rfloor}$}是$\lfloor n/r \rfloor$个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)，当n≥24

: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

# 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，

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

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

## 时间分析

出现了相同的元素。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。

# Example

## 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的元素中至多有一半可能在落在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，此时步骤④和⑥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$T(n) \ne O(n)$


# 思考：

如何恢复其O(n)的时间复杂度？

# 时间分析

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 $T(n) = O(n)$

## 时间分析

方法二：选取其它的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

相等元素的一半

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

基于上述分析，有新的递推式：

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

# 总结：

## 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。

# 伪代码实例

```
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//
```

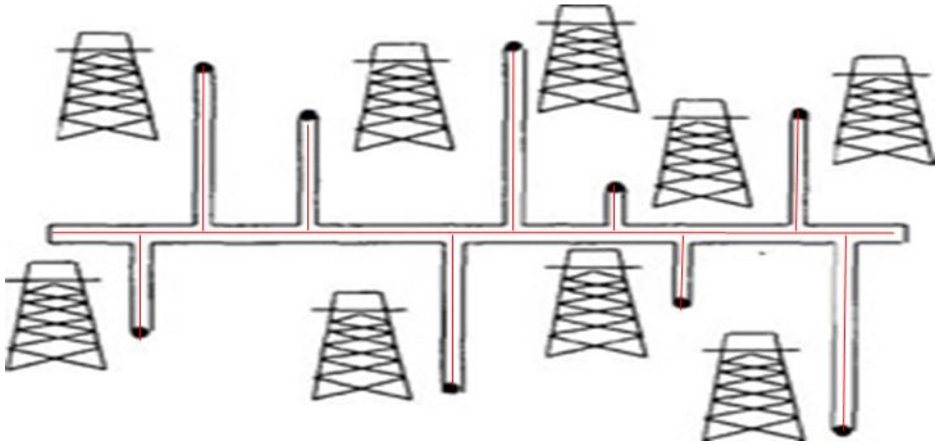
# 伪代码实例

**续：**

```
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
```

# 中位数问题　石油管的最优位置

Olay教授正在为一家石油公司咨询，公司正在计划建造一条由东向西的大型管道。该管道要穿过一个有n口井的油田。从每口井中都有一条喷油管沿最短路径与主管道直接相连(或南或北)，如图所示



问题：给定各口井的x坐标和y坐标。问，Olay教授如何选择主管道的最优位置，使得喷管长度总和最小？

# 中位数问题

## *Example*

1）由于主管道是由东向西的，因此要使相连油井与主管道的喷油管最短，喷油管方向必须南北相连，与主管道垂直，即主管道的最优位置应为一条 $y = y_k$ 的水平线。

即，问题的解是求最优位置$y_k$。

2）为了使$y_k$与各油井的y坐标 $y_1,……,y_n$ 间的距离和最短，我们将 $y_1$，…，$y_n$ 由小到大排序，选择最中间的那个点作为$y_k$。

即，主管道的最优位置是这n个油井的 y 坐标的中位数。

## Example

主管道最优位置(y坐标的中位数)：

- 若油井数为奇数，则第(n + 1)/2小的y坐标作为$y_k$；
- 若油井数为偶数，则第n/2小的y坐标值与第(n/2 + 1)小的y坐标值的平均数作为$y_k$的值。

疑问：

1）按照上述策略设计的主管道位置是最优的吗？

2）该最优位置可在线性时间内确定吗？

# 带权中位数

对分别具有正的权重$\omega_1, \omega_2, \ldots, \omega_n$且 $\sum_{i=1}^{n}\omega_i = 1$ 的n个不同元素$x_1, x_2, \ldots, x_n$，带权中位数是满足如下条件的元素$x_k$：

$$\sum_{x_i < x_k}\omega_i < \frac{1}{2} \quad 和 \quad \sum_{x_i > x_k}\omega_i \le \frac{1}{2}$$

