算法与数据结构

华东理工大学 叶琪

一

算法与数据结构

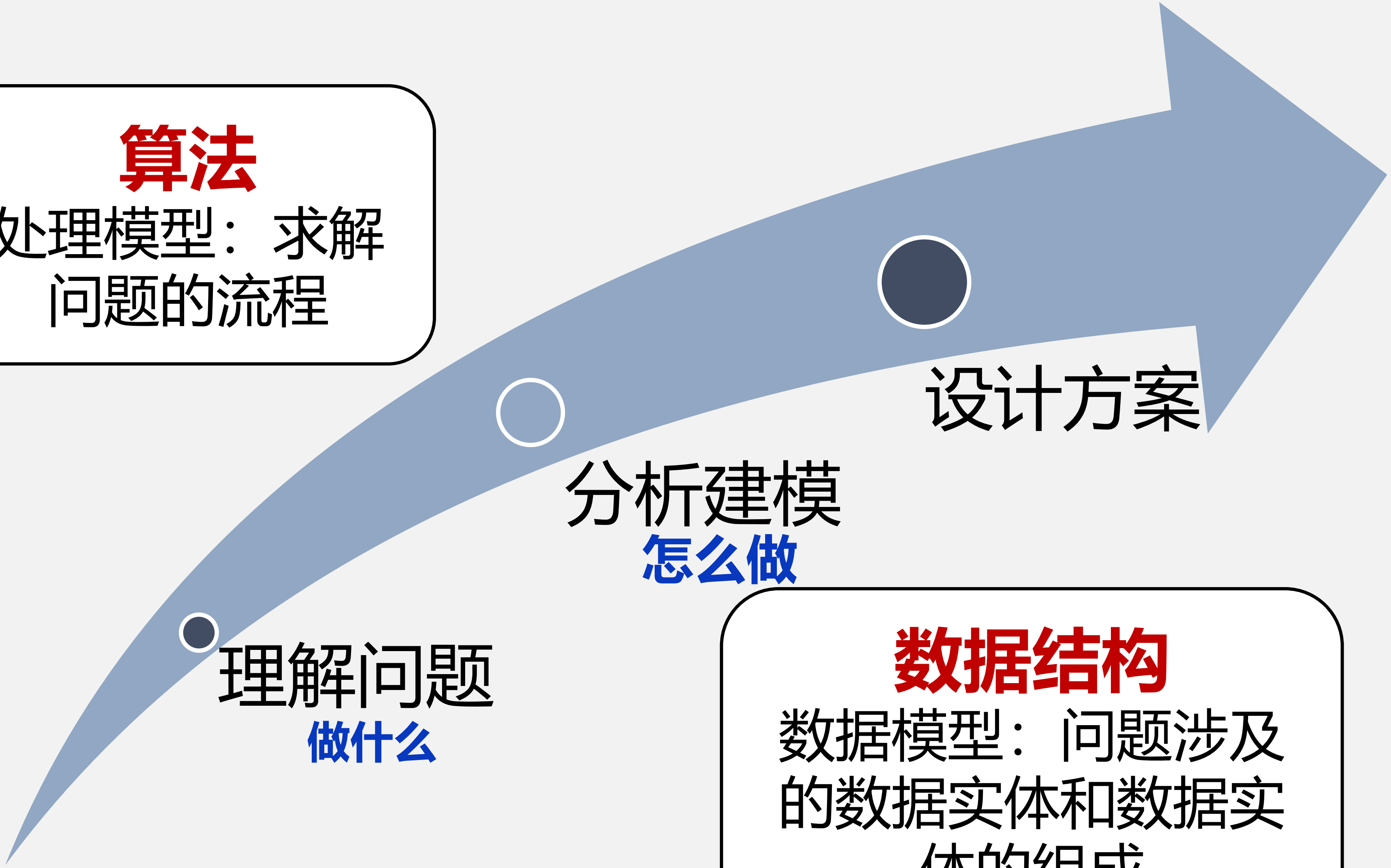
第一章 概论

CONTENTS

目录

1.1引子

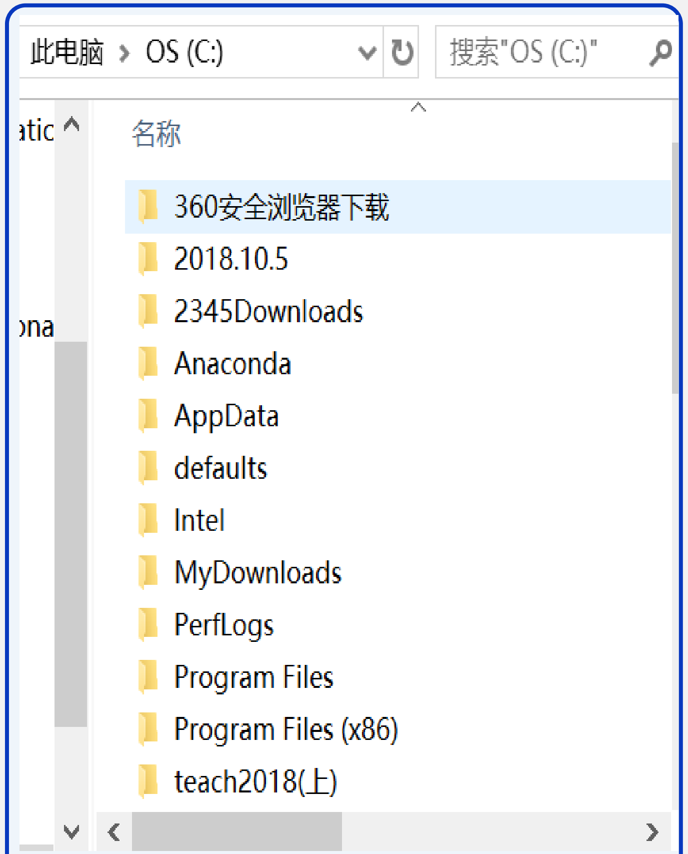
算法
处理模型：求解问题的流程



数据结构
数据模型：问题涉及的数据实体和数据实体的组成

解决哪些具体问题?

火车按时间排序



查找C:中的文件

选合理线路

解决问题的**效率**受到哪些因素影响?

