Algorithm and Object-Oriented Programming for Modeling
Part 7: More about Algorithms and Designs

MSDM 5051, Yi Wang (王一), HKUST

Computational and Modeling Tools

What’s the most powerful tool?

Photographer:

牛頭、狗頭，不如人頭

Programmer:

CPU， GPU，不如 U

Plan:

- Introduction
- The Big O, P, NP, NP hard and complete
- Programming Paradigms
- Software Engineering

$$\begin{tikzcd} \text{Input} \arrow[r] & \boxed{\text{algorithm}} \arrow[r] & \text{output} \end{tikzcd}$$

$$\text{Input} \rightarrow \boxed{\text{algorithm}} \rightarrow \text{output}$$

Among the earliest algorithms:

Euclidean algorithm (300 BC) for Greatest common divisor

```
def GCD(a, b):
    return a if b == 0 else GCD(b, a % b)

print(GCD(1071, 462))
```

See also: 《孫子算經》、《九章算術》

$$\text{Input} \rightarrow \boxed{\text{algorithm}} \rightarrow \text{output}$$

Algorithm in the industrial revolution: Jacquard loom

$\text{Input} \rightarrow \boxed{\text{algorithm}} \rightarrow \text{output}$

Algorithm in a modern computer

$$\left\{\begin{array}{l}\text{From machine code to high level language} \\ \text{From high level language to solving actual problem (♡)}\end{array}\right.$$

Input → $\boxed{\text{algorithm}}$ → output

Correct:

買一打鷄蛋，如果有西瓜，買一個

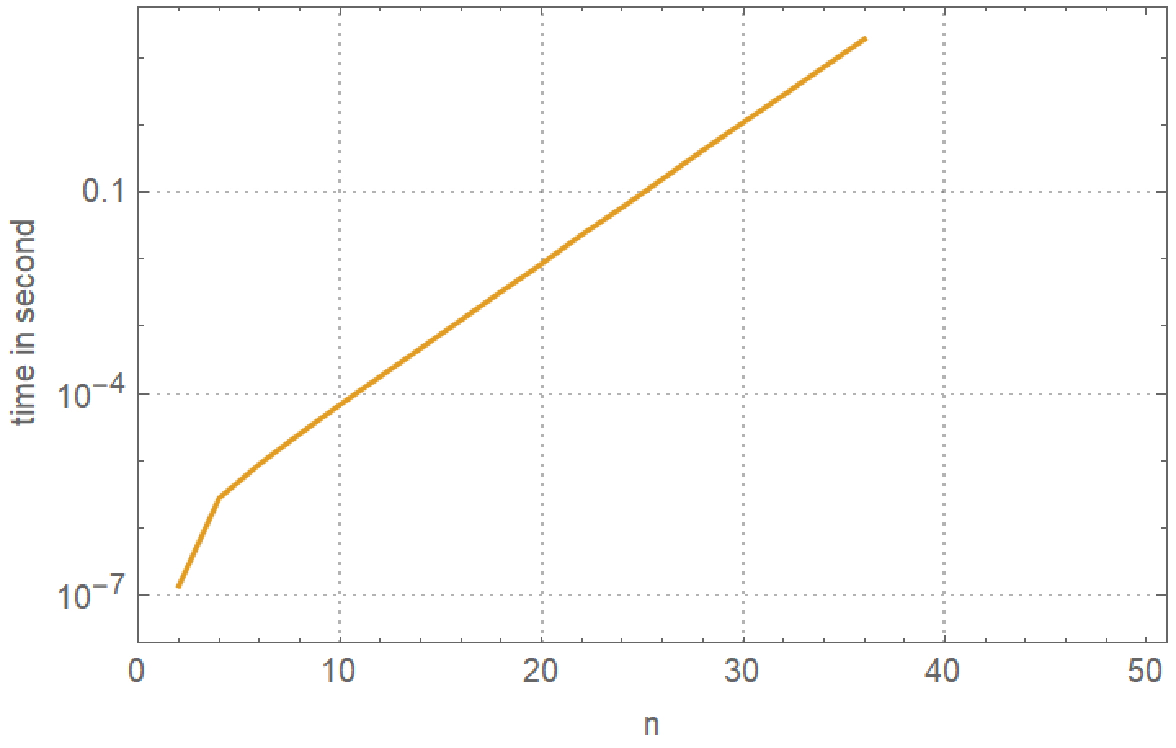
eggs_bought = 12 if no watermelon else 1

Efficient:

Difference between algorithm efficiency could be n times, n^2 times, … e^n times, n^n times, infinity times. Extremely important for big data.

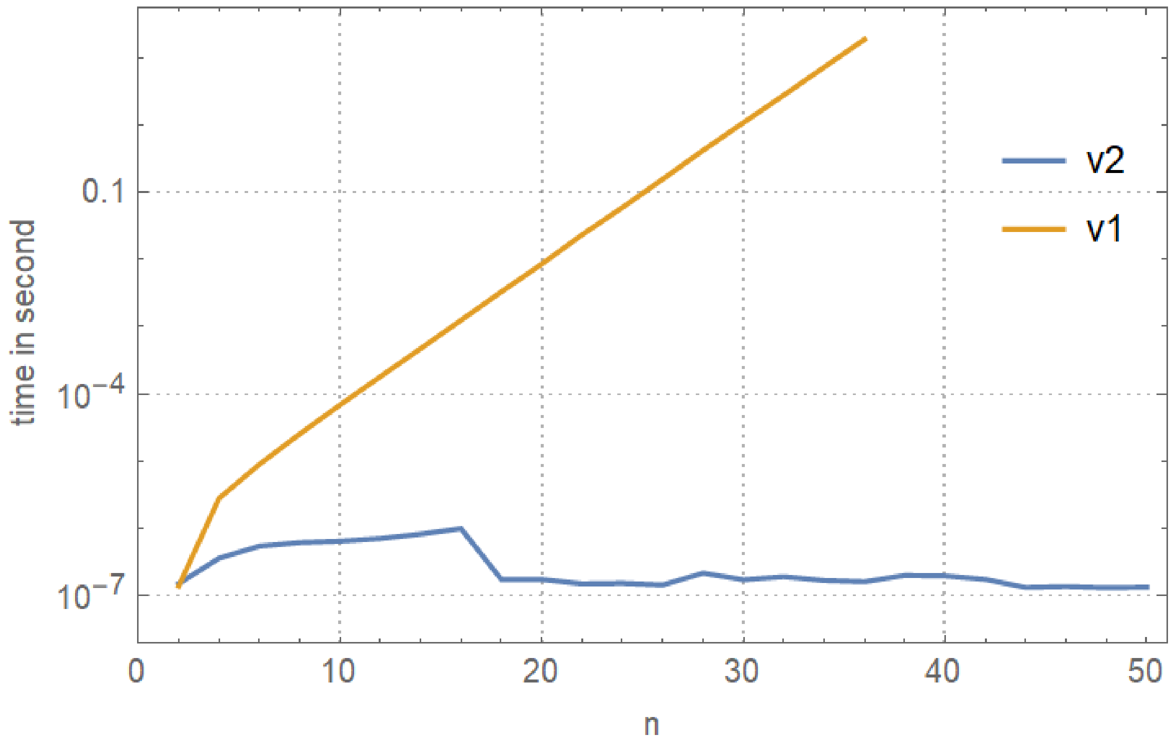
For example: Fibonacci sequence

```
fib[1] = fib[2] = 1;
fib[n_] := fib[n - 1] + fib[n - 2]
```



For example: Fibonacci sequence

```
fib[1] = fib[2] = 1;
fib[n_] := fib[n - 1] + fib[n - 2]
```



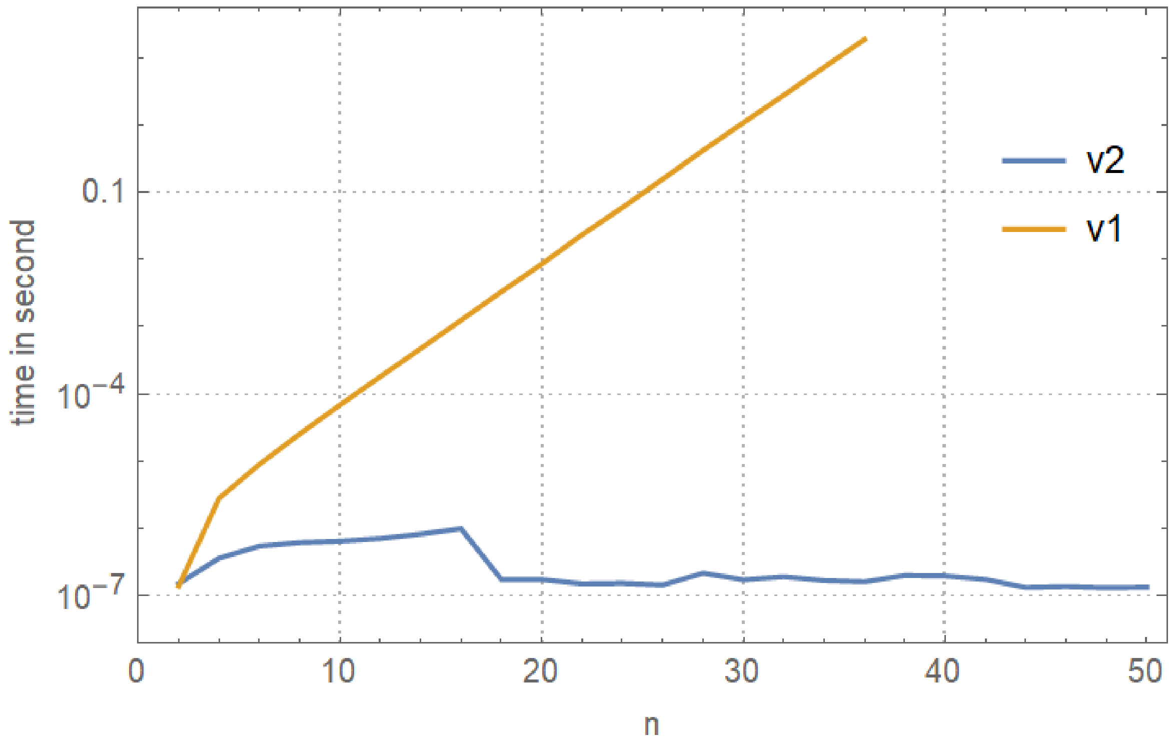
What did I do to make the change happen?

