蘇州大學
SOOCHOW UNIVERSITY

Chap3 Algorithms

Jin-Hui Wu

2026-03-20

大纲

- 渐进符号 (3.2)

- 具体算法
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

函数增长速度

- 将包含n个数字的序列从小到大排序
  - 算法1：需要$n^2 + n$次操作
  - 算法2：需要$100\ n \log n$次操作

- 随着问题规模n的增长
  - 算法1的操作次数以n^2速度增长
  - 算法2的操作次数以$n \log n$速度增长

  - 当n足够大时，算法2更高效

渐进符号

- 渐进符号 (asymptotic notation)
  - 只关心函数的主要增长部分
  - 如$2n^2 + 3n$的主要增长部分是n^2

  - $O(\cdot)$
  - $\Omega(\cdot)$
  - $\Theta(\cdot)$
  - $o(\cdot)$
  - $\omega(\cdot)$

$O(\cdot)$

- 定义
  - f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若存在常数$C$和$k$，使得

    $$|f(x)| \le C|g(x)| \qquad \forall x \ge k$$

    则称$f(x)$是$O(g(x))$的

  - 当x足够大时，f增长速度不超过g

  - 记作$f(x) = O(g(x))$
    - 更准确的写法是$f(x) \in O(g(x))$，但普遍使用等号

例

- 证明$f(x) = x^2 + 2x$满足$f(x) = O(x^2)$

- f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若存在常数$C$和$k$，使得

$$|f(x)| \leq C|g(x)| \qquad \forall x \geq k$$

则称$f(x)$是$O\big(g(x)\big)$的

例

- 证明$f(x) = 7x^2$满足$f(x) = O(x^3)$

- f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若存在常数$C$和$k$，使得

$$|f(x)| \leq C|g(x)| \qquad \forall x \geq k$$

则称$f(x)$是$O\big(g(x)\big)$的

例

- 证明$f(x) = x^2$不满足$f(x) = O(x)$

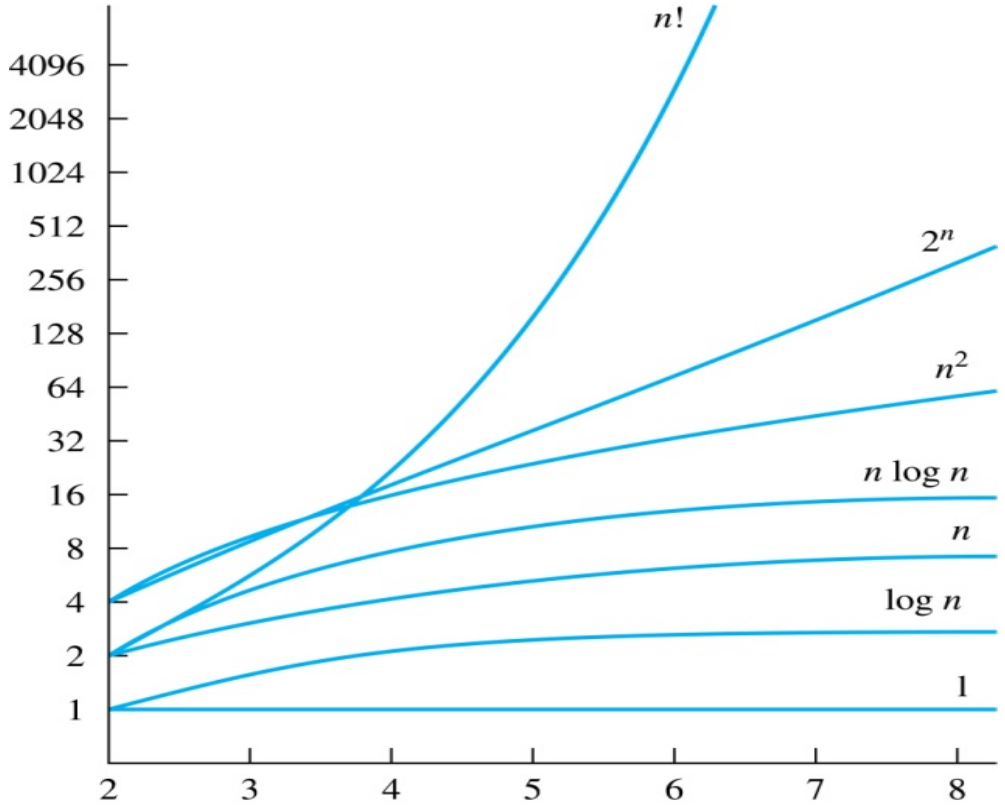
- f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若存在常数$C$和$k$，使得

