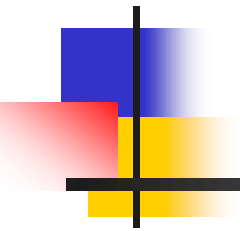


算法设计与分析

主讲教师：曲雯毓
天津大学智算学部
wenyu.qu@tju.edu.cn
Tel/WeChat：18502259631

课程目的

- 课程目的：
  - 计算机算法设计与分析导引
  - 介绍算法分析概念
  - 介绍算法设计的主要方法和基本思想；
  - 能够用算法思想编写小程序，并能分析算法复杂度
- 不是程序设计课，也不是数学课

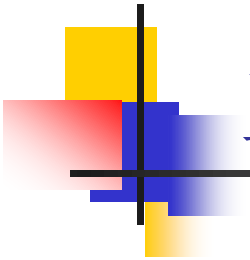
课程安排

- 课程安排（4-17周）授课40学时+实验16学时
- 学生成绩评定方法：平时成绩+实验+期末考试
- 上课时间及地点:

  周一3、4节

  周三1 ,2节

  55-B316

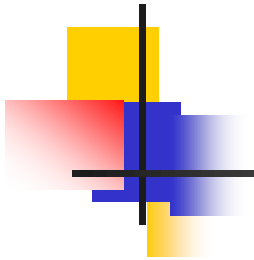


主要参考书目

- Sartaj Sahni，《Data Structures, Algorithms, and Applications in C++》，机械工业出版社，2000（影印版）

- Sartaj Sahni著，汪诗林等译，《数据结构、算法与应用--C++语言描述》，机械工业出版社，2003（翻译版）

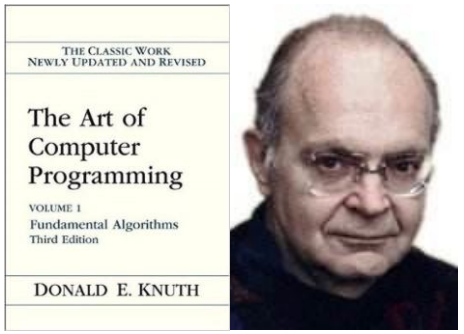
- T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms (the second edition)，The MIT Press，2001《算法导论（第二版）》（影印版，中文版），高等教育出版社，2003（精装版：amazon,688元,1-3个月发货）



第1章　算法概述

算法是计算机科学基础的重要主题

- 70年代前
  - 计算机科学基础的主题没有被清楚地认清。
- 70年代
  - Donald E. Knuth出版了《The Art of Computer Programming》
  - 以算法研究为主线
  - 确立了算法为计算机科学基础的重要主题
  - 1974年获得图灵奖。



- 70年代后
  - 算法作为计算机科学核心推动了计算机科学技术飞速发展.

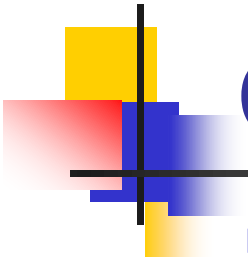
计算机科学的体系

- 解决一个计算问题的过程

什么是算法

- An algorithm is a finite, definite, effective procedure with some input and some output.”
  —— Donald Knuth
- “A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation.” —— webster.com

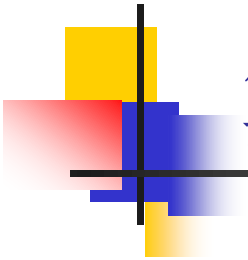
- 算法（algorithm）是对特定问题求解步骤的一种描述，是指令的有限序列。



Greatest Common Divisor

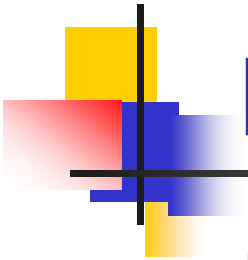
- Problem: Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n
- Examples:

  gcd(60,24)=12, gcd(60,0)=60, gcd(0,0)=?
- Middle-school procedure
  - Find the prime factorization of m
  - Find the prime factorization of n
  - Find all the common prime factors
  - Compute the product of all the common prive factors and return it as gcd(m,n)
- Is this an algorithm?



算法的5个特征

- 输入(input): 算法有零个或多个输入量；
- 输出(output): 算法至少产生一个输出量；
- 确定性(definiteness): 算法的每一条指令都有确切的定义，没有二义性；
- 能行性(effectiveness):算法的每一条指令必须足够基本，它们可以通过已经实现的基本运算执行有限次来实现；
- 有穷性(finiteness): 算法必须总能在执行有限步之后终止



NP问题

- 当数据的规模 n 比较小时，由于现在的计算机的速度都比较快，显现不出差距。
- 随着 n 的增加，不同算法的效率差别会很大。
- 对于一台每秒钟可以执行100万次计算的机器，当n=50时，复杂度为O(logn)的算法用时不足十万分之一秒，复杂度为$O(n^2)$的算法大约需要千分之一秒，对于复杂度为$O(2^n)$的算法需要36年，而执行一次复杂度为O(n!)的算法则要花费10^{57}年！
- 大整数因式分解是一个NP问题。

Euclid's Algorithm

- Euclid's algorithm is based on repeated application of equality

  $$gcd(m, n)=gcd(n, m \bmod n)$$

  until the second number becomes 0, which makes the problem trivial.

