数据分析与算法设计
第4章 减治法
（Decrease-and-Conquer）

李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn

上节回顾：蛮力法

- 蛮力法：基于问题的描述和所涉及的定义直接求解
  - 计算a^n：连乘n-1次
  - 基于定义的矩阵乘法算法
  - 求最大公约数：连续整数检测算法

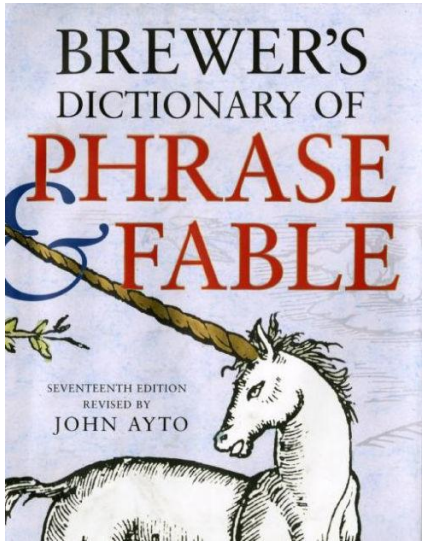
just do it

- 明显的缺陷：效率不高

- 存在的意义：
  - 简单性和广泛的适用性
  - 基线算法，可作为基础衍生更高效的算法
  - 仍可解决规模小的问题（“杀鸡焉用牛刀”）

普卢塔克说，萨特斯为了教士兵明白毅力和智慧比蛮力更重要的道理，就把两匹马带到他们面前，然后让两个人拔光马尾。一个人是魁梧的大力士，他抓住尾巴拔了又拔，但一点儿效果也没有；另一个人是一个瘦削、长脸的裁缝，他微笑着，每次拔掉一根毛，很快就把尾巴拔得光秃秃的。[①]

——科巴姆·布鲁尔



减治法

- 基本思想：利用一个问题给定实例的解和同样问题较小实例的解之间的某种关系，将问题的规模递减，然后基于小规模问题的解，得到大规模问题的解
  - 求解的过程可以是自顶向下（递归），也可以是自底向上（迭代）

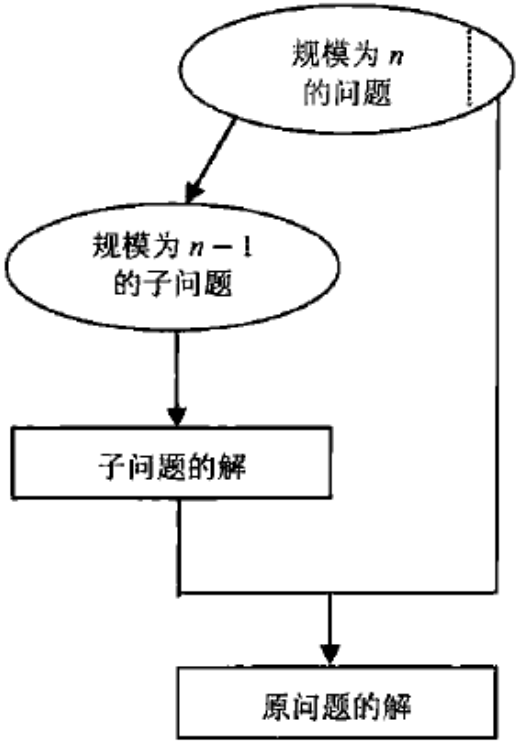
- 3种主要的“减”形式
  - 减常量：如减1，每次迭代规模 n -> n-1
  - 减常因子：如减半，每次迭代规模 n -> n/2
  - 减可变规模：如 gcd(m,n)= gcd(n,m mod n)

减治法：减常量

- 减（一）治：计算 a^n
  - 递归方式

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n > 0, \\ 1 & \text{if } n = 0, \end{cases}$$

  - 非递归方式
    - 自底向上，把a自乘n−1次
    - 等效为蛮力的做法

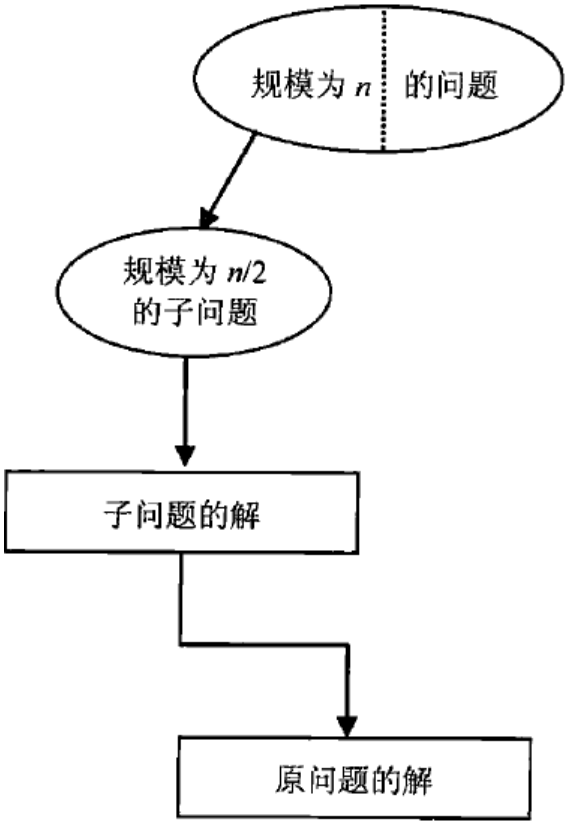


减治法：减常因子

- 减（半）治：计算 a^n

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive,} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd,} \\ 1 & \text{if } n = 0. \end{cases}$$

  - 时间效率：$\Theta(\log n)$

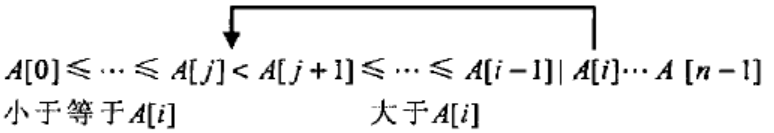


目录

- 减常量算法
  - 插入排序
  - 拓扑排序
  - 生成组合对象的算法
- 减常因子算法
- 减可变规模算法

插入排序

- 算法思想：减（一）治
  - 假设对较小数组A[0:i-1]已完成排序，则可从后往前扫描该有序子数组，直到遇到第一个小于或等于A[i]的元素，然后把A[i]插在该元素的后面

