《数据结构》

（计科、电信专业）

计算机的应用：

数值计算：如进制转换、求圆面积、体积、求解一元二次方程的解等

——只要是人脑能解决的，就能通过编程得出和人相同的结果。

非数值计算：控制、管理、数据处理等方面

——相对来说比较复杂：历史、现状、未来；直觉、灵感、发散性思维

《数据结构》：

研究数据的逻辑结构、物理结构及其操作的学科。

程序＝算法＋数据结构＋语言＋程序设计方法

算法是灵魂，数据结构是加工对象，语言是工具，编程需采用合适的方法。

而算法在很大程度上受到加工对象即数据结构的限制，甚至在某些情况下数据结构起决定性的作用。

第一章　　绪论

1．1 什么是数据结构

1．2 基本概念和术语

1．3 抽象数据类型的表示与实现

1．4 算法和算法分析

1．4．1 算法

1．4．2 算法设计的要求

1．4．3 算法效率的度量

1．4．4 算法的存储空间需求

一、教学目的与要求

了解数据结构的基本概念，认识算法和算法分析；初步学会对空间复杂度和时间复杂度进行估算

二、主要教学内容

数据结构的基本概念和相关术语；抽象数据类型的表示与实现；算法和算法分析；算法效率的度量；算法的存储空间的需求

三、教学重点、难点

数据结构、算法和算法分析、空间复杂度、时间复杂度

四、授课方法及手段

采用多媒体大屏幕投影授课

五、讲课具体内容（讲稿）

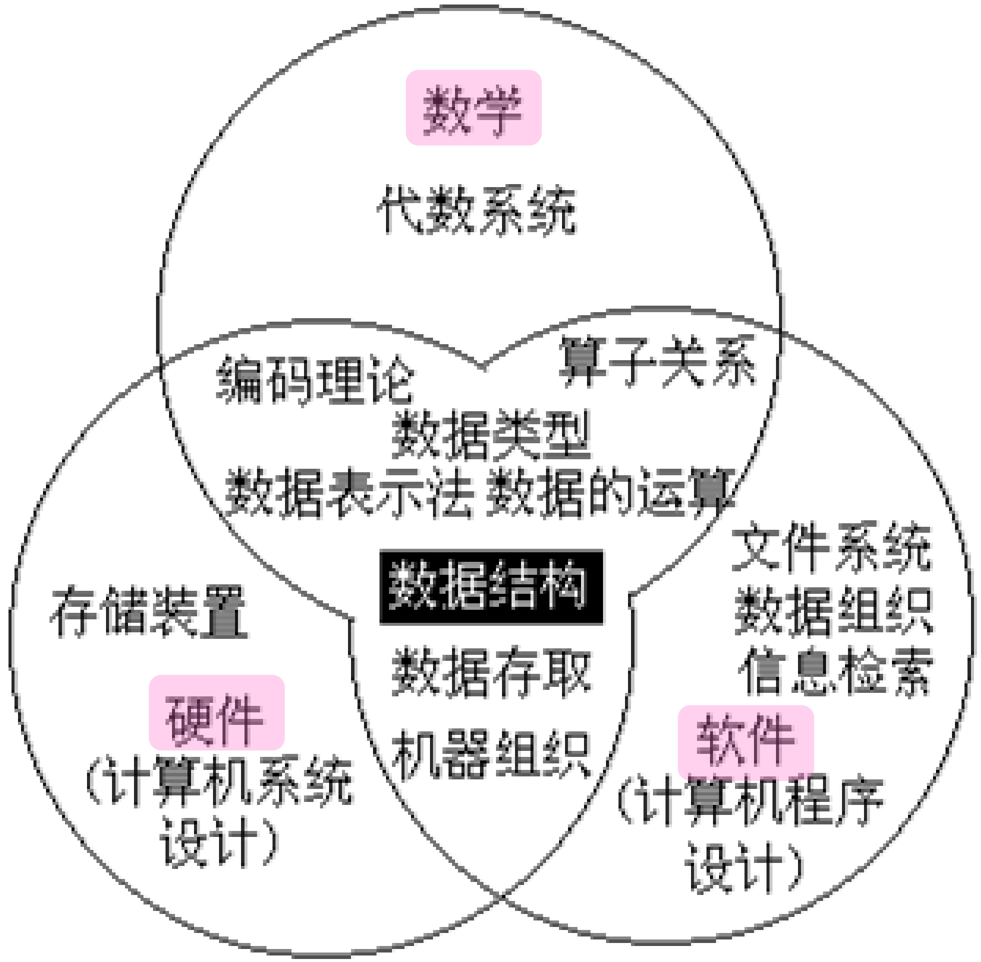
什么是数据结构

- 计算机解决问题的步骤

实际问题 —工程师→ 数学模型 —数学家→ 算法 —程序员→ 程序 → 结果

数据结构——研究计算机的操作对象（数据）以及它们之间的关系和操作等的学科。

《数据结构》的地位——综合性的专业基础课



基本概念和术语

- **数据**：计算机程序处理的符号的总称。
- **数据元素**：数据的基本单位。
  - 通常作为一个整体进行处理。
- **数据项**：数据的不可分割的最小单位。
  - 一个数据元素可以由若干个数据项构成。
- **数据对象**：性质相同的数据元素的集合。
- **数据结构**：相互间存在一种或多种关系的数据元素的集合。

例:图书信息表

数据对象

登录号	书名	作者	出版社	定价（元）
00001	C++语言基础教程	徐孝凯	清华大学	26.00
00002	计算机辅助制造	李德庆	机械工业	3.30
00003	计算机系统原理	张基温	电子工业	25.00
00004	数据结构	严蔚敏	清华大学	28.00

