

《算法设计与分析》

算法设计与分析

马丙鹏

2024年09月02日



中国科学院大学

University of Chinese Academy of Sciences

第一章 算法引论

- 1.1 算法的定义及特性
- 1.2 复杂性分析初步
- 1.3 递归



1.2 复杂性分析初步

■ 1. 分析算法的目的

□ 算法选择的需要

➤ 对同一个问题可以设计不同的算法，不同的算法对时间和空间需求是不同的。

通过对算法的分析可以了解算法的性能，从而在不同算法之间比较性能的好坏，选择好的算法，避免无益的人力和物力浪费。

□ 算法优化的需要

➤ 有没有可以改进的地方，以使算法工作的更好？

➤ 通过对算法分析，可以对算法作深入了解，从而可以进一步优化算法，让其更好地工作。



1.2 复杂性分析初步

■ 2. 重要的假设和约定:

① 计算机模型的假设

- 计算机形式理论模型：有限自动机，图灵机
- RAM模型（**random-access machine**，随机访问机）
 - ✓ 在RAM模型中，指令一条一条被取出和执行，没有并发操作。
- 通用的顺序计算机模型：
 - ✓ 单CPU--串行算法，部分内容涉及多CPU（核）和并行处理的概念
 - ✓ 有足够的“内存”，并能在固定的时间内存取数据单元
- 区分计算机的理论模型与实际的计算机



1.2 复杂性分析初步

■ 2. 重要的假设和约定:

② 计算的约定

- 算法的执行时间是算法中所有运算执行时间的总和，可以表示为：

$$\text{算法的执行时间} = \sum F_i * t_i$$

- F_i 是运算 i 的执行次数，称为该运算的**频率计数**，仅与算法的控制流程有关，与实际使用的计算机硬件和编制程序的语言无关。
- t_i 是运算 i 在实际的计算机上每执行一次所用的时间，与程序设计语言和计算机硬件有关。



1.2 复杂性分析初步

■ 2. 重要的假设和约定:

机器	运算速度	运行算法	运行时间
天河三号	百亿亿次/秒	插入排序	无法公平比较
个人电脑	十亿次/秒	选择排序	



分析算法的运行时间应独立于机器

1.2 复杂性分析初步

■ 2. 重要的假设和约定:

② 计算的约定

➤ 运算的分类: 依照运算的**时间特性**, 将运算分为**时间囿 (you) 界于常数的运算**和**时间非囿界于常数的运算**。

➤ **时间囿界于常数的运算:**

✓ 基本算术运算, 如整数、浮点数的加、减、乘、除字符运算、赋值运算、过程调用等

✓ **特点:** **执行时间是固定量**, 与操作数无关。

✓ 例: $1+1=2$ vs $10000+10000=20000$

✓ $100*100=10000$ vs $10000*10000=100000000$

✓ CALL INSERTIONSORT



中国科学院大学

University of Chinese Academy of Sciences 7

1.2 复杂性分析初步

■ 2. 重要的假设和约定:

② 计算的约定

➤ 时间囿界于常数的运算:

- ✓ 设有 n 种运算 c_1, c_2, \dots, c_n , 它们的实际执行时间分别是 t_1, t_2, \dots, t_n 。
- ✓ 令 $t_0 = \max(t_1, t_2, \dots, t_n)$, 则每种运算执行一次的时间都可以用 t_0 进行限界（上界）。
- ✓ 视 t_0 为一个单位时间, 则这些运算“理论”上都可视为仅花一个固定的单位时间 t_0 即可完成的运算
- ✓ 具有这种性质的运算称为时间囿界于常数的运算。



1.2 复杂性分析初步

■ 2. 重要的假设和约定:

② 计算的约定

➤ 时间非囿界于常数的运算:

✓特点: 运算的执行时间与操作数相关, 每次执行的时间是一个不定的量。

✓如: 字符串操作: 与字符串中字符的数量成正比, 如字符串的比较运算`strcmp`。

记录操作: 与记录的属性数、属性类型等有关



1.2 复杂性分析初步

■ 2. 重要的假设和约定:

② 计算的约定

➤ 时间非囿界于常数的运算:

- ✓ 这类运算通常是由更基本的运算组成的。而这些基本运算是时间囿界于常数的运算
- ✓ 时间非囿界于常数的运算的一次执行时间是其包含的所有基本运算的执行时间之和。
- ✓ 如：字符串比较时间 t_{string}
$$t_{\text{string}} = \text{Length}(\text{String}) * t_{\text{char}}。$$
- ✓ 故：分析时间非囿界于常数的运算时，将其分解成若干时间囿界于常数的运算即可。



1.2 复杂性分析初步

■ 2. 重要的假设和约定:

