01-C-03 复杂度总结

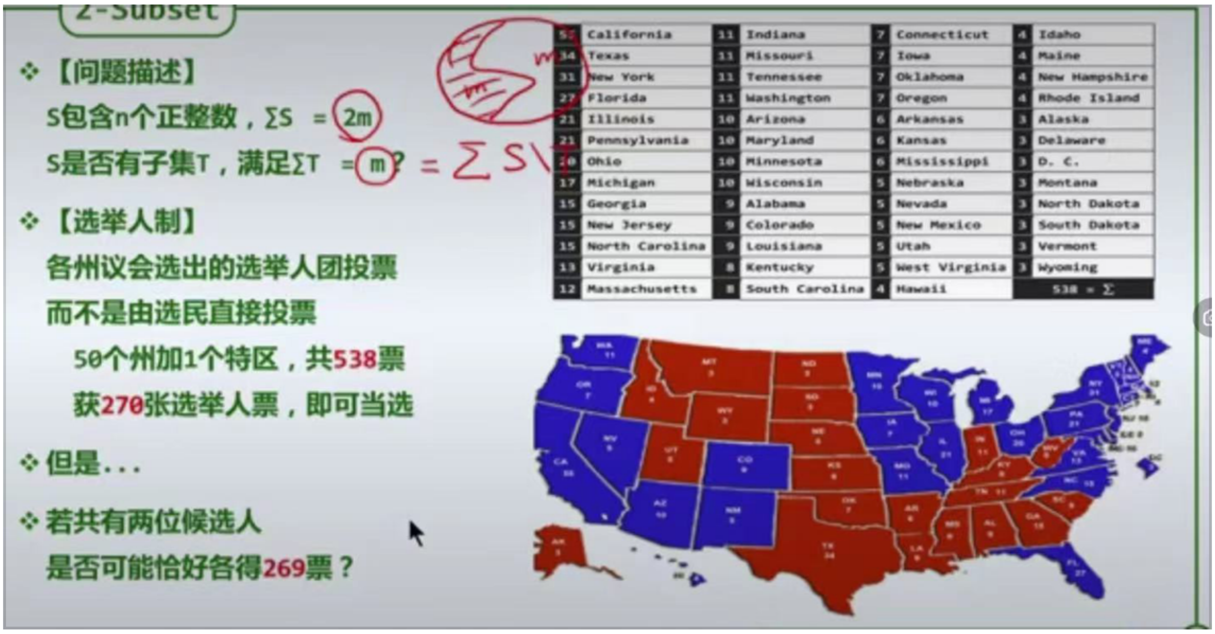
#数据结构邓神

指数复杂度

```
指数(exponential function): T(n) = a^n

任取 c > 1 , n^c = O(2^n)
n^1000 = O(1.0000001^n) = O(2^n)
1.0000001^n = Ω(n^1000)

这类算法的计算成本是增长的及其之快，这一般是不可接受的!

从 O(n^c) 到 O(2^n) 是从有效算法到无效算法到分水岭

注意:  很多问题的指数形式的算法都显而易见
然而设计出多项式形式的算法却极其不易
甚至，有些时候注定是徒劳无功

更糟糕的是，这类问题比我们想象的要多得多
```

Q.1 Subset

```
问题描述:  S包含n个正整数 ∑S = 2m
S 是否有子集 T 满足 ∑T = m ?

就是说S能否分为两部分，使得总和都为m
```



直觉解法

```
枚举S的所有子集，并统计元素的总和是否等于 m
此时 算法的复杂度等于 子集的个数 也就是 2^n 是一个指数级的算法，是不可以接受的


这种算法的正确性是毋庸置疑的!
但是计算成本是非常高的
```

