



# Complexity

上課補充 by double  
Credit by Howard41436

# Sprout



- 影片看了嗎
- Q&A

Sprout



## 今天要學的東西

- 今天的課以理論為主，不太會寫到題目
- 但是對於複雜度的知識絕對是寫程式最重要的一環
- 大家都怎麼算複雜度？

Sprout



## 今天要學的東西

- 今天的課以理論為主，不太會寫到題目
- 但是對於複雜度的知識絕對是寫程式最重要的一環
- 大家都怎麼算複雜度？



**Benson Tan** 哈，你真瞭解 big-O? 一個指令沒有迴圈我們都稱為  $O(1)$ ，不知你在哪裡學到  $O(\log N)$ ，你有學過 compiler 嗎? interpreter language 也有 big-O 知道吧?

...

讚 · 回覆 · 56分鐘



# Sprout



## 今天要學的東西

- 今天的課以理論為主，不太會寫到題目
- 但是對於複雜度的知識絕對是寫程式最重要的一環
- 大家都怎麼算複雜度？
- 「沒有迴圈就是  $O(1)$ ，一層迴圈就是  $O(n)$ ，兩層就  $O(n^2)$ 」
- By Foxconn Specialist

Sprout



## 今天要學的東西

- 今天的課以理論為主，不太會寫到題目
- 但是對於複雜度的知識絕對是寫程式最重要的一環
- 大家都怎麼算複雜度？
- 「沒有迴圈就是  $O(1)$ ，一層迴圈就是  $O(n)$ ，兩層就  $O(n^2)$ 」
- By Foxconn Specialist
- 真的是如此嗎？

Sprout





## 複雜度的正確算法

- 數迴圈有幾層的優缺點：
- 優點：大部分的程式我們大概都可以這樣算啦
- 缺點：
  1. 有些操作可能不是  $O(1)$
  2. 無法計算遞迴的時間複雜度
  3. 可能會錯估均攤複雜度

Sprout



## 複雜度的正確算法

- 首先，先來討論複雜度的正確算法
- 正式的數學定義手寫作業有，也會讓大家證一些東西
- 不過在 99% 的時候，複雜度都這樣算就好了：
- 計算總共需要的操作數，留下量級最大那一項，常數去掉
- ex:  $O(3n^2 \log n + 2n^2 + 4n + \log n) = O(n^2 \log n)$

Sprout





## 案例 1

- 計算  $a$  的  $n$  次方模  $10^9+7$ :
- Foxconn specialist:
- `return a**n % 1000000007`
- 沒有迴圈, 所以  $O(1)$
- 實測時間

Sprout



## 案例 1

```
import time
time1=time.time()
a=880301
n=26
mod=1e9+7
x=a**n
time2=time.time()
n=1000026
y=a**n
time3=time.time()
print(time2-time1)
print(time3-time2)
```

Sprout



## 案例 1

```
import time
time1=time.time()
a=880301
n=26
mod=1e9+7
x=a**n
time2=time.time()
n=1000026
y=a**n
time3=time.time()
print(time2-time1)
print(time3-time2)
```

```
2.86102294921875e-06
4.153032302856445
```



## 案例 1

- 事實上，計算  $a$  的  $n$  次方時，通常是在  $n$  很小，或著是在做  $a$  或  $n$  為浮點數的運算時，我們才會用內建的 `pow` 函數，且將複雜度當作  $O(1)$
- 當計算整數  $a$  的  $n$  次方時，由於乘法會變慢，時間會增加到不可忽視的程度，這時候就明顯不是  $O(1)$  了

Sprout



## 案例 2

- 估算遞迴的複雜度
- `gcd(p, q)`:
  - if `q = 0`:
    - return `p`
  - else:
    - return `gcd(q, p % q)`
- 這種自己呼叫自己的函式，沒有迴圈可以數，複雜度要怎麼算？

Sprout



## 案例 2

- 估算遞迴的複雜度
- `fib(n)`:
  - `if n = 1 or 2:`
    - `return 1`
  - `else:`
    - `return fib(n - 1) + fib(n - 2)`
- 這種自己呼叫自己的函式，沒有迴圈可以數，複雜度要怎麼算？

Sprout





## 案例 3

- 模擬一個 stack, 有 3 種操作
- 1. push(x)
- 2. top()
- 3. pop(k): 連續 pop 掉 k 個物品, 其中  $k \leq \text{stack 的大小}$
- 每個操作的時間複雜度是多少?
- 執行 n 次操作的總複雜度是多少?

Sprout



## 案例 3

- 模擬一個 stack, 有 3 種操作
- 1. push(x)
- 2. top()
- 3. pop(k): 連續 pop 掉 k 個物品, 其中  $k \leq \text{stack 的大小}$
- 每個操作的時間複雜度是多少?  $O(1), O(1), O(n)$
- 執行 n 次操作的總複雜度是多少?  $O(n*n) = O(n^2)$ ?

