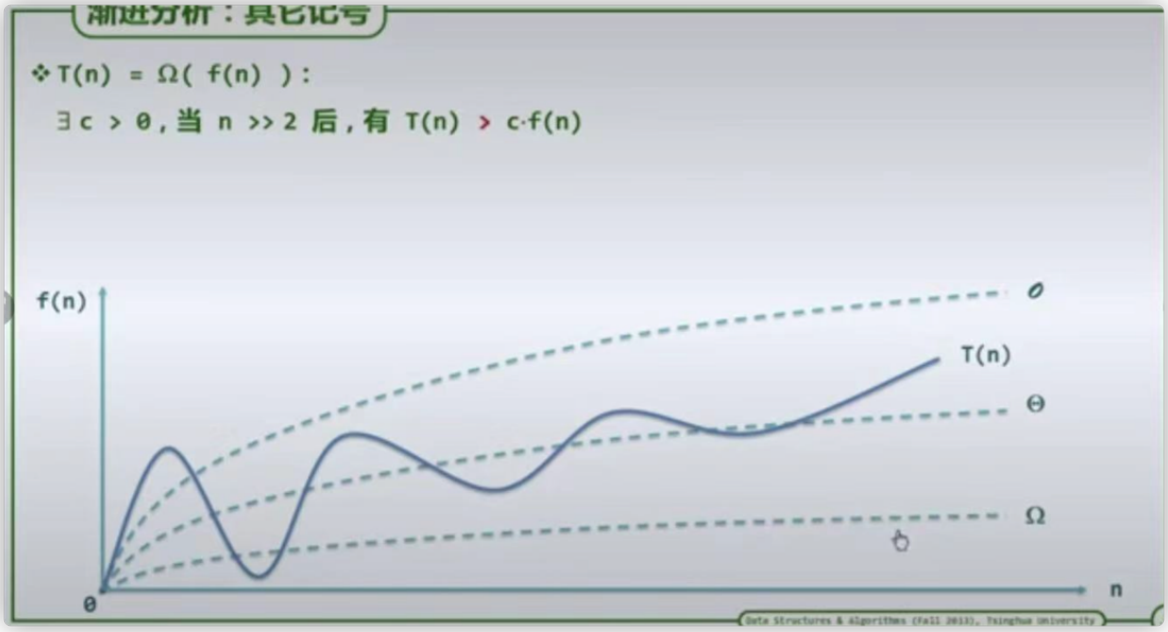
01-C-2 BigΩ , BigΘ & 复杂度介绍

#数据结构邓神

渐进分析其他符号




```
T(n) = Ω( f(n) )
存在 c > 0 当 n >> 2 时，有 T(n) > c * f(n)

BigΩ 构成了原来的复杂度的下界
属于算法最好的情况

T(n) = Θ( f(n) )
存在 c1 > c2 > 0 当 n >> 2 时，有 c1 * f(n) > T(n) > c2 * f(n)

Θ 可以认为是BigO 和 BigΩ 的一个组合
Θ 是一个准确的估计

但是即使是这样，我们用的最多的还是 BigO
```

O(1) 常数复杂度 constant function

记住无论是多大的常数，只要没有变量，就属于常数 即便是 1024^1024次方也算是常数

哪些代码属于常数复杂度:

1. 一定不含循环 ？？？

```
for (int i = 0 ; i < n ; i++);
for (int i = 0 ; i < n ; i = 1 << i);
```

复杂度 Log n 几乎为常数

2. 一定不含分支转向 ？ // 转向不可以到达

```
if (( n + m ) * ( n + m ) < 4 * n * m) goto UNREACHABLE  // 不考虑溢出
```

3.一定不能有（递归调用）？？？

```
 if ( 2 == ( n * n) % 5 ) 01(n)
```

O(Log n) 对数复杂度

对数： O(logn)

lnn | lgn | log_100 n | log_1023 n

1. 常数底数无所谓

任取 a,b > 0 log_a n = log_a b * log_b n = θ(log_b n)

因为无论以什么为底都只有一个常数的区别

2. 常数次幂无所谓

任取 c > 0 , logn^c = c * logn = θ(logn)

3. 对数多项式 （ploy-log function）

123*log^123 n + log^105 (n^2 - n + 1) = θ(log^321 n)

// 把低次项忽略

这类算法非常高效，复杂度无限接近于常数

任取 c > 0 , log n = O(n^c)

O(n^c) 多项式复杂度 (polynomial function)

100n + 200 = o(n)

(100n − 500)(20n^2 − 300n + 2013) = o(n ∗ n^2) = o(n^3)

(2013n^2 − 20)/(1999n − 1) = o(n^2/n) = o(n)

一般的 a_k ∗ n^k + a_k−1 ∗ n^k−1 + ... + a_1 ∗ n + a_0 = o(n^k) , a_k > 0

其中特殊的有 线性 (linear function) : 所有 o(n) 类的函数

编程习题主要覆盖的范围： o(n) ~ o(n^2)

这类算法的效率通常是可以令人满意的，然而这个标准是否是太低了呢?