1.3, 1.7, 1.18, 1.20

算法设计与分析

第一章：算法基础知识（1）

算法课程主要内容及有关概念

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

两个例子:调度问题与投资问题

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

例1：调度问题

Finishing time=Waiting time + Processing time

问题： 有 n 项任务，每项任务加工时间已知. 从 0 时刻开始陆续安排（这n 项任务）到一台机器上加工.

每个任务的完成时间是从 0 时刻到任务加工完成的时间.

求: 总完成时间（所有任务完成时间之和）最短的安排方案.

实例

任务集 $S = \{1, 2, 3, 4, 5\}$，

加工时间：$t_1=3$，$t_2=8$，$t_3=5$，$t_4=10$，$t_5=15$

贪心法的解

实例

任务集 $S = \{1, 2, 3, 4, 5\}$,

加工时间：$t_1=3$, $t_2=8$, $t_3=5$, $t_4=10$, $t_5=15$

算法：按加工时间 (3, 8, 5, 10, 15) 从小到大安排

解：**1, 3, 2, 4, 5**

总完成时间：

$$t = 3+(3+5)+(3+5+8)+(3+5+8+10)+(3+5+8+10+15)$$
$$= 3\times\underline{5} + 5\times\underline{4} + 8\times\underline{3} + 10\times\underline{2} + 15\times\underline{1}$$
$$= 94$$

加工时间的倍数n-(k-1): n 项任务; 任务被排在第k个位置

调度问题建模

输入： 任务集： $S = \{1, 2, \ldots, n\}$,

第 j 项任务加工时间： $t_j \in Z^+$, $j = 1, 2, \ldots, n$.

输出： 调度 I， S 的排列 $i_1, i_2, \ldots, i_n$,

目标函数： I 的完成时间， $\boxed{t(I) = \sum_{k=1}^{n}(n-k+1)t_{i_k}}$

解 I^*： 使得 $t(I^*)$ 达到最小，即

$$t(I^*)=\min\{t(I) \mid I \text{ 为} S \text{的排列}\}$$

贪心算法

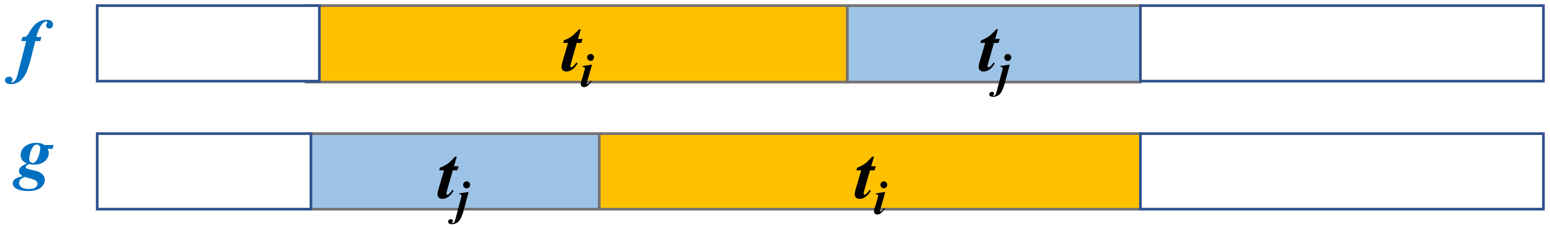
$$t(I)=\sum_{k=1}^{n}(n-k+1)t_{i_k}$$

设计策略：加工时间短的先做

算法：根据加工时间从小到大排序,依次加工

算法正确性：对调度问题的所有输入实例都得到最优解

证：假如调度 f 第 i, j 项任务相邻且有逆序，即 $t_i>t_j$.
交换任务 i 和 j 得到调度 g,



两种调度总完成时间之差： $t(g)-t(f)=t_j-t_i<0$ ■

“直觉”不一定是正确的

反例:

有4 件物品要装入背包, 物品重量和价值如下:

标号	1	2	3	4
重量 w_i	3	4	5	2
价值 v_i	7	9	9	2

背包重量限制是 6,

问: 如何选择物品，使得不超重的情况下装入背包的物品价值达到最大?

实例的解

标号	1	2	3	4
重量 w_i	3	4	5	2
价值 v_i	7	9	9	2

贪心法：单位重量价值大的优先，总重不超 6

按照 $\dfrac{v_i}{w_i}$ 从大到小排序：1, 2, 3, 4

$$\boxed{\frac{7}{3}} > \boxed{\frac{9}{4}} > \frac{9}{5} > \boxed{\frac{2}{2}}$$

贪心法的解：**{1, 4}**，重量 **5**，价值为 **9.**

更好的解：**{2, 4}**，重量 **6**，价值 **11.**

■

算法设计

1. 问题建模
2. 选择什么算法？如何描述这个方法？
3. 这个方法是否对所有实例都得到最优解？如何证明？
4. 如果不是，能否找到反例？

调度问题

背包问题

例2：投资问题

问题： m元钱，投资 n 个项目. 效益函数 $f_i(x_i)$，表示第 i 个项目投 x_i 元的效益，$i=1, 2, \ldots, n$.

求如何分配每个项目的钱数使得总效益最大？

实例： 5 万元，投资给 4 个项目，

效益函数 $f_i(x_i)$：$i=1, 2, 3, 4$.

x_i	$f_1(x_1)$	$f_2(x_2)$	$f_3(x_3)$	$f_4(x_4)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

投资问题建模

输入： $n, m, f_i(x_i)$, $i = 1,2, \ldots, n$, $x_i = 0,1,2, \ldots, m$ (x_i非负整数，$0 \le x_i \le m$)

解： n 维向量$\langle x_1, x_2, \ldots, x_n \rangle$, x_i 是第 i 个项目的钱数，使得下述条件满足：

目标函数 $$\max \sum_{i=1}^{n} f_i(x_i),$$

约束条件 $$\sum_{i=1}^{n} x_i = m, \quad x_i \in \mathrm{N}$$

蛮力算法（Brute Force）

对所有满足下述条件的向量 $<x_1, \ldots, x_i, \ldots, x_n>$

$$x_1 + \ldots + x_i + \ldots + x_n = m$$

x_i 为非负整数, $i = 1, 2, \ldots, n$

计算相应的效益

$$f_1(x_1) + \ldots + f_i(x_i) + \ldots + f_n(x_n)$$

从中确认效益最大的向量.

