数据结构与算法

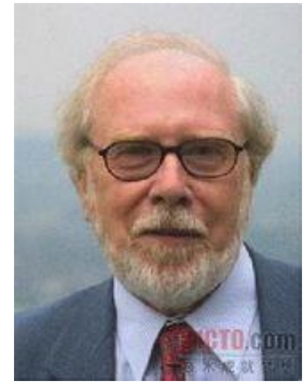
Data Structures and Algorithms

张 正

本章主要内容

1.1	数据结构研究对象
1.2	数据结构发展概况
1.3	抽象数据型(ADT)*
1.4	数据结构与程序设计
1.5	算法描述与算法分析*

1.4 数据结构与程序设计



Algorithms + Data Structures = Programs
算法 + 数据结构 = 程序设计

尼古拉斯•沃斯（Niklaus Wirth，1934年2月15日）；
生于瑞士温特图尔，瑞士计算机科学家；
苏黎世工学院获得学士学位，美国加州大学伯克利分校获得博士学位；
代表性著作有：

- 《系统程序设计导论》（《Systematic Programming：An Introduction》，Prentice-Hall，1973；
- 《算法+数据结构=程序》（《Algorithms + Data Structures=Programs》，Prentice-Hall，1976）；
- 《算法和数据结构》（《Algorithms and Data Structures》，Prentice-Hall，1986）；
- 《Modula-2程序设计》（《Programming in Modula-2》， Springer，1988，第4版）。
- 《PASCAL用户手册和报告：ISO PASCAL标准》 （《PASCAL User Manual and Report：ISO PASCAL Standard》，Springer，1991）；
- 《Oberon计划：操作系统和编译器的设计》（《Project Oberon：the Design of an Operating System and Compiler》，ACM Pr.，1992）；
- 《Oberon程序设计：超越Pascal和Modula》（《Programming in Oberon：Steps beyond Pascal and Modula》，ACM Pr.，1922）；
- 《数字电路设计教材》（《Digital Circuit Design for Computer Science Students：An Introductory Textbook》，Springer，1995）。

获奖：
1984年：获图灵奖（ACM）；
1983年：Emanual Piore奖（IEEE）；
1992年：加州大学伯克利分校命名沃斯为“杰出校友”。
1987年：获计算机科学教育杰出贡献奖（ACM）；
1988年：计算机先驱奖（IEEE）；

