

Mathematics is more in need  
of good notations than  
of new theorems.

- Alan Turing

好读书不求甚解

每有会意，便欣然忘食

——陶渊明

## 1. 绪论

(c) 大O记号

邓俊辉

deng@tsinghua.edu.cn

## 渐进分析：大O记号

❖ 回到原先的问题：随着问题规模的增长，计算成本如何增长？

注意：这里更关心**足够大**的问题，注重考察成本的增长趋势

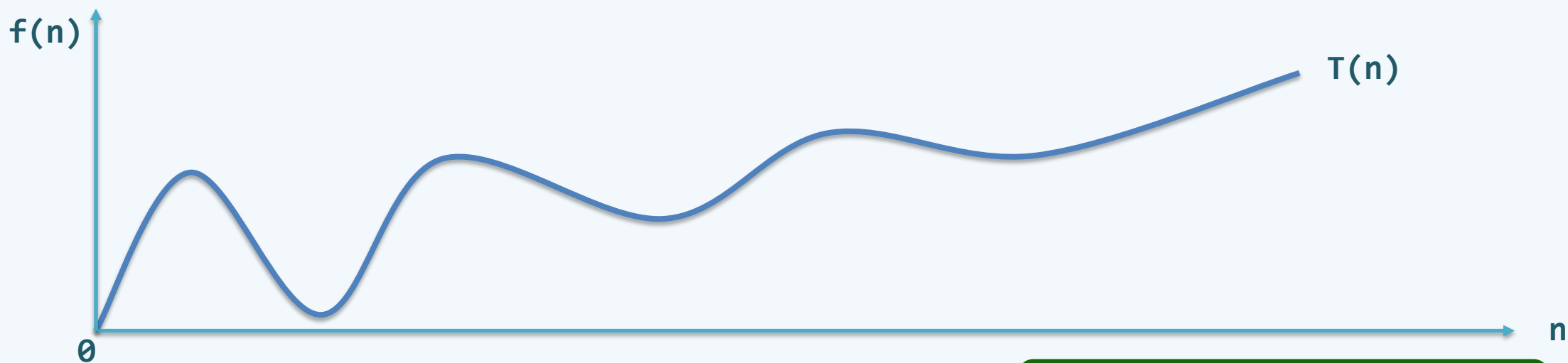
❖ 渐进分析：在问题规模足够大后，计算成本如何增长？

Asymptotic analysis：当 $n \gg 2$ 后，对于规模为 $n$ 输入，算法

需执行的基本操作次数： $T(n) = ?$

需占用的存储单元数： $S(n) = ?$

//通常可不考虑，为什么？



## 渐进分析：大O记号

❖ 大O记号 (big-O notation)

//Paul Bachmann, 1894

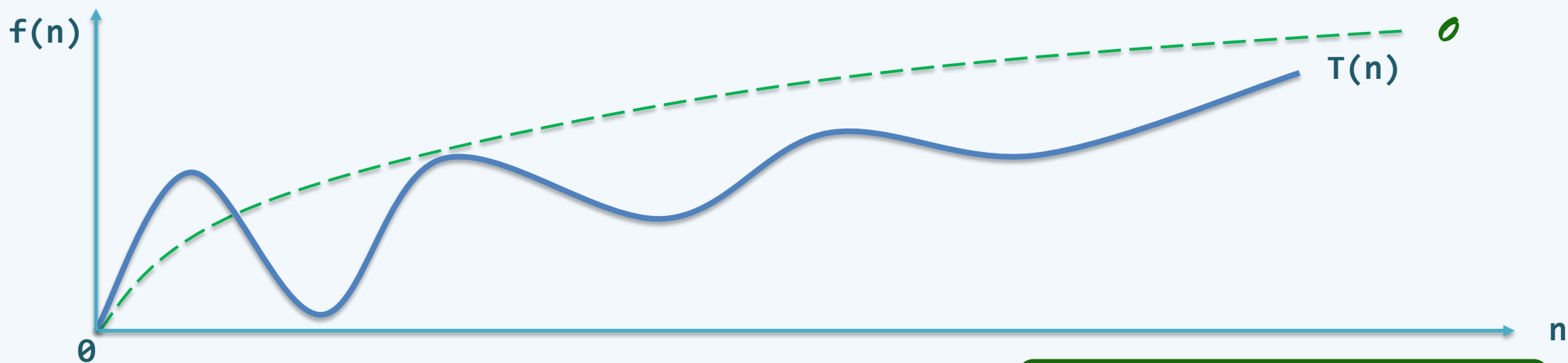
$T(n) = O(f(n))$  iff  $\exists c > 0$ , 当  $n \gg 2$  后, 有  $T(n) < c \cdot f(n)$

$$\sqrt{5n \cdot [3n \cdot (n+2) + 4] + 6} < \sqrt{5n \cdot [6n^2 + 4] + 6} < \sqrt{35n^3 + 6} < 6 \cdot n^{1.5} = O(n^{1.5})$$