实例计算

实例： 5 万元，投资给 4 个项目，效益函数 $f_i(x_i)$： i =1, 2, 3, 4.

x	$f_1(x_1)$	$f_2(x_2)$	$f_3(x_3)$	$f_4(x_4)$
0	0	$\boxed{0}$	0	0
1	$\boxed{11}$	0	2	$\boxed{20}$
2	12	5	10	21
3	13	10	$\boxed{30}$	22
4	14	15	32	23
5	15	20	40	24

$$x_1 + x_2 + x_3 + x_4 = 5$$

s_1=<0,0,0,5>, $v(s_1)$=24

s_2=<0,0,1,4>, $v(s_2)$=25

s_3=<0,0,2,3>, $v(s_3)$=32

…

s_{56}=<5,0,0,0>,$v(s_{56})$=15

解： s = <1, 0, 3, 1>,

最大效益： 11+30+20 = 61

蛮力算法的效率 (1/2)

方程 $x_1 + x_2 + \ldots + x_n = m$ 的非负整数解

一个可行解（分配）的加法的次数：n - 1

$< x_1, x_2, \ldots, x_n >$ 的个数估计：

可行解表示成 0-1 序列：m 个1, n-1个 0

$$\underbrace{1\ldots1}_{x_1\text{个}}\;0\;\underbrace{1\ldots1}_{x_2\text{个}}\;0\ldots\;0\;\underbrace{1\ldots1}_{x_n\text{个}}$$

实例：n=4, m=7；可行解 $<1, 2, 3, 1>$

$\Leftrightarrow$ 序列 1 0 1 1 0 1 1 1 0 1

蛮力算法的效率 (2/2)

0-1序列个数是输入规模的指数函数

$$C(m+n-1,m)$$
$$=\frac{(m+n-1)!}{m!(n-1)!}$$
$$=\Omega((1+\varepsilon)^{m+n-1}) \quad (\varepsilon > 0)$$

指数函数

例2：投资问题-实例

n=4，m=5

$$C_{5+4-1}^{5}=\frac{8!}{5!\,3!}=56$$

有没有更好的算法?

小 结

问题求解的关键

- 问题建模：对输入参数和解给出形式化或半形式化的描述
- 设计算法：

  采用什么算法设计技术

  正确性——是否对所有的问题实例都得到正确的解
- 分析算法——效率

问题计算复杂度的界定:排序问题

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

例3 排序问题算法的效率

以元素比较作基本运算

算法	最坏情况下	平均情况下
插入排序	$O(n^2)$	$O(n^2)$
冒泡排序	$O(n^2)$	$O(n^2)$
快速排序	$O(n^2)$	$O(n\log n)$
堆排序	$O(n\log n)$	$O(n\log n)$
二分归并排序	$O(n\log n)$	$O(n\log n)$

插入排序的插入操作

前面已经排好，插入2

输入

5	7	1	3	6	2	4

插入2

1	3	5	6	7	2	4

插入后

1	2	3	5	6	7	4

插入排序运行实例

输入	5	7	1	3	6	2	4
初始	5	7	1	3	6	2	4
插入7	5	7	1	3	6	2	4
插入1	1	5	7	3	6	2	4
插入3	1	3	5	7	6	2	4
插入6	1	3	5	6	7	2	4
插入2	1	2	3	5	6	7	4
插入4	1	2	3	4	5	6	7

冒泡排序的一次巡回

巡回前：

5	1	6	2	8	3	4	7

一次巡回：

5	1	6	2	8	3	4	7

巡回后：

1	5	2	6	3	4	7	8

冒泡排序运行实例

	5	8	1	3	6	2	4	7
巡回1	5	1	3	6	2	4	7	8
巡回2	1	3	5	2	4	6	7	8
巡回3	1	3	2	4	5	6	7	8
巡回4	1	2	3	4	5	6	7	8
巡回5	1	2	3	4	5	6	7	8

快速排序一次递归运行

	5	8	1	3	6	2	4	7
交换1	5	8	1	3	6	2	4	7
交换2	5	4	1	3	6	2	8	7
划分	5	4	1	3	2	6	8	7
子问题	2	4	1	3	5	6	8	7

二分归并排序运行实例

5	8	1	3	6	2	4	7

划分

5	8	1	3	6	2	4	7

递归排序

1	3	5	8	2	4	6	7

1	3	5	8	2	4	6	7

合并后的输出

问题(Problem)的计算复杂度分析

问题（questions）：

→ 哪个排序算法效率最高?

→ 是否可找到更好的排序算法?

→ 排序问题计算难度如何?

→ 其他问题的计算复杂度

→ 问题计算复杂度**估计方法**

排序问题算法的最坏情况下

n^2	插入排序 冒泡排序 快速排序
$n\log n$	堆排序 归并排序
?	更好的算法下界

那个排序算法效率最高?

排序问题的难度?

小结

- 几种排序算法简介

  插入排序

  冒泡排序

  快速排序

  归并排序

- 排序问题的难度估计——界定什么是最好的排序算法

货郎问题与计算复杂性理论

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

例4 货郎问题

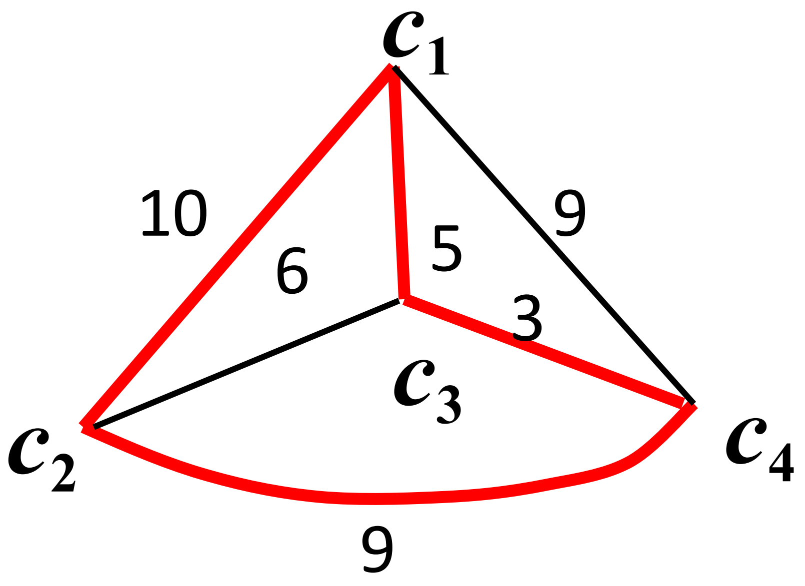
处在能不能有效计算的边界状态

问题：

有n个城市，已知任两个城市之间的距离. 求一条每个城市恰好经过1次的回路，使得总长度最小.

实例：

有4个城市，连线上的数字表示两个城市之间的距离. 求一条每个城市恰好经过1次的回路，使得总长度最小.



货郎问题：建模与算法

- 输入

  有穷个城市的集合 $C = \{c_1, c_2, \ldots, c_n\}$，

  距离 $d(c_i, c_j) = d(c_j, c_i) \in \mathrm{Z}^+$，$1 \le i < j \le n$

- 解：$1, 2 \ldots, n$ 的排列 $k_1, k_2, \ldots, k_n$ 使得：

$$\boxed{\min\{\sum_{i=1}^{n-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_n}, c_{k_1})\}}$$

- 现状：至今没找到有效的（多项式时间）算法

0-1背包问题

0-1背包问题：有n件物品要装入背包，第i件物品的重量 w_i, 价值v_i，$i=1, 2, \ldots, n$. 背包最多允许装入的重量为B, 问如何选择装入背包的物品，使得总价值达到最大?

