算法效率分析基础

讲授者 王爱娟

aijuan321@foxmail.com

重庆理工大学 计算机科学与工程学院

August 22, 2024

目录

算法分析的基本策略

算法分析主要是对算法的效率进行分析，包含两方面：

- **时间效率**—运行速度的时间效率
- **空间效率**—占用空间大小的空间效率

对于早期的计算机来说，时间与空间都是极其珍贵的资源

算法分析的基本策略

算法分析主要是对算法的效率进行分析，包含两方面：

- **时间效率**—运行速度的时间效率
- **空间效率**—占用空间大小的空间效率

对于早期的计算机来说，时间与空间都是极其珍贵的资源

空间重要性降低

- 对于早期的计算机，时间和空间两种资源都是非常昂贵的。
- 随着电脑工艺的进步，计算机的存储容量已经提升了好几个数量级，降低了空间效率对算法性能的影响。
- 算法的时间效率没有得到相同程度的提高。

算法分析的基本策略

算法分析主要是对算法的效率进行分析，包含两方面：

- **时间效率**—运行速度的时间效率
- **空间效率**—占用空间大小的空间效率

对于早期的计算机来说，时间与空间都是极其珍贵的资源

空间重要性降低

- 对于早期的计算机，时间和空间两种资源都是非常昂贵的。
- 随着电脑工艺的进步，计算机的存储容量已经提升了好几个数量级，降低了空间效率对算法性能的影响。
- 算法的时间效率没有得到相同程度的提高。

时间的重要性

算法的时间效率分析是算法分析中的关键部分

表示算法时间效率的两个因子

算法分析主要是对算法的效率进行分析，包含两方面：

- 问题的规模（输入规模的度量）

规模的表示

- 大部分算法的执行时间随着输入量的增加而增大。数组排序，矩阵相乘。
- 从逻辑上来说，算法的效率应该是输入规模的函数。
- 我们一般使用n表示问题的输入规模，或者n的对数。

- 在**此规模下**的运行时间
  只有在规模确定的条件下，计算算法的运行时间，才能有效地反应出算法的时间效率。

运行时间的度量单位

- 算法的三要素：数据，数据的操作，操作的控制
- 数据—仅占用空间，基本不消耗时间
- 控制本质也是一种操作，比如for(i=0;i<n;i++)

运行时间的度量单位

- 算法的三要素：数据，数据的操作，操作的控制
- 数据—仅占用空间，基本不消耗时间
- 控制本质也是一种操作，比如for(i=0;i<n;i++)

导致时间开销的主要因素

- 操作是导致算法时间开销的主要因素
- 统计算法的操作次数是衡量算法时间效率的一个重要方式
- 操作次数是衡量算法的一种度量单位

运行时间的度量单位

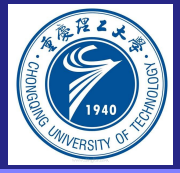
- 算法的三要素：数据，数据的操作，操作的控制
- 数据—仅占用空间，基本不消耗时间
- 控制本质也是一种操作，比如for(i=0;i<n;i++)

导致时间开销的主要因素

- 操作是导致算法时间开销的主要因素
- 统计算法的操作次数是衡量算法时间效率的一个重要方式
- 操作次数是衡量算法的一种度量单位

- 但是操作有很多种：
  - + - * /
  - 赋值
  - ···

统计算法每一种操作的次数，有困难，也不可行！



运行时间的度量单位

- 统计算法中最重要的操作—**基本操作**的执行次数
- 基本操作（basic operation）—时间开销最大的操作

运行时间的度量单位

- 统计算法中最重要的操作—**基本操作**的执行次数
- 基本操作（basic operation）—时间开销最大的操作

基本操作

- + - * /, /开销最大，*次之
- 排序的基本操作：比较
- 矩阵乘法的基本操作：乘法

运行时间的度量单位

- 统计算法中最重要的操作—**基本操作**的执行次数
- 基本操作（basic operation）—时间开销最大的操作

基本操作

- + - * /, /开销最大，*次之
- 排序的基本操作：比较
- 矩阵乘法的基本操作：乘法

- 执行次数C(n)表示是输入规模n时，**基本操作**的次数
- 算法运行时间$T(n) \approx c_{op} * C(n)$, c_{op} 是一个常数因子

算法的比较

那个熊是安全？

算法的比较

那个熊是安全？

那个熊是安全？

- 并不一定要跑的最快才更安全，只要跑的比你快，就安全！
- 算法的分析也与之类似，只要比其他算法好就行

增长次数—算法优劣的度量

为何要对大规模的输入强调执行次数的增长次数呢？

- 增长次数——n的变化与T(n) 的变化之间的关系
- $T(n)$和$T'(n)$：两个算法的时间效率

基本操作

例如，考虑$T(n) = n, T'(n) = n!$，当$n = 1$时，它们的时间效率是一样的，可是它们真的一样吗？

- 当n较小时，$T(n)$和$T'(n)$ 之间的差距并不明显，无法明显地衡量它们的不同或好与坏
- 只有当n较大时才能展示它们之间的差异，增长次数才能成为衡量算法时间好坏的一个重要指标