② 计算的约定

- 引入“单位时间”的概念，将“单位时间”视为“1”，
从而将算法的理论执行时间转换成“单位时间”
意义下的执行次数，
从而实现“执行时间”与时间复杂度概念的统一。
- 算法的执行时间 = $\sum F_i * t_i$



1.2 复杂性分析初步

■ 2. 重要的假设和约定:

③ 工作数据集的选择

➤ 算法的执行情况与数据的特征相关:

✓ 算法的执行时间与输入数据的规模相关，一般规模越大，执行时间越长。

● 如：插入算法，10个数的排序时间显然小于1000个数的排序时间

✓ 在不同的数据配置上，同一算法有不同的执行情况，可分为最好、最坏和平均等情况讨论。

➤ 编制不同的数据配置，分析算法的最好、最坏、平均工作情况是算法分析的一项重要工作

✓ 测试数据集的生成在目前算法证明与程序正确性证明没有取得理论上的突破性进展的情况下，是程序测试与算法分析中的关键技术之一。



1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 全面分析算法的两个阶段：事前分析：

- 通过对算法执行性能的理论分析，试图得出关于算法执行特性的一种形式描述，以“理论上”衡量算法的“好坏”。
- 与计算机物理软硬件没有直接关系。
- 时间分析：
 - ✓ 了解算法中使用了那些运算(具有单位执行时间)。
 - ✓ 分析程序的控制流程，从而确定各类运算的执行次数。
 - ✓ 将对运算执行次数的分析结果表示成问题规模 n 的特征函数。
 - ✓ 分析典型的数据配置，了解算法在最好、平均、最坏等情况下的执行情况。



1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 全面分析算法的两个阶段：事前分析：

➤ 空间分析：

- ✓ 分析算法中各类变量的定义情况
- ✓ 分析算法的嵌套结构中变量的使用情况
- ✓ 将空间占用量表示成为问题规模 n 的特征函数
- ✓ 分析典型的数据配置，了解算法在最大、平均、最少等情况下的空间占用情况



1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 全面分析算法的两个阶段：事后测试：

- 将算法编制成程序后实际放到计算机上运行，收集其执行时间和空间占用等统计资料，进行分析判断
- 直接与物理实现有关
- 验证先前的分析结论——包括正确性、执行性能等，比较、优化所设计的算法
- 分析手段：作时、空性能分布图

□ 算法分析主要集中于与物理实现无关的事前分析阶段——获取算法时间/空间复杂度函数



1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--如何给出反映算法执行特性的描述？

➤ 最直接方法：统计算法中各种运算的执行情况，包括：

① 运用了哪些运算

✓ 基本运算：时间囿界于常数的运算

✓ 复合运算：具有固定执行时间的程序块，如一条语句、一个过程，甚至一个函数，由基本运算组成。

② 每种运算被执行的次数

③ 该种运算执行一次所花费的时间等。

➤ 算法的执行时间 = $\sum F_i * t_i$



中国科学院大学

University of Chinese Academy of Sciences 16

1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--频率计数

- 一条语句或运算在算法(或程序)体中的执行次数。
- 顺序结构中的运算/语句：执行次数计为1
- 嵌套结构中的运算/语句：执行次数等于被循环执行的次数

```
begin  
   $x \leftarrow x + y$   
end
```

FC: 1

```
begin  
  for  $i \leftarrow 1$  to  $n$  do  
     $x \leftarrow x + y$   
  end
```

FC: n

```
begin  
  for  $i \leftarrow 1$  to  $n$  do  
    for  $j \leftarrow 1$  to  $n$  do  
       $x \leftarrow x + y$   
    end
```

FC: n^2



1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--频率计数

➤ 例：寻找数组中最大元素

```
template <class T>
int Max(T a[], int n)
{ //寻找a[0:n-1]中的最大元素
    int pos=0;
    for (int i=1; i<n; i++) —————→ n-1次
        if (a[pos]<a[i]) —————→ n-1次
            pos=i; —————→ 最多n-1次,最少0次
    return pos;
}
```






1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--频率计数

➤ 例: n 次多项式求值程序

```
template <class T>
T PolyEval(T coeff[], int n, const T& x)
{ // 计算  $n$  次多项式的值, coeff[0:n] 为多项式的系数
  T y=1, value=coeff[0];
  for (int i=1; i<=n; i++)   $n$  循环
  { // 累加下一项
    y*=x;   $n$  次乘法
    value+=y*coeff[i];   $n$  次加法和  $n$  次乘法
  }
  return value;
}
```

$3n$ 次基本运算

1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--频率计数

➤ 例: 计算名次

这里的关键操作是比较，次数为： $1+2+\dots+n-1=n(n-1)/2$

```
template <class T>
void Rank(T a[ ], int n, int r[ ])
{ // 计算a[0:n-1]中元素的排名
    for(int i=0; i<n; i++)
        r[i]=0; // 初始化
    // 逐对比较所有的元素
    for(int i=1; i<n; i++)
        for(int j=0; j<i; j++)
            if(a[j]<=a[i]) r[i]++;
            else r[j]++;
}
```