- Example:

  $$gcd(60,24)=gcd(24,12)=gcd(12,0)=12$$

Two descriptions of Euclid's algorithm

- Description One:

  Step 1 if n=0, return m and stop; otherwise go to Step 2

  Step 2 Divide m by n and assign the value of the remainder to r

  Step 3 Assign the value of n to m and the value of r to n. Go to Step 1

- Description Two:

```
while n ≠ 0 do
        r ← m mod n
        m ← n
        n ← r
return m
```

程序(Program)

- 程序是算法用某种程序设计语言的具体实现。

算法=程序?

程序(Program)

- 程序是算法用某种程序设计语言的具体实现。
- 程序可以不满足算法的性质(5)。
- 操作系统，是程序还是算法?
- 操作系统，是一个在无限循环中执行的程序，因而不是一个算法。
- 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

为什么学习算法

- 算法是计算机科学的**基础**，更是程序的**基石**，只有具有良好的算法基础才能成为训练有素的软件人才。
- 算法的目的是**求解问题**。

算法分析的内容

算法的性能-

算法所需的计算时间(快慢)和占用的内存空间

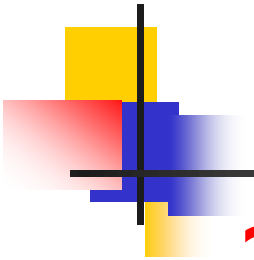
- 为什么要研究算法的性能？
  - 比较算法的“好坏”；
  - 判断所设计的算法是否“能用”：占用空间太大和计算时间过长的算法是实际上不能用的！(NP-难度问题)
- 性能评价的方法
  - 解析的方法-建立算法的运行时间和算法输入量之间的函数关系
  - 测量的方法
- 本课程以解析方法为主

问题和问题求解

- 常见的应用问题类型有：

1、搜索问题

- **所谓搜索，就是在给定的数据集合中寻找满足条件的数据对象。**
- **例如，在图书管理中，若要了解全部读者的借阅信息，就需要对每一位读者的借阅记录依次输出，这是数据对象的遍历问题。如果我们要查找借书超期的读者，则是按照规定的最大借书期限这一特征值去查找满足该条件的记录。**



2、排序问题

- 所谓排序，就是把一组无序的数据按照一定的规则排列起来。排序结果有升序和降序两种。
- 例如，为便于查询学生的考试成绩，对于成绩单，通常按照学号排序；而在招生录取中，也可以按照成绩的高低排序。
- 对于同一个问题，排序的算法是多种的，如插入排序、选择排序、交换排序等。

3、图论问题

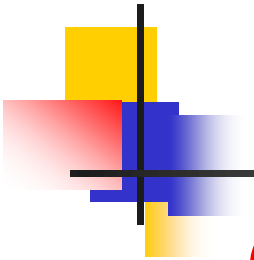
- 图论问题主要是研究**数据结构是图形结构或树形结构的算法问题**。简单的定义，可以认为图是由一些顶点和顶点之间的边所构成的集合。
- 图结构可以用来对各种各样的实际应用问题建模，包括**交通和通讯网络、工程项目时间表和各种竞赛**。图应用算法主要包括图的遍历算法、最短路径算法、最小生成树算法、拓扑排序算法和网络流算法等。

4、组合数学问题

- 有一些问题要求寻找一个组合对象，比如一个排列、一个组合或者一个子集，这些对象能够满足特定的条件并具有我们想要的特性(如价值最大化或者成本最小化)。从更抽象的角度来看，这类问题是组合数学问题，例如旅行售货商问题和图着色问题。
- 组合数学问题所涉及的应用领域更加广阔，例如拓扑学、图论、博弈论、线性规划等。组合数学问题的特点是随着问题规模的增加，已达到计算机的处理能力的极限。所以，称组合数学问题是计算机中的难解问题。

5、几何问题

几何算法处理类似于点、线、多面体这样的几何对象。经典的计算几何问题，如最近点对问题和凸包问题。顾名思义，最近点对问题求的是给定平面上的n个点中，寻找距离最近的两个点。凸包问题要求找一个能把给定集合中所有点包含都在里面最小凸多边形。当前，几何问题算法在**计算机图形学、机器人技术和断层X摄像技术**等方面都有广泛的应用。



