第一章　绪　　论

本章学习的四个问题：

- 什么是数据结构
- 基本概念、术语
- 抽象数据类型
- 算法和算法分析

第一章　绪　　论

1.1什么是数据、结构(关系)、数据结构?

建立数学模型是分析具体问题的过程，包括：

- 分析具体问题中的操作对象
- 找出这些对象间的关系，并用数学语言描述

数学模型分两类：

1）数值计算类：

例：根据三条边，求三角形面积。

假定：三条边依次为a, b, c三个实型数，

满足：a>0，　b>0，　c>0，　a+b>c，　b+c>a，　c+a>b，

则　　$s=\frac{a+b+c}{2}$　　$area=\sqrt{s*(s-a)*(s-b)*(s-c)}$

2）非数值计算类：

例1：5个整数组成的集合：

D={20,-5，66，15,44}

其中:20,-5,66等称为数据元素(元素)，

元素与元素之间关系是它们同属于集合D。

元素与元素间无直接关系。

例2：一列整数：(线性结构)

L=(20,-5，66，15,44)

其中:元素与元素之间在L中是前后关系或线性关系。

L=(20,-5,66,15,44)是一个线性表。

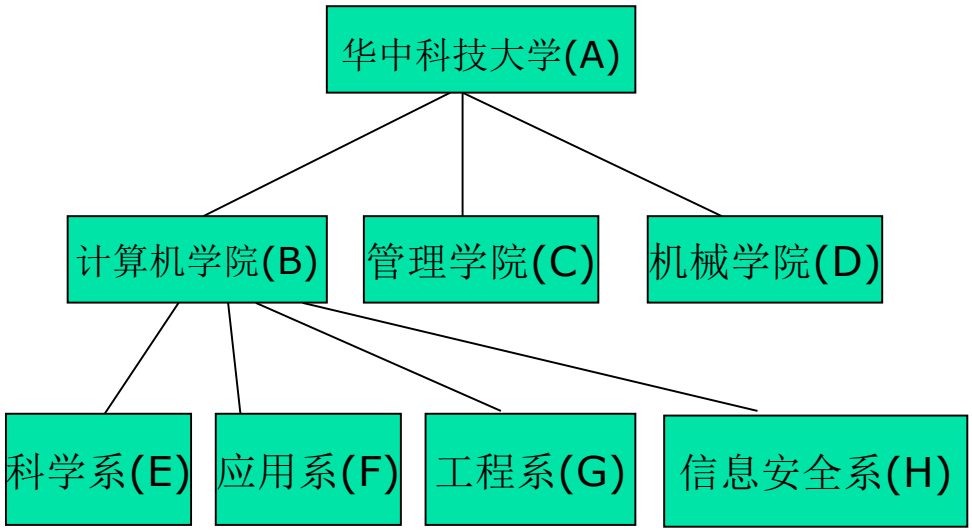
例3 一张登记表DL

序号	姓名	性别	年龄	
1	李　刚	男	25	记录1
2	王　霞	女	29	记录2
3	刘大海	男	40	记录3
4	李爱林	男	44	记录4

其中：

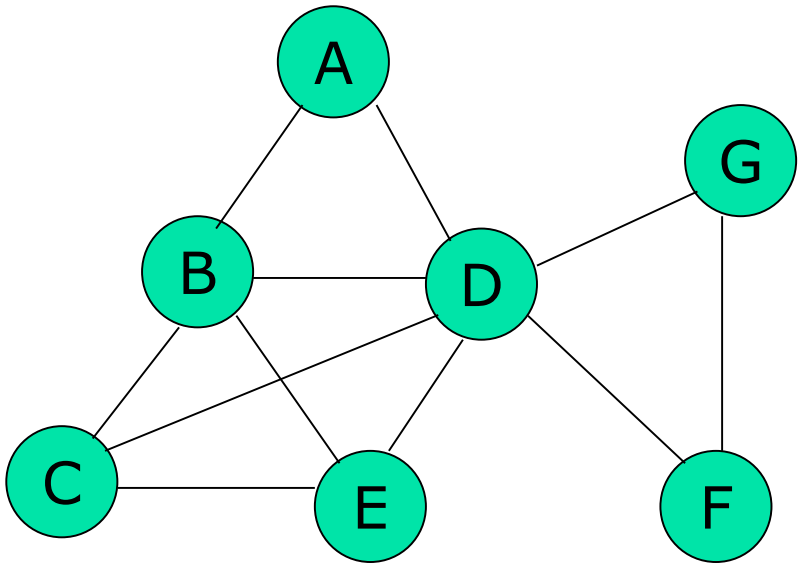
- 姓名、性别、年龄是数据项(item)、数据域(field)；
- (姓名，性别，年龄)是记录(record)，
- C语言将"记录"(record)定义为”结构”(struct)；
- 登记表也是一个线性表。