❖ 与 $T(n)$ 相比,  $f(n)$ 更为简洁, 但依然反映前者的增长趋势

常系数可忽略:  $O(f(n)) = O(c \times f(n))$

低次项可忽略:  $O(n^a + n^b) = O(n^a)$ ,  $a > b > 0$



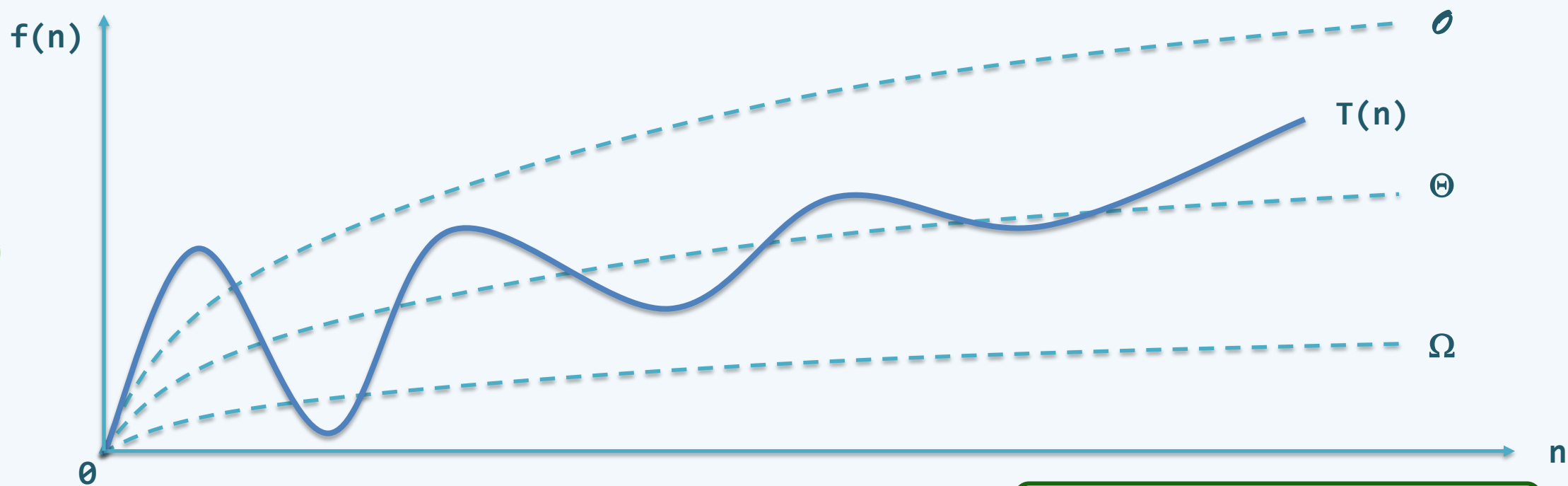
## 渐进分析：其它记号

❖  $T(n) = \Omega(f(n))$  :

$\exists c > 0$ , 当  $n \gg 2$  后, 有  $T(n) > c \cdot f(n)$

❖  $T(n) = \Theta(f(n))$  :

$\exists c_1 > c_2 > 0$ , 当  $n \gg 2$  后, 有  $c_1 \cdot f(n) > T(n) > c_2 \cdot f(n)$



$O(1)$

❖ 常数 ( constant function )

$2 = 2013 = 2013 \times 2013 = O(1)$  , 甚至  $2013^{2013} = O(1)$  //含RAM各基本操作

❖ 这类算法的效率最高

//总不能奢望不劳而获吧

❖ 什么样的代码段对应于常数执行时间 ?

//应具体分析

一定不含循环 ?

```
for (i = 0; i < n; i += n/2013 + 1);
```

```
for (i = 1; i < n; i = 1 << i);
```

//log\*n , 几乎常数

一定不含分支转向 ?

```
if ((n + m) * (n + m) < 4 * n * m) goto UNREACHABLE;
```

//不考虑溢出

一定不能有 ( 递归 ) 调用 ?

```
if (2 == (n * n) % 5) O1(n);
```

...