实例：$n=4$，$B=6$，物品的重量和价值如下

标号	1	2	3	4
重量 w_i	3	4	5	2
价值 v_i	7	9	9	2

0-1背包问题建模

输入：有n件物品，第i件物品的重量w_i, 价值v_i, $i=1, 2, \ldots, n$.
背包最多允许装入的重量为B

解：0-1向量 $<x_1, x_2, \ldots, x_n>$

$x_i=1 \Leftrightarrow$ 物品 i 装入背包　$i=1, 2, \ldots, n$.

目标函数　$\max \sum_{i=1}^{n} v_i x_i$

约束条件　$\sum_{i=1}^{n} w_i x_i \le B$

$x_i = 0,1, i = 1,2,\ldots,n$

双机调度问题

完成时间：从任务角度
完工时间：从机器角度

双机调度问题：有n项任务，任务 i 的加工时间为 t_i, $t_i \in Z^+$, i=1, 2,…, n. 用两台相同的机器加工，从0时刻开始计时，完工时间是后停止加工机器的停机时间．问如何把这些任务分配到两台机器上，使得完工时间达到最小？

实例：任务集 S ={1, 2, 3, 4, 5, 6}

t_1=3, t_2=10, t_3=6, t_4=2, t_5=1, t_6=7

解：机器 1 的任务：1, 2, 4

机器 2 的任务：3, 5, 6

完工时间 ： max{3+10+2, 6+1+7}=15

Makespan

双机调度问题建模

输入: 有n项任务, 任务 i 的加工时间为 t_i, $t_i \in Z^+$, i=1, 2,…, n. 用两台相同的机器加工,从0时刻开始计时,完工时间是后停止加工机器的停机时间.

解: 0-1向量 $<x_1, x_2, ..., x_n>$, x_i=1表示任务 i分配到第一台机器,

i =1, 2,..., n.

不妨设：机器1的加工时间 ≤ 机器2的加工时间

令: $T=t_1+t_2+...+t_n$, $D=\lfloor T/2 \rfloor$,

机器1的加工时间不超过D，且达到最大.

如何对该问题建模?目标函数与约束条件是什么？

NP-hard问题

- 这样的问题有数千个，大量存在于各个应用领域.
- 至今没找到有效算法(多项式时间的算法)：现有的算法的运行时间是输入规模的指数或更高阶函数.
- 至今没有人能够证明对于这类问题不存在多项式时间的算法.
- 从是否存在多项式时间算法的角度看，这些问题彼此是等价的. 这些问题的难度处于可有效计算和难解问题的边界.

Algorithm + Data Structure = Programming

好的算法

- ✓ 提高求解问题的效率
- ✓ 节省存储空间

算法的研究目标

□ 问题→建模并寻找算法	算法设计技术
□ 算法→算法的评价	算法分析方法
□ 算法类→问题复杂度估计	问题复杂度分析
□ 问题类→能够有效求解的边界	计算复杂性理论

课程主要内容

近似算法 | 随机算法

NP 完全理论

算法分析与问题的计算复杂性

分治策略 | 动态规划 | 贪心算法 | 回溯 & 分支限界 | 线性规划 | 网络流 | 强化学习

算法基本概念、数学基础

计算复杂性理论:

NP完全理论

其他算法

算法设计:

算法分析方法

算法设计技术

基础知识

算法研究的重要性

算法设计与分析技术在计算机科学与技术领域有着重要的应用背景

算法设计分析与计算复杂性理论研究是计算机科学技术的核心研究领域

- **1966-2021期间，Turing奖获奖72人, 其中18人以算法设计, 12人以计算理论、自动机和复杂性研究领域的杰出贡献获奖**
- **计算复杂性理论的核心课题 “P=NP?” 是本世纪 7个最重要的数学问题之一**

培养计算思维，分析问题和解决问题的能力，提高学生理论素质

小结

- NP-hard问题的三个例子：货郎问题
  0-1背包问题、双机调度问题等
- NP-hard问题的计算现状
- 计算复杂性理论的核心 — NP完全理论
- 算法研究的主要内容及重要意义

算法及其时间复杂度

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

问题及实例

- 问题（Problem）

  需要回答的一般性提问，通常含若干参数

- 问题描述

  → 定义问题参数（集合,变量,函数,序列等）

  → 说明每个参数的取值范围及参数间的关系

  → 定义问题的解

  → 说明解满足的条件（优化目标或约束条件）

- 问题实例

  参数的一组赋值可得到问题的一个实例

算法

- 算法

  → 有限条指令的序列

  → 这个指令序列确定了解决某个问题的一系列运算或操作

- 算法 A 解问题 P

  把问题 P 的任何一个实例作为算法 A 的输入

  → 每步计算是确定性的

  → 算法 A 能够在有限步停机

  → 输出该实例的正确的解

基本运算与输入规模

- 算法时间复杂度: 针对指定基本运算，计数算法所做运算次数
- 基本运算: 比较, 加法, 乘法, 置指针, 交换…

- 输入规模： 输入串编码长度
  通常用下述参数度量：数组元素多少，调度问题的任务个数，图的顶点数与边数等.

第九章

- 算法基本运算次数可表为输入规模的函数
- 给定问题和基本运算就决定了一个算法类

输入规模

- 排序：数组中元素个数 n
- 检索：被检索数组的元素个数 n
- 整数乘法：两个整数的编码位数 m, n
- 矩阵相乘：矩阵的行列数 i, j, k
- 图的遍历：图的顶点数 n, 边数 m

…

基本运算

- 排序: 元素之间的比较
- 检索: 被检索元素 x与数组元素的比较

- 整数乘法: 每位数字相乘(位乘) 1 次

  m位和n位整数相乘要做mn次位乘

- 矩阵相乘: 每对元素乘 1 次

  $i\times j$矩阵与 $j\times k$ 矩阵相乘要做 $i\,jk$ 次乘法

- 图的遍历: 置指针

  …

算法的两种时间复杂度

对于相同输入规模的不同实例，算法的基本运算次数也不一样，可定义两种时间复杂度

最坏情况下的时间复杂度 $W(n)$

算法求解输入规模为 n 的实例所需要的最长时间

平均情况下的时间复杂度 $A(n)$

在给定同样规模为 n 的输入实例的概率分布下，算法求解这些实例所需要的平均时间

$A(n)$ 计算公式

平均情况下的时间复杂度 $A(n)$

设 S 是规模为 n 的实例集

实例 $I \in S$ 的概率是 P_I

算法对实例 I 执行的基本运算次数是 t_I

$$\boxed{A(n) = \sum_{I \in S} P_I t_I}$$

在某些情况下可以假定每个输入实例概率相等，简化计算

例子：检索问题

检索问题

输入：非降顺序排列的数组 L, 元素数 n，

数 x

输出：j

若 x 在 L中, j是x首次出现的下标；

否则 $j = 0$

基本运算：x 与 L 中元素的比较

顺序检索算法

j=1, 将x与$L[j]$比较. 如果 $x=L[j]$，则算法停止，输出 j；如果不等，则把 j 加1，继续x与$L[j]$的比较，如果 $j>n$，则停机并输出0.