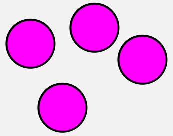
数据组织

时、空利用率

算法

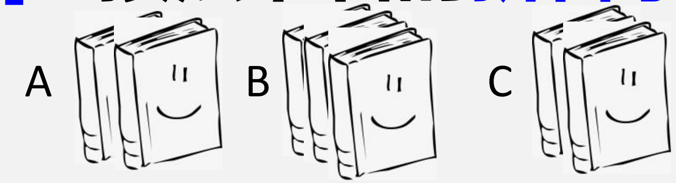
例1：如何在书架上摆放图书？

[方法1]　随便放——任何时候有新书进来，哪里有空就把书插到哪里。

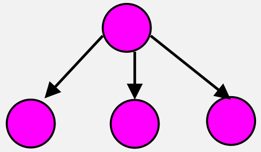
放书方便，但查找效率极低！

[方法2]　按照书名的拼音字母顺序排放。



查找方便，但插入新书很困难！

[方法3]　把书架划分成几块区域，每块区域指定摆放某种类别的图书；在每种类别内，按照书名的拼音字母顺序排放。



查找、插入方便，但每类书数量无法预知，有可能造成空间的浪费！

解决问题方法的效率，跟数据的组织方式有关。

例2：写程序实现一个函数PrintN，使得传入一个正整数为N的参数后，能顺序打印从1到N的全部正整数

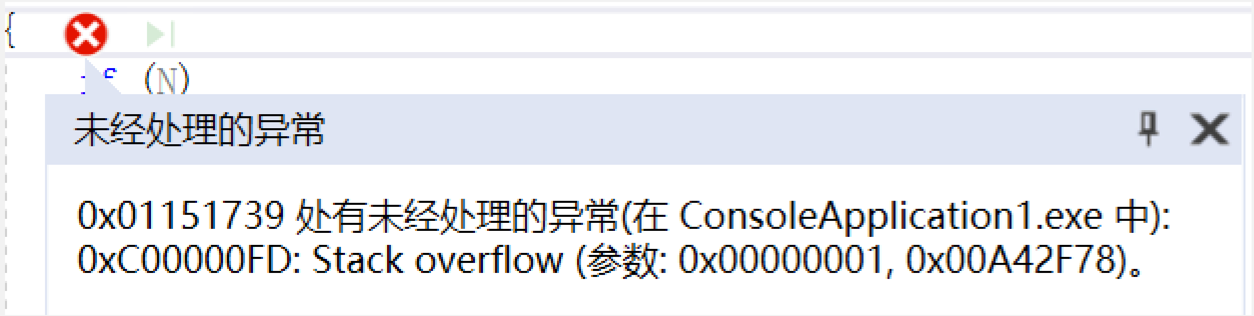
循环实现

```c
void PrintN(int N)
{
    for(int i = 1;i < N;i++)
        printf("%d\n",i);
    return;
}
```

递归实现

```c
void PrintN(int N)
{
    if(N)
    {
        PrintN(N - 1);
        printf("%d\n",N);
    }
    return;
}
```

考虑当N足够大时会出现什么问题！



解决问题方法的效率，跟空间的利用效率有关。

例3：写程序计算给定多项式在给定点x处的值。

方法一

- $f(x) = a_0 + a_1*x + a_2*x\wedge 2 + \ldots\ldots + a_{n-1}*x\wedge(n-1) + a_n*x\wedge n$

```
double f1(int n,double a[],double x)
{    int i;
     double p = a[0];
     for (i = 1; i <= n; i++)
        p = p + a[i] * pow(x,i);
     return p;
}
```

方法二

- $f(X) = a_0 + x*(a_1 + x*(\ldots+x*(a_{n-1} + x*(a_n))\ldots$秦九韶算法

```
double f2(int n, double a[], double x)
{   int i;
    double p = a[n];
    for (i = n; i > 0; i--)
       p = a[i - 1] + x * p;
    return p;
}
```

计算时间：C语言提供了clock()：捕捉从程序开始运行到clock()被调用时所耗费的时间。

- 时间单位是clock tick，即“时钟打点”。
- 常数CLK_TCK：机器时钟每秒所走的时钟打点数。

```
#include <stdio.h>
#include <time.h>/<ctime.h>

clock_t  start, stop; /* clock_t是clock()函数返回的变量类型 */
double  duration;  /* 记录被测函数运行时间，以秒为单位 */

int main ()
{ /* 不在测试范围内的准备工作写在clock()调用之前，例如预处理*/

  start = clock(); /* 开始计时 */
  MyFunction();       /* 把被测函数加在这里 */
  stop = clock();     /* 停止计时 */
  duration = ((double)(stop - start))/CLK_TCK;/* 计算运行时间 */

  /* 其他不在测试范围的处理写在后面，例如输出duration的值 */

  return 0;
}
```

【测试】一个100000阶多项式，使用不同方法计算*f(1.0)*，比较运行时间.

【分析】考虑计算三次多项式 $ax^3 + bx^2 + cx + d$

方法1： $s = a \times x \times x \times x + b \times x \times x + c \times x + d$

(6次乘法，3次加法) 计算量大

方法2： $s = a$； $s = s \times x + b$； $s = s \times x + c$； $s = s \times x + d$；

(3次乘法，3次加法) 计算量小

方法2 的原理：

$$
\begin{aligned}
& a \times x \times x \times x + b \times x \times x + c \times x + d \\
&= (a\,x + b)\,x^2 + c\,x + d \\
&= ((a\,x + b)\,x + c)\,x + d
\end{aligned}
$$