## $O(\log^c n)$

### ❖ 对数 $O(\log n)$

//为何不注明底数？

$$\ln n \mid \lg n \mid \log_{100} n \mid \log_{2013} n$$

### ❖ 常底数无所谓

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

### ❖ 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 function)

$$123 \cdot \log^{321} n + \log^{105}(n^2 - n + 1) = \Theta(\log^{321} n)$$

### ❖ 这类算法非常有效，复杂度无限接近于常数

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

❖ 多项式 ( polynomial function )

$$100n + 200 = O(n)$$

$$(100n - 500)(20n^2 - 300n + 2013) = O(n \times n^2) = O(n^3)$$

$$(2013n^2 - 20)/(1999n - 1) = O(n^2/n) = O(n)$$

一般地：  $a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 = O(n^k)$ ,  $a_k > 0$

❖ 线性 ( linear function ) : 所有  $O(n)$  类函数

❖ 从  $O(n)$  到  $O(n^2)$  : 编程习题主要覆盖的范围

❖ 幂：  $[(n^{2013} - 24n^{2009})^{1/3} + 512n^{567} - 1978n^{123}]^{1/11} = O(n^{61})$

❖ 这类算法的效率通常认为已可令人满意，然而...

这个标准是否太低了？

//P难度！

## $O(2^n)$

❖ 指数 (exponential function) :  $T(n) = a^n$

❖  $\forall c > 1, n^c = O(2^n)$  //  $e^n = 1 + n + n^2/2! + n^3/3! + n^4/4! + \dots$

$$n^{1000} = O(1.0000001^n) = O(2^n)$$

$$1.0000001^n = \Omega(n^{1000})$$

❖ 这类算法的计算成本增长极快，通常被认为不可忍受

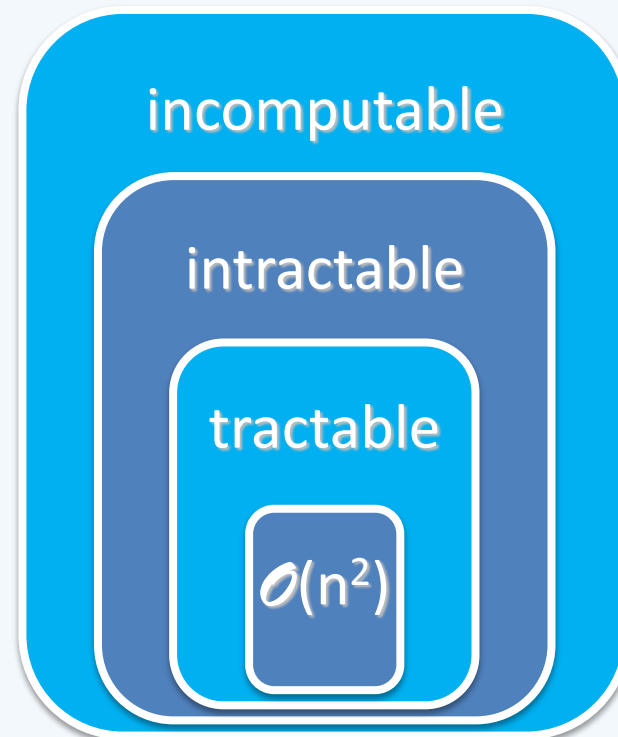
❖ 从  $O(n^c)$  到  $O(2^n)$ ，是从有效算法到无效算法的分水岭

❖ 很多问题的  $O(2^n)$  算法往往显而易见

然而，设计出  $O(n^c)$  算法却极其不易

甚至，有时注定地只能是徒劳无功

❖ 更糟糕的是，这类问题要远比我们想象的多得多...





## 2-Subset

### ❖ 【问题描述】

$S$  包含  $n$  个正整数,  $\sum S = 2m$

$S$  是否有子集  $T$ , 满足  $\sum T = m$ ?