Version 1

```
fib[1] = fib[2] = 1;
fib[n_] := fib[n - 1] + fib[n - 2]
```

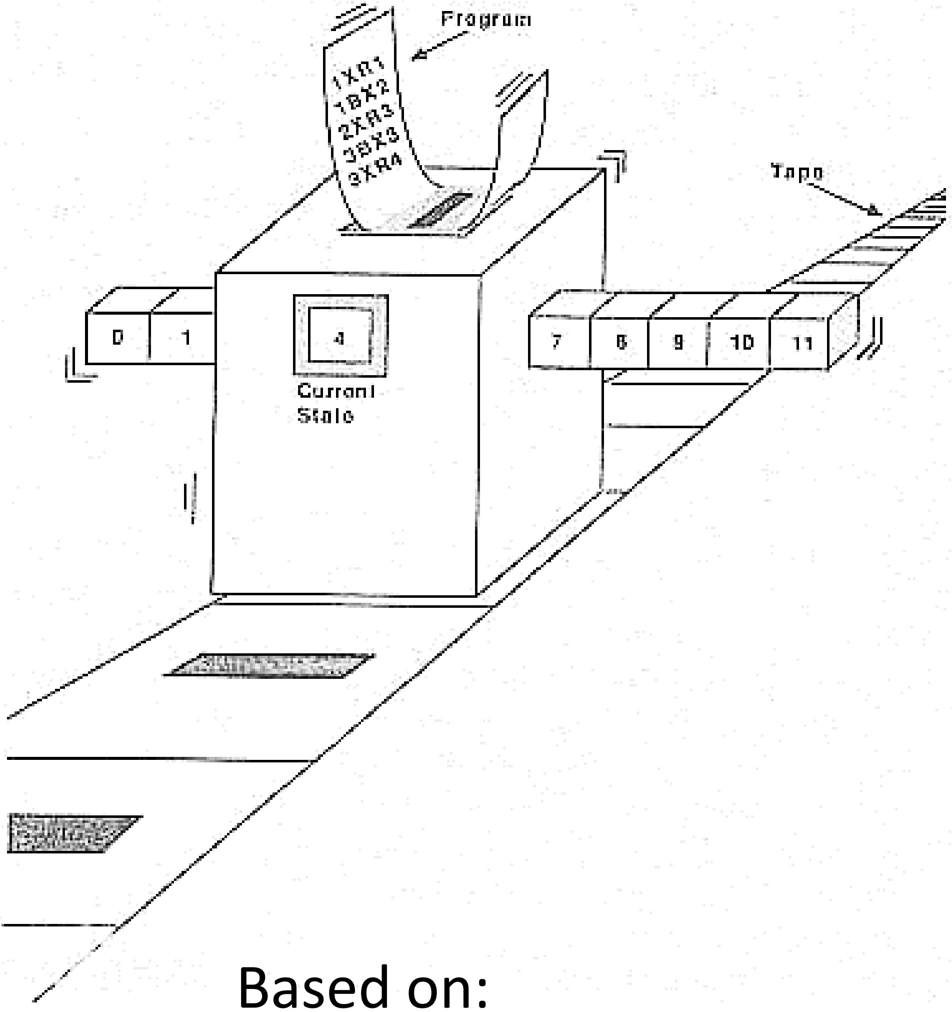
Version 2

```
fib[1] = fib[2] = 1;
fib[n_] := fib[n] = fib[n - 1] + fib[n - 2]
```



What did I do to make the change happen?

How to measure the efficiency of an algorithm?



Turing Machine

Has:

(1) Finite number of internal states
(2) Finite table of rules (program)
(3) Infinite tape in both directions
(4) A “head” to read & write on tape

Based on:

(1) The current symbol
(2) Internal state of TM
(3) A table of rules

Decide to do one of:

(1) Overwrite a new symbol
(2) Move tape for/backward 1 grids
(3) Switch to a new state
(4) Halt

Recall the Jacquard loom machine

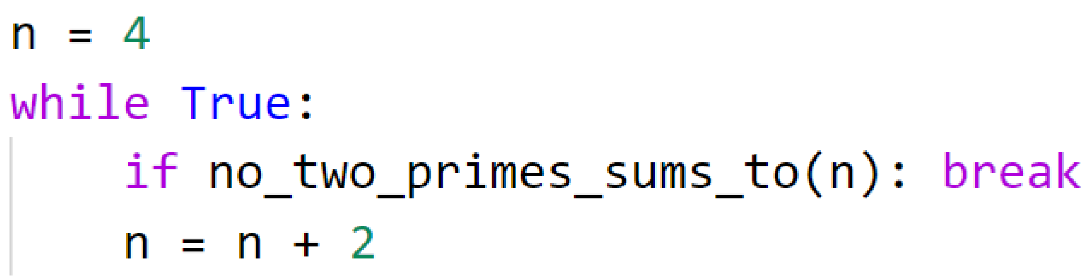
Halting problem: Is there such a program, to decide (in finite time) whether any program halts in finite time or not?

Fantastic if such program exist! Let's call it HP:

For example, using HP to solve the Goldbach's conjecture:

Every even number n>4 can be written as a sum of two primes.

Solution: Use HP to decide whether the below program halts:



```
n = 4
while True:
    if no_two_primes_sums_to(n): break
    n = n + 2
```

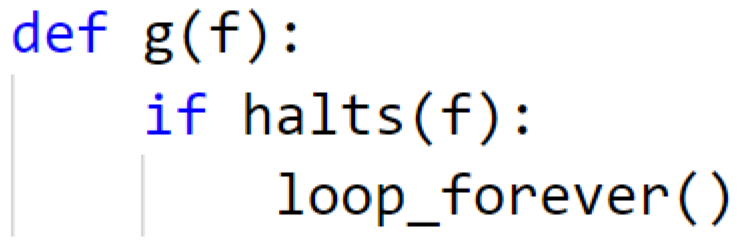
Halt: Goldbach's conjecture is wrong. Does not halt: proved!

A generic road to infinity!

Halting problem: Is there such a program, to decide (in finite time) whether any program halts in finite time or not?

Unfortunately: such HP does not exist. Rough proof:

If HP exists, then one can use HP to write a function:



```
def g(f):
    if halts(f):
        loop_forever()
```

What’s the output of g(g)?

Halting problem: Is there such a program, to decide (in finite time) whether any program halts in finite time or not?

Unfortunately: such HP does not exist. Rough proof:

If HP exists, then one can use HP to write a function:

```
def g(f):
    if halts(f):
        loop_forever()
```

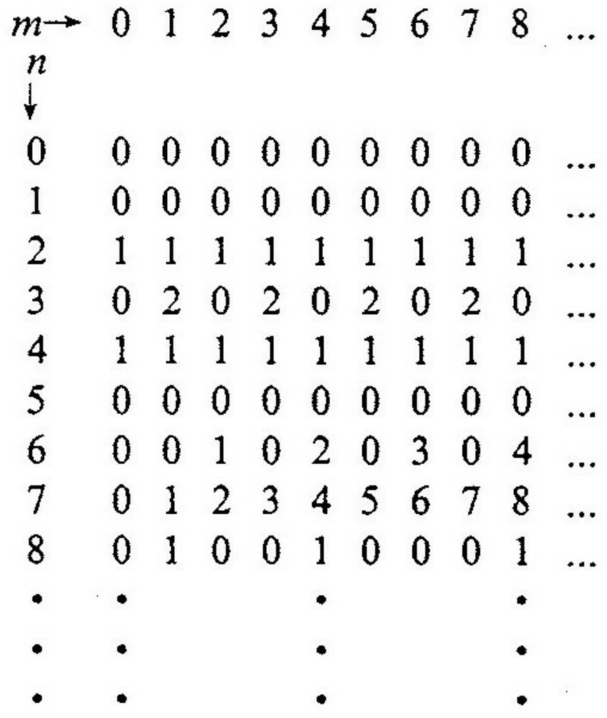
What's the output of g(g)?

If g halts, then g runs forever;

If g runs forever, then g halts.

有的人活着，他已经死了
有的人死了，他还活着

Alternative proof of halting problem: Diagonal method



m→ n↓	0	1	2	3	4	5	6	7	8	...
0	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	...
2	1	1	1	1	1	1	1	1	1	...
3	0	2	0	2	0	2	0	2	0	...
4	1	1	1	1	1	1	1	1	1	...
5	0	0	0	0	0	0	0	0	0	...
6	0	0	1	0	2	0	3	0	4	...
7	0	1	2	3	4	5	6	7	8	...
8	0	1	0	0	1	0	0	0	1	...
•	•				•				•	
•	•				•				•	
•	•				•				•	

Assuming HP exists.

The n-th turing machine

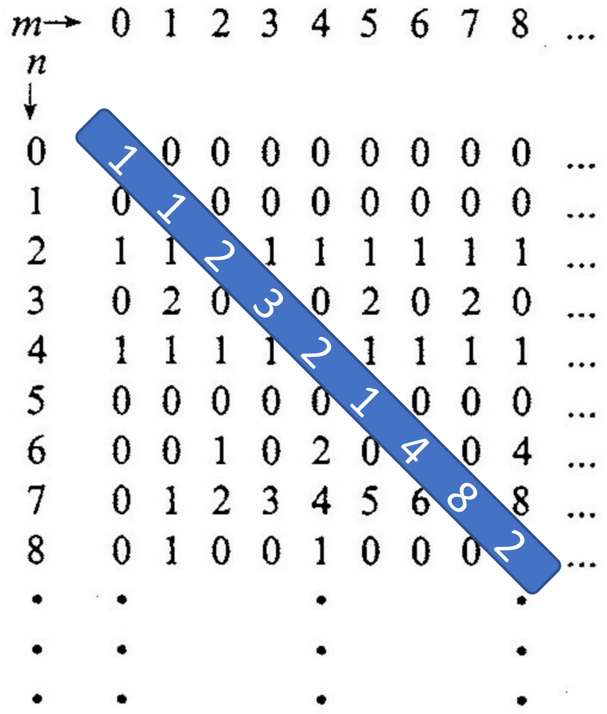
+ The m-th input data

= Definite output

Definite output:

- Either calculation finished in finite time,
- Or 0 means halt.

Alternative proof of halting problem: Diagonal method



$m\rightarrow$ / $n\downarrow$	0	1	2	3	4	5	6	7	8	…
0	1	0	0	0	0	0	0	0	0	…
1	0	1	0	0	0	0	0	0	0	…
2	1	1	2	1	1	1	1	1	1	…
3	0	2	0	3	0	2	0	2	0	…
4	1	1	1	1	2	1	1	1	1	…
5	0	0	0	0	0	1	0	0	0	…
6	0	0	1	0	2	0	4	0	4	…
7	0	1	2	3	4	5	6	8	8	…
8	0	1	0	0	1	0	0	0	2	…
⋮	⋮				⋮				⋮	

Assuming HP exists.

The n-th turing machine

+ The m-th input data

= Definite output

Definite output:

- Either calculation finished in finite time,
- Or 0 means halt.

Now, add one to each diagonal number

(1) This is the output of a Turing machine – we have described the algorithm

(2) This is not an output of any Turing machine, since

1st dig differs from 1st Turing machine, 2nd dig differ from 2nd Turing machine, etc

Contradiction. Thus halting cannot be determined for all Turing machines in finite time.

Infinite time is too mathematical;

Let’s now talk about finite things.