Note: 小规模输入在运行时间上的差别不足以将高效的算法和低效的算法区分开来。

常见函数的增长次数

- 如果算法运行时间T(n)已知，T(n)与n之间的动态关系需进一步明确
- 假设$T(n) = \log_2 n, n\log_2 n, n^2, n^3, 2^n, n!$

表：增长次数

n	$\log_2 n$	$n\log_2 n$	n^2	n^3	2^n	n!
10	3.3	3.3×10	10^2	10^3	10^3	3.6×10^6
10^2	6.6	6.6×10^2	10^4	10^6	1.3×10^{30}	3.6×10^{157}
10^3	10	1.0×10^4	10^6	10^9		
10^5	17	1.7×10^6	10^{10}	10^{15}		

注：当n增长为$2n$时，增长分别是：log_2n为1倍，n为2倍，$nlog_2n$为2倍多，n^2为4倍...

注意：对数函数的操作次数依赖于对数的低，由于$log_an = log_ab * log_bn$，因此底不相同的两个对数函数只差一个乘法常量，当我们关心增长次数的时候，忽略对数的底，简单写成$logn$。

算法的最优、最差和平均效率

查找效率

- 函数search查找数组A中整数10
- 最好情况：A=[10,1,2,3,4,5,6]
- 最坏情况：A=[1,2,3,4,5,6,10]
- 平均情况：A=[1,2,3,10,4,5,6]

算法的最优、最差和平均效率

查找效率

- 函数search查找数组A中整数10
- 最好情况：A=[10,1,2,3,4,5,6]
- 最坏情况：A=[1,2,3,4,5,6,10]
- 平均情况：A=[1,2,3,10,4,5,6]

同一个算法，不同的效率

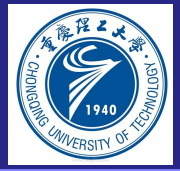
上述例子表明，同一函数(search函数)针对不同的输入(数组A)，它的时间效率是不一样的！

算法最优、最差和平均效率

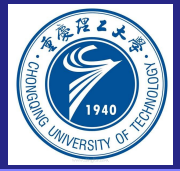
- 算法最差效率（worst-case efficiency）：当输入规模为n时，算法在最坏情况下的效率
- 算法最优效率（best-case efficiency）：当输入规模为n时，算法在最理想情况下的效率。
- 算法平均效率（average-case efficiency）：在“随机”或“典型”输入时（规模仍为n），算法的效率。
- 平均效率的研究方法：一般将规模为n的实例分为几种类型，在假设各种输入的概率分布，推导出基本操作的平均次数

小结：算法分析框架的要点

- 算法的时间效率与空间效率与算法输入相关，都用输入规模的函数来度量。
- 采用算法基本操作的执行次数对**时间效率**的进行度量；通过计算算法消耗的额外存储单元的数量来度量**空间效率**。
- 输入规模相同时，有些算法的时空复杂度可能会显著不同，因此需要考虑最好情况下、最坏情况下以及一般情况下的算法时间复杂度，即最优效率、最差效率和平均效率。
- 算法的增长次数是衡量算法好坏的关键指标之一。本框架主要关注的是，当算法的输入规模趋于无限大时，其运行时间（消耗的额外空间）函数的增长次数。

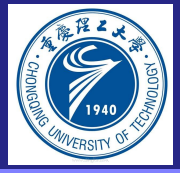


回顾



回顾

1. 输入规模的度量



回顾

1. 输入规模的度量
2. 运行时间的度量：基本操作次数

回顾

1. 输入规模的度量
2. 运行时间的度量：基本操作次数
3. 增长次数

回顾

1. 输入规模的度量
2. 运行时间的度量：基本操作次数
3. 增长次数
4. 算法的最优、最差和平均效率

回顾

1. 输入规模的度量
2. 运行时间的度量：基本操作次数
3. 增长次数
4. 算法的最优、最差和平均效率

如何对不同算法的基本操作次数进行比较和归类？

算法增长次数的表示

- 三个渐进符号：O, Ω, Θ
- 常用函数符号：
  - $t(n)$：一个算法运行的时间
  - $C(n)$：基本操作次数
  - $g(n)$：用来比较的函数

注：$t(n)$和$g(n)$可以是定义在自然数集合上的任意非负函数。

上界O的定义

1. 存在常数$c > 0$
2. 非负整数n_0
3. 使得对所有$n \geq n_0$ 有$t(n) \leq cg(n)$

我们称函数$t(n)$包含在$O(g(n))$中，记为$t(n) \in O(g(n))$，也称函数$t(n)$在n充分大时有上界$g(n)$，非正式而言，即$t(n)$的增长次数小于$g(n)$