### ❖ 【选举人制】

各州议会选出的选举人团投票

而不是由选民直接投票

50个州加1个特区, 共**538**票

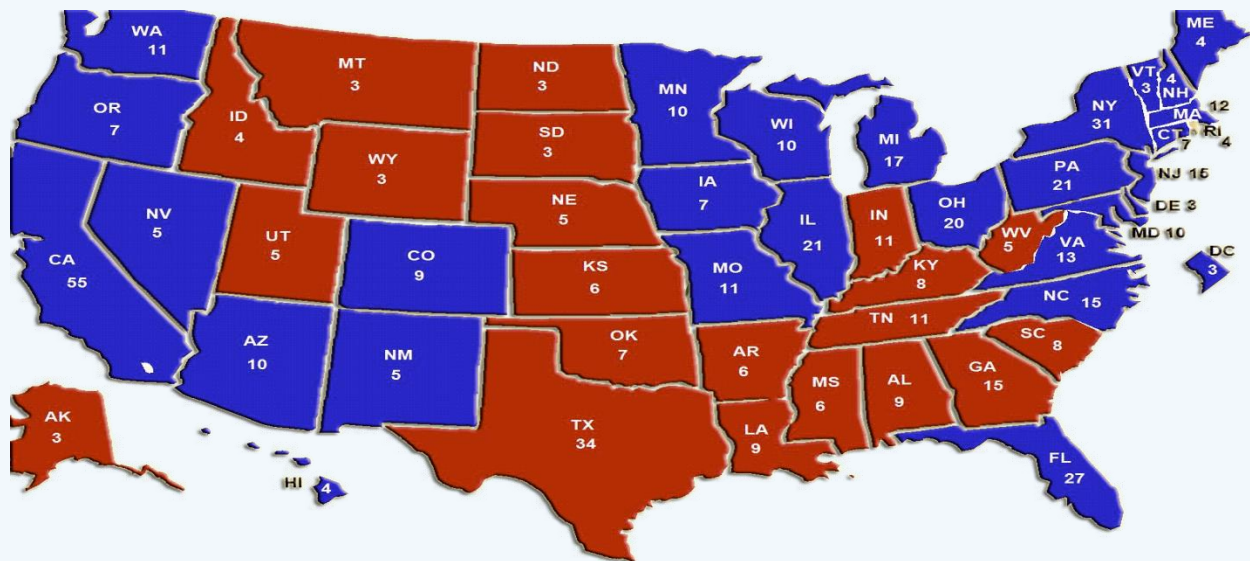
获**270**张选举人票, 即可当选

### ❖ 但是...

### ❖ 若共有两位候选人

是否可能恰好各得**269**票?

