

理解  
数据  
结构

程序  
和  
算法

性能  
分析  
与  
度量

# 什么是数据结构?



- 数据：数据是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。
  - 数值数据，非数值性数据
- 数据对象：数据的子集。具有相同性质的数据成员（数据元素）的集合。
  - 整数数据对象  $N = \{ 0, 1, 2, \dots \}$
  - 学生数据对象

# 什么是数据结构?



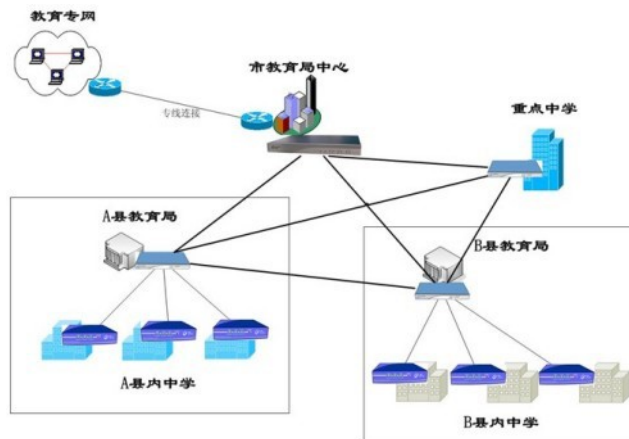
**定义：**由某一数据对象及该对象中所有数据成员之间的关系 组成。记为：

$\text{Data\_Structure} = \{D, R\}$

其中， D 是某一数据对象， R 是该对象中所有数据成员之间的关系 的有限集合

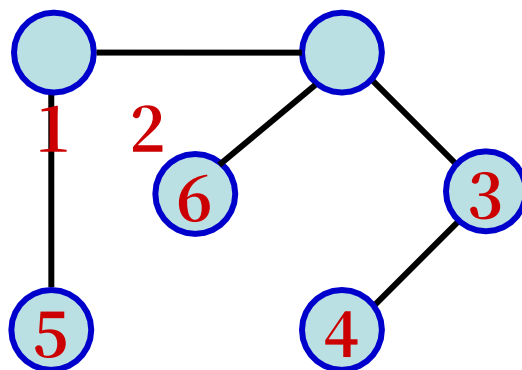
## 例子

- N 个网站之间的连接关系
- 复数的组成

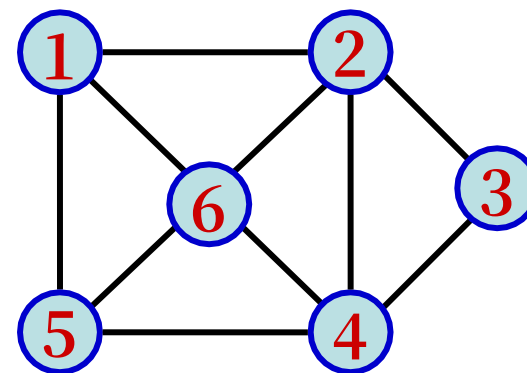


## $N$ 个网站之间的连通关系

树形关系

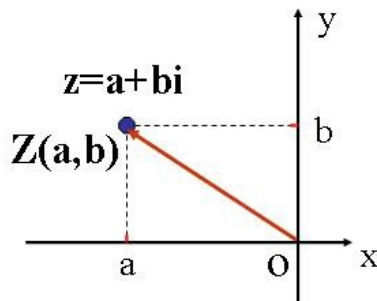


网状关系



复数 $z=a+bi$   $\longleftrightarrow$  直角坐标系中的点 $Z(a, b)$

$\longleftrightarrow$  平面向量  $\overrightarrow{OZ}$   $\longleftrightarrow$



➤ 复数的数据结构定义如下：

**Complex=(C , R)**

**C** 是包含两个实数的集合 { **C1** , **C2** }

**R={P}** , **P** 是定义在集合上的一种关系

{ **< C1** , **C2 >** }

# 数据结构是数据的组织形式

- 包括三个方面：
  - 数据元素间的逻辑关系，即数据的**逻辑结构**；
  - 数据元素及其关系在计算机存储内的表示，即数据的**存储表示**；
  - 数据的运算，即对数据元素施加的操作。