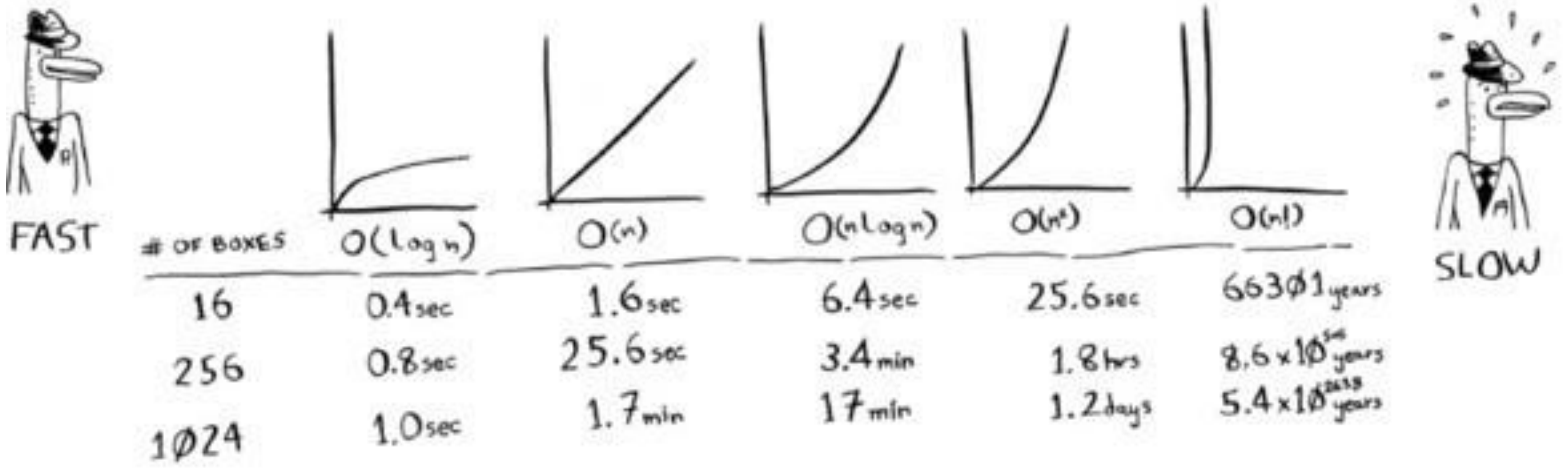
The Big O, P, NP, NP hard and complete

Fly
Emirates

What does a big O mean?

Based on the computation model of the Turing machine:

Problem with input size n can be solved in O(f(n)) steps.

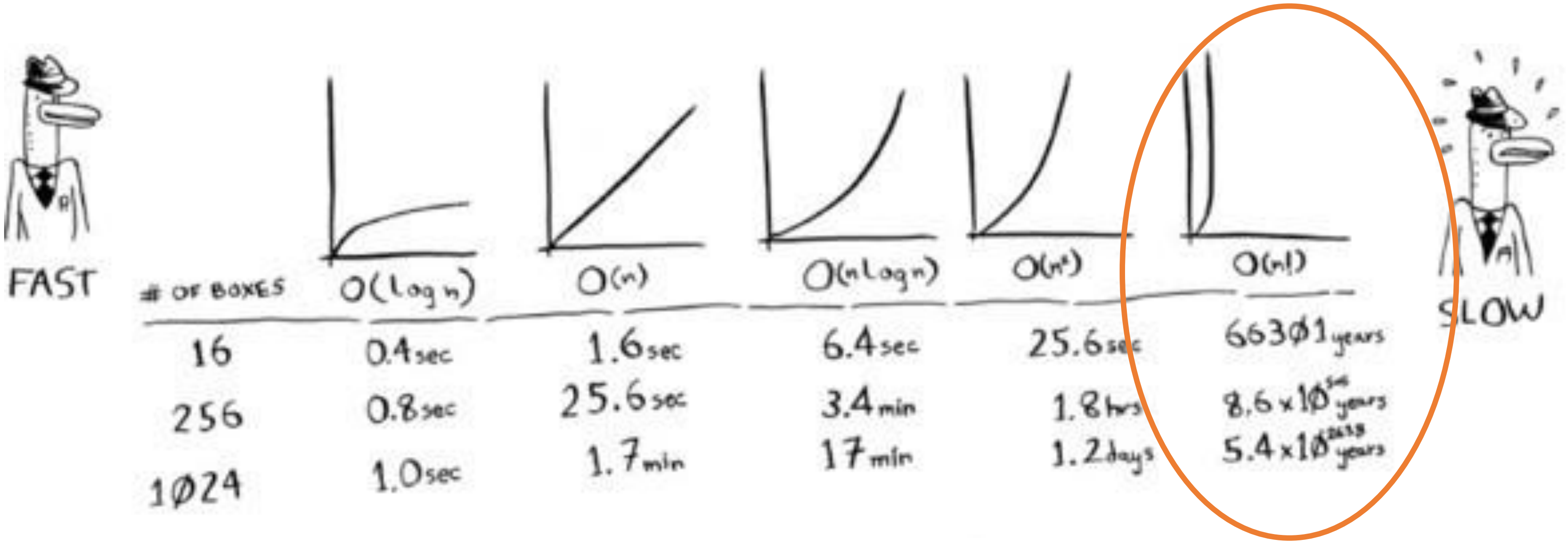


# OF BOXES	O(Log n)	O(n)	O(n Log n)	$O(n^2)$	O(n!)
16	0.4sec	1.6sec	6.4sec	25.6sec	663Ø1years
256	0.8sec	25.6sec	3.4min	1.8hrs	$8.6 \times 1Ø^{54}$ years
1Ø24	1.0sec	1.7min	17min	1.2days	$5.4 \times 1Ø^{2638}$ years

What does a big O mean?

Based on the computation model of the Turing machine:

Problem with input size n can be solved in O(f(n)) steps.



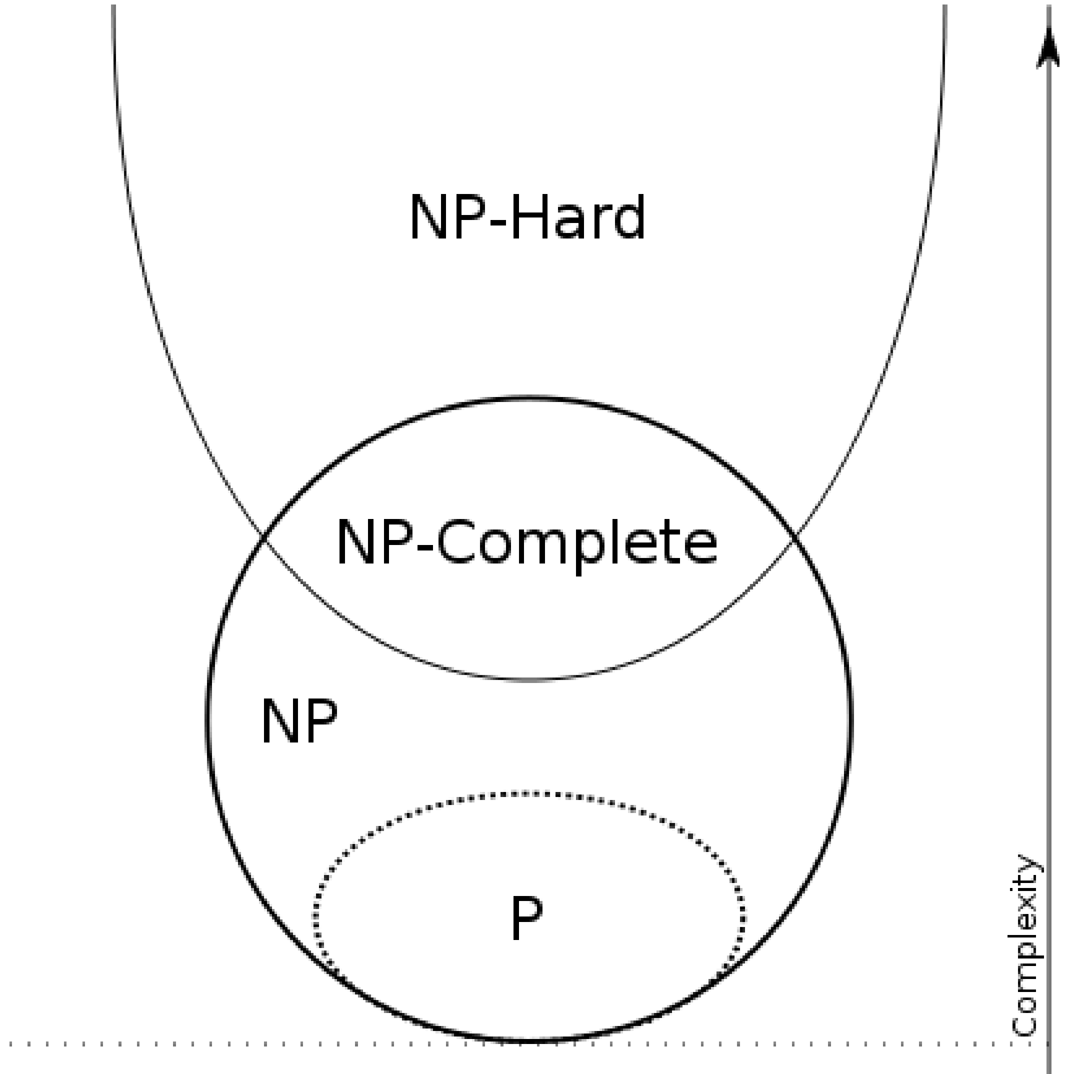
# OF BOXES	O(log n)	O(n)	O(n log n)	O(n²)	O(n!)
16	0.4 sec	1.6 sec	6.4 sec	25.6 sec	66301 years
256	0.8 sec	25.6 sec	3.4 min	1.8 hrs	8.6×10^{546} years
1024	1.0 sec	1.7 min	17 min	1.2 days	5.4×10^{2638} years

Exponential is terrible.

Classification of problems (mathematicians like classifications)

Classification of problems (mathematicians like classifications)

Classification of problems (mathematicians like classifications)



Decision problem: input n bit string, return True or False

On a Turing machine:

P: all decision problems that can be solved in polynomial time.

NP: if True, the proof can be checked in polynomial time.

For example, Sudoku
is a NP problem.

Given a solution (proof of True),
can be checked in polynomial time.

Not known if it is P: No polynomial
time general solution known.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Decision problem: input n bit string, return True or False

On a Turing machine:

P: all decision problems that can be solved in polynomial time.

NP: if True, the proof can be checked in polynomial time.

Another example, big number factorization.

Cole (1903) talk on AMS meeting: the entire content of the talk:

(1) Calculate $2^{67}-1$ (2) Calculate 193707721×761838257287

(Both equal to 147573952589676412927).

But given 147573952589676412927, can you factorize it? (NP)

Is P equal to NP?

What happens if P = NP?

Creativity is no longer valuable.

One can find proofs of all math theorems in P time.

Too good to be true.

However, there is still no proof on $P \neq NP$ yet.

What happens if $P = NP$?

