山东师范大学 SHANDONG NORMAL UNIVERSITY

《计算复杂性理论》
第4讲 递归与分治方法

山东师范大学信息科学与工程学院
段会川

2015年9月

目录

递归—回顾

递归—普适性

- 递归是一种普适性的问题解决方法

递归—GCD算法的递归表述

递归—概述

递归技术是设计和描述算法的一种强有力的工具，它在算法设计与分析中起着非常重要的作用，采用递归技术编写出的程序通常比较简洁且易于理解，并且证明算法的正确性要比相应的非递归形式容易得多。因此在实际的编程中，人们常采用该技术来解决某些复杂的计算问题。有些数据结构如二叉树，结构本身就具有递归特性；此外还有一类问题，其本身没有明显的递归结构，但用递归程序求解比其他方法更容易编写程序，如八皇后问题、汉诺塔问题等。鉴于该技术的优点和重要性，在介绍其他算法设计方法之前先对其进行讨论。

求解问题的递归方法

子程序(或函数)直接调用自己或通过一系列调用语句间接调用自己,称为递归。直接或间接调用自身的算法称为递归算法。递归的基本思想就是"自己调用自己",体现了"以此类推"、"重复同样的步骤"这样的理念。实际上,递归是把一个不能或不好解决的大问题转化为一个或几个小问题,再把这些小问题进一步分解成更小的小问题,直至每个小问题都可以直接解决。

通常,采用递归算法来求解问题的一般步骤是:

(1) 分析问题、寻找递归关系。找出大规模问题和小规模问题的关系。换句话说,如果一个问题能用递归方法解决,它必须可以向下分解为若干个性质相同的规模较小的问题。

(2) 找出停止条件,该停止条件用来控制递归何时终止,在设计递归算法时需要给出明确的结束条件。

(3) 设计递归算法、确定参数,即构建递归体。

递归算法的运行过程包含两个阶段:递推和回归。递推指的是将原问题不断分解为新的子问题,逐渐从未知向已知推进,最终达到已知的条件,即递归结束的条件。回归指的是从已知的条件出发,按照递推的逆过程,逐一求值回归,最后达到递推的开始处,即求得问题的解。

n!的递归算法及其复杂度的递推分析法

- 算法名称：n!的递归计算factorial
- 输入：n
- 输出：n!
- 1: factorial(n)
- 2: if n=0 then
- 3: return 1
- 4: else
- 5: return n*factorial(n-1)

任意大n的n!计算方法

目录

分治法—概述

- 在计算机科学中，分治法(Divide and Conquer)是建立在多项分支递归的一种很重要的算法范式。
  - 字面上的解释是“分而治之”，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，直到最后子问题可以简单到直接求解，原问题的解即子问题解的合并。
- 这个技巧是很多高效算法的基础，如排序算法(快速排序、归并排序)、傅立叶变换(快速傅立叶变换)。
- 分治算法的正确性通常以数学归纳法证明，而它的计算复杂度则多以解递推关系式求取。
  - 如果可能则应用主定理。

分治法—三个步骤

1. 分解：
   - 将原问题分解为若干个规模较小、相对独立、与原问题形式相同的子问题。
2. 解决：
   - 若子问题规模较小且易于解决时则直接解出。
   - 否则递归地解决各子问题。
3. 合并：
   - 将各子问题的解合并为原问题的解。

分治法—历史

- 折半搜索算法(二叉搜索，binary search)
  - 将原来问题连续地拆分成大约一半大小的单一子问题的分治算法的构想早已在公元前200年的巴比伦尼亚时代就已经出现。
  - 算法在计算机上的清楚描述出现在1946年约翰·莫齐利（John Mauchly）的一篇文章里。
- 辗转相除法—欧几里德算法
  - 它是一个通过将问题转化为单一的更小问题从而快速求解的方法，因而也可看成是一种分治算法。
  - 它也是在2000多年前的公元前提出的。

分治法—历史

- 专门用于计算机之上而且正确地分析的分治算法最早期的例子，则是约翰 冯 诺伊曼于1945年发明的归并排序算法(merge sort)，也称为合并排序算法。
- A. A. Karatsuba基于高斯乘法方法于1960年发明的在$O(n^{\log_2 3})$步骤内将两个n位数相乘的算法是另一个分治算法的经典例子。
  - 它反证了安德列 柯尔莫哥洛夫(安德列 尼古拉耶维奇 柯尔莫哥洛夫，Andrey Nikolaevich Kolmogorov，Андрей Николаевич Колмогоров，1903.4.25—1987.10.20)于1956年认为两个n位数相乘需要$\Omega(n^2)$步骤的猜想。

约翰 冯 诺伊曼

- 出生于匈牙利的美国籍犹太人数学家，现代计算机创始人之一
  - 在计算机科学、经济学、物理学中的量子力学及几乎所有数学领域都作过重大贡献

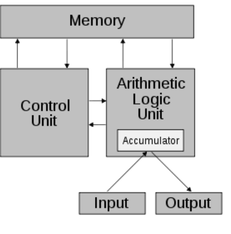
John von Neumann
1903.12.28－1957.2.8

约翰 冯 诺伊曼—计算机之父

- 1945年6月，冯 诺伊曼与戈德斯坦、勃克斯等人，联名发表了一篇长达101页纸的报告，即计算机史上著名的“101页报告”，是现代计算机科学发展里程碑式的文献。
  - 该报告明确在计算机中用二进制替代十进制运算，并将计算机分成五大组件，这一卓越的思想为电子计算机的逻辑结构设计奠定了基础，已成为计算机设计的基本原则。
  - 由于他在计算机逻辑结构设计上的伟大贡献，他被誉为“计算机之父”。

冯 诺伊曼体系结构—Von Neumann Architecture

- 也称普林斯顿体系结构，是一种将程序指令和数据一起存储的计算机概念结构
  - 现代计算机基本上基于该架构设计，因而也称为存储程序计算机，它是通用图灵机的具体实现
  - IEEE(Institute of Electrical and Electronics Engineers, 国际电气与电子工程师学会，读作I-Triple-E)以他的名字命名了IEEE冯 诺伊曼奖，奖励计算机科学和技术上具有杰出成就的科学家



