第1章 绪论

1.1 什么是数据结构

- 数据结构是以某种方式联系在一起的数据元素的集合。程序中的数据结构反映了程序员在程序中表示信息的方法，算法反映了如何处理信息的方法。
- 数据结构研究的是数据元素之间抽象化的相互关系及这种关系在计算机中的存储表示，对每种结构定义各自的运算，设计出相应的算法，并用某种语言实现该算法。
- 数据结构的地位：数学、硬件、软件之间。
- 核心专业基础课

1.2 基本概念和术语

1.基本术语

（1）**数据：**描述客观事物的数字、字符以及所有能输入到计算机中并被计算机程序处理的符号的集合。（数字、字符、声音、图形、图像等等）

（2）**数据元素：**数据的基本单位，在计算机程序中常常作为一个整体进行考虑和处理，如记录/结构。

（3）**数据项：**数据的不可分割的最小单位，如结构中的域。

（4）**数据对象：**性质相同的数据元素的集合，是数据的一个子集。

1.2 基本概念和术语

1.基本术语

（5）**逻辑结构**：数据与数据之间的联系

（6）**存储结构**：数据结构在计算机中的存储表示（或称映像），又称物理结构。它包括数据元素的表示和关系的表示。

1.2 基本概念和术语

2.数据结构

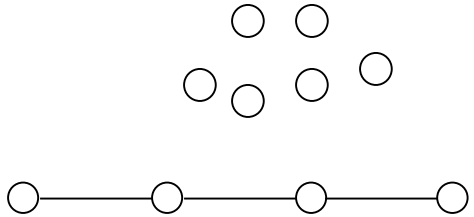
（1）定义：是相互之间存在一种或多种特定关系的数据元素的集合。

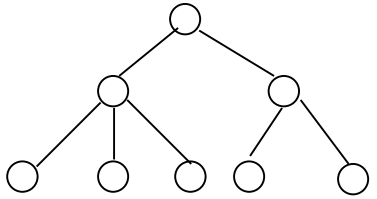
数据之间不是相互独立的，他们之间有某种特定的关系，这种数据元素之间的关系，称为“结构”。

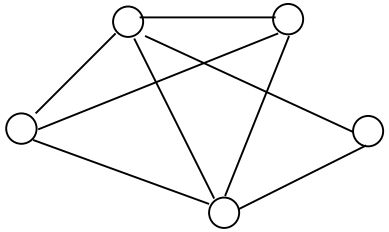
结构=关系+实体

- 另一种定义： 按照逻辑关系组织起来的一批数据,按一定的存储方法把它存储在计算机中,并在这些数据上定义了一个运算的集合。
- 形式定义：二元组 (D,S) 其中D是数据元素的有限集，S是D上关系的有限集

1.2 基本概念和术语







2.数据结构

（2）四种基本结构（逻辑结构） p5

集合：元素仅属于同一个集体，没有其他关系。

线性结构：存在一对一 关系，序列相邻，次序关系。

树型结构：存在一对多关系，层次关系。

图状结构：（网状结构）存在多对多关系，任意性

- 存储器模型：一个存储器M是一系列固定大小的存储单元，每个单元U有一个唯一的地址A(U)，该地址被连续地编码。每个单元U有一个唯一的后继单元U’=succ(U)
- 物理结构就是逻辑结构到存储器的一个映射。
- 四种存储结构：顺序存储、链接存储、索引存储、散列存储

（3）实例：
表：计算机系人事表

工号	姓名	性别	职务	教研室	工作时间	发表论文
01	张三	男	系主任	软件	1981.1	A,B
02	李四	女	教研室主任	软件	1985.1	B,C,E,F
03	王五	男	教师	软件	1990.8	C,D
04	赵六	男	教师	应用	1987.8	A,G
05	XXX	男	教师	应用	1975.9	E,I
06	XXX2	男	教师	应用	1992.2	F,J
07	XXX3	女	教师	软件	1983.8	D,L
08	XXX4	女	教研室主任	应用	1986.7	G,H
09	XXX5	男	教师	应用	1995.8	H,I,J,K
10	XXX	女	教师	软件	1989.2	L,K

- 线性结构示例 R={<05,01>, <01,07>, <07,02>, <02,08>, <08,04>, <04,10>, <10,03>, <03,06>,<06,09>}

- 树型结构示例
  R={<01,02>,<01,08>,<02,03>,<02,10>,<02,07>,<08,04>,<08,05>,<08,06>,<08,09>}

