算法设计与分析

Design and Analysis of Computer Algorithm

刘 瑶

电子科技大学信息与软件学院

liuyao@uestc.edu.cn

办公室：沙河校区主楼中432

参考书

1. **计算机算法设计与分析（第4版），王晓东著，电子工业出版社，2012年**
2. **计算机算法基础（第二版），余祥宣等著，华中理工大学出版社，2000年**
3. **计算机算法导引——设计与分析，卢开澄著，清华大学出版社，1996年**
4. **计算机算法设计与分析，卢开澄，中国铁道出版社，1998年**
5. **Introduction to Algorithms（第二版 影印版），Thomas H. Cormen著，高等教育出版社**

课程考核安排

- 学时：40学时
- 平时成绩（40%）+ 期末闭卷考试（60%）
- 平时成绩构成：

  考勤、作业、课堂讨论
- 作业：平时作业和一个报告

  报告：实现一个算法，算法尽量不要相同

  题目可以为实现书中的程序

课件下载和作业提交

网络学堂

http://www.wlxt.uestc.edu.cn/wlxt/

QQ群号： 526268077

加群密码：ada-ss2018

第1章
算 法 概 述
Algorithm Introduction

介绍算法设计的基本概念
及算法分析的方法和准则.

学习要点：

- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述。
- 掌握用C++语言描述算法的方法。

1. 1 算法 Algorithm

算法是什么？

算法，一个既陌生又熟悉的名词。从小学就开始接触算法。例如，做四则运算要先乘除后加减，从里往外脱括弧等等都是算法，只要按照一定的程序一步一步做，一定不会错。因此，算法其实是耳熟能详的数学对象。一般地，**算法是指在解决问题时按照某种机械程序步骤一定可以得到结果的处理过程。**这种过程必须是确定的、有效的、有限的。

“如果你在森林里迷路了，保持冷静，调动常识，走一步看一步。”

——这里是建议而非算法。

童子军的条例：

如果你在森林里迷路了，一直往下走，直到溪流旁，然后顺流而下，最后你会到达一个城镇。

——这是一个算法。

1. 算法定义

一系列将问题的输入转换为输出的计算或操作步骤。

2. 计算机算法与人工算法

有些问题没有计算机算法.

有些问题计算机算法与人工算法不同.

3. 计算机算法的一般特征

(1)输 入:

有外部提供的量作为算法的输入。

(2)输 出:

算法产生至少一个量作为输出。

(3)确定性：**definiteness**

组成算法的每条指令是清晰，无歧义的。

(4)有限性：**finiteness**

算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

4. 算法的三个要素

1).数据: 运算序列中作为运算对象和结果的数据.

2).运算: 运算序列中的各种运算:赋值, 算术和逻辑运算

3).控制和转移: 运算序列中的控制和转移.

5. 算法描述语言

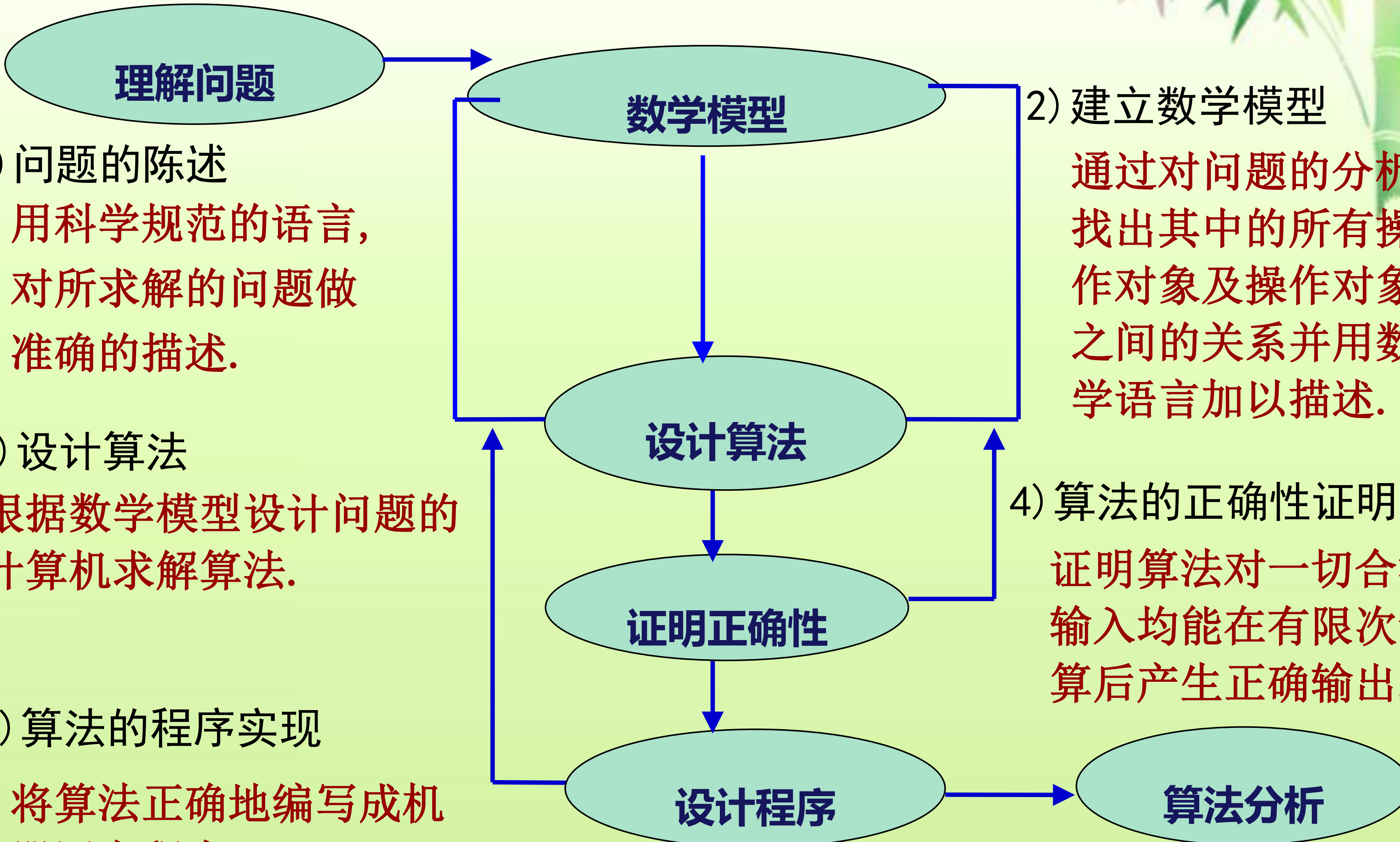
自然语言, 数学语言, 流程图, 程序设计语言等等.