6、数值计算问题

数值计算问题是另一大类问题，它要解决的问题包括：求解方程或者方程组和求数值积分等。这些问题常常只能给出一个近似的结果，而不是精确的解析答案。

问题求解

- **问题求解（problem solving）是寻找一种方法来实现目标。**
- **计算机求解问题的关键之一是寻找一种问题求解策略（problem solving strategy），得到求解问题的算法，从而得到问题的解。**

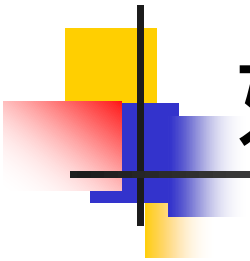
算法设计的一般过程

一般情况,问题求解过程包括:

- **理解问题**
- **预测所有可能的输入**
- **在精确解和近似解间做选择**
- **确定适当的数据结构**
- **算法设计技术:穷举法、分治法、动态规划法等**
- **描述算法**
- **跟踪算法**
- **分析算法效率**
- **根据算法编写代码**

算法分类

- 一个精确算法（exact algorithm）总能保证求得问题的解。
- 一个启发式算法（heuristic algorithm）通过使用某种规则、简化或智能猜测来减少问题求解时间。
- 对于最优化问题，一个算法如果致力于寻找近似解而不是最优解，被称为近似算法（approximation algorithm）
- 如果在算法中需做出某些随机选择，则称为随机算法（randomized algorithm）



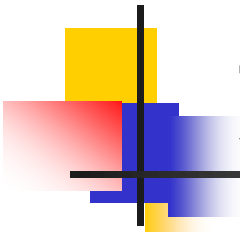
如何设计算法

- 使用面向计算机的问题求解策略 —— 算法设计策略（algorithm design strategy）。

如何表示算法

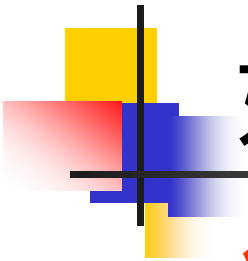
- 算法描述方法
  - 自然语言
  - 流程图
  - 伪代码
  - 程序设计语言

本课程使用 C++语言描述算法



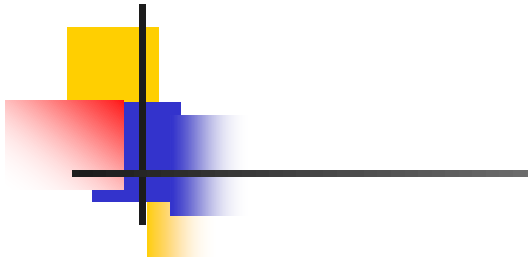
如何验证算法

- 确认一个算法是否正确的活动称为算法验证（algorithm validation）
- 使用数学方法证明算法的正确性，称为算法证明（algorithm proof）。
- 程序测试（program testing）是指对程序模块或程序总体，输入事先准备好的样本数据（称为测试用例，test case），检查该程序的输出，来发现程序存在的错误及判定程序是否满足其设计要求的一项积极活动。



如何分析算法

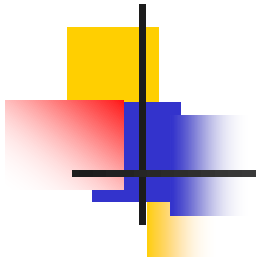
- **算法分析**（algorithm analysis）活动是指对算法的执行时间和所需空间的估算
- 当然在算法写成程序后，便可使用样本数据，实际测量一个程序所消耗的时间和空间，这称为程序的**性能测量**（performance measurement)



第2章　程序性能

程序的性能

- 程序的性能指程序运行时所需的内存空间量和计算时间:系统开销和求解问题本身的开销.
- 为什么要研究性能？忽略系统开销，区分出算法的性能，设计满足要求的算法.
- 怎样定义“要求”？需要某种度量指标.
- 很多问题找不到满足要求的算法



- 性能评价的方法
  - 解析的方法
  - 测量的方法
- 本课程以解析方法为主－算法分析。

算法复杂性

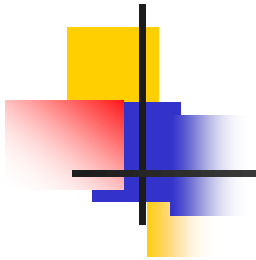
- 算法复杂性是算法运行所需要的计算机资源的量，需要时间资源的量称为时间复杂性，需要的空间资源的量称为空间复杂性。

- 算法复杂性只依赖于算法要解的问题的规模、算法的输入和算法本身的函数

问题和问题的实例

- 问题(排序)：任意给定具有“序”的关系的集合U上的n个元素,试(给出一算法)将这些元素按“从小到大”进行排列.
- 问题实例(算法输入):任给n＝100个整数,试将其排序.

- 实例长度(size):表示实例的数据结构的长度.例如,向量的长度nu, u为单个元素的字节数.为了简化分析,我们往往忽略u,而用实例特征表示实例长度.
- 实例特征:描述问题实例长度(size)的参数.例如矩阵的阶数n是反映实例长度(n^2)的参数.
- 选取n为实例特征.
- 实例特征的选取有随意性.