上界O的定义

O的图解

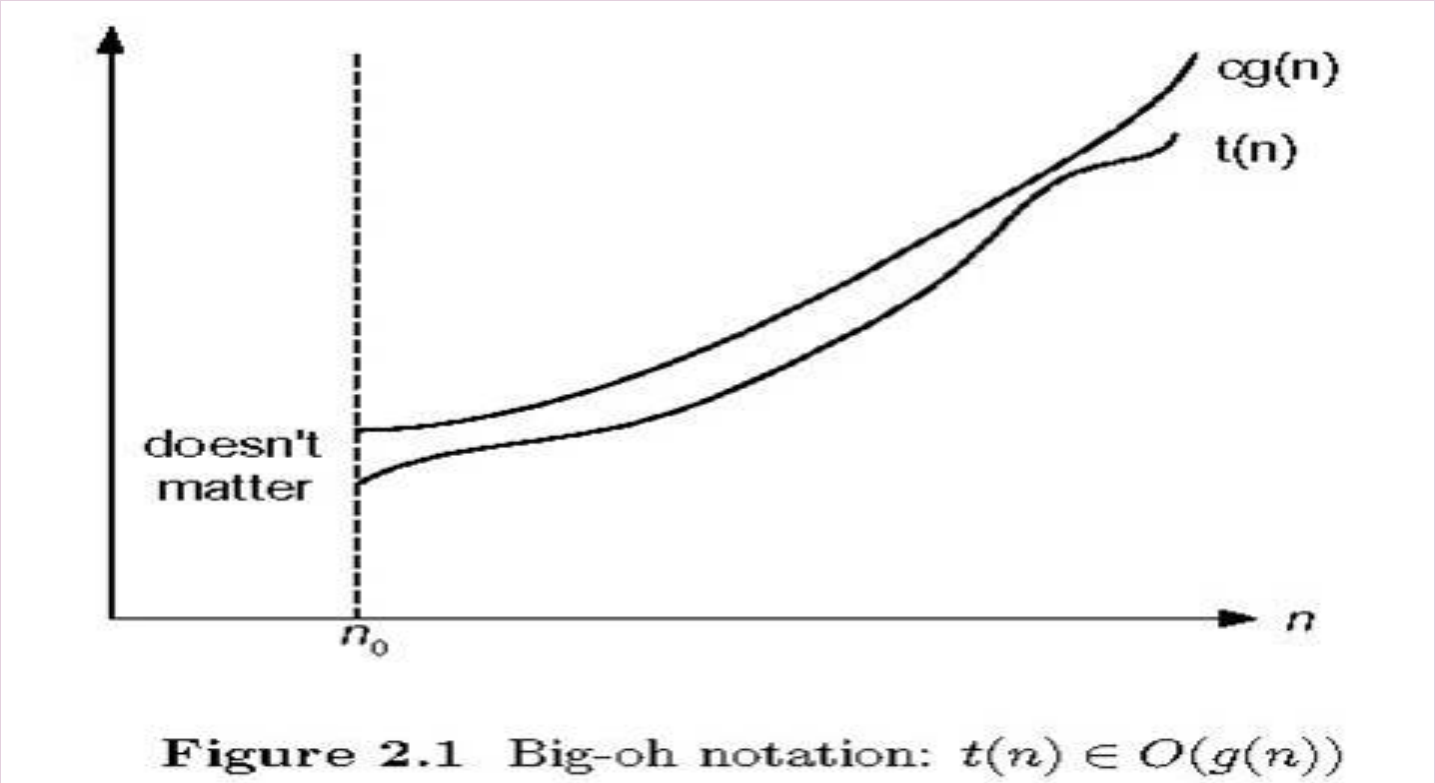


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

例如，$100n + 5 \leq 100n + n\ (n \geq 5\ ) = 101n \leq 101n^2$

$100n + 5$ 的上界是$n^2 \Rightarrow 100n+5 \in O(n^2)$

$4n\log n+7 \in O(n\log n)$, $n^2 + \sin n \in O(n^2)$ 与$n^2 + \log n \in O(n^2)$?

下界Ω的定义

下界Ω的定义

1. 存在常数c>0
2. 非负整数n_0
3. 使得对所有$n \geq n_0$ 并且$t(n) \geq cg(n)$

我们称函数t(n)包含在Ω(g(n))中，记为t(n)∈Ω(g(n))
也称函数t(n)在n充分大时有下界g(n)
非正式而言，即t(n)的增长次数大于等于g(n)

下界Ω的定义

下界Ω的定义

1. 存在常数c>0
2. 非负整数n_0
3. 使得对所有$n \geq n_0$ 并且$t(n) \geq cg(n)$

我们称函数t(n)包含在Ω(g(n))中，记为t(n)∈Ω(g(n))
也称函数t(n)在n充分大时有下界g(n)
非正式而言，即t(n)的增长次数大于等于g(n)