例4　树状结构



其中:A、B、C等是结点(node);
A与B, B与E, A与C之间是层次关系或父子关系。

例5 图(网)状结构



其中：A、B、C 等是顶点(vertex)，
图中任意两个顶点之间都可能有关系。

例6 田径赛的时间安排问题：

- 设有六个比赛项目，规定每个选手至多可参加三个项目，有五人报名参加比赛（如下表所示）设计比赛日程表，使得在尽可能短的时间内完成比赛。

姓 名	项目 1	项目 2	项目 3
丁 一	跳 高	跳 远	100 米
马 二	标 枪	铅 球	
张 三	标 枪	100 米	200 米
李 四	铅 球	200 米	跳 高
王 五	跳 远	200 米	

- 用如下六个不同的代号代表不同的项目：

  跳高　跳远　标枪　铅球　100米　200米

  A　　B　　C　　D　　E　　F

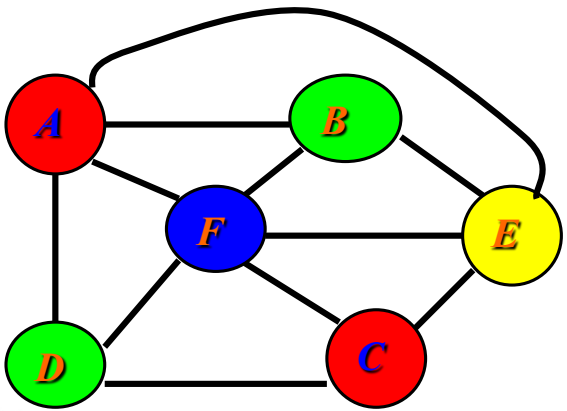
- 用顶点代表比赛项目
  - 不能同时进行比赛的项目之间连上一条边。
  - 某选手比赛的项目必定有边相连（不能同时比赛）。

姓名	项目1	项目2	项目3
丁一	A	B	E
马二	C	D	
张三	C	E	F
李四	D	F	A
王五	B	F	

- 对图上的每个顶点染一种颜色，并且要求有线相连的两个顶点不能具有相同颜色，而总的颜色种类应尽可能地少。同色可以同时比赛。

姓名	项目1	项目2	项目3
丁一	A	B	E
马二	C	D	
张三	C	E	F
李四	D	F	A
王五	B	F	

只需安排四个单位时间进行比赛



比赛时间	比赛项目
1	A，C
2	B，D
3	E
4	F

- 教材P1-2
  - 例1、例2、例3

- 综上几个例子可见，描述这类非数值问题的数学模型不再是数学方程，而是诸如表、树和图之类的数据结构。

- 数据结构是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作等相关问题的学科。

- 计算机内的数值运算依靠方程式，而非数值运算（如表、树、图等）则要依靠数据结构。
- 同样的数据对象，用不同的数据结构来表示，运算效率可能有明显的差异。
- 程序设计的实质是对实际问题选择一个好的数据结构，加之设计一个好的算法。而好的算法在很大程度上取决于描述实际问题的数据结构。

程序设计=数据结构+算法

1.2 基本概念和术语

1.数据（data)：

所有能输入到计算机中并被计算机程序加工、处理的符号的总称。

如：整数、实数、字符、声音、图象、图形等。

2.数据元素(data element)：

数据的基本单位。（元素、记录、结点、顶点）

在计算机程序中通常作为一个整体进行考虑和处理。

3.数据项(data item)：

是数据的不可分割的最小单位。如：姓名、年龄等。

一个数据元素可由一个或多个数据项组成。

如：(姓名、年龄)

4. 数据对象(data object)

由性质相同(类型相同)的数据元素组成的集合。

数据对象是数据的一个子集。

例1 由4个整数组成的数据对象

D1={20, -30, 88, 45}

例2 由正整数组成的数据对象

D2={1, 2, 3, . . . }

例3 由26个字母组成的数据对象

D3={A, B, C, . . . , Z}

其中：D1，D3是有穷集，D2是无穷集。

5. 数据结构(data structure)

