Algorithms Design and Analysis

Lecture 1

Lecturer: 许可 (Xu Ke)

Beihang University

2017

Reference books (参考书)

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
2. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1991.
3. Jon Kleinberg, Eva Tardos. *Algorithm Design*. Pearson/ Addison-Wesley, 2006.
4. M.H. Alsuwaiyel. 算法设计技巧与分析. 电子工业出版社, 2006.
5. 屈婉玲等人. 算法设计与分析. 清华大学出版社, 2015.

作业提交和讲课笔记

作业应在两周之内完成并通过 http://judge.buaa.edu.cn 提交。

讲课笔记也可通过此网站下载。

有问题请发邮件至 algbuaa@gmail.com。

Basics (基本要素)

Goal (目标)

– Find the best algorithm for a given computational (计算) problem
– If infeasible (不可实行的), good ones

Three things you will learn

– Design a good algorithm
– Analyze (分析) it
– Lower bounds (下界): Know if it is the best algorithm

Question:

Write a function to compute 1-2+3-4+5-6+7+.....+*n*, where *n* is very large.

1.

```
long fn(long n)
{ long temp=0;
   int i,flag=1;
   if(n<=0)
  { printf("error: n must > 0);
     exit(1); }
   for(i=1;i<=n;i++)
  { temp=temp+flag*i;
     flag=(-1)*flag; }
   return temp;
}
```

2.

```
long fn(long n)
{long temp=0;
 int j=1, i=1;
  if(n<=0)
 {printf("error: n must > 0);
    exit(1); }
  while(j<=n)
 { temp=temp+i;
    i=-i;
    i>0?i++:i--;
    j++; }
  return temp; }
```

3.

```
long fn(long n)
{
  if(n<=0)
 {
    printf("error: n must > 0);
    exit(1);
 }
  if(0==n%2)
  return (n/2)*(-1);
  else
  return (n-1)/2*(-1)+n;
}
```