解决问题方法的效率，跟算法的巧妙程度有关。

– 解决一个非常简单的问题，往往也有多种方法，且不同方法之间的效率可能相差甚远

– 解决问题方法的效率

- 跟数据的组织方式有关（如例1）
- 跟空间的利用效率有关（如例2）
- 跟算法的巧妙程度有关（如例3）

数据结构讨论的范畴

计算机应用

数值计算：加工处理的对象——纯粹的数值。

非数值计算

工业检测
过程控制
管理系统
文字处理
……

加工处理的对象

字符
表格
图象
声音
……

占据了当今计算机应用的绝大多数。

具有一定的结构

研究对象的特性及其相互之间的关系

有效地组织计算机存储

有效地实现对象之间的"运算"关系

逻辑结构

存储结构

算法

研究内容

1.2 数据结构

- 定义：
  – 数据、数据元素、数据项、数据对象
  – 数据结构
  – 数据类型
  – 数据的逻辑结构
  – 数据的存储结构
- 抽象数据类型

1.2.1定义

(1) 数据、数据元素、数据项、数据对象

- 数据(data)：所有能被计算机识别、存储和处理的符号的集合（包括数字、字符、声音、图像等信息）。
- 数据元素(data element)：是数据的基本单位，具有完整确定的实际意义(又称元素、结点，顶点、记录等)。
- 数据项(Data item)：构成数据元素的项。数据项是具有独立含义的最小标识单位（又称字段、域、属性 等)。
- 数据对象(data)：具有相同特征的数据元素的集合，是数据的一个子集。

【例】学生成绩表

数据对象

数据项

序号	学号	姓名	班级	平时成绩	试卷成绩	总成绩	备注
	173510	张天天	计171	0	0	0	缺考
2	10171662	黄小宋	计171	0	0	0	缺考
3	10171327	王明阳	计171	90	75	80	
4	10173111	李乐雯	计172	73	60	64	
5	10173122	胡一一	计173	81	65	70	
6	10173123	赵淑芬	计173	92	90	91	
7	10170960	宋思	计174	80	80	80	

数据元素

关系：数据 > 数据对象 > 数据元素 > 数据项

1.2.1定义

(2)数据结构（Data Structures）

- **数据结构**是与特定问题相关的某一数据元素的集合和该集合数据元素之间的关系组成的。定义为：

  Data_Structure = { D, R }

- 其中：
  - D 是某一数据元素的集合；
  - R 是该集合中所有数据成员之间的关系的有限集合。

【例】长整数 “321465879”，以三个 3 位的十进制数表示，则可用如下描述的数学模型表示：

可用 $a_1 = 321$，$a_2 = 465$ 和 $a_3 = 879$ 的集合表示，且三者之间的次序关系必须是，a_1 表示最高 3 位，a_3 表示最低的 3 位，a_2 则是中间 3 位。

a_1，a_2 和 a_3 之间存在着次序关系：$<a_1, a_2>$ 和 $<a_2, a_3>$

1.2.1定义

(3) 数据类型（Data Type）

- **数据类型**:一组性质相同的值的集合, 以及定义于这个值集合上的一组操作的总称。
  - **基本数据类型（原子类型）**可以看作是计算机中已实现的数据结构。 例如:

char	int	float	double
字符型	整型	浮点型	双精度型

  - **结构数据类型**是由**基本数据类型**和**子结构**则构造而成。

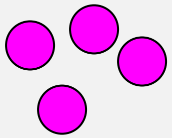
```
#define maxSize 100
typedef struct student{
   char cname[maxSize];
   int  dsscore;
   int  osscore;
  }
```

1.2.1定义

(4)数据的逻辑结构

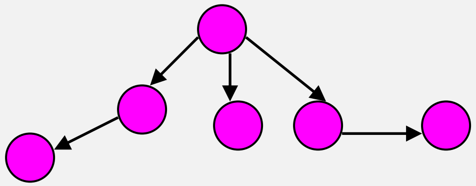
- **逻辑结构**：数据对象的逻辑组织关系。
- 数据的逻辑结构可以看作是从具体问题**抽象出来**的数据模型（面向应用的）；
- 数据的逻辑结构与数据元素**本身的形式、内容**无关；数据的逻辑结构与数据元素的**相对存储位置**无关。

四类基本结构

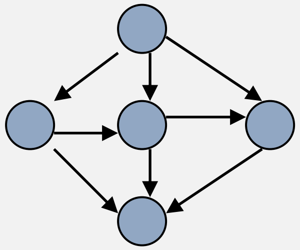


集合： 数据元素除了**同属于一种类型**外，别无其它关系。

线性结构：一对一。



树型结构：一对多。



图状结构或网状结构：多对多。

例1：摆放图书进行说明，数据逻辑结构：

- 图书按拼音顺序一本接着一本，一个编号对应一本书，即线性结构。
- 先把图书分类，某个类一个编号，一个编号对应多本书，一对多的逻辑结构，即树。
- 图书查询是一个多对多的关系网，即每个人可以查询多本书，每本书可以被多个人查询，关系网对应着图。