$$|f(x)| \leq C|g(x)| \qquad \forall x \geq k$$

则称$f(x)$是$O\big(g(x)\big)$的

一些常见的渐进关系

- $1 = O(\log n)$
- $\log n = O(n^a)$
  - $a > 0$
- $n^a = O(2^n)$
- $2^n = O(n!)$



例

- 将下列函数排序，使得上一个是下一个的O
  - $f_1(n) = (1.5)^n$
  - $f_2(n) = 8n^3+17n^2 +111$
  - $f_3(n) = (\log \mathrm{n})^2$
  - $f_4(n) = 2^n$
  - $f_5(n) = \log(\log n)$
  - $f_6(n) = n^2(\log n)^3$
  - $f_7(n) = 2^n(n^2 +1)$
  - $f_8(n) = n^3+ n(\log n)^2$
  - $f_9(n) = 10000$
  - $f_{10}(n) = n!$

$\Omega(\cdot)$

- 定义
  - f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若存在常数$C$和$k$，使得

    $$|f(x)| \geq C|g(x)| \qquad \forall x \geq k$$

    则称$f(x)$是$\Omega\big(g(x)\big)$的

  - 记作$f(x) = \Omega\big(g(x)\big)$

  - 与$g(x) = O\big(f(x)\big)$等价

$\Theta(\cdot)$

- 定义
  - f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若
    - $f(x) = O(g(x))$
    - $f(x) = \Omega(g(x))$
  - 则称$f(x)$是$\Theta(g(x))$的
  - 记作$f(x) = \Theta(g(x))$
  - 也称$f(x)$和$g(x)$是同阶的 (f and g are of the same order)

例

- 证明$3x^2 + 8x \log x = \Theta(x^2)$

例

- 证明前n个正整数的立方和是$\Theta(n^4)$

拓展：$o(\cdot)$

- 定义
  - f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0,$$

    则称$f(x)$是$o\big(g(x)\big)$的

  - 记作$f(x) = o\big(g(x)\big)$

  - 表明f增长得比g慢

拓展：$\omega(\cdot)$

- 定义
  - f和g为$\mathbb{Z}^+ \to \mathbb{R}$或$\mathbb{R} \to \mathbb{R}$的函数，若

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \infty,$$

    则称$f(x)$是$\omega\big(g(x)\big)$的

  - 记作$f(x) = \omega\big(g(x)\big)$

  - 表明f增长得比g快

大纲

- 渐进符号

- 具体算法 (3.1, 3.3)
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

序列最大元

- 问题 (problem)
  - 一个算法解决一个问题
  - 问题需要描述输入 (input)、预期的输出 (output)

- 序列最大元
  - 输入：一串有限长度的整数
  - 输出：这串整数中的最大值

序列最大元

- 算法 (algorithm)
  - 算法是一组有限的精确的指令
  - 有限：计算机可以在有限时间内完成
    - “计算所有正整数的倒数的平方和”
  - 精确：计算机可执行
    - “找出这本相册中最漂亮的猫”

序列最大元

- 算法可以用文字描述
  - 临时最大值 = 第一个整数
  - 将下一个整数与临时最大值进行比较
    - 若下一个数更大，则临时最大值等于该整数
  - 重复上一步直至没有更多整数
  - 临时最大值即序列中的最大整数