实例

1	2	3	4	5

$x = 4$，需要比较 4 次

$x = 2.5$，需要比较 5 次

最坏情况的时间估计

最坏情况下的时间复杂度 $W(n)$
算法求解输入规模为 n 的实例中所需要的最长时间

输入规模为n的不同的输入实例共有 $2n+1$个，分别对应：

n 个：$x = L[1]$, $x = L[2]$, … , $x = L[n]$

$n+1$个：$x<L[1]$, $L[1]<x<L[2]$, $L[2]<x<L[3]$, … , $L[n]<x$

最坏情况下时间：$W(n) = n$

最坏的输入：（1）x 不在 L中，或（2） $x = L[n]$

两种情形都要做 n 次比较

平均情况的时间估计

$$A(n)=\sum_{I\in S} P_I t_I$$

$A(n)$ 计算公式

输入实例的概率分布：

假设x在L中概率是 p, 且每个位置概率相等

$$A(n)=\sum_{i=1}^{n} i\frac{p}{n}+(1-p)n$$

$$\frac{1-p}{n+1}n(n+1)$$

等差数列求和

$$=\frac{p(n+1)}{2}+(1-p)n$$

当 $p=1/2$时，

$$A(n)=\frac{n+1}{4}+\frac{n}{2}\approx\frac{3n}{4}$$

改进顺序检索算法

j=1, 将 x与$L[j]$比较. 如果 x=$L[j]$，则算法停止，输出 j；如果 $x > L[j]$，则把 j 加1，继续 x与 $L[j]$的比较；如果 $x < L[j]$，则停机并输出0. 如果 $j > n$，则停机并输出 0.

实例

1	2	3	4	5

$x = 4$，需要比较 4 次

$x = 2.5$，需要比较 3 次

时间估计

最坏情况下： $W(n) = n$

平均情况下：输入实例的概率分布：假设 x 在 L 中每个位置与空隙的概率都相等

改进检索算法平均时间复杂度是多少?

$$A(n) = \sum_{i=1}^{n} i\frac{p}{n} + \sum_{i=1}^{n} i\frac{1-p}{n+1} + \frac{1-p}{n+1}n$$

$$= \frac{p(1+n)}{2} + n(1-p)(\frac{1}{2} + \frac{1}{n+1})$$

小结：

- 算法最坏和平均情况下的时间复杂度定义
- 如何计算上述时间复杂度

算法的伪码表示

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

算法的伪码描述

赋值语句：←

分支语句：if …then … [else…]

循环语句：while，for， repeat until

转向语句：goto

输出语句：return

调用：直接写过程的名字

注释：//…

例：求最大公约数(GCD, Greatest Common Divisor)

算法 Euclid (m, n)

输入：非负整数 m, n,其中m与n不全为0

输出：m 与 n 的最大公约数

1. while $m > 0$ do
2. $\quad r \leftarrow n \bmod m$
3. $\quad n \leftarrow m$
4. $\quad m \leftarrow r$
5. return n

定理：$\gcd(a, b) = \gcd(b, a \bmod b)$

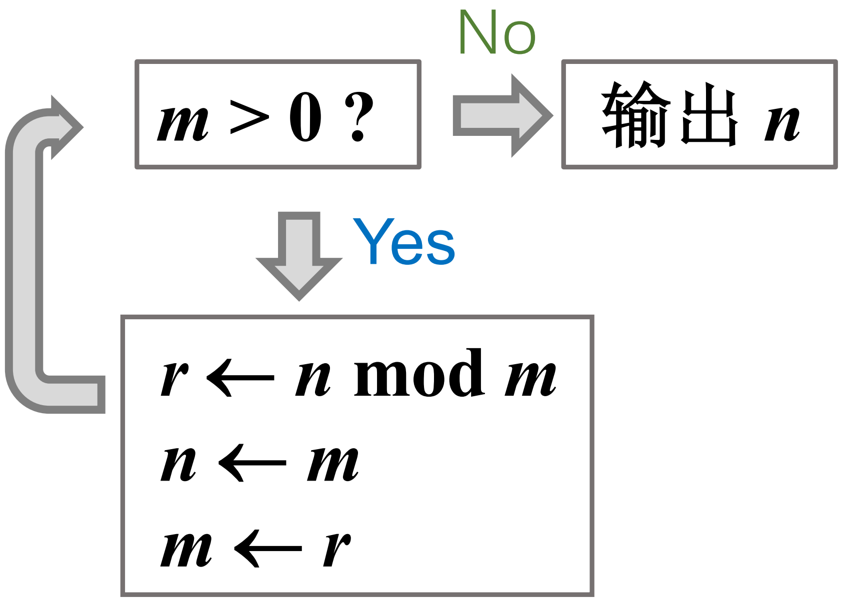
欧几里德算法，又称辗转相除法，用于计算两个整数a,b的最大公约数。

运行实例：n=36，m=15

while	n	m	r
第1次	36	15	6
第2次	15	6	3
第3次	6	3	0
	3	0	0

⇩

输出3



例：改进的顺序检索

算法 Search (L, x)

输入：数组 $L[1..n]$, 元素从小到大排列, 数 x.

输出：若 x 在 L中，输出 x 的位置下标 j;

否则输出0.

1. $j \leftarrow 1$
2. while $j \le n$ and $x > L[j]$ do $j \leftarrow j+1$
3. if $x < L[j]$ or $j > n$ then $j \leftarrow 0$
4. return j

例：插入排序

算法 Insert Sort (A, n)

输入：n个数的数组 A

输出：按照递增顺序排好序的数组 A

1. for $j \leftarrow 2$ to n do
2. $\quad x \leftarrow A[j]$
3. $\quad i \leftarrow j-1$ //3-7 行把 $A[j]$ 插入$A[1..j-1]$
4. $\quad$ while $i > 0$ and $x < A[i]$ do
5. $\qquad A[i+1] \leftarrow A[i]$
6. $\qquad i \leftarrow i-1$
7. $\quad A[i+1] \leftarrow x$