相关：

➤ 逻辑结构

➤ 物理结构

➤ 相关操作

➤ 实现



- 从逻辑关系上描述数据，与数据的存储无关；
- 看作是从具体问题抽象出来的数据模型；
- 与数据元素本身的形式、内容无关；
- 与数据元素的相对存储位置无关。

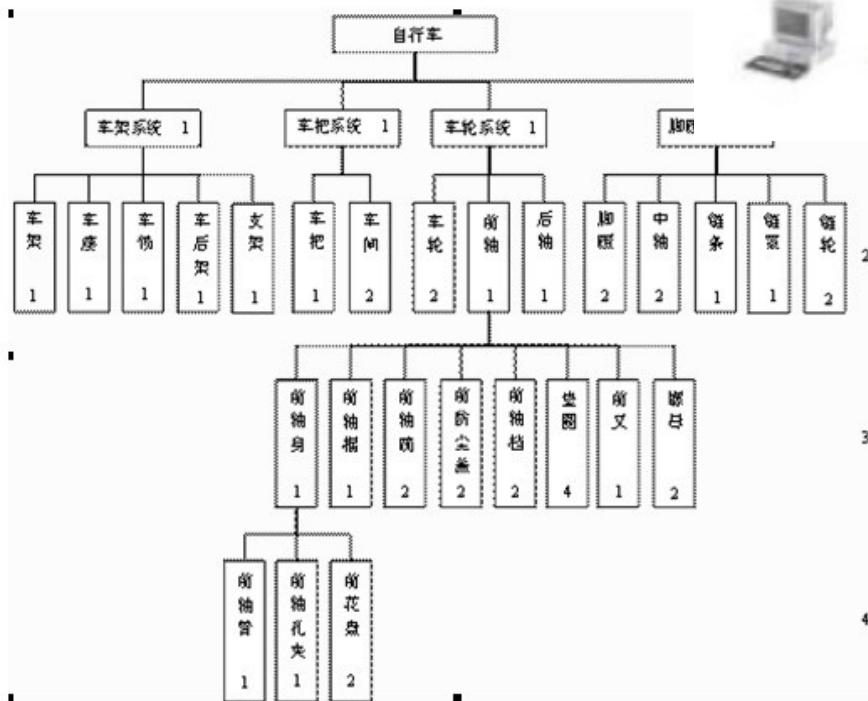
# 数据的逻辑结构分类

- 线性结构
  - 线性表
- 非线性结构
  - 树
  - 图（或网络）

# 线性结构



# 树型结构

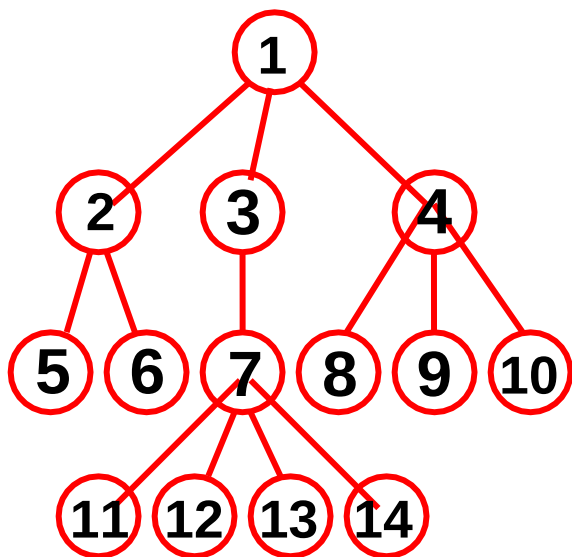


## 读取Servlet数据

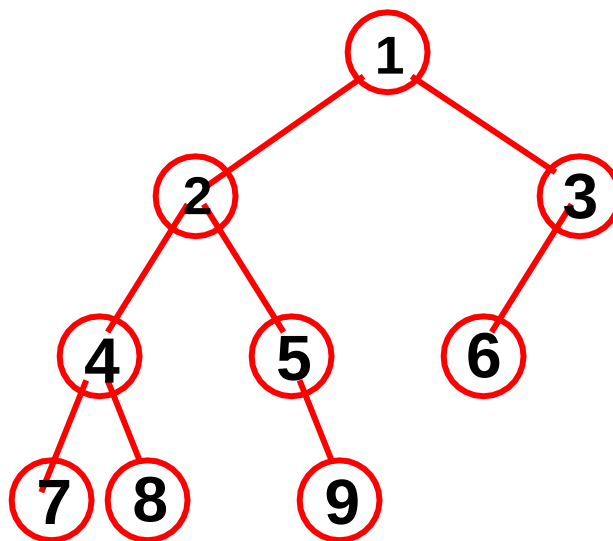
- 全部好友
  - 我的好友
    - 张三
    - 里斯
    - 王五
  - 我的同学
    - 周瑜
    - 黄盖
    - 诸葛亮
  - 陌生人