- 图状结构示例
  R={(01,02),(01,04),(02,05),(02,06),(02,03),(03,07),(04,08),(05,09),(06,09),(07,10),(08,09),(09,10)}

1.2 基本概念和术语

3. 数据结构的划分

（1）按数据结构的性质划分

- 数据的逻辑结构——数据元素之间的逻辑关系（设计算法--数学模型）
- 数据的物理结构——数据结构在计算机中的映像。（存储结构，算法的实现）

（2）按数据结构在计算机内的存储方式来划分

- 顺序存储结构——借助元素在存储器的相对位置来表示数据元素之间的逻辑关系。
- 链式存储结构——借助指示元素存储地址的指针表示数据元素之间的逻辑关系。
- 索引存储方法——在存储结点的同时，还建立附加的索引表，索引表
- 中的每一项称为索引项，形式为：关键字，地址。
- 散列存储方法——根据结点的关键字直接计算出该结点的存储地址。

1.2 基本概念和术语

3. 数据结构的划分

（3）按数据结构的操作来划分

- **静态结构**——经过操作后，数据的结构特征保持不变（如数组）。
- **半静态结构**——经过操作后，数据的结构特性只允许很小变迁（如栈、队列）。
- **动态结构**——经过操作后，数据的结构特性变化比较灵活，可随机地重新组织结构（如指针）。

1.3 抽象数据类型——ADT

- 定义：是指基于一个逻辑类型的数据模型以及定义在该模型上的一组操作。每一个操作由它的输入和输出定义。
- 形式：三元组(D,S,P) S是D上的关系集，P是对D的基本操作集
- 说明：一个ADT的定义并不涉及它的实现细节。这些实现细节对于ADT的用户是隐藏的。隐藏实现细节的过程称为封装。数据结构是ADT的物理实现。ADT的每一个操作均由一个或多个子程序来实现。

1.3 抽象数据类型——ADT

- 抽象数据类型和数据类型实质上是一个概念。数据类型是一个值的集合和定义在这个值集上的一组操作的总称。分为原子类型和结构类型
- 数据抽象与过程抽象，实现信息隐蔽和局部化以一个严格定义的过程接口的方式，在数据结构上提供一个抽象。数据的实现和处理细节被隐蔽了

1.4 算法和算法分析

- 1. 指一系列确定的而且是在有限步骤内能完成的操作。

1.4 算法和算法分析

2. 算法的特性

(1) 有穷性 （能执行结束）

(2) 确定性 （对于相同的输入执行相同的路径）

(3) 0至多个输入

(4) 1至多个输出

(5) 有效性(可行性)

——用于描述算法的操作都是足够基本的

●问题：程序是不是算法?

如操作系统，只要系统不遭破坏，它就永远不会停止，即使没有作业要处理，仍处于一个等待循环中，等待新作业的进入。因此操作系统程序不是一个算法。

1.4 算法和算法分析

3. 算法与数据结构的关系

- 计算机科学家沃斯（N.Wirth）提出的“**算法+数据结构=程序**”揭示了程序设计的本质：对实际问题选择一种好的数据结构，加上设计一个好的算法，而好的算法很大程度上取决于描述实际问题的数据结构。算法与数据结构是互相依赖、互相联系的。
- 一个算法总是建立在一定数据结构上的；反之，算法不确定，就无法决定如何构造数据。
- 一个算法的设计取决于选定的数据（逻辑）结构，而算法的实现依赖于采用的存储结构。

●例1：编写程序查询某城市某人的电话号码

建立一张登记表，存放2个数据项：姓名+Tel。好的算法取决于这张表的结构及存储方式。

（1）将表中结点按照姓名顺序地存储在计算机中，依次查找，可能遍历整个表都找不到。

（2）再建立一张姓氏索引表，姓+表中的起始地址。不需查找其他姓氏。查找效率提高。

- 例2: 设计一个考试日程安排表，使在尽可能短的时间内安排完考试，要求同一个学生选修的几门课程不能安排在同一个时间内。

姓名	选修1	选修2	选修3
张三	A	B	E
李四	C	D	
王五	C	E	F
赵六	D	F	A
	B	F	

- 解决该问题，首先选择一个合适的数据结构。用无向图表示，图中的顶点表示课程，不能同时考试的课程之间连上一条边。则该问题就抽象成对该无向图进行“着色”操作，即用尽可能少的颜色去给图中每个顶点着色，使得任意两个相邻的顶点着不同的颜色。同一种颜色表示一个考试时间。

