FSE598 前沿计算技术

模块 3 算法设计与分析
单元 1 算法基础
第2讲　渐进符号

本课程的部分内容是基于 Thomas H.Cormen、Charles E.Leiserson 等“算法简介”这篇课文

课程概要

学习内容

- 为什么算法很重要？
- 解决同一问题的不同算法
- 运算数量的计数
- 渐进符号
- 算法复杂度：执行时间的增长阶
- 算法的效率

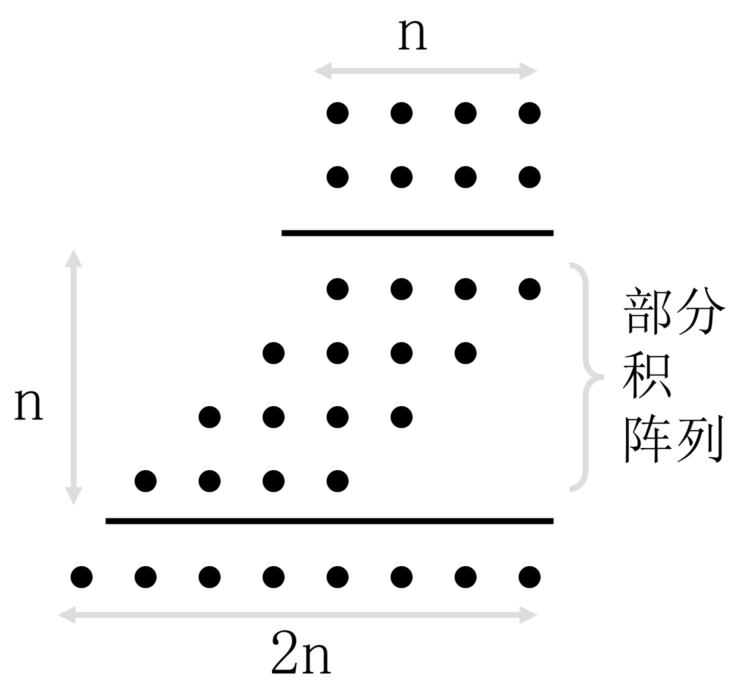
为什么算法很重要？

考虑将两个正整数 X 和 Y 相乘

- 示例：55 * 17
  - 传统算法

```
    55
x   17
------
   385
+  55
------
   935
```

- 复杂度是多少？
  - 假设 X 和 Y 都有 n 位数
  - 复杂度是：n^2



对于较大的 n，一种算法可能比另一种算法更快

- 示例：55 * 17
  - 另一种算法

（如果第 1 列中的相应数字为奇数，则第 3 列采用第 2 列中的数字）

第 1 列 乘数 (X÷2)	第 2 列 被乘数 (Y*2)	第 3 列 结果 X*Y
55	17	17
27	34	34
13	68	68
6	136	
3	272	272
1	544	+ 544
		935

- 假设 X 和 Y 有 n 位数，则复杂度是多少？
  - 很多整数乘法算法的效率要优于 n^2。
  - 我们会在 CSE551 介绍其中一些算法。

运算次数

通常，我们会用运算次数来比较算法。

考虑以下 3 个程序：

程序（1）

```
x ← a + b
y ← c - d
```

2 次运算

程序（2）

```
for i ← 1 to n do
    x ← a + b
    y ← c - d
end
```

$2*n$ 次运算

程序（3）

```
for i ← 1 to n do
    for j ← 1 to n do
        x ← a + b
        y ← c - d
    end
end
```