Sprout



## 案例 3

- 模擬一個stack, 有3種操作
- 1. push(x)
- 2. top()
- 3. pop(k): 連續 pop 掉 k 個物品, 其中  $k \leq \text{stack 的大小}$
- 每個操作的時間複雜度是多少?  $O(1), O(1), O(n)$
- 執行 n 次操作的總複雜度是多少?  $O(n*n)=O(n^2)$ ?
- pop 掉的物品數  $\leq$  push 進的物品數  $\leq n$
- 均攤複雜度  $O(n)$

Sprout



## 複雜度的重要性

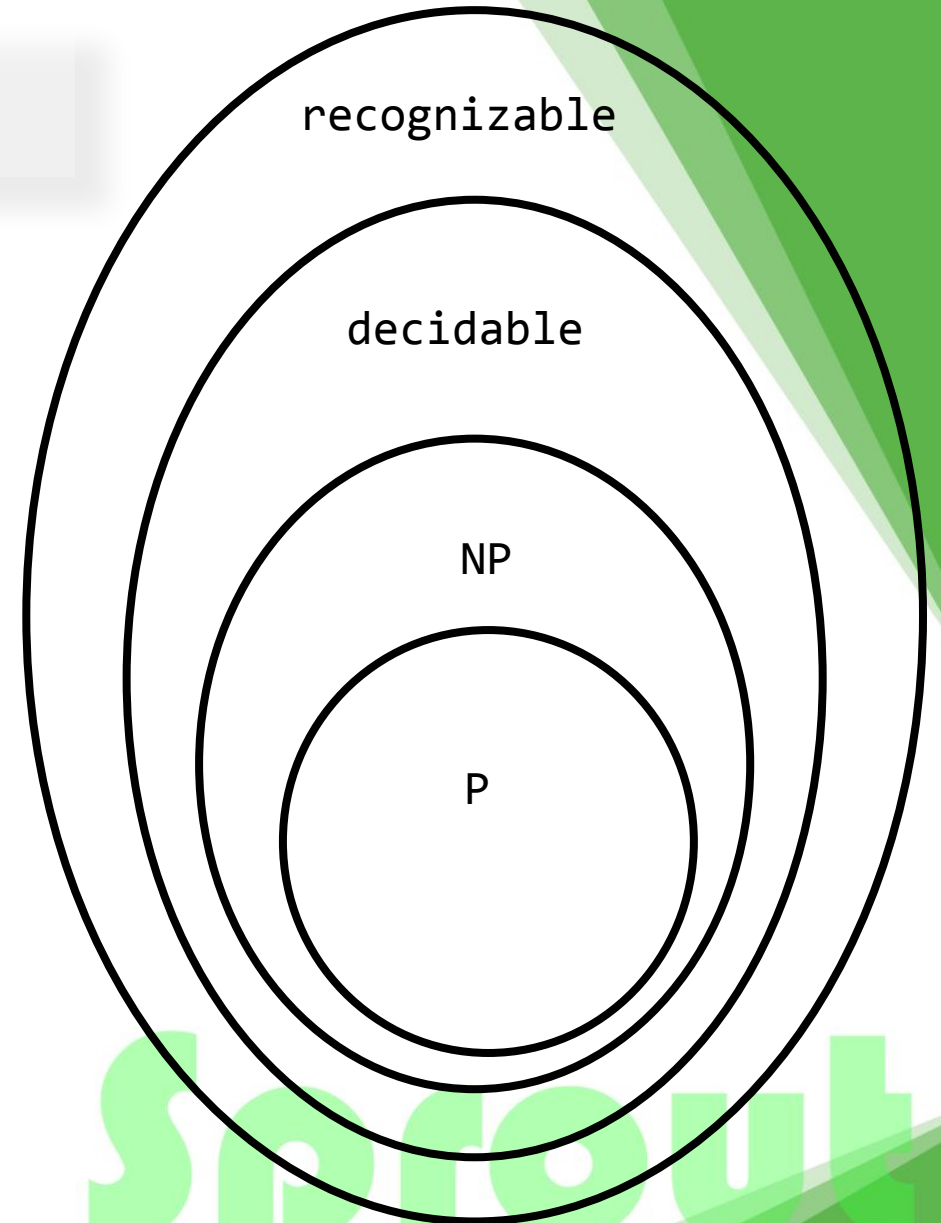
- 在競賽程式中，複雜度計算通常是為了讓我們判斷這個演算法寫了會不會 TLE
- 我們可以假設機器每秒大概跑  $10^8$  左右的操作
- 這件事也告訴我們，除非複雜度真的在  $10^8$  左右，否則演算法的常數不是很重要，如果複雜度相同挑最好寫的方式寫就好