1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--频率计数

➤ 例:选择排序

```
template<class T>
void Selectionsort(T a[], int n)
{//对数组a[0:n-1]中元素排序
    for(int k=n; k>1; k--)
        { int j=Max(a,k);
          Swap(a[j],a[k-1]);}
}
```

3次赋值

```
template<class T>
void Swap(T& a, T& b)
{ T temp=a; a=b; b=temp; }
```



1.2 复杂性分析初步

■ 3. 如何进行算法分析?

□ 事前分析--频率计数

➤ 例:冒泡排序

最坏情形下, 关键操作数: $\sum 4(i-1)=2n(n-1)$

一次比较, 3次赋值

```
template<class T>
void BubbleSort (T a[ ], int n)
{
    for(int i=n; i>0; i--)
        Bubble (a, i);
}
```

一次冒泡

```
template<class T>
void Bubble (T a[ ], int n)
{
    for (int i=0; i<n-1; i++)
        if (a[i]>a[i+1])
            Swap(a[i],a[i+1]);
}
```

```
template<class T>
void Swap(T& a, T& b)
{ T temp=a; a=b; b=temp; }
```

1.2 复杂性分析初步

■ 3. 如何进行算法分析？

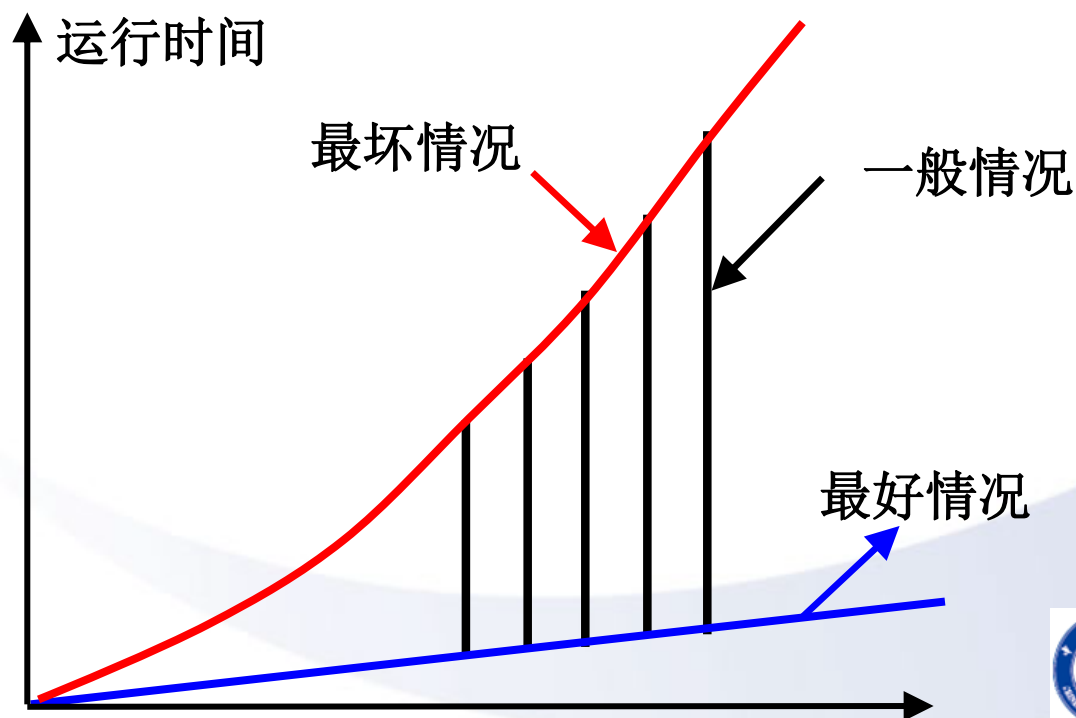
□ 事前分析--算法的最坏情况执行时间

- 一般更关心算法的**最坏情况**执行时间。
- 算法的最坏情况执行时间给出了任何输入的运行时间的一个**上界**。
- 知道了这个界，就能确保算法绝不需要更长的时间，我们就不需要再对算法做更坏的打算。
- 对某些算法，最坏情况经常出现。
- 对很多算法，平均情况往往与最坏情况大致一样
 - ✓ 如插入排序，就一般情况而言，`while`循环中为确定当前 $A[j]$ 的插入位置，平均要检查一半的元素。那么导致的平均执行时间就数量级来说和最坏情况一样，依然是 n 的二次函数，只是常数系数小了一些。