6. 算法分类

从解法上
- 数值型算法:算法中的基本运算为算术运算.
- 非数值型算法:算法中的基本运算为逻辑运算.

从处理方式上
- 串行算法:串行计算机上执行的算法.
- 并行算法:并行计算机上执行的算法.

7. 问题的求解过程



1) 问题的陈述

用科学规范的语言，对所求解的问题做准确的描述.

2) 建立数学模型

通过对问题的分析，找出其中的所有操作对象及操作对象之间的关系并用数学语言加以描述.

3) 设计算法

根据数学模型设计问题的计算机求解算法.

4) 算法的正确性证明

证明算法对一切合法输入均能在有限次计算后产生正确输出.

5) 算法的程序实现

将算法正确地编写成机器语言程序.

6) 算法分析

对执行该算法所消耗的计算机资源进行估算.

8. 算法与程序、数据结构的关系

过程：算法+数据结构≈程序

对象：对象+消息≈程序

侧重点不同

数据的结构，直接影响算法的选择和效率。

算法——程序的灵魂

8. 算法与程序、数据结构的关系

数据结构的三个方面：

- 数据的逻辑结构
  - 线性结构
    - 线性表
    - 栈
    - 队列
    - 串及数组
  - 非线性结构
    - 树形结构
    - 图形结构
- 数据的存储结构
  - 顺序存储
  - 链式存储
  - 索引存储
  - 散列存储
- 数据的运算：检索、排序、插入、删除、修改等

1.2 算法复杂性分析

1. 复杂性的计量

算法的复杂性:算法执行所需的时间和空间的数量.

- **显然,它与问题的规模,算法的输入数据及算法本身有关.**

令 N:问题的规模 I:输入数据 A:算法本身

则算法的复杂性 C=F (N,I,A)

将时间复杂性和空间复杂性分别考虑,并用T和S表示.则

$$T=T(N,I,A) \qquad S=S(N,I,A)$$

将A隐含在函数名中,则 T=T(N,I,A) 简化为T=T(N,I)

设一台抽象计算机提供的元运算有k种,分别记作$O_1,\ldots,O_k$

设这些元运算每执行一次所需时间分别为$t_1, t_2,\ldots,t_k$,

设算法A中用到O_i的次数为 e_i, $i=1,\ldots,k$,则 $e_i= e_i(N,I)$

$$T=T(N,I)=\sum_{i=1}^{k} t_i \cdot e_i(N, I)$$

最好情况： $T_{min}(N) = \min_{I \in D_N} T(N,I) = \min_{I \in D_N} \sum_{i=1}^{k} t_i \cdot e_i(N,I)$

$$= \sum_{i=1}^{k} t_i \cdot e_i(N,\tilde{I}) = T(N,\tilde{I})$$

最坏情况: $T_{max}(N) = \max_{I \in D_N} T(N,I) = \max_{I \in D_N} \sum_{i=1}^{k} t_i \cdot e_i(N,I)$

$$= \sum_{i=1}^{k} t_i \cdot e_i(N,I^*) = T(N,I^*)$$

平均情况: $T_{avg}(N) = \sum_{I \in D_N} P(I) \cdot T(N,I) = \sum_{I \in D_n} P(I) \cdot \sum_{i=1}^{k} t_i \cdot e_i(N,I)$

其中 D_N:规模为N的所有合法输入的集合

I^*: D_N中达到T_{max} (N)的一个输入

$\tilde{I}$: D_N中达到T_{min} (N)的一个输入

P(I): 出现输入为I的概率

例题1-1 已知不重复且从小到大排列的m个整数的数组A[1...m], $m=2^K$, K为正整数.对于给定的整数c,要求找到一个下标i,使得A[i]=c.找不到返回0.

算法1-1:顺序查找

```
function search(c)
    { i:=1;                                   a
      while A[i]<c and i<m do                 2mt
          i:=i+1;                             (m-1) (s+a)
      if A[i]=c                               t
      then search:=i;
      else search:=0;                         a
    }
```

分析:问题的规模为m,设元运算执行时间为赋值:a,判断:t,加法:s, 并设c在A中.