- 在任何问题中，数据元素都不是孤立存在的，而是在它们之间存在着某种关系，这种数据元素相互之间的关系称为结构(Structure)。

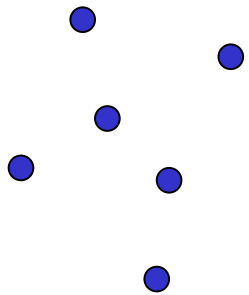
- 数据结构就是相互之间存在一种或多种特定关系的数据元素的集合。

按照视点不同，数据结构分为：

- 逻辑结构 — 面向问题
- 物理结构 — 面向计算机

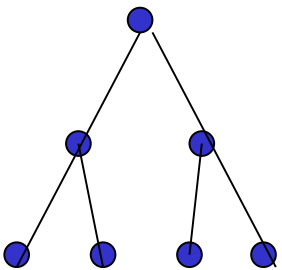
5.1 数据的逻辑结构

数据之间的相互关系称为逻辑结构，即从逻辑关系上描述数据，与数据的存储无关，是独立于计算机的。通常分为四类基本结构：

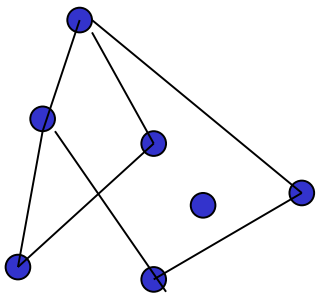


集合

线性结构



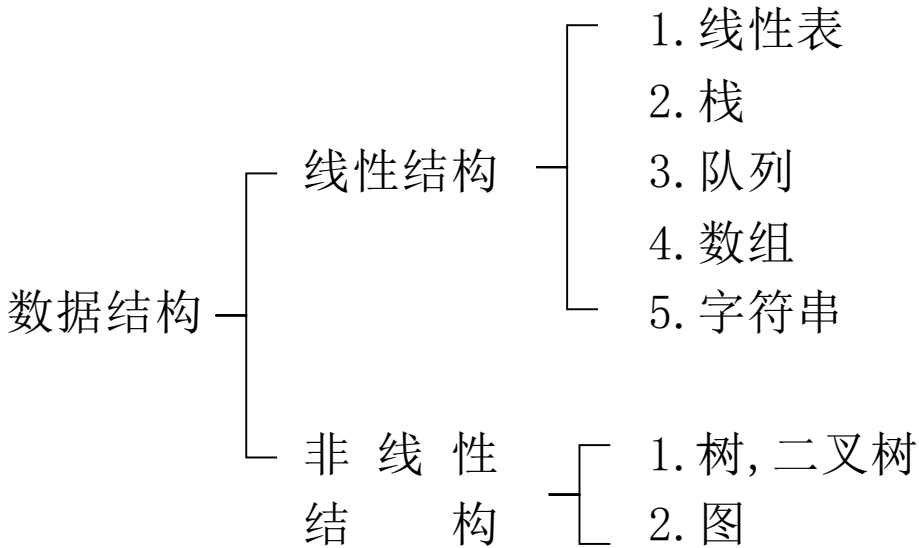
树形结构



图（网）状结构

5.1 数据的逻辑结构

分类：

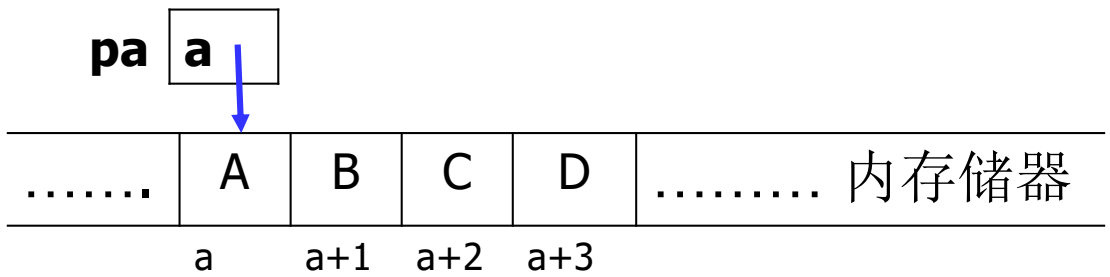


5.2 数据的物理（存储）结构

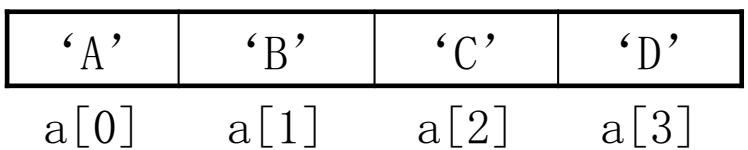
物理结构亦称存储结构，是数据的逻辑结构在计算机存储器内的表示（或映像），依赖于计算机。存储结构可以分为四大类：