$2*n^2$ 次运算

这两条语句的运算计数分别为 2、$2n$、$2n^2$

不管我们用什么机器来运行这些程序，我们都知道

- 程序（2）的执行时间是程序（1）执行时间的 n 倍。
- 程序（3）的执行时间是程序（1）执行时间的 n^2 倍。

渐进符号

关于算法的运行时间（运算计数）函数 $T(n)$ ：

- 我们其实不关注：
  - 与机器相关的常数系数
  - n 很小的情况 — 如果 n 很小，则任何算法都应该能在短时间内运行完成
- 我们关注的其实是：
  - 函数的增长阶，如 $\log n, n, n^2, 2^n$, 等。
  - 当 n 较大或非常大的情况。

$$an+b \Rightarrow \Theta(n)$$

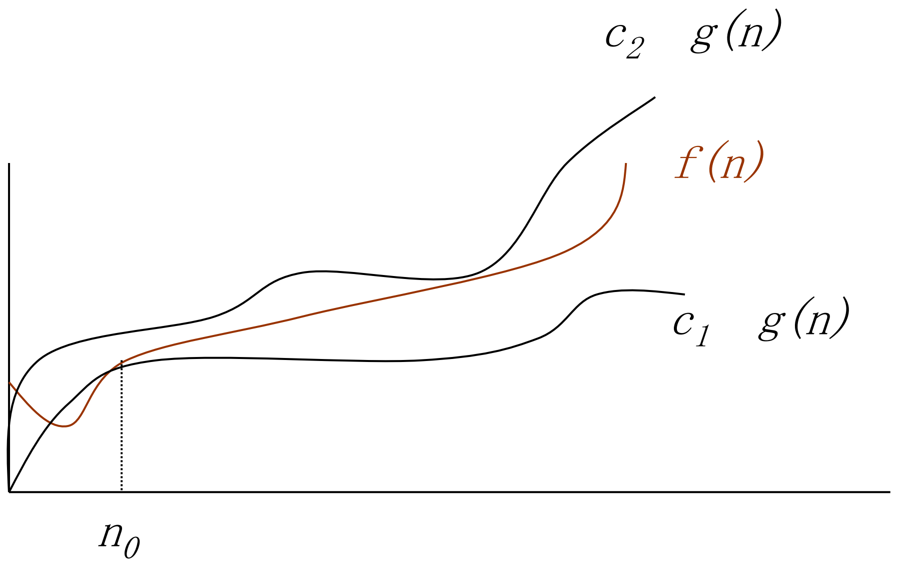
$$an^2+bn+c \Rightarrow \Theta(n^2)$$

渐进符号

渐进符号的定义：Θ、O、Ω、o、ω

Θ 符号 $g(n)$ 表示的是 $f(n)$ 的紧界：

$\Theta(g(n)) = \{f(n)$：存在正常数 c_1, c_2 和 n_0，使得对于所有 $n \geqslant n_0$ 的情况，满足 $0 \leqslant c_1\, g(n) \leqslant f(n) \leqslant c_2\, g(n)\}$