数据元素

数据项

数据的结构

- **逻辑结构**：数据元素之间的逻辑关系
- **物理结构**：数据结构在计算机中的表示，又称**存储结构**

算法的设计取决于选定的逻辑结构

算法的实现依赖于采用的存储结构

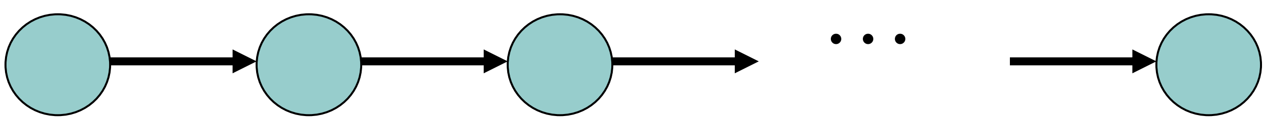
逻辑结构：

(1) 线性结构

结点之间关系：一对一

特点：开始结点和终端结点都是惟一的，除了开始结点和终端结点以外，其余结点都有且仅有一个前驱结点，有且仅有一个后继结点。

“队列”就是典型的线性结构。

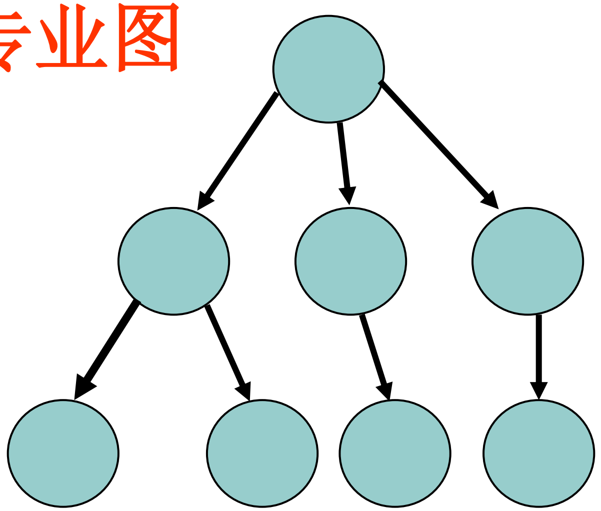


(2) 树形结构

结点之间关系：一对多。

特点：开始结点惟一，终端结点不惟一。除终端结点以外，每个结点有一个或多个后续结点；除开始结点外，每个结点有且仅有一个前驱结点。

如：西南林业大学学科专业图

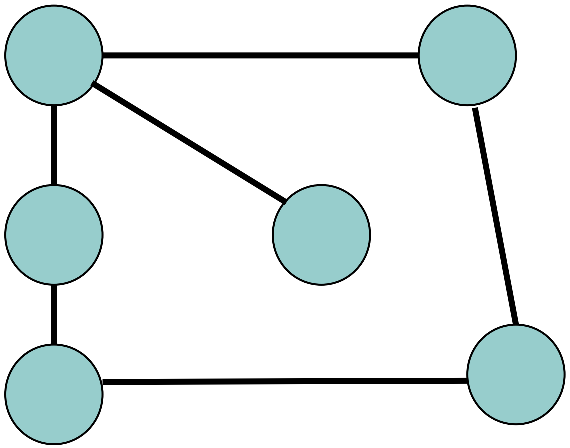


(3) 图形结构

结点之间关系：多对多。

特点：没有开始结点和终端结点，所有结点都可能有多个前驱结点和多个后继结点。

如：昆明市公交车站点图



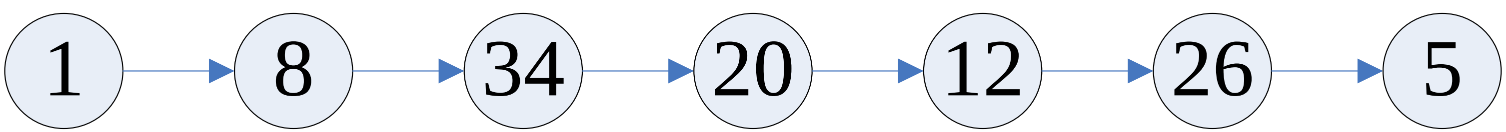
逻辑结构的表示

例1：有一种数据结构B1=(D,S)，其中，

D={1,5,8,12,20,26,34}，S={s}，

s={<1,8>,<8,34>,<34,20>,<20,12>,

<12,26>,<26,5>}，画出其逻辑结构表示



逻辑结构的表示

例2：有一种数据结构B2=(D,S)，

