

# Algorithm Design and Analysis

Vinsong

September 17, 2025

## **Abstract**

The lecture note of 2025 Fall Algorithm Design and Analysis by professor 呂學一. 希望我可以活著度過這學期~~~~~

# Contents

<b>0</b>	<b>Introduction</b>	<b>2</b>
0.1	Design and Analysis . . . . .	2
0.2	Jargons . . . . .	4
<b>1</b>	<b>Complexity for a Problem</b>	<b>5</b>
1.1	函數成長率 (Rate of Growth) . . . . .	5
1.2	成長率的比較 . . . . .	5
1.3	Big Oh Notation . . . . .	6
1.4	Big-Oh 的運算 . . . . .	7
1.5	More Asymptotic Notation . . . . .	10

# Chapter 0

## Introduction

### Lecture 1

#### 0.1 Design and Analysis

4 Sep. 14:20

##### 0.1.1 Design

**Remark.** Find the point to cut into the problem.

**Question (Coffee and Milk).** 把 500 毫升的咖啡倒入 10 毫升，再從 510 毫升牛奶咖啡取 10 毫升倒入 490 毫升牛奶中，試問兩邊比例？

**Answer.** 兩邊都固定 500 毫升，一邊少的必定出現在另一邊，**切入點對了根本不用計算**  $\circledast$

##### 0.1.2 Analysis

**Question (Card).** 把牌洗亂（平均）需要幾次？

**Note.** 定義何為亂？

排列出現機率皆為

$$\frac{1}{52!}$$

七次是充分必要條件（嚴謹分析 on paper） $n$  card should shuffle  $\frac{3}{2} \log_2 n + \theta$  times.  $\circledast$

**Definition 0.1.1 (亂).** With  $n$ -cards, we have to let the probability of every combination become

$$\frac{1}{n!}$$

**Question (Top-in shuffle).** Consider Top-in shuffle with the cards. How to get it "randomly" ?

**Answer.** Define the  $k$ -th section to be 初始底牌從底下數上來是  $k$ -th card.

1. bottom  $k - 1$  cards must be 亂
2. 每次都可以用  $n/k$  次將他洗亂，因為出現機率皆為  $k/n$

We can shuffle  $n \cdot H_n$  times.

⊗

**Theorem 0.1.1.** 底下  $k - 1$  張卡片永遠是亂的

**Proof.** 考慮 top-in shuffle，利用數學歸納法

- 第一輪要插入底牌下方，只有 1 個空隙，因此必須插入，因此插入的機率是

$$\frac{1}{1!}$$

- 底下如果有  $k$  張牌，假設下面  $k$  張是亂的，表示他的排列  $k!$  種，每種順序機率都是

$$\frac{1}{k!}$$

- 再插入一張，共有  $k + 1$  個空隙，排起來每種順序出現的機率為

$$\frac{1}{(k+1)} \cdot \frac{1}{k!} = \frac{1}{(k+1)!}$$

符合亂的定義

■

第  $k$  階段插入到下面都是從  $n$  個空隙裡面找到  $k$  個空隙插入，因此出現機率必定為  $\frac{k}{n}$ ，因此需要 shuffle 次數為

$$\frac{n}{k}$$

接著考慮第  $n$  階段，底牌不是亂的，因此要再洗一次，因此最終的和為

$$\sum_{i=1}^n \frac{n}{i} = n \cdot \sum_{i=1}^n \frac{1}{i} = n \cdot H_n$$

**Note.** choose another card to be "bottom", 可以減少第一次的  $1/n$  就可以少  $n/1$  次 shuffle. 因此可以把次數減少為：

$$n \cdot H_n - n$$

**Remark.** 簡單的分析點交換就可以造成巨大的影響

## 0.2 Jargons

**Definition 0.2.1 (Problems).** 「問題」 (Problem) 是一個對應關係，就是一個函數

- 演算法核心是在探討問題的解決難易度
- 有些問題確定很難，就不用妄想想出簡單演算法

**Definition 0.2.2 (Instance).** 「個例」 (instance)，也就是問題的合法輸入

**Definition 0.2.3 (Computation Model).** 「計算模型」 (Computation Model)，也就是遊戲規則，同一個問題在不同的規則下可能難易度不同

- Comparison base & Computation base

**Definition 0.2.4 (Algorithm).** 「演算法」 Algorithm is a detail step-by-step instruction

- 符合規則
- 詳細步驟

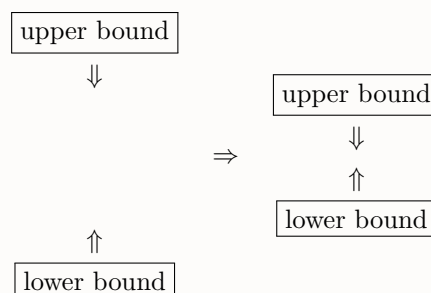
**Definition 0.2.5 (Hardness).** 「難度」 (Hardness)，想知道一個「問題」有多難解，用最厲害的一個「解法」，對於每個「個例」，都至少要用多少「工夫」才能解完