# 树型结构

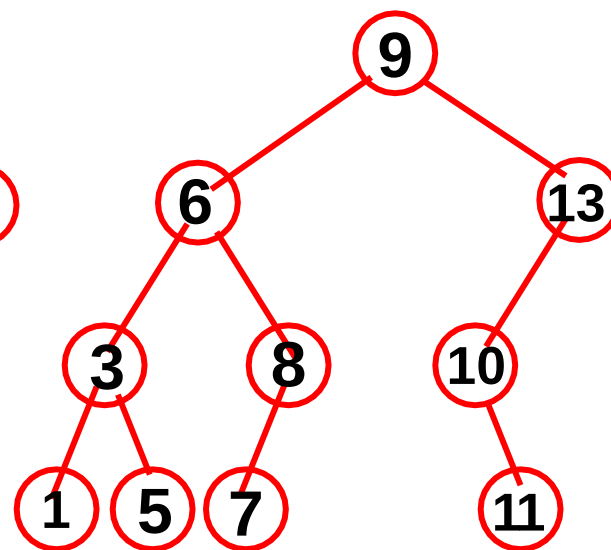
树



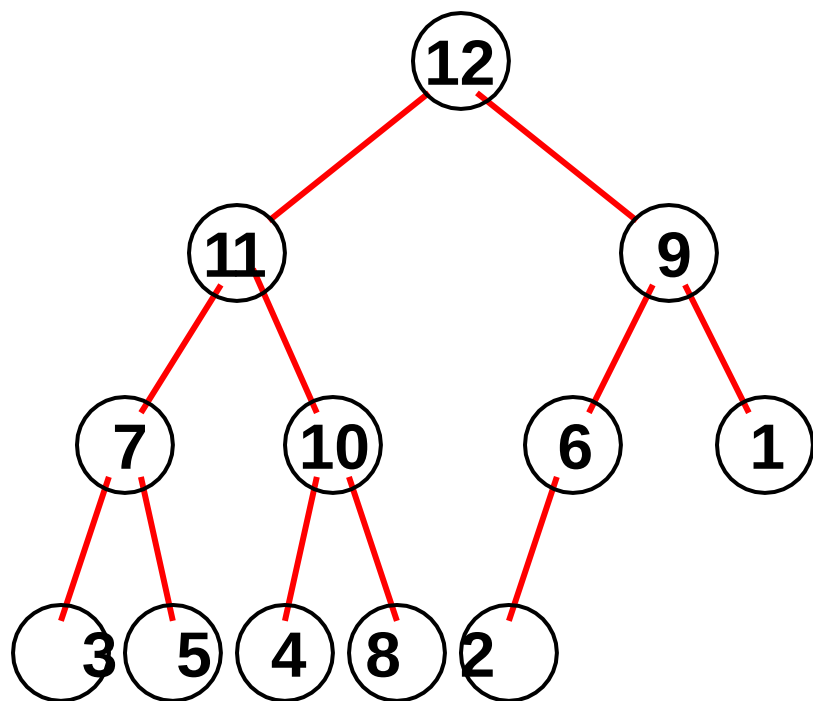
二叉树



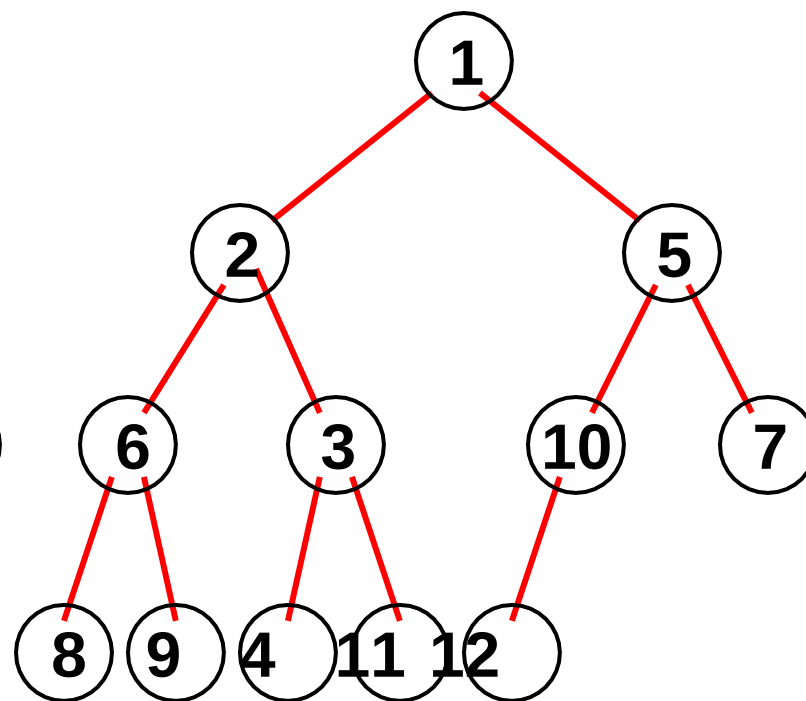