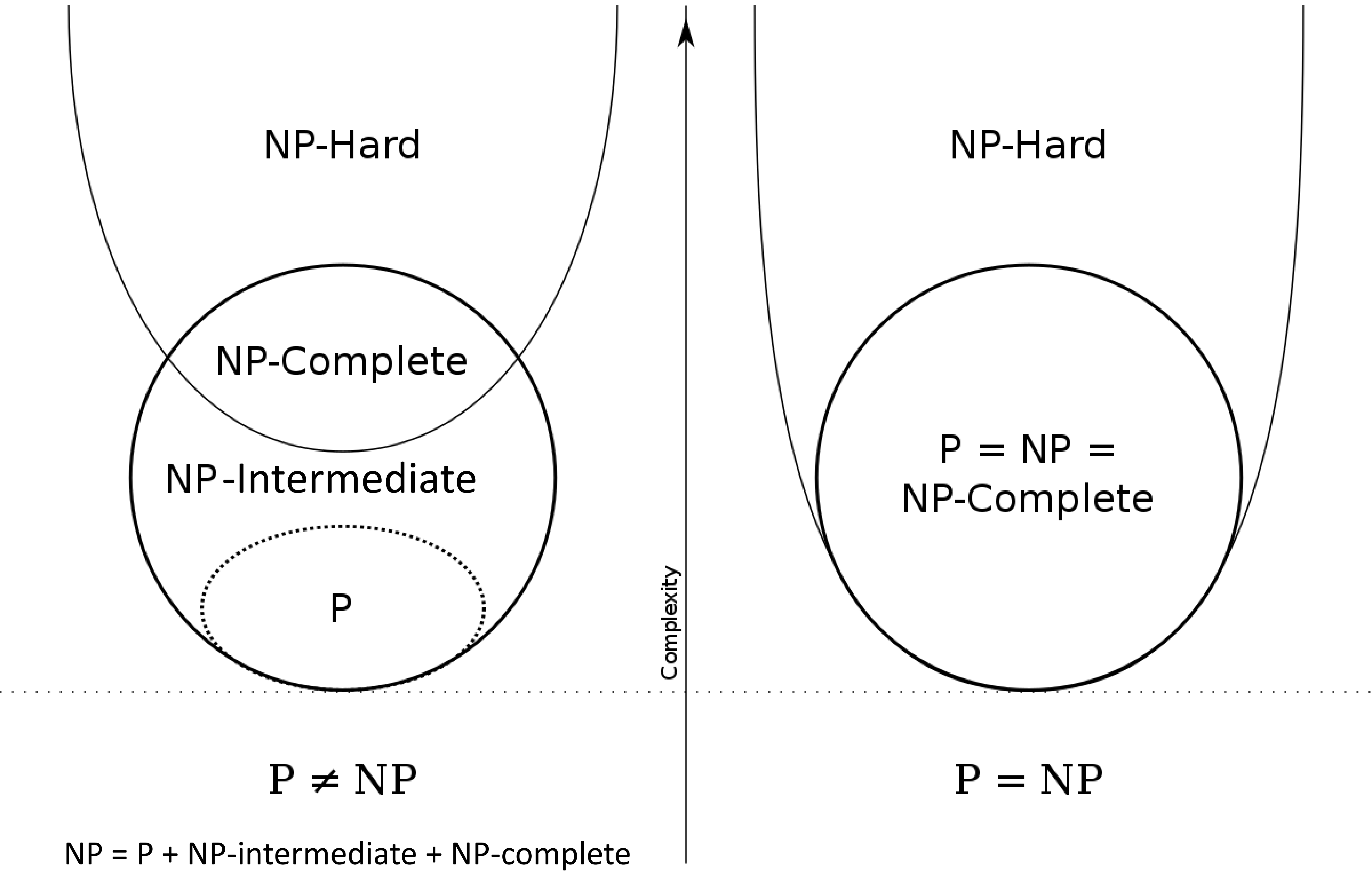
NP-hard:

A problem that any NP problem can be reduced to.

If an oracle can immediately solve a NP-hard problem,
then we can reduce any NP problem to P.

Two kinds of NP-hard problems:

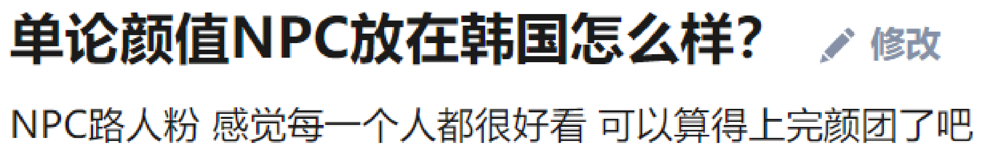
- Harder than the hardest problems in NP, e.g. halting problem.
- As hard as the hardest problems in NP: NP-complete.



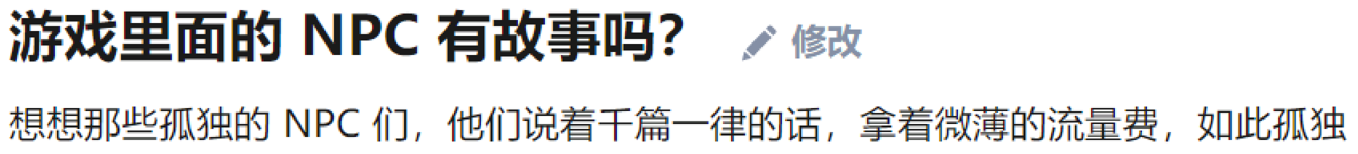
NP = P + NP-intermediate + NP-complete

Ladner's theorem (1975): If $P \neq NP$, NP-intermediate is not empty.

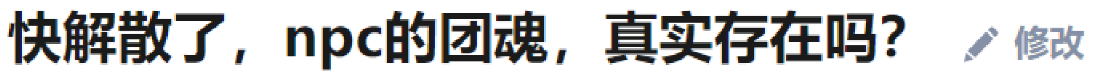
More about NPC

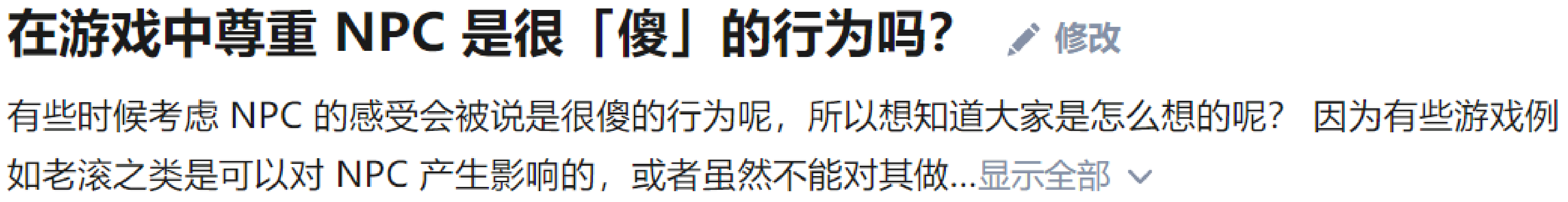

单论颜值NPC放在韩国怎么样？

NPC路人粉 感觉每一个人都很好看 可以算得上完颜团了吧


游戏里面的 NPC 有故事吗？

想想那些孤独的 NPC 们，他们说着千篇一律的话，拿着微薄的流量费，如此孤独


快解散了，npc的团魂，真实存在吗？


在游戏中尊重 NPC 是很「傻」的行为吗？

有些时候考虑 NPC 的感受会被说是很傻的行为呢，所以想知道大家是怎么想的呢？ 因为有些游戏例如老滚之类是可以对 NPC 产生影响的，或者虽然不能对其做...显示全部

More about NP-Complete Problems

If one can solve one NPC, one can solve all NP problems.
Does such NPC ever exist?

Abstract proof: Yes, with the following construction of a NPC:
Given a polynomial-time Turing machine M, find an input y that causes M(y) to accept.
[accept by a polynomial-time Turning machine means this input can halt within polynomial time.]

NP: Can be checked by actually run & return in polynomial time
NPC: Given any NP problem -> convert to a Turning machine M with solution y to halt. The above program finds y for you.

More about NP-Complete Problems

If one can solve one NPC, one can solve all NP problems.

Does such NPC ever exist?

Explicit construction: the 3-satisfiability problem.

For n Boolean variables $x_1, \ldots, x_n$,

given a set of constraints, each has at most 3 vars, e.g.

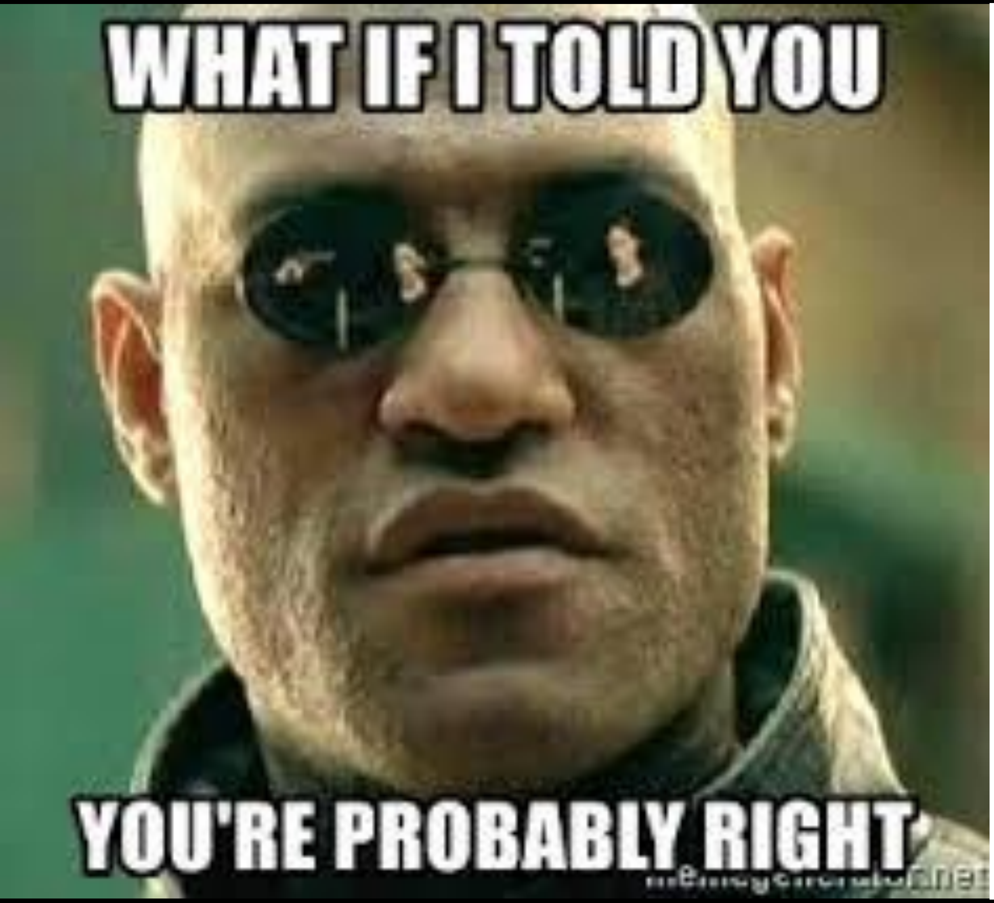
$$x_2 \; or \; x_5 \; or \; not(x_6)$$

$$not(x_2) \; or \; x_4$$

$$not(x_4) or \; not \; (x_5) \; or \; x_6$$

Is there set of variables $x_1, \ldots, x_n$ s.t. all constraints are true?