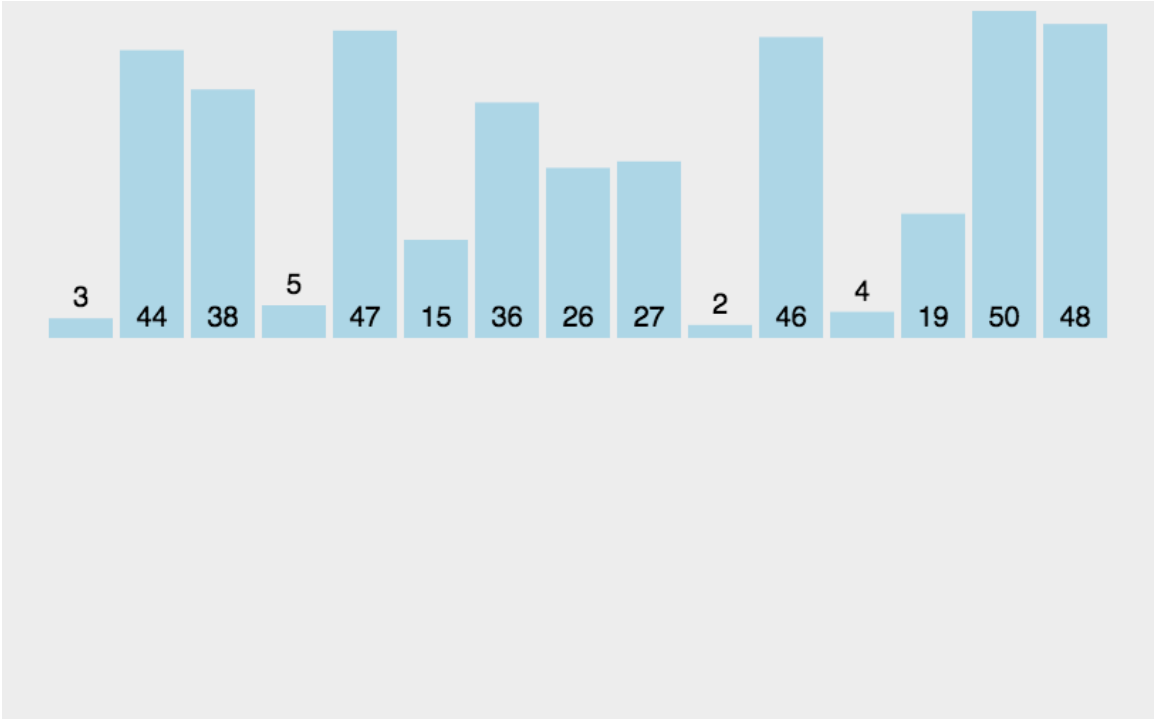


- 算法伪代码

```
算法 InsertionSort(A[0..n − 1])
    //用插入排序对给定数组排序
    //输入：n 个可排序元素构成的一个数组 A[0..n − 1]
    //输出：非降序排列的数组 A[0..n − 1]
    for i ← 1 to n − 1 do
        v ← A[i]
        j ← i − 1
        while j ≥ 0 and A[j] > v do
            A[j + 1] ← A[j]
            j ← j − 1
        A[j + 1] ← v
```

插入排序

- 动态示例



- 时间效率
  - 输入规模：n
  - 基本操作：比较（A[j]>v）

插入排序

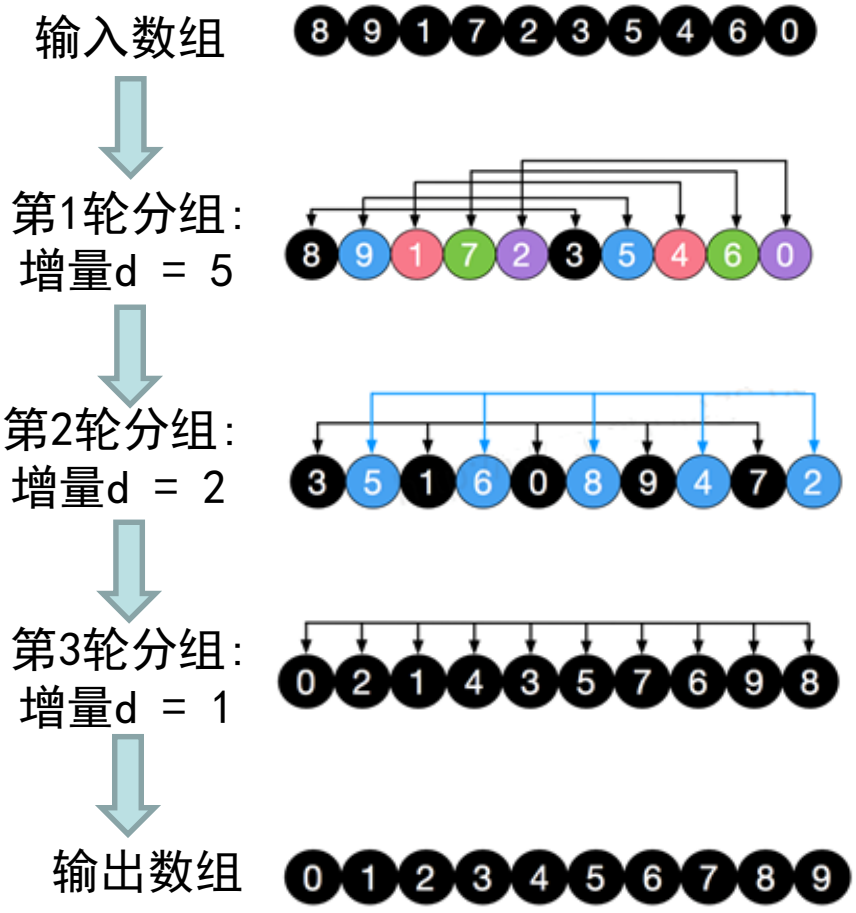
- 时间效率：取决于比较操作的次数
  - 最差：$C_{worst}(n) = \sum_{i=1}^{n-1}\sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2).$ 输入为严格递减数组
  - 最好：$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n-1 \in \Theta(n).$ 输入为已排好序的数组
  - 平均：$C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2).$ 输入为任意随机的数组