Sprout



## 計算理論中的複雜度類

- recognizable
- decidable
- nondeterministic polynomial
- polynomial

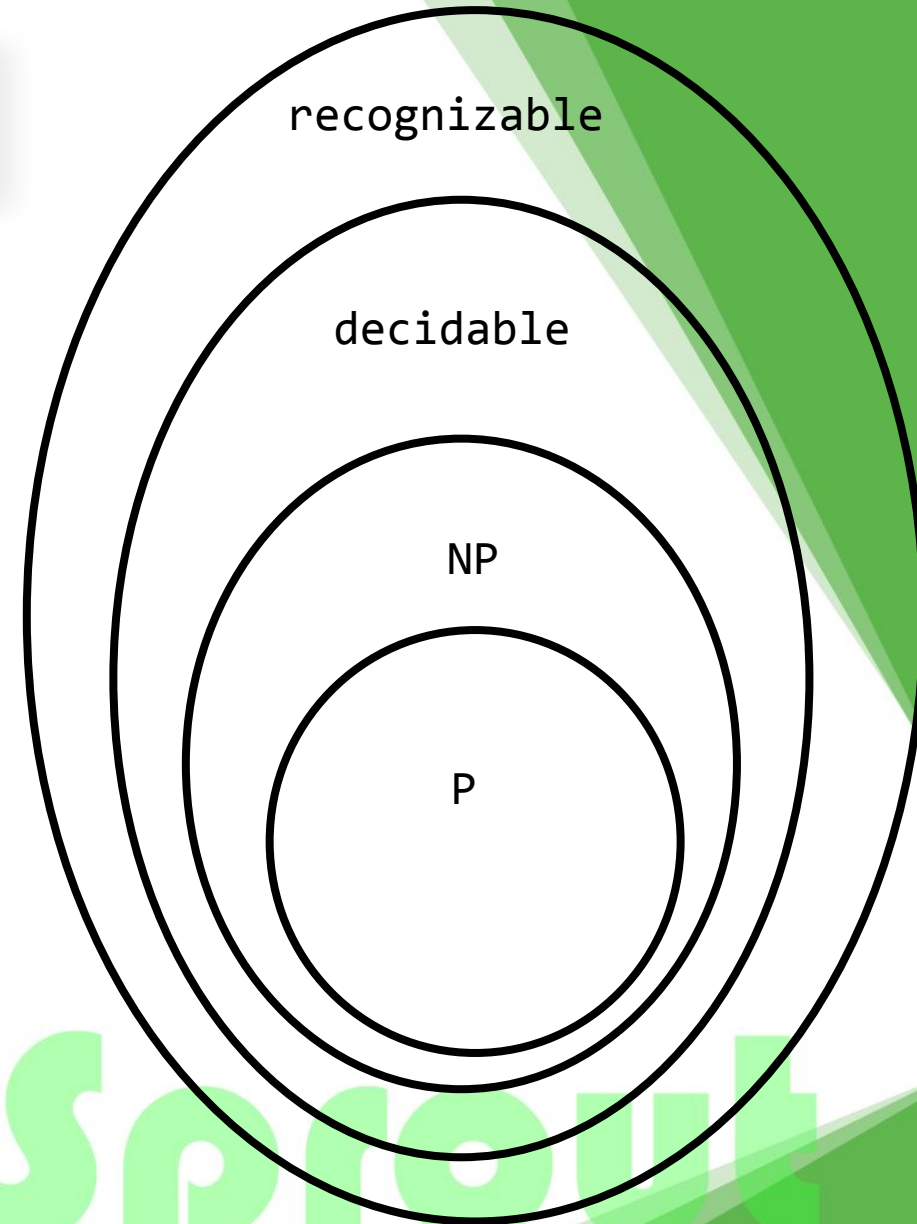


Sprout



## 計算理論中的複雜度類

- recognizable  
對於答案是 yes 的輸入, 能輸出 yes  
對於答案是 no 的輸入, 不一定跑的完
- decidable
- nondeterministic polynomial
- polynomial