In NP: obvious. Is NP-complete: Cook-Levin Theorem (1971)



PP: Probabilistic Polynomial-Time

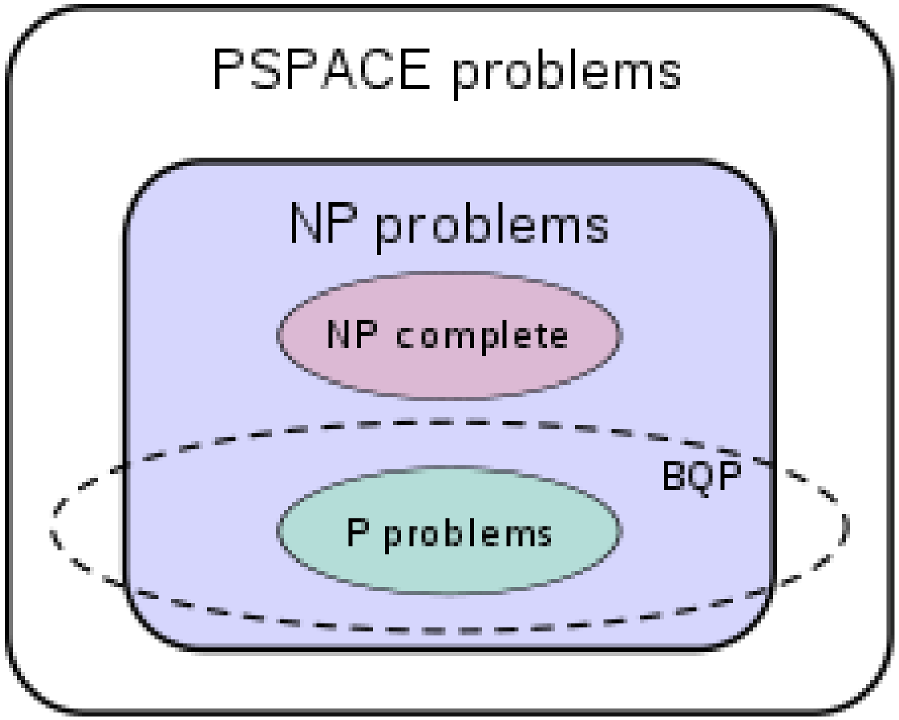
Exists a polynomial-time algorithm to accept with p>1/2 if the answer is yes or p<1/2 if the answer is no

BPP: Bounded-Error PP

Exists a polynomial-time algorithm to accept with p>2/3 if the answer is yes or p<2/3 if the answer is no

Where is quantum computing?

Bounded-error quantum polynomial time (BQP)



P<BQP or P=BQP unknown.

Relation between BQP and NP unknown.

Computational complexity [edit]

Main article: Computational complexity theory

- P versus NP problem
- What is the relationship between BQP and NP?
- NC = P problem
- NP = co-NP problem
- P = BPP problem
- P = PSPACE proble
- L = NL problem
- PH = PSPACE problem
- L = P problem
- L = RL problem
- Unique games conjecture
- Is the exponential time hypothesis true?
    - Is the strong exponential time hypothesis (SETH) true?
- Do one-way functions exist?
    - Is public-key cryptography possible?
- Log-rank conjecture

List of unsolved problems in computer science

From Wikipedia, the free encyclopedia

Programming Paradigms

1960s – 70s: Unstructured vs Structured

GOTO 1960s – 70s

http://david.tribble.com/text/goto.html

Towards the modern era:

- Procedural programming
  (Traditional structured programming)
- Object-oriented programming
- Functional programming

Object-oriented programming:

To model objects, a few considerations:

- Pack data and functions into an object
- Multiple independent realizations
- Inheritance, encapsulation, polymorphism, composition

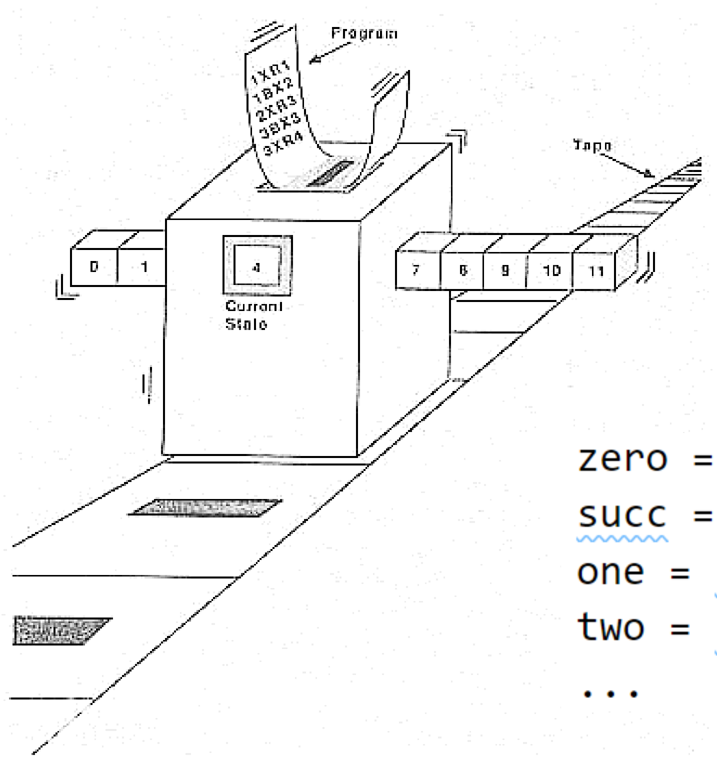
Interface: What an object can do.

- Encapsulation: Hide internal details.
- Polymorphism: Allow behavior differences.

Why OO?

Think about a car engine

Functional programming: origin



Turing Machine

Alan Turing 1930s

== λ-calculus

Alonzo Church 1930s

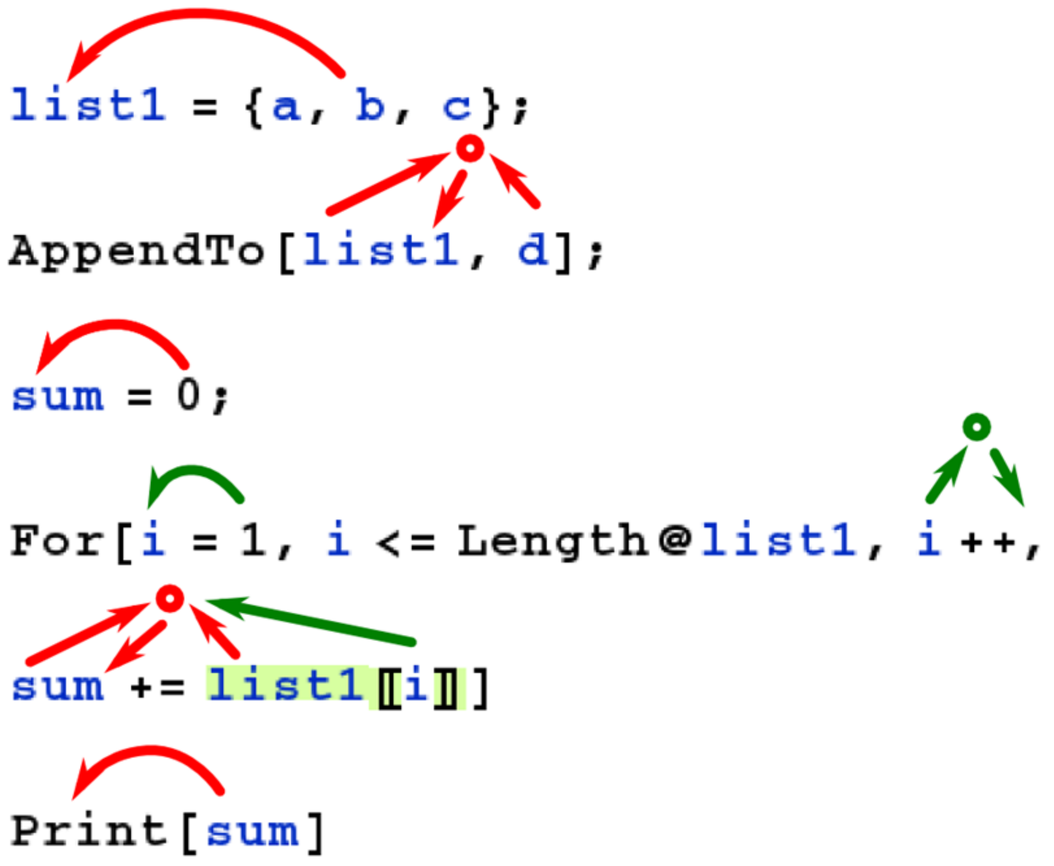
```
zero = lambda f: lambda x: x
succ = lambda n: lambda f: lambda x: f(n(f)(x))
one = succ(zero)
two = succ(one)
...
```

Turing Machine and λ-calculus can simulate each other

Imperative Programming

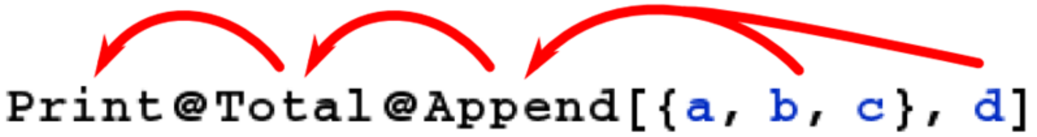
Functional programming:

Program as evaluating math functions, eliminate side effects.



```
list1 = {a, b, c};

AppendTo[list1, d];

sum = 0;

For[i = 1, i <= Length@list1, i ++,

sum += list1[[i]] ]

Print[sum]
```

Functional Programming



```
Print@Total@Append[{a, b, c}, d]
```

Higher order functions

A functional: function of function $S[\phi] = \frac{1}{2} \int d^4 x \, (\partial \phi)^2$

```
SortBy[{{1, 2}, {0, 10}, {5, -1}}, Total]
```

Anonymous functions (lambda calculus)

```
f[x1_, x2_] := x1 + x2
ClearAll[f]

Function[{x1, x2}, x1 + x2]
Function[Slot[1] + Slot[2]]
#1 + #2 &


SortBy[Range@10, Mod[#, 3] &]

FixedPoint[(# + 2 / # ) / 2 &, 1., SameTest → (Abs[#1 - #2] < 0.01 &)]
```

Remark: functional aspects of Linux shell

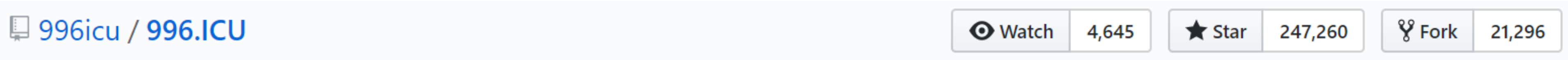
* Program "Do one thing, and do it well"
* Defined stdin, stdout, pipe
* Plain text flows through pipes

E.g.

```
  ps aux | grep conky | grep - v grep | awk ' {print $2}' | xargs kill
```

Software Engineering

See e.g. “The Mythical Man-Month” (人月神话)


996icu / 996.ICU

In the 1960s to 1980s, in software development,
much more problems encountered than expected.

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful!

To put it quite bluntly:

as long as there were no machines, programming was no problem at all;

when we had a few weak computers, programming became a mild problem,

and now we have gigantic computers, programming has become an equally gigantic problem.