最好情况T_{min} (m)=a+2t+t+a=2a+3t

最坏情况T_{max}(m) =(m+1)a+(2m+1)t+(m-1)s

平均情况T_{avg}(m)=0.5(m+3)a+(m+2)t+0.5(m-1)s

例 题 1-2 算法1-2:二分查找 (假定c是A的最后一元)

```
function b-search(c)
    {  L:=1;  U:=m;                                2a
       found:=false;                               a
       while not found  and U>=L do                3t (logm+2)
         { i:=(L+U)div2;                           (a+s+d)(logm+1)
           if c=A[i]                               2t (logm+1)
           then   found:=true                      a
           else if c>A[i]                          2t (logm+1)
                then L:=i+1                        (s+a)(logm+1)
                 else  U:=i-1
           }
       if found                                    t
       then b-search:=i
       else b-search:=0                            a
      }
```

分析:问题规模为m,元运算执行时间设为赋值a,判断t, 加法s, 除法d, 减法b.

最坏情况$T_{max}(m) = 7a+11t+2s+d+(2a+2s+7t+d)$ logm

2 复杂性的渐进性态

1). 渐进性态

设T(n)为算法A的时间复杂性函数,则它是n的单增函数,如果存在一个函数$\tilde{T}(n)$使得当$n\to\infty$,有

$$(T(n)-\tilde{T}(n))/T(n)\to 0$$

称$\tilde{T}(n)$是T(n)当$n\to\infty$时的渐进性态 或 渐进复杂性.

- 在数学上,T(n)与$\tilde{T}(n)$有相同的最高阶项.可取$\tilde{T}(n)$为略去T(n)的低阶项后剩余的主项.当n充分大时我们用$\tilde{T}(n)$代替T(n)作为算法复杂性的度量,从而简化分析.

  例如 $T(n)=3n^2+4n\log n+7$, 则$\tilde{T}(n)$可以是$3n^2$.

  若进一步假定算法中所有不同元运算的单位执行时间相同,则可不考虑$\tilde{T}(n)$所包含的系数或常数因子。

- 渐进分析适用于n充分大的情况,当问题的规模很小时,或比较的两算法同阶时,则不能做这种简化.

2).渐进性态的阶

设f(N)和 g (N) 是定义在正整数集上的正函数,

(1)**大O表示法**（算法运行时间的上限 ）

若存在正常数c和自然数N_0 使得当 $N \geq N_0$ 时,有$f(N) \leq cg(N)$
则称函数 $f(N)$在N充分大时有**上界**, 且 $g(N)$是它的一个上界,
记为 $f(N) = O(g(N))$，也称 $f(N)$ **的阶不高于**$g(N)$ **的阶**.

例如 $3n=O(n)$, $n+1024=O(n)$, $2n^2+11n-10=O(n^2)$

$n^2=O(n^3)$? $n^3=O(n^2)$?

又如算法1-1中

$T_{min}(m)=2a+3t$, $T_{min}(m)=O(1)$

$T_{max}(m)=(m+1)a+(2m+1)t+(m-1)s$,

$T_{max}(m)=O(m)$

运算法则	
	1. O(f)+O(g)=O(max(f,g))
	2. O(f)+O(g)=O(f+g)
	3. O(f)·O(g)=O(f·g)
	4. 如果 g(n)=O(f(n)),则 O(f)+O(g)=O(f)
	5. f=O(f)
	6. O(cf(n))=O(f(n))

例如 估计如下二重循环算法在最坏情况下时间复杂性T(n)的阶.

```
for i:= 1 to n do
    for j:=1 to i do
     {s1,s2,s3,s4} ; //s1,s2,s3,s4为单一赋值语句
```

分析:内循环体只需O(1)时间,故

内循环共需 $\sum_{j=1}^{i} O(1) = O(\sum_{j=1}^{i} 1) = O(i)$

外循环共需 $\sum_{i=1}^{N} O(i) = O(\sum_{i=1}^{N} i) = O(\frac{N(N+1)}{2}) = O(N^2)$

(2)大Ω表示法（算法运行时间的下限）

如果∃正常数c和自然数N_0使得当 $N \geq N_0$ 时，有$f(N) \geq cg(N)$ 则称函数f(N)在N充分大时有下限，且 $g(N)$是它的一个下限，记为$f(N) = \Omega(g(N))$ 也称f(N)的阶不低于$g(N)$的阶。

(3)θ表示法

$f(N) = \theta(g(N))$ 当且仅当 $f(N) = O(g(N))$ 且$f(N) = \Omega(g(N))$ 称函数f(N)与$g(N)$同阶.