其中，

D={a,b,c,d,e,f,g,h,i,j}

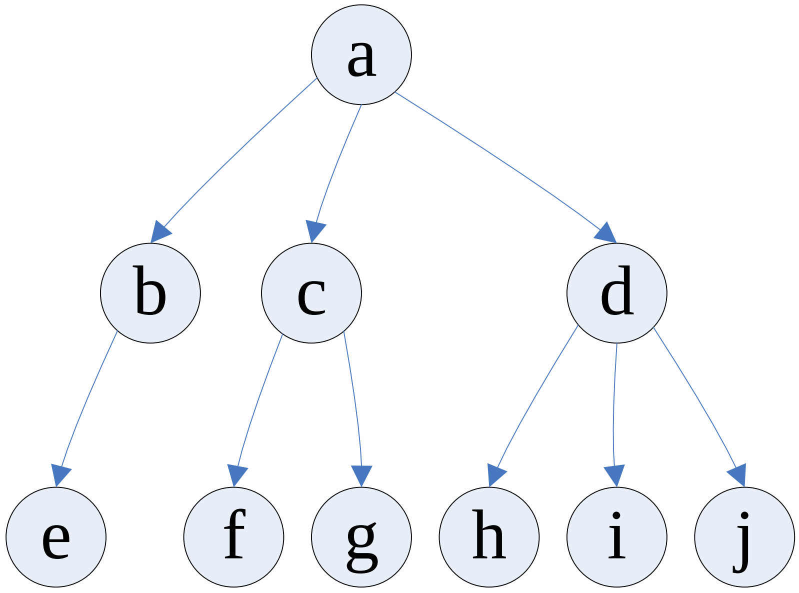
S={s}

s={<a,b>,<a,c>, <a,d>,

<b,e>, <c,f>,<c,g>,

<d,h>, <d,i>, <d,j>},

画出其逻辑结构表示。



本书结构—数据结构部分

（1）集合（离散点） ——数据结构中不讨论

（2）线性结构（一对一）

- 第二章 线性表
- 第三章 栈和队列
- 第四章 串
- 第五章 数组和广义表

（3）树形结构（一对多）：第六章 树和二叉树

（4）图状结构或网状结构（多对多）：第七章 图

物理结构

（1）顺序存储结构：所有存储结点相继存放在一个连续的存储区中。用存储结点间的位置关系表示数据元素之间的逻辑关系。

（2）链式存储结构：通过在结点上附加一个指针域来表示结点间的逻辑关系，每个指针指向一个与本结点有逻辑关系的结点。

（3）索引存储结构

（4）散列存储结构

顺序存储：

存储地址	M
1001	k_1
1002	k_2
1003	k_3
1004	k_4
1005	k_5
1006	k_6
1007	k_7
1008	k_8
1009	k_9

链式存储：

存储地址	info	next
1000		
1001	k_1	1003
1002		
1003	k_2	1007
1004		
1005	k_4	1006
1006	k_5	$\wedge$
1007	k_3	1005
1008		

算法和算法分析

- 算法（Algorithm）:对特定问题求解步骤的描述.
- 算法的五个重要特性:

  （1）有穷性：算法必须在执行有穷步之后结束，每一步都可在有穷时间内完成。

  （2）确定性：对相同的输入只能得出相同的输出

  （3）可行性：算法所描述的操作都是可实现的

  （4）输入：0个或多个输入

  （5）输出：1个或多个输出

算法描述

- 用文字描述
- 用流程图描述
- 用一种程序设计语言描述
- ……

算法描述

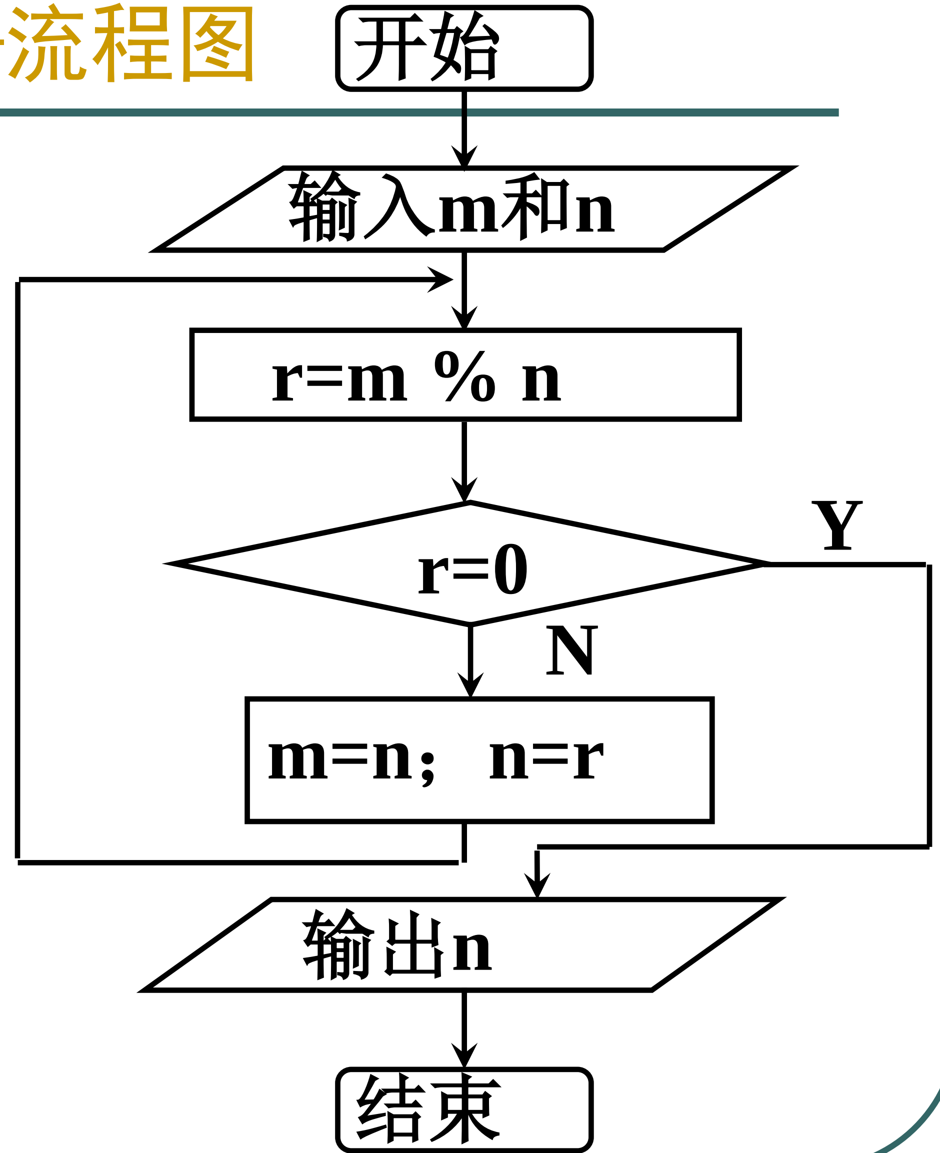
- 例：欧几里德算法——辗转相除法求两个自然数 m 和 n 的最大公约数

算法描述（一）——自然语言

步骤1：将m除以n得到余数r；
步骤2：若r等于0，则n为最大公约数，
　　　　算法结束；否则执行步骤3；
步骤3：将n的值放在m中，将r的值放在n中，
　　　　重新执行步骤1；

- 优点：容易理解
- 缺点：冗长、二义性
- 使用方法：粗线条描述算法思想
- 注意事项：避免写成自然段

算法描述（二）——流程图



- 优点：流程直观
- 缺点：缺少严密性、灵活性
- 使用方法：描述简单算法
- 注意事项：注意抽象层次

算法描述（三）——程序设计语言

```
#include <iostream.h>
int CommonFactor(int m, int n)
{
    int r = m % n;
    while (r != 0)
    {
      m = n;
      n = r;
      r = m % n;
    }
    return n;
}
void main( )
{
   cout<<CommonFactor(63, 54)<<endl;
}
```