下界Ω的图解

Ω的图解

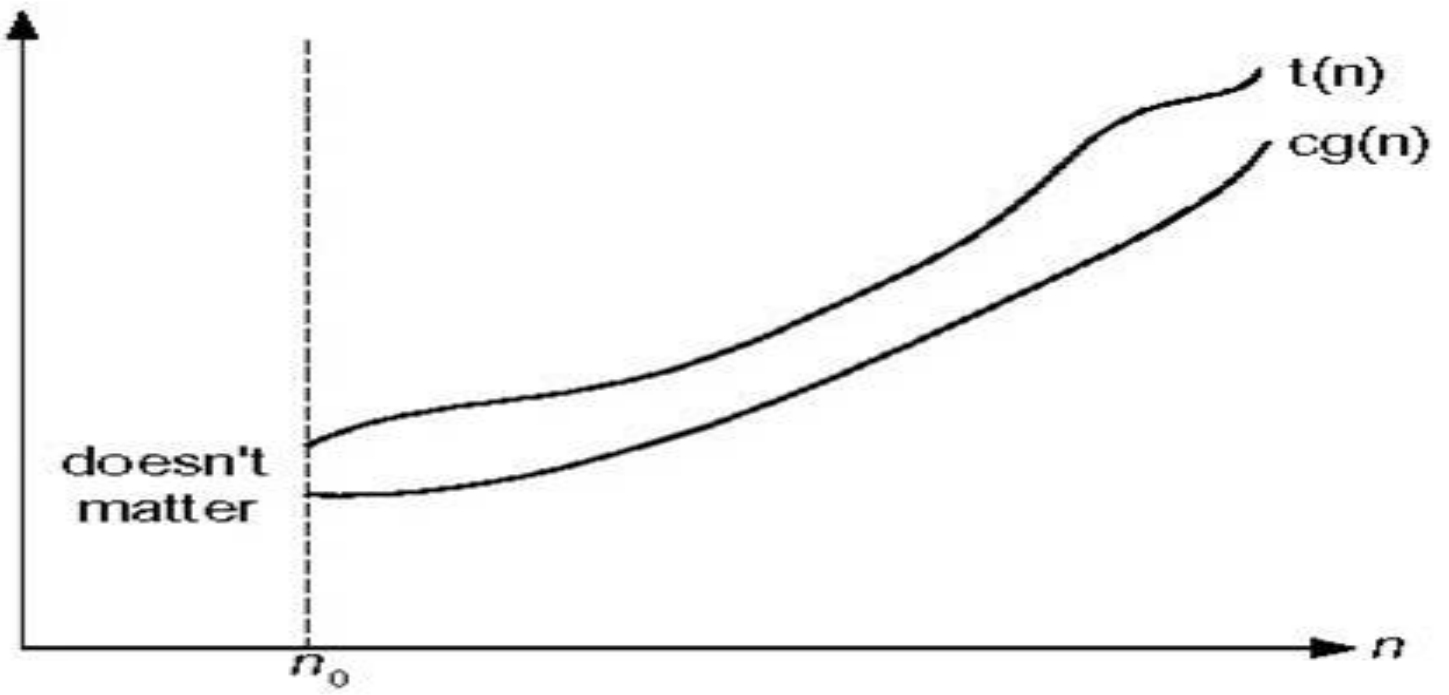


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

例子

如何说明$2n^2 + 11n - 10 \in \Omega(n^2)$？$c$和$n_0$ 等于多少？

下界Ω的图解

Ω的图解

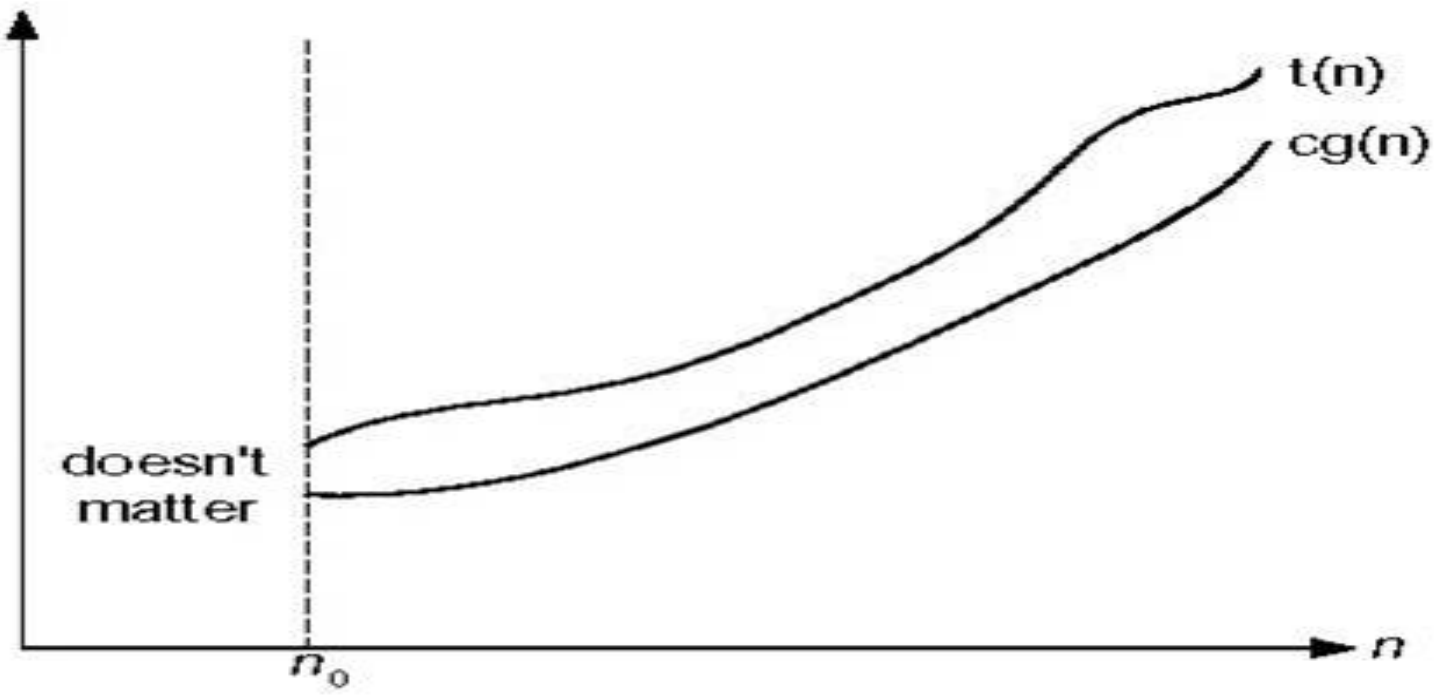


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

例子

如何说明$2n^2 + 11n - 10 \in \Omega(n^2)$？$c$和$n_0$ 等于多少？

$c = 1, 2$，n_0可取值很多例如1.

Θ 的定义

Θ的定义

1. 存在常数$c_1 > 0, c_2 > 0$
2. 非负整数n_0
3. 使得对所有$n \geq n_0$ 并且$c_1 g(n) \leq t(n) \leq c_2 g(n)$

我们称函数$t(n)$包含在$\Theta(g(n))$中，记为$t(n) \in \Theta(g(n))$, 非正式而言，即$t(n)$的增长次数等于$g(n)$.

Θ 的定义

Θ的定义

1. 存在常数$c_1 > 0, c_2 > 0$
2. 非负整数n_0
3. 使得对所有$n \geq n_0$ 并且$c_1 g(n) \leq t(n) \leq c_2 g(n)$

我们称函数$t(n)$包含在$\Theta(g(n))$中，记为$t(n) \in \Theta(g(n))$, 非正式而言，即$t(n)$的增长次数等于$g(n)$.

例子

当c_1, c_2和n_0等于多少，$3n^2 + 16n + 68 \in \Theta(n^2)$?

Θ 的定义

Θ的定义

1. 存在常数$c_1 > 0, c_2 > 0$
2. 非负整数n_0
3. 使得对所有$n \geq n_0$ 并且$c_1 g(n) \leq t(n) \leq c_2 g(n)$

我们称函数$t(n)$包含在$\Theta(g(n))$中，记为$t(n) \in \Theta(g(n))$, 非正式而言，即$t(n)$的增长次数等于$g(n)$.

例子

当c_1, c_2和n_0等于多少，$3n^2 + 16n + 68 \in \Theta(n^2)$?

c_1可取值$1, 2, 3$，c_2可取值$4, 5, \ldots$;