二叉搜索树



# 堆结构

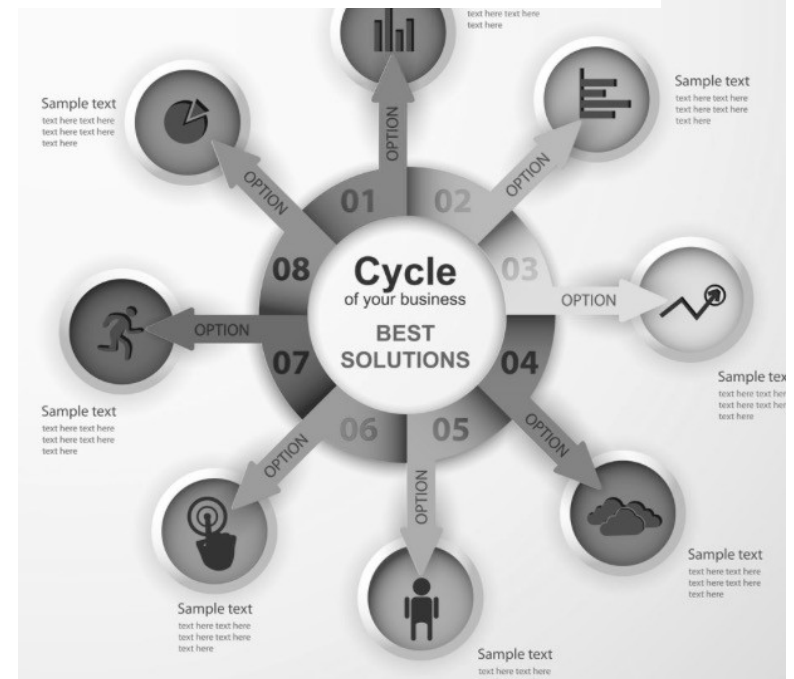
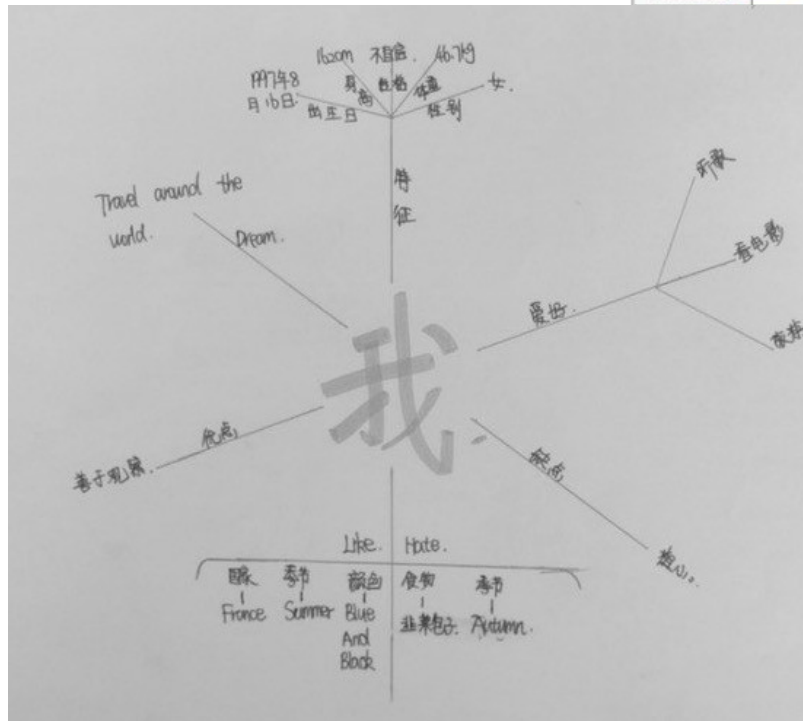
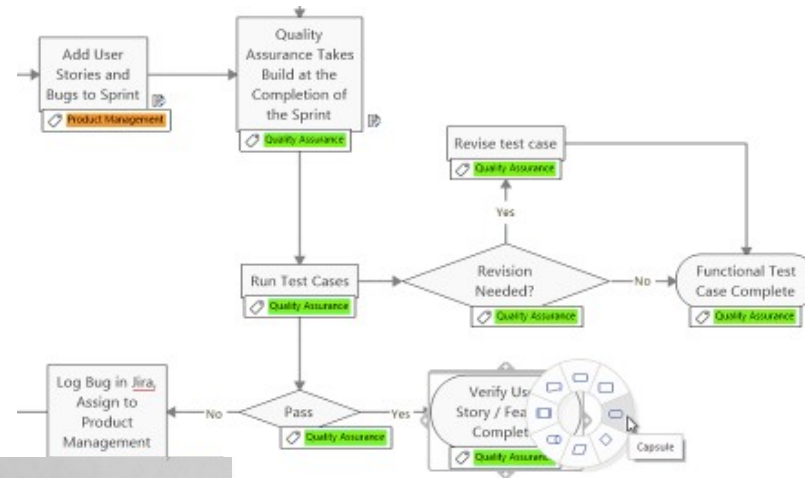