1.2 复杂性分析初步

输入情况	情况说明
最好情况	不常出现，不具普遍性
最坏情况	确定上界，更具一般性
一般情况	情况复杂，分析难度大



常用最坏情况分析
算法运行时间



1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--函数表达式的数量级：

- 函数表达式中的最高次项的次数称为函数表达式的数量级（阶）
- 在频率计数函数表达式中，数量级是衡量频率计数的“大小”的一种测度。
- 数量级从本质上反映了算法复杂度的高低
 - ✓ 数量级越小，算法的复杂度越低，同等规模下算法的执行时间越短。
 - ✓ 数量级越大，算法的复杂度越高，同等规模下算法的执行时间越长。例：假如求解同一个问题的三个算法分别具有 n ， n^2 ， n^3 数量级。
- 若 $n=10$ ，则可能的执行时间将分别是10，100，1000个单位时间——与环境因素无关。



1.2 复杂性分析初步

■ 3. 如何进行算法分析？

□ 事前分析--算法的输入规模

- 算法的执行时间随问题规模的增长而增长，增长的速度随不同的算法而不同
- 没有一个方法可以准确的计算算法的具体执行时间：语言、编译系统、计算机
- 实际上，在评估算法的性能时，并不需要对算法的执行时间作出准确的统计
- 人们希望算法与实现的语言无关、与执行的计算机无关
- 我们所关心的是：算法的执行时间，随着输入规模的增长而增长的情况



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

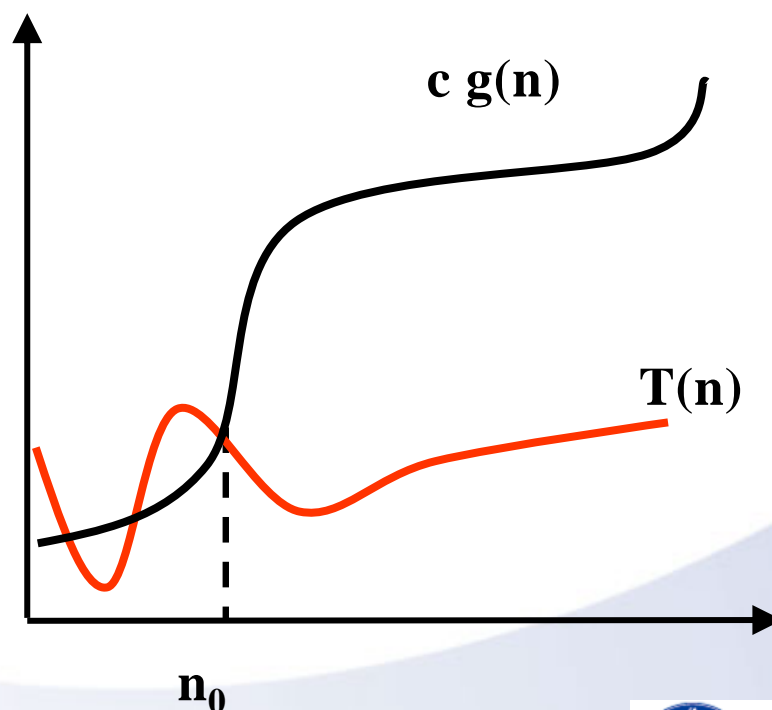
- 记：算法的**计算时间为 $f(n)$** ，数量级**限界函数为 $g(n)$** ，其中， n 是输入或输出规模的某种测度。
- $f(n)$ 表示算法的“实际”执行时间——与机器及语言有关。
- $g(n)$ 是形式简单的函数，如 n^m ， $\log n$ ， 2^n ， $n!$ 等。是事前分析中通过对计算时间或频率计数统计分析所得的、与机器及语言无关的函数。
- 以下给出算法执行时间：上界(O)、下界(Ω)、平均(Θ)的定义。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 定义1 如果存在两个正常数 c 和 n_0 , 对于所有的 $n \geq n_0$, 有 $|f(n)| \leq c|g(n)|$, 则记作 $f(n) = O(g(n))$



$$T(n) = O(g(n))$$



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 含义:

- 一个算法具有 $O(g(n))$ 的计算时间是指如果算法用 n 值不变的同一类数据在某台机器上运行时，**所用的时间总是小于 $|g(n)|$ 的一个常数倍**。
- $g(n)$ 是计算时间 $f(n)$ 的一个上界函数。 $f(n)$ 的数量级就是 $g(n)$ 。
- $f(n)$ 的增长最多像 $g(n)$ 的增长那样快。
- 试图**求出最小的 $g(n)$** ，使得 $f(n) = O(g(n))$ 。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 多项式定理:

定理1 若 $A(n) = a_m n^m + \dots + a_1 n + a_0$ 是一个 m 次多项式, 则有 $A(n) = O(n^m)$

即: 变量 n 的固定阶数为 m 的任一多项式, 与此多项式的最高阶 n^m 同阶。

证明: 取 $n_0=1$, 当 $n \geq n_0$ 时, 有

$$\begin{aligned} |A(n)| &\leq |a_m|n^m + \dots + |a_1|n + |a_0| \\ &\leq (|a_m| + |a_{m-1}|/n + \dots + |a_0|/n^m) n^m \\ &\leq (|a_m| + |a_{m-1}| + \dots + |a_0|) n^m \end{aligned}$$

令 $c = |a_m| + |a_{m-1}| + \dots + |a_0|$

则, 定理得证。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 计算时间的数量级对算法有效性的影响

- 数量级的大小对算法的有效性有决定性的影响。
- 例：假设解决同一个问题的两个算法，它们都有 n 个输入，计算时间的数量级分别是 n^2 和 $n\log n$ 。则，
 $n=1024$ ：分别需要1048576和10240次运算。
 $n=2048$ ：分别需要4194304和22528次运算。
- 分析：在 n 加倍的情况下，一个 $O(n^2)$ 的算法计算时间增长4倍，而一个 $O(n\log n)$ 算法则只用两倍多一点的时间即可完成。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 多项式时间算法:

- 可用多项式(函数)对其计算时间限界的算法。常见的多项式限界函数有:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)$$

□ 指数时间算法:

- 计算时间用指数函数限界的算法, 常见的指数时间限界函数:

$$O(2^n) < O(n!) < O(n^n)$$

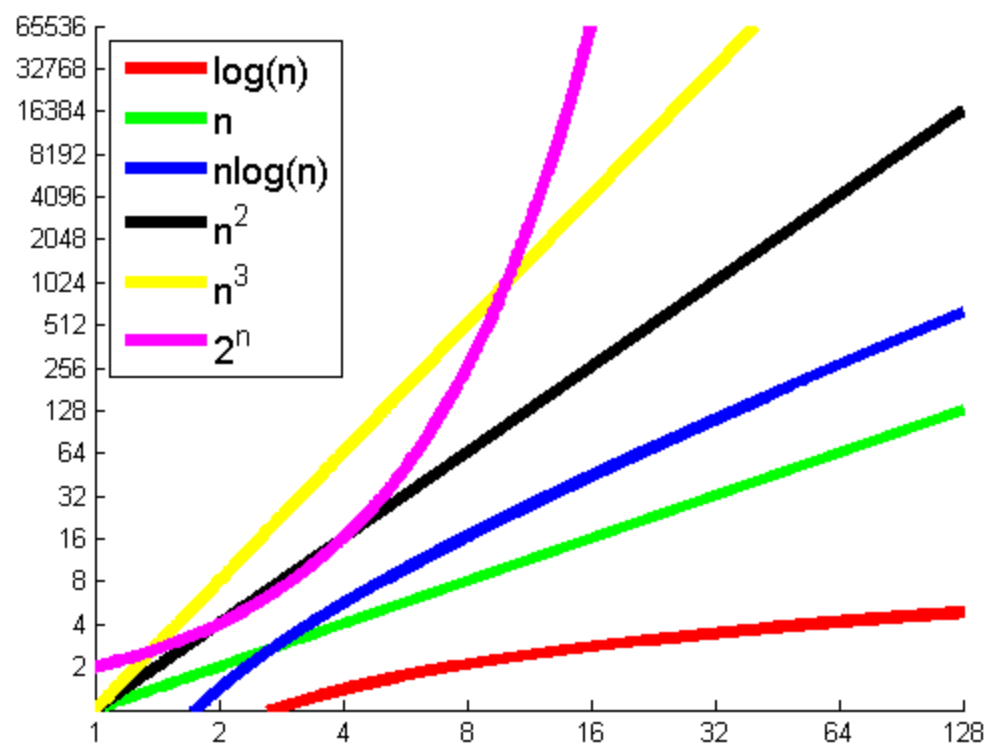
- 说明: 当 n 取值较大时, 指数时间算法和多项式时间算法在计算时间上非常悬殊。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 典型的计算时间函数曲线



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 计算时间函数值比较

- $0.999^{10000} = 0.00004517$
- $1.001^{10000} = 21916.6813$

- 运行具有比 $O(n \log n)$ 复杂度还高的算法是比较困难的。
- 指数时间算法只有在 n 取值非常小时才实用。
- 降低算法的计算复杂度高于提高计算机的速度。