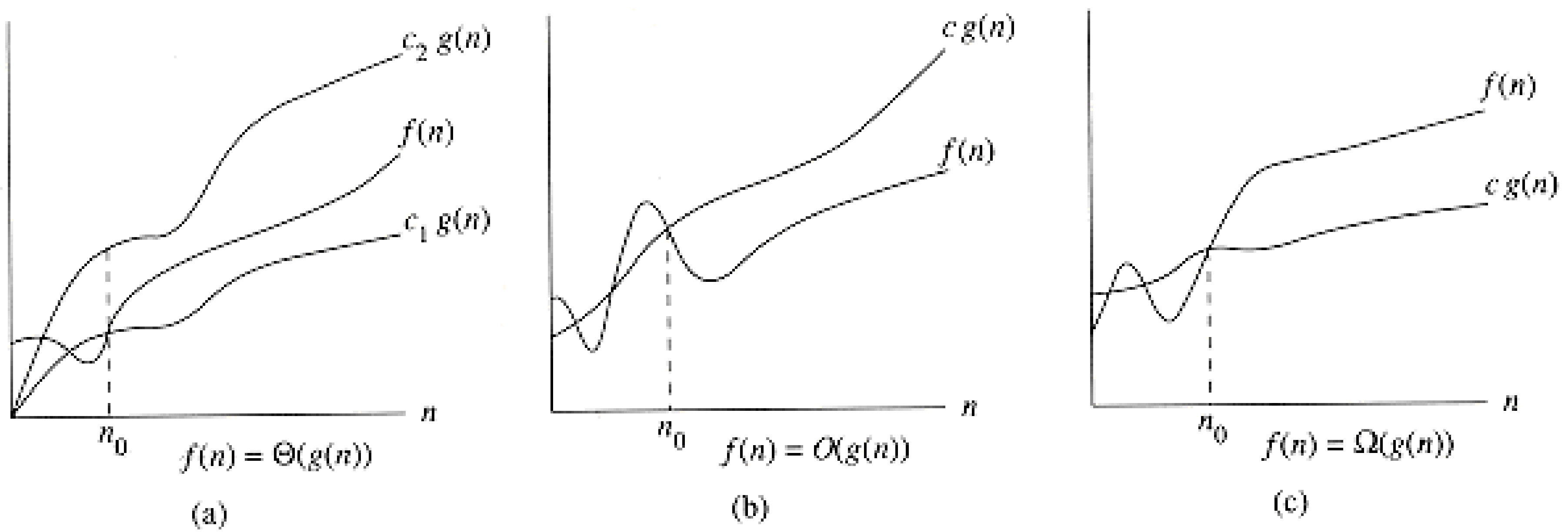
算法的渐进复杂性的阶对于算法的效率有着决定性的意义：

多项式阶算法(有效算法)：时间复杂性与规模N 的幂同阶.

指数阶算法：时间复杂性与规模N 的一个指数函数同阶.

最优算法：时间复杂性达到其下界的算法.

渐近分析的符号



$f(n)=\Theta(g(n))$	$f(n)=O(g(n))$	$f(n)= \Omega(g(n))$
$\Updownarrow \cong$	$\Updownarrow \cong$	$\Updownarrow \cong$
$f(n)=g(n)$	$f(n) \le g(n)$	$f(n) \ge g(n)$

常见的多项式阶有:

$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3)$

常见的指数阶有：$O(2^n) < O(n!) < O(n^n)$

- 对规模较小的问题,决定算法工作效率的可能是算法的简单性而不是算法执行的时间.
- 当比较两个算法的效率时,若两个算法是同阶的,必须进一步考察阶的常数因子才能辨别优劣.