约翰 冯 诺伊曼—博弈论与经济学贡献

- 在经济学领域，1944年冯 诺伊曼与摩根施特恩合著的巨作《博弈论与经济行为》(Theory of Games and Economic Behavior)出版，标志着现代系统博弈理论的的初步形成
  - 他因此被称为“博弈论之父”。
  - 博弈论被认为是20世纪经济学最伟大的成果之一
  - INFORMS(Institute for Operations Research and the Management Sciences, 运筹学与管理科学学会)设立了冯 诺伊曼理论奖，以奖励在运筹学与管理科学领域做出基础性和持续性贡献的科学家

高斯乘法

- 两个复数相乘
  - $(a+bi)(c+di)=ac-bd+(ad+bc)i$
  - 包括4次乘法和两次加法运算
- 高斯乘法
  - $ad+bc=(a+b)(c+d)-ac-bd$
  - 这使两个复数相乘的运算变换为3次乘法和5次加法运算
  - 虽然初看没有多大改进，但由于乘法运算是$O(n^2)$的复杂度，而加法运算是$O(n)$的复杂度，如果迭代进行，复杂度的改进还是很可观的

乘法的分解算法

- **n**位的二进制数可以分解为两个$\frac{n}{2}$位的组成部分

$$x=\boxed{x_L}\boxed{x_R}=2^{n/2}x_L+x_R$$

$$y=\boxed{y_L}\boxed{y_R}=2^{n/2}y_L+y_R.$$

- 如：$x=10110110_2=1011_2\times 2^4+0110_2$
  即: $x_L=1011, x_R=0110.$
- 经过上述分解，两个数的积可以表达为
  - $xy=x_Ly_L2^n+(x_Ly_R+y_Lx_R)2^{n/2}+x_Ry_R$
  - 由4次乘法和3次加法组成，而乘法可以递归进行下去
  - 注意：乘以2^n和$2^{n/2}$可以用左移操作在线性时间里实现

乘法分解算法—伪代码

- 算法名称：乘法分解算法
- 输入：两个**n**位的正整数**x**和**y**
- 输出：**xy**
- **1：multiply0(x, y)**
- **2：n ← x,y**的位数
- **3：if n=1: return xy**
- 4：$x_L, x_R=x$的高$\lceil n/2\rceil$位和低$\lfloor n/2\rfloor$位.
- 5：$y_L, y_R=y$的高$\lceil n/2\rceil$位和低$\lfloor n/2\rfloor$位.
- 6：$P1=\text{multiply0}(x_L, y_L)$. $P2=\text{multiply0}(x_L, y_R)$.
- 7：$P3=\text{multiply0}(x_R, y_L)$. $P4=\text{multiply0}(x_R, y_R)$.
- **8：return** $P1\times 2^n+(P2+P3)\times 2^{n/2}+P4$

乘法分解算法—复杂度

- 乘法分解算法的一次调用中要执行4次乘法和3次加法，以及2次移位操作
  - 加法和移位操作的复杂度为$O(n)$
  - 乘法需要递归进行
  - 当$n=1$时，复杂度为$O(1)$，递归结束
  - 算法的总复杂度可以用如下的递推式表示
  - $T(n)=\begin{cases} O(1) & \text{当}n=1 \\ 4T(n/2)+O(n) & \text{当}n>1 \end{cases}$
  - 该递推式的解后面将会看到是$T(n)=O(n^2)$

Karatsuba乘法算法

- **A. A. Karatsuba**借助于高斯乘法，于**1960**年提出了第**1**个快速乘法算法
  - 基本思路是将

    $$xy=x_Ly_L2^n+(x_Ly_R+y_Lx_R)2^{n/2}+x_Ry_R$$

    中的$x_Ly_R+y_Lx_R$表达为

    $$(x_L+x_R)(y_L+y_R)-x_Ly_L-x_Ry_R.$$

  - 使乘法运算由4次减为3次，代价是增加了3次加法

Karatsuba乘法算法—伪代码

- 算法名称： **Karatsuba**乘法算法
- 输入：两个**n**位的正整数**x**和**y**
- 输出：**xy**
- **1：multiplyK(x, y)**
- **2：n ← x,y**的位数
- **3：if n=1: return xy**
- 4：$x_L, x_R=x$的高$\lceil n/2\rceil$位和低$\lfloor n/2\rfloor$位.
- 5：$y_L, y_R=y$的高$\lceil n/2\rceil$位和低$\lfloor n/2\rfloor$位.
- 6：$P1=\text{multiply}K(x_L, y_L)$. $P2=\text{multiply}K(x_R, y_R)$.
- 7：$P3=\text{multiply0}(x_L+x_R, y_L+y_R)$.
- **8：return** $P1\times 2^n+(P3-P1-P2)\times 2^{n/2}+P2$

Karatsuba乘法算法—复杂度

- Karatsuba乘法算法的一次调用中要执行3次乘法和6次加法，以及2次移位操作
  - 加法和移位操作的复杂度为$O(n)$
  - 乘法需要递归进行
  - 当$n=1$时，复杂度为$O(1)$，递归结束
  - 算法的总复杂度可以用如下的递推式表示
  - $T(n)=\begin{cases} O(1) & 当n=1 \\ 3T(n/2)+O(n) & 当n>1 \end{cases}$
  - 该递推式的解后面将会看到是
    $T(n)=O\left(n^{\log_2 3}\right)\approx O(n^{1.59})$

矩阵乘法—一般方法

$$\mathbf{A}=\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}, \quad \mathbf{B}=\begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix}$$

$$\mathbf{AB}=\begin{pmatrix} (AB)_{11} & (AB)_{12} & \cdots & (AB)_{1p} \\ (AB)_{21} & (AB)_{22} & \cdots & (AB)_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ (AB)_{n1} & (AB)_{n2} & \cdots & (AB)_{np} \end{pmatrix}$$

$$(AB)_{ij}=\sum_{k=1}^{m} A_{ik}B_{kj}.$$

一般矩阵乘法要产生n^2个元素，而每个元素需要n次乘法，因而复杂度为n^3

矩阵乘法—分块方法

- 将两个待乘的矩阵分别分解为4个大小为$n/2\times n/2$的子矩阵

$$X=\begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y=\begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

$$XY=\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix}=\begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