序列最大元

- 算法可以用伪代码 (pseudocode) 描述

算法的输入

```
procedure max(a_1, a_2, ...., a_n: integers)
    max := a_1
    for i := 2 to n
        if max < a_i then max := a_i
    return max{max is the largest element}
```

算法的输出

- 伪代码比自然语言和代码更简洁
- 具有自然语言的可读性和代码的结构

序列最大元

- 算法复杂度
  - 时间复杂度 (time complexity)
    - 算法解决规模为n的问题所需的运行时间
  - 空间复杂度 (space complexity)
    - 算法解决规模为n的问题所需的计算机内存
  - 通常更关心时间复杂度，因为空间复杂度往往不大于时间复杂度

序列最大元

1次赋值

n-1次赋值 (加法)
n-1次比较 (i<=n)

```
procedure max(a_1, a_2, ...., a_n: integers)
    max := a_1
    for i := 2 to n
        if max < a_i then max := a_i
    return max{max is the largest element}
```

n-1次比较
至多n-1次赋值

- $2(n-1)$次比较
- n到$2n-1$次赋值
- 时间复杂度：$\Theta(n)$

大纲

- 渐进符号

- 具体算法 (3.1, 3.3)
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

序列搜索

- 问题：搜索 (searching)
  - 判断x是否在一个有限序列中
  - 若在，则返回其位置；否则，返回不在

  - 应用：判断想借阅的书是否在图书馆书库中

线性搜索

- 线性搜索 (linear searching)
  - 想法：从头搜到尾，逐一判断是否为x

procedure *linear search*(x:integer, a_1, a_2, ...,a_n: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* := i
else *location* := 0
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

线性搜索

- 最坏情况复杂度 (worst-case complexity)
  - 当x不在序列中时，需要逐一对比所有元素
  - 时间复杂度：$\Theta(n)$

procedure *linear search*(x:integer, a_1, a_2, ...,a_n: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* := i
else *location* := 0
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

二分搜索

- 二分搜索 (binary searching)
  - 要求：序列单调增 (或单调减)
  - 想法：对比剩余序列中点与x的大小

二分搜索

Example: The steps taken by a binary search for 19 in the list:

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

1. The list has 16 elements, so the midpoint is 8. The value in the 8^{th} position is 10. Since $19 > 10$, further search is restricted to positions 9 through 16.

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

2. The midpoint of the list (positions 9 through 16) is now the 12^{th} position with a value of 16. Since $19 > 16$, further search is restricted to the 13^{th} position and above.

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

3. The midpoint of the current list is now the 14^{th} position with a value of 19. Since $19 \not> 19$, further search is restricted to the portion from the 13^{th} through the 14^{th} positions .

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

4. The midpoint of the current list is now the 13^{th} position with a value of 18. Since $19 > 18$, search is restricted to the portion from the 14^{th} position through the 14^{th}.

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

5. Now the list has a single element and the loop ends. Since $19 = 19$, the location 14 is returned.

二分搜索

- 二分搜索 (binary searching)
  - 要求：序列单调增 (或单调减)
  - 想法：对比剩余序列中点与x的大小

```
procedure binary search(x: integer, a_1,a_2,..., a_n: increasing integers)
 i := 1 {i is the left endpoint of interval}
 j := n {j is right endpoint of interval}
 while i < j
       m := ⌊(i + j)/2⌋
       if x > a_m then i := m + 1
       else j := m
 if x = a_i then location := i
 else location := 0
 return location{location is the subscript i of the term a_i equal to x,
                                or 0 if x is not found}
```

二分搜索

- 最坏情况复杂度
  - 区间长度每次减半，只能减半$\log n$次
  - 时间复杂度：$\Theta(\log n)$

```
procedure binary search(x: integer, a_1,a_2,..., a_n: increasing integers)
 i := 1 {i is the left endpoint of interval}
 j := n {j is right endpoint of interval}
 while i < j
       m := ⌊(i + j)/2⌋
       if x > a_m then i := m + 1
       else j := m
  if x = a_i then location := i
  else location := 0
  return location{location is the subscript i of the term a_i equal to x,
                                  or 0 if x is not found}
```