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296



1.2 复杂性分析初步

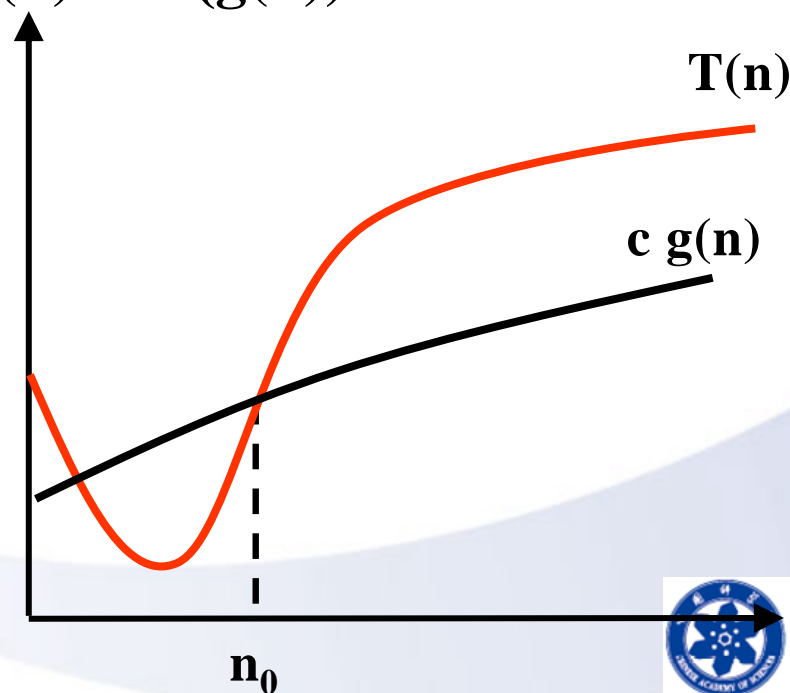
■ 4. 计算时间的渐近表示

□ 下界函数:

□ 定义1.2 如果存在两个正常数 c 和 n_0 , 对于所有的 $n \geq n_0$, 有

$$|f(n)| \geq c|g(n)|$$

则记作 $f(n) = \Omega(g(n))$



$$T(n) = \Omega(g(n))$$



中国科学院大学

University of Chinese Academy of Sciences 35

1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 下界函数:

□ 含义:

- 如果算法用 n 值不变的同一类数据在某台机器上运行时，所用的时间总是不小于 $|g(n)|$ 的一个常数倍。所以 $g(n)$ 是计算时间 $f(n)$ 的一个下界函数。
- $f(n)$ 的增长至少像 $g(n)$ 的增长那样快
- 试图求出最大的 $g(n)$ ，使得 $f(n) = \Omega(g(n))$ 。



1.2 复杂性分析初步

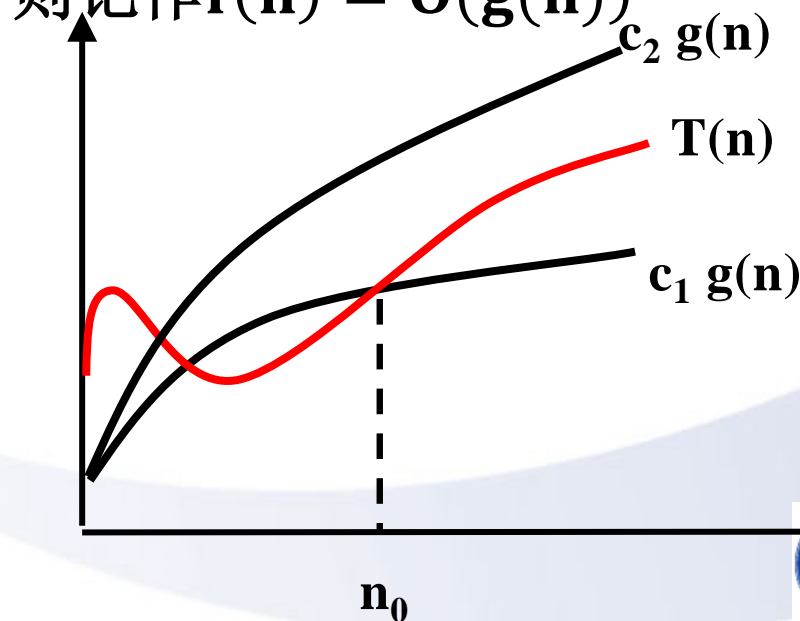
■ 4. 计算时间的渐近表示

□ “平均情况” 限界函数

□ 定义1.3 如果存在正常数 c_1 , c_2 和 n_0 , 对于所有的 $n \geq n_0$, 有

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$$

则记作 $f(n) = \Theta(g(n))$



$$T(n) = \Theta(g(n))$$



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ “平均情况” 限界函数

□ 含义:

- 算法在最好和最坏情况下的计算时间就一个常数因子范围内而言是相同的。可看作：既有 $f(n) = \Omega(g(n))$ ，又有 $f(n) = O(g(n))$
- 记号表明算法的运行时间有一个较准确的界



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□例：指数函数 $f(n) = 5 \times 2^n + n^2$