例一：求一元二次方程的根

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

这里的数据：a, b, c　　　三个离散的变量

例二：给定3个整型数，设计算法将3个整型数按升序排列

例三：给定100个整型数，设计算法将这100个整型数按升序排列

（1）int A[100]； 确定数据结构，组织数据；

（2）“冒泡”排序算法；

《数据结构与算法》

设计适合算法实现的数据结构；

是对数据结构适用性的验证；

侧重算法的实现；

《算法设计与分析》

用数学方法研究算法；

侧重算法的正确性证明和算法分析；

问题总是先于算法

问题 ⇨ 解决问题的算法 ⇨ 实现算法的程序

程序设计的四个里程碑：

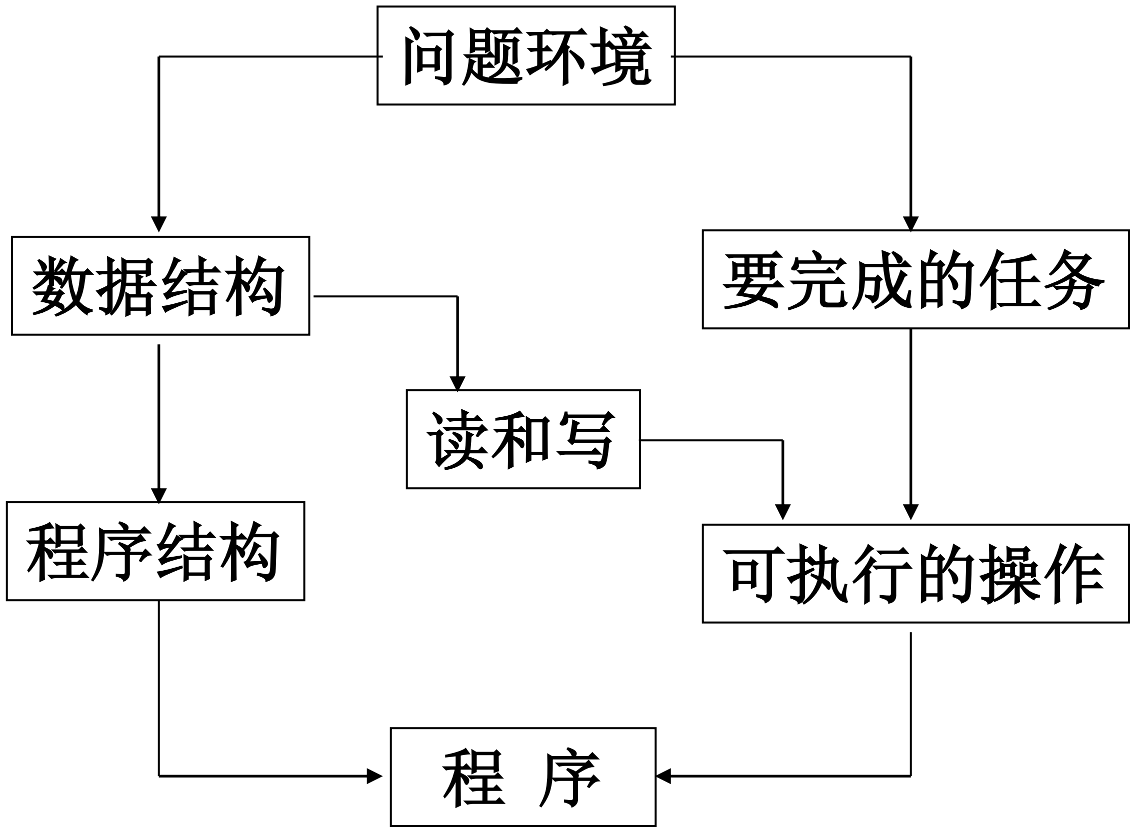
①子程序、②高级语言、③结构程序设计、④面向对象(**OOP**)

结构化程序设计：

①限制使用**GO TO**语句（基于三种基本结构）；
②逐步求精的设计方法；
③自顶向下的设计、编码与调试；
④主程序员组的组织形式；

提高算法设计能力

读 → 用 → 改 → 设



程序结构基于数据结构的根源

基于数据结构的jackson设计方法：

①研究问题环境，确定要处理的数据结构；

②基于数据结构，形成程序结构（骨架）；

③用初等操作来定义要完成的任务，并分配初等操作。

“从上到下，逐步求精”

“我们对复杂性问题的最重要的办法是抽象，对一个复杂问题，不应马上用计算机指令、数字与逻辑字来表示，而应该用较为自然的抽象语句来表示，从而得出抽象程序。抽象程序对抽象的数据进行某些特定的运算并用某些合适的记号(可能是自然语言)来表示。

对抽象程序作进一步的分解，并进入下一层的抽象，这样的精细化过程一直进行下去，直到程序能被计算机接受为止。

此时的程序可能是某种高级语言或机器指令书写的。”

——*N. wirth*

本章主要内容

1.1	数据结构研究对象
1.2	数据结构发展概况
1.3	抽象数据型(ADT*)
1.4	数据结构与程序设计
1.5	算法描述与算法分析*

统考（408）数据结构考查内容

考查目标：

1、掌握数据结构的基本概念、基本原理和基本方法。

2、掌握数据的逻辑结构、存储结构及基本操作的实现，能够对算法进行基本的时间复杂度与空间复杂度的分析。

3、能够运用数据结构的基本原理和方法进行问题的分析与求解，具备采用C或C++语言设计与实现算法的能力。

HIT考研大纲

考试要求

1. 要求考生全面系统地掌握数据结构与算法的基本概念、数据的逻辑结构和存储结构及操作算法，并能灵活运用；能够利用数据结构和算法的基本知识，为应用问题设计有效的数据结构和算法；能够分析算法的复杂性。