答案：1： A,C 2:B,D 3:E 4: F

- 解决问题的关键步骤是先选取合适的数据结构表示问题，才能写出有效的算法。

1.4 算法和算法分析

4. 算法设计的要求 p13～p14

(1) 正确性（四层含义）

(a) 不含语法错误

(b) 对于几组输入得到预期的输出

(c) 对于精心选择的典型、苛刻而带有刁难性的输入产生期望的输出

(d) 对于所有的合法输入产生期望的输出。

(2) 可读性

首先是给人读，然后才是机器执行

(3) 健壮性 容错性

(4) 效率与低存储量需求

1.4 算法和算法分析

- 定义：一个算法如果能在所要求的**资源限制**内将问题解决好，则称这个算法是**有效率的**。

例如，一个资源限制是：

可用来存储数据的全部空间——可能是分离的内存空间限制和磁盘空间限制

和允许执行每一个子任务所需要的时间。

1.4 算法和算法分析

5. 算法的分析 ——算法性能的评价

●评价标准：

1）算法所需的计算时间

2）算法所需的存储空间

3）算法的简单性

● 度量算法执行时间的两种方法

1）事后统计法 （计时）

此方法有两个缺陷 （算法写成程序，时间统计依赖硬件）

2）事前分析估算法

此方法取决于多个因素：

算法策略，问题规模，程序语言，编译质量，机器速度等

1.4 算法和算法分析

6. 时间复杂度

（1）定义：

一般情况下，算法中基本操作重复执行的时间是问题规模n的某个函数f(n)，算法的时间量度记作

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

它表示随问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同，称作算法的渐进时间复杂度，简称**时间复杂度**。

●语句的**频度**：该语句重复执行的次数。

1.4 算法和算法分析

- 6. 时间复杂度

（2）例子

（a） {++x; s=0 } ++x 的频度为1

（b） for (i=1; i<=n; ++i) {++x; s+=x;} ++x的频度为n

（c） for (j=1;j<=n;++j)

for (k=1;k<=n;++k) {++x; s+=x;} ++x的频度为n^2

●理解:

a) n*n+4*n=O(n*n);

b) n的平方或n的2000次方总没有2的n次方增长快

大O运算规则

规则1：kf(n)=O(f(n)) 大O忽略常数因子

规则2：if f(n) = O(g(n))and g(n)=O(h(n)) then f(n)=O(h(n)) 传递性

规则3：f(n) + g(n)= O(max{f(n),g(n)})

规则4：if f1(n)=O(g1(n)) and f2(n)=O(g2(n))then

f1(n)*f2(n)=O(g1(n)*g2(n))

1.4 算法和算法分析

- 最佳、最差、平均时间复杂度
  - 无法精确计算基本操作的执行次数（频度）时，只须求出其关于n的增长率
  - 与输入数据集有关，如冒泡程序，可考虑平均时间复杂度和最坏情况下的时间复杂度

1.4 算法和算法分析

表：时间复杂度和算法运行时间的关系

T(n) / n	O(logn)	O(n)	O(nlogn)	$O(n^2)$	$O(n^3)$	$O(n^5)$	$O(2^n)$	O(n!))
20	4.3us	20us	86.4us	400us	8ms	3.2s	1.05s	771世纪
40	5.3	40	213	1600	64ms	1.7min	12.7天	2.59*1032世纪
60	5.9	60	354	3600	216ms	13min	366世纪	2.64*1066世纪

1.4 算法和算法分析

结论：

（1）当f(n)为对数函数、幂函数、或它们的乘积时，算法的运行时间是可以接受的，称这些算法是有效算法；当f(n)为指数函数或阶乘函数时，算法的运行时间是不可接受的，称这些算法是无效的算法。

（2）随着n值的增大，增长速度各不相同，n足够大时，存在下列关系：

对数函数<幂函数<指数函数

1.4 算法和算法分析

O(1)	常量阶，与n无关
O(log n)	logn阶
O(n)	n阶
O(nlogn)	nlogn阶
$O(n^2)$	平方阶
$O(n^3)$	立方阶
$O(2^n)$	指数阶