1.2.1定义

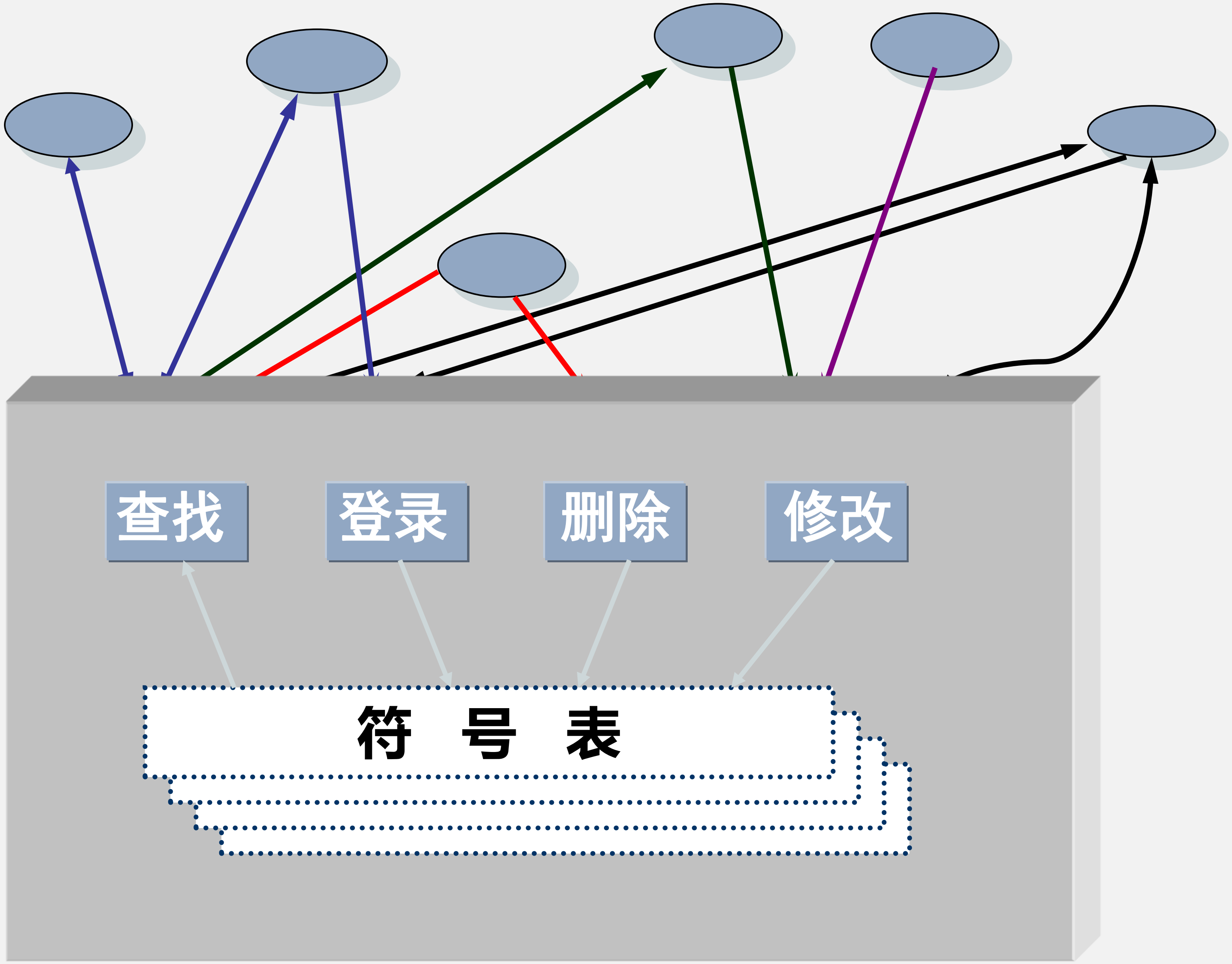
(5)数据的存储结构（物理结构）

- **物理结构**：数据对象信息在计算机内存中的存储组织关系，数据的存储结构依赖于计算机语言。
- 存储结构包括:
  - **顺序存储表示**：逻辑上相邻的元素存放到物理上相邻的存储单元
  - **链接存储表示**：元素的逻辑关系由附加的链接指针指示
  - **索引存储表示**：建立索引表，通过索引得到元素的存储地址
  - **散列存储表示**：根据元素的关键字，通过函数计算得到元素的存储地址

1.2.2 抽象数据类型

- **抽象数据类型：**是指一个数学模型以及定义在该模型上的一组操作。
- 抽象数据类型的定义仅取决于它的一组逻辑特征，不依赖于具体实现的，即数据对象集和操作集的描述与存放数据的**机器无关**、与数据存储的**物理结构无关**、与实现操作的**算法和编程语言均无关**。
- 抽象数据类型只描述数据对象集和相关操作集**“是什么”**，并不涉及**“如何做到”**的问题。
- 抽象数据类型由**构造数据类型**组成, 并包括一组相关的服务（或操作）

抽象数据类型



- **优点：**使程序编写得易于编程、易于测试、易于修改。
  - 实现**信息隐藏**，把所有数据和操作分为公有和私有，可减少接口复杂性，从而减少出错机会。
  - 实现**数据封装**，把数据和操作封装在一起，从语义上更加完整。
  - 实现**使用与实现相分离**，使用者只能通过接口上的操作来访问数据，一旦将来修改数据结构，可以使得修改局部化，提高系统灵活性。

抽象数据类型可以用以下的三元组来表示：

ADT ＝ （D，S，P）

数据对象　D上的关系集　D上的操作集

ADT常用定义格式

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抽象数据类型名

例：抽象数据类型“复数”的定义

ADT Complex {

数据对象：D = {e1, e2 | e1, e2∈RealSet }

数据关系：R1 = {<e1, e2> | e1是复数的实部，e2是复数的虚部 }

基本操作：

InitComplex(&Z, v1, v2)

操作结果：构造复数 Z，其实部和虚部分别被赋以参数 v1 和 v2 的值。

DestroyComplex(&Z)

初始条件：复数 Z 已存在。 操作结果：复数 Z 被销毁。

GetReal(Z, &realPart)

初始条件：复数 Z 已存在。操作结果：用realPart 返回 Z 的实部值。

GetImag(Z, &ImagPart)

初始条件：复数 Z 已存在。操作结果：用ImagPart 返回 Z 的虚部值。

Add(Z1, Z2, &sum)

初始条件：Z1, Z2 是复数。操作结果：用sum 返回 z1, z2 的和值。

} ADT Complex

用两个实数来表示复数，将复数定义为两个实数的有序对，并约定实部是前驱，虚部是后继。

【例】利用 C 语言实现的“复数”类型如下描述：

数据对象：D = {e1, e2 | e1, e2∈RealSet }

数据关系：R1 = {<e1, e2> | e1 是复数的实部，

e2 是复数的虚部 }

// 存储结构的定义

```c
typedef struct {
    float realpart;
    float imagpart;
} complex;
```

基本操作：

InitComplex(&Z, v1, v2)　DestroyComplex(&Z)　GetReal(Z, &realPart)

GetImag(Z, &ImagPart)　Add(Z1, Z2, &sum)

```
// 基本操作的函数原型说明
void add( complex z1, complex z2, complex &sum )
{
    // 以 sum 返回两个复数 z1，z2 的和基本操作的实现
  sum.realpart = z1.realpart + z2.realpart;
  sum.imagpart = z1.imagpart + z2.imagpart;
}
```

1.3 算法

- 定义
  - 算法的定义、算法与程序的关系、算法的描述、算法的结构、算法与人工算法的关系
- 算法复杂度
  - 算法的评价标准、与时间相关的因素、衡量标准（时间复杂度、空间复杂度)
- 渐进表示法
  - 大O表示法、大Ω表示法、 θ表示法

1.3.1定义

- 定义：一组完成特定任务的有穷指令序列。所有算法须满足如下标准：
- 特性：
  - 输入 有0个或多个输入
  - 输出 有一个或多个输出(处理结果)
  - 确定性 每步定义都是确切、无歧义的
  - 有穷性 算法应在执行有穷步后结束
  - 有效性 每一条运算应足够基本，可用计算机指令实现