“最大”堆

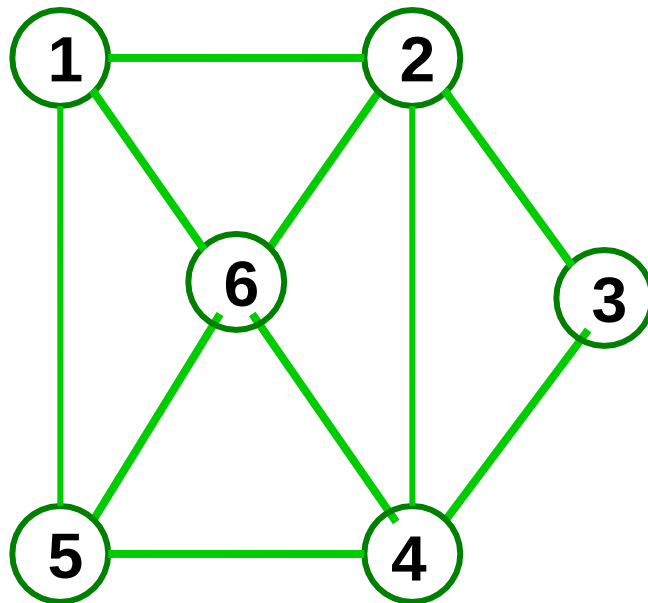


“最小”堆

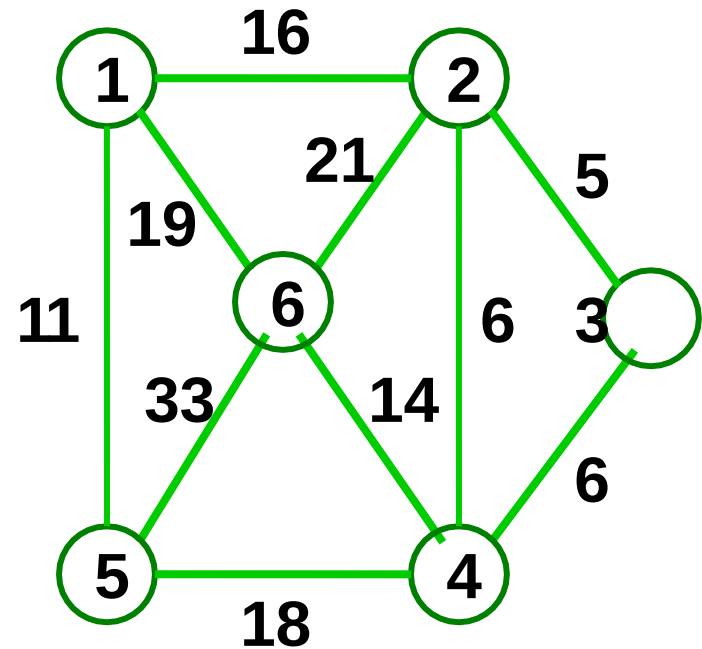
# 图结构



## 图结构



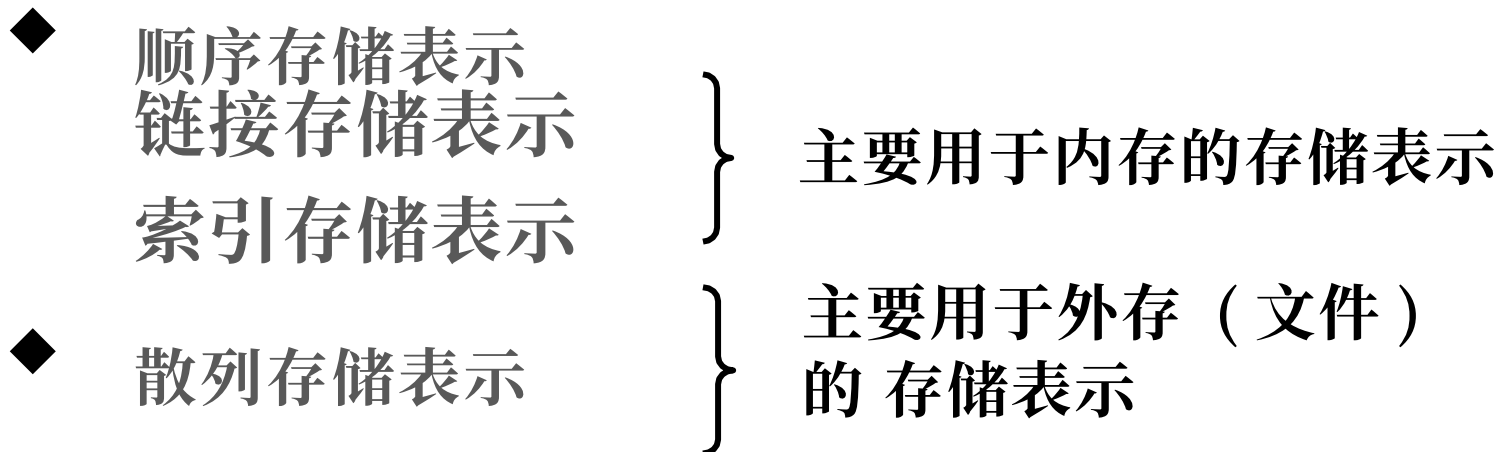
## 网络结构





**数据的存储结构是逻辑结构用计算机语言的实现；**

■ **数据的存储结构依赖于计算机语言。**



有何性 质?

如何定  
义?

如何设  
计?