- 优点：能由计算机执行
- 缺点：抽象性差，对语言要求高
- 使用方法：算法需要验证
- 注意事项：将算法写成子函数

算法描述（四）——伪代码

伪代码（**Pseudocode**）：介于自然语言和程序设计语言之间的方法，它采用某一程序设计语言的基本语法，操作指令可以结合自然语言来设计。

优点：表达能力强，抽象性强，容易理解

使用方法：7 ± 2

算法描述（四）——伪代码

1. r = m % n;
2. 循环直到 r 等于0
   - 2.1 m = n;
   - 2.2 n = r;
   - 2.3 r = m % n;
3. 输出 n ;

算法描述（四）——类C伪代码

```
int CommonFactor(int m, int n)
{
    r = m % n;
    while (r != 0)
    {
        m = n;
        n = r;
        r = m % n;
    }
    return n;
}
```

对C++语言进行了如下简化：
(1) 局部变量可以不声明；
(2) 写出子函数即可，子函数不用在主函数中调用，省略主函数；
(3) 所有的包含函数（头函数.h）可以省略；
(4) 交换两个变量的语句可以简写为a←→b。

数据类型与抽象数据类型

- 数据类型（Data Type）：
  - 值的集合以及定义在这个集合上的一组操作。
  - 例如: C语言中的整数类型以及字符类型
- 抽象数据类型(ADT)
  - 数学模型以及定义在该模型上的一组操作。
  - 与其在计算机中的表示和实现无关。
  - ADT可用三元组表示：（D,S,P）