证明示例：求三个常数 c_1, c_2 和 n_0

示例：证明 $(\frac{1}{4})n^2 + 5n = \Theta(n^2)$

$\Rightarrow c_1 n^2 \le (\frac{1}{4})n^2 + 5n \le c_2 n^2$

$\Rightarrow c_1 \le (\frac{1}{4}) + 5/n \le c_2$

右边：$n \geqslant 5$ 且 $c_2 \geqslant 1\frac{1}{4}$

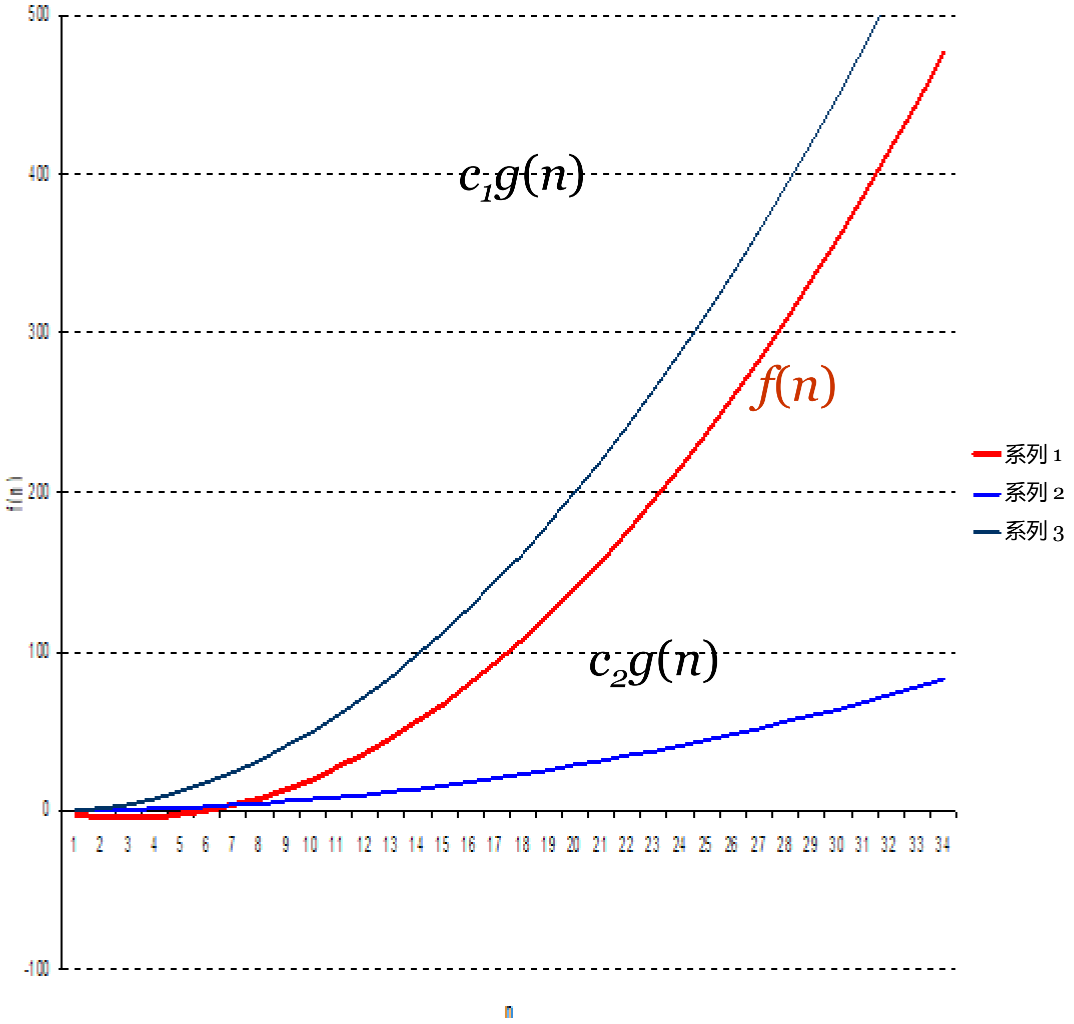
左边：$n \geqslant 1$ 且 $c_1 \leqslant \frac{1}{4}$