1Byte = 8 Bit

1 KB = 1,024 Bytes

1 MB = 1,024 KB = 1,048,576 Bytes

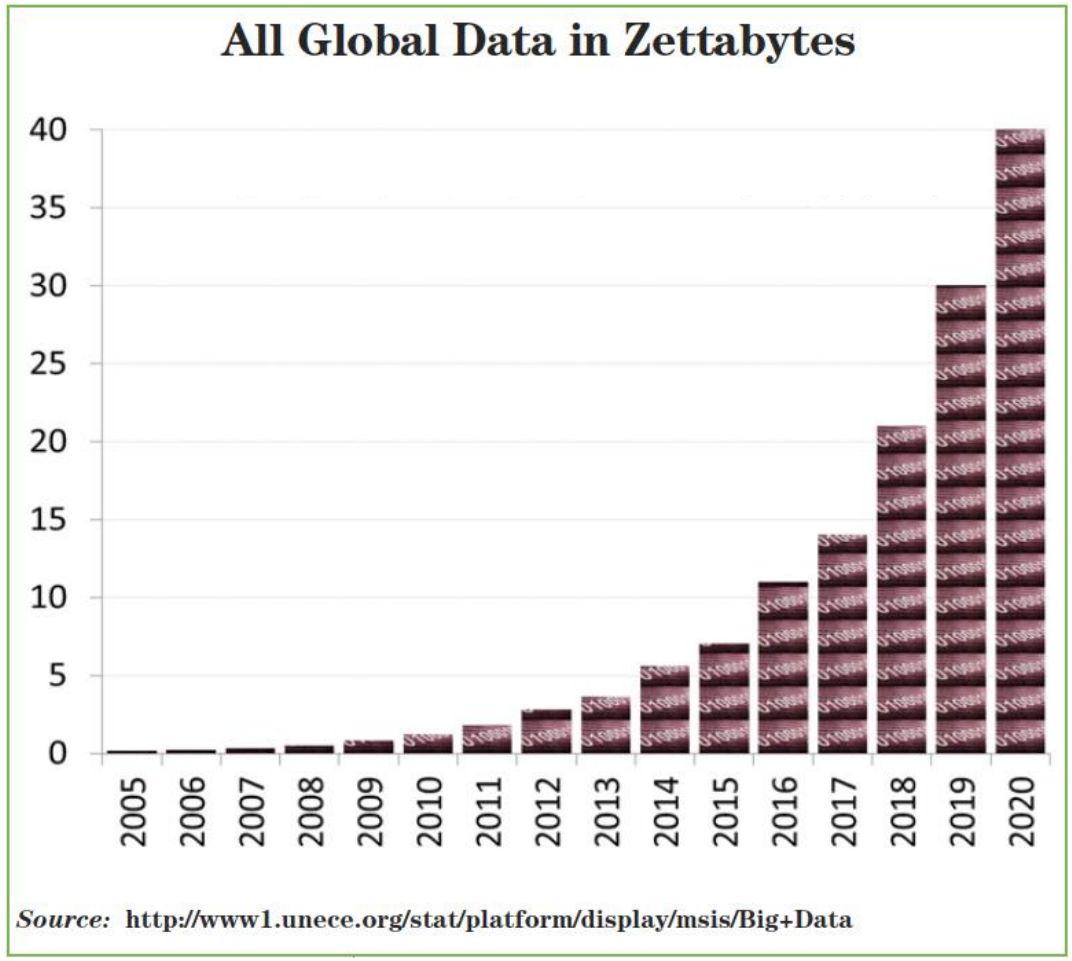
1 GB = 1,024 MB = 1,073,741,824 Bytes

1 TB = 1,024 GB = 1,099,511,627,776 Bytes

1 PB = 1,024 TB = 1,125,899,906,842,624 Bytes

1 EB = 1,024 PB = 1,152,921,504,606,846,976 Bytes

1 ZB = 1,024 EB = 1,180,591,620,717,411,303,424 Bytes



The Importance of Algorithms

- Algorithms are the soul of software.
- Fast computers can be inefficient when algorithms are inefficient.
- Slow computers can be efficient when algorithms are efficient.

图灵奖获得者（算法和计算理论领域）

Richard Hamming (1968) -- 汉明码

James Hardy Wilkinson (1970) -- 数值分析

Donald E. Knuth (1974) – The Art of Computer Programming

Michael Oser Rabin Dana Stewart Scott (1976) -- 自动机

Stephen Arthur Cook (1982) -- NP完全性

Richard Manning Karp (1985) – 算法理论，尤其是NP完全性理论

John E. Hopcroft，Robert Endre. Tarjan (1986) -- 数据结构和算法设计

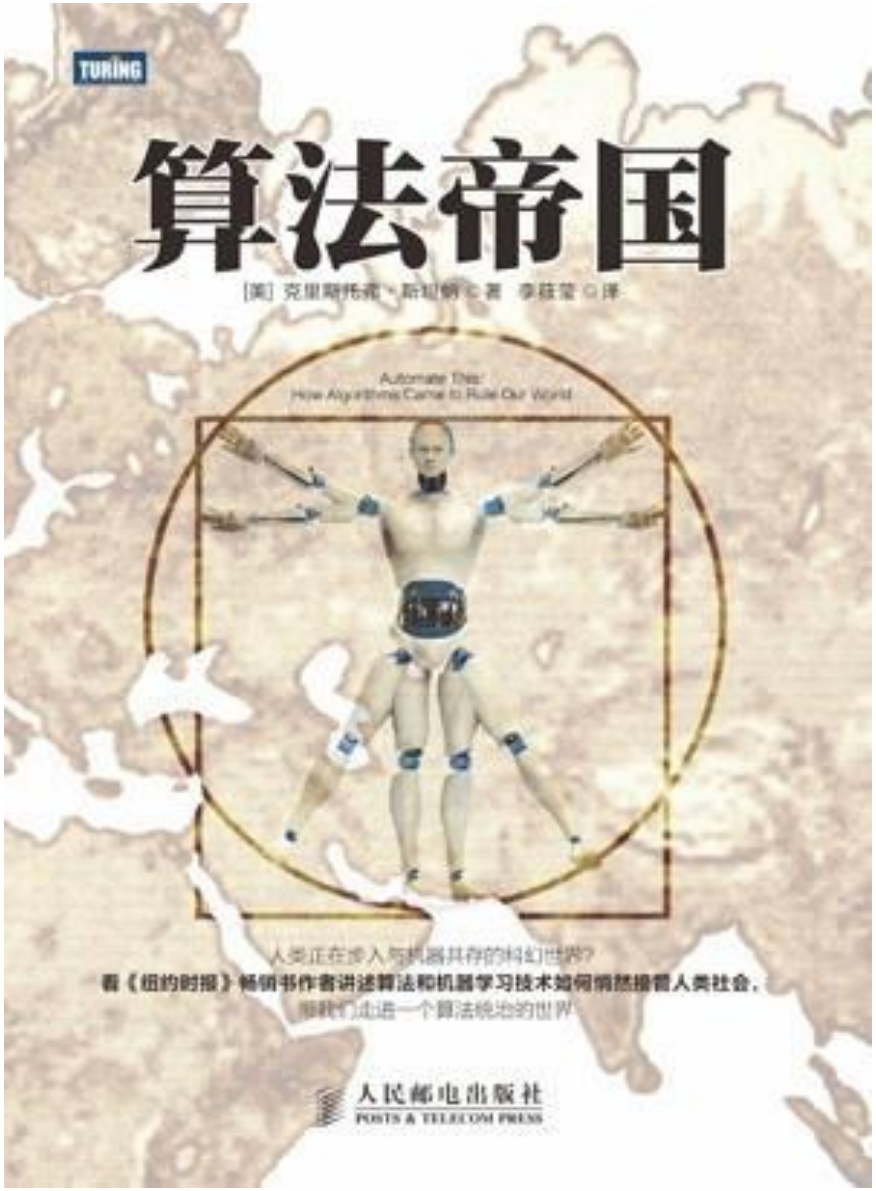
William (Velvel) Morton Kahan (1989) -- 浮点运算