— Edsger Dijkstra (1972, ACM Turing Award Lecture)

Coding before the 70s

or,

now coding at school

From 70s to now:

Coding for production

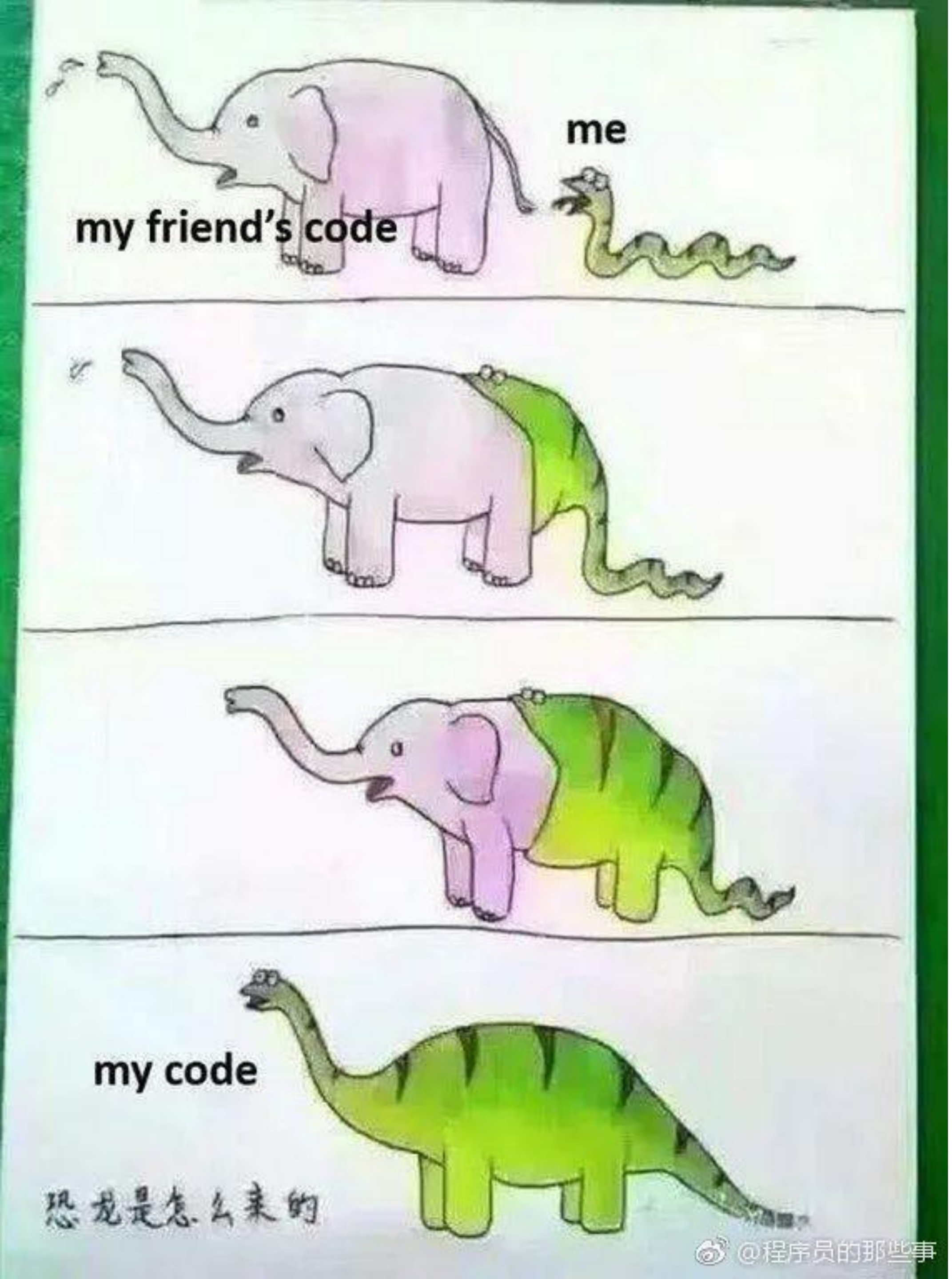
如何激怒一位计算机科学爱好者?

Long road from Programming to Production

Lessons

Balance between standard and hand-made tools

Don't reinvent the wheel!

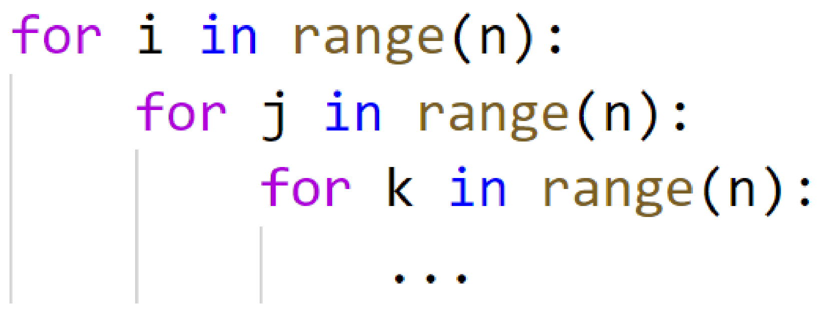


But don't use the wheels ugly, either.

Also try to make our code reusable.

只要你足够废物，就没人可以利用你
As long as you have enough waste, no one can take advantage of you

Toward self-documenting code



```python
for i in range(n):
    for j in range(n):
        for k in range(n):
            ...
```

Reviewer: index names should reflect their meanings

```python
for i in range(n):
    for j in range(n):
        for k in range(n):
            ...
```

Reviewer: index names should reflect their meanings

```python
for index in range(n):
```

```python
for i in range(n):
    for j in range(n):
        for k in range(n):
            ...
```

Reviewer: index names should reflect their meanings

```python
for index in range(n):
    for jndex in range(n):
        for kndex in range(n):
            ...
```

Reviewer: ಠ_ಠ

```
            ...
```

Reviewer: index names should reflect their meanings

```python
for index in range(n):
    for jndex in range(n):
        for kndex in range(n):
            ...
```

Reviewer: ಠ_ಠ

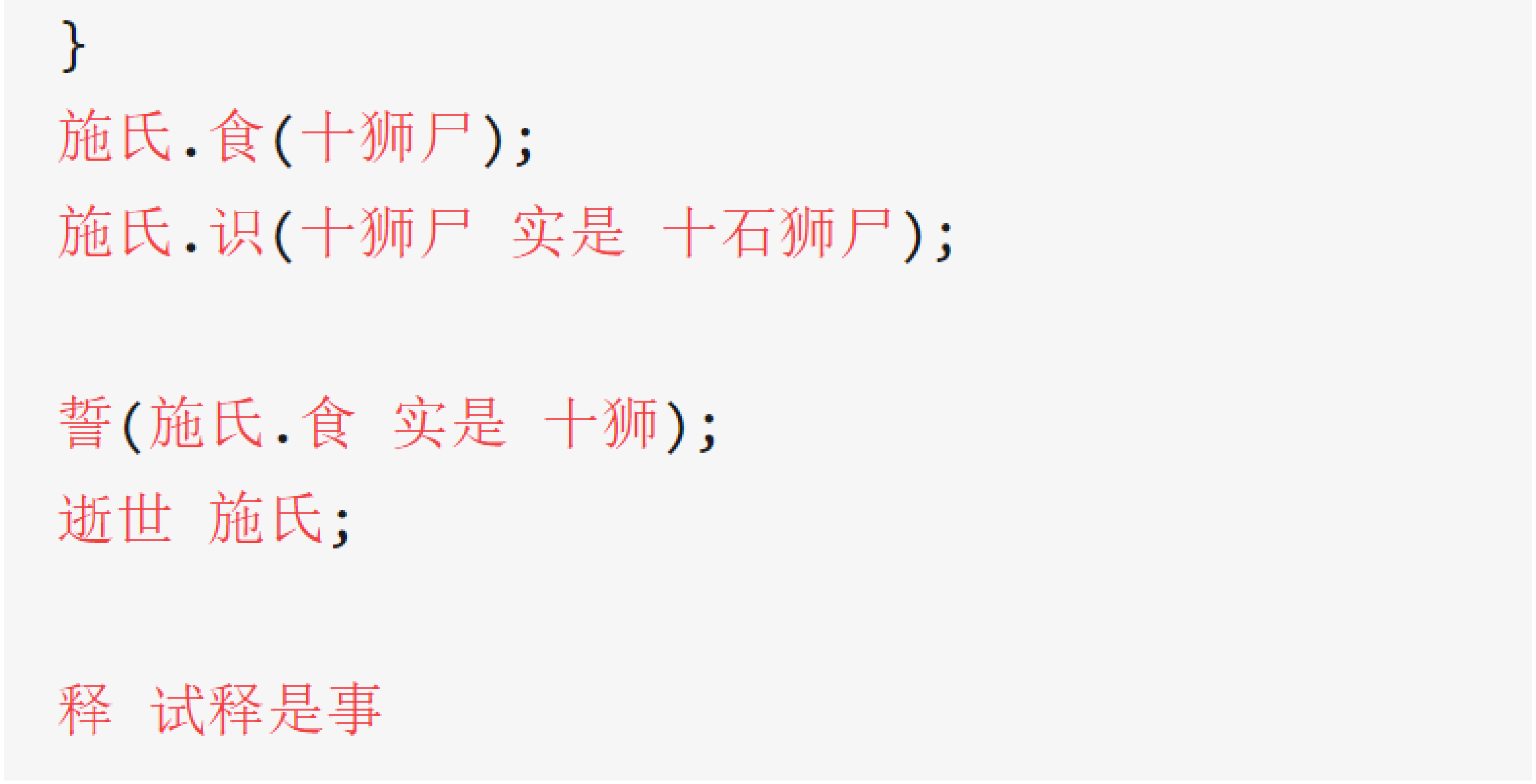
```python
for index in range(n):
    for indey in range(n):
        for indez in range(n):
            ...
```

Reviewer: 卒

为什么没有或流行的拼音编程语言?

```
#include "施氏食狮史.h"


#define 始 new
#define 逝世 delete
#define 誓 assert
#define 十时 "10:00"
#define 是时 SetSystemTime
#define 视适实施 while
#define 是 =
#define 实是 ==
#define 事实 True
#define 释 //
```

```
施氏 是 始 石室诗士();
施氏.嗜 是 狮;

是时(十时);
施氏.适(市);
施氏.视(狮);
视适实施(!十狮逝世()){
    矢(狮);
}
施氏.拾(十狮尸);

施氏.适(石室);
视适实施(石室.湿 实是 事实){
    施氏.侍.拭(石室);
```



```
}
施氏.食(十狮尸);
施氏.识(十狮尸 实是 十石狮尸);

誓(施氏.食 实是 十狮);
逝世 施氏;

释 试释是事
```

麻煩改成拼音編程，在綫等，挺急的

Try to write self-documenting code:

Functions as verbs; variables as nouns; classes as subjects

Code as meaningful as sentences.

Write tests. Decent test coverage.

Test driven if helps (write tests before code).