参数：

$c_1 \leqslant \frac{1}{4}$, $c_2 \geqslant 1\frac{1}{4}$, $n_0 = 5$

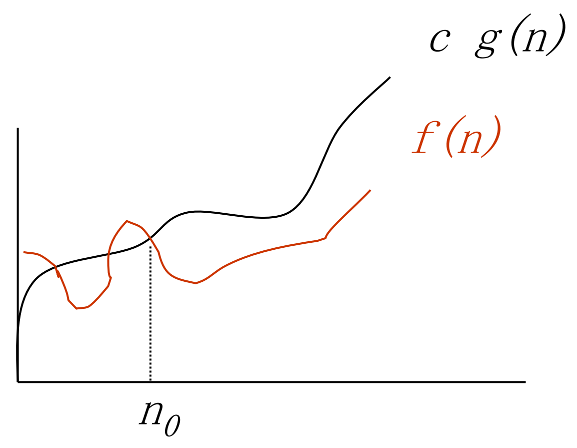
即：

$(\frac{1}{4})n^2 \leqslant (\frac{1}{4})n^2 + 5n \leqslant (1\frac{1}{4})n^2$

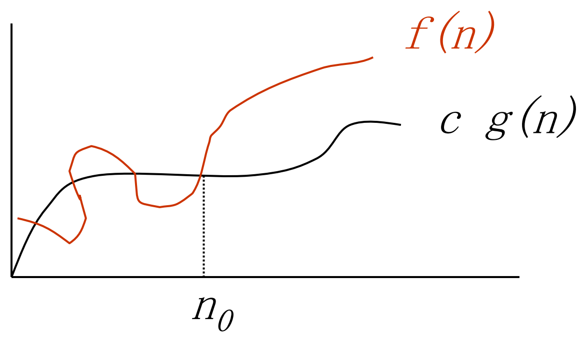


O 符号 — f(n) 的上界；Ω 符号 — f(n) 的下界

- $O(g(n)) = \{f(n)$ ：存在正常数 c 和 n_0，使得对于所有 $n \geq n_0\}$ 的情况，满足 $0 \leqslant f(n) \leqslant c\ g(n)$



- $\Omega(g(n)) = \{f(n)$ ：存在正常数 c 和 n_0，使得对于所有 $n \geq n0\}$ 的情况，满足 $0 \leqslant c\ g(n) \leqslant f(n)$



o- 和 ω- 符号表示的是 f(n) 的紧界

- O(g(n)) 表示的是 f(n) 的上界，但不一定是紧界。
- o(g(n)) 表示的是 f(n) 的上界，但**不是**紧界。
- Ω (g(n)) 表示的是 f(n) 的下界，但不一定是紧界。
- ω (g(n)) 表示的是 f(n) 的下界，但**不是**紧界。

- $\Omega(g(n)) \leq \Theta(g(n)) \leq O(g(n))$
- $\omega(g(n)) < \Theta(g(n)) < o(g(n))$

Θ-、O- 和 Ω 符号练习

- $(1/4)n^2 + 5n = \Theta(n^2)$　　是的，我们已经证明了。
  $(1/4)n^2 + 5n = O(n^2)$　　？
  $(1/4)n^2 + 5n = \Omega(n^2)$
- $(1/4)n^2 + 5n = \Theta(n^3)$
  $(1/4)n^2 + 5n = O(n^3)$
  $(1/4)n^2 + 5n = \Omega(n^3)$
- $(1/4)n^2 + 5n = \Theta(n)$
  $(1/4)n^2 + 5n = O(n)$
  $(1/4)n^2 + 5n = \Omega(n)$
- $(1/4)n^2 + 5n = o(n^2)$
  $(1/4)n^2 + 5n = \omega(n^3)$

定理

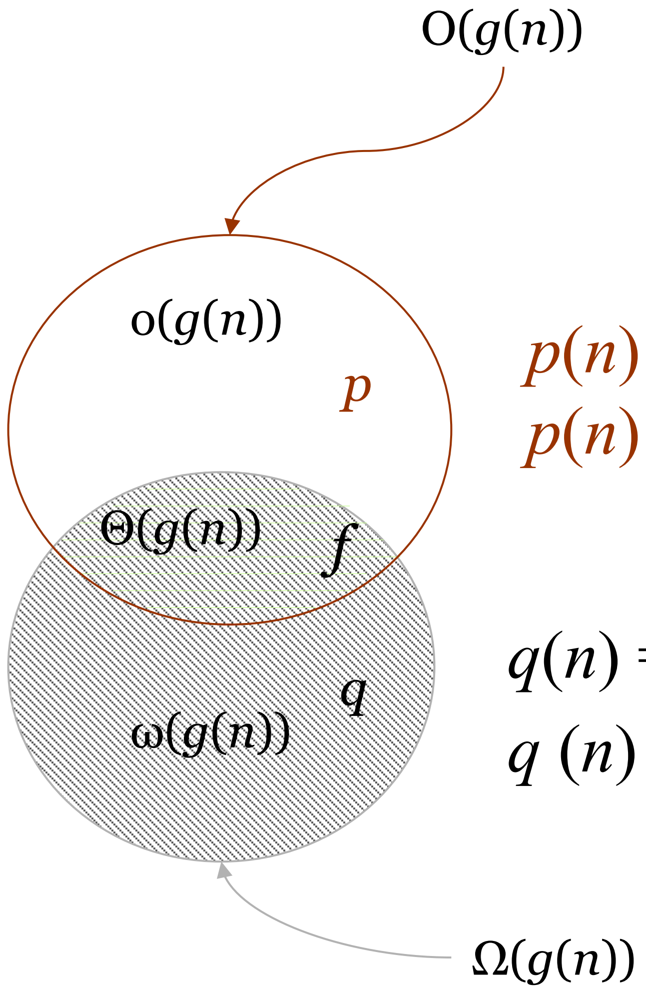
对于任意两个函数 $f(n)$ 和 $g(n)$,
$f(n) = \Theta(g(n))$,
当且仅当
$f(n) = O(g(n))$ 和
$f(n) = \Omega(g(n))$