大纲

- 渐进符号

- 具体算法 (3.1, 3.3)
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

序列排序

- 问题：排序 (sorting)
  - 将序列中的元素按某种顺序排序
    - 将整数从小到大排列
    - 将字符串按字母序升序排列
  - 应用：点名表按学号升序排序

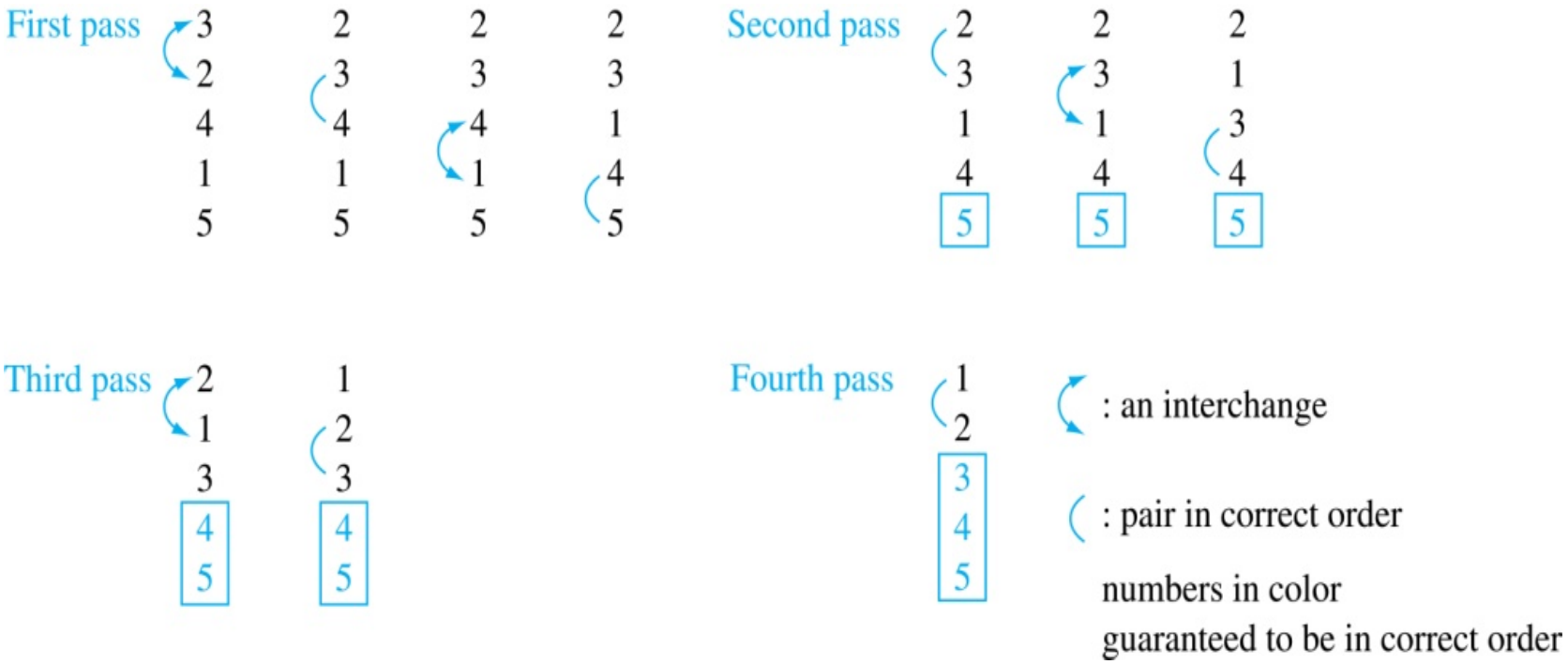
冒泡排序

- 冒泡排序 (bubble sort)
  - 想法：多次遍历列表，将无序的相邻元素互换

procedure *bubblesort*(a_1,...,a_n: real numbers with $n \geq 2$)
  for $i := 1$ to $n-1$
    for $j := 1$ to $n-i$
      if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
{a_1,..., a_n is now in increasing order}