所有小于 $x_k$ 的元素

所有大于 $x_k$ 的元素

隐含有序

# 带权中位数

## （1）一维空间上的问题

一条直线上有若干个带权的点$p_1,p_2,\dots,p_n$，它们的权重分别是$\omega_1, \omega_2,\dots, \omega_n$，在该直线上寻找一个点p，使得

$$\sum_{i=1}^{n}\omega_i d(p, p_i)$$

最小，其中*d(a,b)*表示点a与b之间的距离*d(a,b)=|a-b|*

——称点p为该n个点的一维带权中位数

## 带权中位数

由于各点被赋了权，因此上述的带权中位数p不一定是按递增排序后的$p_1,p_2,\ldots,p_n$中处于中间位置的那个点（甚至p不一定是$p_1,p_2,\ldots,p_n$ 中的一个），而是满足下述条件的点$p_k$：

在递增序列$p_1,\ldots,p_{k-1},p_k,p_{k+1},\ldots,p_n$中，子序列$p_1,\ldots,p_{k-1}$各点的权的和小于等于1/2，并且子序列$p_{k+1},\ldots,p_n$各点的权的和也小于等于1/2（这里$\sum_{i=1}^{n}\omega_i=1$），即

$$\sum_{x_i<x_k}\omega_i\le\frac{1}{2} \quad 和 \quad \sum_{x_i>x_k}\omega_i\le\frac{1}{2}$$

(试比较上述定义和前面的带权中位数的定义)

# Example

## 一维邮局问题

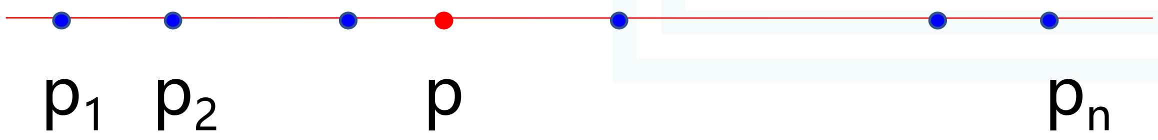
已知n个邮局分布在一条直线上，坐标点分别为$p_1$，$p_2$，…，$p_n$。一邮递员每天需要多次到这些邮局取邮件，设邮递员所处位置为点p。由于时间不一致，邮递员每次到一个邮局取件后需要先回到p点，然后再去下一个邮局。设邮递员每天到这些邮局的次数分别$\omega_1, \omega_2, \ldots, \omega_n$。

**问**，p设在哪里可使得邮递员每天到各个邮局走的总里程最短？

# Example

## 一维邮局问题

1）图示



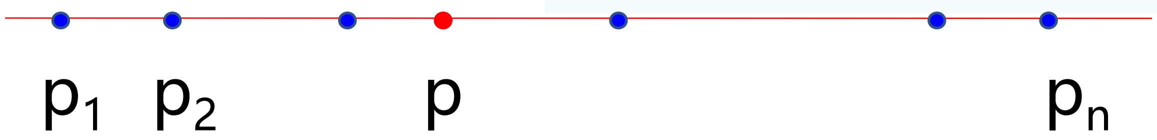

2）权重：邮递员每天需要到邮局的取件次数即为该问题的权重，可换算成为 ***[0..1]*** 值

3）里程：对邮局i，邮递员从p处出发到$p_i$处，每天的里程数为$\omega_i d(p,p_i)$，

这里，$d(p,p_i)=|p-p_i|$，代表p到$p_i$的距离（注：这里只考虑单向）；

## Example

### 一维邮局问题




该问题即是求 $\sum_{i=1}^{n} \omega_i d(p, p_i)$ 的最小值 —— 带权中位数问题

# 带权中位数

## （2）二维空间上的问题

设二维平面上分布着n个点 $p_1, p_2, \ldots p_n$，点 $p_i$ 的坐标用$(x_i,y_i)$表示，每个点附有一个权重$\omega_i$，$\sum_{i=1}^{n}\omega_i = 1$。定义点 $p_1(x_1,y_1)$ 与点 $p_2(x_2,y_2)$ 之间的距离是