Sprout





## recognizable 的例子

- 停機問題
  - 是否存在一個程式，可以讓你輸入一支程式，以及這支程式執行的輸入之後，決定這支程式在這組輸入下，會不會在有限的時間內結束？

Sprout



## recognizable 的例子

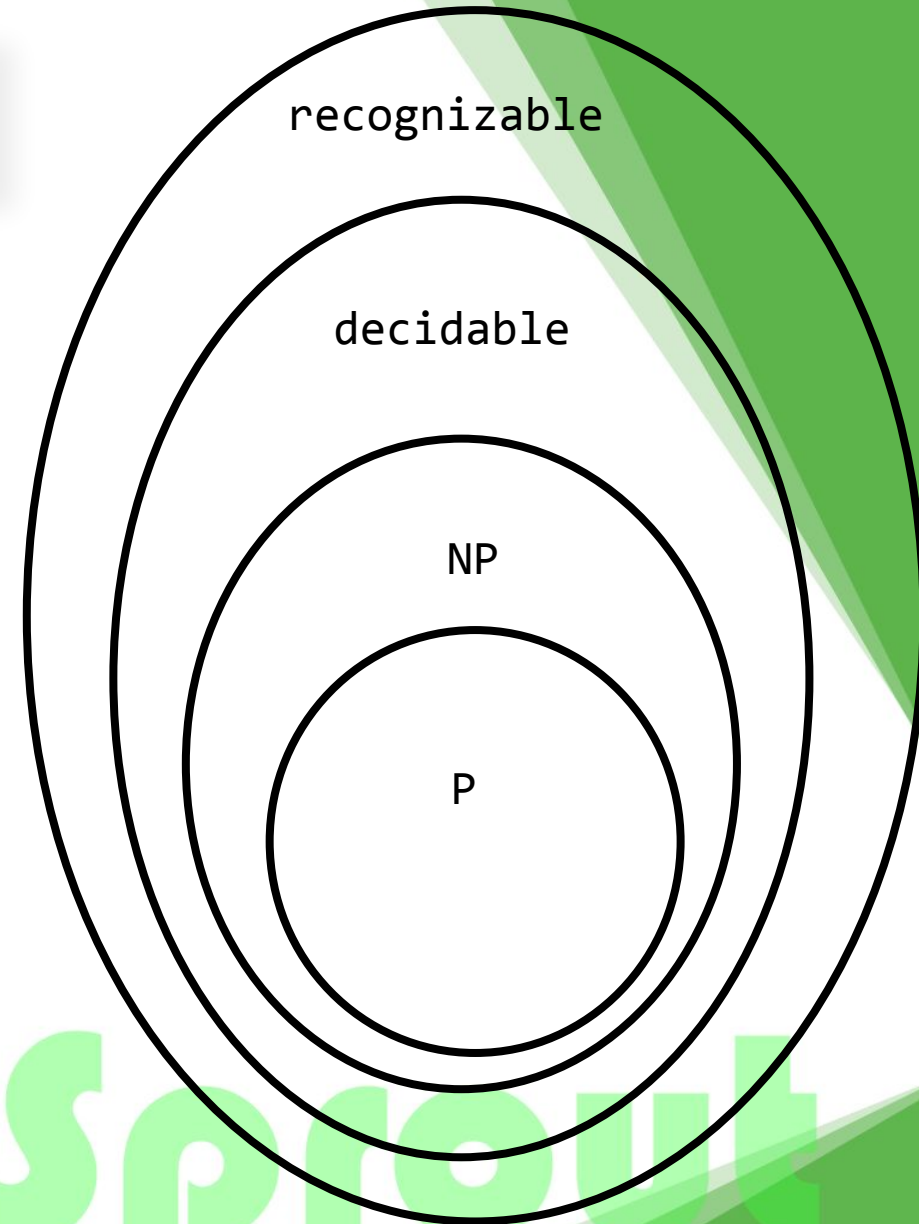
- 停機問題
  - 是否存在一個程式，可以讓你輸入一支程式，以及這支程式執行的輸入之後，決定這支程式在這組輸入下，會不會在有限的時間內結束？
- ```
int H(procedure, Input); // 死循環(0)或停機(1)
int U(P) {
    while (H(P,P)) {}
    return 0;
}
```

Sprout



## 計算理論中的複雜度類

- recognizable
- decidable  
對於答案是 yes 的, 能輸出 yes  
對於答案是 no 的, 能輸出 no  
此複雜度類的問題才有「演算法」
- nondeterministic polynomial
- polynomial