冒泡排序

- 对下列数组冒泡排序：3 2 4 1 5



冒泡排序

- 最坏情况复杂度
  - 对下标i和j的两层循环：

$$(n-1)+(n-2)+\cdots+1$$

  - 时间复杂度：$\Theta(n^2)$

procedure *bubblesort*(a_1,...,a_n: real numbers with $n \geq 2$)

for i := 1 to $n-1$

for j := 1 to $n-i$

if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{a_1,..., a_n is now in increasing order}

插入排序

- 插入排序 (insertion sort)
  - 想法：第j轮中，将第j个元素插入到按序排列的前$j-1$个元素中

插入排序

Example: Show all the steps of insertion sort with the input: 3 2 4 1 5

i. 2 3 4 1 5

ii. 2 3 4 1 5

iii. 1 2 3 4 5

iv. 1 2 3 4 5

插入排序

- 插入排序 (insertion sort)
  - 想法：第j轮中，将第j个元素插入到按序排列的前$j-1$个元素中

```
procedure insertion sort(a_1,...,a_n: real numbers with n ≥ 2)
    for j := 2 to n
        i := 1
        while a_j > a_i
            i := i + 1
        m := a_j
        for k := 0 to j − i − 1
            a_{j-k} := a_{j-k-1}
        a_i := m
{Now a_1,...,a_n is in increasing order}
```

插入排序

- 最坏情况复杂度
  - 两层循环，外层j，内层i和k：

$$2 + 3 + \cdots + n$$

  - 时间复杂度：$\Theta(n^2)$

procedure *insertion sort*(a_1,...,a_n: real numbers with $n \geq 2$)
  for $j := 2$ to n
    $i := 1$
    while $a_j > a_i$
      $i := i + 1$
    $m := a_j$
    for $k := 0$ to $j - i - 1$
      $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{Now a_1,...,a_n is in increasing order}

大纲

- 渐进符号

- 具体算法 (3.1, 3.3)
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

矩阵乘法

- 目标
  - 计算$A \in \mathbb{R}^{m \times k}$和$B \in \mathbb{R}^{k \times n}$的乘积

```
procedure matrix multiplication(A,B: matrices)
    for i := 1 to m
        for j := 1 to n
            c_ij := 0
            for q := 1 to k
                c_ij := c_ij + a_iq b_qj
return C{C = [c_ij] is the product of A and B}
```

矩阵乘法

- 时间复杂度
  - 下标i, j, k的三层循环
  - $\Theta(mnk)$
  - 方阵： $\Theta(n^3)$

procedure *matrix multiplication*(**A**,**B**: matrices)
  for i := 1 to m
    for j := 1 to n
      c_{ij} := 0
      for q := 1 to k
        c_{ij} := c_{ij} + a_{iq} b_{qj}
return C{**C** = [c_{ij}] is the product of **A** and **B**}

矩阵乘法

- 矩阵链乘法 (matrix-chain multiplication)
  - n个矩阵相乘：$A_1 A_2 \cdots A_n$
  - 其中，$A_i \in \mathbb{R}^{m_i \times m_{i+1}}$
  - 利用矩阵乘法结合律，可以降低复杂度
  - 最优结合方式需要用到动态规划算法 (chap 8)

大纲

- 渐进符号

- 具体算法 (3.1, 3.3)
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

优化问题

- 优化 (optimization)
  - 寻找能使得目标最大/小化的一组变量取值
  - 例
    - 在一房间内摆放一些家具，在家具可正常使用的前提下，最大化可活动空间
    - 用尽可能少的纸币张数完成特定金额的找零
    - 从天赐庄校区到独墅湖校区的最快公交方案