  D(Data) – 数据对象;

  S(Structure) – D上的关系;

  P(Process) – 对D的基本操作集

抽象数据类型的定义格式

- **ADT 抽象数据类型名{**

  数据对象：<数据对象的定义>

  数据关系：<数据关系的定义>

  基本操作：<基本操作的定义>

  } ADT 抽象的数据类型名

- 基本操作的定义格式为：

  基本操作名（参数表）

  初始条件：<初始条件描述>

  操作结果：<操作结果描述>

抽象数据类型三元组的定义举例

- **ADT Triplet{**
  - 数据对象：**D = {e1,e2,e3 | e1,e2,e3**属于Elemset(定义了关系的某个集合)}
  - 数据关系：**R1 = {<e1,e2>|<e2,e3>}**
  - 基本操作：
    - **InitTriplet(&T,v1,v2,v3)**

      初始条件：无

      操作结果: 构造三元组T,元素e1,e2和e3分别被赋予参数v1,v2和v3的值。

抽象数据类型三元组的定义举例

- **DestroyTriplet(&T)**

  初始条件: 三元组T已经存在。

  操作结果: 销毁三元组T。

- **Get(T,i,&e)**

  初始条件: 三元组T已经存在,1<=i<=3。

  操作结果: 用e返回三元组T的第i个元素。

- **Put(&T,i,e)**

  初始条件: 三元组T已经存在,1<=i<=3。

  操作结果: 用e值取代三元组T的第i个元素。

抽象数据类型三元组的定义举例

- IsAscending(T)

  初始条件: 三元组T已经存在。

  操作结果: 如果三元组T的三个元素按升序排列, 则返回TRUE; 否则返回FALSE。

- IsDescending(T)

  初始条件: 三元组T已经存在。

  操作结果: 如果三元组T的三个元素按降序排列, 则返回TRUE; 否则返回FALSE。

抽象数据类型三元组的定义举例

- Max(T,&e)
  初始条件: 三元组T已经存在。
  操作结果: 用e返回三元组T的最大值。

- Min(T,&e)
  初始条件: 三元组T已经存在。
  操作结果: 用e返回三元组T的最小值。

} ADT Triplet

抽象数据类型的表示与实现

- 类C语言（作了扩充和修改）的表示
- 如：预定义常量和类型
-
```
# define TRUE           1
# define FALSE          0
# define OK             1
# define ERROR          0
# define INFEASIBLE     -1
# define OVERFLOW       -2
typedef int Status
```
- 其它：P10—11

三元组基本操作实现——举例

```
Status Get(Triple T, int i, Elemtype *e)
// 初始条件: 三元组T已经存在。
// 操作结果: 用e返回三元组T的第i个元素。
{
    if (i<1 || i>3) return ERROR;
    *e=T[i-1];
    return OK;
}
```

算法评价

- 算法评价的**目的**：
  - 从解决问题的不同算法中选择出较为合适的一种；
  - 对现有算法进行改进，从而设计出更好的算法

算法应达到的目标

（1）正确性

层次a：程序不含语法错误；

层次b：程序对于几组输入数据能得出满足要求的结果；

层次c：程序对于精心选择的典型、苛刻的几组输入数据能够得出满足规格说明要求的结果；

层次d：程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

（2）可读性

（3）健壮性

（4）高效率与低存贮量

算法效率的度量

（1）事后统计法

例：algo1-1、algo1-2

缺点：必须先运行依据算法编制的程序；所得时间的统计量依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。

algo1-1.cpp：计算1-1/x+1/x*x…

```
#include<stdio.h>
 #include<sys/timeb.h>
 void main()
 {
   struct timeb t1,t2;
   long t;
   double x,sum=1,sum1;
   int i,j,n;

   printf("请输入x n：");
   scanf("%lf%d",&x,&n);
   ftime(&t1);          /* 求得当前时间 */
```