$d(p_1, p_2)=|x1-x2|+|y1-y2|$ （称为Manhattan距离）

问题：在二维平面上找一个点p(x,y)，使得 $\sum_{i=1}^{n}\omega_i d(p, p_i)$ 最小，则称

p为该二维平面上n个点的带权中位数。

# 带权中位数

## （2）二维空间上的问题

由于$d(p_1, p_2)=|x_1-x_2|+|y_1-y_2|$，故可将问题转换为在x与y两个方向上分别求解带权中位数的问题，从而将二维问题转化为一维问题。

设最佳点为p，则满足：

$$\sum_{p_i<p,} \omega_i \le \frac{1}{2} \quad 和 \quad \sum_{p_i>p} \omega_i \le \frac{1}{2}$$

即带权中位数问题

# Example

## 二维邮局问题

一维邮局问题的推广：设这些邮局分布在二维平面上，邮局$p_i$的坐标记为$(x_i,y_i)$。最佳点记为p(x,y)。点之间的距离取Manhattan距离，即，

$$d(p_1, p_2) = |x_1-x_2| + |y_1-y_2|$$

**问**，p设在哪里可使得邮递员每天到各个邮局走的总里程最短?

**分析**：该问题即是求 $\sum_{i=1}^{n} \omega_i d(p, p_i)$ 的最小值

**二维带权中位数问题**


# 思考：



**为什么使** $\sum_{i=1}^{n} \omega_i d(p, p_i)$ **最小的点满足**

$$\sum_{p_i < p} \omega_i \le \frac{1}{2} \quad \text{和} \quad \sum_{p_i > p} \omega_i < \frac{1}{2}$$

# 证明

记d(i,j)是点i到点j的距离，令 d(i,j)=|num$_i$-num$_j$|，且有 d(i,j)=d(j,i)。

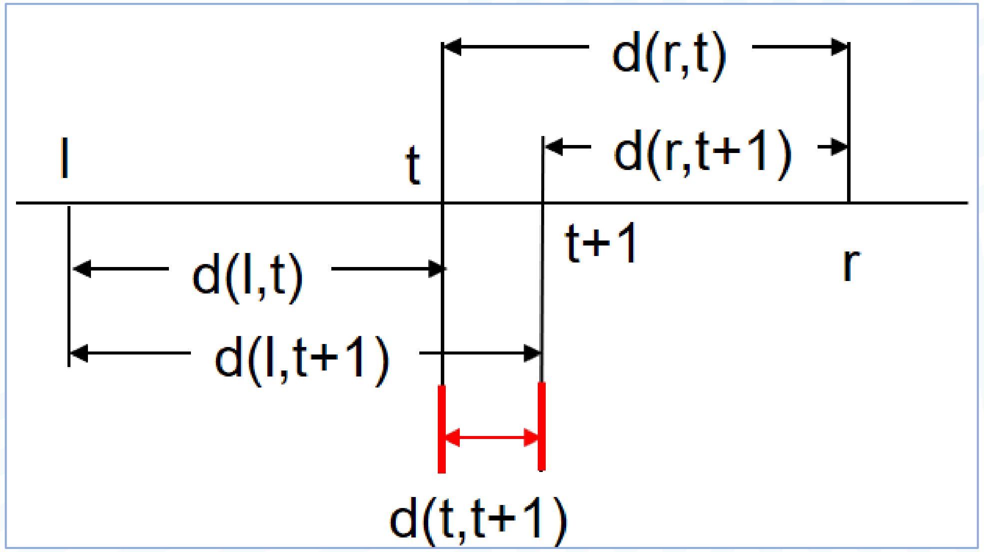
若最优点在t，则到其它点的带权距离均应大于等于到t的带权距离。

首先看t右边的点：不失一般性，取点t+1，则有：

$$\sum_{i\neq t}\omega_i d(i,t)\leq\sum_{i\neq t+1}\omega_i d(i,t+1)$$

进一步，上式可转化为：

$$\sum_{l<t}\omega_l d(l,t)+\sum_{t+1<r}\omega_r d(r,t)+\omega_{t+1}d(t+1,t)$$
$$\leq\sum_{l<t}\omega_l d(l,t+1)+\sum_{t+1<r}\omega_r d(r,t+1)+\omega_t d(t,t+1)$$