Juris Hartmanis，Richard Edwin Stearns (1993) -- 计算复杂性

Manuel Blum (1995) -- 计算复杂性，密码系统和程序检查验证

姚期智(Andrew Chi-Chih Yao) (2000) -- 伪随机数生成，密码系统和通讯复杂性

Ronald L. Rivest, Adi Shamir, Leonard M. Adleman(2002) -- 公钥密码技术–RSA


TURING
算法帝国
[美] 克里斯托弗·斯坦纳 著 李筱莹 译
Automate This:
How Algorithms Came to Rule Our World
人类正在步入与机器共存的科幻世界?
看《纽约时报》畅销书作者讲述算法和机器学习技术如何悄然接管人类社会,
带我们走进一个算法统治的世界
人民邮电出版社
POSTS & TELECOM PRESS

Algorithms

– An algorithm is a precise, step-by-step process (过程) for solving a problem in a finite (有限的) number of steps.

– Examples:
  - Adding two numbers
  - Cooking
  - Mathematical Mechanization

吴文俊

– But in reality (现实) not all problems are solved using algorithms.

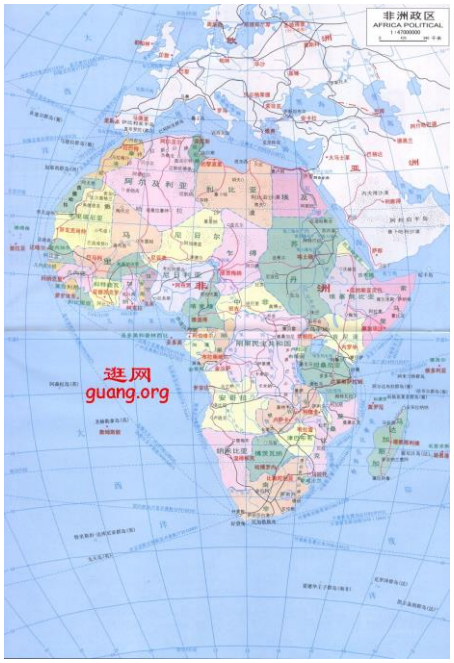
Cooking and Algorithms

- 西餐：黑椒牛排
  - 主料：牛排300克
  - 辅料：洋葱（白皮）25克，胡萝卜25克，西芹30克，青豆20克，菜花30克
  - 调料：盐4克，胡椒粉8克，黄油50克

- 中餐：辣焖牛腩
  - 主料：牛腩1000克
  - 辅料：红辣椒、葱、姜、蒜各适量
  - 调料：老抽2大匙，南味腐乳汁2大匙，盐、冰糖、料酒适量

Example: Hunt Elephants

1. Go to Africa
2. Start at the Cape of Good Hope (好望角)
3. Traverse the continent in an orderly (有序的) manner.
4. During each traverse
    a. Catch each animal seen
    b. Compare each animal caught to a known elephant
    c. Stop when a match is detected.



Remark: Experienced computer programmers modify the above algorithm by placing a known elephant in Cairo to ensure that the algorithm will terminate.

Learning from Nature's Way of Optimization(优化)

- Genetic Algorithms (遗传算法)
- Simulated Annealing Algorithms (模拟退火算法)
- Ant Colony Algorithms (蚁群算法)
- ……

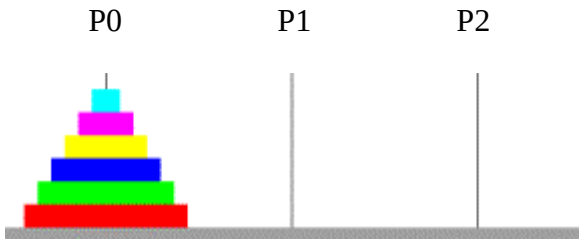
History

– “Algorithm” was derived (起源) from Mohammed AI-khowarizmi, 9th century Persian (波斯人), mathematician (数学家) credited with (归于) formalizing (形式化) pencil-and paper methods for addition (加法), subtraction (减法), multiplication (乘法) and division (除法).