➤令 $n_0 = 0$, 当 $n \geq n_0$ 时, 有 $c_1 = 5$, $g(n) = 2^n$, 使
$$f(n) \geq 5 \times 2^n = c_1 g(n)$$

$$f(n) = \Omega(g(n)) = \Omega(2^n)$$

➤令 $n_0 = 4$, 当 $n \geq n_0$ 时, 有 $c_1 = 6$, $g(n) = 2^n$, 使
$$f(n) \leq 5 \times 2^n + 2^n = 6 \times 2^n = c_2 g(n)$$

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

➤又

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$f(n) = \Theta(2^n)$$



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□例：对数函数 $f(n) = \log n^k$, 其中, k 为某一正常数
 $\log n^k = k \log n$

➤令 $n_0 = 1$,

$$c_1 = k, \quad c_2 = k + 1, \quad g(n) = \log n$$

$$c_1 g(n) \leq k \log n \leq c_2 g(n)$$

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$f(n) = \Theta(\log n)$$



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□例: $f(n) = \log n!$

➤首先, $f(n) = \sum_{j=1}^n \log j \leq \sum_{j=1}^n \log n = n \log n$

➤令 $n_0 = 1, c_1 = 1, g(n) = n \log n$

$$f(n) \leq c_1 g(n) = O(g(n)) = O(n \log n)$$

➤设 n 为偶数:

$$\begin{aligned} f(n) &= \sum_{j=1}^n \log j \geq \sum_{j=1}^{n/2} \log \frac{n}{2} = \frac{n}{2} \log \frac{n}{2} \\ &= \frac{n}{2} (\log n - 1) = \frac{n}{4} (\log n + \log n - 2) \end{aligned}$$

➤令

$$n_0 = 4, c_1 = 1/4, g(n) = n \log n$$

$$f(n) \geq c_2 g(n) = \Omega(g(n)) = \Omega(n \log n)$$



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ 定理1.1

- ① 如果 $f(n)=O(g(n))$ ，且 $g(n)=O(h(n))$ ，则 $f(n)=O(h(n))$
- ② 如果 $f(n)=\Omega(g(n))$ ，且 $g(n)=\Omega(h(n))$ ，则 $f(n)=\Omega(h(n))$

□ 证明：

- 由 $f(n)=O(g(n))$ ，存在某个正常数 c 和 n_0 ，对所有的自然数 $n > n_0$ ，有 $f(n) \leq c(g(n))$ ；
- 由 $g(n)=O(h(n))$ ，存在某个正常数 c' 和 n'_0 ，对所有的自然数 $n > n'_0$ 有 $g(n) \leq c'(h(n))$
- 因此，存在正常数 $c_1 = c+c'$ ，对所有的自然数 $n > \max(n_0, n'_0)$ ，有 $f(n) \leq c_1(h(n))$ 。
- 所以， $f(n)=O(h(n))$ 。

□ (2) 的证明类似 (1) 的证明。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ **定理1.2** 对任意给定的函数 $f_1(n)$ 和 $f_2(n)$ ，存在函数 $g_1(n)$ 和 $g_2(n)$ ，满足 $f_1(n) = O(g_1(n))$ ， $f_2(n) = O(g_2(n))$ ，则 $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$

□ 证明：

- 由 $f_1(n) = O(g_1(n))$ ，存在某个正常数 c_1 和 n_1 ，对所有的自然数 $n \geq n_1$ ，有 $f_1(n) \leq c_1 g_1(n)$ 。
- 由 $f_2(n) = O(g_2(n))$ ，存在某个正常数 c_2 和 n_2 ，对所有的自然数 $n \geq n_2$ ，有 $f_2(n) \leq c_2 g_2(n)$
- 因此，对所有的自然数 $n \geq \max(n_1, n_2)$ ，有 $f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$
- 令 $c = 2\max(c_1, c_2)$ ，则有： $f_1(n) + f_2(n) \leq c (\max(g_1(n), g_2(n)))$
- 所以有： $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ **定理1.3** 对任意给定的函数 $f_1(n)$ 和 $f_2(n)$, 存在函数 $g_1(n)$ 和 $g_2(n)$, 满足 $f_1(n) = \Omega(g_1(n))$, $f_2(n) = \Omega(g_2(n))$, 则 $f_1(n) + f_2(n) = \Omega(\min(g_1(n), g_2(n)))$

□ 证明: 类似定理1.2的证明。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ **定理1.4** 如果 $f(n)$ 和 $g(n)$ 是非负函数，且 $f(n)=O(g(n))$ ，则 $f(n)+g(n) = \Theta(g(n))$ 。