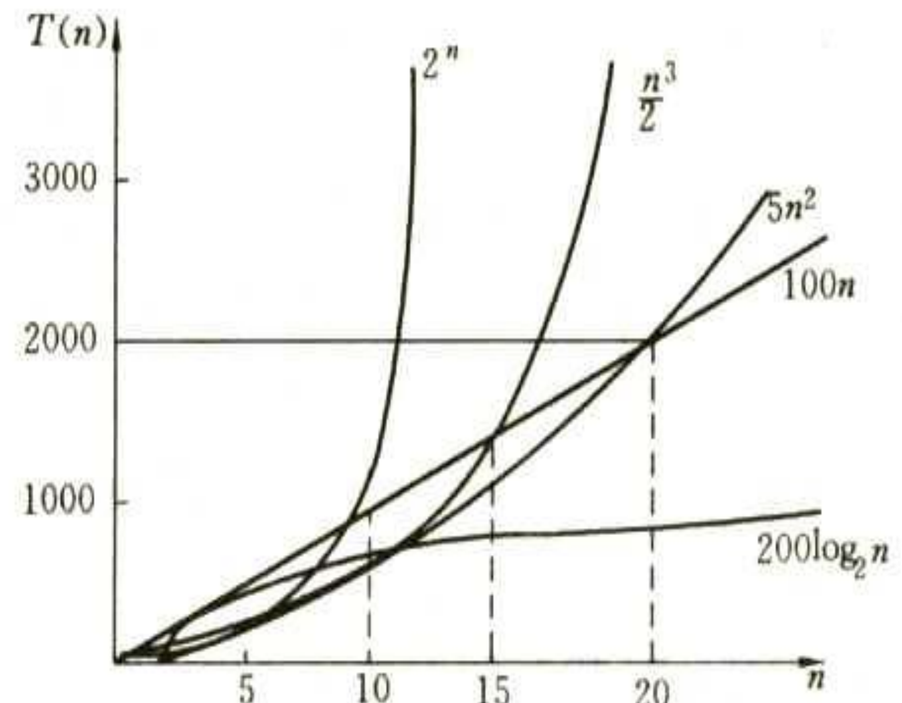


图1_时间函数的增长率

最常用的关系式

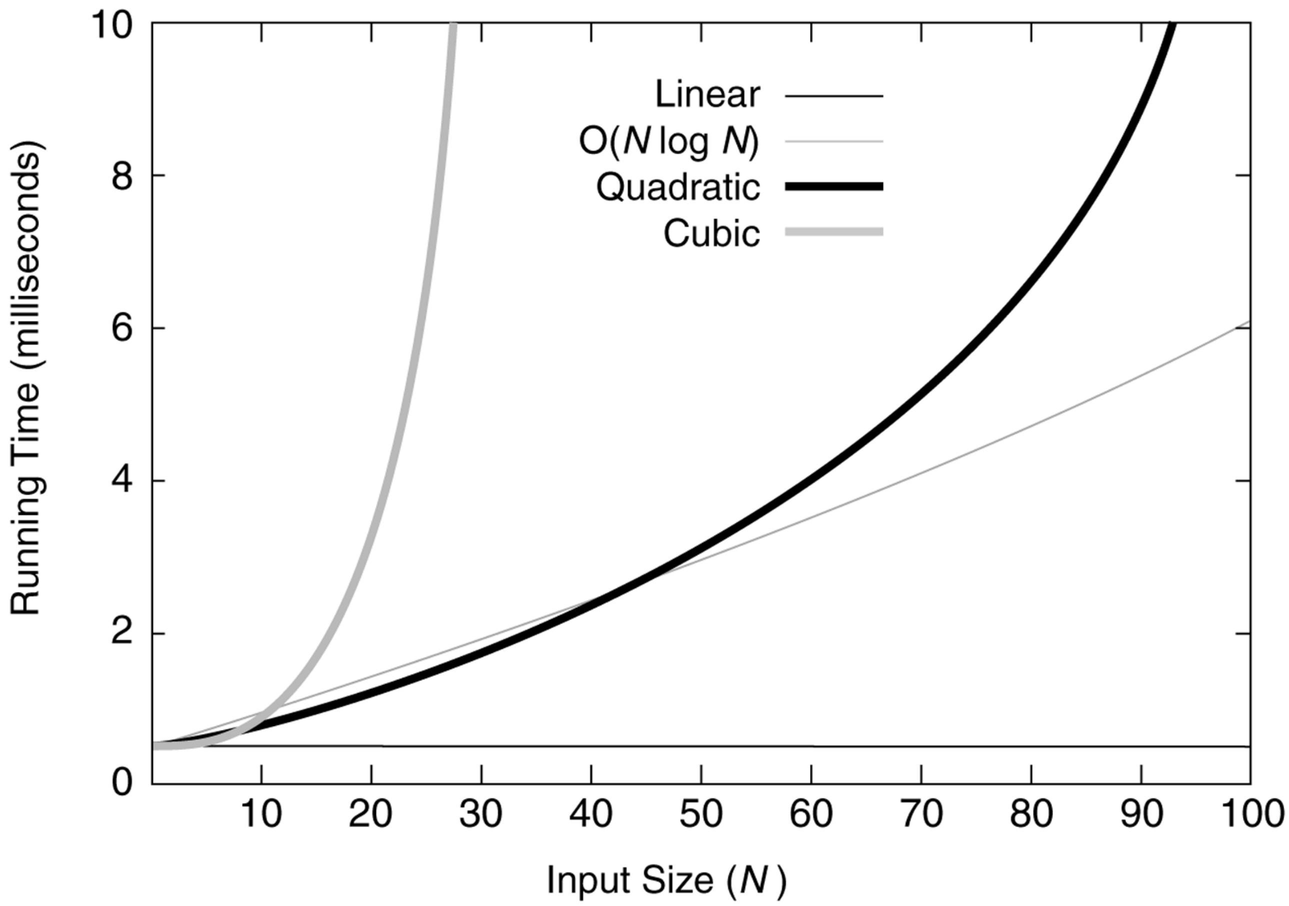
- 多项式. $a_0 + a_1 n + \ldots + a_d n^d = \Theta(n^d)$ 其中 $a_d > 0$.
- 对数. $O(\log_a n) = O(\log_b n)$ 其中 $a, b > 0$为常数.
- 对数. 对任意 $x > 0$, $\log n = O(n^x)$.
- 指数. 对任意 $r > 1$ 和 $d > 0$, $n^d = O(r^n)$.

重点记住内容!

4. 算法分析中常见的复杂性函数

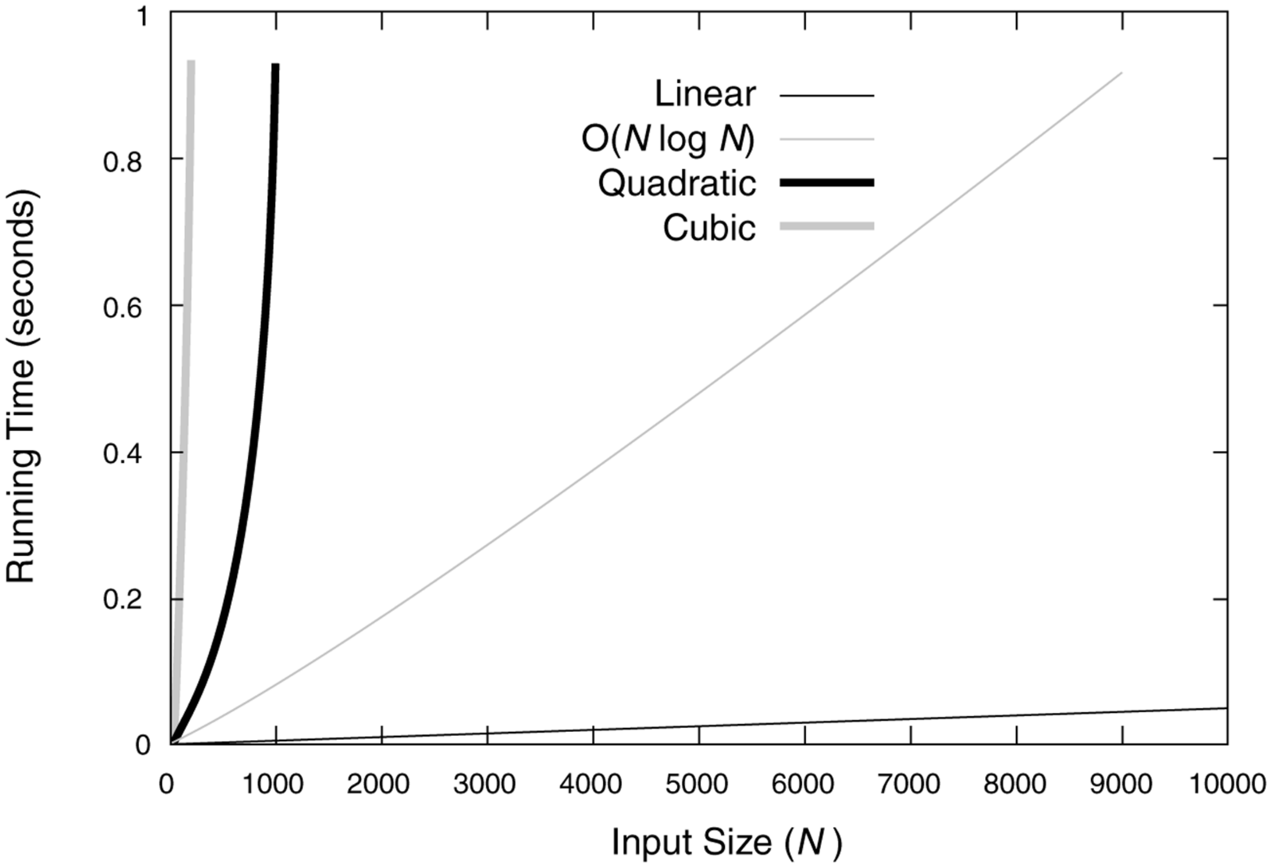
FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	$N \log N$
N^2	Quadratic
N^3	Cubic
2^N	Exponential