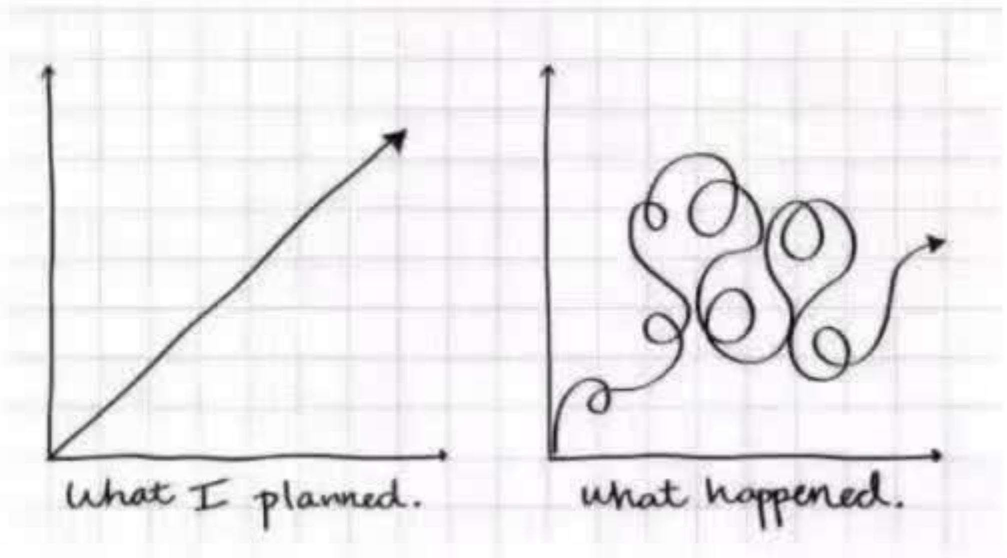
一个测试工程师走进一家酒吧，要了一杯啤酒
一个测试工程师走进一家酒吧，要了一杯咖啡
一个测试工程师走进一家酒吧，要了0.7杯啤酒
一个测试工程师走进一家酒吧，要了-1杯啤酒
一个测试工程师走进一家酒吧，要了2^32杯啤酒
一个测试工程师走进一家酒吧，要了一杯洗脚水
一个测试工程师走进一家酒吧，要了一杯蜥蜴
一个测试工程师走进一家酒吧，要了一份asdfQwer@24dg!&*(@
一个测试工程师走进一家酒吧，什么也没要
一个测试工程师走进一家酒吧，又走出去又从窗户进来又从后门出去从下水道钻进来
一个测试工程师走进一
一个测试工程师走进一家酒吧，要了一杯烫烫烫的锟斤拷
一个测试工程师走进一家酒吧，要了NaN杯Null
1T测试工程师冲进一家酒吧，要了500T啤酒咖啡洗脚水野猫狼牙棒奶茶
1T测试工程师把酒吧拆了
一个测试工程师化装成老板走进一家酒吧，要了500杯啤酒并且不付钱
一万个测试工程师在酒吧门外呼啸而过
一个测试工程师走进一家酒吧，要了一杯啤酒';DROP TABLE 酒吧

测试工程师们满意地离开了酒吧。然后一名顾客点了一份炒饭，酒吧炸了

Credit: 今日飞雪https://www.zhihu.com/question/20034686/answer/52063718

Realistic planning

暑假前制定了一个plan，半个暑假结束了，完成了个p，因为lan。



Live with bugs

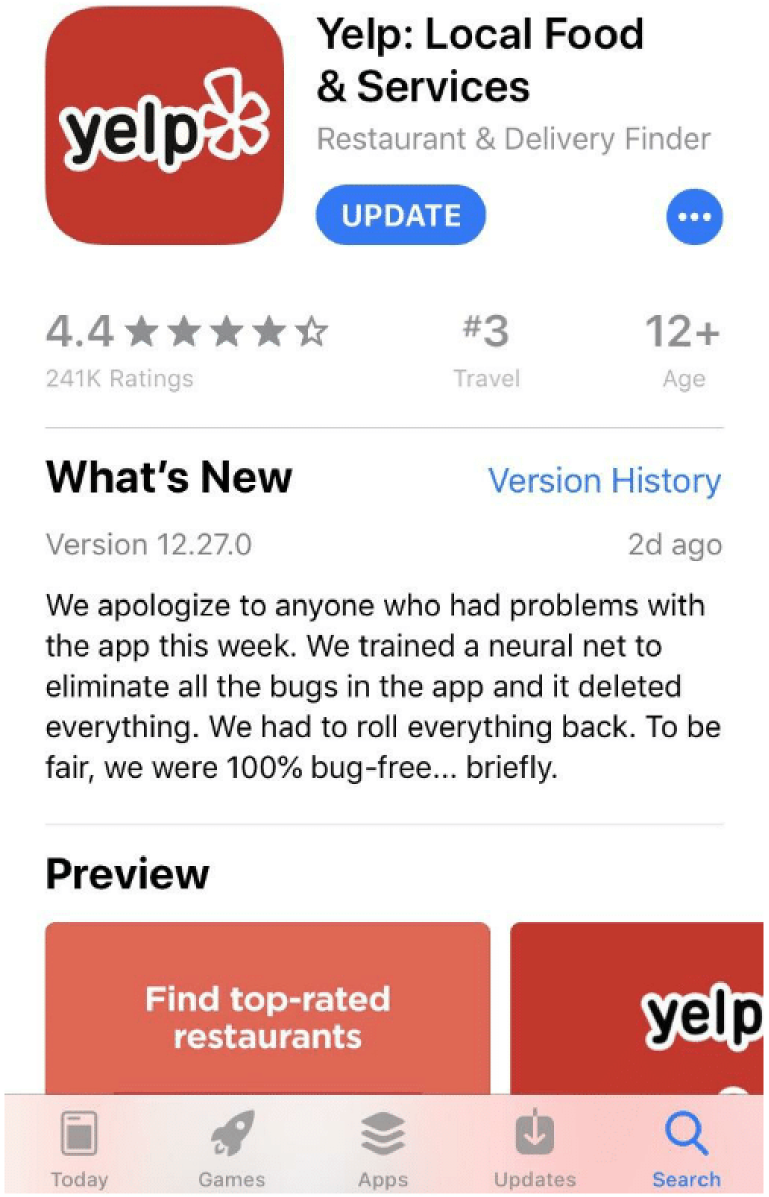
Era of long human life expectation:

“Living with a chronic illness”

Software engineering:

“Living with bugs”

No way to eliminate all bugs, except --


Yelp: Local Food & Services
Restaurant & Delivery Finder
UPDATE
4.4 ★★★★☆
241K Ratings
#3
Travel
12+
Age
Preview
Find top-rated restaurants
Today
Games
Apps
Updates
Search

What’s New

Version History

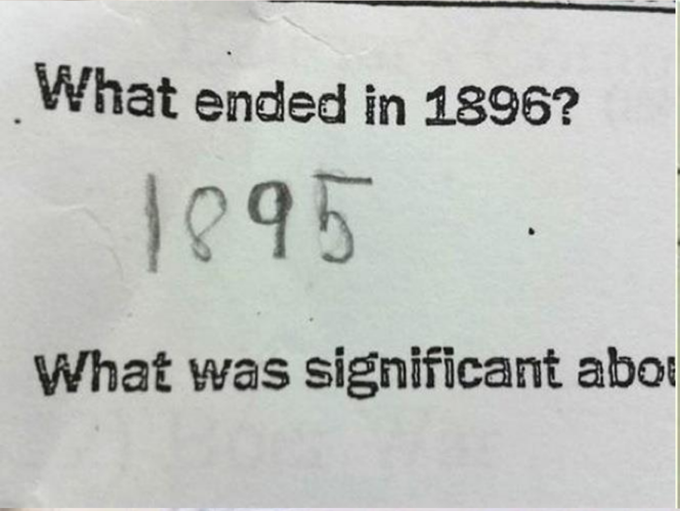
Version 12.27.0 2d ago

We apologize to anyone who had problems with the app this week. We trained a neural net to eliminate all the bugs in the app and it deleted everything. We had to roll everything back. To be fair, we were 100% bug-free... briefly.

Thanks to AI, Yelp iOS app achieved 100% bug free for a brief moment.

Nobody understands everything of a large project.

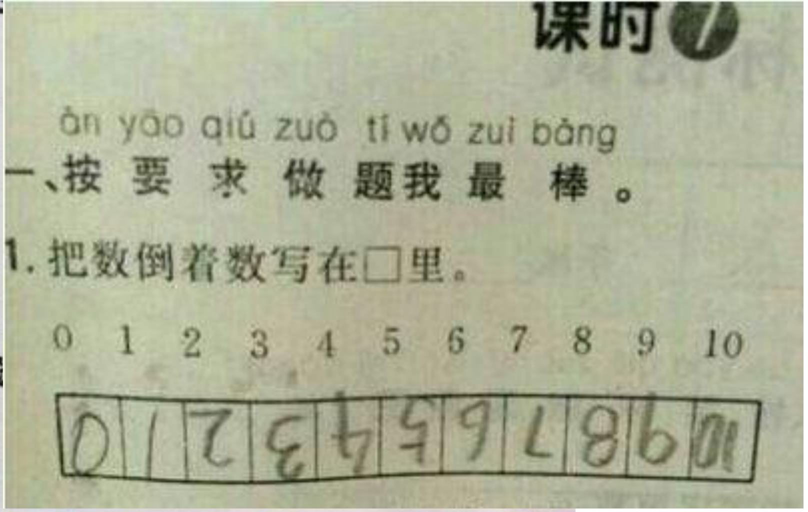
Try to make collaborators’ life easier.



What ended in 1896?