Algorithm Control Structures

- Direct (直接) sequencing (先后顺序): ***x*=1; *y*=2;**
- Conditional (条件) branching (分支): **if (*x*=0) then…else…**
- Bounded (有限) Iteration (循环): **for (*i*=1; *i*<*n*; *i*++) …**
- Conditional Iteration: **while (*i* < *n*) …**
- Subroutines (子程序)
- Recursion (递归)

The Tower of Hanoi (Edouard Lucas, 1883)



Recursion version

Hanoi(*n*, P0, P1, P2)

IF *n*=1 THEN move (P0, 1st, P2)

ELSE

{Hanoi(*n*-1, P0, P2, P1)

move(P0, *n*th, P2)

Hanoi(*n*-1, P1, P0, P2)

}

RETURN

Complexity: Let T(*n*) denote the number of moves performed by Hanoi on *n* disks. Obviously, T(1)=1, T(*n*)=2T(*n*-1)+1. Not hard to prove that T(*n*)=2^n-1.

Practice

- Define (定义) the problem or refine (精化) an existing definition
- Design (设计) an algorithm
- Show its correctness (正确性)
- Analyze (分析) its performance (性能)
- Prove (证明) if it is the best
- Repeat (重复) the previous steps, if necessary

Techniques for Designing Algorithms

- Reduction to a known problem (归结到一个已知的问题)
- Divide and conquer (分而治之)
- Greedy approach (贪心策略)
- Dynamic programming (动态规划)
- Better data structures (更好的数据结构)
- Probabilistic solutions (概率解)
- Approximate solutions (近似解)

Complexity of an Algorithm

- The complexity of an algorithm is a device-independent (与设备无关) measure of the efficiency (效率) of the algorithm.

John E. Hopcroft

Cornell University

- The result is expressed as a function, giving the number of operations (操作) in terms of the size (大小、规模) of the input, e.g. $T(n)=2^n-1$ in the example of the Tower of Hanoi.

- Size: problem dependent (相关的)

– number of bits (multiplication), number of elements (sorting(排序)), number of vertices (顶点) and edges (边) (graph problems)

- Number of operations: model dependent

– RAM (Random Access Machine): LOAD, ADD, MULT, STORE; Each costs 1 unit of time.

– Turing Machine:

– Special (特殊的) models: Comparison (比较) tree for sorting (排序) and comparison.

Types of Complexity (复杂性的分类)

Let $t(x)$ be the time taken on input $x \in X$

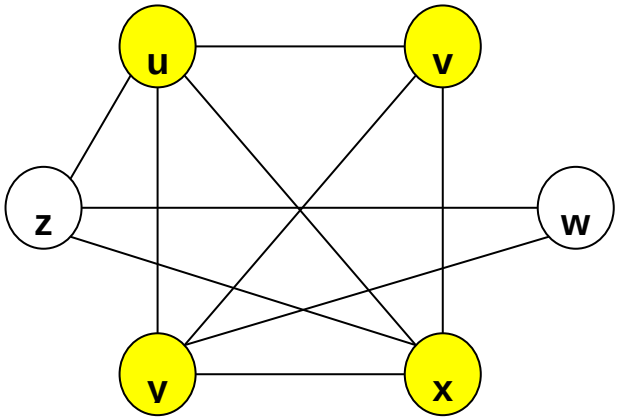
- Worst case (最坏情况): $W(n) = \max\{t(x), |x| = n\}$
  - Provides an upper bound (上界) on running time
  - An absolute (绝对的) guarantee (保证)
- Average case (平均情况): $A(n) = \sum_{|x|=n} t(x)\,\mathrm{Prob}(x)$
  - Provides the expected (期望的) running time
  - Very useful, but unrealistic (不实际的).

Experimental Evaluation (评估) of Algorithms

- Usually based on benchmarks (基准)
- Widely used in international algorithm competitions
  - SAT Competition
  - CSP Competition
  - ……

A Hard Problem: Maximum Clique (团) Problem

- Given a graph G, a clique is a subset S of vertices (顶点) in G such that each pair of vertices in S is connected by an edge.
- Maximum Clique Problem: Finding a clique with the largest number of vertices.