小规模数据



Linear
O(N log N)
Quadratic
Cubic
Running Time (milliseconds)
Input Size (N)
0
2
4
6
8
10
10
20
30
40
50
60
70
80
90
100

中等规模数据



Linear
O(N log N)
Quadratic
Cubic
Running Time (seconds)
Input Size (N)
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

3. 渐进分析

例 题1-1

时间复杂性渐进阶分析的规则:(最坏情况)

1). 赋值,比较,算术运算,逻辑运算,读写单个变量(常量)只需1单位时间

2). 执行条件语句 **if** ***c*** **then** S1 **else** S2 的时间为$T_C+\max(T_{S1},T_{S2})$.

3). 选择语句 **case A of a1**: **s1**; **a2**: **s2**; …; ***am***: ***sm***

需要的时间为 $\max(T_{S1},T_{S2},\ldots,T_{Sm})$.

4). 访问数组的单个分量或纪录的单个域需要**一个单位时间**.

5). 执行**for**循环语句的时间=**执行循环体时间*循环次数.**

6). **while** ***c*** **do** ***s*** **(repeat** ***s*** **until** ***c*****)**语句时间=**(T*c*+T*s*)*循环次数.**

7). 用**goto**从循环体内跳到循环体末或循环后面的语句时,不需额外时间

8). 过程或函数调用语句

对非递归调用,根据调用层次由里向外用规则1-7进行分析;

对递归调用,可建立关于**T(n)的**递归方程,求解该方程得到**T(n)**.

例题 1-2 算法1-2:二分查找 (假定c是A的最后一元)

```
function b-search(c)
    { L:=1;  U:=m;                                  2
      found:=false;                                 1
      while not found  and U>=L do                    3
        { i:=(L+U)div2;                           3
          if c=A[i]                              2
          then   found:=true                   1      Logm+1
          else if c>A[i]                        2
                then L:=i+1                     2
                else  U:=i-1
        }
      if found                                          1
      then b-search:=i
      else b-search:=0                           1
   }
```

分析:问题规模为m,元运算执行时间设为赋值a,判断t, 加法s, 除法d, 减法b.

最坏情况$T_{max}(m) = 7a+11t+2s+d+(2a+2s+7t+d)\log m = 21+12\log m$

例 题 1-3 已知不重复且从小到大排列的m个整数的数组A[1...m], $m=2^K$, K为正整数.对于给定的整数c,要求找到一个下标i,使得A[i]=c.找不到返回0.

算法1-3:二分查找递归算法

```
function b-search(c,L,U)
{ if  U<L  then                                    1
      b-search:=0                                      1
  else {  index:=(L+U)div2 ;                       3
          element:=A[index];                       2
          if   element = c   then                  1
              b-search:= index;                    1
          else  if  element >c then                    1
                  b-search:= b-search(c,L, index-1);   3+T(m/2)
               else
                  b-search:= b-search(c,index+1,U);    3+T(m/2)
      };   }
```

设T(m)是b-search在最坏情况下的时间复杂性,则T(m)满足如下递归方程:

$$T(m)=\begin{cases} 2 & m=0 \\ 13 & m=1 \\ 11+T(m/2) & m>1 \end{cases}$$

解得: T(m) =logm+13= θ(logm)

例 题 1-3 已知不重复且从小到大排列的m个整数的数组A[1…m], $m=2^K$, K为正整数.对于给定的整数c,要求找到一个下标i,使得A[i]=c.找不到返回0.

算法1-3:二分查找递归算法

```
function b-search(c,L,U)                                      m=1
{ if  U<L  then                                                1      1
      b-search:=0
  else {  index:=(L+U)div2 ;                                   3
          element:=A[index];                                   2
          if   element = c   then                              1
               b-search:= index;
          else  if  element >c then                                   1
                    b-search:= b-search(c,L, index-1);             3
                else                                                  or
                    b-search:= b-search(c,index+1,U);              3
      };   }
```

设T(m)是b-search在最坏况的时间复杂性,则T(m)满足如下递归方程:

T(m)=1+3+2+1+1+3+1+1=13　(m=1)

例题1-3 已知不重复且从小到大排列的m个整数的数组A[1…m],m=2^K, K为正整数.对于给定的整数c,要求找到一个下标i,使得A[i]=c.找不到返回0.