尽量少用指数阶的算法

- 说明：除特别指明外，均指最坏情况下的时间复杂度

1.4 算法和算法分析

空间复杂度

（1）存储算法本身所占用的空间

（2）算法的输入/输出数据占用的空间

（3）算法在运行过程中临时占用的辅助空间

- 原地工作：若辅助空间相对于输入数据量是常数，则称此算法是原地工作。
- 若所占空间量依赖于特定的输入，按最坏情况来分析。

算法复杂性分析

- 算法复杂性 = 算法所需要的计算机资源
- 算法的时间复杂性$T(n)$;
- 算法的空间复杂性$S(n)$。

其中n是问题的规模（输入大小）。

算法的时间复杂性

(1) 最坏情况下的时间复杂性

$T_{max}(n) = \max\{ T(I) \mid size(I)=n \}$

(2) 最好情况下的时间复杂性

$T_{min}(n) = \min\{ T(I) \mid size(I)=n \}$

(3) 平均情况下的时间复杂性

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

其中I是问题的规模为n的实例，$p(I)$是实例I出现的概率。

算法渐近复杂性

$T(n) \to \infty$，as $n \to \infty$；$(T(n) - t(n)) / T(n) \to 0$，as $n \to \infty$;

- $t(n)$是$T(n)$的渐近性态，为算法的渐近复杂性。
- 在数学上，$t(n)$是$T(n)$的渐近表达式，是$T(n)$略去低阶项留下的主项。它比$T(n)$简单。

渐近分析的记号Q, O, W

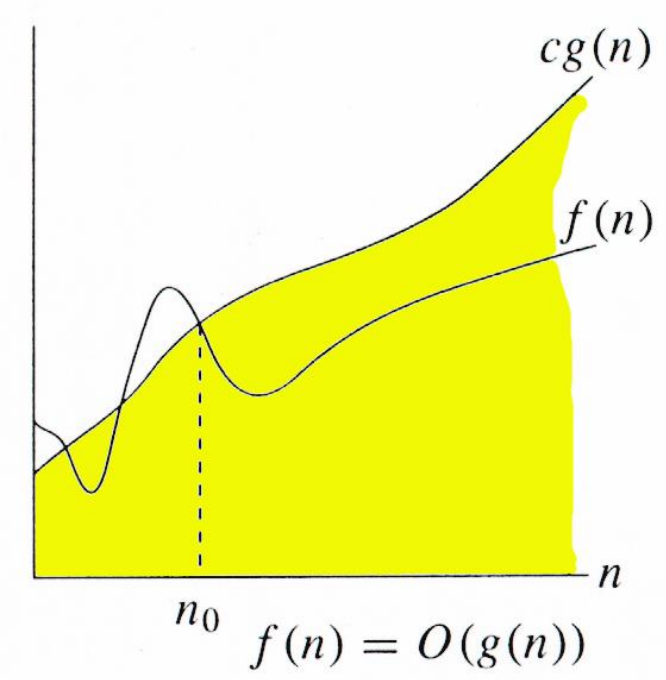
在下面的讨论中，对所有n，$f(n) \geq 0$，$g(n) \geq 0$。

（1）渐近上界记号**O**

$O(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq f(n) \leq cg(n) \}$

$$2n + 3 = O(n^2).$$

$$3n^3 = O(n^4)$$



- 用来作比较的函数g(n)应该尽量接近所考虑的函数f(n). $2n+3=O(n^2)$ 松散的界限；$2n+3=O(n)$ 较好的界限。

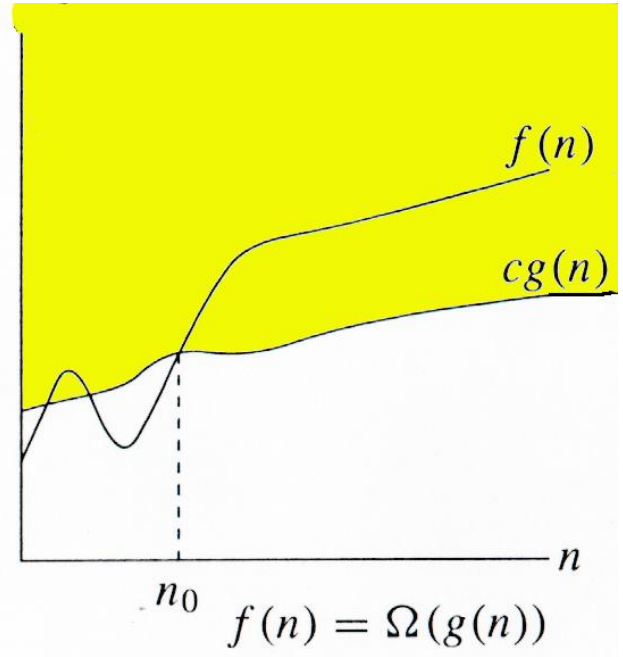
- f(n)=O(g(n))不能写成g(n)=O(f(n))，因为两者并不等价。实际上，这里的等号并不是通常相等的含义。按照定义，用集合符号更准确些。