(1) 顺序存储结构(向量, 一维数组)

例. 线性表L＝（‘A’，‘B’，‘C’，‘D’）；

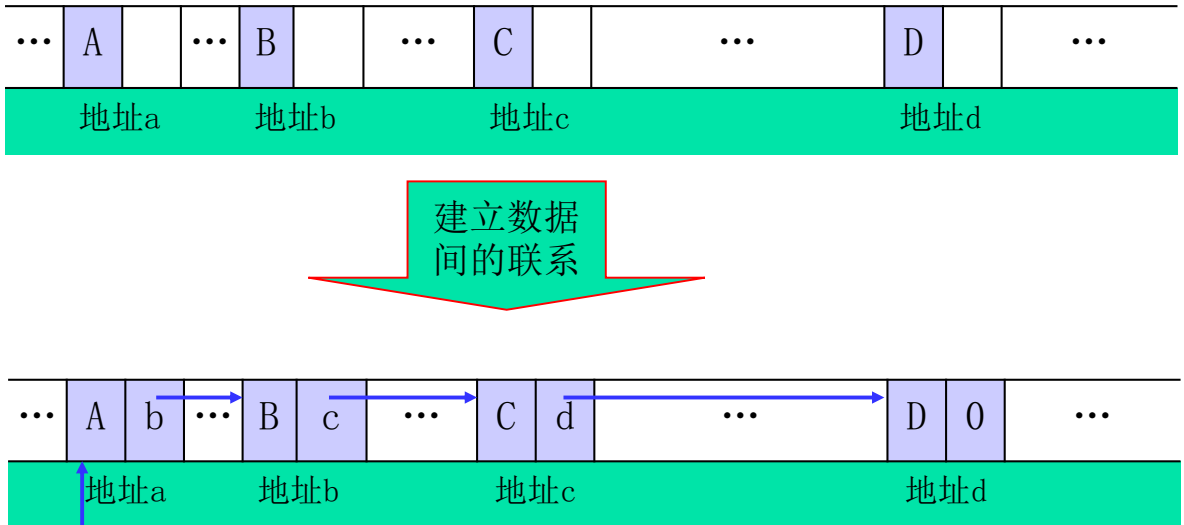


C语言实现方法：char a[4]={‘A’，‘B’，‘C’，‘D’}；



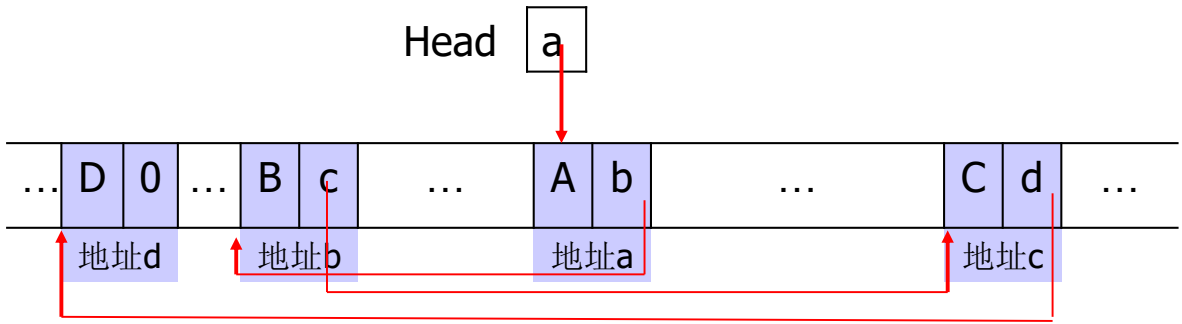
(2) 非顺序存储结构(链接表、链式存储结构)

例. 单链表：分配不一定连续的空间

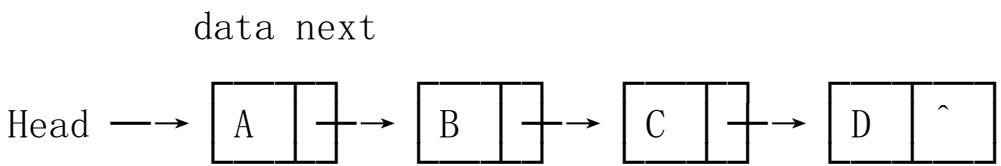


(2) 非顺序存储结构(链接表、链式存储结构)

