# 第一部分：框架

## 基本思路

本章介绍一个贯穿本书的框架，后续算法的设计与分析都是在这个框架中进行的。

框架就是：

算法伪代码 —> 算法正确性的证明 —> 算法时间复杂度的分析

**接下来记录的就是以上框架各个阶段对应的方法**

1. 部分求和公式见附录A
2. 插入排序的python下的实现

# coding=utf-8  
**def Insertion\_sort**(sort\_list):  
 lenth = len(sort\_list) # 这里sort\_list第一个元素充当哨兵  
 i = 2  
 **while** i < lenth:  
 **print** sort\_list  
 sort\_list[0] = sort\_list[i]  
 j = i-1  
 **while** sort\_list[0] < sort\_list[j]:  
 sort\_list[j+1] = sort\_list[j]  
 j -= 1  
 sort\_list[j+1] = sort\_list[0]  
 # print sort\_list  
 i += 1  
 **print** u"插入排序完成！"  
# return sort\_list  
  
sort\_list = [1, 2, 34, 12, 190, 990, 123]  
**print** sort\_list  
**print** u"开始排序！"  
sort\_list.insert(0, 0) # 第一个位置用作哨兵  
Insertion\_sort(sort\_list)  
**del** sort\_list[0]  
**print** sort\_list

## 伪代码中的约定

1. 缩进表示块结构
2. 退出循环后，循环计数器保持退出时的值。
3. 注释用 //
4. 支持多重赋值 I = j = e，这里i，j是两个对象，和python不同
5. 只使用局部变量，不适用全局变量
6. 数组通过下标访问
7. 符合数据通常被组织成对象，对象又由属性组成。

例：数组A看成对象，有A.lenth

我们把表示一个数组或对象的变量看做指向表示数组或对象数据的一个指针。

我们的属性记号可以“串联”。例如：y = x.f 那么x.f.g = y.g

有事一个指针根本不指向任何对象，这时，我们赋给它特殊值**NIL**

1. 我们按值把参数传递给过程：

当对象被传递时，指向表示对象数据的指针被复制

当非对象被传递时，仅传递了一个副本

1. return 语句可以返回多个值
2. 布尔运算符 and 和 or 都是**短路**的

x and y 若x 为假 则不计算y

1. 关键词error 表示因为已被调用的过程情况不对而出现了一个错误。调用过程负责处理该错误，所以我们不用说明将采取什么行动。

## 算法正确性证明：循环不变式

每次循环从数组A中取出第j个元素插入有序区A[1 .. j-1]，然后递增j。这样A[1 .. j-1]的有序性始终得到保持，这就是所谓的"**循环不变 (loop invariant)**"了。

**需要证明三点：**

1. 初始化：循环的第一次迭代之前，它为真
2. 保持：如果循环的某次迭代之前他为真，那么下次迭代之前它为真
3. 终止：再循环终止时，不变式为我们提供一个有用的性质，该性质有助于证明算法是正确的

例如：

1. 初始化（循环第一次迭代之前）的时候，*A*[1 ‥ *j* -1]的“有序性”是成立的；  
2. 在循环的每次迭代过程中，*A*[1 ‥ *j* -1]的“有序性”仍然保持；  
3. 循环结束的时候，*A*[1 ‥ *j* -1]的“有序性”仍然成立。

## 算法性能分析（时间复杂度数量级的求解）

### 分治法

**将原问题分解为几个规模较小的但类似于原问题的子问题，递归的求解这些子问题，然后再合并这些子问题的解来建立原问题的解。**

**分治模式：**



### 计算方法

**若算法的时间复杂度可以描述成递归式，那么可以用以下三个方法来求解其O或 Θ渐进界：**

**（这三个方法是递进式的，不是并列的，通常最先使用主方法，否则使用递归树法，代入法需要提前假设（看出来的），所以代入法多用来验证正确性——**

#### 主方法

****

****

****

#### 递归树法（用来计算算法时间复杂度的方法）：

将递归式转换为一棵树，其节点表示不同层次的递归调用产生的代价。然后采用边界和技术来求解递归式。（中文版第三版51页）

#### 代入法

我们猜测一个界，然后用数学归纳法证明这个界是正确的。