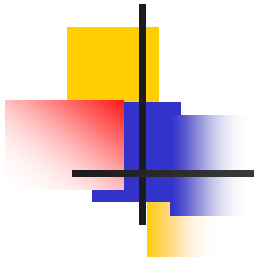
- 算法必须对任意问题实例都能给出结果-算法的一般性.
- 算法使用一些能机械执行的基本操作-算法的机械性.
- 算法由有穷长度的一组规则组成(例如有限自动机,计算机语言写的程序).
- 算法分析建立算法执行时间或空间和实例特征的函数关系

空间复杂度(Space Complexity) S(n)

- 程序的空间复杂度指程序运行时所需的内存空间大小和实例特征的函数关系.
- 程序运行时所需空间包括:
  - 指令空间-与实例特征无关的常数
  - 数据空间：常量和简单变量-实例无关

    复合变量(数组、链表、树和图等)

    环境栈空间(函数调用)-是否递归?
  - 复合变量所需空间常常和问题实例特征有关



- 程序P的空间需求量包括两部分：

$$S(n)=c+S_p(n)$$

c为常量(实例无关部分)，S_p为可变部分，n是实力特征。

- 在分析空间复杂度时，我们忽略与实例特征无关的空间需求量 c，仅考虑函数$S_p(n)$。

例 顺序查找

T a[]和T& x需2bytes
指针(假定T为整形)

形参n需2bytes

```
template <class T>
int SequentialSearch(T a[], const T& x, int n)
{// Search the unordered list a[0:n-1] for x.
 // Return position if found; return -1 otherwise.
   int i;
   for (i = 0; i < n && a[i] != x; i++);
   if (i == n) return -1;
   return i;
}
```

i需2bytes

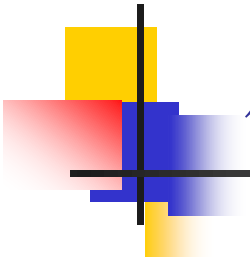
以上均为实例特征独立的空间需求量，S(n)=0

注：上述分析是从程序调用的角度看，存放无序表的数组占用的空间记在上层程序的帐上。

例 计算n!

```
int Factorial(int n)
{// Compute n!.
    if (n <= 1) return 1;
    else return n * Factorial(n - 1);
}
```

实例特征：n， S(n)= 4*max{n,1}



小结

- 对非递归算法
  - 分析与实例特征有关的数据结构的大小
- 对递归算法
  - 还要分析递归调用的深度和实例特征的关系

时间复杂度(Time complexity)T(n)

- 时间复杂度指程序执行时所用的时间。
- 在使用解析方法时程序P的时间复杂度表示为输入量的函数T。
- 机器独立的分析方法-解析的方法.
- 在解析地分析时间复杂度时,使用以下两种时间单位并计算:
  - 操作计数(operation count):算法的基本操作
  - (程序)步计数(step count):分析全部程序
- 要点:基本操作或程序步的执行时间必须是常数。

例 查找最大元素

```
template <class T>
int Max(T a[], int n)
{// Locate the largest element in a[0:n-1].
    int pos = 0;
    for (int i = 1; i < n; i++)
      if (a[pos] < a[i])
        pos = i;
    return pos;
}
```

实例特征：n，
operation: comparison
t(n)=n-1 且不可少于n-1

如果包含其他operations，则时间复杂度<某常数*t(n)。

例 Ranking

```
template <class T>
void Rank(T a[], int n, int r[])
{// Rank the n elements a[0:n-1].
   for (int i = 0; i < n; i++)
     r[i] = 0; //initialize
   // compare all element pairs
   for (int i = 1; i < n; i++)
    for (int j = 0; j < i; j++)
      if (a[j] <= a[i]) r[i]++;
      else r[j]++;
}
```

Instance size : n ,
operation: comparison
t(n)=n*(n-1)/2

- 例如a=[4,3,9,3,7] 则r=[2,0,4,1,3]
- 又如a＝[9,3,9,3,7]则r=[3,0,4,1,2]
- a中两个3有不同的rank值。
- 每个元素有不同的rank值; rank取值于{0,1,2,...,n-1}.
- 如果r[i]<r[j],则a[i]≤a[j]
- 上述关系可用于排序:rank值等于该元素在排序序列中的位置,按rank值将这些元素放到数组的相应位置.