能否进一步优化算法，使得平均效率突破$O(n^2)$？

希尔排序

Donald Shell
(Computer Scientist, 1924-2015)

- 多轮分组插入排序
  - 分组方式取决于增量序列
  - 优先对距离较远的元素进行大小的比较
- 平均效率：$O(n^{1.3})$



未完待续…

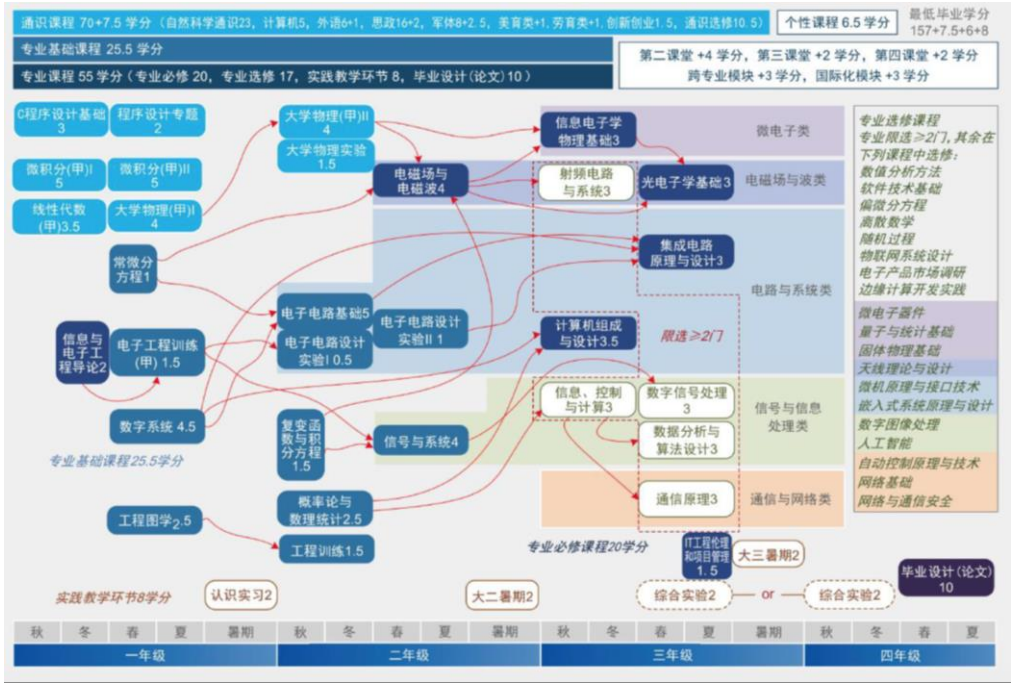
排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

目录

- 减常量算法
  - 插入排序
  - 拓扑排序
  - 生成组合对象的算法
- 减常因子算法
- 减可变规模算法

拓扑排序

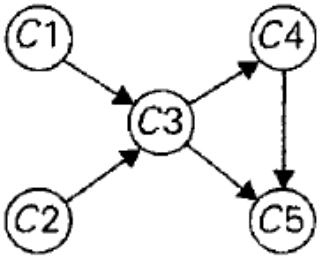
- 应用举例：
  - 在大学学习的过程中，各门课程的学习是有先后顺序的，有些课程需要先修课程，有些则不需要；有些**课程之间有先后的关系**，有些课程则可以并行的进行。要求**确定一个学习方案使得各门课程的学习能够有序进行**



- 拓扑排序
  - 对于给定的无环有向图，要求按照某种顺序列出它的顶点序列，使图的每一条边的起点总在结束顶点之前

拓扑排序

- 算法1：基于DFS
  - 执行DFS遍历，记录顶点变成端点（出栈）的顺序
  - 将出栈顺序倒过来就得到拓扑排序的一个解



(a)

$C5_1$
$C4_2$
$C3_3$
$C1_4$ $C2_5$

(b)

出栈次序：
C5, C4, C3, C1, C2
拓扑排序表：
C2 C1→C3→C4→C5

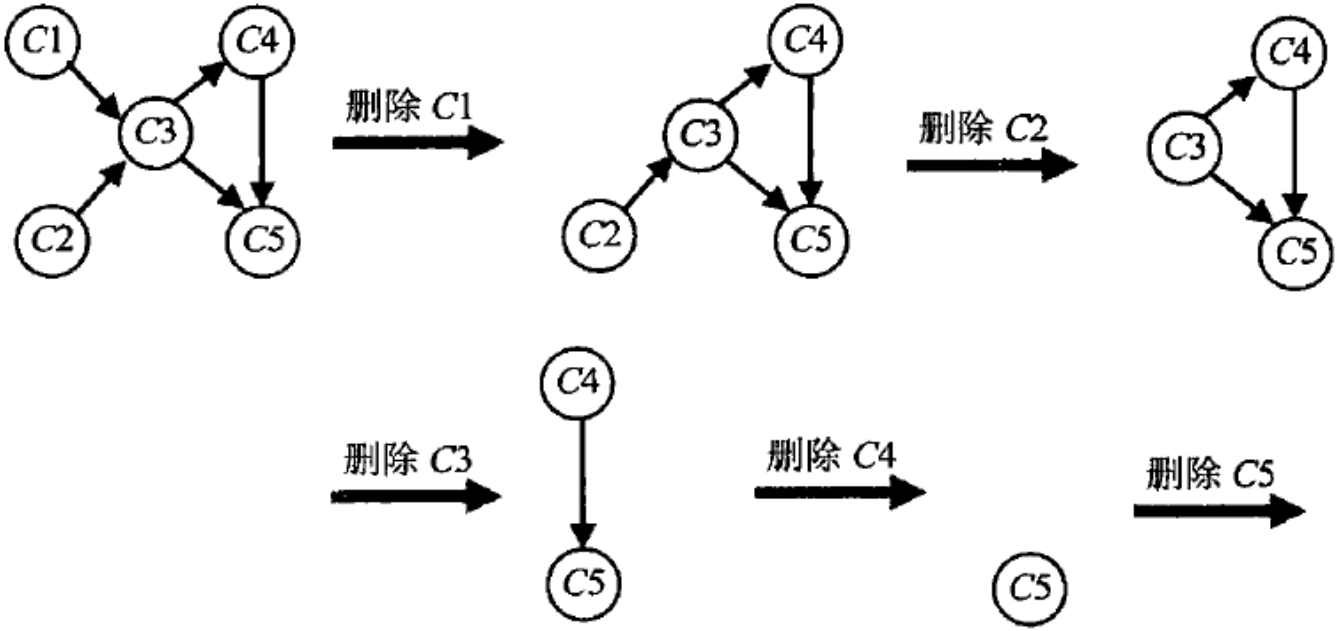
(c)