  - 分块后需要进行8次$n/2\times n/2$子矩阵乘法和4次$n/2\times n/2$子矩阵的加法，而加法的复杂度为$O(n^2)$
  - 因而复杂度递推公式为：$T(n)=8T\left(\frac{n}{2}\right)+O(n^2)$

矩阵乘法—分块方法复杂度

- 复杂度递推式
  - $T(n)=8T\left(\frac{n}{2}\right)+O(n^2)$
  - $a=8, b=2, d=2<\log_b a=3$
  - 所以：$T(n)=O\left(n^{\log_b a}\right)=O(n^3)$

矩阵乘法—Strassen算法

- 德国数学家Volker Strassen于1969年提出了一个快速的矩阵乘法算法

$$\mathbf{C}=\mathbf{AB} \qquad \mathbf{A},\mathbf{B},\mathbf{C}\in R^{2^n\times 2^n}$$

$$\mathbf{A}=\begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B}=\begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C}=\begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\mathbf{A}_{i,j},\mathbf{B}_{i,j},\mathbf{C}_{i,j}\in R^{2^{n-1}\times 2^{n-1}}$$

$$\mathbf{C}_{1,1}=\mathbf{A}_{1,1}\mathbf{B}_{1,1}+\mathbf{A}_{1,2}\mathbf{B}_{2,1} \qquad \mathbf{C}_{1,2}=\mathbf{A}_{1,1}\mathbf{B}_{1,2}+\mathbf{A}_{1,2}\mathbf{B}_{2,2}$$

$$\mathbf{C}_{2,1}=\mathbf{A}_{2,1}\mathbf{B}_{1,1}+\mathbf{A}_{2,2}\mathbf{B}_{2,1} \qquad \mathbf{C}_{2,2}=\mathbf{A}_{2,1}\mathbf{B}_{1,2}+\mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

矩阵乘法—Strassen算法

- 德国数学家Volker Strassen于1969年提出了一个快速的矩阵乘法算法

$$\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1}+\mathbf{A}_{2,2})(\mathbf{B}_{1,1}+\mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1}+\mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2}-\mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1}-\mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1}+\mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1}-\mathbf{A}_{1,1})(\mathbf{B}_{1,1}+\mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2}-\mathbf{A}_{2,2})(\mathbf{B}_{2,1}+\mathbf{B}_{2,2})
\end{aligned}$$

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1+\mathbf{M}_4-\mathbf{M}_5+\mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3+\mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2+\mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1-\mathbf{M}_2+\mathbf{M}_3+\mathbf{M}_6
\end{aligned}$$

矩阵乘法—Strassen算法

- 德国数学家Volker Strassen于1969年提出了一个快速的矩阵乘法算法

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

$P_1 = A(F - H)$

$P_2 = (A + B)H$

$P_3 = (C + D)E$

$P_4 = D(G - E)$

$P_5 = (A + D)(E + H)$

$P_6 = (B - D)(G + H)$

$P_7 = (A - C)(E + F)$

P67

  - 该算法包括7次规模为$\frac{n}{2}$的矩阵乘法和18次加法

矩阵乘法—Strassen算法

- Strassen算法复杂度的递推式
  - $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$
- 运用主定理
  - $a = 7, b = 2, d = 2 < \log_b a = \log_2 7 \approx 2.81$
  - 所以：$T(n) = O\left(n^{\log_2 7}\right) \approx O(n^{2.81})$

递推式的通解—主定理(多项式合并函数)

- 主定理(master theorem)
  - 如果对于常数$a > 0, b > 1, d \geq 0$, 有 $T(n) = aT(\lceil n/b \rceil) + O(n^d)$
  - 则有：$T(n) = \begin{cases} O(n^d) & 当 d > \log_b a \\ O(n^d \log n) & 当 d = \log_b a \\ O\left(n^{\log_b a}\right) & 当 d < \log_b a \end{cases}$
- 当分治法每次将问题分解为a个规模为n/b的子问题，而将各子问题的解合并的时间复杂度为$O(n^d)$时，该分治法的运算时间的递推式便是
  - $T(n) = aT(\lceil n/b \rceil) + O(n^d)$
  - 因而可以套用主定理求解

递推式的通解—主定理(普适合并函数)

4.5 用主方法求解递归式

主方法为如下形式的递归式提供了一种“菜谱”式的求解方法

$$T(n) = aT(n/b) + f(n) \tag{4.20}$$

其中 $a \geqslant 1$ 和 $b > 1$ 是常数，$f(n)$是渐近正函数。为了使用主方法，需要牢记三种情况，但随后你就可以很容易地求解很多递归式，通常不需要纸和笔的帮助。

递归式(4.20)描述的是这样一种算法的运行时间：它将规模为 n 的问题分解为 a 个子问题，每个子问题规模为 n/b，其中 a 和 b 都是正常数。a 个子问题递归地进行求解，每个花费时间 $T(n/b)$。函数 $f(n)$包含了问题分解和子问题解合并的代价。例如，描述 Strassen 算法的递归式中，$a=7$，$b=2$，$f(n)=\Theta(n^2)$。

从技术的正确性方面看，此递归式实际上并不是良好定义的，因为 n/b 可能不是整数。但将 a 项 $T(n/b)$都替换为 $T(\lfloor n/b \rfloor)$或 $T(\lceil n/b \rceil)$并不会影响递归式的渐近性质(我们将在下一节证明这个断言)。因此，我们通常发现当写下这种形式的分治算法的递归式时，忽略舍入问题是很方便的。

递推式的通解—主定理(普适合并函数)

主定理

主方法依赖于下面的定理。

定理 4.1(主定理) 令 $a \geqslant 1$ 和 $b > 1$ 是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将 n/b 解释为$\lfloor n/b \rfloor$或$\lceil n/b \rceil$。那么 $T(n)$有如下渐近界：

1. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = O(n^{\log_b a - \varepsilon})$，则 $T(n) = \Theta(n^{\log_b a})$。
2. 若 $f(n) = \Theta(n^{\log_b a})$，则 $T(n) = \Theta(n^{\log_b a} \lg n)$。
3. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$，且对某个常数 $c < 1$ 和所有足够大的 n 有 $af(n/b) \leqslant cf(n)$，则 $T(n) = \Theta(f(n))$。 ■

在使用主定理之前，我们花一点儿时间尝试理解一下它的含义。对于三种情况的每一种，我们将函数 $f(n)$与函数 $n^{\log_b a}$进行比较。直觉上，两个函数较大者决定了递归式的解。若函数 $n^{\log_b a}$更大，如情况 1，则解为 $T(n) = \Theta(n^{\log_b a})$。若函数 $f(n)$更大，如情况 3，则解为 $T(n) = \Theta(f(n))$。若两个函数大小相当，如情况 2，则乘上一个对数因子，解为 $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$。

递推式的通解—主定理(普适合并函数)

在此直觉之外，我们需要了解一些技术细节。在第一种情况中，不是 $f(n)$小于 $n^{\log_b a}$就够了，而是要多项式意义上的小于。也就是说，$f(n)$必须渐近小于 $n^{\log_b a}$，要相差一个因子 n^ε，其中 ε 是大于 0 的常数。在第三种情况中，不是 $f(n)$大于 $n^{\log_b a}$就够了，而是要多项式意义上的大于，而且还要满足“正则”条件 $af(n/b) \leqslant cf(n)$。我们将会遇到的多项式界的函数中，多数都满足此条件。

注意，这三种情况并未覆盖 $f(n)$的所有可能性。情况 1 和情况 2 之间有一定间隙，$f(n)$可能小于 $n^{\log_b a}$但不是多项式意义上的小于。类似地，情况 2 和情况 3 之间也有一定间隙，$f(n)$可能大于 $n^{\log_b a}$但不是多项式意义上的大于。如果函数 $f(n)$落在这两个间隙中，或者情况 3 中要求的正则条件不成立，就不能使用主方法来求解递归式。

递推式的通解—主定理(普适合并函数)

使用主方法

使用主方法很简单，我们只需确定主定理的哪种情况成立，即可得到解。

我们先看下面这个例子

$$T(n) = 9T(n/3) + n$$

对于这个递归式，我们有 $a=9$，$b=3$，$f(n)=n$，因此 $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$。由于 $f(n)=O(n^{\log_3 9-\epsilon})$，其中 $\epsilon=1$，因此可以应用主定理的情况 1，从而得到解 $T(n)=\Theta(n^2)$。

现在考虑

$$T(n) = T(2n/3) + 1$$

其中 $a=1$，$b=3/2$，$f(n)=1$，因此 $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$。由于 $f(n)=\Theta(n^{\log_b a})=\Theta(1)$，因此应用情况 2，从而得到解 $T(n)=\Theta(\lg n)$。

递推式的通解—主定理(普适合并函数)

对于递归式

$$T(n) = 3T(n/4) + n\lg n$$

我们有 $a=3$，$b=4$，$f(n)=n\lg n$，因此 $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$。由于 $f(n)=\Omega(n^{\log_4 3+\epsilon})$，其中 $\epsilon\approx 0.2$，因此，如果可以证明正则条件成立，即可应用情况 3。当 n 足够大时，对于 $c=3/4$，$af(n/b)=3(n/4)\lg(n/4)\leqslant(3/4)n\lg n=cf(n)$。因此，由情况 3，递归式的解为 $T(n)=\Theta(n\lg n)$。

主方法不能用于如下递归式：

$$T(n) = 2T(n/2) + n\lg n$$

虽然这个递归式看起来有恰当的形式：$a=2$，$b=2$，$f(n)=n\lg n$，以及 $n^{\log_b a}=n$。你可能错误地认为应该应用情况 3，因为 $f(n)=n\lg n$ 渐近大于 $n^{\log_b a}=n$。问题出在它并不是多项式意义上的大于。对任意正常数 ϵ，比值 $f(n)/n^{\log_b a}=(n\lg n)/n=\lg n$ 都渐近小于 n^{ϵ}。因此，递归式落入了情况 2 和情况 3 之间的间隙(此递归式的解参见练习 4.6-2)。

***4.6-2** 证明：如果 $f(n)=\Theta(n^{\log_b a}\lg^k n)$，其中 $k\geqslant 0$，那么主递归式的解为 $T(n)=\Theta(n^{\log_b a}\lg^{k+1} n)$。为简单起见，假定 n 是 b 的幂。

递推式的通解—主定理(普适合并函数)

我们利用主方法求解在 4.1 节和 4.2 节中曾见过的递归式(4.7)，

$$T(n) = 2T(n/2) + \Theta(n)$$

它刻画了最大子数组问题和归并排序的分治算法的运行时间(按照通常的做法，我们忽略了递归式中基本情况的描述)。这里，我们有 $a=2$，$b=2$，$f(n)=\Theta(n)$，因此 $n^{\log_b a}=n^{\log_2 2}=n$。由于 $f(n)=\Theta(n)$，应用情况 2，于是得到解 $T(n)=\Theta(n\lg n)$。

递归式(4.17)，

$$T(n) = 8T(n/2) + \Theta(n^2)$$

它描述了矩阵乘法问题第一个分治算法的运行时间。我们有 $a=8$，$b=2$，$f(n)=\Theta(n^2)$，因此 $n^{\log_b a}=n^{\log_2 8}=n^3$。由于 n^3 多项式意义上大于 $f(n)$(即对 $\epsilon=1$，$f(n)=O(n^{3-\epsilon})$)，应用情况 1，解为 $T(n)=\Theta(n^3)$。