例 Rank Sort

```
template <class T>
void Rearrange(T a[], int n, int r[])
{// Rearrange the elements of a into sorted order
 // using an additional array u.
   T *u = new T [n+1];
   // move to correct place in u
   for (int i = 0; i < n; i++)
     u[r[i]] = a[i];
   // move back to a
   for (int i = 0; i < n; i++)
     a[i] = u[i];
   delete [] u;
}
```

实例特征 ：n ，
operation: element move(即赋值),
t(n)=2n

该题调用程序2.5，r[i]=rank of a[i]

例 Sequential Search

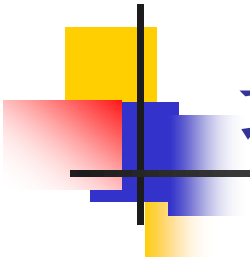
```
template <class T>
int SequentialSearch(T a[], const T& x, int n)
{// Search the unordered list a[0:n-1] for x.
 // Return position if found; return -1 otherwise.
   int i;
   for (i = 0; i < n && a[i] != x; i++);
   if (i == n) return -1;
   return i;
}
```

Best case: t(n)=1 ,
Worst case : t(n)=n

当仅考虑成功查找时，average case : t(n)=（n+1）/2；

最好，最坏和平均情形时间复杂度

- 当长度相同的不同输入有不同的计算时间时，时间复杂度分析分别考虑三种情形：即最好，最坏和平均.
- 当应用对计算时间有严格要求时，应做最坏情形分析-upper bound.
- 最好情形分析给出一个算法的计算时间的下界，用来否定一个算法.



最好，最坏和平均情形时间复杂度

Worst-case: (usually)

- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (bogus)

例 插入排序-1

best case : 1 ,
worst case : n ,
average case: (n/2)+n/(n+1)

```
template <class T>
void Insert(T a[], int & n, co    T& x)
{// Insert x into the sorted a   ay a[0:n-1].
 // Assume a is of size > n.
   int i;
   for (i = n-1; i >= 0 && x < a[i]; i--)
     a[i+1] = a[i];
   a[i+1] = x;
   n++; // one element added to a
}
```

分析：被插入的数据x有相同的机会被插入到任何一个位置上，共有n＋1个可能的插入位置，如果x最终被插入到a的i＋1处，则执行的比较次数为n－i，如果x被插入到a[0]，则比较次数为n。然后求平均的比较次数

例 Insertion Sort-2

```
template <class T>
void InsertionSort(T a[], int n)
{// Sort a[0:n-1].
  for (int i = 1; i < n; i++) {
    T t = a[i];
    Insert(a, i, t);
  }
```

best case : 1 ,
worst case : i ,
average case: (i/2)+i/(i+1)

- Insertion Sort 最好情形使用(n-1)次比较： 已排好序输入.
- 最坏情形使用为n(n-1)/2次比较：从大到小排列的输入(reverse ordered).
- 平均情形分析很难.

Step Count 方法

- 一条或多条执行时间是常数的语句称为“1步”，例如简单赋值语句；不涉及函数调用的表达式求值；简单的条件判断等。
- Step Count 方法以程序执行的总“步数”，即“步数”计数，作为算法的时间复杂度T(n)。
- 设c1≤每“步”的执行时间≤c2,则实际执行时间满足：$c_1t(n)$≤实际执行时间≤$c_2t(n)$
- 划分程序步的方法不唯一，t(n)也不唯一,但数量级唯一.

例 顺序查找——最好情况

Statement	s/e	Frequency	Total steps
int SequentialSearch(T a[], T& x, int n)	0	0	0
{	0	0	0
int i;	1	1	1
for (int i = 0; i < n && a[i] != x; i++)	1	1	1
if (i == n) return -1;	1	1	1
return i;	1	1	1
}	0	0	0
Total			4

Figure 2.9 Best-case step count for Program 2.1

例 顺序查找—最坏情况

Statement	s/e	Frequency	Total steps
int SequentialSearch(T a[], T& x, int n)	0	0	0
{	0	0	0
int i;	1	1	1
for (i = 0; i < n && a[i] != x; i++)	1	$n + 1$	$n + 1$
if (i == n) return -1;	1	1	1
return i;	1	0	0
}	0	0	0
Total			$n + 3$

Figure 2.10 Worst-case step count for Program 2.1

例 求和

Statement	s/e	Frequency	Total steps
T Sum(T a[], int n)	0	0	0
{	0	0	0
T tsum = 0;	1	1	1
for (int i = 0; i < n; i++)	1	$n+1$	$n+1$
tsum += a[i];	1	n	n
return tsum;	1	1	1
}	0	0	0
Total			$2n+3$

s/e代表该语句执行后步 数 (count) 的 变 化（增量）；

Frequency代表该语句的执行次数；

Total steps代表该语句在整个程序执行过程中引发的总步数。

Frequency可能不等于Total steps，

```
template <class T>
void Inef(T a[], T b[], int n)
{// Compute prefix sums.
   for (int j = 0; j < n; j++)
    b[j] = Sum(a, j + 1);
}
```

该赋值语句不能当做“1步”:sum要执行2(j+1)+3步，j是变量

Step table for Program Inef

Statement	s/e	Frequency	Total steps
void Inef(T a[], T b[], int n)	0	0	0
{	0	0	0
for (int j = 0; j < n; j++)	1	$n + 1$	$n + 1$
b[j] = Sum(a, j + 1);	$2j + 6$	n	$n(n + 5)$
}	0	0	0
Total			$n^2 + 6n + 1$

Figure 2.8 Step table for Program 2.23

Frequency不等于Total steps，应求和得到total.