華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

# 中位数问题

## 证明

整理一下：

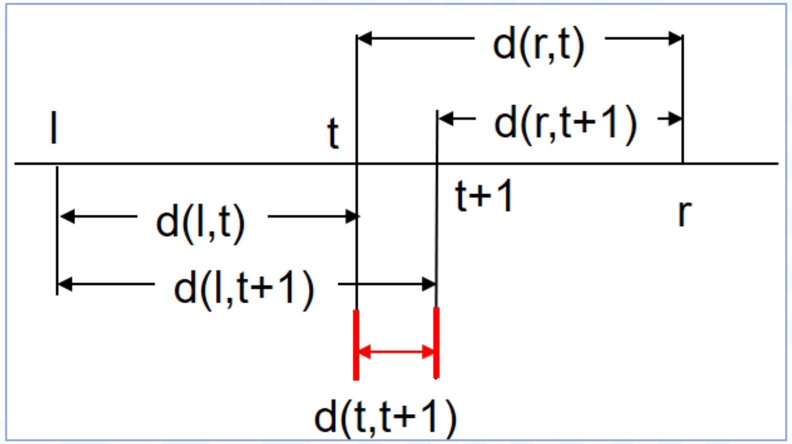
$$\sum_{t+1<r}\omega_r d(r,t) - \sum_{t+1<r}\omega_r d(r,t+1) + \omega_{t+1} d(t+1,t)$$

$$\le \sum_{l<t}\omega_l d(l,t+1) - \sum_{l<t}\omega_l d(l,t) + \omega_t d(t,t+1)$$

进一步有：

$$\sum_{t+1<r}\omega_r \left(\underline{d(r,t)-d(r,t+1)}\right) + \omega_{t+1} d(t,t+1)$$

$$\le \sum_{l<t}\omega_l \left(\underline{d(l,t+1)-d(l,t)}\right) + \omega_t d(t+1,t)$$




而，

$$\underline{d(l,t+1)-d(l,t)} = d(t,t+1)$$

$$\underline{d(r,t)-d(r,t+1)} = d(t+1,t)$$

且，$d(t,t+1) = d(t+1,t)$

因此：

$$\sum_{l\le t}\omega_l \ge \sum_{t+1\le r}\omega_r$$

即：

$$\sum_{l<t}\omega_l + \omega_t \ge \sum_{t<r}\omega_r$$

## 证明

$$\sum_{l<t} \omega_l + \omega_t \geq \sum_{t<r} \omega_r$$

因此，若t是最优点，则必有其左边的权值之和加上$\omega_t$后大于右边的权值之和。

同理，取t左边的点t-1，与上述讨论类似，可以证明其右边的权值之和加上$\omega_t$后也大于左边的权值之和，即

$$\sum_{t<r} \omega_r + \omega_t \geq \sum_{l<t} \omega_l$$

## 证明

此时，点的选择已经和具体的距离没有关系了，主要和各点的位置和权值相关。

因为左边的权值之和+$\omega_t$ ≥ 右边的权值之和，所以：

$$\sum_{l<t}\omega_l+\omega_t \ge \sum_{t<r}\omega_r=\sum_{i=1}^{n}\omega_i-(\sum_{l<t}\omega_l+\omega_t)$$

$$\Rightarrow 2*(\sum_{l<t}\omega_l+\omega_t)\ge\sum_{i=1}^{n}\omega_i$$

$$\Rightarrow 2*\sum_{t<r}\omega_r\le\sum_{i=1}^{n}\omega_i \longrightarrow \sum_{t<r}\omega_r<\frac{1}{2}$$

同理可得：

$$2*\sum_{l<t}\omega_l\le\sum_{i=1}^{n}\omega_i \longrightarrow \sum_{l<t}\omega_l<\frac{1}{2}$$

令$\sum_{i=1}^{n}\omega_i=1$

**证毕**。 （这正是带权中位数所具备的性质）