Finding a maximum clique is like finding a needle in a heystack

Copied from Lenka's report

But it might be easy in some cases

Copied from Lenka’s report

A Challenging Benchmark

– Feb.22, 2005: A large instance with 4000 nodes was generated which has a hidden maximum independent set of 100 nodes.

- Jul., 2005: Lengning Liu at the Univ. Kentucky found an IS of 96 nodes.
- Jul., 2007: Silvia Richter at Griffth Univ. found an IS of 97 nodes. (KI-07)
- Mar., 2010: Shaowei Cai at Peking Univ. found an IS of 98 nodes. (AAAI-10)
- Dec., 2014: C.D. Rosin at Amara Health Analytics. found an IS of 99 nodes.

For more details, please visit

http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm OR

http://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark

or Google the keyword: BHOSLIB

Asymptotic (渐近的) Performance (性能)

- In this course, we care most about asymptotic performance.
- How does the algorithm behave as the problem size gets very large?
  - Running time
  - Memory/storage requirements
  - Bandwidth

Example

Suppose that you have a small business and need a program to keep track of 1024 customers. Two programmers A and B both come up with algorithms. A proposes an algorithm that sorts n names using $256n\log n$ comparisons and B presents an algorithm that uses n^2 comparisons.

Your current computer takes 10^{-3} seconds to make one comparison, and so when you benchmark (衡量) the algorithms you might conclude that B's algorithm is better.

Size	A	B
1024	2621	1049

Expansion

As the business expands, A’s algorithm becomes more and more competitive than B's.

Size	A	B
1024	2621	1049
2048	5767	4194
4096	12583	16777
8192	27263	67109

We should think about a problem from a developing point of view!

Hardware improvement

If an application requires you to sort as many items as possible in one hour.

Thus if A's algorithm can sort n_A items , and B's n_B items, then

$$3600000 = 256n_A \log n_A = n^2_B$$

yields $n_A = 1352$ and $n_B = 1897$.

But suppose that you replace the computer with one that is four times as fast.

So you can make 14400000 comparisons in the same time. Then

$$14400000 = 256n_A \log n_A = n^2_B$$

yields $n_A = 4620$ and $n_B = 3794$.

A's algorithm gains much more from the faster computer than B's.

Why Efficient (有效率的) Algorithms ?

$f(n)$	1 second	1 day	1 century
n	10^6	8.6×10^{10}	3.1×10^{15}
n^2	1000	2.9×10^5	5.6×10^7
2^n	20	36	51

Table 1. Assume each operation takes 10^{-6} second.

$f(n)$	1 second	1 day	1 century
n	10^8	8.6×10^{12}	3.1×10^{17}
n^2	10^4	2.9×10^6	5.6×10^8
2^n	27	43	58

Table 2. Assume each operation takes 10^{-8} second.

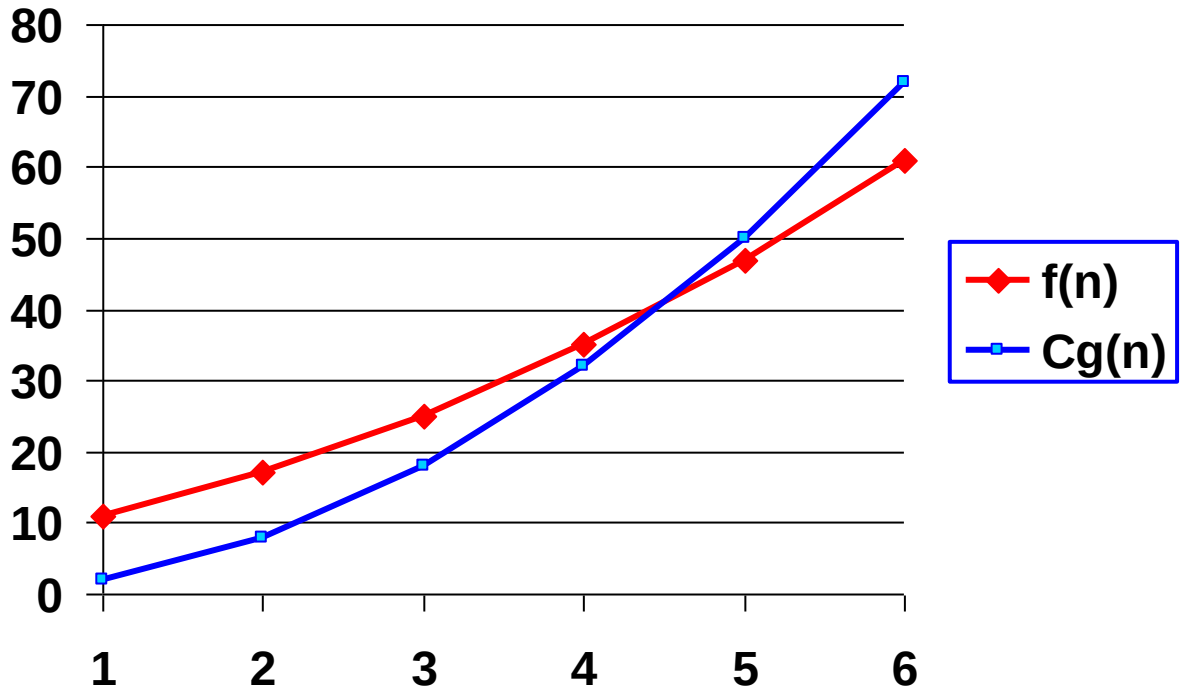
Asymptotic notation (记号) for (positive (正的)) functions:

- $f(n) = O(g(n))$ if
  - ✓ there exists constants c, n_0 such that $f(n) \le cg(n)$ for all $n \ge n_0$.

- For example, $n^2 + 3n + 7 = O(n^2)$.

$$f(n) = n^2 + 3n + 7 \quad g(n) = n^2$$

There exists $c = 2$, and $n_0 = 5$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.



Other Asymptotic notations (记号):

- $f(n) = \Omega(g(n))$ if there exists constants c, n_0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$, e.g. $0.5n^2 - 10 = \Omega(n)$.
- $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- $f(n) = o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$, e.g. $256n \ln n = o(n^2)$.
- Intuitively (直观地), $o()$ is like $<$; $O()$ is like $\leq$; $\Omega()$ is like $\geq$; $\Theta()$ is like $=$.
- We will usually just use $O()$ notation.

Exercise: For the following pairs of functions f, g write all equations that apply of the form $f=?(g)$, where $? \in \{ o, O, \Omega, \Theta\}$.

(a) $f(n)=4n^3-n^2\log n$, $g(n)=n^3+4n+5$

(b) $f(n)=2^n$, $g(n)= 2^{n+4}$

(c) $f(n)=2^n$, $g(n)= 2^{2n}$

(d) $f(n)=\begin{cases} n, \text{ if } n \text{ is odd (奇数)} \\ n^2, \text{ if } n \text{ is even (偶数)} \end{cases}$, $g(n)=\begin{cases} n^2, \text{ if } n \text{ is odd} \\ n, \text{ if } n \text{ is even} \end{cases}$

O-Manipulation (*O*-运算)

Remember: *O* is used to evaluate (评估) the asymptotic (渐近) order (量级、阶) of functions (函数). If f(*n*)=*O*(g(*n*)), then it means that the order of f(*n*) is less than or equal (≤) to g(*n*) when *n* tends to (趋于) infinity (无穷大).

For example, $\frac{n^2+\log n}{n-1}=O(n)$.

Basic Properties (基本性质) about *O*-Manipulation

- f(n)=O(f(n)), e.g. (例如) $n^2=O(n^2)$,

- If g(n)=o(f(n)), then O(f(n)) + O(g(n)) = O(f(n)),
  - e.g. (例如) $O(n^2) + O(n) = O(n^2)$,

Proof: $O(n^2) \le c_1n^2$ for $n \ge n_1$, and $O(n) \le c_2n$ for $n \ge n_2$. So, we have $O(n^2)+O(n) \le c_1n^2+c_2n \le c_1n^2+c_2n^2 \le (c_1+c_2)n^2$ for $n \ge \max\{n_1, n_2\}$.

- e.g. $O(n) + O(n\log n) = O(n\log n)$.

- If c is a constant (常数), then $c\ O(\mathrm{f}(n)) = O(\mathrm{f}(n))$,
  - e.g., $O(\log_{10} n) = O(\log_2 n) = O(\ln n)$.

- $\mathrm{f}(n)\ O(\mathrm{g}(n)) = O(\mathrm{f}(n)\ \mathrm{g}(n))$, e.g. $nO(\log n)=O(n\log n)$.

- $O(\mathrm{f}(n))\ O(\mathrm{g}(n)) = O(\mathrm{f}(n)\ \mathrm{g}(n))$, e.g. $O(n)O(\log n)= O(n\log n)$.

o-Hierarchy (*o*-等级结构)

- 1 – constant (常数)
- $\log\log n$- double logarithmic (对数)
- $\log n$- logarithmic ($\log_2 n$, $\ln n$, $\log_{10} n$)
- n^{θ}- $0 < \theta < 1$
- n^{k}- polynomial (integer $k \geq 1$)
- $n^{\log n}$
- 2^{n}- exponential (指数)
- $n!$- factorial (阶乘)=$\sqrt{2\pi n}(\frac{n}{e})^{n}(1+\Theta(\frac{1}{n}))$
- n^{n}

Polynomial Time Algorithm (多项式时间算法)

An algorithm runs in **Polynomial Time** (多项式时间) if there exists an integer (整数) k such that its worst-case time complexity (最坏时间复杂性) is $O(n^k)$.