算法1-3:二分查找递归算法

```
function b-search(c,L,U)                                  m>1
{ if  U<L  then                                              1
      b-search:=0
  else {  index:=(L+U)div2 ;                                 3
          element:=A[index];                                 2
          if   element = c   then                            1
               b-search:= index;
          else  if  element >c then                                 1
                   b-search:= b-search(c,L, index-1);             3
               else                                                    or
                   b-search:= b-search(c,index+1,U);              3
      };   }
```

设T(m)是b-search在最坏况的时间复杂性,则T(m)满足如下递归方程:

T(m)=1+3+2+1+1+3+T(m/2)=11+T(m/2)   (m>1)

例题 1-4 求 Fibonacci 数列的前N项 $a_0, a_1, \ldots\ a_N$ 其中，$a_0=0, a_1=1, a_i=a_{i-1}+a_{i-2}$

算法1-4	
`Procedure seq(n)`	
`function A(n)`	
`{ if n=0 then`	1
`A:=0`	1
`else if n=1 then`	1
`A:=1`	1
`else A:=A(n-1)+A(n-2)`	6+F(n-1)+F(n-2)
`};`	
`{ if n<0 then`	1
`error`	
`else for i:=0 to n do`	
`writeln (A(i))`	$\sum_{i=0}^{n}(1+F(i))$
`};`	

设F(n)是函数A在最坏情况下的时间复杂性,则F(n)满足如下递归方程:

$$\mathbf{F(n)}=\begin{cases} 2 & n=0 \\ 3 & n=1 \\ 8+\mathbf{F(n-1)}+\mathbf{F(n-2)} & n>1 \end{cases}$$

算法分析的基本法则

- **非递归算法：**
- （1）for / while 循环
- 循环体内计算时间*循环次数；
- （2）嵌套循环
- 循环体内计算时间*所有循环次数；
- （3）顺序语句
- 各语句计算时间相加；
- （4）if-else语句
- if语句计算时间和else语句计算时间的较大者。

5 算法复杂度的影响

- 对问题处理能力、运行时间有影响的因素有：
  - 硬件设备的性能
  - 系统软件
  - 输入数据
- 起决定性作用的是算法渐近复杂度。
- 在问题规模较小时，常数因子也不可忽视。
- 实际工作中考虑的因素

例1-5：解决某问题有三种算法，复杂性分别为1000N，$10N^2$， 2^N，在一台机器上可处理问题的规模分别为S1， S2， S3 。若机器速度提高到原来的10倍，问在同样时间内可处理问题的大小如何？

解：

复杂性	原来处理问题规模	速度提高以后处理问题规模
1000N	S1	10S1
$10N^2$	S2	3.16S2
2^N	S3	S3 +log10≈ S3 +3.32

例1-6： 问题P的算法复杂度为$T(n)=n^3$（毫秒），现改善为$T(n)=n^2$（毫秒）。问原来运行一小时的问题实例，现在要运行多少时间?

解： 设实例大小为n，

则 $n^3=3600*1000$

$n=153.3$

∴ 现在需要时间$t=153.3^2$毫秒≈ 23.5秒

6. 算法渐近复杂性分析中常用函数

- **（1）单调函数**
- 单调递增：$m \le n \Rightarrow f(m) \le f(n)$;
- 单调递减：$m \le n \Rightarrow f(m) \ge f(n)$;
- 严格单调递增：$m < n \Rightarrow f(m) < f(n)$;
- 严格单调递减：$m < n \Rightarrow f(m) > f(n)$.
- **（2）取整函数**
- $\lfloor x \rfloor$：不大于x的最大整数；
- $\lceil x \rceil$：不小于x的最小整数。

取整函数的若干性质

- $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$；
- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;
- 对于$n \ge 0$，$a,b>0$，有：
- $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$;
- $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;
- $\lceil a/b \rceil \le (a+(b-1))/b$;
- $\lfloor a/b \rfloor \ge (a-(b-1))/b$;
- $f(x)=\lfloor x \rfloor$, $g(x)=\lceil x \rceil$为单调递增函数。

- **（3）多项式函数**
- $p(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$； $a_d>0$;

  $p(n) = \theta(n^d)$;
- $f(n) = O(n^k) \Leftrightarrow f(n)$多项式有界；
- $f(n) = O(1) \Leftrightarrow f(n) \leq c$;
- $k \geq d \Rightarrow p(n) = O(n^k)$;
- $k \leq d \Rightarrow p(n) = \Omega(n^k)$;

- **（4）指数函数**
- 对于正整数m,n和实数$a>0$:
- $a^0=1$;
- $a^1=a$;
- $a^{-1}=1/a$;
- $(a^m)^n = a^{mn}$;
- $(a^m)^n = (a^n)^m$;
- $a^m a^n = a^{m+n}$;
- $a>1 \Rightarrow a^n$为单调递增函数;
- $a>1 \Rightarrow \lim_{n\to\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n)$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

- $e^x \geq 1+x$;
- $|x| \leq 1 \Rightarrow 1+x \leq e^x \leq 1+x+x^2$;
- $e^x = 1+x+ \theta(x^2)$, as $x \to 0$;