(a)需要求拓扑排序问题的有向图；(b)DFS 遍历栈，下标数字指出出栈的次序；(c)该问题的解

  - 注意：拓扑顺序不唯一

拓扑排序

- 算法2：减（一）治



求得的解是 C1，C2，C3，C4，C5

拓扑排序问题的源删除算法的图示。在每次迭代时，没有输入边的节点从有向图中删除

目录

- 减常量算法
  - 插入排序
  - 拓扑排序
  - 生成组合对象的算法
- 减常因子算法
- 减可变规模算法

生成组合对象的算法

- 组合对象
  - 排列、组合
  - 给定集合的子集
- 组合数学
  - 离散数学分支
  - 专门研究组合对象
  - 关心各种计算组合数量的方程

{a,b,c}

{},
{a}, {b}, {c},
{a,b}, {a,c}, {b,c},
{a,b,c}

生成排列

- 对于给定的多个元素，生成各种可能的排列
- 假设元素的下标为1, 2, …, n，简化为下标的排列问题

- 介绍三种生成方法
  - 插入法
  - Johnson-Trotter法
  - 字典顺序法

生成排列：插入法

- 减（一）治：生成{1, …, n−1}的排列，在此基础上，对每个排列，插入n，从而得到{1, …, n}的排列
- 插入顺序的技巧
  - 从右到左
  - 从左到右
  - 从右到左
  - …交替改变方向

开始	1
从右到左将 2 插入 1	12　　21
从右到左将 3 插入 12	123 132 312
从左到右将 3 插入 21	321 231 213

- 好处：最小变化
  - 相邻的2个排列：只需交换某2个相邻元素即可

生成排列：Johnson-Trotter法

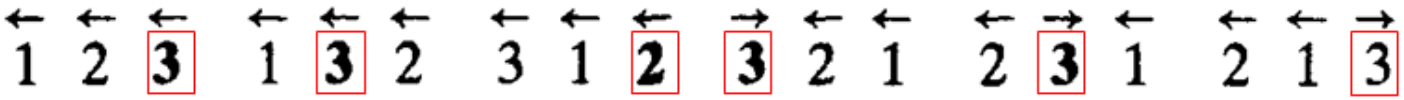
- 不需要知道规模为n−1的排列，直接得到规模为n的排列结果
- 算法思想
  - 给每个元素赋予一个方向（以箭头标记）
  - 移动元素：若元素k的箭头指向一个相邻的较小元素，则该元素可移动
  - 通过迭代地交换移动元素和它箭头指向的相邻元素，并调整部分元素的箭头方向即可

$\overrightarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$

算法 JohnsonTrotter(n)
//实现用来生成排列的 Johnson-Trotter 算法
//输入：一个正整数 n
//输出：$\{1, \cdots, n\}$的所有排列的列表
将第一个排列初始化为 $\overleftarrow{1}\ \overleftarrow{2} \cdots \overleftarrow{n}$
while 存在一个移动元素 **do**
　　求最大的移动元素 k
　　把 k 和它箭头指向的相邻元素互换
　　调转所有大于 k 的元素的方向
　　将新排列添加到列表中

生成排列：Johnson-Trotter法

- 实例（n=3）



- 时间效率为$\Theta(n!)$
  - 不是算法的“错”，而是这个问题要求生成的项太多
- J-T算法生成的排列仍然具有“最小变化”特性
- 但是它生成的排列并不是“字典序”
  - n=3的字典序：123 132 213 231 312 321
  - n=4的字典序：1234 1243 1324 1342 … 4321

生成排列：字典顺序法

- 算法思想
  - 假设要生成$a_1, a_2, \ldots, a_{n-1}, a_n$后面的排序，有两种可能
    - 如果$a_{n-1} < a_n$，则交换这两个元素的位置
    - 如果$a_{n-1} > a_n$，则
      - 找到序列的最长递减后缀$a_{i+1} > a_{i+2} \ldots > a_n$，但$a_i < a_{i+1}$
      - 将a_i与后缀中大于它的最小元素进行交换，以使a_i增大
      - 再将新后缀颠倒，变成递增排列

```
算法  LexicographicPermute(n)
      //以字典序产生排列
      //输入：一个正整数 n
      //输出：在字典序下{1, …, n}所有排列的列表
      初始化第一个排列为 12 … n
      While 最后一个排列有两个连续升序的元素 do
          找出使得 a_i < a_{i+1} 的最大的 i   //a_{i+1} > a_{i+2} > … > a_n
          找到使得 a_i < a_j 的最大索引 j   //j ≥ i+1，因为 a_i < a_{i+1}
          交换 a_i 和 a_j   //a_{i+1} a_{i+2} … a_n 仍保持降序
          将 a_{i+1} 到 a_n 的元素反序
          将这个新排列添加到列表中
```