最后，我们考虑递归式(4.18)，

$$T(n) = 7T(n/2) + \Theta(n^2)$$

它描述了 Strassen 算法的运行时间。这里，我们有 $a=7$，$b=2$，$f(n)=\Theta(n^2)$，因此 $n^{\log_b a}=n^{\log_2 7}$。将 $\log_2 7$ 改写为 $\lg 7$，由于 $2.80<\lg 7<2.81$，我们知道对 $\epsilon=0.8$，有 $f(n)=O(n^{\lg 7-\epsilon})$。再次应用情况 1，我们得到解 $T(n)=\Theta(n^{\lg 7})$。

乘法分解算法复杂度的主定理解

- 乘法分解算法复杂度的递推式
  - $T(n) = \begin{cases} O(1) & 当n = 1 \\ 4T(n/2) + O(n) & 当n > 1 \end{cases}$
  - 显然，$a = 4, b = 2, d = 1$
  - 此时有，$d = 1 < \log_b a = 2$
  - 因而，$T(n) = O\left(n^{\log_b a}\right) = O(n^2)$

Karatsuba乘法算法复杂度的主定理解

- Karatsuba乘法算法复杂度的递推式
  - $T(n) = \begin{cases} O(1) & 当n = 1 \\ 3T(n/2) + O(n) & 当n > 1 \end{cases}$
  - 显然，$a = 3, b = 2, d = 1$
  - 此时有，$d = 1 < \log_b a = \log_2 3$
  - 因而，$T(n) = O\left(n^{\log_b a}\right) = O\left(n^{\log 3}\right) \approx O(n^{1.59})$

二分搜索算法(binary search, 折半查找)

- 算法名称：二分搜索算法(BinarySearch)
- 输入：元素由低到高排序的数组a和待搜索的数据x
- 输出：x在a中的位置，-1表示未找到
- 1：BinarySearch(a, low, high, x)
- 2：if low>high: return -1
- 3：mid = ⌊(low+high)/2⌋
- 4：if x=a[mid] then return mid
- 5：if x<a[mid] then BinarySearch(a, low, mid-1, x)
- 6：else BinarySearch(a, mid+1, high, x)

二分搜索算法—复杂度分析

- 最好情况复杂度
  - 在第1次调用时，中间的元素就是要找的元素
  - 因而，复杂度为$O(1)$
- 最坏情况复杂度
  - 当low=high时才找到，或没有找到
  - $T(n) = T(n/2) + O(1)$
  - 运用主定理
    - $a = 1, b = 2, d = 0 = \log_b a = 0$
    - 因而，$T(n) = O(n^d \log n) = O(\log n)$

x^n的计算算法

- 算法名称：x^n的计算算法(power)
- 输入：x, n
- 输出：x^n
- 1：power(x, n)
- 2：if n=1 then return x
- 3：p = power(x, n/2)
- 4：p = p*p
- 5：if n is odd then p = p*x

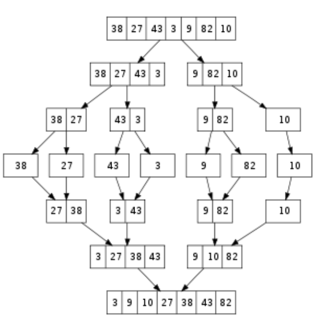
x^n计算算法—复杂度分析

- 显然，x^n计算算法与二分搜索算法相似
  - $T(n) = T(n/2) + O(2)$
  - 运用主定理
    - $a = 1, b = 2, d = 0 = \log_b a = 0$
    - 因而，$T(n) = O(n^d \log n) = O(\log n)$

归并排序(merge sort, 合并排序)—描述

- 归并排序由冯 诺伊曼于1945年提出，是一个典型的分治算法
  - 分解：将排序数组分解为两个等大的子数组
  - 解决：无
  - 合并：将两个已经有序的数组合并为一个有序的数组
    - 大小分别为m和n的有序数组可以在线性时间$O(m+n)$内合并

归并排序—示例



归并排序—分治算法伪代码

- 算法名称：归并排序(MergeSort)
- 输入：n个元素的数组a
- 输出：排序的a
- 1：MergeSort(a, low, high, b)
- 2：if high-low<=1 then return
- 3：mid = (low+high)/2
- 4：MergeSort(a, low, mid, b)
- 5：MergeSort(a, mid+1, high, b)
- 6：Merge(a, low, mid, high, b)
- 7：copy(b, low, high, a)

归并排序—合并算法描述

1. 给定两个相邻的分别已经排好序的数组
2. 申请临时空间，使其大小为两个已经排序数组大小之和，该空间用来存放合并后的序列
3. 设定两个指针，最初位置分别为两个已经排序数组的起始位置
4. 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置
5. 重复步骤4直到某一指针到达序列尾
6. 将有剩余元素的数组中的元素复制到合并数组末尾

归并排序—合并算法伪代码

- 算法名称：合并两个有序数组(Merge)
- 输入：数组a，low到mid及mid+1到high间已排序
- 输出：数组a，从low到high是排序的

```
i0 = low, i1 = mid+1
for j = low to high
    if i0 < mid and (A[i0] <= A[i1] or i1 >= high))
        B[j] = A[i0]
        i0 = i0 + 1
    else
        B[j] = A[i1]
        i1 = i1 + 1
    end if
```


归并排序—复杂度分析

- 显然归并排序计算复杂度的递推式为
  - $T(n) = 2T(n/2) + O(n)$
  - 运用主定理
    $a = 2, b = 2, d = 1 = \log_b a = 1.$
    因而，$T(n) = O(n^d \log n) = O(n\log n)$
  - 归并排序的合并阶段需要一个规模为n的数组，因而空间复杂度为$O(n)$
  - 归并排序是基于比较的排序算法中的一个最优算法，因为可以证明基于比较的排序算法最少需要$\Omega(n\log n)$的复杂度

目录

- 递归
- 分治方法
- 算法分析的数学基础
- 对数公式
- 组合公式
- 求和公式
- 二项式定理
- 取整函数
  - 低函数与顶函数
- 调和数及其复杂度
  - 数学家欧拉

对数公式

定义 9 令 b 是大于 1 的实数，x 是实数。如果对某些正实数 y，有 $y=b^x$，那么 x 称为 y 以 b 为底的对数，记为：$x=\log_b y$。其中，b 称为对数的底数，y 称为真数。