- O(g(n))={f(n)|f(n)满足：存在正的常数c和n_0，使得当n>=n_0时f(n)<=cg(n)}所以，人们常常把f(n)=O(g(n))读作：“f(n)是g(n)的一个大O成员”。

（2）渐近下界记号Ω

$\Omega(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq cg(n) \leq f(n) \}$

$$3n^3 = \Omega(n^2)$$

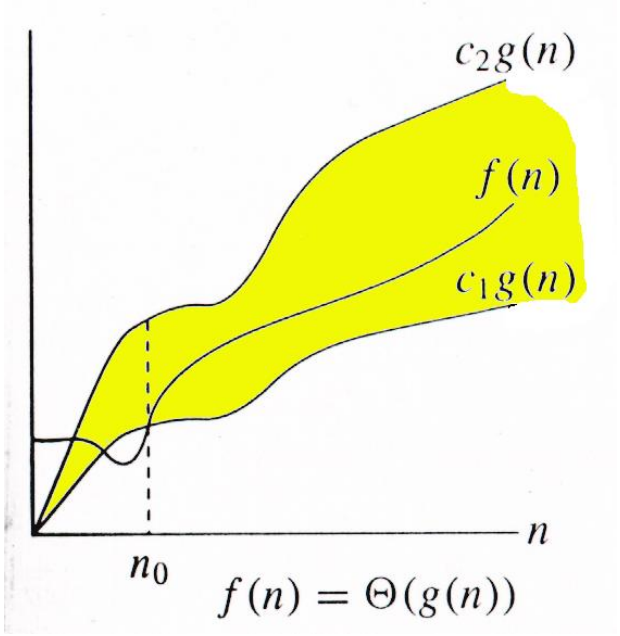


（3）紧渐近界记号Θ

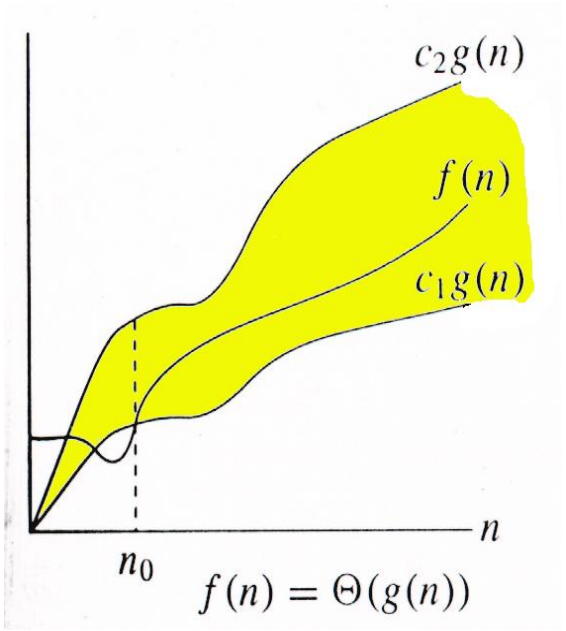
$\Theta(g(n)) = \{ f(n) \mid$ 存在正常数c_1, c_2和n_0使得对所有$n \geq n_0$有：$c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

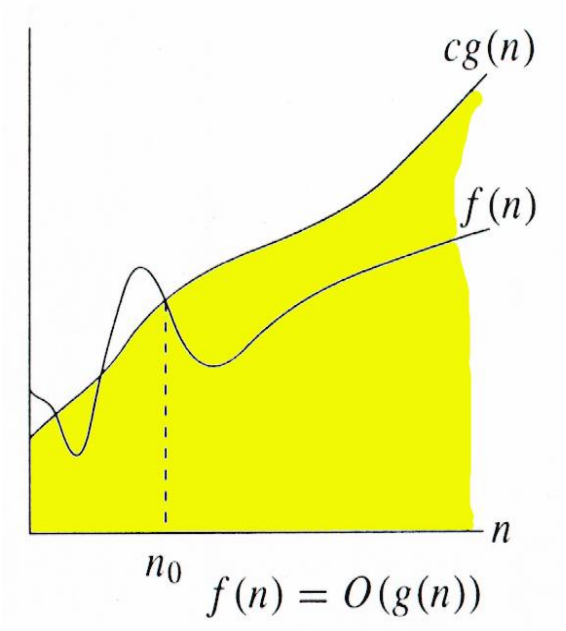
定理1： $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