➤ 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。

➤ 特性：

➤ 输入

➤ 输出

➤ 确定性

# 算法设计：自顶向下，逐步求精

## 例：选择排序问题

算法框架： 初始值  $a[k]$

```
for ( int  $i=0$ ;  $i < n-1$ ;  $i++$  ) { 从  
     $a[i]$  检查到  $a[n-1]$ ;  
                                     //  $n-1$   
                                     趟
```

```
    若最小的整数在  $a[i]$ , 交换  $a[i]$  与  $a[k]$ ;  
}
```

■ 细化程序：程序 SelectSort



Dat

```
void selectSort ( int a[ ], const int n ) {  
    // 对  $n$  个整数  $a[0], a[1], \dots, a[n-1]$ , 按非递减顺序  
    tructu 排 res 序  
    for ( int i = 0; i < n-1; i++ ) { int k = i;  
        // 从  $a[i]$  检查到  $a[n-1]$ , 找最小的整数, 在  $a[k]$   
        for ( int j = i+1; j < n; j++ )  
            if (  $a[j] < a[k]$  ) k = j;  
        // k 指示当前找到的最小整数  
        int temp = a[i]; a[i] = a[k]; a[k] = temp;  
        // 交换  $a[i]$  与  $a[k]$   
    }  
}
```

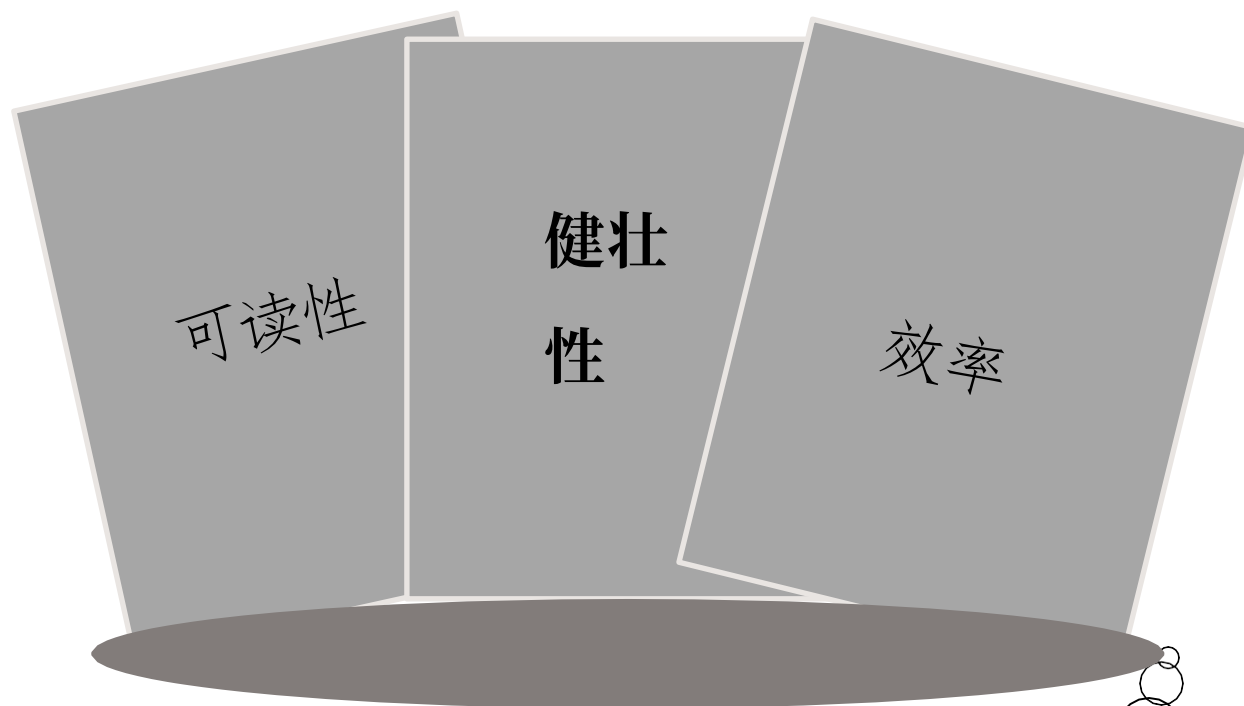
## 1.5 算法的性能分析和度量

### ➤ 正确性

#### 四个层次：

- 程序不含语法错误；
- 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；
- 程序对于精心选择的典型、苛刻而带有刁难性的几组输 入数 据能够得出满足规格说明要求的结果；
- 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...



时间代价

空间代价

➤ 后期测试

➤ 事前估计

—— 空间复杂性度量

—— 时间复杂性度量



我们对空间复杂度感兴趣的原因主要是：

- 如果一个程序要运行在一个多用户计算机系统中，我们需要指明该程序所需内存的大小。
- 运行程序，是否有足够的内存。
- 若干种解决方法，对内存的需求不同。
- 估算一个程序所解决的问题是什么规模的。

例 1

求表达式  $a+b+b*c+(a+b-c)/(a+b)+4.0$

```
float abc (float a, float b, float c ) {  
    return a+b+b*c+(a+b-c)/(a+b)  
    +4.0  
}
```