序列中的每个操作都是可以简单完成的，其本身不存在算法问题，含义明确。例如，“求增加变量x的值”就不够基本。

注 算法的含义与程序十分相似，但二者是有区别的。

1、程序不一定满足有穷性（如一个操作系统在用户未使用前一直处于“等待” 的循环中，直到出现新的用户事件为止。这样的系统可以无休止地运行，直到系统停工）；

2、程序中的指令必须是机器可执行的，而算法中的指令则无此限制。算法若用计算机语言来书写，则它就可以是程序。

```c
#include <math.h>
#include <stdio.h>
#define MAXN 101
void sort(int List[],int n) {
  int i, j, min, temp;
  for (i = 0; i < n-1; i++) {
   min = i;
   for (j = i + 1; j < n; j++)
     if (List[j] < List[min]) min = j;
   temp = List[i];
   List[i] = List[min];
   List[min] = temp;
   }
}
int main(){
  int i, n;
  int List[MAXN];
  printf("请输入需要随机生成的数目,N=");
  scanf_s("%d", &n);
if (n<0 || n>MAXN) {
   fprintf(stderr,"数字错误! \n");
   exit(1);
   }
printf("\n排序前的数组为: ");
for (i = 0; i < n; i++) {
   List[i] = rand() % 1000;
   printf("%d    ",List[i]);
   }
sort(List, n);
printf("\n排序后的数组为: ");
for (i = 0; i < n; i++)
    printf("%d    ", List[i]);
return 0;
}
```

算法的描述

描述算法的方式一般有三种:

- 自然语言
- 伪代码语言
- 流程图

【例】从键盘中输入一个正整数，然后计算它和10之和。

（1）自然语言描述算法

①输入a；

②判断a是否大于0；

③如果a大于0，则计算b，b等于a加10,输出b;

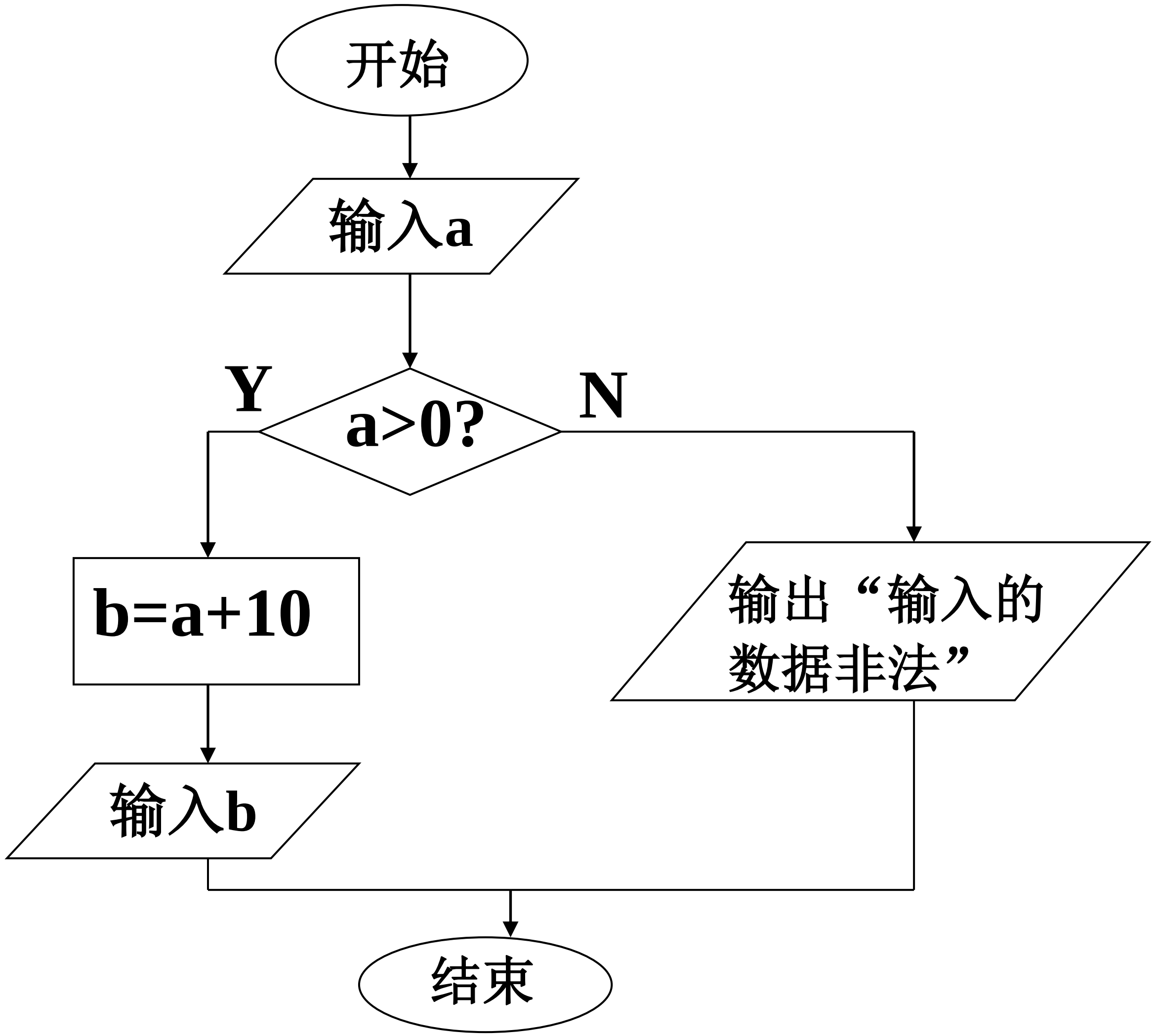
④如果a小于等于0，则输出“数据不合法”

（2）伪代码描述算法

```
{  输入a的值；
   if （a>0）
      {b=a+10；输出 b; }
   else
      输出“数据不合法！”
}
```