2. 要求能够用 C/C++/Java 等程序设计语言描述数据结构和算法。

1.5 算法描述与算法分析

算法（Algorithm）：是对特定问题求解步骤的一种描述，它是指令（规则）的有限序列，其中每一条指令表示一个或多个操作。

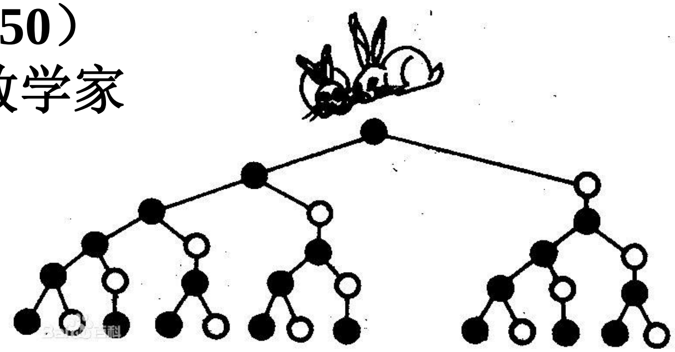
- 算法是在有限步骤内求解某一问题所使用的一组定义明确的规则；
- 通俗点说, 就是计算机解题的过程；
- 在这个过程中，无论是形成解题思路还是编写程序，都是在实施某种算法。前者是推理实现的算法，后者是操作实现的算法。

资料：Algorithm与Logarithm

- 早期的语言学家：algiros(费力的)+arithmos(数字)组合派生而成，但另一些人认为这个词是从“喀斯迪尔国王Algor”派生而来的;
- 数学史学家发现了algorism(算术)一词的真实起源：它来源于著名的*Persian Textbook(《波斯教科书》)*的作者的名字Abu Ja‘far Mohammed ibn Mûsâ al-Khowârizm （约公元前825年）意思是“Ja’far 的父亲，Mohammed 和 Mûsâ 的儿子，Khowârizm 的本地人”。Khowârizm 是前苏联XИBA(基发) 的小城镇 。Al-Khowârizm 写了著名的书*Kitab al jabr w ‘al-muqabala (《复原和化简的规则》)*；另一个词，“algebra”(代数)，是从他的书的标题引出来的;
- 牛津英语字典：这个词是由于同arithmetic(算术)相混淆而形成的错拼词。由algorism又变成algorithm;
- 德文数学词典 *Vollstandiges Mathematisches Lexicon (《数学大全辞典》)*，给出了Algorithmus (算法)一词的如下定义：“在这个名称之下，组合了四种类型的算术计算的概念，即加法、乘法、减法、除法”。拉顶短语*algorithmus infinitesimalis* (无限小方法)，在当时就用来表示Leibnitz(莱布尼兹)所发明的以无限小量进行计算的微积分方法;
- 1950年左右，algorithm一词经常地同欧几里德算法(Euclid’s algorithm)联系在一起。这个算法就是在欧几里德的*《几何原本》*中所阐述的求两个数的最大公约数的过程(即辗转相除法)。

从Fibonacci数列开始……,神奇的数列！

(1175-1250）
意大利数学家



斐波那契在《算盘书》中提出了一个有趣的兔子问题：

一般而言，兔子在出生两个月后，就有繁殖能力，1对兔子每个月能生出1对小兔子来。如果所有兔都不死，那么1年以后可以繁殖多少对兔子？

我们不妨拿新出生的1对小兔子分析一下：
第1个月小兔子没有繁殖能力，所以还是1对；
2个月后，生下1对小兔总数共有2对；
3个月以后，老兔子又生下1对，因为小兔子还没有繁殖能力，所以一共是3对；
……. 12个月以后呢？

经过月数	兔子对数
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55
11	89
12	144

Fibonacci： 0, 1, 1, 2 , 3, 5, 8, 13, 21, 34, 55, 89,144,………

Fibonacci 数列的生成规则： $F_n = \begin{cases} 0 & \text{如果} n=0 \\ 1 & \text{如果} n=1 \\ F_{n-1} + F_{n-2} & \text{如果} n>1 \end{cases}$

$F_n \approx 2^{0.694n}$, Fibonacci数增长的速度几乎与2的幂增长的速度相当!