Θ的图解

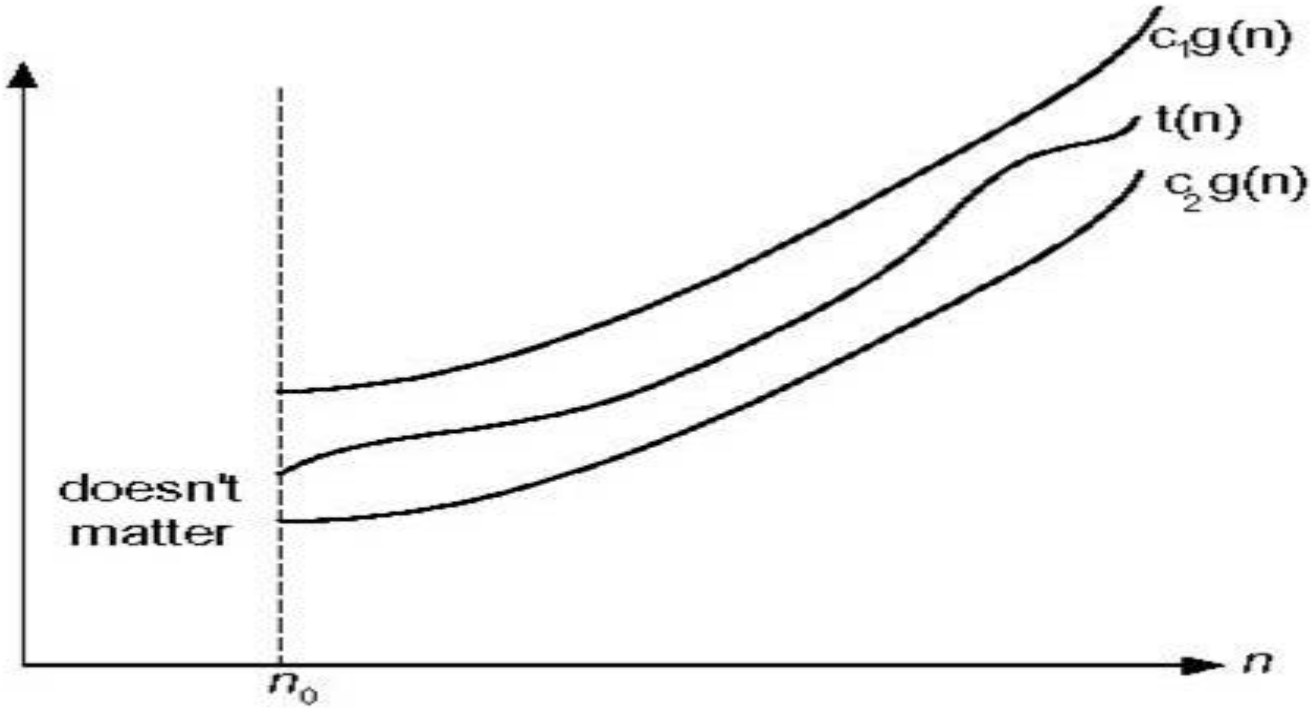


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

渐进符号的有用特性

运算法则：

- $O(f)+O(g)=O(max(f,g))$
- $O(f)*O(g)=O(f*g)$
- 如果$g(n)\in O(f(n))$,则$O(f)+O(g)=O(f)$
- $O(cf(n))=O(f(n))$

渐进符号的有用特性

运算法则：

- $O(f)+O(g)=O(max(f,g))$
- $O(f)*O(g)=O(f*g)$
- 如果$g(n)\in O(f(n))$,则$O(f)+O(g)=O(f)$
- $O(cf(n))=O(f(n))$

Ω与Θ

对符号Ω与Θ，有类似结论

渐进符号的有用特性

运算法则：

- $O(f)+O(g)=O(max(f,g))$
- $O(f)*O(g)=O(f*g)$
- 如果$g(n)\in O(f(n))$,则$O(f)+O(g)=O(f)$
- $O(cf(n))=O(f(n))$

Ω与Θ

对符号Ω与Θ，有类似结论

定理

如果$t_1(n)\in f(n), t_2(n)\in g(n)$，则$t_1(n)+t_2(n)\in O(\max\{f(n),g(n)\})$

证明：见$P43$

利用极限比较增长次数

利用极限比较增长次数

定理

设f和g是定义域为自然数集合N上的非负函数，有：

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\begin{cases}0: t(n)\in O(g(n))\\ c>0: t(n)\in O(g(n)); t(n)\in \Omega(g(n)); t(n)\in \Theta(g\\ \infty: t(n)\in \Omega(g(n))\end{cases}$$

第一种情况表明$t(n)$的增长次数比$g(n)$小

第二种情况表明$t(n)$的增长次数和$g(n)$相同

第三种情况表明$t(n)$的增长次数比$g(n)$大；

例子:$\log_2 n$和$\sqrt{n}$

利用极限比较增长次数

利用极限比较增长次数

定理

设f和g是定义域为自然数集合N上的非负函数，有：

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\begin{cases}0: t(n)\in O(g(n))\\ c>0: t(n)\in O(g(n)); t(n)\in \Omega(g(n)); t(n)\in \Theta(g\\ \infty: t(n)\in \Omega(g(n))\end{cases}$$

第一种情况表明$t(n)$的增长次数比$g(n)$小

第二种情况表明$t(n)$的增长次数和$g(n)$相同

第三种情况表明$t(n)$的增长次数比$g(n)$大；

例子:$\log_2 n$和$\sqrt{n}$

$$\lim_{n\to\infty}\frac{\log_2 n}{\sqrt{n}}=\lim_{n\to\infty}\frac{(\log_2 n)'}{(\sqrt{n})'}=\lim_{n\to\infty}\frac{(\log_2 e)\frac{1}{n}}{\frac{1}{2\sqrt{n}}}=2\log_2 e\lim_{n\to\infty}\frac{1}{\sqrt{n}}=0$$