递归程序

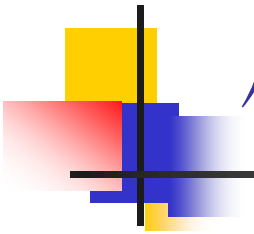
- 当分析递归程序时，我们使用解递归方程的方法。

```
template <class T>
T Rsum(T a[], int n)
{// Return sum of numbers a[0:n - 1].
    if (n > 0)
      return Rsum(a, n-1) + a[n-1];
    return 0;
}
```

t(0)=2

t(n)=2+t(n-1)

解此方程可得t(n)=2(n+1)。



例 Matrix Addition

Statement	s/eFrequency	Total steps
void Add(T **a · · ·)	0 0	0
{	0 0	0
for (int i=0; i<rows; i++)	1 $rows + 1$	$rows + 1$
for (int j = 0; j < cols; j++)	1 $rows \cdot (cols + 1)$	$rows \cdot cols + rows$
c[i][j] = a[i][j] + b[i][j];	1 $rows \cdot cols$	$rows \cdot cols$
}	0 0	0
Total		$2rows \cdot cols + 2rows + 1$

Figure 2.6 Step table for program 2.19

2.4渐近分析,符号(O,Ω,Θ)

- 计算机科学使用最多的符号-讨论算法时使用的共同语言.
- f(n)=O(g(n)) iff 存在常数c和n_0使得对所有$n>n_0$ 有$f(n)<cg(n)$ 成立.
- $Lim_{n->\infty} | f(n)/g(n) | =c,\ 0\leq c<\infty$,则

  f(n)=O(g(n))

渐近分析(续)

- $10n^2+3n+100=O(n^2)$,lim=10;
- $1000n^2=O(n^{2.1})$,lim=0
- 称g(n)为f(n)的渐近上界(Asymptotic Upper Bound).
- 约定O(1)代表常数.
- g(n)常取一些简单的初等函数,n^k,$\log_2 n$和$n^k\log_2 n$等:

2.4渐近分析(续)

Function	Name
1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n log n
n^2	quadratic
n^3	cubic
2^n	exponential
n !	factorial

Figure 2.14 Common asymptotic functions

Can a Algorithm work with large input ?

$\log n$	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	65,536
5	32	160	1024	32,768	4,294,967,296

Figure 2.23 Value of various functions

渐近分析-随n的增加T(n)的增长率

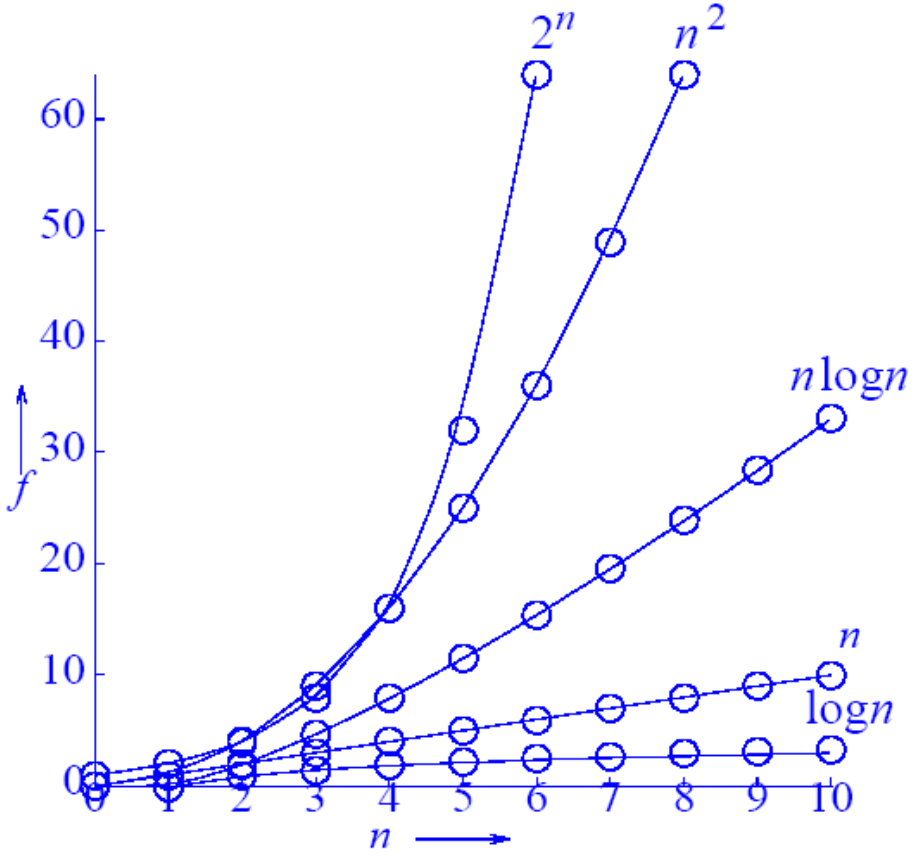


Figure 2.24 Plot of various functions

2.4渐近分析(续)

- 大Ω

  $f(n)=\Omega(g(n))$ iff 存在常数c和n_0使得对所有$n>n_0$,有$f(n)>cg(n)$成立.