F_{30}超过了100万， F_{100}已经达到 21 位数字！

F_{200}是多少？ 谁能告诉 Fibonacci ?

```
Long int fib1(int n)
{
  if(n==0) return(0);
  if(n==1) return(1);
  return(fib1(n-1)+fib(n-2));
}
```

fib1(200) 执行的基本操作次数:

$T(200)=T(199)+T(198)+3 >= 2^{138}$

以每秒33.86千万亿次的计算机测算

Fib1(200)的计算时间>=2^{82}秒

$F_n \approx 2^{0.694n} \approx (1.6)^n$, 计算$F_{n+1}$的时间是计算$F_n$的1.6倍，$F_{n+1}=1.6F_n$

摩尔定律(Moore’s low)：计算机的运算速度每年增长约1.6倍。

计算F_{200}的时间> $1.11*2^{56}$ 年

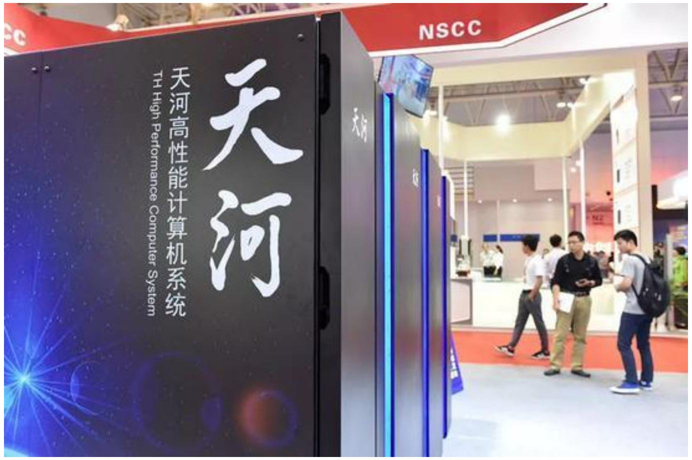
另外一个计算方法：

```
long int  fib2( int n )
{
    long int F[200] , i ;
    if( n==0 ) return(0) ;
    F[0] = 0;   F[1] = 1 ;
    for(i=2; i<=200; i++)
       F[i]=F[i-1]+F[i-2] ;
     return(F[200]);
}
```

Fib2(n) 执行的基本操作次数 T(n) 是关于n的线性函数!

$T(n)=O(f(n))=O(n)$

这仅是以加法为基本操作，你发现新的问题了吗？



5月在天津举办的第二届世界智能大会上，中国国家超算天津中心对外展示了我国新一代百亿亿次超级计算机“天河三号”原型机，有望在2020年研制成功并重回超算榜首。

常见的计算机算法：

递归技术：最常用的算法设计思想，体现于许多优秀算法之中；
分治法：分而制之的算法思想，体现了一分为二的哲学思想 ；
模拟法：用计算机模拟实际场景，经常用于与概率有关的问题 ；
贪心算法：采用贪心策略的算法设计 ；
状态空间搜索法：被称为“万能算法”的算法设计策略 ；
随机算法：利用随机选择自适应地决定优先搜索的方向 ；
动态规划：常用的最优化问题解决方法 。

算法的特征：

①有穷性、②确定性、③输入、④输出、⑤能行性

“好”的算法的标准：

①正确性，算法能满足具体问题的需求；

②可读性，首先方便阅读与交流，其次才是机器执行；

③健壮性，输入错误时，能作出反应，避免异常出错；

④效率与低存储量要求。

算法效率衡量和准则：

①事后统计法

必须执行程序，

可能有其他因素掩盖算法的本质；

②事前分析估算法

对算法“正确性”的要求：

①不含语法错误；

②对于几组输入数据能得到满足要求的结果；

③对精心选择苛刻并带有刁难的数据能得到满足要求的结果；

④对于一切合法的输入均得到满足要求的结果；

算法描述：

①自然语言；②程序设计语言；③类语言*；

关于本书采用的类语言描述：

①结构类型说明；

②输入输出约定(**cin >> v，cout << v**)；

③ **new** 和 **delete**；

④引入引用类型；

⑤其他；

■程序运行时间

影响算法执行的因素：

①算法实现后所消耗的时间**；

②算法实现后所占存储空间的大小*；

③算法是否易读、易移植等等其它问题。

影响时间特性的五个因素：

①算法选用的策略；

②程序运行时输入数据的总量，即问题的规模；

③对源程序编译所需的时间，产生机器代码的质量；

④计算机执行每条指令所需的时间/速度；

⑤程序中指令重复执行的次数*。

【定义】 语句频度：语句重复执行的次数。

渐近时间复杂度（时间复杂度）T(n)