例. 单链表：存放数据的空间顺序可任意



一般表达形式：



4个结点的单链表

(3) 索引存储结构

除建立存储结点信息外，还建立附加的索引表来标识结点的地址。

(4) 哈希（Hash）存储结构

又称散列存储，是一种力图将数据元素的存储位置与关键码之间建立确定对应关系的存储技术。散列法存储的基本思想是：由节点的关键码值决定节点的存储地址。

6. 数据类型（data type）

是一个值的集合和定义在这个值上的一组操作的总称。

(1)原子类型(如： int,char,float等)

(2)结构类型(如： 数组,结构,联合体等)

7. 抽象数据类型(Abstract Data Type)

与计算机的实现无关的数据类型。

- 形式定义：

ADT 抽象数据类型名

{ 1.数据对象；

2.数据关系:一个或多个关系；

3.一组基本操作/运算

} ADT 抽象数据类型名

- 数据对象和数据关系的定义用伪码描述
- 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>

  基本操作有两种参数：传值参数，引用参数（以&符号打头）
- 例1-6（参见书第9页）

1.3. 抽象数据类型的表示与实现

- 采用类C语言描述数据结构。类C语言精选了C语言的一个核心子集，同时作了若干扩充修改（伪码），增强了语言的描述功能
- 类C语言简要说明（参见书10-11页）
- 为了便于算法描述，使算法易于理解，类C中函数的形参，除了允许值参外，增添了C++语言的引用参数。在函数的形参表中，以&打头的参数即为引用参数

  *注意：引用参数是实参的别名

〈1〉预定义变量和类型：

```
#define TRUE     1
#define FALSE    0
#define OK         1
#define ERROR   0
#define INFEASIBLE     -1
#define OVERFLOW      -2
typedef int Status;      //一般用于算法函数的返回类型
typedef char ElemType;  // 此时char等价于Elemtype;
```

〈2〉函数：　用以表示算法

函数类型　函数名（函数参数表）　{

//算法说明

语句序列

} //函数名

注：

算法说明应包括功能说明，输入，输出；

为提高算法可读性，关键位置加以说明或注释；

明确函数实参和形参的匹配规则，以便能正确使用算法函数。

〈3〉其他

理解引用参数

例：交换两个整数变量

```
void swap ( int n, int m) {
//参数为值参
  int temp;
  temp=n; n
}

main( ) {
  int a=10,b
  swap(a,b);
}
```

结果： a=10, b=20

```
void swapx ( int &n, int &m ){
//参数为引用参数

  =temp;
```

引用

结果： c=20, d=10

原因：

调用swap(a,b)时，实参a,b的值被传递给swap(int n,int m)的形参n,m, 调用swap(a,b)的结果是n和m的值交换而变量a,b的值并没有交换；

调用swapx(c,d)时，形参n,m作为实参c,d 别名，n和c对应同一个变量， m和d对应同一个变量，调用swapx(c,d)的结果是n和m的值交换，即变量c,d的值交换；

1.4 算法和算法分析

1.算法定义：

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

例.求a[0]..a[n-1]中n个数的平均值(假定n>0)。

```
float average(float a[ ],int n)
{ int i; float s=0.0;    //累加器赋初值
  for (i=0; i<n; i++)
     s=s+a[i];           //a[i]累加到s中
  s=s/n;                 //计算平均值
  printf(“ave=%f”,s);    //输出
  return(s);             //返回
 }
```

其中：a,n为输入量；s为输出量。

2. 算法的5个特征：

(1) 有穷性：在有限步(或有限时间)之后算法终止。

例. { i=0; s=0;
while (i<=10)
{ s+=i;
i++; }
}

(2) 确定性：无二义性。

(3) 可行性：算法中的操作都是已经实现的基本运算执行有限次来实现的。

(4) 输入：有0或多个输入量。

(5) 输出：至少有一个输出量。

3. 算法设计要求：

(1) 正确性：

a. 无语法错误；

b. 对n组输入产生正确结果；

c. 对特殊输入产生正确结果；

d. 对所有输入产生正确结果。

(2) 可读性：“算法主要是为了人的阅读与交流”。

(3) 健壮性：

scanf(“%d”,&x)； y/=x；？？？

(4) 高效率与低存储量

下列描述不符合算法的什么特征和要求？

例1

```
void suanfa1( )
{ int i,s=0;
  for (i=0; i>=0; i++)   //死循环
    s++;                 //不能终止
 }
```