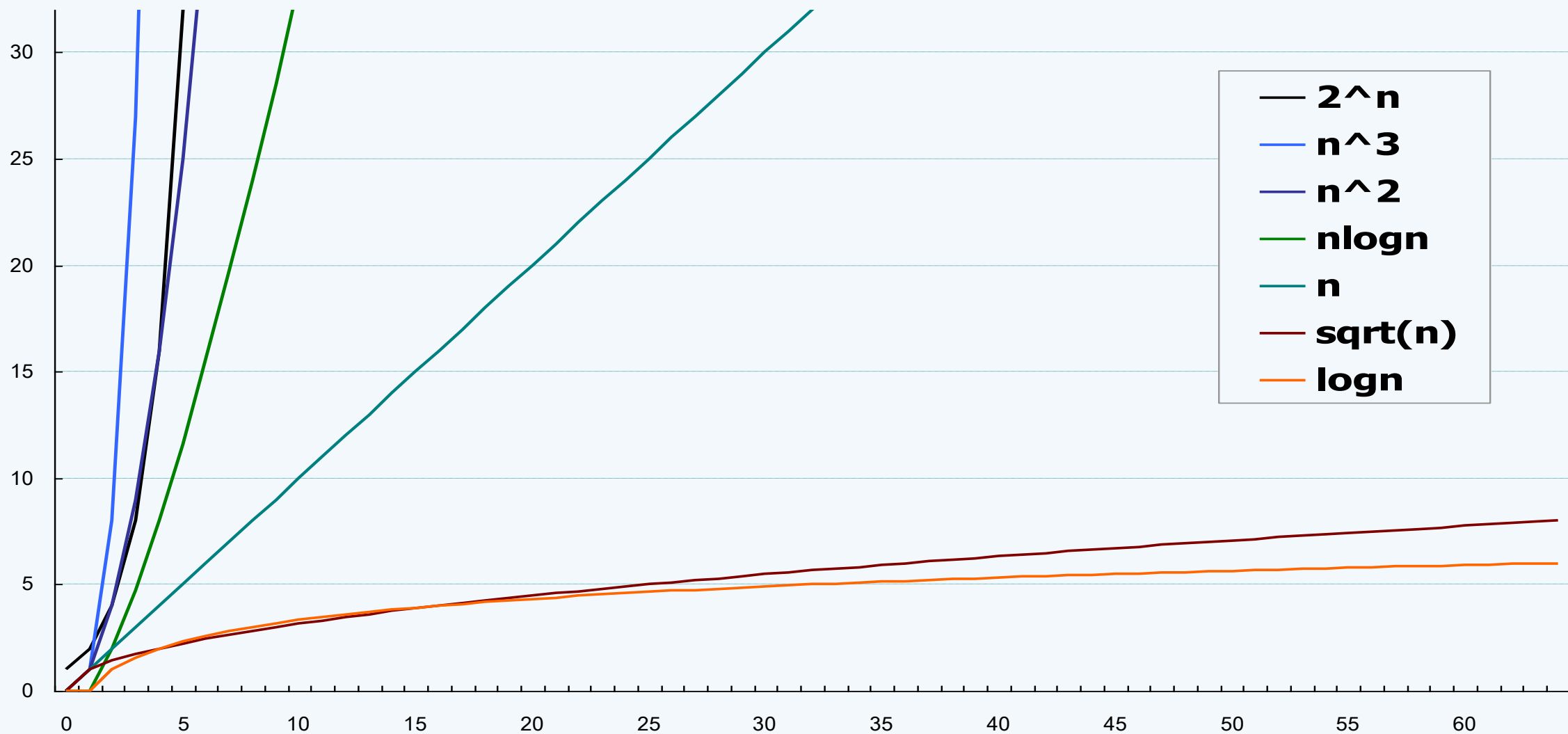
55	California	11	Indiana	7	Connecticut	4	Idaho
34	Texas	11	Missouri	7	Iowa	4	Maine
31	New York	11	Tennessee	7	Oklahoma	4	New Hampshire
27	Florida	11	Washington	7	Oregon	4	Rhode Island
21	Illinois	10	Arizona	6	Arkansas	3	Alaska
21	Pennsylvania	10	Maryland	6	Kansas	3	Delaware
20	Ohio	10	Minnesota	6	Mississippi	3	D. C.
17	Michigan	10	Wisconsin	5	Nebraska	3	Montana
15	Georgia	9	Alabama	5	Nevada	3	North Dakota
15	New Jersey	9	Colorado	5	New Mexico	3	South Dakota
15	North Carolina	9	Louisiana	5	Utah	3	Vermont
13	Virginia	8	Kentucky	5	West Virginia	3	Wyoming
12	Massachusetts	8	South Carolina	4	Hawaii	538 = $\Sigma$	