关于对数公式，有下列性质：

(1) 负数和零没有对数。

(2) 1 的对数是 0；即 $\log_b 1=0$。

(3) 底数的对数是 1，即 $\log_b b=1$。

(4) $\log_b b^n=n$。

(5) $b^{\log_b n}=n$。

两个特殊对数：以 10 为底的对数称为常用对数，即 N 的常用对数记做 $\lg N$；以无理数 e(e=2.71828…)为底的对数称为自然对数，N 的自然对数记做 $\ln N$；以 2 为底 N 的对数简记为 $\log N$。

排列与组合

- 全排列：$P_n^n = n!$
- 组合数：

$$C_n^k = \binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k!}$$

$$= \prod_{i=1}^{k} \frac{n-i+1}{i}$$

组合公式

定义 10 从 n 个不同元素中取 $m(m\leqslant n)$ 个不重复的元素组成一个子集，而不考虑其元素的顺序，称为从 n 个元素中取 m 个元素的无重组合。组合的全体组成的集合用 C_n^m 表示。

组合公式为：

$$C_n^m = \frac{n!}{m!(n-m)!} = \frac{n(n-1)\cdots(n-m+1)}{m(m-1)\cdots 2} \quad (m \leqslant n)$$

组合公式的性质：

(1) $C_n^m = C_n^{n-m}(m\leqslant n)$。

(2) $C_n^0 + C_n^1 + C_n^2 + \cdots + C_n^n = 2^n$。

(3) n 为奇数时，$C_n^0 + C_n^2 + \cdots + C_n^{n-1} = C_n^1 + C_n^3 + \cdots + C_n^n = 2^{n-1}$，

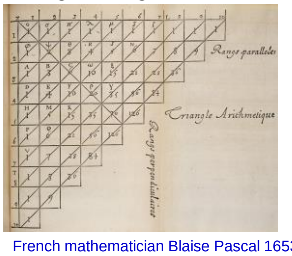
n 为偶数时，$C_n^0 + C_n^2 + \cdots + C_n^n = C_n^1 + C_n^3 + \cdots + C_n^{n-1} = 2^{n-1}$。

二项式展开

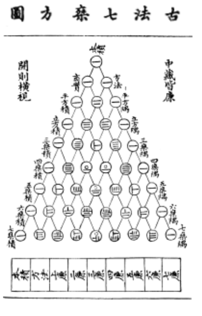
- $(x+y)^2$
- $(1+x)^2$
- $(x+y)^3$
- $(1+x)^3$
- $(x+y)^4$
- $(1+x)^4$

杨辉/Pascal三角形

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1



French mathematician Blaise Pascal 1653



北宋贾宪贾(1050)
南宋杨辉(1261)

Pascal等式

- $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$, $1 \le k \le n$

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

二项式定理(Isaac Newton, 1665)

- 对于任意的正整数n和实数x、y
- $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k = \sum_{k=0}^{n} C_n^k x^{n-k} y^k$
- $(1+x)^n$

广义二项式定理(Isaac Newton, 1665)

- 对于任意的复数r，定义二项式系数如下：
  - $\binom{r}{0} = 1$
  - $\binom{r}{k} = \frac{r(r-1)\cdots(r-k+1)}{k!}$, $k \ge 1$
- 则对于任意$|x| < |y|$的实数x、y和任意复数r，有：
  - $(x+y)^r = \sum_{k=0}^{\infty} \binom{r}{k} x^{r-k} y^k$

集合的子集个数

- n个元素的集合共有多少个子集？
  - $\binom{n}{0}, \binom{n}{1}, \cdots, \binom{n}{n-1}, \binom{n}{n}$
  - 总数：$\sum_{i=0}^{n} \binom{n}{i}$

求和公式

（1）算术级数。

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$

（2）平方和。

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$$

（4）调和级数。

把调和级数前 n 项之和记为 H_n，则 $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i}$。

求和公式

（3）几何级数。

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1} = \Theta(a^n), \quad a \neq 1$$

当 $a=2$ 时，

$$\sum_{i=0}^{n} 2^i = \frac{2^{n+1}-1}{2-1} = \Theta(2^n)$$

当 $a=1/2$ 时，

$$\sum_{i=0}^{n} \frac{1}{2^i} = 2 - \frac{1}{2^n} < 2 = \Theta(1)$$

当 $|a|<1$ 时，有如下的无穷级数：

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a} = \Theta(1)$$

取整函数

如果 x 是任意实数，则记：

$\lfloor x \rfloor$＝小于或等于 x 的最大整数，简称为 x 的下限，即$\lfloor\ \rfloor$为向下取整公式。

$\lceil x \rceil$＝大于或等于 x 的最小整数，简称为 x 的上限，即$\lceil\ \rceil$为向上取整公式。

例如：$\lfloor\sqrt{3}\rfloor = 1$，$\lceil\sqrt{3}\rceil = 2$，$\lfloor -\frac{1}{2} \rfloor = -1$，$\lceil -\frac{1}{2} \rceil = 0$。

容易证明下面的关系成立：

（1）$\lceil x \rceil = \lfloor x \rfloor$，当且仅当 x 是整数

（2）$\lceil x \rceil = \lfloor x \rfloor + 1$，当且仅当 x 不是整数

（3）$\lfloor -x \rfloor = -\lceil x \rceil$

（4）$x-1 < \lfloor x \rfloor \leqslant x \leqslant \lceil x \rceil < x+1$

（5）$\lfloor \frac{x}{2} \rfloor + \lceil \frac{x}{2} \rceil = x$

（6）一个很有用的定理。令 $f(x)$是一个单调递增函数，使得当 $f(x)$是整数时，x 也是整数。那么，有$\lfloor f(\lfloor x \rfloor) \rfloor = \lfloor f(x) \rfloor$并且$\lceil f(\lceil x \rceil) \rceil = \lceil f(x) \rceil$。