For example,

$T(n)=a_k n^k+a_{k-1}n^{k-1}+\ldots+a_1 n+a_0=O(n^k)$,

$T(n)=n^3+n^2\log n=O(n^3)+O(n^2\log n)=O(n^3)$,

$T(n)=O(n^{2.5}\log n)=O(n^3)$.

Useful Technical (技术的) Tool: Recurrences (递推)

Example 1. T(1)=1 and T(n)=T(n/2)+n

Induction (归纳) method

T(1)=1

T(2)=3

T(4)=7

T(8)=15

…

T(n)=2n-1 for $n=2^k$?

Prove the above guess (猜测) by induction (归纳) on k.

Substitution (替换) method

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

$$T(n/2)=T(n/4)+n/2$$

$$T(n/4)=T(n/8)+n/4$$

$$\dots$$

$$T(4)=T(2)+4$$

$$T(2)=T(1)+2$$

So we have $T(n)=1+2+4+\dots+n/4+n/2+n$. For $n=2^k$, we have

$$T(n)=1+2+4+\dots+2^k=2^{k+1}-1=2n-1.$$

Example 2. T(1)=1 and T(n)=T($\lceil \frac{n}{2} \rceil$)+$n$.

$\lceil x \rceil$ is equal to the smallest integer greater than or equal to x, e.g. $\lceil 2.5 \rceil$=3.

First step: Guess that T(n)=O(n), i.e. (即) T(n)≤cn for sufficiently large (充分大) n and some constant c.

Second Step: First make c large enough, e.g. c=4. Then prove T(n)≤4n by induction (归纳) on n. Easy to see that T(1)=1 ≤4×1=4. Assume (假设) that T(n)≤4n for n≤k, where (其中) k is a natural number (自然数).

Third Step: Now we have $T(k+1)=T(\lceil \frac{k+1}{2} \rceil)+k+1$. If k is an odd number (奇数), then

$$T(k+1)=T(\frac{k+1}{2})+k+1\leq 4(\frac{k+1}{2})+k+1=3k+3\leq 4(k+1).$$

If k is an even number (偶数), then

$$T(k+1)=T(\frac{k}{2}+1)+k+1\leq 4(\frac{k}{2}+1)+k+1=3k+5\leq 4(k+1).$$

Combining (综合) the above two cases (情况), we finish the proof.

Example 3. $T(1)=1$ and $T(n)=2T(n/2)+n$.

Assume (假设) that $n=2^k$, we have

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

$$T(n/2)=2T(n/4)+n/2 \Rightarrow 2T(n/2)=2^2T(n/4)+n$$

$$T(n/4)=2T(n/8)+n/4 \Rightarrow 2^2T(n/4)=2^3T(n/8)+n$$

$$\cdots$$

$$2^{k-2}T(4)=2^{k-1}T(2)+n$$

$$2^{k-1}T(2)=2^kT(1)+n$$

So we have $T(n)=2^k+kn=n+n\log_2 n=O(n\log n)$.

