1.3 绪论-大O记号

备注	
学习日	@2020/03/07
视频起始	https://www.bilibili.com/video/av75509584?p=8
讲义	01.Introduction.C.Big_o.pdf

▼ 01-C-1 大O记号

随着计算规模增大，计算成本如何增加

渐进分析：在问题规模足够大后，计算成本如何增长?

Asymptotic analysis：当n>>2后，对于规模为n输入，算法

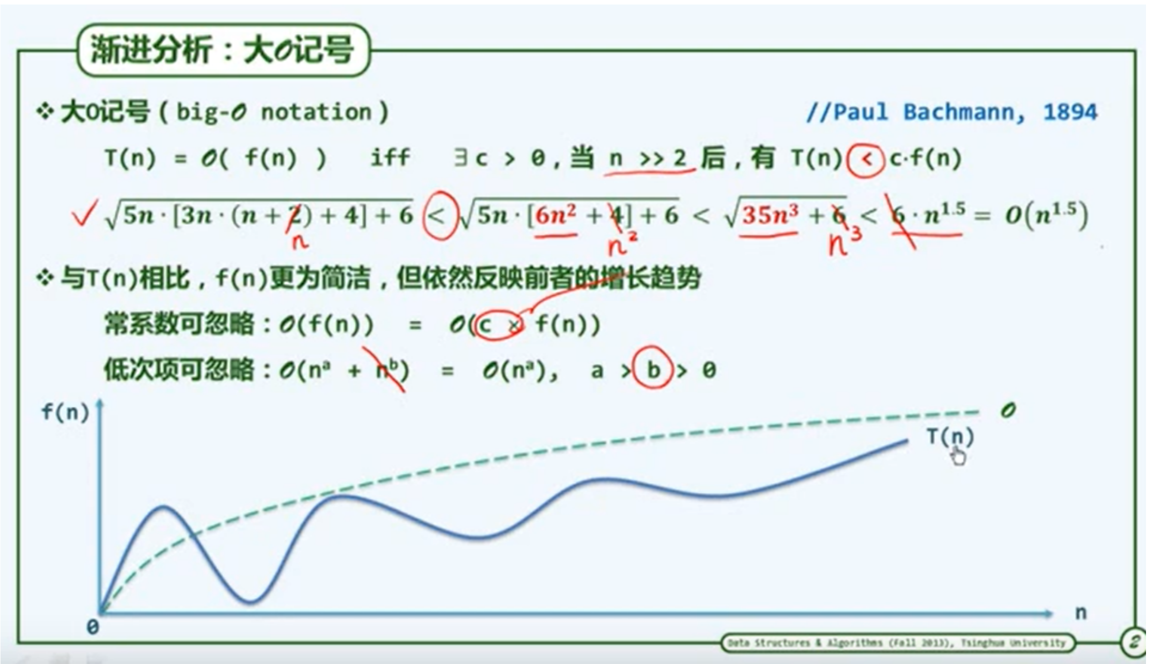
- 所需执行基本操作次数T(n) = ?
- 需占用的储存单元数S(n) = ?

主要看长远地变化趋势!

- 大O记号

T(n) = O(f(n))，存在一个c，当n>>2后，有T(n) < c f(n)