例2

```
float suanfa2( )
{ int x,y;
  scanf(“%d”, &x);
  y=sqrt(x);             //当x<0时,出错
  return(y);
 }
```

4. 算法效率的度量

- 事后统计： 收集此算法的执行时间和实际占用空间的统计资料
- 事先分析： 求出该算法的时间界限函数
- 将一个算法转换成程序并在计算机上执行时，其运行所需要的时间取决于下列因素：

  (1)硬件的速度。(2)书写程序的语言。(3)编译程序所生成目标代码的质量。(4)问题的规模。
- 在各种因素都不能确定的情况下，很难比较出算法的执行时间。也就是说，使用执行算法的绝对时间来衡量算法的效率是不合适的。为此，将上述各种与计算机相关的软、硬件因素都确定下来，这样一个特定算法的运行工作量的大小就只依赖于问题的规模（通常用正整数n表示），或者说它是问题规模的函数。

5. 算法的时间复杂度:

1、时间频度:

- 一个算法中的原操作执行次数称为语句频度或时间频度，记为f(n)

2、时间复杂度:

- T(n) = O(f(n)) 称为算法的渐近时间复杂度，简称时间复杂度。大O（Order的缩写，意指数量级）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。
- 大O操作的本质是求f(n)的同阶无穷大。当n趋近于无穷大时，f(n)/g(n)的极限值为不等于零的常数，则称g(n)是f(n)的同数量级函数。

  例如，若$f(n)=n(n+1)/2$, 则有 $1/2 \le f(n)/n^2 \le 1$，故时间复杂度为$T(n)=O(f(n)) = O(n^2)$, 即T(n)与n^2 数量级相同

已知三个表示算法语句频度的函数f，g和h如下，求它们对应的时间复杂度

$f(n) = 100n^3 + n^2 + 1000$

$g(n) = 25n^4 + 500000n$

$h(n) = n^{1.5} + 5000n\log_2 n$

$O(f(n)) = O(n^3)$　$O(g(n)) = O(n^4)$

因当$n->\infty$时，$n^{1.5}>n\log_2 n$

故有 $O(h(n)) = O(n^{1.5})$

如何分析算法的时间复杂度？

例1 {
```
    int s;
    scanf(“%d”,&s);
    s++;
    printf(“%d”,s);
}
```
其中：语句频度为：f(n)=3

时间复杂度为：T(n)=O(f(n))=O(3)=O(1)

O(1)称为常量阶/常量数量级

例2 分析下面的算法

```
void sum(int a[],int n)
{ int s=0,i;                    // 1次
  for(i=0; i<n; i++)            // n次(严格为2*(n+1)次)
     s=s+a[i];                  // n次
  printf("%d",s);               // 1次
}
```

其中：语句频度为： f(n)=1+n+n+1

时间复杂度为：T(n)=O(f(n))=O(2n+2)=O(n)

O(n)称为线性阶/线性数量级

例3 分析下面的算法

```
1.  void sum(int m,int n)
2.  { int i,j,s=0;                          // 1次
3.    for(i=1; i<=m; i++)                   // m次
4.     { for(j=1; j<=n; j++)                // m*n次
5.         s++;                             // m*n次
6.       printf("%d",s);                    // m次
7.     }
8.  }
```

其中:$f(m,n)=1+m+2*m*n+m=2mn+2m+1$

当m=n时,$f(n)=2n^2+2n+1$

$T(n)=O(f(n))=O(2n^2+2n+1)=O(n^2)$

$O(n^2)$ 称为平方阶/平方数量级

例4 分析下面的算法

```
1.  void sum(int n)
2.  { int i,j,s=0;                          // 1次
3.    for(i=1; i<=n; i++)                   // n次
4.     { for(j=1; j<=i; j++)                // ?次
5.        s++;                              // ?次
6.       printf("%d",s);                   // n次
7.     }
8.  }
```