□ 证明：

- 显然，对所有的自然数 $n \geq 0$ ，都有： $f(n)+g(n) \geq g(n)$ ，
- 因此，存在 $c=1$ ，使得 $f(n)+g(n) \geq c g(n)$ ，
- 所以，有 $f(n)+g(n) = \Omega(g(n))$ 。
- 此外，由 $f(n)=O(g(n))$ ，并且 $g(n)=O(g(n))$ ，
- 根据推论1.2，有 $f(n)+g(n) = O(g(n))$ 。
- 所以， $f(n)+g(n) = \Theta(g(n))$ 。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示

□ **推论1.1** 如果 $f(n) = \Theta(g(n))$, 且 $g(n) = \Theta(h(n))$, 则 $f(n) = \Theta(h(n))$ 。

□ **推论1.2** 如果 $f_1(n) = O(g(n))$, $f_2(n) = O(g(n))$, 则 $f_1(n) + f_2(n) = O(g(n))$ 。

□ **推论1.3** 如果存在某个正常数 k , 有函数 $f_1(n)$, $f_2(n), \dots, f_k(n)$, 对所有的 i , $1 \leq i \leq k$, 都有 $f_i(n) = O(g(n))$, 则 $f_1(n) + f_2(n) + \dots + f_k(n) = O(g(n))$ 。



1.2 复杂性分析初步

■ 4. 计算时间的渐近表示



渐近记号	名称
Θ	渐近紧确界
O	渐近上界
Ω	渐近下界



输入情况	情况说明
最好情况	不常出现，不具普遍性
最坏情况	确定上界，更具一般性



1.2 复杂性分析初步

■ 5. 程序运行时间的计算

□ 例1: 简单的赋值语句

`a = b;`

该语句执行时间为一常量, 为 $\Theta(1)$

□ 例2:

`sum = 0;`

`for(i=1; i<=n; i++)`

`sum+=n;`

□ 该语句执行时间为 $\Theta(n)$



1.2 复杂性分析初步

■ 5. 程序运行时间的计算

□ 例3 比较两个程序

```
sum1=0;  
  for(i=1;i<=n;i++)  
    for(j=1;j<=n;j++)  
      sum1++;
```

两个程序的执行时间都是 $\Theta(n^2)$, 不过第二个程序的运行时间约为第一个的一半

```
sum2=0;  
  for(i=1;i<=n;i++)  
    for(j=1;j<=i;j++)  
      sum2++;
```



例：插入分类

procedure INSERTIONSORT(A,n)

//将A(1: n)中的元素按非降次序分类, $n \geq 1$ //

A(0) \leftarrow $-\infty$ //设置初始边界值

执行次数

1

单位执行时间

t1

for j \leftarrow 2 **to** n **do** //A(1:j-1)已分类//

n-1

t2

item \leftarrow A(j);

n-1

t3

i \leftarrow j-1

n-1

t4

while item < A(i) **do** //0 \leq i < j//

最多j次,最少1次

t5

A(i+1) \leftarrow A(i);

最多j-1次,最少0次

t6

i \leftarrow i-1;

最多j-1次,最少0次

t4

repeat

A(i+1) \leftarrow item;

n-1次

t7

repeat

end INSERTIONSORT

算法的执行时间是T(n)



中国科学院大学

University of Chinese Academy of Sciences 50

最坏情况

$$T(n) = t_1 + (n-1)t_2 + (n-1)t_3 + (n-1)t_4 + \left(\sum_{j=2}^n j\right)t_5 + \left(\sum_{j=2}^n (j-1)\right)t_6 + \left(\sum_{j=2}^n (j-1)\right)t_4 + (n-1)t_7$$

令 $t_0 = \max(t_1, t_2, t_3, t_4, t_5, t_6, t_7)$

则

$$\begin{aligned} T(n) &\leq (1 + (n-1) + (n-1) + (n-1) + \left(\sum_{j=2}^n j\right) + \left(\sum_{j=2}^n (j-1)\right) + \left(\sum_{j=2}^n (j-1)\right) + (n-1))t_0 \\ &= (an^2 + bn + c)t_0 \\ &= O(n^2) \end{aligned}$$

最好情况

$$\begin{aligned} T(n) &= t_1 + (n-1)t_2 + (n-1)t_3 + (n-1)t_4 + (n-1)t_5 + (n-1)t_7 \\ &= an + b \\ &= O(n) \end{aligned}$$



作业-课后练习1

■ 问题描述

□ 硬件厂商XYZ公司宣称他们最新研制的微处理器运行速度为其竞争对手ABC公司的同类产品的100倍。对于计算机复杂性分别为 n , n^2 , n^3 和 $n!$ 的各算法,若用ABC公司的计算机在1h内分别能解输入规模为 n 问题,那么用XYZ公司的计算机在1h内分别能解输入规模为多大的问题?

■ 要求

□ 上载到 **国科大在线** 上



End