Sprout



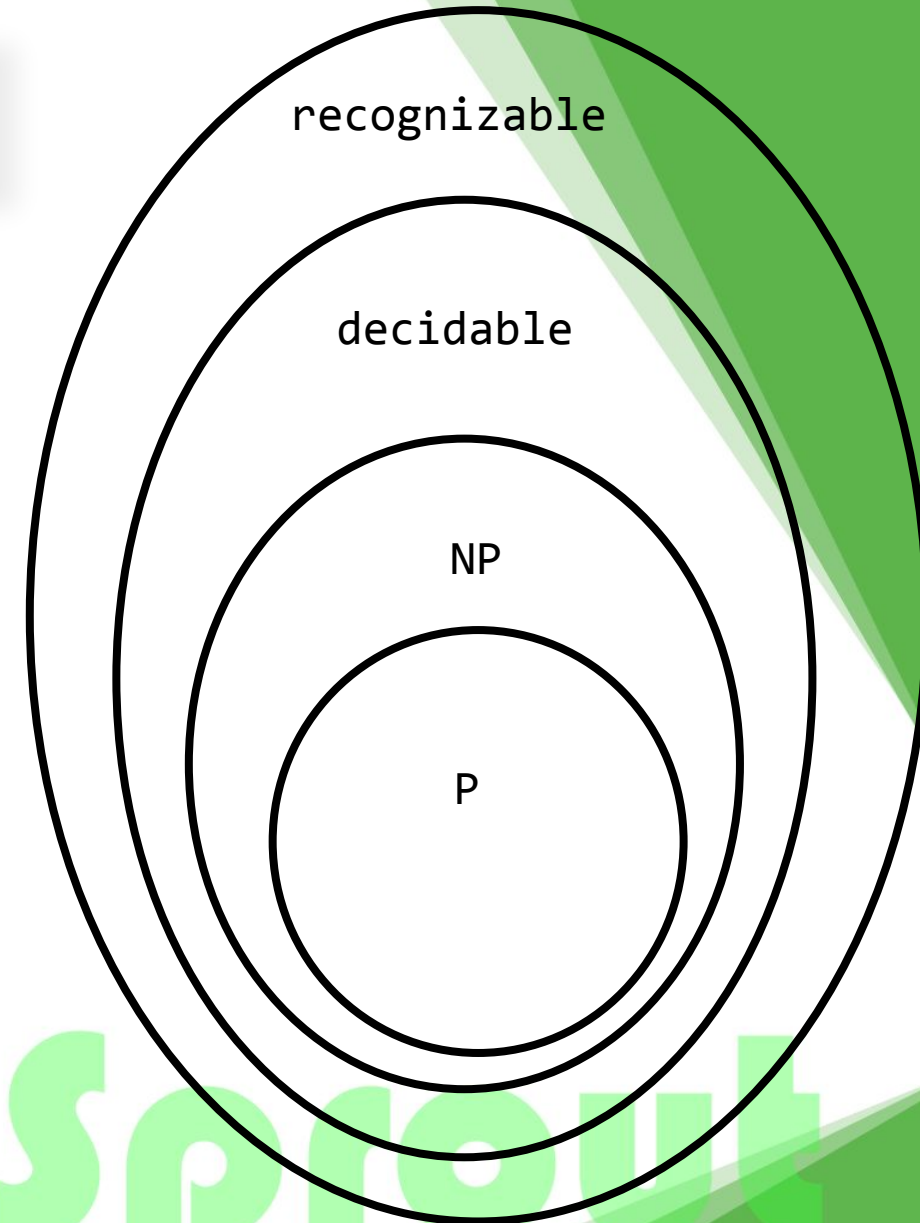
## 計算理論中的複雜度類

- recognizable
- decidable

對於答案是 yes 的, 能輸出 yes  
對於答案是 no 的, 能輸出 no

此複雜度類的問題才有「演算法」

- 演算法(algorithm)  
在數學和電腦科學之中, 指一個被定義好的、計算機可施行其指示的有限步驟或次序



Sprout



## decidable 的例子

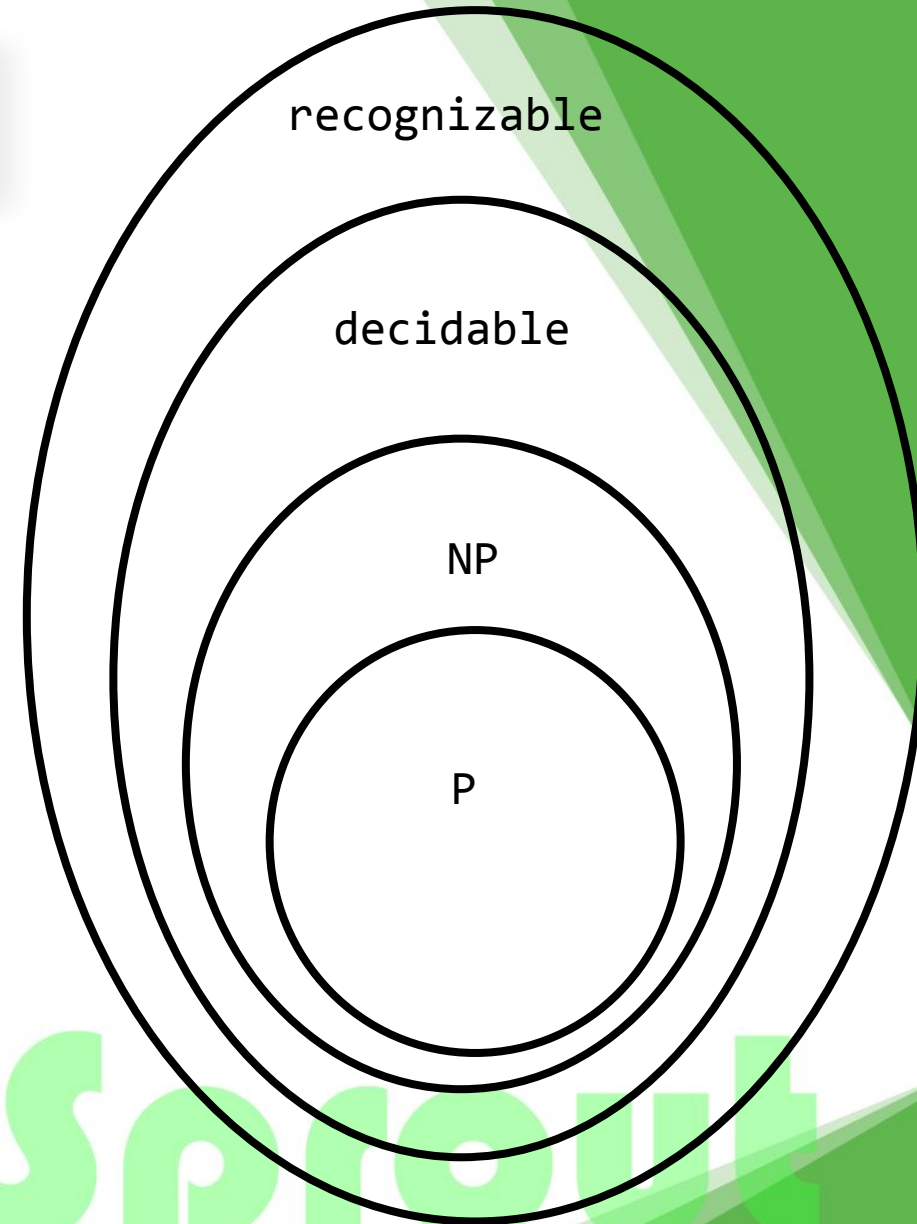
- 停機問題 weak
  - 是否存在一個程式，可以讓你輸入一支程式，以及這支程式執行的輸入之後，決定這支程式在這組輸入下，會不會在 10 秒內結束？