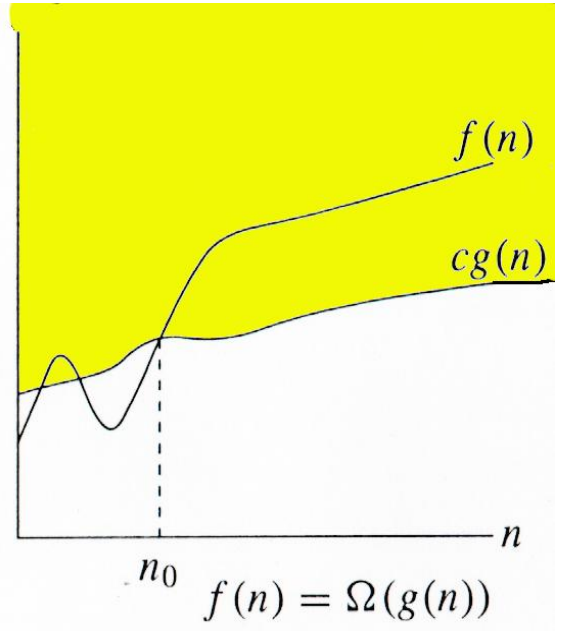
$$10n^2 - 3n = \Theta(n^2)$$



Θ, O, Ω之间的关系







Big-O

略去低阶项和常数系数项留下的主项

略去低阶项

- $4n + 5 \Rightarrow 4n$
- $0.5\ n \log n - 2n + 7 \Rightarrow 0.5\ n \log n$

略去常数系数项

- $4n \Rightarrow n$
- $0.5\ n \log n \Rightarrow n \log n$
- $\log n^2 = 2 \log n \Rightarrow \log n$
- $\log_3 n = (\log_3 2) \log n \Rightarrow \log n$

$2n^2 + 4n = O(n^2)$ ✓

$O(n^2) = 2n^2 + 4n$ ✗

Big-O 实例

$n^2 + 100\,n = O(n^2)$

$(n^2 + 100\,n) \leq 2\,n^2 \quad \text{for } n \geq 10$

$n^2 + 100\,n = \Omega(n^2)$

$n^2 + 100\,n) \geq 1\,n^2 \quad \text{for } n \geq 0$

$n^2 + 100\,n = \theta(n^2)$

$n \log n = O(n^2)$

$n \log n = \theta(n \log n)$

$n \log n = \Omega(n)$

- **插入排序在最坏的情况下需要** $\theta(n^2)$，所以排序是 $O(n^2)$
- 任意的排序算法都需要查看每个元素，所以排序是 $\Omega(n)$.
- 实际上，合并排序在最坏的情况下是 $\theta(n\log n)$