算法中基本操作重复执行的次数是问题规模n的某个频度函数f(n)，算法的时间度量记作：

$$T(n) = O(f(n))$$

它表示随问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同。只与基本运算频度有关，即最深层循环内的语句。

渐近空间复杂度（空间复杂度）S(n)

$$S(n) = O(g(n))$$

运算法则：

设：$T_1(n)=O(f(n))$，$T_2(n)=O(g(n))$

加法规则：$T_1(n)+T_2(n) = O(\max\{f(n), g(n)\})$

乘法规则：$T_1(n) \cdot T_2(n) = O(f(n) \cdot g(n))$

渐近时间复杂度（时间复杂度）T(n)

算法中基本操作重复执行的次数是问题规模n的某个频度函数f(n)，算法的时间度量记作：

$$T(n) = O(f(n))$$

只与基本运算频度有关，即最深层循环内的语句。

例. 在数组A[0,…,n-1]中，查找给定值k的算法大致如下：

```
(1) i = n-1;
(2) while (i>=0 && A[i]!=k)
(3)         i--;
(4) return i;
```

最好时间复杂度？

最坏时间复杂度？ 一般考虑

平均时间复杂度？

O(n)

若存在正的常数c和函数f(n)，使得对任何n >> 2都有：

$$T(n) \le c \cdot f(n)$$

则认为在 n 足够大之后，f(n) 给出了 T(n) 增长速度的一个渐进上界，记为:

$$T(n) = O(f(n))$$

大O记号的性质:

(1)对于任一常数c > 0，有O(f(n)) = O(c·f(n))

在大O记号的意义下，函数各项正的常系数可以忽略并等同于1。

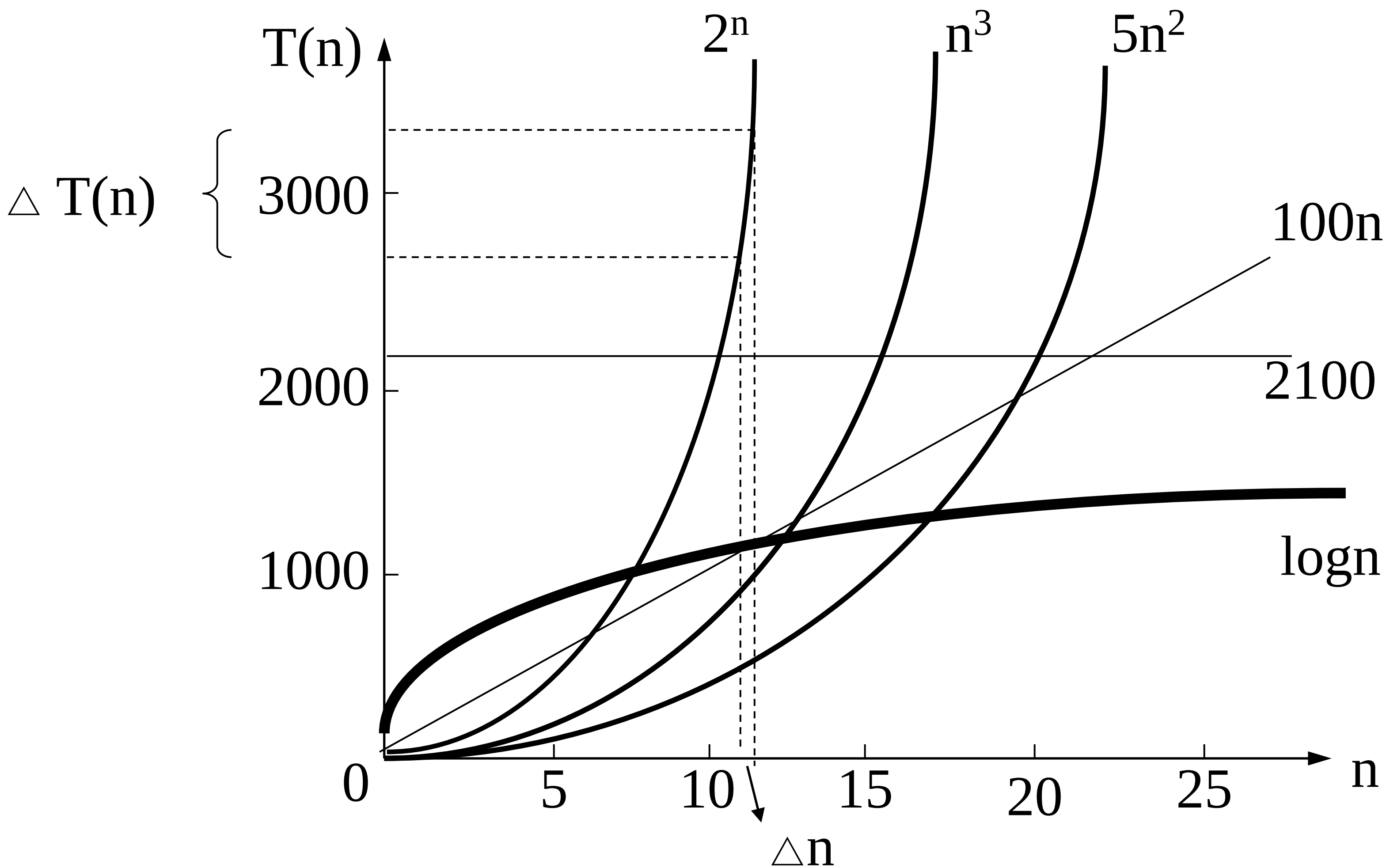
(2) 对于任意常数a > b > 0，有$O(n^a + n^b) = O(n^a)$