运行实例

2	4	1	5	3

$j = 3,\ x = A[3] = 1$

$i = 2,\ \ A[2] = 4$

$i > 0,\ x < A[2]$ √

2	4	4	5	3

$A[3]=4,\ i = 1, x=1$

$i > 0,\ x < A[1]$ √

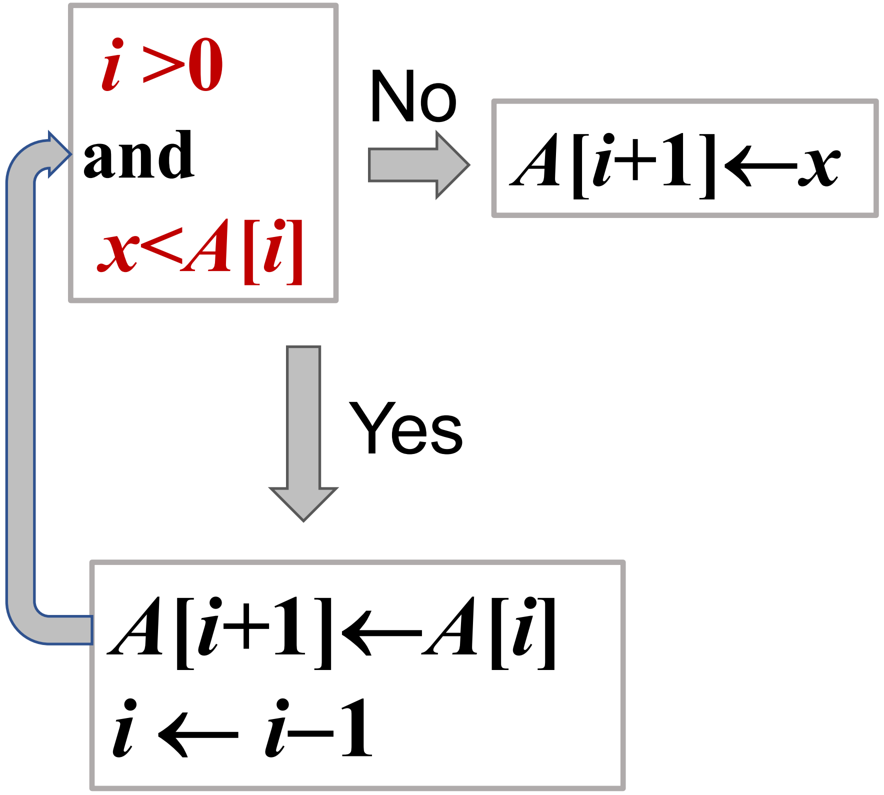
2	2	4	5	3

$A[2]=2,\ i = 0, x=1$

$i > 0$ ×

1	2	4	5	3

4. **while** $i>0$ **and** $x<A[i]$ **do**
5. $\quad A[i+1] \leftarrow A[i]$
6. $\quad i \leftarrow i-1$
7. $A[i+1] \leftarrow x$



例：二分归并排序

MergeSort (A, p, r)

输入：数组 $A[p..r]$

输出：按递增顺序排序的数组 A

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. **MergeSort** (A, p, q)
4. **MergeSort** $(A, q+1, r)$
5. **Merge** (A, p, q, r)

MergeSort：既有递归调用,也有过程**Merge**调用

例：算法A的伪码

算法 A

输入：实数的数组 $P[0..n]$，实数 x

输出：y

1. $y \leftarrow P[0]$；$power \leftarrow 1$
2. for $i \leftarrow 1$ to n do
3. $\quad power \leftarrow power * x$
4. $\quad y \leftarrow y + P[i] * power$
5. return y

算法A计算什么值？

对 $i = 1, 2, \ldots, n$ ⇨

$$power \leftarrow power * x$$
$$y \leftarrow y + P[i] * power$$

	i	$power$	y
初值		1	$P[0]$
循环	1	x	$P[0] + P[1]*x$
	2	x^2	$P[0] + P[1]*x + P[2]*x^2$
	3	x^3	$P[0] + P[1]*x + P[2]*x^2 + P[3]*x^3$
			…

输入 $P[0..n]$是 n 次多项式函数 $P(x)$的系数

算法 A 计算该多项式在 x 的值

小结

用伪码表示算法

- 伪码不是程序代码，只是给出算法的主要步骤
- 伪码中有哪些关键字？
- 伪码中允许过程调用

函数的渐近的界

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：函数渐近的界

5 算法的伪码表示

4 算法及其时间复杂度的定义

大O符号

定义：设 f 和 g是定义域为自然数集N上的函数.

若存在正数 c 和 n_0，使得对一切 $n \geq n_0$有

$$0 \leq f(n) \leq c\,g(n)$$

成立, 则称 $f(n)$ 的渐近的上界是 $g(n)$，记作

$$f(n) = O(g(n))$$

例子

设 $f(n) = n^2 + n$，则

$f(n)=O(n^2)$，$g(n)= n^2$，取 $c = 2$，$n_0 =1$ 即可

$f(n)=O(n^3)$，$g(n)= n^3$，取 $c = 1$，$n_0 =2$ 即可

1. $f(n) = O(g(n))$，$f(n)$的阶不高于$g(n)$的阶.
2. 可能存在多个正数c，只要指出一个即可.
3. 对前面有限个值可以不满足不等式.
4. 常函数可以写作$O(1)$.

大Ω符号

定义：设 f 和 g是定义域为自然数集N上的函数.

若存在正数 c 和 n_0，使得对一切 $n \geq n_0$ 有

$$0 \leq cg(n) \leq f(n)$$

成立，则称 $f(n)$ 的渐近的下界是 $g(n)$，记作

$$f(n) = \Omega(g(n))$$

例子

设 $f(n) = n^2 + n$，则

$f(n) = \Omega(n^2)$， $g(n)= n^2$，取 $c = 1, n_0 = 1$即可

$f(n) = \Omega(100n)$, $g(n)= 100n$，取 $c=1/100$, $n_0 = 1$即可

1. $f(n)=\Omega(g(n))$，$f(n)$的阶不低于$g(n)$的阶.

2. 可能存在多个正数c，指出一个即可.

3. 对前面有限个 n 值可以不满足上述不等式.

小 o 符号

定义 设 f 和 g是定义域为自然数集 N上的函数.

若对于任意正数 c 都存在 n_0，使得对一切 $n \geq n_0$有

$$0 \leq f(n) < c\,g(n)$$

成立, 则记作

$$f(n) = o(g(n))$$

例子

例子：$f(n)=n^2+n$，则　$f(n)=o(n^3)$，　$g(n)= n^3$

$c\geq 1$显然成立，因为$n^2+n<cn^3$（$n_0=2$）

任给$1>c>0$, 取 $n_0>\lceil 2/c\rceil$ 即可. 因为

$$cn \geq cn_0 > 2 \qquad （当 n \geq n_0）$$

$n>2/c,\ n>2$

$$n^2+n < 2n^2 < cn^3$$

$2< cn$

1. $f(n) = o(g(n))$, $f(n)$的阶低于$g(n)$的阶
2. 对不同正数c, n_0不一样. c越小n_0越大.
3. 对前面有限个 n 值可以不满足不等式.

小 ω 符号

定义：设 f 和 g是定义域为自然数集 N上的函数.

若对于任意正数 c 都存在 n_0，使得对一切 $n \geq n_0$有

$$0 \leq cg(n) < f(n)$$

成立, 则记作

$$f(n) = \omega(g(n))$$

例子

设 $f(n) = n^2 + n$，则

$$f(n) = \omega(n), \quad g(n) = n$$

不能写 $f(n) = \omega(n^2)$， $g(n) = n^2$，因为取 $c = 2$，不存在 n_0 使得对一切 $n \geq n_0$ 有下式成立

$$c\,n^2 = \boxed{2n^2 < n^2 + n \quad \times}$$

1. $f(n)=\omega(g(n))$, $f(n)$的阶高于$g(n)$ 的阶.
2. 对不同的正数c, n_0不等， c 越大n_0 越大.
3. 对前面有限个 n 值可以 不满足不等式.