表明$\log_2 n$的增长次数比$\sqrt{n}$小，且有$\log_2 n\in O(\sqrt{n})$

常见的渐进(增长次数)时间效率类型

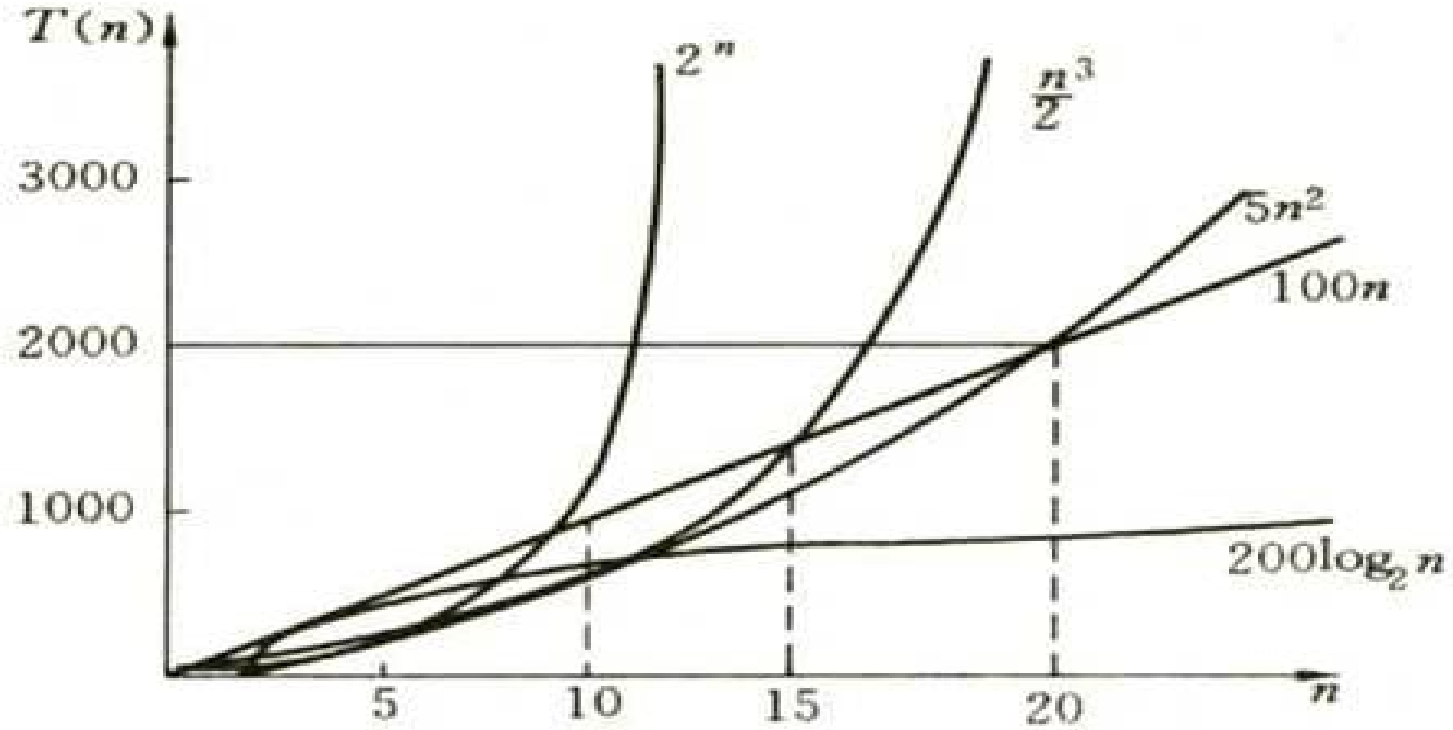
- 常见的渐进(增长次数)时间效率类型

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

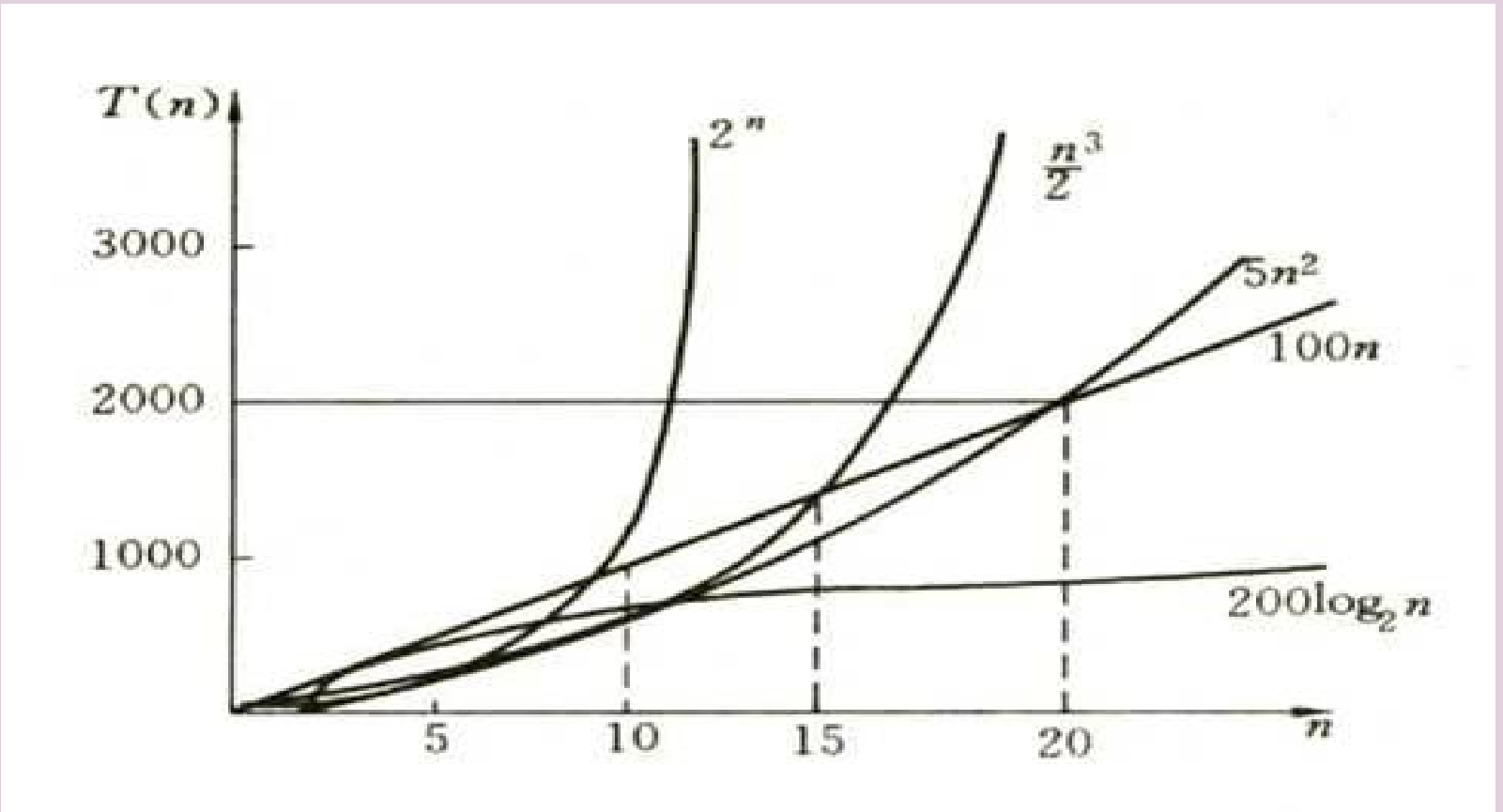
表：增长次数

$T(n) \in O(1)$	Constant
$T(n) \in O(\log n)$	logarithmic
$T(n) \in O(n)$	linear
$T(n) \in O(n \log n)$	$n \log n$
$T(n) \in O(n^2)$	quadratic
$T(n) \in O(n^3)$	cubic
$T(n) \in O(2^n)$	exponential
$T(n) \in O(n!)$	factorial

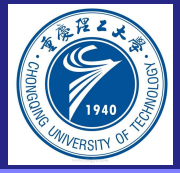
增长次数图解



增长次数图解



习题2.2： 2, 3, 5



非递归算法的时间效率分析

主要内容：

- 例子：求数组的最大元素的算法时效分析
- 非递归算法分析的基本过程
- 例子：矩阵相乘的算法时效分析

非递归算法的时间效率分析

主要内容：

- 例子：求数组的最大元素的算法时效分析
- 非递归算法分析的基本过程
- 例子：矩阵相乘的算法时效分析

- 查找最大值元素算法的时效分析

问题：从n个元素的数组中查找最大值元素

非递归算法的时间效率分析

主要内容：

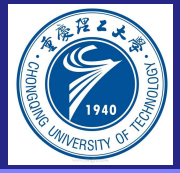
- 例子：求数组的最大元素的算法时效分析
- 非递归算法分析的基本过程
- 例子：矩阵相乘的算法时效分析

- 查找最大值元素算法的时效分析

问题：从n个元素的数组中查找最大值元素

算法伪代码：MaxElement(A[0..n-1]):

maxval $\leftarrow$ A[0]
for $i = 1$ to $n - 1$ **do**
    if A[i] $>$ maxval **then**
        maxval $\leftarrow$ A[i]
    end if
end for
return maxval



查找最大值元素算法的时效分析

- 确定基本操作：是赋值运算还是比较运算？

查找最大值元素算法的时效分析

- 确定基本操作：是赋值运算还是比较运算?
- 求基本操作的执行次数

查找最大值元素算法的时效分析