通过不断简化，可以得出反应整体算法的增长趋势：



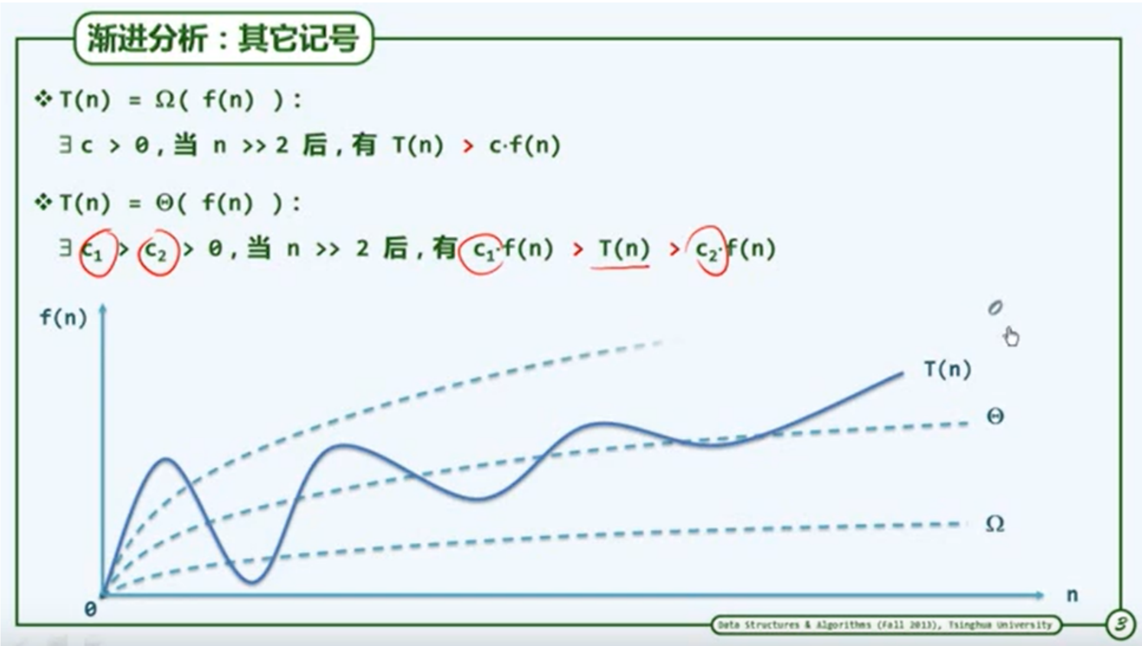
▼ 01-C-2 大Omega/Theta记号

- 大 Omega 记号

T(n) = Omega(f(n))，存在一个c，当n>>2后，有T(n) > c f(n)

- Theta记号（上下界定）

T(n) = Omega(f(n))，存在一个c1>c2>9，当n>>2后，有c1 f(n) > T(n) > c2 f(n)



- 常数复杂度 O(1)

不包含显式的循环，只包含判断语句，则顺序执行一般为常数复杂度

- 对数复杂度（往往不标明具体的底数，不影响复杂度）
  - 常底数无所谓

$$\forall a, b > 0, \log_a n = \log_a b \cdot \log_b n = \Theta(\log_b n)$$

  不同的底数都可以通过常数乘法进行转化

  - 常数次幂无所谓

$$\forall c > 0, \log n^c = c \cdot \log n = \Theta(\log n)$$

- 对数多项式（poly-log func）则只需要高次幂

$$123 \log^{123} n + \log^{10}(n^2 - n + 1) = \Theta(\log^{123} n)$$

- 这类算法非常高效。复杂度无限接近于常数

$$\forall c > 0, \log n = \mathcal{O}(n^c)$$

算法的复杂度无限接近于常数

- 多项式复杂度，一般直接取最高次项

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + c = \mathcal{O}(n^k)$$

  - 线性函数，则幂次c=1
  - 从O(n)到O(n2)则为变成习题主要覆盖的范围

  直接看最高复杂度的计算即可

▼ 01-C-3 复杂度总结

- 指数（exponential）函数复杂度

$$T(n) = a^n$$

他永远构成了多项式函数上界

$$n^{1000} = \mathcal{O}(1.000001^n) = \mathcal{O}(2^n)$$

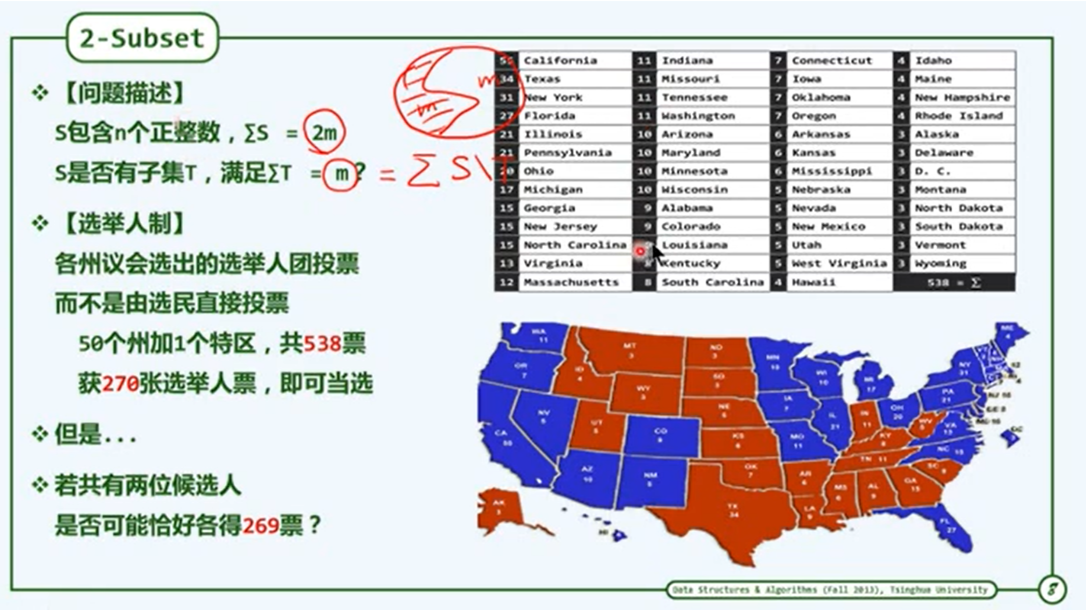
$$1.000001^n = \Omega(n^{1000})$$

计算成本增长极快！

  - 从O(n^c)到O(2^n)是从有效到无效的分水岭

    O(2^n)往往显而易见而O(n^c)极其不易

- 案例分析，能否把和为2m的n个正整数集合S，能否平均分为两部分，和为m



直觉算法：逐一枚举S的子集

$$|2^s| = 2^{|s|} = 2^n$$

这个2-Subse问题是t是NP完备问题！难以求解

- 增长速度

指数函数可能一开始增长速度不快，但是长远看来，算术复杂度增长最快！