algo1-1.cpp：

```
for(i=1;i<=n;i++)
  {
    sum1=1;
    for(j=1;j<=i;j++)
        sum1=-sum1/x;
    sum+=sum1;
  }
ftime(&t2);              //求得当前时间
t=(t2.time-t1.time)*1000+ (t2.millitm-t1.millitm);
                         // 计算时间差
printf("sum=%lf 用时%ld毫秒\n",sum,t);
}
```

algo1-2.cpp（改进算法）：

将前一程序的黄色部分修改为：

```
sum1=1;
for(i=1;i<=n;i++)
  {
    sum1=-sum1/x;
    sum+=sum1;
  }
```

- 运行以上两个程序，比较当n增长时，两个程序的运行时间的差别。

算法效率的度量

（2）事前分析估算法

通常把算法中包含简单操作次数的多少叫做**算法的时间复杂度**，用它来衡量一个算法的**运行时间性能**或称**计算性能**。

它是问题规模n的某个函数f(n):

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

算法评价

【例】分析以下程序段的时间复杂度

```
for  (i=0; i<n; i++)             //①
   {  y=y+1;                      //②
      for  (j=0; j<=2*n; j++)     //③
           x++;                   //④
   }
```

n+1

n

n*(2n+2)

n*(2n+1)

$$T(n)=(n+1)+n+(2n^2+2n)+(2n^2+n)=4n^2+5n+1$$

算法评价

- 在难以精确计算基本操作执行次数时, 求时间复杂度仅考虑对于问题规模的**增长率(或阶)**即可.
- 例： for (i=2; i<=n;++i)

```
for (i=2; i<=n;++i)
    for (j=2; j<=i-1;++j)
        {  ++x; a[i, j] = x;  }
```

语句频度为： $0+1+\ldots+(n-3)+(n-2) = (n-1)(n-2)/2$

阶为： $O(n^2)$

算法评价

常涉及的增长率（阶）

- $O(1)$——常量阶　　$O(n)$ ——线性阶
- $O(n^2)$ ——平方阶　　$O(n^k)$——多项式阶
- $O(\log n)$ ——对数阶　　$O(2^n)$ ——指数阶