渐近分析记号在等式和不等式中的意义

$f(n)= \Theta(g(n))$的确切意义是：$f(n) \in \Theta(g(n))$。

一般情况下，等式和不等式中的渐近记号$\Theta(g(n))$表示$\Theta(g(n))$中的某个函数。

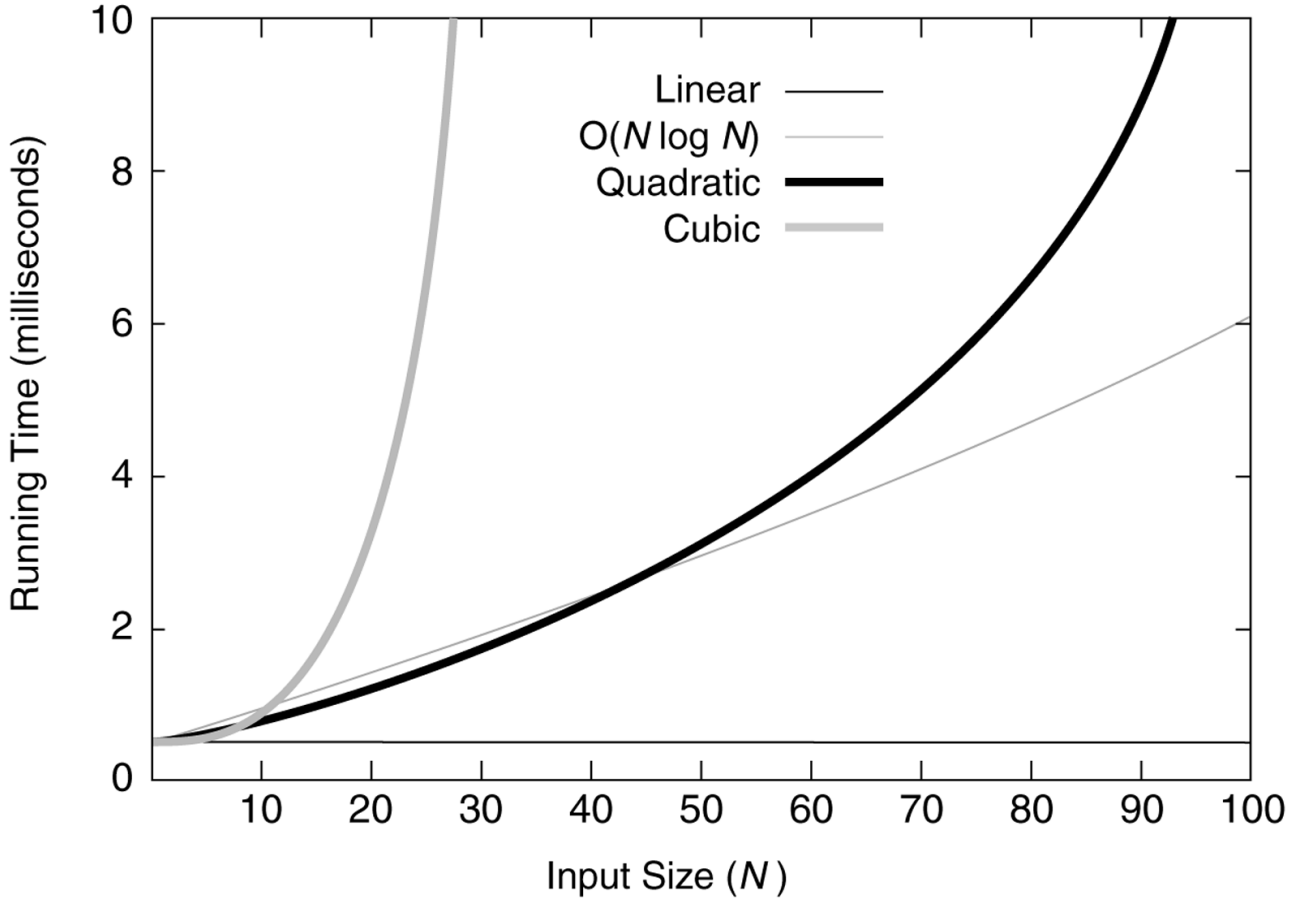
例如：$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ 表示