Sprout



## 計算理論中的複雜度類

- recognizable
- decidable
- nondeterministic polynomial  
常常聽到的 NP 問題, 並不是指非多項式時間(decidable 裡面很多非多項式但也不是 NP 的問題)  
是指如果我們多給程式一個提示, 就有辦法在多項式時間內得知答案是 yes
- polynomial



Sprout



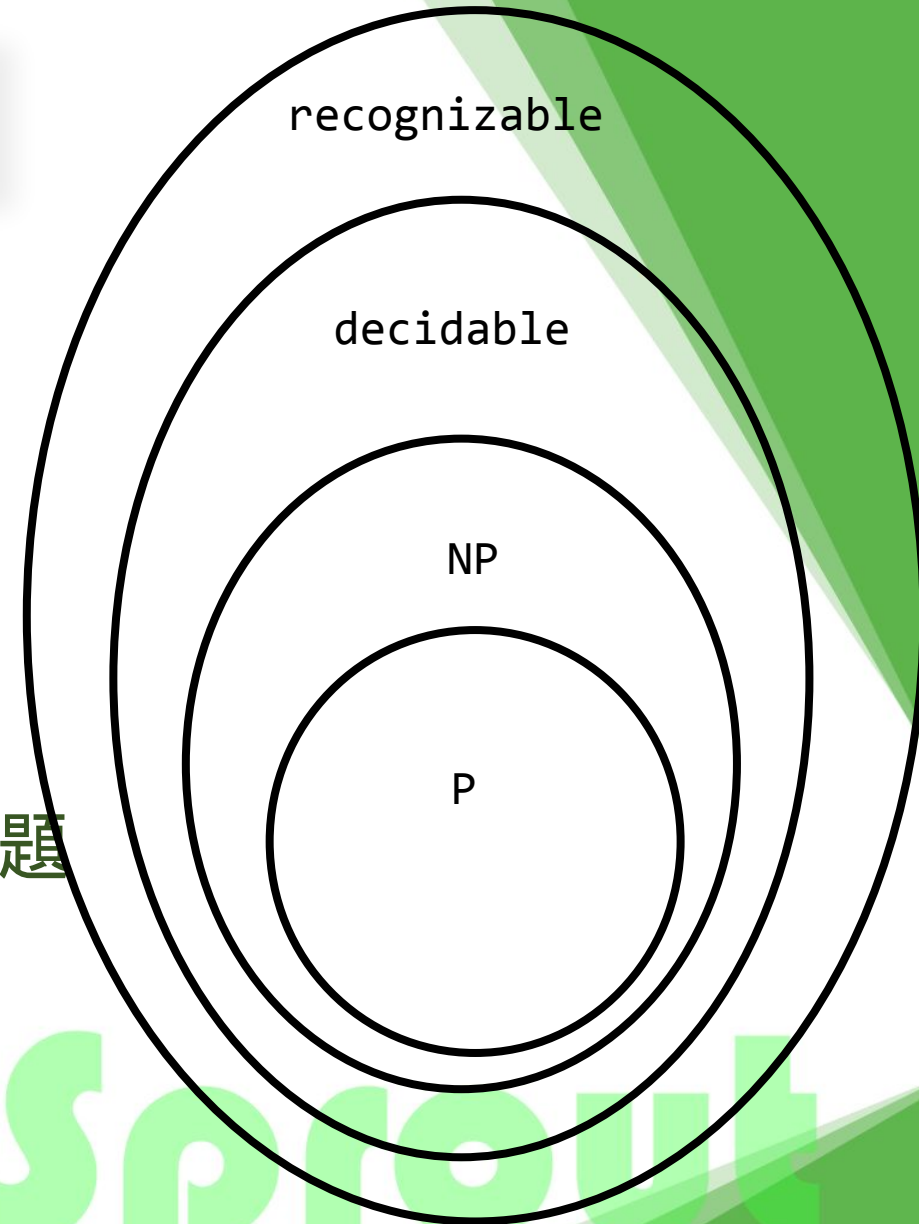


## 計算理論中的複雜度類

- recognizable
- decidable
- nondeterministic polynomial
- polynomial

這個就很直觀了，是指能夠在多項式時間內判斷答案是 yes 還是 no

一般我們能有效率解決的問題都是 P 問題

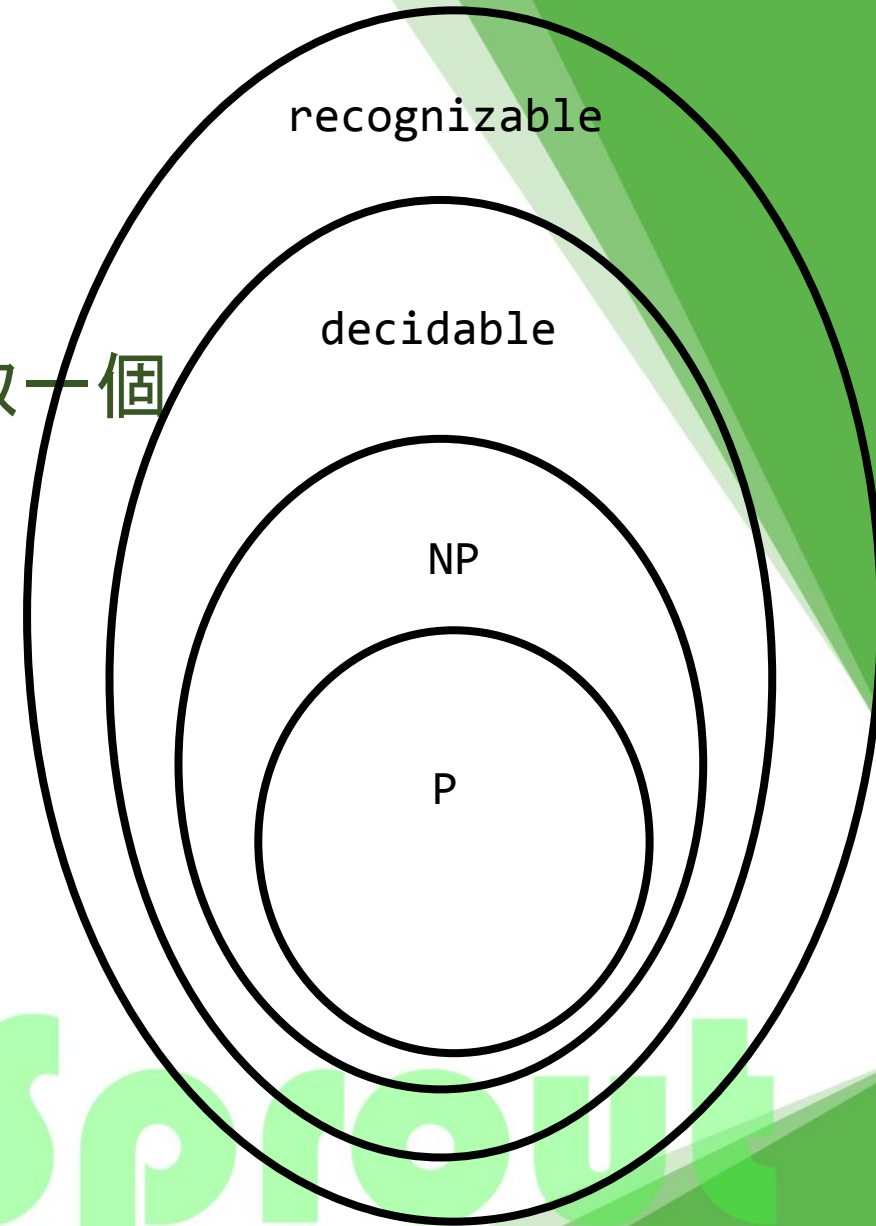


Sprout



## P 與 NP

- 所有 P 問題都是 NP 問題
- 我們可能會很想幫 NP 和 P 之間的區域取一個名字, 這樣當我們發現一個問題很難, 我可就可以試圖去證明它屬於那個區域

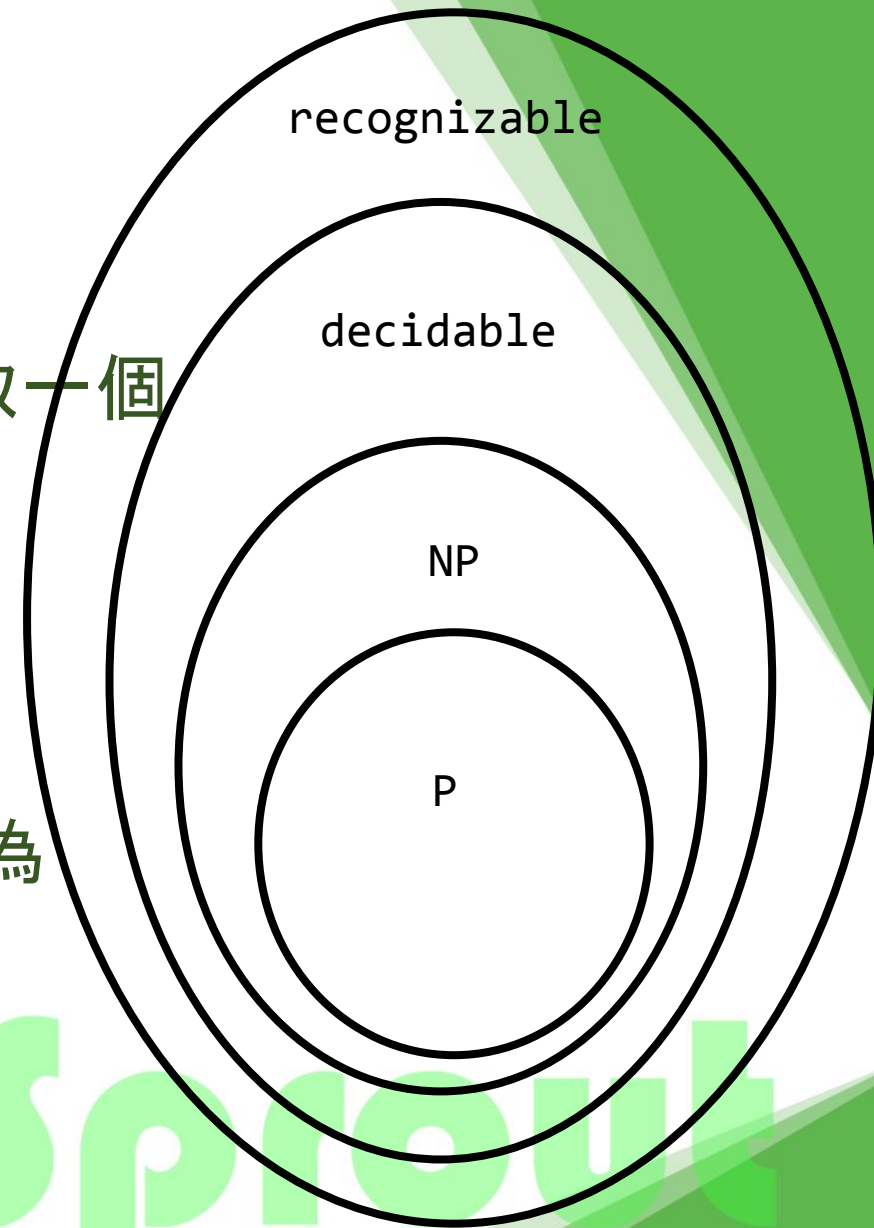


Sprout



## P 與 NP

- 所有 P 問題都是 NP 問題