Θ 符号

若 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$, 则记作

$$f(n) = \Theta(g(n))$$

例子：$f(n) = n^2 + n$, $g(n) = 100n^2$，那么有

$$f(n) = \Theta(g(n))$$

1. $f(n)$ 的阶与 $g(n)$ 的阶相等.
2. 对前面有限个 n 值可以不满足条件.

例子：素数测试 (试除法，埃拉托斯特尼筛法， Sieve of Eratosthenes)

算法PrimalityTest(n)

输入：n, 大于2的奇整数

输出：**true** 或者**false**

1．$s \leftarrow \lfloor n^{1/2} \rfloor$

2．**for** $j \leftarrow 2$ **to** s

3．　　**if** j **整除** n

4．　　**then return false**

5．**return true**

问题：

若$n^{1/2}$ 可在$O(1)$计算，基本运算是整除，以下表示是否正确？

$W(n)=O(n^{1/2})$

$W(n)=\Theta(n^{1/2})$

为什么？

(1)以伪码行3的除法作为基本运算，行2的循环最多执行$(s-1)$次，于是$W(n)=O(s)=O(\sqrt{n})$

(2) 对于大素数n，算法确实需要做$\lfloor\sqrt{n}\rfloor$次除法。但是如果$n=3k$，k为任意大的奇整数，那么算法只需2次除法；因此不存在常数c和n_0，使得当$n \geq n_0$时都有$c\lfloor\sqrt{n}\rfloor \leq W(n)$成立，即$\sqrt{n}$不是$W(n)$的渐近的下界。

小结

- 五种表示函数的阶的符号

  $\boldsymbol{O}, \boldsymbol{\Omega}, \boldsymbol{o}, \boldsymbol{\omega}, \boldsymbol{\Theta}$

- 五种符号的定义
- 如何用定义证明函数的阶？

有关函数渐近的界的定理

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

定理 1

定理 设 f 和 g 是定义域为自然数集合 N 上的函数.

(1) 如果 $\lim_{n\to\infty} f(n)/g(n)$ 存在, 并且等于某个常数 $c>0$, 那么

$$f(n) = \Theta(g(n)).$$

(2) 如果 $\lim_{n\to\infty} f(n)/g(n) = 0$，那么

$$f(n) = o(g(n)).$$

(3) 如果 $\lim_{n\to\infty} f(n)/g(n) = +\infty$，那么

$$f(n) = \omega(g(n)).$$

证明用到 Θ, o, ω 定义

证明定理 1 (1)

根据极限定义，对于给定正数 ε 存在某个n_0，只要$n > n_0$，就有

$$|f(n)/g(n)-c| < \varepsilon$$

$$c-\varepsilon < f(n)/g(n) < c+\varepsilon$$

$\boxed{\text{取}\ \varepsilon = c/2,}$

$$c/2 < f(n)/g(n) < 3c/2 < 2c$$

对所有$n \geq n_0$, $f(n) \leq 2cg(n)$, 于是 $f(n)=O(g(n))$;

对所有$n \geq n_0$, $f(n) \geq (c/2)g(n)$, 于是 $f(n)=\Omega(g(n))$.

从而 $f(n) = \Theta(g(n))$. ■

例：估计函数的阶

例1 设 $f(n)=\frac{1}{2}n^2-3n$ ，证明 $f(n)=\Theta(n^2)$.

证 因为

$$\lim_{n\to\infty}\frac{f(n)}{n^2}=\lim_{n\to\infty}\frac{\frac{1}{2}n^2-3n}{n^2}=\frac{1}{2}$$

根据定理1，有 $f(n)=\Theta(n^2)$.

■

一些重要结果

定理 1

定理 设 f 和 g 是定义域为自然数集合的函数.

(1) 如果 $\lim_{n\to\infty} f(n)/g(n)$ 存在, 并且等于某个常数 $c>0$, 那么 $f(n)=\Theta(g(n))$.

(2) 如果 $\lim_{n\to\infty} f(n)/g(n)=0$, 那么 $f(n)=o(g(n))$.

(3) 如果 $\lim_{n\to\infty} f(n)/g(n)=+\infty$, 那么 $f(n)=\omega(g(n))$.

可证明：多项式函数的阶低于指数函数的阶

$$\boxed{n^d = o(r^n),\ r>1,\ d>0}$$

证 不妨设 d 为正整数，

$$\lim_{n\to\infty}\frac{n^d}{r^n}=\lim_{n\to\infty}\frac{dn^{d-1}}{r^n\ln r}=\lim_{n\to\infty}\frac{d(d-1)n^{d-2}}{r^n(\ln r)^2}$$

$$=\ldots=\lim_{n\to\infty}\frac{d!}{r^n(\ln r)^d}=0$$

分子分母
求导数

■

一些重要结果(续)

定理 1

定理　设 f 和 g 是定义域为自然数集合的函数.

(1) 如果 $\lim_{n\to\infty} f(n)/g(n)$ 存在, 并且等于某个常数 $c>0$，那么 $f(n)=\Theta(g(n))$.

(2) 如果 $\lim_{n\to\infty} f(n)/g(n)=0$，那么 $f(n)=o(g(n))$.

(3) 如果 $\lim_{n\to\infty} f(n)/g(n)=+\infty$，那么 $f(n)=\omega(g(n))$.

可证明：对数函数的阶低于幂函数的阶

$$\boxed{\ln n = o(n^d),\ d>0}$$

证

$$\lim_{n\to\infty}\frac{\ln n}{n^d}=\lim_{n\to\infty}\frac{\frac{1}{n}}{dn^{d-1}}$$

分子分母求导数

$$=\lim_{n\to\infty}\frac{1}{dn^d}=0$$

■

定理 2

定理 设函数f, g, h的定义域为自然数集合N，

(1) 如果 $f=O(g)$ 且 $g=O(h)$，那么 $f=O(h)$.

(2) 如果 $f=\Omega(g)$ 且 $g=\Omega(h)$，那么 $f=\Omega(h)$.

(3) 如果 $f=\Theta(g)$ 和 $g=\Theta(h)$，那么 $f=\Theta(h)$

函数的阶之间的关系具有**传递性**

例子

按照阶从高到低排序以下函数：

$f(n)=(n^2+n)/2$，$g(n)=10n$

$h(n)=1.5^n$ ，$t(n)=n^{1/2}$

$$h(n) = \omega(f(n)),$$
$$f(n) = \omega(g(n)),$$
$$g(n) = \omega(t(n)),$$

排序 $h(n), f(n), g(n), t(n)$

定理3

定理 假设函数f和g的定义域为自然数集N，若对某个其它函数 h, 有 $f=O(h)$ 和 $g=O(h)$，那么

$$f+g=O(h).$$

该性质可以推广到有限个函数.