- 确定基本操作：是赋值运算还是比较运算?
- 求基本操作的执行次数

基本操作的执行次数

- 比较操作A[i] $>$ maxval，为基本操作，该操作比赋值操作时间开销大
- 记$C(n)$为比较操作的执行次数，则 $C(n) = \sum_{i=1}^{n-1} 1 = n - 1$
- $C(n) \in \Theta(n)$?

查找最大值元素算法的时效分析

- 确定基本操作：是赋值运算还是比较运算?
- 求基本操作的执行次数

基本操作的执行次数

- 比较操作A[i] $>$ maxval，为基本操作，该操作比赋值操作时间开销大
- 记$C(n)$为比较操作的执行次数，则 $C(n) = \sum_{i=1}^{n-1} 1 = n - 1$
- $C(n) \in \Theta(n)$?

$C(n)$为什么$\in \Theta(n)$

是否存在c_1, c_2和n_0，当$n \geq n_0$ 时，有$c_1 n \leq n \leq c_2 n$.

分析非递归算法效率的通用方法

分析的一般步骤：

- 决定哪些参数作为输入规模的度量
- 找出算法的基本操作
- 检查基本操作的执行次数的表达式
- 确定基本操作执行次数的上界与下界

计算两个n 阶方阵乘积的例子

算法伪代码：MaxtrixMultiplication(A[0..n-1,0..n-1],B[0..n-1,0..n-1]):

```
for i = 0 to n-1 do
    for j = 0 to n-1 do
        C[i,j] = 0
        for k = 0 to n-1 do

            C[i,j]+=A[i,k]*B[k,j]
        end for
    end for
end for
return C
```

计算两个n 阶方阵乘积的例子

算法伪代码：MaxtrixMultiplication(A[0..n-1,0..n-1],B[0..n-1,0..n-1]):

```
for i = 0 to n-1 do
    for j = 0 to n-1 do
        C[i,j] = 0
        for k = 0 to n-1 do

            C[i,j]+=A[i,k]*B[k,j]
        end for
    end for
end for
return C
```

算法时效分析

- 基本操作是什么？乘法
- 基本操作的次数是多少：$\Sigma_{i=0}^{n-1}\Sigma_{j=0}^{n-1}\Sigma_{k=0}^{n-1}1 = n^3$
- 算法的时效：$C(n) = n^3 \in \Theta(n^3)$