- 我們可能會很想幫 NP 和 P 之間的區域取一個名字, 這樣當我們發現一個問題很難, 我  
可就可以試圖去證明它屬於那個區域
- 但目前我們不確定那個區域是否存在
- 也就是有可能  $P=NP$
- 於是我們定義 NP 問題裡最難的那些問題為  
NP-Complete, 如果有一天確定了  $P \neq NP$ ,  
那那些 NPC 的問題肯定不屬於 P



Sprout



## 多項式時間複雜度

- 剛剛 P 和 NP 都有提到的多項式時間，大家都知道多項式要有一個變數，那麼題目裡的變數有時候是指個數，有時候是數字範圍，有時候甚至超過一個變數，多項式時間是指哪個變數的多項式呢？
- 印出  $1+2+\dots+n$ ：
  - A. 使用公式解
  - B. 使用暴力法
- 上述哪些是多項式時間演算法，哪些不是呢？

Sprout



## 多項式時間複雜度

- 剛剛 P 和 NP 都有提到的多項式時間，大家都知道多項式要有一個變數，那麼題目裡的變數有時候是指個數，有時候是數字範圍，有時候甚至超過一個變數，多項式時間是指哪個變數的多項式呢
- A: 輸入總長度(字元數),  $N$ , 的多項式
- 因此當你的複雜度裡面有代表數字大小而非個數的變數時，其實不能算是多項式演算法，因為長度為  $N$  的輸入可以表示出約  $10^N$  大小的數字。
- 這種看似是多項式時間的演算法，我們會稱其為「偽多項式時間」演算法。在題目有限制數字範圍不會太大時依然很有效率。

Sprout



## 問題時間

- 為什麼我們剛剛都只討論 yes/no 問題？這樣適用於其他問題嗎？

Sprout





## P 問題的重要性

- 當我們在歸類一個問題為 P 問題時，等於不在乎他的複雜度是  $O(N^2)$  還是  $O(N^3)$  之類的，只要是多項式時間就好。
- 這是因為多項式次數的差別可以透過運算速度隨著科技進步的提升，或是更多更高級的運算資源來突破。但是非多項式時間的問題我們基本上就是永遠沒辦法有效率的解決。

Sprout



## P 問題的重要性

- 當我們在歸類一個問題為 P 問題時，等於不在乎他的複雜度是  $O(N^2)$  還是  $O(N^3)$  之類的，只要是多項式時間就好。
- 這是因為多項式次數的差別可以透過運算速度隨著科技進步的提升，或是更多更高級的運算資源來突破。但是非多項式時間的問題我們基本上就是永遠沒辦法有效率的解決。
- 存在無法多項式時間解決的問題並不是壞事，這可以製造 computational gap
  - 可以用在密碼學中
- 如果  $P=NP$  會發生什麼事？

Sprout



## $P = ? NP$

- If  $P = NP$ , then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett.

**Sprout**  
by Scott Aaronson