多项式中的低次项均可忽略，只需保留最高次项。

O的含义是T(n)的数量级。

- 上述性质体现了对函数总体渐进增长趋势的关注和刻画

$$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$



程序运行时间比较　T(n) = O (f (n))

大Ω记号

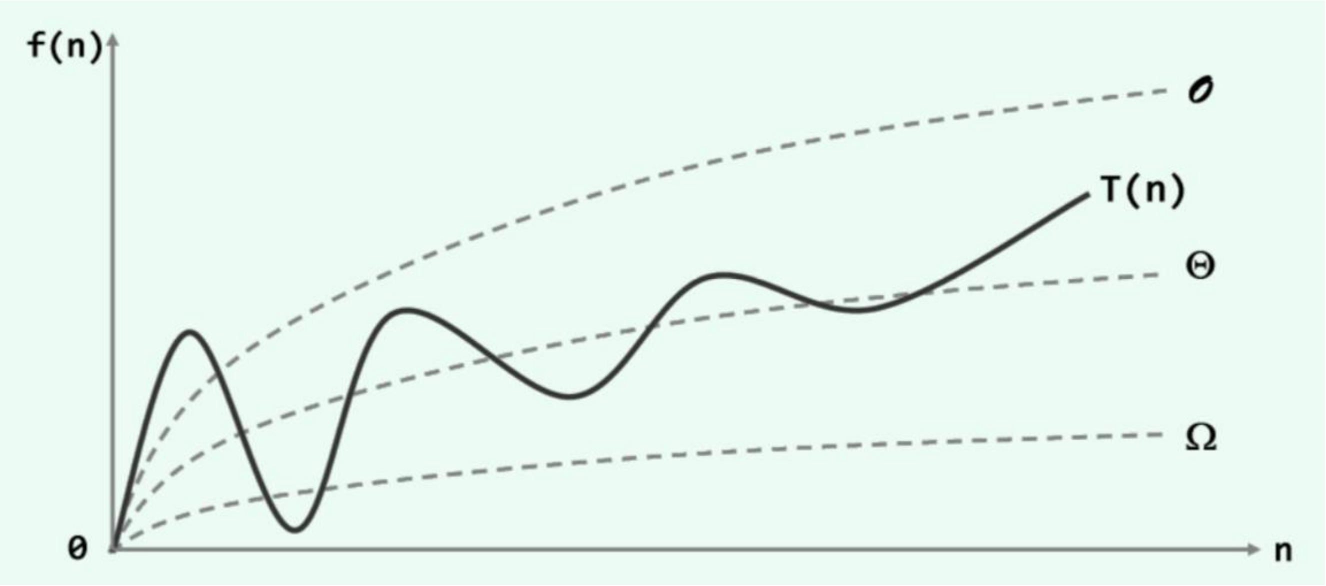
对算法的复杂度最好情况做出估计。

与大O记号相反，大Ω记号是对算法执行效率的**乐观估计**，对于规模为n的任意输入，算法的运行时间都不低于Ω(g(n))。

大Θ记号

对算法复杂度的**准确估计**。

对于规模为n的任何输入，算法的运行时间T(n)都与Θ(h(n))同阶。



空间复杂度

空间复杂度S(n)定义为该算法所消耗的存储空间，它是问题规模n的函数，渐进空间复杂度也常称为空间复杂度，记为S(n)=O(g(n))。

一个程序所需存储空间一般包括：

- **输入数据所占空间(指令、常数、变量等)**
- **存放对数据进行操作的工作单元(程序本身所占空间)**
- **辅助变量所占空间**

如果输入数据所需空间只取决于问题本身与算法无关，则只需分析除输入和程序外的**额外空间**。

算法原地工作是指算法所需的辅助空间为常数，即O(1)。

【例1-3】

①s = 0 ;

→ $f(n) = 1$; $T_1(n) = O(f(n)) = O(1)$ *常量阶*

②for (i=1 ; i <= n ; ++i) { ++x; s += x; }

→ $f(n) = 3n+1$; $T_2(n) = O(f(n)) = O(n)$ *线性阶*

③for (i=1; i<=n ; ++i)

  for(j=1 ; j <=n ; ++j) { ++x ; s += x; }

→ $f(n) = 3n^2+2n+1$; $T_3(n) = O(f(n)) = O(n^2)$ *平方阶*

④for (i=1; i<=n ; ++i)

  for (j=1 ; j <=n ; ++j)

  { c[i][j] = 0;

    for (k=1 ; k <= n; ++k)

      c[i][j] += a[i][k] * b[k][j] ; }

→ $f(n) = 2n^3+3n^2+2n+1$; $T_4(n) = O(f(n)) = O(n^3)$ *立方阶*