渐进符号之间的关系

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

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

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



$p(n) = o(g(n))$

$p(n) = O(g(n))$

$q(n) = \omega(g(n))$

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

$o(g(n)) \cap \omega(g(n)) = \Phi$

$o(g(n)) \cap \Theta(g(n)) = \Phi$

$\omega(g(n)) \cap \Theta(g(n)) = \Phi$

$O(g(n)) \cap \Theta(g(n)) = \Theta(g(n))$

$\Omega(g(n)) \cap \Theta(g(n)) = \Theta(g(n))$

$O(g(n)) \cup \Theta(g(n)) = O(g(n))$

$\Omega(g(n)) \cup \Theta(g(n)) = \Omega(g(n))$

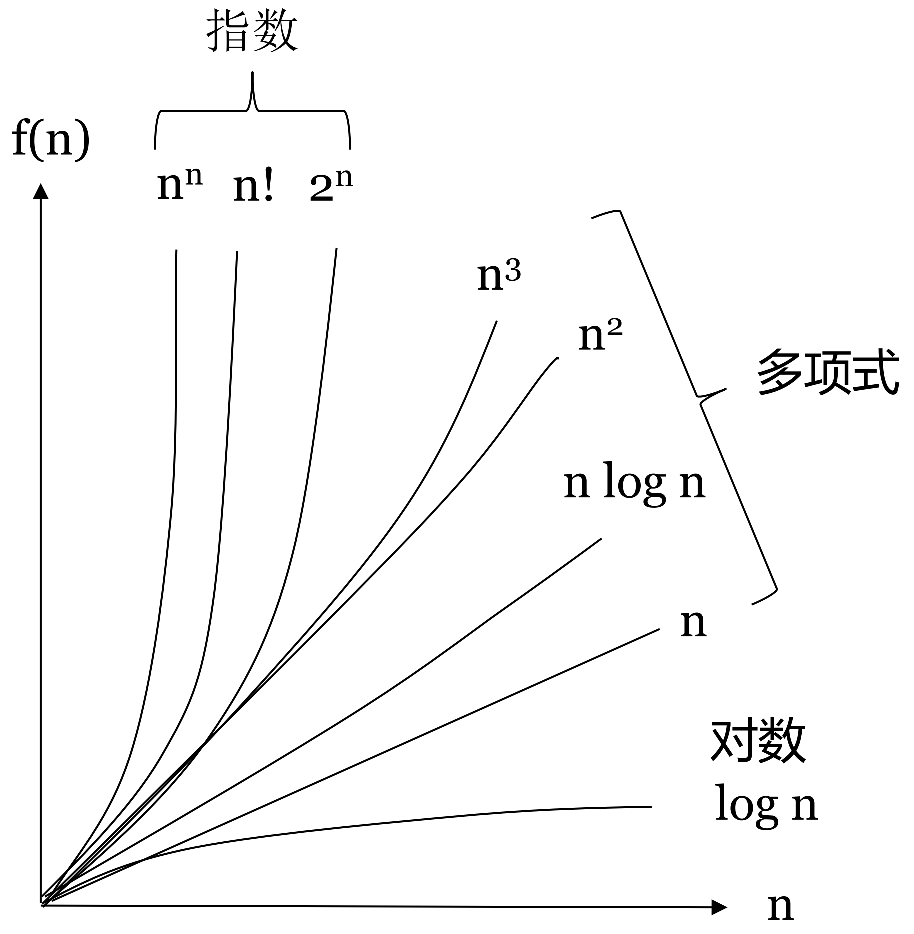
$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$