其中：第4行的次数为 1+2+...+n=n(1+n)/2

第5行的次数为 1+2+...+n=n(1+n)/2

$f(n)=1+n+n(n+1)+n=n^2+3n+1$

$T(n)=O(f(n))=O(n^2)$

$O(n^2)$ 称为平方阶/平方数量级

例5 冒泡排序的C语言算法(对数组a中n个数按递增次序排序)

```
1.  void  bubble1(int a[],int n)
2.  { int i,j,temp;
3.    for(i=1; i<n; i++)                    // ? 次
4.      for(j=0; j<n-i; j++)                // ? 次
5.        if (a[j]>a[j+1])                  // ? 次
6.          { temp=a[j];                    // ? 次
7.            a[j]=a[j+1];                  // ? 次
8.            a[j+1]=temp;                  // ? 次
9.          }
10.   for(i=0; i<n; i++)                    // n 次
11.     printf("%d",a[i]);                    // n 次
12. }
```

思考：在最好情况下，f(n)= ?　　T(n)=O(f(n))= ?

　　在最坏情况下，f(n)= ?　　T(n)=O(f(n))= ?

一般情况：

i值	原序列：	10	3	7	8	5	2	1	4	9	6	交换范围
1	第1遍：	3	7	8	5	2	1	4	9	6	**10**	[0—10-1)
2	第2遍：	3	7	5	2	1	4	8	6	**9**	**10**	[0—10-2)
3	第3遍：	3	5	2	1	4	7	6	**8**	**9**	**10**	[0—10-3)
4	第4遍：	3	2	1	4	5	6	**7**	**8**	**9**	**10**	[0—10-4)
5	第5遍：	2	1	3	4	5	**6**	**7**	**8**	**9**	**10**	[0—10-5)
6	第6遍：	1	2	3	4	**5**	**6**	**7**	**8**	**9**	**10**	[0—10-6)
7	第7遍：	1	2	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**	[0—10-7)
8	第8遍：	2	1	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	[0—10-8)
9	第9遍：	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	[0—10-9)

最坏情况：

10 9 8 7 6 5 4 3 2 1

每次比较都发生数据交换。

最好情况：

1 2 3 4 5 6 7 8 9 10

每次比较都不发生数据交换。

```
1.  void  bubble1(int a[],int n)
2.  { int i,j,temp;
3.    for(i=1; i<n; i++)              // n-1 次
4.      for(j=0; j<n-i; j++)          //n-1+...+1=n(n-1)/2
5.        if (a[j]>a[j+1])            // n(n-1)/2次
6.          { temp=a[j];              // 0 或n(n-1)/2次
7.            a[j]=a[j+1];            // 0 或n(n-1)/2次
8.            a[j+1]=temp; }          // 0 或n(n-1)/2次
9.   for(i=0; i<n; i++)               // n 次
10.     printf("%d",a[i]);            // n 次
11. }
```

在最好情况下，$f(n)= n-1+n(n-1)+2n=n^2+2n-1$

在最坏情况下，$f(n)=5n^2/2+n/2-1$

$T_{最好}(n)= T_{最坏}(n) = O(n^2)$

算法改进思想：

每一遍开始时，设置change=false，处理时，一旦发生数据交换，就将change修改成true。结束时，若change未变，表示未发生数据交换，即已递增有序。

如当前序列为：

$a_0, a_1, \cdots, a_{i-1}, a_i, a_{i+1} \cdots a_{n-1}$.

当前需要处理的范围是0~i，　　$0 \leqslant i \leqslant n-1$

处理前，设置 change=false，

若$a_0 \leqslant a_1 \leqslant \cdots \leqslant a_{i-1} \leqslant a_i$，显然不会发生数据交换，所以change未变，即已是递增有序，排序操作可就此结束；否则一定会发生数据的交换，change被修改为true，需要进行下一遍的处理。

例6 改进的冒泡排序算法时间复杂度分析

```
1.  void  bubble2(int a[],int n)
2.  { i=n-1;                                  // 1
      do {
3.       change=FALSE;                        // 1 或n-1
4.       for(j=0; j<i; ++j)                   //n-1或n*(n-1)/2
5.         if (a[j]>a[j+1])                   //n-1或n*(n-1)/2
6.           { a[j] <--> a[j+1];              // 0 或n*(n-1)/2
7.             change=TRUE;                   // 0 或n*(n-1)/2
8.           }
9.       } while(change && --i>=1)            // 1 或n-1
10. }
```

在最好情况下： $T_{最好}(n)=O(f_{最好}(n))= O(n)$

在最坏情况下： $T_{最坏}(n)=O(f_{最坏}(n))= O(n^2)$

算法时间复杂度： $T(n)= T_{最坏}(n)= O(n^2)$

例7： 求解：S=1!+2!+.....+n!

```
long  sum_of_fac(int n)
   {
    long s=0,t,i,j;
    for(i=1;i<=n;i++)                    //n次
      {
       t=1;                              //n次
       for(j=1;j<=i;j++)                 //n*(n+1)/2次
              t*=j;                      //n*(n+1)/2次
       s+=t;                             //n次
      }
    return (s);                          //1次
   }
```

求i!