- $\lim_{n\to\infty} |f(n)/g(n)| = c$, $0<c\leq\infty$,则

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

渐近分析(续)

- 称g(n)为f(n)的渐近下界
- 例如,

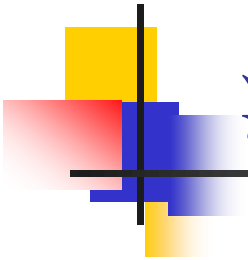
  $f(n)=0.001n^2-10n-1000=\Omega(n^2)$

  因为:$\lim f(n)/n^2=0.001$

2.4渐近分析(续)

- 符号Θ
- 如果$f(n)=O(g(n))$同时 $f(n)=\Omega(g(n))$则$f(n)=\Theta(g(n))$,并称$f(n)$与$g(n)$同阶.
- Lim $f(n)/g(n)=c$, $0<c<\infty$,则

  $f(n)=\Theta(g(n))$
- $g(n)$取上述初等函数



渐近分析(续)

- 当f(n)为算法的时间复杂度函数时，称g(n)为该算法的复杂度的阶。

2.4渐近分析时用到的一些等式

	$f(n)$	Asymptotic
E1	c	$\oplus(1)$
E2	$\sum_{i=0}^{k} c_i n^i$	$\oplus(n^k)$
E3	$\sum_{i=1}^{n} i$	$\oplus(n^2)$
E4	$\sum_{i=1}^{n} i^2$	$\oplus(n^3)$
E5	$\sum_{i=1}^{n} i^k, k>0$	$\oplus(n^{k+1})$
E6	$\sum_{i=0}^{n} r^i, r>1$	$\oplus(r^n)$
E7	$n!$	$\oplus(n\ (n/e)^n)$

续

E8 $\sum_{i=1}^{n} 1/i$ $\oplus(\log n)$

$\oplus$ can be any one of O, Ω, and Θ
Figure 2.15 Asymptotic identities

$$n! \sim (2\pi)^{\frac{1}{2}} n^{n+\frac{1}{2}} e^{-n}$$

推导规则

I1 $\{f(n) = \oplus(g(n))\} \rightarrow \sum_{n=a}^{b} f(n) = \oplus(\sum_{n=a}^{b} g(n))$

I2 $\{f_i(n) = \oplus(g_i(n)),\ 1 \le i \le k\} \rightarrow \sum_{1}^{k} f_i(n) = \oplus(\max_{1 \le i \le k} \{g_i(n)\})$

I3 $\{f_i(n) = \oplus(g_i(n)),\ 1 \le i \le k\} \rightarrow \prod_{1}^{k} f_i(n) = \oplus(\prod_{1}^{k} g_i(n))$

I4 $\{f_1(n) = O(g_1(n)), f_2(n) = \Theta(g_2(n))\} \rightarrow$
$f_1(n)+f_2(n) = O(g_1(n)+g_2(n))$

推导规则(续)

I5 $\{f_1(n) = \Theta(g_1(n)), f_2(n) = \Omega(g_2(n))\} \rightarrow$
$f_1(n)+f_2(n) = \Omega(g_1(n)+g_2(n))$

I6 $\{f_1(n) = O(g(n)), f_2(n) = \Theta(g(n))\} \rightarrow$
$f_1(n)+f_2(n) = \Theta(g(n))$

Figure 2.16 Inference rules for $\oplus \in \{O,\Omega,\Theta\}$

渐近分析例题

Statement	s/e	Frequency	Total steps
T Sum(T a[], int n)	0	0	$\Theta(0)$
{	0	0	$\Theta(0)$
T tsum = 0;	1	1	$\Theta(1)$
for (int i = 0; i < n; i++)	1	$n+1$	$\Theta(n)$
tsum += a[i];	1	n	$\Theta(n)$
return tsum;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{Sum}(n)=\Theta(\max\{g_i(n)\}) = \Theta(n)$$

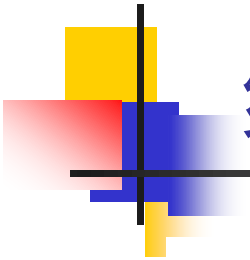
Figure 2.17 Asymptotic complexity of Sum (Program 1.8)