算法由有限步骤构成. 若每一步的时间复杂度函数的上界都是 $h(n)$，那么该算法的时间复杂度函数可以写作 $O(h(n))$.

小结

- 估计函数的阶的方法：

  → 计算极限： Θ, o, ω

  → 阶具有传递性

- 对数函数的阶低于幂函数的阶，多项式函数的阶低于指数函数的阶.

- 算法的时间复杂度是各步操作时间之和，在常数步的情况下取最高阶的函数即可.

几类重要的函数

算法研究内容

3 计算复杂性理论：
货郎问题
背包问题
双机调度问题

2 问题复杂度概念：
排序问题

1 算法设计与分析：
调度问题
投资问题

算法的有关概念

5 算法的伪码表示

8 几类重要函数的性质

7 有关函数渐近的界的定理

6 时间复杂度函数的表示：
函数渐近的界

4 算法及其时间复杂度的定义

基本函数类

阶的高低

至少指数级： $2^n, 3^n, n!, \ldots$

多项式级： $n,\ n^2,\ n\log n,\ n^{1/2}, \ldots$

对数多项式级： $\log n,\ \log^2 n,\ \log\log n, \ldots$

对数函数

符号(简略写):

$\log n = \log_2 n$

$\log^k n = (\log n)^k$

$\log\log n = \log(\log n)$

前一个模块由定理1得到的一个结果

性质:

(1) $\log_2 n = \Theta(\log_l n)$ (l>0，且 $l \neq 1$)

(2) $\log_b n = o(n^\alpha)$ $\alpha > 0$ $\boxed{\ln n = o\,(n^d),\ d > 0}$

(3) $a^{\log_b n} = n^{\log_b a}$

有关性质(1)的证明

$$\boxed{\log_k n = \frac{\log_l n}{\log_l k} \quad \log_l k\text{为常数}}$$

对数函数性质：

(1) $\log_2 n = \Theta(\log_l n)$

(2) $\log_b n = o(n^\alpha) \quad \alpha > 0$

(3) $a^{\log_b n} = n^{\log_b a}$

$$\lim_{n\to\infty} \frac{\log_k n}{\log_l n} = \lim_{n\to\infty} \frac{\log_l n}{\log_l k \cdot \log_l n} = \frac{1}{\log_l k}$$

根据定理1， (1) $\log_k n = \Theta(\log_l n)$

■

有关性质(2)(3)的说明

对数函数性质：

(1) $\log_2 n = \Theta(\log_l n)$

(2) $\log_b n = o(n^\alpha)\quad \alpha > 0$

(3) $a^{\log_b n} = n^{\log_b a}$

(2) $\log_b n = \Theta(\ln n)$

$\ln n = o(n^\alpha) \quad \Rightarrow \quad \log_b n = o(n^\alpha)\quad \alpha > 0$

前一个模块由定理1
得到的一个结果

(3) $a^{\log_b n} = n^{\log_b a}$

$\Updownarrow$

$\log_b n \log_b a = \log_b a \log_b n$

指数函数与阶乘

Stirling公式 $$n! = \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n\log n)$$

应用：估计搜索空间大小

$$C_{m+n-1}^{m} = \frac{(m+n-1)!}{m!(n-1)!}$$

m元钱、投资n个项目的分配方案数

$$= \frac{\sqrt{2\pi(m+n-1)}(m+n-1)^{m+n-1}(1+\Theta(\frac{1}{m+n-1}))}{\sqrt{2\pi m}m^{m}(1+\Theta(\frac{1}{m}))\sqrt{2\pi(n-1)}(n-1)^{n-1}(1+\Theta(\frac{1}{n-1}))}$$

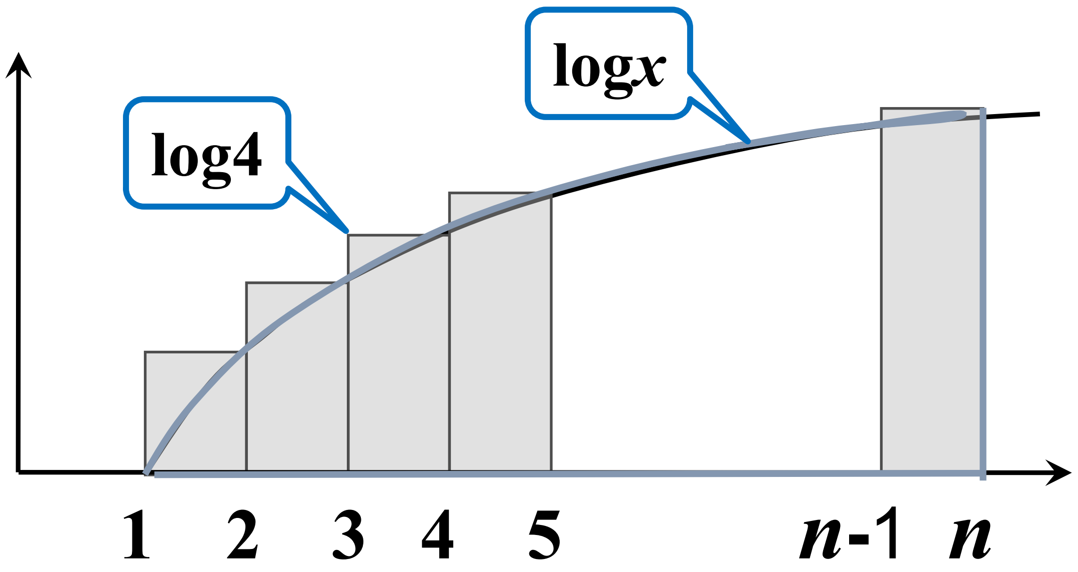
$$\boxed{= \Theta((1+\varepsilon)^{m+n-1})}$$