伪代码描述介于自然语言与程序设计语言之间

（3）流程图描述算法



说明：

(1)椭圆：表示起始、终止

(2)平行四边形：输入输出

(3)菱形：判断

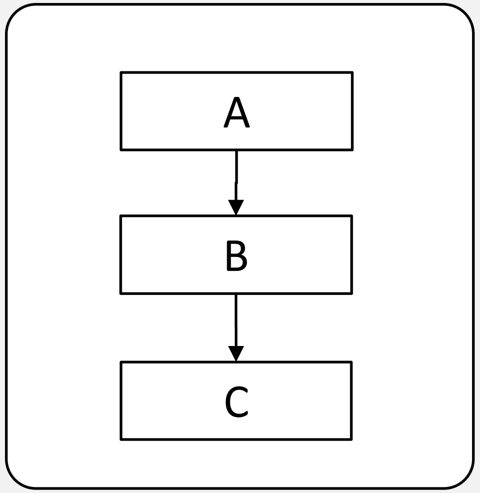
(4)矩形：执行表达式和赋值

(5)开口矩形：注释框

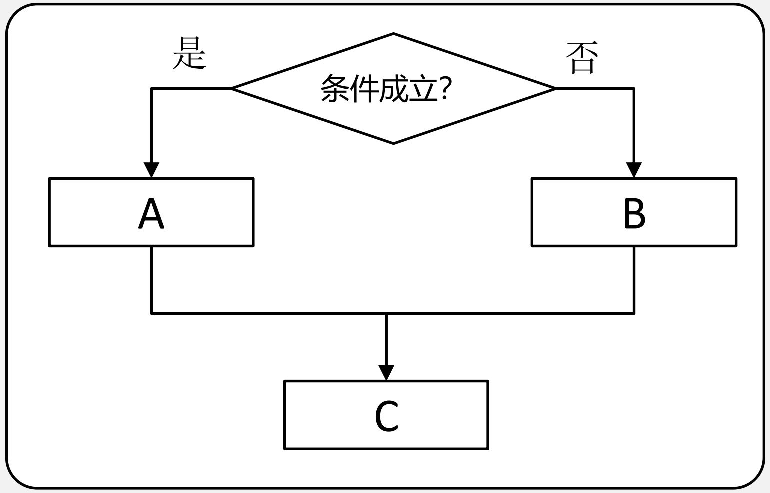
(6)箭头：数据流向

算法结构

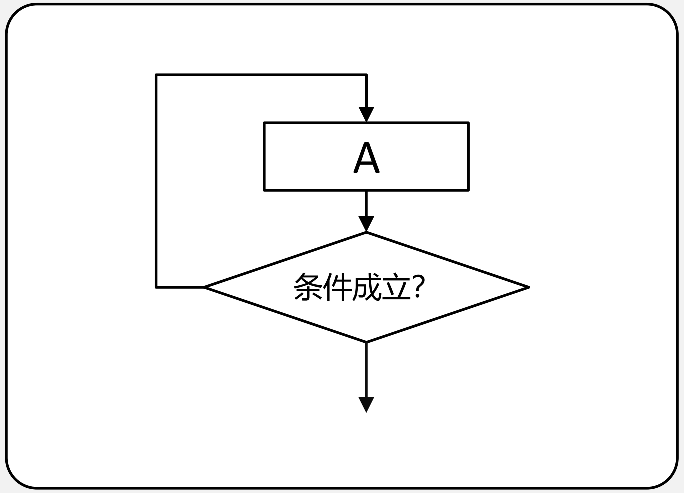
任何算法都可由顺序结构、选择结构、循环结构三种结构通过组合和嵌套表达出来。



顺序结构



选择结构



循环结构

计算机算法与人工算法的关系

例如 求定积分: s= $\int_a^b f(x)dx$

人工处理	**计算机算法**
找出f(x)的源函数F(x) 利用牛-莱公式:s=F(b)-F(a)	计算定积分采用数值积分的方法,得到一个近似解.

- **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**

1.3.2算法复杂度

- 算法的评价标准
- 与算法执行时间相关的因素
- 算法复杂度的衡量标准
  - 时间复杂度
  - 空间复杂度

什么是“**好**”的算法?

算法的评价标准

（1）正确性：算法应满足具体问题的需求。满足以特定的“规格说明”方式给出的需求。

1. 程序**不含语法错误**；
2. 程序对于**几组输入数据**能够得出满足规格说明要求的结果；
3. 程序对于**精心选择的典型、苛刻而带有刁难性的几组数据**能够得出满足规格说明要求的结果；
4. 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。

算法的评价标准

（2）可读性：

- 算法主要是为了人的**阅读**与**交流**，其次才是为计算机执行。
- 算法应该有助于人对算法的**理解**，晦涩难读的程序易于隐藏错误而难以调试。

（3）健壮性：算法应具有容错处理功能。

- 当输入的数据非法时，算法应当**恰当地作出反映**或**进行相应处理**，而不是产生莫名奇妙的输出结果。
- 处理出错的方法不应是中断程序的执行，而应是返回**一个表示错误或错误性质的值**，以便在更高的抽象层次上进行处理。

算法的评价标准

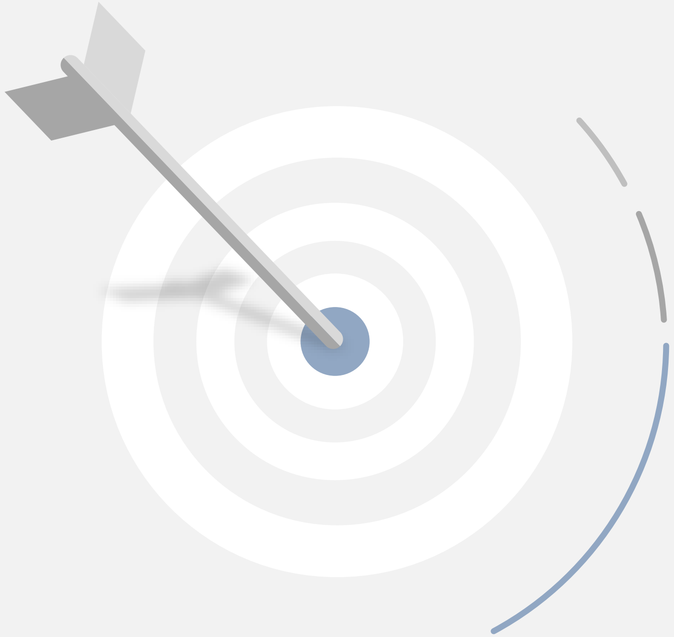
（4）效率与低存储量需求：

- **效率**指的是算法执行的时间（时间复杂性）；
- **存储量需求**指算法执行过程中所需要的最大存储空间（空间复杂性）

（5）简单性

- 所采用数据结构和方法的简单程度
- 算法越简单，出错率越低

与算法执行时间相关的因素



- 算法选用的策略
- 问题的规模
- 编写程序的语言
- 编译程序产生的机器代码的质量
- 计算机执行指令的速度

与算法执行时间相关的因素(算法策略)