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

$f(n) = O(g(n)) \not\Rightarrow f(n) = \Theta(g(n))$

$f(n) = \Omega(g(n)) \not\Rightarrow f(n) = \Theta(g(n))$

增长阶：对数 < 多项式 < 指数



$$\sum_{i=0}^{d} c_i n^i = O(n^d) = \Theta(n^d)$$

增长率中的低阶项可以忽略，
因为对于较大的 n 而言，这些项不重要
我们不考虑常数因子
e.g., $\Theta(5n^4) = \Theta(n^4)$
和 $\Theta(5n^4 + 8n^3 + 2n^2 + 7n + 6) = \Theta(n^4)$

$\Theta(\lg^a n) < \Theta(n)$, and $(n^b) < c^n$, 其中 a, b, c >1

Θ 是最好的。为什么用 O 符号?

- 要想求出 $\Theta(g(n))$，我们得证明 $O(g(n)) = \Omega(g(n))$
- 求 $\Omega(g(n))$ 可能会很困难
- 在很多情况下，只要知道上界就够了
- 示例：

$$f(n) = \sum_{k=1}^{n} \left(\frac{1}{k} \right)$$

我们很容易知道：$f(n) \le n = O(n)$

我们甚至可以证明：$f(n) \le \lg(n) = O(\lg n)$

但我们无法证明：$f(n) = \Theta(n)$

渐近符号之间的关系性质

- 实数的很多关系性质适用于渐近符号：
  - **传递性**

    $f(n) = O(g(n))$ 且 $g(n) = O(h(n))$ 意味着 $f(n) = O(h(n))$

    $f(n) = \Theta(g(n))$ 且 $g(n) = \Theta(h(n))$ 意味着 $f(n) = \Theta(h(n))$

    $f(n) = \Omega(g(n))$ 且 $g(n) = \Omega(h(n))$ 意味着 $f(n) = \Omega(h(n))$

    $f(n) = o(g(n))$ 且 $g(n) = o(h(n))$ 意味着 $f(n) = o(h(n))$

    $f(n) = \omega(g(n))$ 且 $g(n) = \omega\ (h(n))$ 意味着 $f(n) = \omega(h(n))$
  - **自返性**

    $f(n) = O(f(n))$ $f(n) = \Theta(f(n))$ $f(n) = \Omega(f(n))$
  - **对称性**

    $f(n) = \Theta(g(n))$ 当且仅当 $g(n) = \Theta(f(n))$
  - **转置性**

    $f(n) = O(g(n))$ 当且仅当 $g(n) = \Omega(f(n))$

    $f(n) = o(g(n))$ 当且仅当 $g(n) = \omega(f(n))$

算法的效率

- 我们认为，如果某种算法在最坏情况下的运行时间具有较低的增长阶，则该算法比另一种算法更高效
- 这种评估方法可能不适用于输入较小的情况，但适用于输入足够大的情况
- 例如：就增长阶而言，
  - 算法 A：$800n^2 + 500n + 1000 \lg n + 9$
  - 算法 B：$8n^3 - 1$
  - 算法 C：2^n
  - 算法 D：$(¼)\ n!$
- 最高效和最低效的算法分别是哪个？

算法的效率（接上页）

- 通常，如果算法在多项式时间内运行，比如 O（g（n）），g（n）是一个多项式表达式，我们则认为该算法效率高。

  算法 A：$800n^9 + 500n^6 + 400n^2 + 9$

  算法 B：$8n^3 + 1$

- 通常，如果算法无法在多项式时间内运行，比如 $\Omega(g(n))$，g（n）是一个指数表达式，我们则认为该算法效率低，

  比如：运行时间为 $\Omega(2^n)$，$\Omega(n!)$，$\Omega(n^n)$ 的算法。