$2n^2 +3n +1=2n^2 + f(n)$，其中$f(n)$ 是$\Theta(n)$中某个函数。

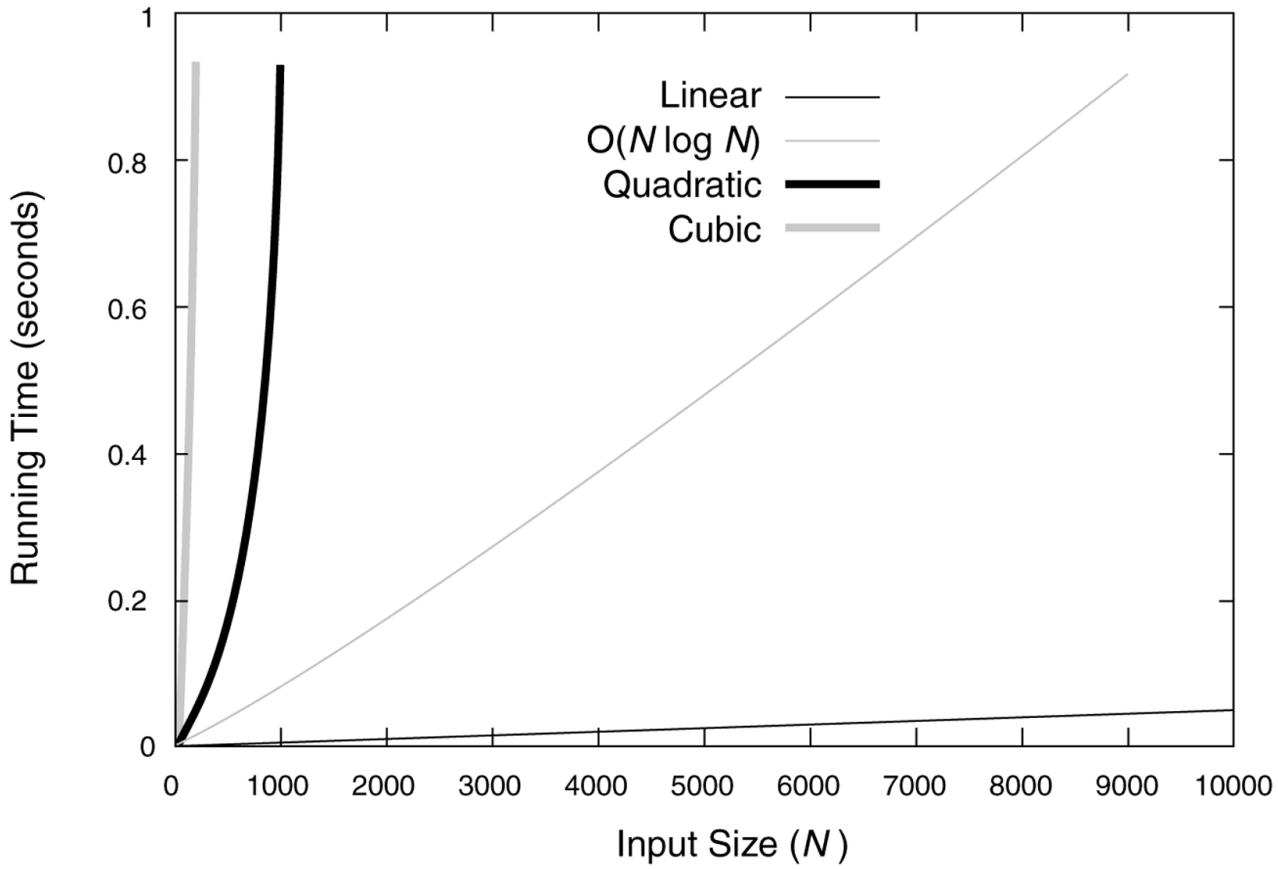
算法分析中常见的复杂性函数

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

小规模数据


Running Time (milliseconds)
Input Size (N)
Linear
O(N log N)
Quadratic
Cubic
0
2
4
6
8
10
10
20
30
40
50
60
70
80
90
100

中等规模数据


Linear
O(N log N)
Quadratic
Cubic
Running Time (seconds)
Input Size (N)
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

分析代码

C++ 操作	常数时间
顺序语句	语句时间和
条件语句	较大分支+条件测试
循环	迭代和
函数调用	函数体代价
递归函数	求解递归方程

嵌套循环

```
for i = 1 to n do
  for j = 1 to n do
    sum = sum + 1
```

$$\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$$

```
for i = 1 to n do
  for j = i to n do
    sum = sum + 1
```

$$\sum_{i=1}^{n}\sum_{j=i}^{n} 1 = \sum_{i=1}^{n}(n-i+1) = \sum_{i=1}^{n}(n+1) - \sum_{i=1}^{n} i =$$

$$n(n+1) - \frac{n(n+1)}{2} = \frac{n(n+1)}{2} \approx n^2$$

算法分析方法

例：顺序搜索算法

```
template<class Type>
int seqSearch(Type *a, int n, Type k)
{
    for(int i=0;i<n;i++)
        if (a[i]==k) return i;
    return -1;
}
```

(1) $T_{\max}(n) = \max\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}=O(n)$

(2) $T_{\min}(n) = \min\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}=O(1)$

(3) 在平均情况下，假设：

(a) 搜索成功的概率为$p\ (0 \le p \le 1)$；

(b) 在数组的每个位置$i\ (0 \le i < n)$搜索成功的概率相同，均为 p/n。

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

$$= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}\right) + n \cdot (1-p)$$

$$= \frac{p}{n}\sum_{i=1}^{n} i + n(1-p) = \frac{p(n+1)}{2} + n(1-p)$$