【例】考虑计算三次多项式 $ax^3 + bx^2 + cx + d$

方法1： $s = a \times x \times x \times x + b \times x \times x + c \times x + d$

(6次乘法，3次加法)　**计算量大**

方法2： $s = a$； $s = s \times x + b$； $s = s \times x + c$； $s = s \times x + d$；

(3次乘法，3次加法)　**计算量小**

方法2 的原理：

$$a \times x \times x \times x + b \times x \times x + c \times x + d$$
$$= (a x + b) x^2 + c x + d$$
$$= ((a x + b) x + c) x + d$$

不同的算法策略的执行时间不同

与算法执行时间相关的因素（问题规模）

问题的规模一般根据问题本身的性质合理地确定：

【例】对 n 个电话号码进行排序，这里 n 即可作为问题的规模。显然对 1000 个电话号码进行排序比对 10 个电话号码进行排序规模要大。

【例】求 n 阶矩阵的转置，这里 n 即可作为问题的规模。

衡量标准——时间复杂度

- 时间复杂度：根据算法写成的程序在执行时耗费时间的长度。
  - 运行时间 = 算法每条语句执行时间之和。
  - 每条语句执行时间 = 该语句的执行次数 (频度)×语句执行一次所需时间。
  - 语句执行一次所需时间取决于机器的指令性能和速度和编译所产生的代码质量，很难确定。
  - 设每条语句执行一次所需时间为单位时间，则一个算法的运行时间就是该算法中所有语句的频度之和。

例：累加求和

各执行1次原操作。

```
int Sum(int b[n], int n)
   {
      int i, s=0 ;
      for(i=0; i<n; i++)
            s+=b[i] ;
      return s ;
   }
```

for 循环语句所包含的原操作：

```
    i=0;                          // 1 次
m1:   if(i>=n) goto m2;           //n+1次
         s+=b[i];                 //n次
         i++;                     //n次
         goto m1;                 //n次
m2:   return s;
```

时间复杂度为： $T(n)=4n+4$

算法的时间复杂度通常是问题的规模 *n* 的某个函数 *T*(*n*)

衡量标准——空间复杂度

空间复杂度：根据算法写成的程序在执行时占用存储单元的大小。

```
int main()
{   int a;
    printf("输入年份");
    scanf_s("%d", &a);
    if (a % 4 == 0 && a % 100 || a
% 400 == 0)
        printf("输入是闰年！\n");
    else
        printf("输入不是闰年！\n");
}
```

```
#define MAXN 9999
int main()
{   int a[MAXN], year;
    …… a[2000]=1;a[2001]=0;    ……
    printf("输入年份");
    scanf_s("%d", &year);
    if (a[year]==1)
      printf("输入是闰年！\n");
    else
      printf("输入不是闰年！\n");
}
```

衡量标准——最好、平均、最坏复杂度

【例】插入排序

```
Insertion-Sort(A)
    for j=2 to A.length
    {key=A[j];
    //将A[j]插入到数组A[1..j-1]中
      i=j-1;
      while i>0 and A[i]>key
      {   A[i+1]=A[i];
          i=i-1;}
      A[i+1]=key;
    }
```

通常关注两种复杂度：

- 最坏情况复杂度；
- 平均复杂度

实践表明，最有使用价值的且操作性好的是最坏时间复杂性.

1.3.3 渐进表示法

- 当一个问题的输入规模很大时,算法的结构又很复杂时,采用精确分析就显得过于繁琐；
- 为降低算法分析的代价,同时又保证估算的精确度,引入一个简化的计算模型来评估算法的开销称为渐进表示法；
- 渐进表示法是观察算法的“增长趋势”，判断哪种算法必定效率更高。

(1)大O表示法 (算法运行时间的上限)

若∃正常数c和 自然数N_0使得当 $n \ge N_0$时,有$T(n) \le cf(n)$ 则称函数 $T(n)$在n充分大时有上界, 且 $f(n)$是它一个上界.

记为 $T(n) = O(f(n))$, 也称 $T(n)$ 的阶不高于$f(n)$ 的阶.

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

$n^2=O(n^3)$?　√　　$n^3=O(n^2)$?　$\neq$

(2) 大Ω表示法 (算法运行时间的下限)

若∃正常数c和自然数N_0使得当 $n \ge N_0$ 时,有$T(n) \ge c\, g(n)$ 则称函数$T(n)$在n充分大时有**下限**, 且 $g(n)$是它的一个下限,

记为$T(\mathrm{n}) = \Omega\,(g\,(n)\,)$ 也称$T(n)$的阶不低于 $g(n)$的阶

例 $T(n)=c_1 n^2+c_2 n$, 则 $T(n)= \Omega\,(n^2)$

上下界函数越接近真实函数越好

(3) θ表示法

$T(n) = \theta(h(n))$等价于

$T(n)=O(h(n))$ 且 $T(n)=\Omega(h(n))$

称函数**$T(n)$**与**$h(n)$** 同阶.