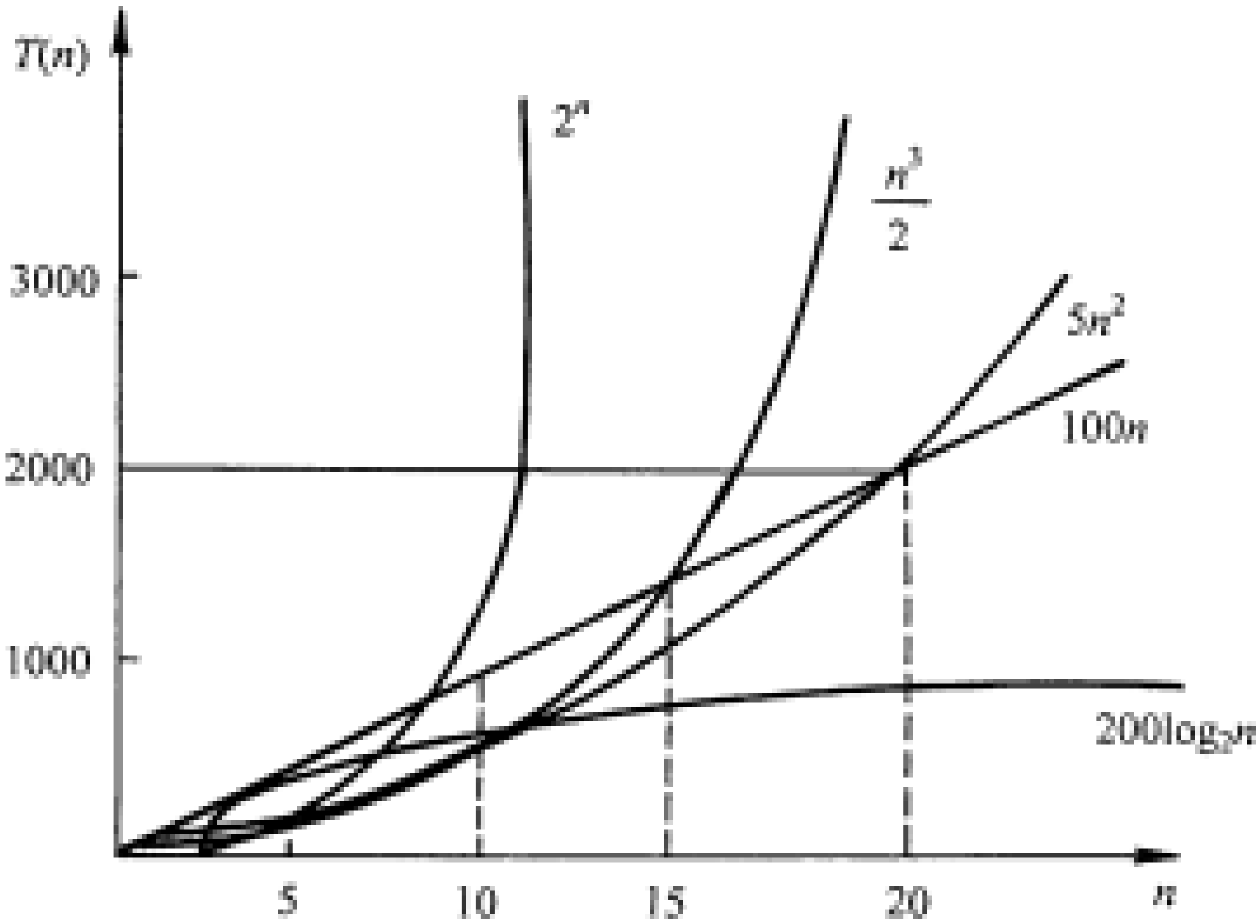
注意：

常 量 阶：$O(1) = O(10)$

多项式阶：$(2n^3+3n^2+4n+5= O(n^3)$

应当尽量选择多项式阶$O(n^k)$的算法

算法评价



常见函数的增长率

算法评价

【例】分析以下程序段的时间复杂度

```
i=1;
while (i<=n)        i=i*2;
```

上述算法中基本操作是语句：i=i*2

设其频度为T(n)，则有：$2^{T(n)}<=n$

即：$T(n)<=\log_2 n=O(\log_2 n)$

算法评价

【例】分析以下程序段的时间复杂度

```
s=0;
for  (i=0; i<=n; i++)
     for  (j=0; j<=i ;j++)
          for  (k=0; k<j; k++)
               s++;
```

算法评价

上述算法中基本操作是语句：**s++,**其频度为:

$$T(n)=\sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=0}^{j-1}1=\sum_{i=0}^{n}\sum_{j=0}^{i}(j-1-0+1)=\sum_{i=0}^{n}\sum_{j=0}^{i}j$$

$$=\sum_{i=0}^{n}\frac{i(i+1)}{2}=\frac{1}{2}\left(\sum_{i=0}^{n}i^{2}+\sum_{i=0}^{n}i\right)$$

$$=\frac{1}{2}\left(\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right)$$

$$=\frac{2n^{3}+6n^{2}+4n}{12}=0(n^{3})$$

算法评价

例：冒泡排序算法

```
Void bubble_sort(int a[], int n){
  for (i=n-1,change = TRUE;  i>1&&change;  --i)  {
      change = false;
      for (j= 0; j<i; ++j)
            if(a[j] > a[j+1])  { a[j] ←→a[j+1];
                                  change=TURE;}
  }
}//bubble_sort
```

时间复杂度与输入数据有关时采用
平均时间复杂度或最坏时间复杂度

算法的存储空间需求

算法的存储量：包括输入数据所占空间、程序本身所占空间和辅助变量所占空间。

空间复杂度：通常指辅助变量所占空间，是对一个算法在运行过程中临时占用的存储空间大小的量度。