底函数与顶函数

- 底函数和顶函数在数学和计算机科学中使用广泛
  - **底函数floor**将一个实型数映射到小于等于它的最大整数，记为：$\mathrm{floor}(x) = \lfloor x \rfloor$。
    $\mathrm{floor}(x) = \lfloor x \rfloor$也形象地称为下取整函数。
  - **顶函数ceiling**或**ceil**将一个实型数映射到大于等于它的最小整数，记为：$\mathrm{ceiling}(x) = \lceil x \rceil$。
    $\mathrm{ceiling}(x) = \lceil x \rceil$也形象地称为上取整函数。
  - 举例
    $\lfloor 2.0 \rfloor = 2, \lfloor 2.1 \rfloor = 2, \lfloor 2.9 \rfloor = 2, \lfloor 3.0 \rfloor = 3,$
    $\lceil 2.0 \rceil = 2, \lceil 2.1 \rceil = 3, \lceil 2.9 \rceil = 3, \lceil 3.0 \rceil = 3,$
    $\lfloor -2.0 \rfloor = -2, \lfloor -2.1 \rfloor = -3, \lfloor -2.9 \rfloor = -3, \lfloor -3.0 \rfloor = -3,$
    $\lceil -2.0 \rceil = -2, \lceil -2.1 \rceil = -2, \lceil -2.9 \rceil = -2, \lceil -3.0 \rceil = -3.$

底函数与顶函数

- 底函数和顶函数在数学和计算机科学中使用广泛
  - 大数学家高斯于1808年引入了floor函数的记法：$[x]$
  - 1979年图灵奖获得者肯尼斯·艾佛森(Kenneth E. Iverson)在其著名的1962年著作《程序设计语言APL》(A Programming Language)中引入了$\lfloor x \rfloor$和$\lceil x \rceil$记法。
  - 几乎所有的现代程序设计语言均提供了floor或ceil(ceiling)函数
  - 现代程序设计语言通常还提供取整函数$\mathrm{int}(x)$，它与$\mathrm{floor}(x) = \lfloor x \rfloor$在$x \geq 0$时结果相同，而$x < 0$时结果不同，如：$\mathrm{int}(-2.3) = -2$
  - 现代程序设计语言通常还提供四舍五入函数$\mathrm{round}(x)$，如：$\mathrm{round}(2.3) = 2, \mathrm{round}(2.6) = 3$

底函数与顶函数

- 底函数与顶函数具有如下性质
  - 对于任何的整数$\boldsymbol{n}$，$\lfloor x+n\rfloor=\lfloor x\rfloor+\mathrm{n}$，$\lceil x+n\rceil=\lceil x\rceil+\mathrm{n}$
  - $\lfloor\lfloor x\rfloor\rfloor=\lfloor x\rfloor$，$\lceil\lceil x\rceil\rceil=\lceil x\rceil$，$\lceil\lfloor x\rfloor\rceil=\lfloor x\rfloor$，$\lfloor\lceil x\rceil\rfloor=\lceil x\rceil$
  - 对于任何的整数$\boldsymbol{n}$，$n=\left\lfloor\frac{n}{2}\right\rfloor+\left\lceil\frac{n}{2}\right\rceil$
  - 对于任何的正整数$\boldsymbol{m}$和$\boldsymbol{n}$，$\left\lfloor\frac{\lfloor x/m\rfloor}{n}\right\rfloor=\left\lfloor\frac{x}{mn}\right\rfloor$，$\left\lceil\frac{\lceil x/m\rceil}{n}\right\rceil=\left\lceil\frac{x}{mn}\right\rceil$
  - 对于取模运算，$x \bmod y=x-y\left\lfloor\frac{x}{y}\right\rfloor$

调和数的复杂度

- 调和数定义为调和级数的前$\boldsymbol{n}$项和
  - $H(n)=\sum_{i=1}^{n}\frac{1}{i}=1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8}+\cdots+\frac{1}{n}$,

    $H(n)\le 1+\frac{1}{2}+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\cdots+\underbrace{\frac{1}{2^k}+\cdots+\frac{1}{2^k}}_{2^k\text{项},k=\lfloor\log n\rfloor}$,

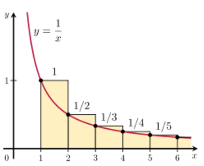
    $H(n)\le\underbrace{1+1\cdots+1}_{k+1\text{项},k=\lfloor\log n\rfloor}=\lfloor\log n\rfloor+1=O(\log n)$.
  - $H(n)\ge 1+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\cdots+\underbrace{\frac{1}{2^k}+\cdots+\frac{1}{2^k}}_{2^{k-1}\text{项},k=\lfloor\log n\rfloor}$,

    $H(n)\ge 1+\underbrace{\frac{1}{2}+\frac{1}{2}\cdots+\frac{1}{2}}_{k\text{项},k=\lfloor\log n\rfloor}=1+\frac{1}{2}\lfloor\log n\rfloor=\Omega(\log n)$.

因此：$H(n)=\Theta(\log n)$

调和级数的发散性

- 调和级数(Harmonic series)指的是如下发散的无穷级数
  - $\sum_{i=1}^{\infty}\frac{1}{i}=1+\frac{1}{2}+\frac{1}{3}+\cdots$



- 积分测试，参考右图
  - 所有矩形的面积和为调和级数
  - $y=\frac{1}{x}$曲线下从$x=1$到∞的面积为:
    $\int_1^{+\infty}\frac{1}{x}dx=\lim_{a\to+\infty}\int_1^a\frac{1}{x}dx=\lim_{a\to+\infty}\ln a=+\infty$.
  - 而$\sum_{i=1}^{\infty}\frac{1}{i}>\int_1^{\infty}\frac{1}{x}dx$，因此发散

调和数复杂度的积分求法

- $\sum_{i=1}^{n}\frac{1}{i}=1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$

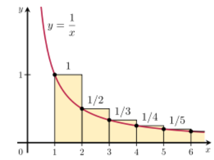
  $>\int_1^n\frac{1}{x}dx=\ln x|_1^n=\ln n=\frac{\log n}{\log e}$.