362541 ⇨ 364125

生成子集

- 集合A的“幂集”：以A的所有子集为元素构成的集合
- 减（一）治生成子集

n	子集
0	$\varnothing$
1	$\varnothing$ $\{a_1\}$
2	$\varnothing$ $\{a_1\}$ $\{a_2\}$ $\{a_1, a_2\}$
3	$\varnothing$ $\{a_1\}$ $\{a_2\}$ $\{a_1, a_2\}$ $\{a_3\}$ $\{a_1, a_3\}$ $\{a_2, a_3\}$ $\{a_1, a_2, a_3\}$

  - 特点：挤压序（Squashed Order）
    - 所有包含a_j的子集必须紧排在所有包含$a_1, a_2 \ldots a_{j-1}$的所有子集后面

生成子集

- 位串法生成幂集
  - 每个子集与一个n位二进制串$b_1, b_2 \ldots b_n$对应，只有当$b_i = 1$时, a_i属于该子集

- 以n=3为例

位串	000	001	010	011	100	101	110	111
子集	$\varnothing$	$\{a_3\}$	$\{a_2\}$	$\{a_2, a_3\}$	$\{a_1\}$	$\{a_1, a_3\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_3\}$

- 上述位串生成的子集不满足“最小变化”特点（即，每个子集跟它的直接前趋之间的区别，要么是增加了一个元素，要么是删除了一个元素）

生成子集

- 格雷编码位串

000　001　011　010　110　111　101　100

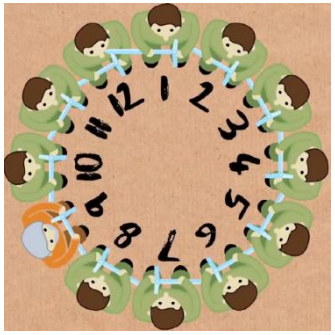
```
算法　BRGC(n)
      //递归生成 n 位的二进制反射格雷码
      //输入：一个正整数 n
      //输出：所有长度为 n 的格雷码位串列表
      if  n=1 ，表 L 包含位串 0 和位串 1
      else 调用 BRGC(n-1)生成长度为 n-1 的位串列表 L1
           把表 L1 倒序后复制给表 L2
           把 0 加到表 L1 中的每个位串前面
           把 1 加到表 L2 中的每个位串前面
           把表 L2 添加到表 L1 后面得到表 L
      return L
```

目录

- 减常量算法
  - 插入排序
  - 拓扑排序
  - 生成组合对象的算法
- 减常因子算法
- 减可变规模算法

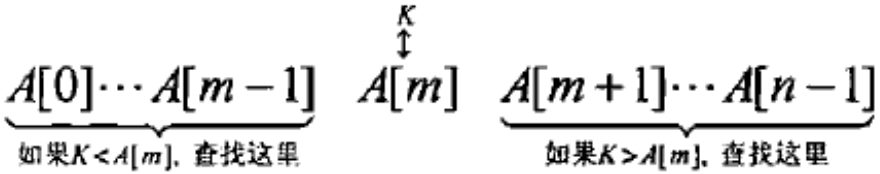
减常因子算法

- 思想：每次迭代问题规模减去一个相同的常数因子
- 问题实例：
  - 折半查找
  - 假币问题
  - 约瑟夫斯问题
  - 俄式乘法（课后阅读）



折半查找（二分查找）

- 问题：给定查找键K，在一个升序数组A中，查找确定跟查找键K匹配的元素位置
- 算法思想：
  - 通过比较查找键K和数组中间元素A[m]来完成查找工作
    - 如果它们相等，算法结束
    - 如果K<A[m]，就对数组前半部分执行该操作
    - 如果K>A[m]，就对数组后半部分执行该操作



折半查找

- 算法伪代码

```
算法  BinarySearch(A[0..n − 1], K)
      //实现非递归的折半查找
      //输入：一个升序数组 A[0..n − 1]和一个查找键 K
      //输出：一个数组元素的下标，该元素等于 K；如果没有这样一个元素，则返回 − 1
      l ← 0 ;  r ← n−1
      while  l ≤ r do
            m ← ⌊(l + r)/2⌋
            if K = A[m]  return m
            else if  K < A[m] r ← m − 1
            else  l ← m + 1
       return − 1
```

K=70

下标	0	1	2	3	4	5	6	7	8	9	10	11	12
值	3	14	27	31	39	42	55	70	74	81	85	93	98
迭代 1	*l*						*m*						*r*
迭代 2								*l*		*m*			*r*
迭代 3								*l*,*m*	*r*				

折半查找

- 时间效率
  - 基本操作：比较
  - 最差效率$\Theta(\log n)$：
    - 递推公式：$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 1.$
    - 当$n = 2^k$时：$C_{worst}(2^k) = k + 1 = \log_2 n + 1.$
    - 对于一般n：$C_{worst}(n) = \lfloor \log_2 n \rfloor + 1 = \lceil \log_2 (n+1) \rceil.$
  - 平均效率

$$C_{avg}(n) \approx \log_2 n$$

如果是考虑“三分”查找呢？效率会更高吗？

假币问题

- 有n个外观相同的硬币，其中一个是假币。假币的重量比真币要轻一点。假定有一架天平，需设计高效的算法（用最少的天平使用次数）来检测出假币

- 减（n/2）治
  - 思路：把n个硬币分为两堆，每堆$\lfloor n/2 \rfloor$个，称一次，可确定假币在哪一堆 （如果n为奇数，就留下一枚多余的硬币）
  - 效率：
    - 基本操作：称重次数
    - 最差效率：$W(n) = W(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1, \quad W(1) = 0.$

$$\Rightarrow W(n) = \lfloor \log_2 n \rfloor.$$

假币问题

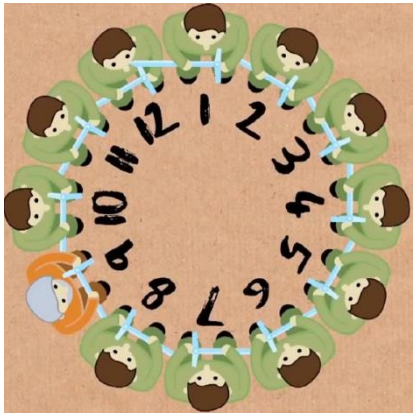
- 改进：
  - 把n个硬币分为三堆，每堆$\lfloor n/3 \rfloor$个，每次称任意两堆，可确定假币在哪一堆
- 举例（n=9）
  - 分成3组，每组3个硬币
  - 第1次称重：将其两组放在天平的两边，如果某组重量轻，则假币在其中；如果两组重量一致，则假币在剩余的另一组中
  - 第2次称重：在包含假币的组中，拿出2个放在天平的两边。如果有一个轻，则其为假币；如果两个等重，则剩下的一个就是假币
- 时间效率：$\log_3 n$