【例1-4】

```
Long  fact ( int  n)
  {   if ( n==0 ) || ( n ==1 )
            return( 1 );
        else
            return( n * fact( n – 1 ) );
  }
```

for(p=1,i=2;i<=n;p=p*i++) ;

T(n)=O(n).

$$f(n) = \begin{cases} C & \text{当 } n=0,\ n=1 \\ G + f(n-1) & \text{当 } n>1 \end{cases}$$

$$\begin{aligned}
f(n) &= G_1 + f(n-1) \\
f(n-1) &= G_2 + f(n-2) \\
f(n-2) &= G_3 + f(n-3) \\
&\cdots\ \cdots \\
f(2) &= G_{n-1} + f(1) \\
+\ \ f(1) &= C \\
\hline
f(n) &= G_1 + G_2 + G_3 + \ldots\ldots + G_{n-1} + C
\end{aligned}$$

$\Longrightarrow$

$$f(n) = n\,G'$$

$$\therefore T(n) = O(f(n)) = O(n)$$

【总结分析方法一】

循环主体中的变量参与循环条件的判断

此类题目应该找出主体语句中与T(n)成正比的循环变量，将之代入条件中计算。

例：以下算法的时间复杂度是（） 设共执行t次

```
void func(){int i=0; while(i*i*i<=n) i++;}
```

$t*t*t <= n$

A. O($\log_2 n$)　B. O($n^{1/2}$)　C. O($n^{1/3}$)　D. O($n\log_2 n$)

$=> t <= n^{1/3}$

```
void func(){int i=1; while(i<=n) i=i*2;}
```

$2^{t+1} <= n/2$

A. O($\log_2 n$)　B. O($n^{1/2}$)　C. O($n^{1/3}$)　D. O($n\log_2 n$)

$=> t <= \log_2 n - 2$

```
void func(){int j=5; while((j+1)*(j+1)<n) j=j+1;}
```