续

Statement	s/e	Frequency	Total steps
T Rsum(T a[], int n)	0	0	$\Theta(0)$
{	0	0	$\Theta(0)$
if (n)	1	$n+1$	$\Theta(n)$
return Rsum(a,n−1) + a[n−1];	1	n	$\Theta(n)$
return 0;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{Rsum}(n) = \Theta(n)$$

Figure 2.18 Asymptotic complexity of Rsum (Program 1.9)

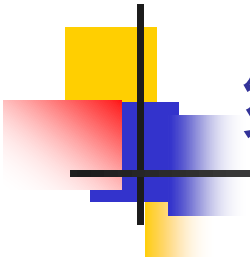


续

Statement	s/e	Frequency	Total steps
void Inef(T a[], T b[], int n)	0	0	$\Theta(0)$
{	0	0	$\Theta(0)$
for (int j = 0; j < n; j++)	1	$\Theta(n)$	$\Theta(n)$
b[j] = Sum(a, j + 1);	$2j + 6$	n	$\Theta(n^2)$
}	0	0	$\Theta(0)$

$$t_{Inef}(n) = \Theta(n^2)$$

Figure 2.21 Asymptotic complexity of Inef Program 2.23



续

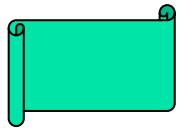
Statement	s/e	Frequency	Total steps
int SequentialSearch(T a[], T& x, int n)	0	0	$\Theta(0)$
{	0	0	$\Theta(0)$
int i;	1	1	$\Theta(1)$
for (i = 0; i < n && a[i] != x; i++)	1	$\Omega(1), O(n)$	$\Omega(1), O(n)$
if (i == n) return -1;	1	1	$\Theta(1)$
return i;	1	$\Omega(0), O(1)$	
}	0	0	$\Theta(0)$

$$t_{SequentialSearch}(n) = \Omega(1)$$

$$t_{SequentialSearch}(n) = O(n)$$

Figure 2.22 Asymptotic complexity of `SequentialSearch` (Program 2.1)

续



- 顺序查找最好情形时间复杂度为$\Theta(1)$
- 顺序查找最坏情形时间复杂度为$\Theta(n)$
- 顺序查找平均情形时间复杂度为$\Theta(n)$
- 插入排序,选择排序和rank排序最坏情形时间复杂度为$O(n^2)$.(upper bound)
- 更准确地讲它们的最坏情形复杂度是$\Theta(n^2)$.(tight bound)
- 平均情形时间复杂度为$\Theta(n^2)$.

多项式时间算法

- 如果一算法的最坏情形时间复杂度$t(n)=O(n^k)$,则称该算法为多项式复杂度的算法或有多项式界的算法.
- 如果一算法的最坏情形时间复杂度t(n)不能用多项式限界,则称该算法为指数复杂度的算法。这类算法可认为计算上不可行的算法。

NP-hard 问题

- 如果一个问题有多项式界的算法称该问题属于多项式类**P**
- 有很多实际上有意义的问题找不到有多项式界的算法称这些问题是**NP-hard**问题,即问题本身难.
- 上述难问题只能通过近似算法或启发式算法求解.

数值比较:1G指令/秒的计算能力

	$f(n)$						
n	n	$n\log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01μs	.03μs	.1μs	1μs	10μs	10s	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84h	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1s
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121d	18m
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1y	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171y	$4*10^{13}$ y
10^3	1μs	9.96μs	1ms	1s	16.67m	$3.17*10^{13}$ y	$32*10^{283}$ y
10^4	10μs	130μs	100ms	16.67m	115.7d	$3.17*10^{23}$ y	
10^5	100μs	1.66ms	10s	11.57d	3171y	$3.17*10^{33}$ y	
10^6	1ms	19.92ms	16.67m	31.71y	$3.17*10^7$ y	$3.17*10^{43}$ y	

μs = microsecond = 10^{-6} seconds; ms = milliseconds = 10^{-3} seconds
s = seconds; m = minutes; h = hours; d = days; y = years

数值比较:1G指令/秒的计算能力

	$f(n)$						
n	n	$n\log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01μs	.03μs	.1μs	1μs	10μs	10s	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84h	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1s
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121d	18m
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1y	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171y	$4*10^{13}$ y
10^3	1μs	9.96μs	1ms	1s	16.67m	$3.17*10^{13}$ y	$32*10^{283}$ y
10^4	10μs	130μs	100ms	16.67m	115.7d	$3.17*10^{23}$ y	
10^5	100μs	1.66ms	10s	11.57d	3171y	$3.17*10^{33}$ y	
10^6	1ms	19.92ms	16.67m	31.71y	$3.17*10^7$ y	$3.17*10^{43}$ y	

μs = microsecond = 10^{-6} seconds; ms = milliseconds = 10^{-3} seconds
s = seconds; m = minutes; h = hours; d = days; y = years

本章作业：

- 第二章程序的性能：
- Q15 、Q16；Q18； Q20；Q24；Q37(2)(4)(6)(7)(8)(9)(13)