例 $T(n)=c_1n^2+c_2n$， 则 $T(n)=\theta(n^2)$

渐进表示法可用于分析时间复杂度，也可用于空间复杂度

- 大O表示法的加法规则（针对并列程序段）

$$\left.\begin{aligned}T_1(n) &= O(\,f(n)\,)\\ T_2(m) &= O(\,g(m)\,)\end{aligned}\right\}\ \textbf{并列}$$

$$T(n, m) = T_1(n) + T_2(m) = O(\max(f(n), g(m)))$$

- 例如，有三段并列程序段

```
x = 0;  y = 0;

for (int k=0; k<n; k++)  x++;

for (int i=0; i<n; i++)
    for (int j=0; j<n;j++)  y++;
```

$T_1(n) = O(1)$

$T_2(n) = O(n)$

$T_3(n) = O(n^2)$

$O(n^2)$

- 大O表示法的乘法规则（针对嵌套程序段）

$$\left.\begin{aligned} T_1(n) &= O(f(n)) \\ T_2(m) &= O(g(m)) \end{aligned}\right\} \textbf{嵌套}$$

$$T(n, m) = T_1(n) \times T_2(m) = O(f(n) \times g(m))$$

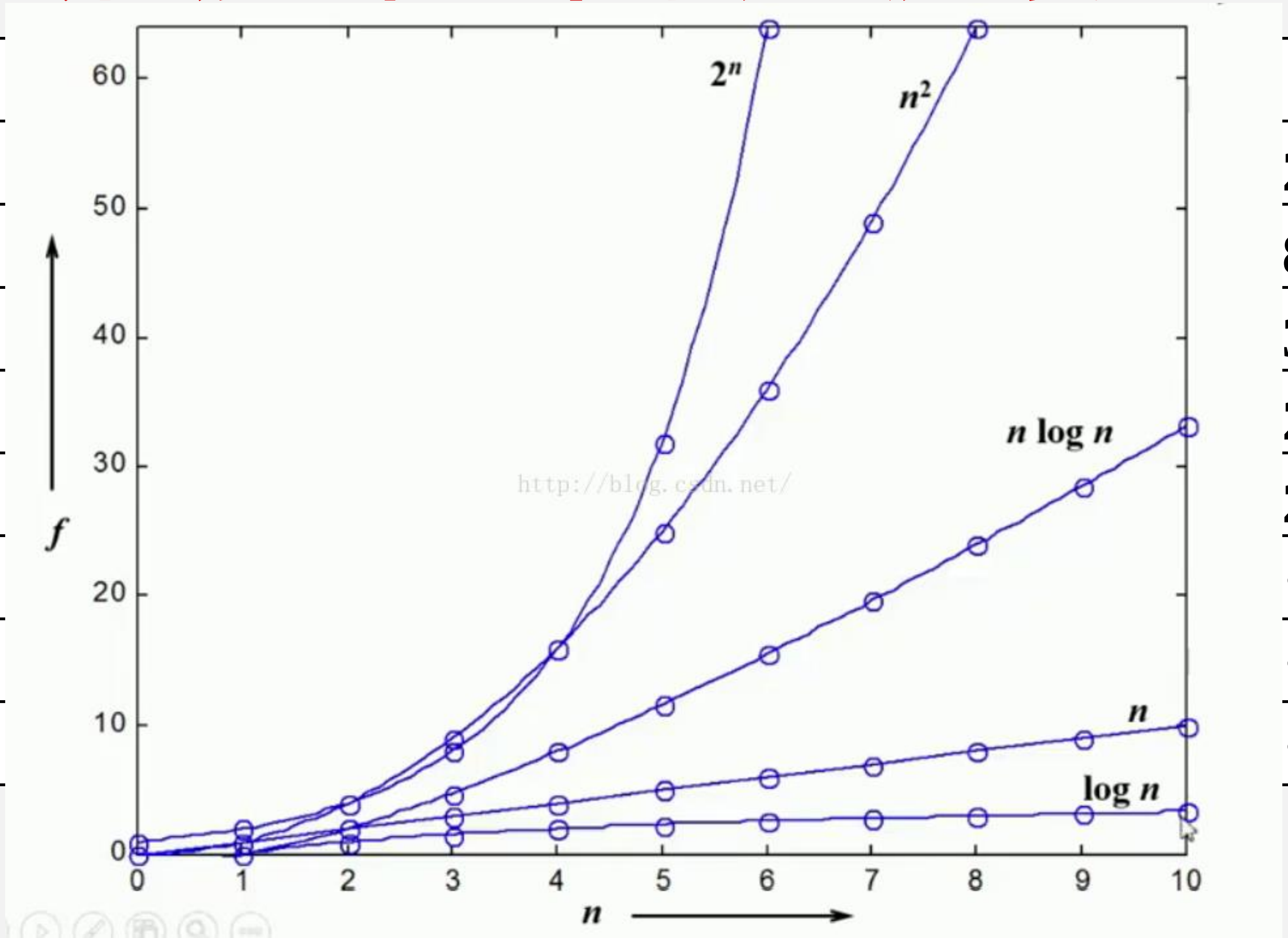
- 例：选择排序算法思路为：外循环共 n–1 次，内循环从 i 到 n 选择最小者，将其对调到第 i 个元素位置。

```
void SelectionSort ( int List[], int N )
{ /* 将N个整数List[0]...List[N-1]进行非递减排序 */
    for ( i = 0; i < N; i ++ ) {                          T1(n) =O(n)
        MinPosition = ScanForMin( List, i, N–1 );  T2(n) = O(n)
  /* 从List[i]到List[N–1]中找最小元，并将其位置赋给MinPosition */
        Swap( List[i], List[MinPosition] );
        /* 将未排序部分的最小元换到有序部分的最后位置 */
}}
```

$O(n^2)$

算法复杂性的不同数量级的变化

n	n!
4	24
8	80320
10	3628800
16	$2.1*10^{13}$
32	$2.6*10^{35}$
128	∞
1024	∞
10000	∞



$$c < \log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n < 3^n < n!$$

著名的计算机科学家沃思 (N.Wirth)教授，提出“程序的构成与数据结构是两个不可分割地联系在一起的问题”。

程序 ＝ 算法 ＋ 数据结构

程序设计：为计算机处理问题编制的一组指令集

算　　法：处理问题的策略

数据结构：问题的数学模型

本章小结

数据 个体 数据元素 性质相同的构成的集合 数据对象

集合结构
线性结构
树形结构
图状结构

由关系不同分类 数据结构 加上数据元素之间的关系

加上操作

映像到内存

数据对象
数据关系
基本操作

抽象数据类型

存储结构

顺序结构
链式结构

算法

特性：有穷性、确定性、可行性、输入、输出

衡量标准：时间复杂度 空间复杂度

THANKS

华东理工大学 叶琪