$\log(n!) = \Omega(n\log n)$ 的证明

$\boxed{\log(n!) = \Theta(n\log n)}$

$$\log(n!) = \sum_{k=1}^{n} \log k \geq \int_1^n \log x dx$$

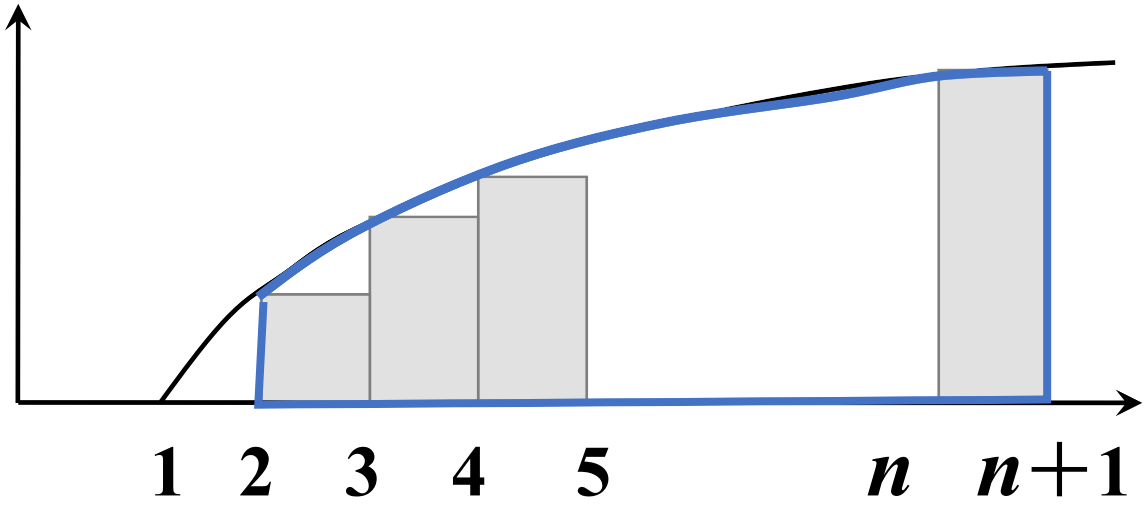
$$= \log e(n\ln n - n + 1) \quad \Rightarrow \quad \log(n!) = \Omega(n\log n) \quad (1)$$



$\log(n!) = O(n\log n)$的证明

$\log(n!) = \Theta(n\log n)$

$$\log(n!) = \sum_{k=1}^{n} \log k \le \int_{2}^{n+1} \log x dx = O(n \log n) \qquad (2)$$



由（1）式和（2）式 $\Rightarrow$ $\log(n!) = \Theta(n\log n)$ ■

取整函数 Floor and ceiling functions

取整函数的定义

$\lfloor x \rfloor$： 表示小于等于 x 的最大的整数

$\lceil x \rceil$： 表示大于等于 x 的最小的整数

实例

$\lfloor 2.6 \rfloor = 2$

$\lceil 2.6 \rceil = 3$

$\lfloor 2 \rfloor = \lceil 2 \rceil = 2$

应用：二分检索，输入数组长度：n

中位数的位置：$\lfloor (1+n)/2 \rfloor$

与中位数比较后子问题大小：$\lfloor n/2 \rfloor$

取整函数的性质

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

(2) $\lfloor x+n\rfloor=\lfloor x\rfloor+n$, $\lceil x+n\rceil=\lceil x\rceil+n$, n为整数

(3) $$\left\lceil\frac{n}{2}\right\rceil+\left\lfloor\frac{n}{2}\right\rfloor=n$$

(4) $$\left\lceil\frac{\left\lceil\frac{n}{a}\right\rceil}{b}\right\rceil=\left\lceil\frac{n}{ab}\right\rceil,\quad \left\lfloor\frac{\left\lfloor\frac{n}{a}\right\rfloor}{b}\right\rfloor=\left\lfloor\frac{n}{ab}\right\rfloor$$

证明(1)

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

(1) 如果x是整数n，根据定义$\lfloor x\rfloor=\lceil x\rceil=n$，

$$x-1<\lfloor x\rfloor=x=\lceil x\rceil<x+1$$

如果 $n<x<n+1$，n为整数，那么

$$\lfloor x\rfloor=n,\ \ \lceil x\rceil=n+1,$$

从而有

$$x-1<n=\lfloor x\rfloor,\ \ n<x<n+1=\lceil x\rceil$$

$$\Rightarrow x-1<n=\lfloor x\rfloor<x<\lceil x\rceil=n+1<x+1$$

■

例：按照阶排序

$$\log(n!),\quad \log^2 n,\quad 2,\quad n!,\quad n2^n,\quad n^{1/\log n},$$

$$(3/2)^n,\quad \sqrt{\log n},\quad (\log n)^{\log n},\quad 2^{2^n},$$

$$n^{\log\log n},\quad n^3,\quad \log\log n,\quad n\log n,\quad n,$$

$$2^{\log n},\quad \log n$$

例：按照阶排序

$\log(n!)$, $\log^2 n$, 2, $n!$, $n2^n$, $n^{1/\log n}$,
$(3/2)^n$, $\sqrt{\log n}$, $(\log n)^{\log n}$, 2^{2^n},
$n^{\log\log n}$, n^3, $\log\log n$, $n\log n$, n,
$2^{\log n}$, $\log n$

指数函数 2^{2^n}, $n!$, $n2^n$, $(3/2)^n$, $\log n^{\log n} = n^{\log\log n}$,

多项式 n^3, $\log(n!) = \Theta(n\log n)$, $n = 2^{\log n}$,

对数函数 $\log^2 n$, $\log n$, $\sqrt{\log n}$, $\log\log n$,

常数 $n^{1/\log n} = O(1)$

小结

- 几类常用函数的阶的性质

  对数函数

  指数函数

  阶乘函数

  取整函数

- 如何利用上述性质估计函数的阶？

$$C_{m+n-1}^{m}=\frac{(m+n-1)!}{m!\,(n-1)!}$$

$$=\frac{\sqrt{2\pi(m+n-1)}(m+n-1)^{m+n-1}(1+\Theta(\frac{1}{m+n-1}))}{\sqrt{2\pi m}m^{m}(1+\Theta(\frac{1}{m}))\sqrt{2\pi(n-1)}(n-1)^{n-1}(1+\Theta(\frac{1}{n-1}))}$$

丢掉放大

$$<\frac{(m+n-1)^{m+n-1}}{m^{m}(n-1)^{n-1}}=\frac{(m+n-1)^{m}(m+n-1)^{n-1}}{m^{m}(n-1)^{n-1}}$$

$$=\left(1+\frac{n-1}{m}\right)^{m}\left(\frac{m}{n-1}+1\right)^{n-1}<(1+\varepsilon)^{m}(\varepsilon+1)^{n-1}=(1+\varepsilon)^{m+n-1}\quad \varepsilon=\max\{\frac{n-1}{m},\frac{m}{n-1}\}$$

$$C_{m+n-1}^{m}=\frac{(m+n-1)!}{m!\,(n-1)!}$$

分子乘一个
根号项放大

$$=\frac{\sqrt{2\pi(m+n-1)}(m+n-1)^{m+n-1}(1+\Theta(\frac{1}{m+n-1}))}{\sqrt{2\pi m}m^{m}(1+\Theta(\frac{1}{m}))\sqrt{2\pi(n-1)}(n-1)^{n-1}(1+\Theta(\frac{1}{n-1}))}$$

$$>\frac{(m+n-1)^{m+n-1}}{m^{m}(n-1)^{n-1}}=\frac{(m+n-1)^{m}(m+n-1)^{n-1}}{m^{m}(n-1)^{n-1}}$$

$$=\left(1+\frac{n-1}{m}\right)^{m}\left(\frac{m}{n-1}+1\right)^{n-1}>(1+\varepsilon)^{m}(\varepsilon+1)^{n-1}=(1+\varepsilon)^{m+n-1}\quad \varepsilon=\min\{\frac{n-1}{m},\frac{m}{n-1}\}$$