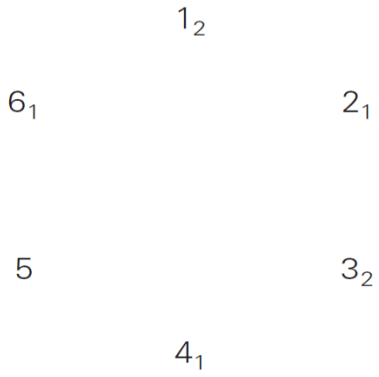
约瑟夫斯问题(Josephus Problem)

- 问题背景
  - 以犹太历史学家弗拉瓦斯·约瑟夫斯命名
  - 约瑟夫斯和他的40个战友被罗马军队包围在洞中。他们讨论是自杀还是被俘，最终决定自杀
  - 约瑟夫斯建议每个人应该轮流杀死他旁边的人，而这个顺序以抽签的方式决定
  - 约瑟夫斯有预谋地抓到了最后一签，作为洞穴的两个幸存者之一，他说服了对方，一起投降罗马军队

约瑟夫斯问题

- 问题数学描述
  - 设有n个以1、2、…、n编号的人，按编码顺序围成一圈，从1号开始报数，每数到m就淘汰一人，请问最后被淘汰的人是几号？（$L(n,m)$ =?）

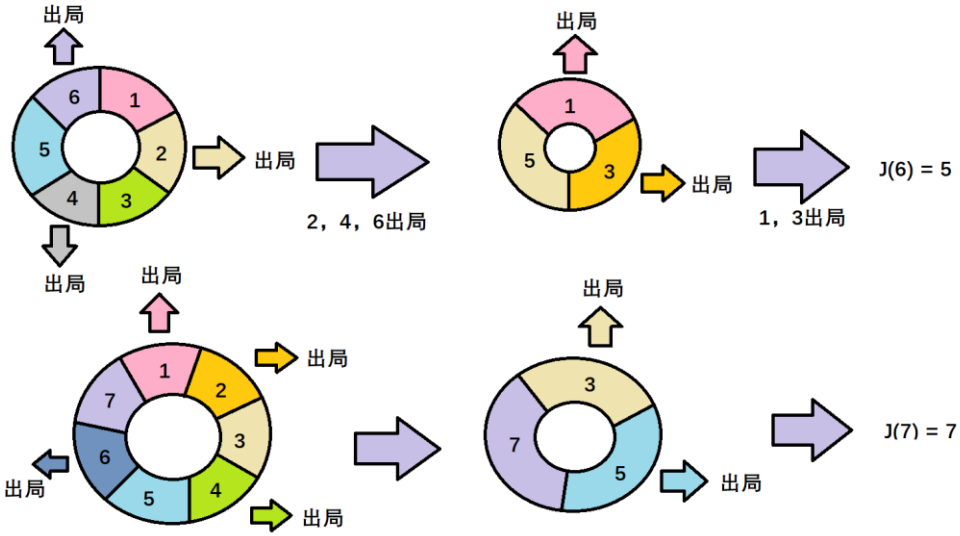




$$L(6,2)=5$$

约瑟夫斯问题

- 当m=2时，应用减治法：
  - n为偶数(n=2k)：J(2k)=2J(k)−1
  - n为奇数(n=2k+1)：J(2k+1)=2J(k)+1



- 根据递推公式求解闭合表达式？其它方法？

约瑟夫斯问题

- 数学归纳法：L(n, 2)=2b+1
  - $n = 2^a + b$，a尽可能大
    - 例如当n=100时，a=6，b=36

L(6,2)=2×2+1=5, // $6=2^2+2$, b=2

L(13,2)= 2×5+1=11, // $13=2^3+5$, b=5

L(41,2)= 2×9+1=19, // $41=2^5+9$, b=9

- 用二进制表示n，对n做一次向左移位

约瑟夫斯问题

- 当m＝3、4、…时，问题的解是什么？
- 存在一个 L(n, m)递推公式
  - L(1,m)=1
  - L(k+1,m)=(L(k,m)+m) mod (k+1)

- 减（一）治思想！

目录

- 减常量算法
  - 插入排序
  - 拓扑排序
  - 生成组合对象的算法
- 减常因子算法
- 减可变规模算法

减可变规模算法

- 每次迭代，问题规模的变化模式不同
- 应用实例
  - 计算中值和选择问题
    - 选择问题：求一个n个数列表的第k个最小元素
    - 中值问题：选择问题特例 $k = \lceil n/2 \rceil$
  - 二叉查找树的查找和插入
  - 拈游戏
  - 插值查找（课后阅读）
  - …

计算中值和选择问题

- 基线方案：排序后选择第k个元素
- 划分算法：
  - 利用快速排序的分区思想，将数组根据某个值p进行划分整理，使得左边部分包含小于或等于p的元素，接着是中轴(pivot)p元素，然后是所有大于或等于p的元素

  - 其中一种实现方式：Lomuto划分

计算中值和选择问题

- Lomuto划分

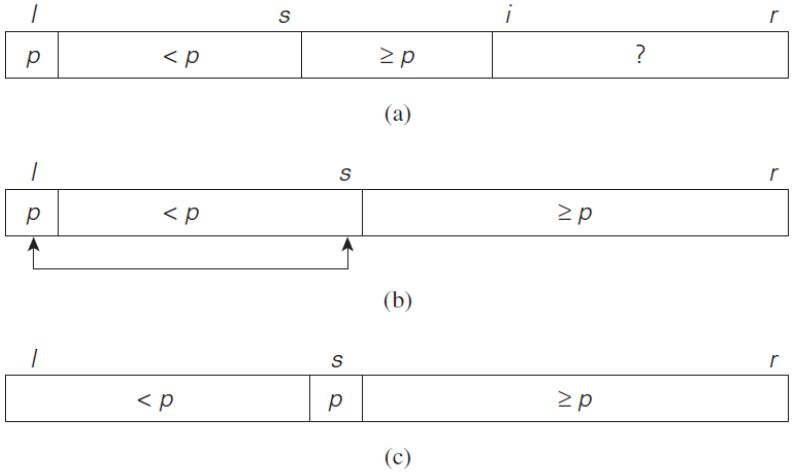


FIGURE 4.13 Illustration of the Lomuto partitioning.