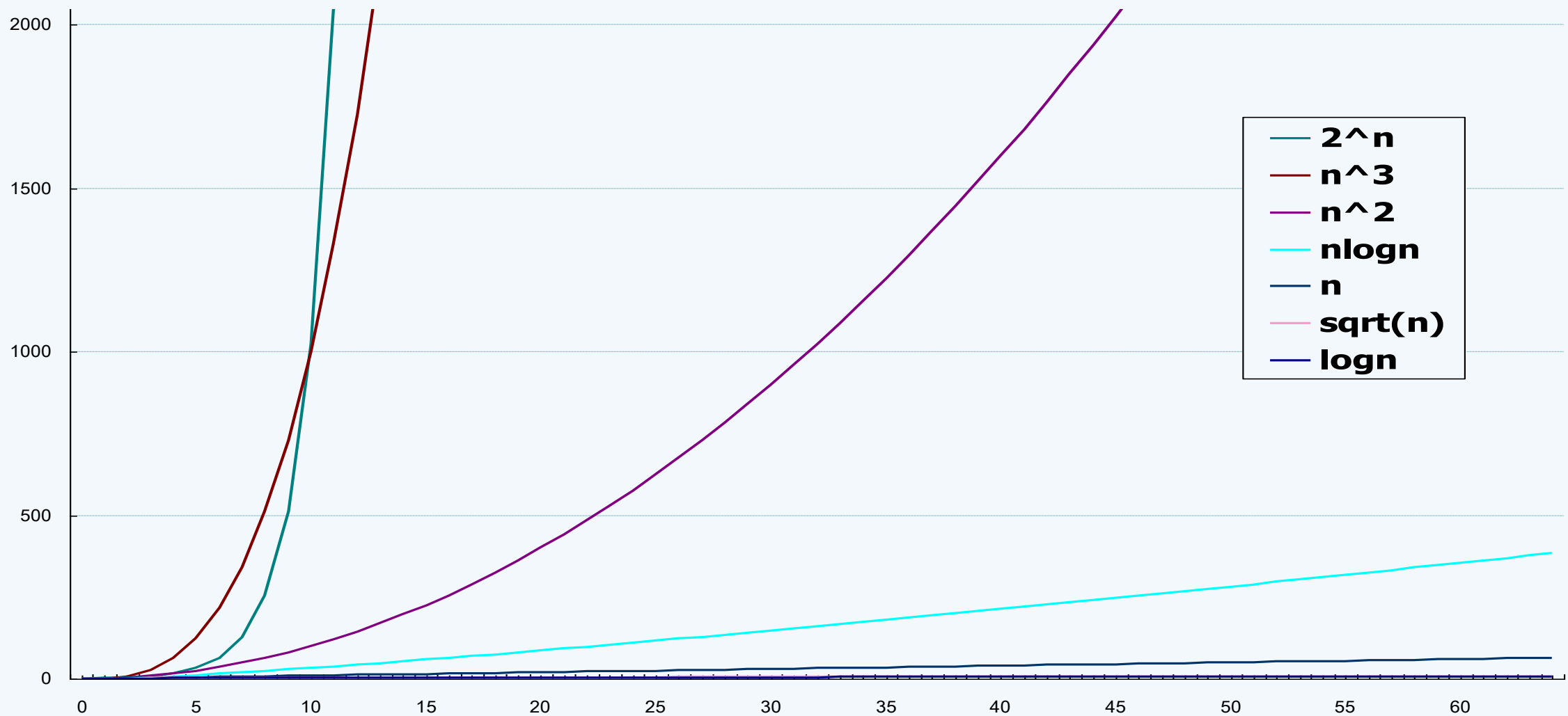
## 2-Subset

- ❖ 直觉算法：逐一枚举 $S$ 的每一子集，并统计其中元素的总和
- ❖ 定理： $|2^S| = 2^{|S|} = 2^n$
- ❖ 亦即：直觉算法需要迭代 $2^n$ 轮，并（在最坏情况下）至少需要花费这么多的时间  
— 不甚理想！
- ❖ 还是直觉：应该有更好的办法吧？
- ❖ 定理：2-Subset is NP-complete  
— 什么意思？
- ❖ 意即：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  
— 就此意义而言，上述的直觉算法已属最优

## 增长速度



## 增长速度



## 复杂度层次

$O(1)$	常数复杂度	再好不过，但难得如此幸运	对数据结构的基本操作
$O(\log^*n)$		在这个宇宙中，几乎就是常数	
$O(\log n)$	对数复杂度	与常数无限接近，且不难遇到	有序向量的二分查找 堆、词典的查询、插入与删除
$O(n)$	线性复杂度	努力目标，经常遇到	树、图的遍历
$O(n \log^*n)$		几乎几乎几乎接近线性	某些MST算法
$O(n \log \log n)$		几乎接近线性	某些三角剖分算法
$O(n \log n)$		最常出现，但不见得最优	排序、EU、Huffman编码
$O(n^2)$	平方复杂度	所有输入对象两两组合	Dijkstra算法
$O(n^3)$	立方复杂度	不常见	矩阵乘法
$O(n^c)$ ，c常数	多项式复杂度	P问题 = 存在多项式算法的问题	
$O(2^n)$	指数复杂度	很多问题的平凡算法，再尽可能优化	
...		绝大多数问题，并不存在算法	

## 课后

❖ 证明、证否或计算： Fibonacci数  $\text{fib}(n) = O(2^n)$

$$12n + 5 = O(n \log n)$$

$$\log^2(n^{1024} - 2 \cdot n^6 + 101) = O(?)$$

$$\log^d n = O(n^c), \forall c > 0, d > 1$$

$$\log^{1.001} n = O(\log(n^{1001}))$$

$$(n^2 + 1) / (2n + 3) = O(n)$$

$$n^{2013} = O(n!)$$

$$n! = O(n^{2013})$$

$$2^n = O(n!)$$

❖ k-Subset：任给整数集S，判定S可否划分为k个不交子集，其和均为 $(\sum S)/k$

证明或证否：(k+1)-Subset的难度不低于k-Subset

❖ Google: small-o notation