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$$

- **（5）对数函数**
- $\log n = \log_2 n$;
- $\lg n = \log_{10} n$;
- $\ln n = \log_e n$;
- $\log^k n = (\log n)^k$;
- $\log \log n = \log(\log n)$;
- for a>0,b>0,c>0

$$a = b^{\log_b a}$$

$$\log_c(ab) = \log_c a + \log_c b$$

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

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b (1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

- $|x| \leq 1 \Rightarrow \quad \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots.$
- for $x > -1$, $\quad \frac{x}{1+x} \leq \ln(1+x) \leq x$
- for any $a > 0$, $\quad \lim_{n\to\infty} \frac{\log^b n}{(2^a)^{\log n}} = \lim_{n\to\infty} \frac{\log^b n}{n^a} = 0 \Rightarrow \log^b n = o(n^a)$

- （6）阶层函数

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

- Stirling's approximation

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

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \qquad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

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

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

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

用c++描述算法

<table>
<tr><th>CATEGORY</th><th>EXAMPLES</th><th>ASSOCIATIVITY</th></tr>
<tr><td>Operations on References</td><td>. []</td><td>Left to right</td></tr>
<tr><td>Unary</td><td>++ -- ! - (type)</td><td>Right to left</td></tr>
<tr><td>Multiplicative</td><td>* / %</td><td>Left to right</td></tr>
<tr><td>Additive</td><td>+ -</td><td>Left to right</td></tr>
<tr><td>Shift (bitwise)</td><td><< >></td><td>Left to right</td></tr>
<tr><td>Relational</td><td>< <= > >= instanceof</td><td>Left to right</td></tr>
<tr><td>Equality</td><td>== !=</td><td>Left to right</td></tr>
<tr><td>Boolean (or bitwise) AND</td><td>&</td><td>Left to right</td></tr>
<tr><td>Boolean (or bitwise) XOR</td><td>^</td><td>Left to right</td></tr>
<tr><td>Boolean (or bitwise) OR</td><td>|</td><td>Left to right</td></tr>
<tr><td>Logical AND</td><td>&&</td><td>Left to right</td></tr>
<tr><td>Logical OR</td><td>||</td><td>Left to right</td></tr>
<tr><td>Conditional</td><td>?:</td><td>Right to left</td></tr>
<tr><td>Assignment</td><td>= *= /= %= += -=</td><td>Right to left</td></tr>
</table>

- **（1）选择语句：**
- **（1.1）if 语句：**

```
if (expression) statement;
else statement;
```

- **（1.2）？语句：**
-

```
exp1?exp2:exp3
y= x>9 ? 100:200; 等价于：
if (x>9) y=100;
else y=200;
```

(1.3) switch语句：

```
switch (expression) {
    case 1:
      statement sequence;
      break;
    case 2:
      statement sequence;
      break;
    …
    default:
      statement sequence;
  }
```

（2）迭代语句：

- **（2.1) for 循环：**
- for (init;condition;inc) statement;
- **（2.2) while 循环：**
- while (condition) statement;
- **（2.3) do-while 循环：**
- do{
-   statement;
- } while (condition);

（3）跳转语句：

- **（3.1) return语句：**
- return expression;
- **（3.2) goto语句：**
- goto label;
- …
- label:

（4）函数：

```
return-type function name(para-list)
{
     body of the function
 }
```

- **例：**

```
int max(int x,int y)
{
   return x>y?x:y;
}
```

（5）模板template：

```
template <class Type>
Type max(Type x,Type y)
{
  return x>y?x:y;
}

int i=max(1,2)；
double x=max(1.0,2.0)；
```

（6）动态存储分配：

- **（6.1）运算符new ：**
- 运算符new用于动态存储分配。
- new返回一个指向所分配空间的指针。
- 例：int *x；y=new int；*y=10；
- 也可将上述各语句作适当合并如下：
- int *y=new int；*y=10；
- 或 int *y=new int(10)；
- 或 int *y；y=new int(10)；

（6.2）一维数组：

- 为了在运行时创建一个大小可动态变化的一维浮点数组x，可先将x声明为一个float类型的指针。然后用new为数组动态地分配存储空间。
- **例：**
- float *x=new float[n]；
- 创建一个大小为n的一维浮点数组。运算符new分配n个浮点数所需的空间，并返回指向第一个浮点数的指针。
- 然后可用x[0]，x[1]，…，x[n-1]来访问每个数组元素。

（6.3）运算符delete：

- 当动态分配的存储空间已不再需要时应及时释放所占用的空间。
- 用运算符delete来释放由new分配的空间。
- **例：**
- delete y；
- delete [ ]x；
- 分别释放分配给*y的空间和分配给一维数组x的空间。

（6.4）动态二维数组：

- 创建类型为Type的动态工作数组，这个数组有rows行和cols列。

```
template <class Type>
void Make2DArray(Type** &x,int rows, int cols)
{
    x=new Type*[rows];
    for (int i=0;i<rows;i++)
      x[i]=new Type[cols];
}
```

- 当不再需要一个动态分配的二维数组时，可按以下步骤释放它所占用的空间。首先释放在for循环中为每一行所分配的空间。然后释放为行指针分配的空间。

```
template <class Type>
void Delete2DArray(Type** &x,int rows)
{
     for (int i=0;i<rows;i++)
        delete []x[i];
    delete []x;
    x=0;
}
```

- 释放空间后将x置为0，以防继续访问已被释放的空间。

总 结

- 算法的概念。
- 程序、数据结构、算法
- 算法的空间复杂度和时间复杂度。
- 大O表示法 、大Ω 表示法、θ表示法。