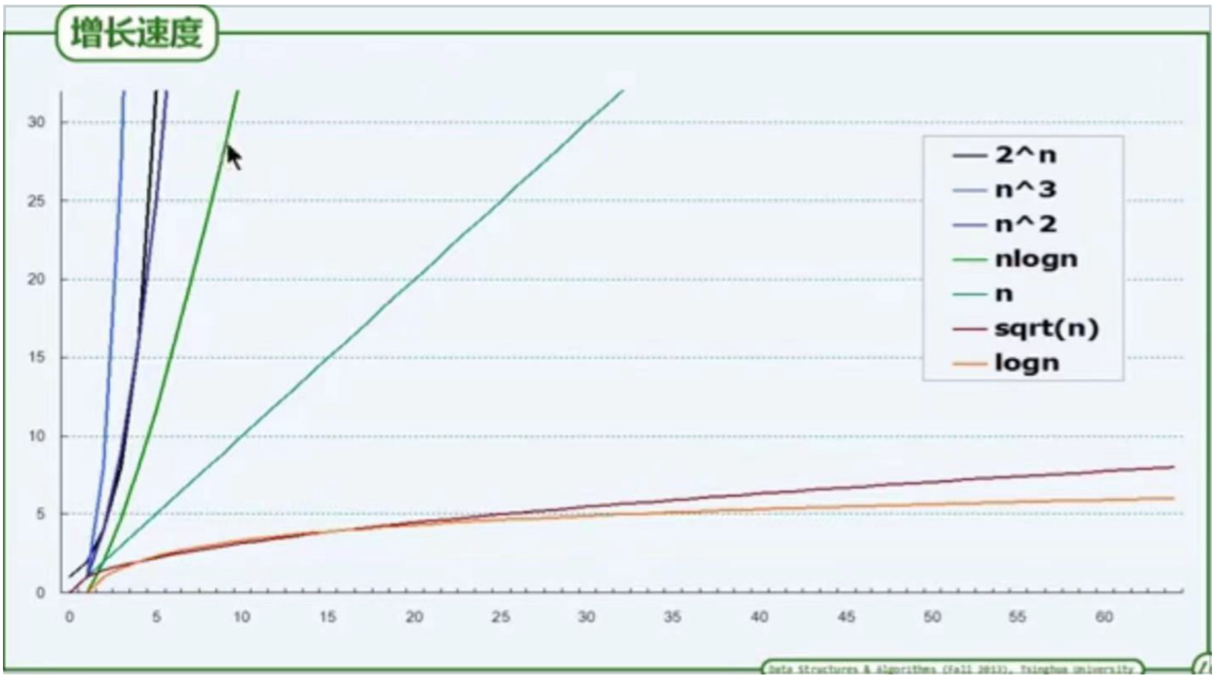
那是否存在优化算法呢?

```
定理 : 2-Subset is NP-complete


意思就是: 就目前的计算模型而言，不存在可在多项式时间内回答此问题的算法
也就是说上述的直觉解发就属于最优解法


很可惜的是这种问题只能是 指数复杂度
```

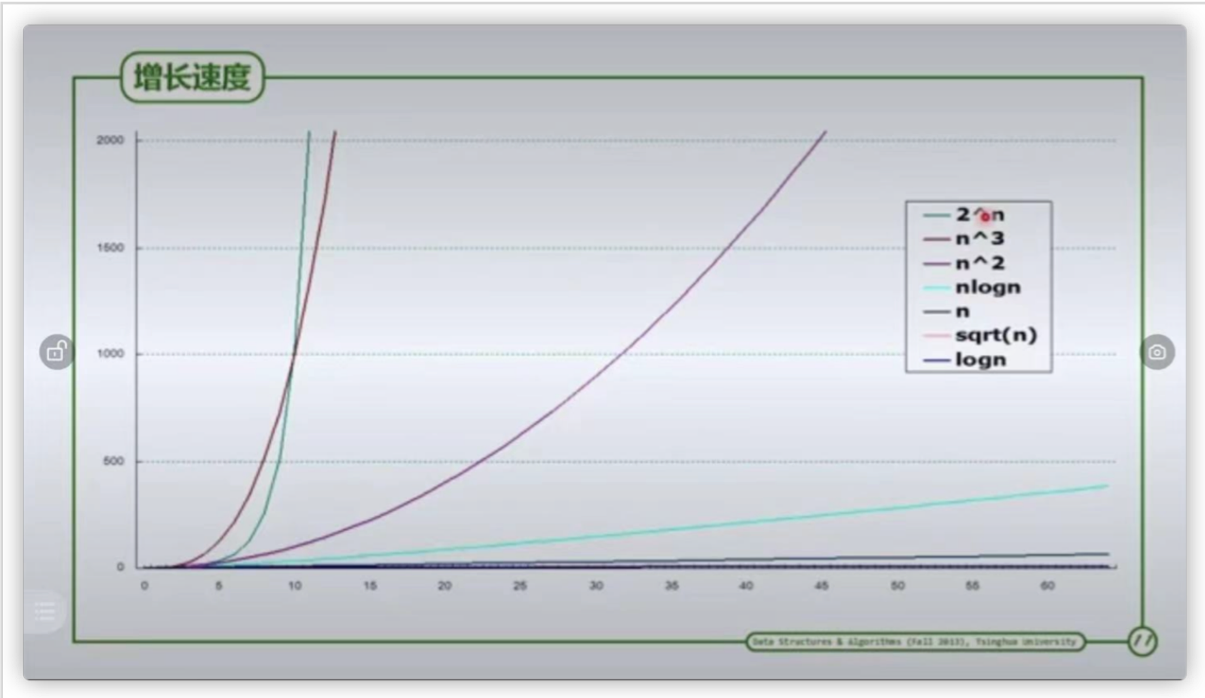
> **NP完全或NP完备**（**NP-Complete**，缩写为 **NP-C** 或 **NPC**），是计算复杂度理论中， 决定性问题 的等级之一。NPC 问题，是 NP （非决定性 多项式时间 ）中最难的 决定性问题 。因此NP完备问题应该是最不可能被化简为（多项式 时间 可决定）的决定性问题的集合。

增长速度

在小范围内 似乎 多项式复杂度与 指数复杂度没有大的区别


增长速度
30
25
20
15
10
5
0
0
5
10
15
20
25
30
35
40
45
50
55
60
2^n
n^3
n^2
nlogn
n
sqrt(n)
logn

要放眼长远


增长速度
2000
1500
1000
500
0
0
5
10
15
20
25
30
35
40
45
50
55
60
2^n
n^3
n^2
nlogn
n
sqrt(n)
logn