In general (**Master Theorem**), $T(1)=d$ and for $n>1$,

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

has solution

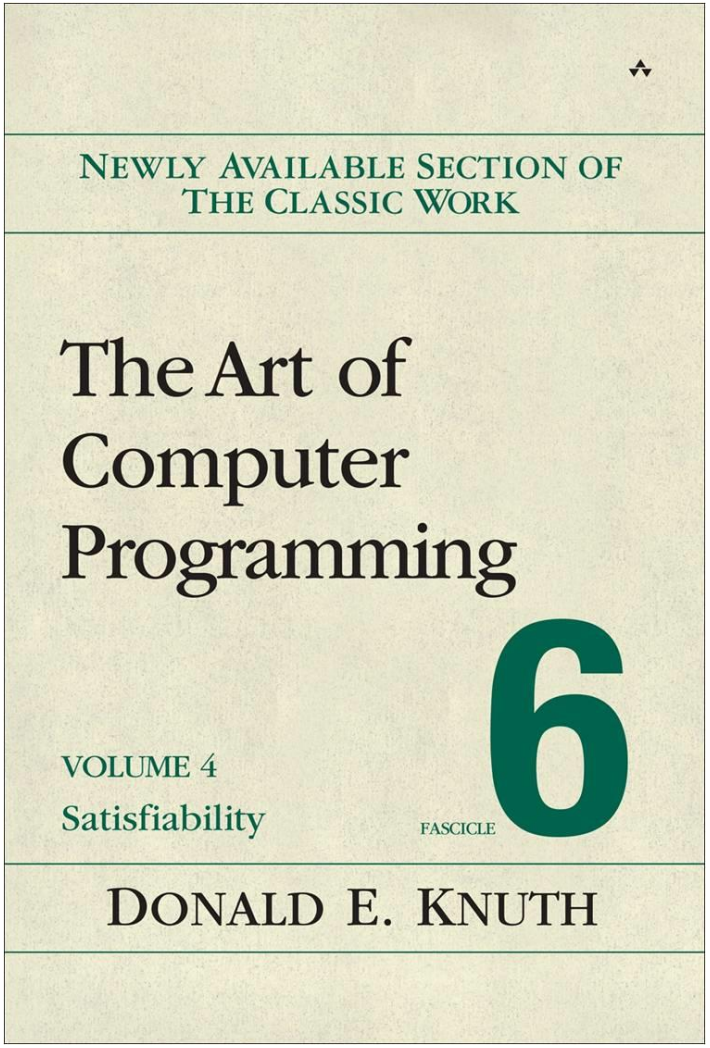
- if $a<b$, $T(n)=O(n)$;
- if $a=b$, $T(n)=O(n\log n)$;
- if $a>b$, $T(n)=O(n^{\log_b a})$

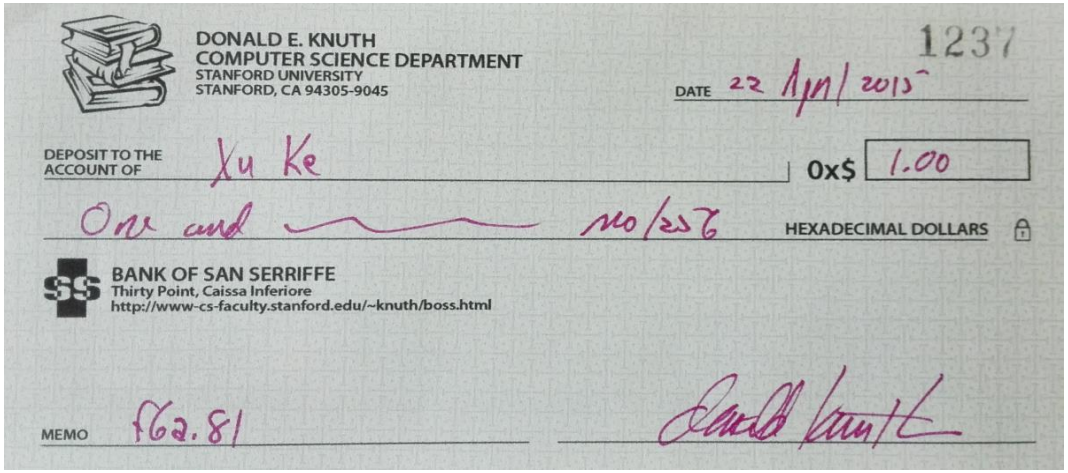
Two Books

Donald L. Graham, Donald E. Knuth, Oren Patashnik. Concrete (具体) Mathematics - A Foundation for Computer Science.

Donald E. Knuth. The Art of Computer Programming.

 Donald E. Knuth (高德纳), Stanford University.


NEWLY AVAILABLE SECTION OF
THE CLASSIC WORK
The Art of
Computer
Programming
VOLUME 4
Satisfiability
FASCICLE
6
DONALD E. KNUTH


DONALD E. KNUTH
COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CA 94305-9045
1237
DATE 22 Apr 2015
DEPOSIT TO THE ACCOUNT OF Xu Ke
0x$ 1.00
One and no/256
HEXADECIMAL DOLLARS
BANK OF SAN SERRIFFE
Thirty Point, Caissa Inferiore
http://www-cs-faculty.stanford.edu/~knuth/boss.html
MEMO f62.81
Donald Knuth

Exercise

Let $p(n) = a_k n^k + a_{k-1} n^{k-1} + ... + a_0$. Show that

$$p(n) = O(n^k)$$

Homework 1

Let $f(n)$, $g(n)$ be two non-negative functions. Prove that

$$\max\{f(n), g(n)\} = \Theta(f(n)+g(n)).$$

Homework 2: Prove Master Theorem.

Homework 3: Try to find an interesting practical problem and describe how to solve it using algorithms.