例 2 以迭代方式求累加和

```
float sum ( float a[ ], const int n )  
{  
    float s = 0.0;  
    for ( int i = 0; i < n; i++ )  
        s += a[i];  
  
    return s;  
}
```

```
float rsum(float a[ ],const int n) {  
    if (n<=0) return 0;  
    else return rsum(a,n-1)+a[n-1]  
}
```

空间复杂度是指当问题的规模以某种单位从 1 增加到  $n$  时，解决这个问题的算法在执行时所占用的存储空间也以某种单位由 1 增加到  $S(n)$

## 空间复杂度的组成

### 程序所需的空間

#### 1、指令空间

编译之后程序指  
令所需的存 储  
空间

#### 2、数据空间

- 常量
- 动态空间

#### 3、环境栈空间

调用和恢复函 数  
或方法时所 需  
要的空间

# 运行时间

## 估算运行时间

### ➤ 程序步

语法上或语义上有意义的一段指令序列

执行时间与实例特性无关

**例如：**

- **注释：程序步数为 0**
- **声明语句：程序步数为 0**
- **表达式：程序步数为 1 ，   return  
a+b+b\*c**
- **赋值语句   < 变量 >=< 表达式 >**

## ➤ 循环语句（循环控制部分，执行语句）

**for(< 初始化语句 >;< 表达式 1>;< 表达式 2>)**  
**... while < 表达式 >do...**  
**do ...while < 表达式 >**

➤ **switch 语句**

➤ **if then**

➤ **函数执行语句：**

➤ **动态存储管理： new delete sizeof( )**

➤ **转移语句： continue,break,goto,return**

## ■ 程序步确定方法

—— 插入计数全局变量 *count*

### 例 以迭代方式求累加和的函数

```
行 float sum ( float a[ ], const int n )  
1 {  
2     float s = 0.0;  
3 for ( int i = 0; i < n; i++ ) 4     s +=  
a[i];  
5     return s; 6 }
```



```

float sum ( float a[ ], const int n ) {
    float s = 0.0;
    count+2; //count 统计执行语句条数
    for ( int i = 0; i < n; i++ ) {
        count+2;    // 针对 for 语
        s += a[i];   句
        count++;
    } // 针对赋值语句
    count+2; // 针对 for 的最后一次
    count++; // 针对 return 语
    句 return s;

```

注意：

一个语句本身的程序步数可能不等于该语句一次执行所具有的程序步数。

例如：赋值语句

$x = \text{sum}(R, n);$

— 建表，列出逐个语句的程序步

## 例：计算累加和程序

| 程序语句                      | 一次执行所需程序步数 | 执行频度 | 程序步数 |
|---------------------------|------------|------|------|
| {                         | 0          | 1    | 0    |
| float s = 0.0;            | 1          | 1    | 1    |
| for ( int i=0; i<n; i++ ) | 2          | n+1  | 2n+2 |
| s += a[i];                | 1          | n    | n    |
| return s;                 | 1          | 1    | 1    |
| }                         | 0          | 1    | 0    |
| 总程序步数                     |            |      | 3n+4 |



```
void selectSort ( int a[ ], const int n ) {  
// 对  $n$  个整数  $a[0], a[1], \dots, a[n-1]$ , 按非递减顺序  
排序
```

```
    for ( int i = 0; i < n-1; i++ ) { int k = i;  
    // 从 a[i] 检查到 a[n-1], 找最小的整数, 在  
    a[k] for ( int j = i+1; j < n; j++ )  
    if ( a[j] < a[k] ) k = j;  
    //k 指示当前找到的最小整数  
    int temp = a[i]; a[i] = a[k];  a[k] =  
    temp;  
    // 交换 a[i] 与 a[k]
```

```
}  
}
```



## 描述最坏情况下的时间代价

定义：如果存在正整数  $c$  和  $n_0$ ，使得当  $n \geq n_0$  时， $T(n) \leq cf(n)$ ，则记为  $T(n) = O(f(n))$

例：( 线性函

数)