优化问题

- 优化 (optimization)
  - 寻找能使得目标最大/小化的一组变量取值
- 贪心算法 (greedy algorithm)
  - 在每一步做最好的选择
    - 找零：先找额度内最大的纸币
    - 家具：先放最大的家具
    - 公交：先坐最快的公交 (地铁)
  - 贪心算法未必能找到最优解

贪心算法

- 找零
  - 优先选择尽可能大的面额

procedure *change*(c_1, c_2, …, c_r: values of coins, where $c_1 > c_2 > \ldots > c_r$; n: a positive integer)
for $i := 1$ to r
  $d_i := 0$ [d_i counts the coins of denomination c_i]
  while $n \geq c_i$
    $d_i := d_i + 1$ [add a coin of denomination c_i]
    $n = n - c_i$
[d_i counts the coins c_i]

例

- 考虑如下找零问题
  - 面额：25, 10, 1
  - 找零金额：31

大纲

- 渐进符号

- 具体算法 (3.1, 3.3)
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

暴力算法

- 暴力算法 (brute-force algorithm)
  - 穷举所有情况、不做任何加速的一种算法范式 (algorithmic paradigm)
  - 例
    - 线性搜索
    - 冒泡排序

素数判断

- 素数判断
  - 输入一个正整数n，判断n是否是素数
  - 暴力算法：判断2到n-1是否是n的因数
    - 时间复杂度：$O(n)$
  - 加速：判断2到$\sqrt{n}$是否为n的因数
    - 时间复杂度：$O(\sqrt{n})$

算法复杂度

- 一些常见的算法复杂度

TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

算法复杂度

- 笔记本1s约可进行10^{11}次运算

TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*

大纲

- 渐进符号

- 具体算法 (3.1, 3.3)
  - 序列最大元
  - 序列搜索
  - 序列排序
  - 矩阵乘法
  - 贪心算法
  - 暴力算法
  - 停机问题、问题复杂度

停机问题

- 停机问题 (halting problem)
  - 能否设计一个算法，使得
    - 输入：一段代码、这段代码的输入
    - 输出：这段代码在这个输出下是否能停止
  - 停机问题是不可判定的 (undecidable)
    - 课本上称之为 unsolvable

问题复杂度

- 问题复杂度 (complexity of problems)
  - P (polynomial)
    - 存在多项式时间算法，也称易解的 (tractable)，如排序

问题复杂度

- 问题复杂度 (complexity of problems)
  - P (polynomial)
    - 存在多项式时间算法，也称易解的 (tractable)，如排序
  - NP (nondeterministic P)
    - 解可在多项式时间内判断

问题复杂度

- 问题复杂度 (complexity of problems)
  - P (polynomial)
    - 存在多项式时间算法，也称易解的 (tractable)，如排序
  - NP (nondeterministic P)
    - 解可在多项式时间内判断
  - NPC (NP complete)
    - 若存在多项式时间算法，该算法可用于解决所有类NP问题
    - 图着色：能否用3中颜色给地图染色，使得相邻区域不同色

问题复杂度

- 问题复杂度 (complexity of problems)
  - P (polynomial)
    - 存在多项式时间算法，也称易解的 (tractable)，如排序
  - NP (nondeterministic P)
    - 解可在多项式时间内判断
  - NPC (NP complete)
    - 若存在多项式时间算法，该算法可用于解决所有类NP问题
  - 不可判定 (undecidable)
    - 不存在判定算法，如停机问题

P/NP问题

- P/NP问题
  - P和NP是否相等？
  - 通常认为P≠NP
    - 是现代公钥密码体系的基础
  - P/NP问题仍未解决
    - Clay数学研究所提供一百万美元奖金

总结

- 渐进符号
  - O, Ω, Θ
  - o, ω
- 算法及其时间复杂度分析
  - 最大元、搜索、排序、矩阵乘法
- 算法范式
  - 贪心算法、暴力算法