$(t+5+1)^2 < n$

A. O($\log_2 n$)　B. O($n^{1/2}$)　C. O(n)　D. O($n\log_2 n$)

$=> t < n^{1/2} - 6$

```
int i=0; k=0;
while(i<n-1)
    k=k+10*i;
    i++;
```

A. O(logn)　B. O(n)　C. O($n^{1/2}$)　D. O(n^2)

```
int i=0; k=0;
while(k<n-1)
    k=k+10*i;
    i++;
```

$$\sum_{i=1}^{t-1} 10i = 10\sum_{i=1}^{t-1} i < n-1$$

【总结分析方法二】

循环主体中的变量与循环条件是无关的

此类题目可以采用数学归纳法或直接累计循环次数。多层循环时从内到外分析，忽略单步语句、条件判断语句，只关注主体语句的执行次数。

（1）递归程序一般使用公式进行推导。

```
int fact(int n){
if(n<=1) return 1;
return n=n*fact(n-1);}
```

求阶乘

即n*(n-1)*…*1

执行n次

（2）非递归比较简单，直接累计次数。

```
for(i=n-1;i>1;i--)
  for(j=1;j<i;j++)
  if(A[j]>A[j+1]) A[j]与A[j+1]互换;
```

$$\sum_{i=2}^{n-1}\sum_{j=1}^{i-1}1=\sum_{i=2}^{n-1}(i-1)=(n-2)(n-1)/2$$

```
in m=0,i,j;
for(i=1;i<=n;i++)
  for(j=1;j<=2*i;j++) m++;
```

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

A. O(n) B. O($n\log_2 n$) C. O($\log_2 n$) D. O(n^2)

【例1-5】

（1）某算法的时间复杂度为O(n^2)，表明该算法（C）

A. 问题规模是n^2　　B. 执行时间是n^2

C. 执行时间和n^2成正比　D. 问题规模与n^2成正比

（2）n为非负整数，以下算法时间复杂度是（ A ）

```
x=2;
while(x < n/2)
x=2*x;
```

A. O($\log_2 n$)　B. O(n)

C. O($n\log_2 n$)　D. O(n^2)

设共执行t次

$2^{t+1} < n/2$

=>

t<log(n)-2

（3）已知两个长度分别为m和n的升序链表，若将它们合并为一个长度m+n的降序链表，则最坏情况下的时间复杂度是（D）

A. O(n)　B. O(mn)　C. O(min(m, n))　D. O(max(m, n))　）

最坏情况是
两个链表元素依次进行比较

【例1-6】考研题

（1）以下算法时间复杂度是（ ）

```
count=0;
for(k=1;k<=n;k*=2)
   for(j=1;j<=n;j++)
      count++;
```

A. $O(\log_2 n)$　　B. $O(n)$

C. $O(n\log_2 n)$　　D. $O(n^2)$

内外循环无关、与基本运算也无关
内循环: j自增1，执行n次，则O(n);
外循环: $2^t<=n$，则$O(\log_2 n)$;
根据嵌套循环、乘法规则:
$O(n \log_2 n)$;

（2）下列算法时间复杂度是（ ）

```
int func(int n){
    int i=0,sum=0;
    while(sum<=n) sum += ++i;
     return i;}
```

A. $O(\log_2 n)$　　B. $O(n^{1/2})$

C. $O(n)$　　D. $O(n\log_2 n)$

仔细分析发现，相当于累加。
sum= (1+t)*t/2 < n
显然选 B

【思考题】

一个算法所需时间由如下递归方程表示，试求该算法的时间复杂度的级别（或阶），给出基本的计算步骤。

$$T(n)=\begin{cases}1, & n=1\\ 2T(n/2)+n, & n>1\end{cases}$$

等式中，n是问题的规模，设n是2的整数次幂。

数据结构基本思路

- ADT抽象数据型
  - 数学模型
  - 操作 → 解决问题的算法
- 逻辑结构
- 存储结构 → 型
- 特殊结构
- 结构应用举例

三大数据结构类型

• 线性表	线性结构
• 树及二叉树	层次结构
• 图	网状结构

数据结构
- 线性结构：线性表
- 非线性结构
  - 树
  - 图

数据结构附加内容

- 查找（检索）
- 排序（分类）
- 文件

The End