考察  $f(n)=3n+2$

例：平方函数

考察

$f(n)=10n^2+4n+2$

例：设有两个算法在同一机器上运行，其执行时间分别为  $100n^2$  和  $2^n$ ，问：要使前者快于后者， $n$  至少要取多大？

解答：

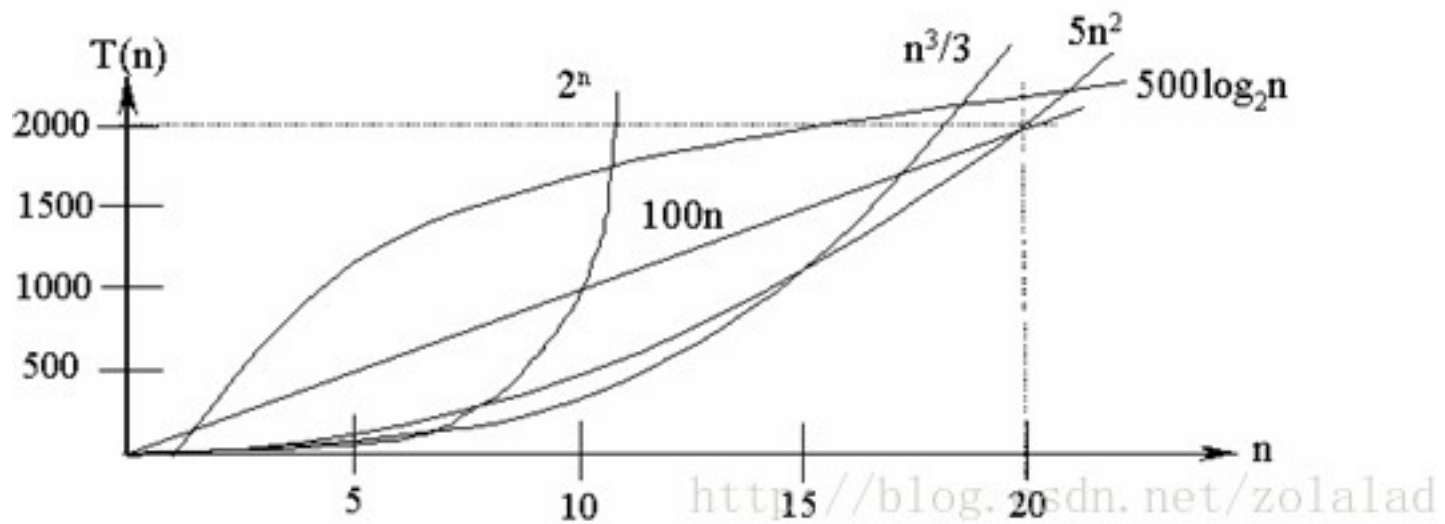
问题是找出满足  $100n^2 \leq 2^n$  的最小的  $n$ 。用试探法：

$n = 13$  时，  $100n^2 = 16900 > 2^n = 8192$

$n = 14$  时，  $100n^2 = 19600 > 2^n = 16384$   $n =$

$15$  时，  $100n^2 = 22500 < 2^n = 32768$

取  $n = 15$  满足要求。





## 大 O 表示

法：

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

## 使用大 ( O ) 表示法:

对于多项式，我们只保留最高次幂的项，常数系数和低阶项可以不要。

$$\log_2 n \quad O(\log_2 n) \longrightarrow$$

加法法则，针对并程序

$$\begin{aligned} T(n, m) &= T1(n) + T2(m) \\ &= O(\max(f(n), g(m))) \end{aligned}$$

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

$$\begin{aligned}T(n, m) &= T1(n) * T2(m) \\ &= O(f(n)*g(m))\end{aligned}$$

特例：若  $T1(n)=O(c)$ ,  $c$  与  $n$  无关的任意常数,  $T2(n)=O(f(n))$ ,  
则  $T(n)=O(f(n))$

任何非 0 正常数都属于同一数量级, 记为  $O(1)$

```

void example (float x[ ][ ], int m, int n, int k)
{
    float sum [ ];
    for ( int i = 0; i < m; i++ ) {
        sum[i] = 0.0;
        for ( int j=0; j<n; j++ )
            sum[i] += x[i][j];
    }
    for ( i = 0; i < m; i++ )
        cout << “Line” << i <<
        “:” << sum [i] << endl;
}
    
```

```
template <class Type>
void dataList<Type>::bubbleSort ( )
{ int i = 1; int exchange = 1;
    // 当 exchange 为 0 则停止排序
    while ( i < ArraySize && exchange ) {
        BubbleExchange ( i, exchange );
        i++;
    } // 一趟比较
}
```

```
template <class Type>
void
```

```
BubbleSort<Type> BubbleSort<Type>(const int i, int & exchange ){
    exchange = 0; // 假定元素未交换    for ( int j
    = ArraySize-1; j>=i; j-- )
        if ( Element[j-1] > Element[j] ) {
            Swap ( j -1, j ); // 发生逆序
            // 交换 Element[j-1] 与 Element[j]
            exchange = 1; // 做 “发生了交换” 标志
        }
    }
```



## BubbleSort n-1 趟

BubbleExchange ( ) n-i 次比较

$n-1$

$i =$   
 $1$

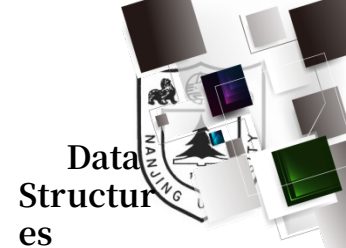
2

渐进时间复杂度

$$O(f(n)*g(n)) = O(n^2)$$

$$S(n) = O(f(n))$$





## 复习

## 1、数据抽象与封装

抽象

封装

面向对象

程序设计

## 2、类

➤ 类

➤ 数据成员

➤ 成员函数

## 3、对象

对象的创

建和标识

对象的初

始化

49

# 讨论环节

## Google 面试题

有一个 random number generator，是生成真实的随机数，而不是伪随机数，会生成几千亿个 32 位整数，打印出现次数 100 的整数。