算法 LomutoPartition($A[l..r]$)

//采用 Lomuto 算法，用第一个元素作为中轴对子数组进行划分

//输入：数组 $A[0..n-1]$ 的一个子数组 $A[l..r]$，它由左右两边的索引 l 和 r ($l \leq r$)定义

//输出：$A[l..r]$的划分和中轴的新位置

$p \leftarrow A[l]$

$s \leftarrow l$

for $i \leftarrow l+1$ **to** r **do**

  if $A[i] < p$

    $s \leftarrow s+1$; swap($A[s]$, $A[i]$)

swap($A[l]$, $A[s]$)

return s

计算中值和选择问题

- 快速选择算法：
  - 基于Lomuto划分，得到划分的分割位置s
  - 如果s=k-1，则中轴p就是第k小的元素
  - 如果s>k-1，则数组的第k小元素就是划分左边部分的第k小元素
  - 如果s<k-1，则为划分右边部分第(k-s)小元素

算法 Quickselect($A[l..r], k$)
//用基于划分的递归算法解决选择问题
//输入：可排序数组 $A[0..n-1]$的子数组 $A[l..r]$和整数 $k\ (1 \leqslant k \leqslant r-l+1)$
//输出：$A[l..r]$中第 k 小元素的值
$s \leftarrow$LomutoPartition($A[l..r]$) //或者另一个划分算法
if $s = l + k - 1$ **return** $A[s]$
else if $s > l + k - 1$ Quickselect($A[l..s-1], k$)
else Quickselect($A[s+1..r], l + k - 1 - s$)

计算中值和选择问题

- 应用实例
  - 找到数列4, 1, 10, 8, 7, 12, 9, 2, 15中第5小的元素

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

0	1	2	3	4	5	6	7	8
			s	*i*				
			8	7	12	9	10	15
				s	*i*			
			8	7	12	9	10	15
				s				*i*
			8	7	12	9	10	15
			7	**8**	12	9	10	15

计算中值和选择问题

- 快速选择算法的时间效率
  - 最好效率：一次Lomuto划分，无需迭代

$$C_{best}(n) = n - 1 \in \Theta(n)$$

  - 最差效率：n-1次迭代，每次都做Lomuto划分

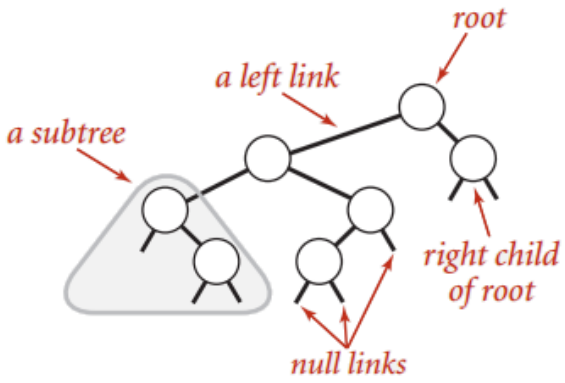
$$C_{worst}(n) = (n-1) + (n-2) + \cdots + 1 = (n-1)n/2 \in \Theta(n^2).$$

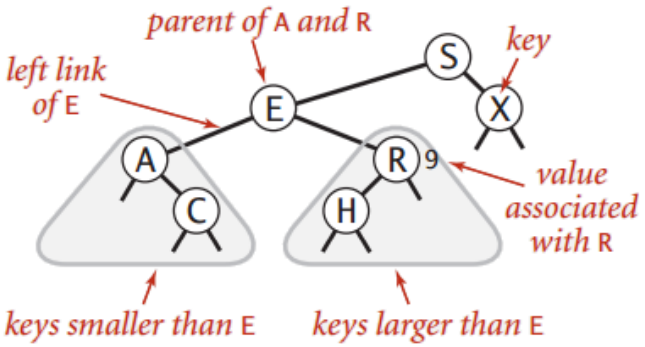
  - 平均效率：

$C_{average} \in \Theta(n)$

二叉查找树的查找和插入

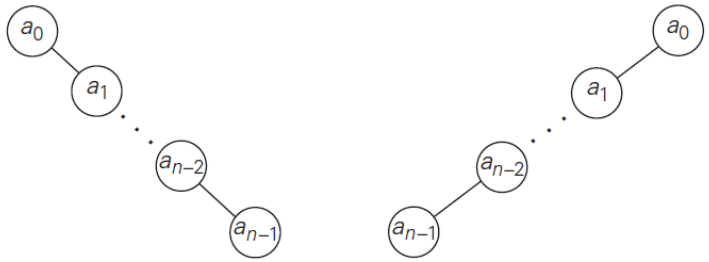
- 二叉查找树，它或者是一棵空，或满足以下的性质：
  - 每个节点作为搜索对象，它的关键字是互不相同的
  - 对于树上的所有节点，如果它有左子树，那么左子树上的所有节点的关键字都小于该节点的关键字
  - 对于树上的所有节点，如果它有右子树，那么右子树上的所有节点的关键字都大于该节点的关键字





二叉查找树的查找和插入

Algorithm *BTS*(*x*, *v*)
//Searches for node with key equal to *v* in BST rooted at node *x*
if *x* = NIL return -1
else if *v* = *K*(*x*) return *x*
else if *v* < *K*(*x*) return *BTS*(*left*(*x*), *v*)
else return *BTS*(*right*(*x*), *v*)



Binary search trees for (a) an increasing sequence of keys and (b) a decreasing sequence of keys.