$f(n)=n+n+n*(n+1)/2+ n*(n+1)/2+n+1=n^2+4n+1$

$T(n)=O(f(n))=O(n^2)$

例7(续)： 求解：S=1!+2!+.....+n!

改进后算法：

```
long sum_of_fac(int n)
 {
  long s=0,t,i;
  t=1;                                  //1次
  for(i=1;i<=n;i++)                     //n次
    {
     t*=i;                              //n次
     s+=t;                              //n次
    }
  return (s);                           //1次
 }
```

求i!

$f(n)=1+n+n+n+n+1=3n+2$

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

例8 下列算法时间复杂度分析

```
1. void  fun1(int n)
2. { i= 1, k=100;
3.   while(i < n );
4.      {  k = k + 1;
5.         i = i + 2;
6.      }
7.  }
```

假设while循环语句执行f(n)，则

i=2f(n)+1 < n 推出 f(n)<=n/2 -1

故时间复杂度为 O(n)

例9 下列算法时间复杂度分析

```
1. void  fun2(int n)
2. { i= 1, s= 0;
3.   while(s < n );
4.      { i++;
5.        s = s + i;
6.      }
7.  }
```

假设while循环语句执行f(n)，则

$s=1+2+\ldots+f(n) < n$

推出 $T(n)=O(f(n))= O(n^{0.5})$

常见函数的时间复杂度按数量递增排列及增长率 (见P16图1.7)

时间复杂度T(n)按数量级递增顺序为：

常数阶	对数阶	线性阶	线性对数阶	平方阶	立方阶	…	K次方阶	指数阶
$O(1)$	$O(\log_2 n)$	$O(n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n^3)$		$O(n^k)$	$O(2^n)$

复杂度低 → **复杂度高**

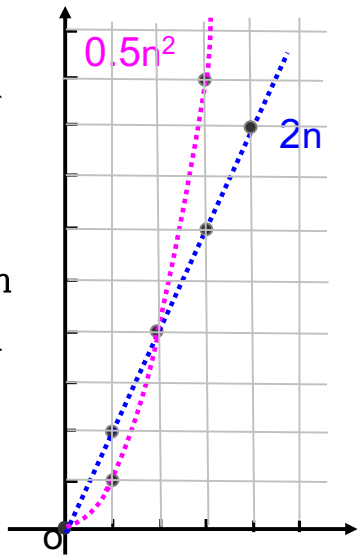
时间复杂度，衡量时间效率意义的正确理解

估算值：算法的时间估算是统计基本操作实际执行的次数，估算值不是“实际执行时间”本身。

时间复杂度：不是时间复杂度高的算法，总是需要更多的执行时间，通常与问题规模有关。

例如，两个复杂度函数2n和$0.5n^2$的，后者的阶高于前者。

它们的变化曲线如右图所示。显然，当n＞4时，$2n<0.5n^2$ 。这时，复杂度为2n的算法，才优于复杂度为$0.5n^2$的算法。



6. 算法的空间复杂度:

一个算法在计算机存储器上所占用的存储空间，包括:

(1) 存储算法本身所占用的存储空间;

(2) 算法的输入输出数据所占用的存储空间;

(3) 算法在运行过程中临时占用的存储空间。

空间复杂度(Space Complexity)是对一个算法在运行过程中临时占用存储空间大小的量度。

小结

本章重点介绍了数据结构研究对象、内容和方法，并重点讨论了数据元素存储结构和算法效率估算方法。

数据逻辑结构基本类型划分为集合结构、线性结构、树形结构和图状结构。

研究的主要内容是（1）数据的逻辑结构、（2）逻辑结构上定义的运算、（3）数据的物理结构、（4）逻辑结构与物理结构的对应关系和（5）基于物理结构的算法实现及效率分析。

提出的基本概念是数据、数据元素、数据对象、逻辑结构、关系、物理结构、数据类型、算法和复杂度等。

重点掌握的内容是①基本概念；②数据元素存储结构；③算法效率估算方法。