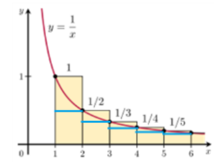
  $\therefore\sum_{i=1}^{n}\frac{1}{i}=\Omega(\log n)$.
- $\sum_{i=1}^{n}\frac{1}{i}=1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$

  $<1+\int_1^n\frac{1}{x}dx=\ln x|_1^n$

  $=1+\ln n=1+\frac{\log n}{\log e}$.

  $\therefore\sum_{i=1}^{n}\frac{1}{i}=O(\log n)$.
- $\therefore\sum_{i=1}^{n}\frac{1}{i}=\Theta(\log n)$





调和数的近似式

- 第$\boldsymbol{n}$项调和数定义为调和级数的前$\boldsymbol{n}$项部分和
  - $H_n=\sum_{i=1}^{n}\frac{1}{i}$
- 调和数的发散速率很慢，因为它是对数增长的
  - $H_n=\ln n+\gamma+\epsilon_n$，其中$\epsilon_n\sim\frac{1}{2n}$，当$n\to\infty$时，$\epsilon_n\to 0$
  - γ为欧拉-马歇罗尼常数(Euler–Mascheroni constant)
  - 该常数首先由欧拉于1734年提出，意大利数学家马歇罗尼(Lorenzo Mascheroni, 1750—1800)于1790年对它进行了进一步的计算

欧拉-马歇罗尼常数

- 欧拉-马歇罗尼常数之所以引起重视，是因为它在数学分析和数论中频繁出现
  - 鉴于其与Γ函数的重要关系将其记为γ
  - $\gamma=\lim_{n\to\infty}\left(\sum_{i=1}^{n}\frac{1}{i}-\ln n\right)=\lim_{b\to\infty}\int_1^b\left(\frac{1}{\lfloor x\rfloor}-\frac{1}{x}\right)dx$
  - $\gamma=0.57721566490153286060651209008240\cdots$
  - 目前尚不知道γ是代数数(algebraic number, 即可以是某个一元n次方程根的数)还是超越数(transcendental, 即不可能是某个一元n次方程根的数)
  - 目前也不知道γ是有理数还是无理数，仅知道如果它是有理数，则其分母必定大于10^{242080}
  - 由于γ在数学上的普适性，其有理性或无理性是数学中的一个重要问题，至今未解

莱昂哈德 保罗 欧拉(Leonhard Paul Euler)

- 瑞士数学家和物理学家，近代数学先驱之一。
- **18世纪杰出的数学家，同时也是有史以来最伟大的数学家之一。**
- 在数学的多个领域，包括微积分和图论都做出过重大发现。他引进的许多数学术语和书写格式，例如函数的记法"$f(x)$"，一直沿用至今。
- 在力学、光学和天文学等学科都有突出的贡献。

莱昂哈德·保罗·欧拉
1707年4月15日－
1783年9月18日

莱昂哈德 保罗 欧拉(Leonhard Paul Euler)

- 他也是一位多产作者，其文学著作约有60-80册。
- 法国数学家皮埃尔-西蒙 拉普拉斯曾这样评价欧拉对于数学的贡献：“读欧拉，再读欧拉！他是我们大家共同的大师！”
- 欧拉是史上发表论文数第二多的数学家，全集共计75卷；他的纪录一直到了20世纪才被保羅 埃尔德什打破。
- 尽管人生最后7年，欧拉的双目完全失明，他还是以惊人的速度产出了生平一半的著作。

前民主德国为纪念欧拉逝世200周年而发行的邮票。票面给出了著名的单连通多面体的边、顶点和面之间存在的关系，即欧拉公式：
$F-E+V=2$

莱昂哈德 保罗 欧拉(Leonhard Paul Euler)

- 欧拉是唯一一位以其名字命名了两个重要数学常数的人
  - **欧拉数(Euler's number)，即自然对数的底数：**
    $e=\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n=\sum_{n=0}^{\infty}\frac{1}{n!}=\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots$
    $e=\ 2.71828182845904523536\ldots\ldots$
  - **欧拉-马歇罗尼常数**
    $\gamma=\lim_{n\to\infty}\left(\sum_{i=1}^{n}\frac{1}{i}-\ln n\right)$=0.57721566490……
  - **以π表示圆周率也是由欧拉普及的**
    π=3.14159 26535 89793 23846 26433 ……

莱昂哈德 保罗 欧拉(Leonhard Paul Euler)

- 欧拉深入开展了幂级数的研究和应用
  - $e^x=\sum_{n=0}^{\infty}\frac{x^n}{n!}=\lim_{n\to\infty}\left(\frac{1}{0!}+\frac{x}{1!}+\frac{x^2}{2!}+\cdots+\frac{x^n}{n!}\right)$
- 于1735年年仅28岁时解决了著名的巴塞尔(以瑞士第三大城市，欧拉和伯努利家族的家乡)问题
  - $\sum_{n=0}^{\infty}\frac{1}{n^2}=\lim_{n\to\infty}\left(\frac{1}{1^2}+\frac{1}{2^2}+\cdots+\frac{1}{n^2}\right)=\frac{\pi^2}{6}$
- 欧拉证明了所有素数的倒数之和是发散的，从而强化了欧几里德定理
  - $\sum_{p\text{为素数}}\frac{1}{p}=\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\frac{1}{11}+\cdots=\infty$

莱昂哈德 保罗 欧拉(Leonhard Paul Euler)

- 欧拉定义了复数的指数形式，并发现了它与三角函数之间的关系，即著名的欧拉公式
  - $e^{i\varphi}=\cos\varphi+i\sin\varphi,\forall\varphi\in R$
  - 当$\varphi=\pi$时，可得著名的欧拉等式：$e^{i\pi}+1=0$
- 著名的物理学家理查德·费曼称欧拉公式为“最卓越的数学公式”，因为它将重要的数学常数0、1、e、i和π各取一次，以常用的加法、乘法、指数和相等等运算各一次组成了一个公式
  - **1988年，《数学信使》(The Mathematical Intelligencer)杂志的读者通过投票，将欧拉公式确定为“有史以来最漂亮的数学公式”**

目录

- 递归
- 分治方法
- 算法分析的数学基础