- 时间效率
  - 最差效率：$C(n) = n$
  - 平均效率：$C(n) \approx 1.39log_2n$

拈游戏

- 问题：有一堆棋子，个数为n，甲、乙玩家轮流从这堆棋子中拿走一定数量（每次只能拿走最少1个，最多m个棋子）的棋子，每次拿走的棋子数量可以不同，哪个玩家能够胜利拿到最后那个棋子？是先走还是后走的？

拈游戏

- 假设甲先拿：
  ① $1 \le n \le \mathrm{m}$: 甲只要把棋子全部拿走，则胜出
  ② $n = \mathrm{m} + 1$: 甲不管怎么拿，必输
  ③ $\mathrm{m} + 2 \le n \le 2\mathrm{m} + 1$: 甲拿走一定数量的棋子，使得剩余棋子数为$\mathrm{m} + 1$，则乙必输

只有当棋子数n不是m+1的倍数时，甲只要每次拿走 n mod（m+1）个棋子后，剩余的棋子数一定为（m+1）的倍数，甲必胜出

拈游戏：多堆棋子

- 多堆拈游戏包含K堆棋子，每堆的棋子数分别为$n_1, n_2, \ldots, n_K$。每次走的时候，玩家可以从任意一堆棋子中拿走允许数量的棋子，甚至可以全拿走。

- 目标：成为最后一个还能走的玩家

拈游戏：多堆棋子

- 精妙的解决方案（哈佛大学教授C.L.Bouton提出）
  - 假设$b_1, b_2, \ldots, b_K$是各堆棋子数目的二进制表示
  - 计算它们的二进制数位和（忽略进位；也称拈和）
  - 当且仅当拈和中包含至少一个1时，为胜局
  - 当且仅当拈和中只包含0时，为败局

- 举例
  - $n_1 = 3, n_2 = 4, n_3 = 5$

策略：先走玩家从第一堆中拿走2个棋

$$\begin{array}{r} 011 \\ 100 \\ 101 \\ \hline 010 \end{array}$$

小结

- 减治法
  - 利用一个问题给定实例的解和同样问题较小实例的解之间的某种关系，将问题的规模递减，然后基于小规模问题的解，得到大规模问题的解
  - 3种变化形式
    - 减常量
    - 减常因子
    - 减可变规模
- 典型应用
  - 插入排序（平均效率比最差情况快1倍）、拓扑排序、生成组合对像
  - 折半查找、选择问题、二叉查找树的查找和插入

课后作业

章 X	节 X. Y	课后作业题 Z	思考题 Z
4	4.1	10	1, 2, 3, 6
	4.2	1, 5	10
	4.3	2, 6	11, 12
	4.4	8	7, 10
	4.5	13	10, 11, 12
	算法设计题 （参见下一页PPT）		

注：只需上交“课后作业题”（包括算法设计题）；以“学号姓名_chX.pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年3月26日

课后作业：算法设计题

- 将一个长度为n的升序数组分割成两部分，然后对调前后部分后，形成一个新的数组A，该数组称为对调有序数组。例如，数组A=[4, 5, 6, 7, 8, 9, 1, 2, 3]就是对调有序数组。
  ① 给定查找键K，请设计一个高效的算法（提供思路描述+伪代码），在任意给定的对调有序数组A中，查找确定跟查找键K匹配的元素位置，如果未查到，则返回-1；
  ② 请分析所设计算法的时间复杂度。

课后阅读（1）：俄式乘法

- 设n、m是整数，以n为实例规模的度量
- 若n为偶数，则

  $n \cdot m = (n/2) \cdot 2m$
- 若n为奇数，则

  $n \cdot m = ((n-1)/2) \cdot 2m + m$
- 以$1 \cdot m = m$为算法停止的条件

俄式乘法实例：50x65

- 若采用二进制表示，主要是移位和相加操作，适合机器实现

n	m	
50	65	
25	130	
12	260	(+130)
6	520	
3	1040	
1	2080	(+1040)
	2080	+(130 + 1040) = 3250

(a)

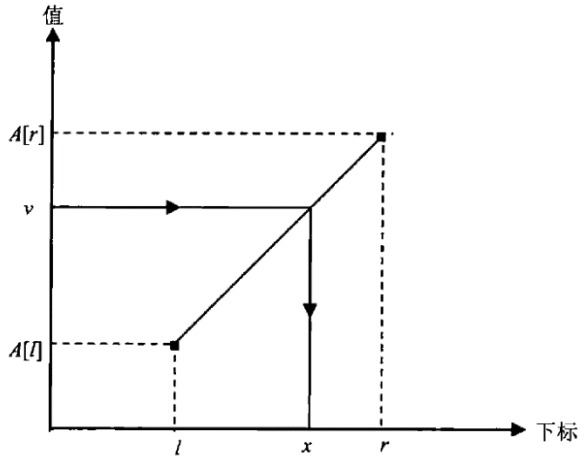
n	m	
50	65	
25	130	130
12	260	
6	520	
3	1040	1040
1	2080	2080
		3250

(b)

课后阅读(2)：插值查找

- 原理：是查找有序数组的一种算法。设某次迭代处理的是数组最左边元素A[l]和最右边元素A[r]之间的部分，要查找的键值为v。写出过点(l, A[l]) 和点(r, A[r])的直线方程，取y=v，求出横坐标x，比较A[x]与v的值，决定是停止，还是在l与x-1之间或x+1与r之间继续查找

$$x = l + \left\lfloor \frac{(v - A[l])(r - l)}{A[r] - A[l]} \right\rfloor$$



- 与折半查找做比较？

插值查找

- 时间效率
  - 最差效率：$C(n) = n$
  - 平均效率：$C(n) < log_2(log_2 n)$+1

- 与折半查找对比
  - 较小的文件，折半查找更优
  - 较大的文件，折半查找开销大，插值查找更优