1895

What was significant abo



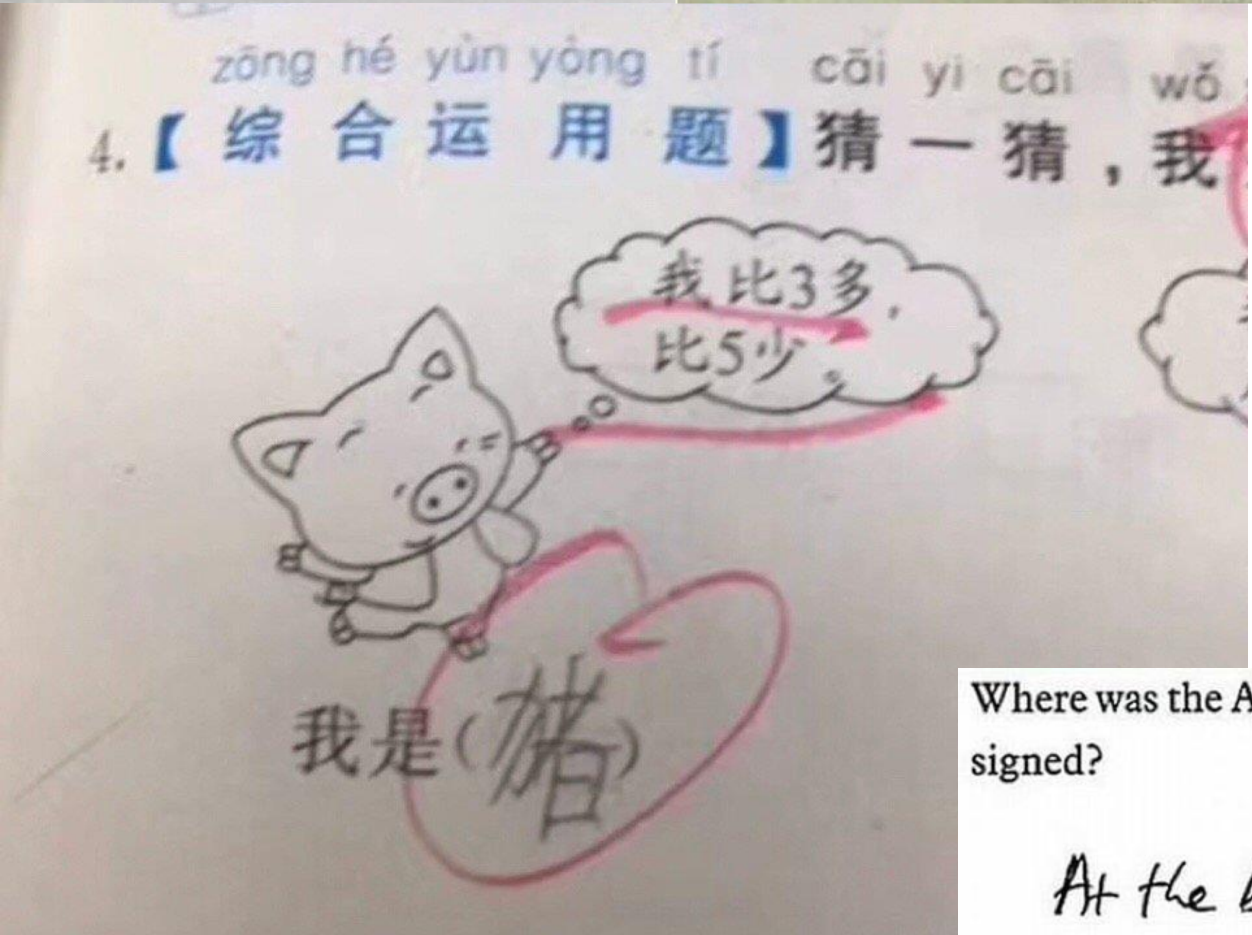
课时 7

àn yāo qiú zuò tí wǒ zuì bàng

一、按要求做题我最棒。

1. 把数倒着数写在□里。

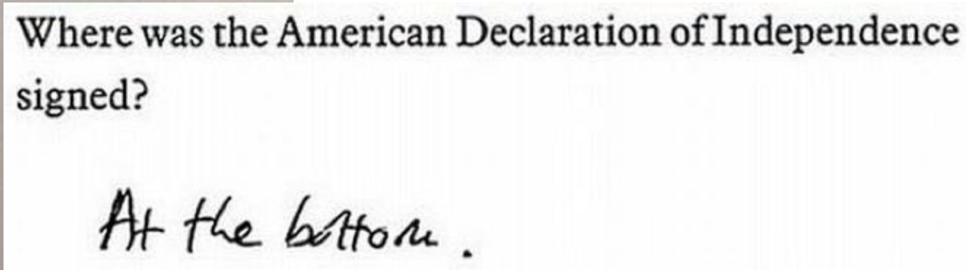
0	1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9	10



zōng hé yùn yòng tí cāi yi cāi wǒ

4.【综合运用题】猜一猜，我

我是（猪）



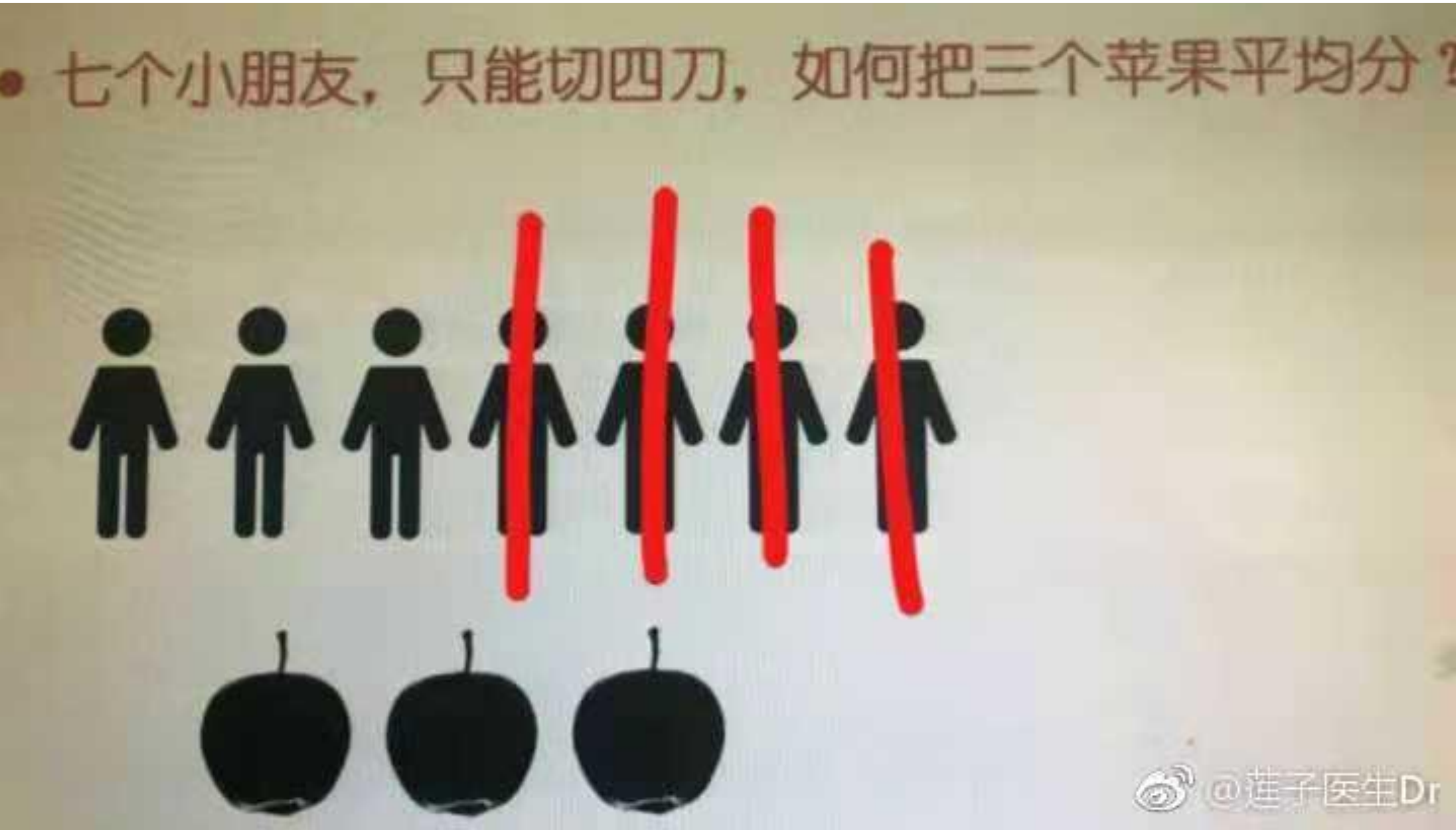
Where was the American Declaration of Independence signed?

At the bottom.

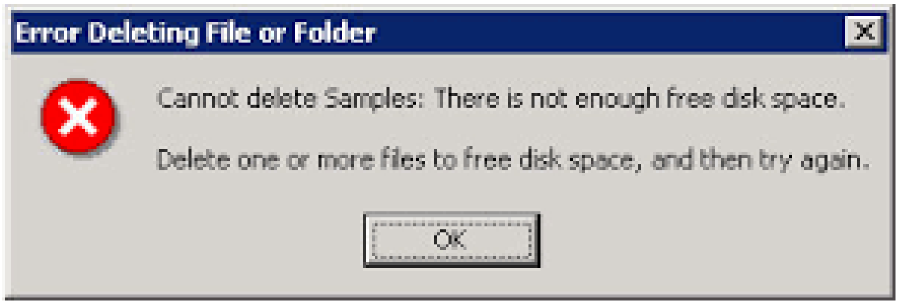
Be explicit.

Don’t expect that your colleagues can always guess what you mean.

Software engineering is NOT an IQ test
Otherwise disasters happen

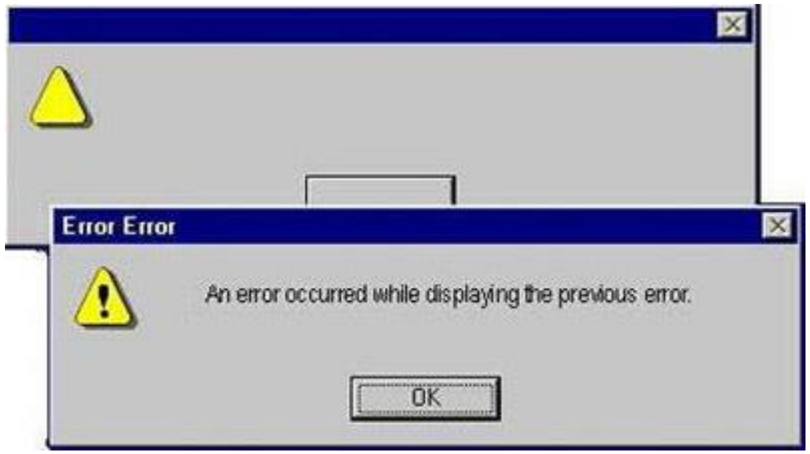


Handle exceptions properly

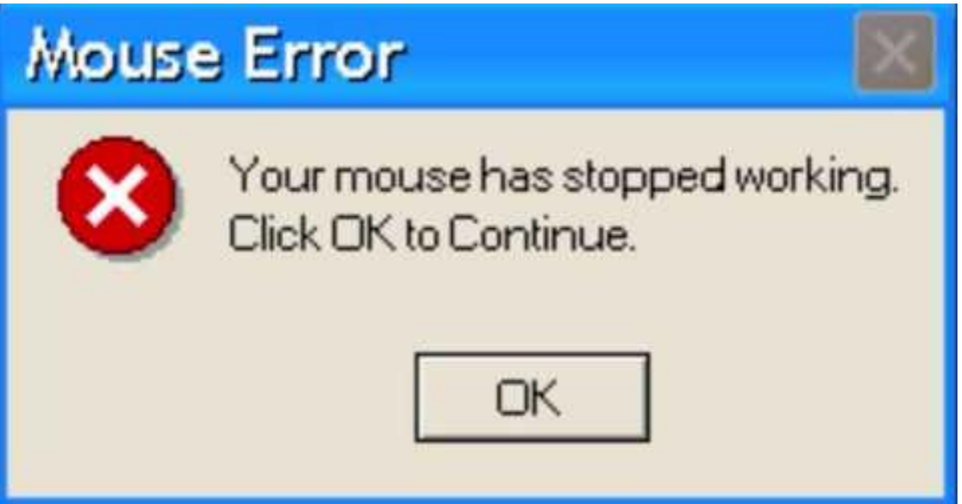

Error Deleting File or Folder

Cannot delete Samples: There is not enough free disk space.

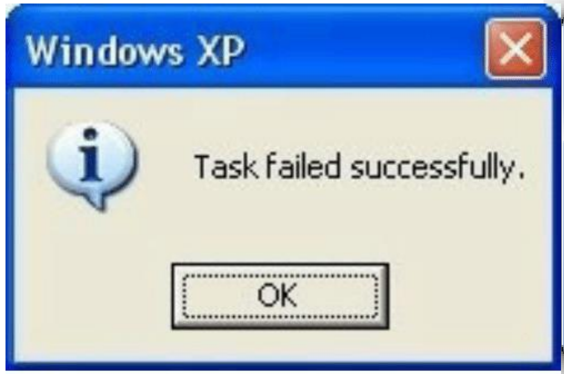
Delete one or more files to free disk space, and then try again.


Error Error

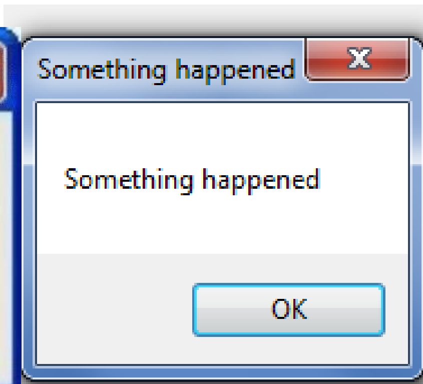
An error occurred while displaying the previous error.


Mouse Error

Your mouse has stopped working. Click OK to Continue.


Windows XP

Task failed successfully.


Something happened

Something happened