- 魔方問題：對於所有解法，存在至少一個初始 instance 讓解法需要 20 次才能轉完，切入點是找到一個固定的初始狀態，這是一個已經最佳化的問題

**Theorem 0.2.1 (Confirm Hardness).** 用 upper bound 和 lower bound 去夾起來決定難度

- 當 upper bound = lower bound 的時候，我們才知道問題的確切難度
- 有些情況，就算夾起來也不一定可以確定難度

**Proof.**



**Note.** 我們在這門課都討論 worst case

# Chapter 1

## Complexity for a Problem

### Lecture 2

#### 1.1 函數成長率 (Rate of Growth)

11 Sep. 13:20

**Question** (棋癡國王與文武大臣). 國王愛下棋，文武大臣要獎賞

- 武大臣每下一個棋子，獎賞多一袋米，起始為一袋米
- 文大臣每下一個棋子，獎賞雙倍，起始為一粒米

**Answer.** 棋盤 64 格

- 武大臣： $n$  袋米
- 文大臣： $2^n$  粒米

$2^n$  的成長率遠遠高於  $n$ ，單位的影響不及成長率

⊗

#### 1.2 成長率的比較

**Note.** 雖然 200 年前就有 Asymptotic Notation 的概念，但直到 1970 年代才被演算法分析之父 Donald Ervin Knuth 正式定義到 CS 領域內。

**Question** (Why Asymptotic Notation). 為什麼要用 Asymptotic Notation ?

**Answer.** 問題難度通常單位不一致

- $n = 3$  魔方問題要 20 轉
- $n$  個信封的老大問題要  $n - 1$  次比較

兩者難度無法比較

⊗

**Definition 1.2.1** (Rate of Growth). 沒有人有明確定義，但是成長率很好比較，有很多東西也是無法定義但可以比較，e.g. 無限集合可以比大小。

## 1.3 Big Oh Notation

**Definition 1.3.1** (Big Oh Notation). For functions  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ , we write

$$f(n) = O(g(n))$$

to satisfy the existence of positive constants  $c$  and  $n_0$  such that the inequality

$$0 \leq f(n) \leq c \cdot g(n)$$

holds for all integer  $n \geq n_0$ .

**Note.**  $f(n), g(n)$  should be non-negative for sufficiently large  $n$ .

The definition of

$$f(n) = O(g(n))$$

says that there exist a positive constant  $c$  such that the value of  $f(n)$  is upper-bounded by  $c \cdot g(n)$  for all sufficiently large positive  $n$ .

**Remark.** 因此  $O(g(n))$  可以理解成一個成長率不高過  $g$  的函數所成的集合

### 1.3.1 等號左邊也有 Big-Oh

**Definition 1.3.2.** The equality  $O(g(n)) = O(h(n))$  signifies that

$$f(n) = O(h(n))$$

holds for all functions  $f(n)$  with

$$f(n) = O(g(n))$$

i.e.  $O(g(n)) = O(h(n))$  signifies that  $f(n) = O(g(n))$  implies  $f(n) = O(h(n))$ .

The equality  $=$  in  $O(g(n)) = O(h(n))$  is more like  $\subseteq$ , i.e.,  $O(g(n)) \subseteq O(h(n))$ .

**Theorem 1.3.1.**  $O(g(n)) = O(h(n))$  if and only if  $g(n) = O(h(n))$ .

**Proof.** Consider the two directions separately.

- For the  $(\Rightarrow)$  case: We can easily proof that

$$g(n) = O(g(n))$$

then we can deduce that

$$g(n) = O(g(n)) = O(h(n))$$

- For the  $(\Leftarrow)$  case:



As previously seen (Definition 1.3.1).

$$g(n) = O(h(n)) \Rightarrow \exists c_1, n_1 > 0, \forall n \geq n_1, 0 \leq g(n) \leq c_1 \cdot h(n)$$

Let  $f$  be the function such that  $f(n) = O(g(n))$ . Then, by definition, we can deduce that

$$\exists c_2, n_2 > 0, \forall n \geq n_2, 0 \leq f(n) \leq c_2 \cdot g(n).$$

Assume  $n \geq \max\{n_1, n_2\}$ . Then, we have

$$0 \leq f(n) \leq c_2 \cdot g(n) \leq c_2 \cdot (c_1 \cdot h(n)) = (c_1 c_2) \cdot h(n).$$

Thus, we can conclude that

$$f(n) = O(g(n)) = O(h(n))$$

Hence,

$$O(g(n)) = O(h(n)) \Leftrightarrow g(n) = O(h(n)).$$

■

## 1.4 Big-Oh 的運算

**Question.** 所以，Big-Oh 相加的意思是什麼？

**Definition 1.4.1 (Big-Oh Addition).** The equality

$$O(g_1(n)) + O(g_2(n)) = O(h(n))$$

signifies that the equality

$$f_1(n) + f_2(n) = O(h(n))$$

holds for any functions  $f_1(n)$  and  $f_2(n)$  with

$$f_1(n) = O(g_1(n))$$

$$f_2(n) = O(g_2(n)).$$

That is,  $f_1(n) = O(g_1(n))$  and  $f_2(n) = O(g_2(n))$  together imply  $f_1(n) + f_2(n) = O(h(n))$ .

**Remark.** 雖然  $O(g_1(n)) + O(g_2(n))$  看起來像是兩個集合的聯集，但相同集合想法無法帶到減乘除。

**Definition 1.4.2 (Big-Oh  $\circ$ ).** The equality

$$O(g_1(n)) \circ O(g_2(n)) = O(h(n))$$

$$g_1(n) \circ O(g_2(n)) = O(h(n))$$

集合的複合操作

**Notation.**

$$\{f_1(n) \circ f_2(n) \mid f_1(n) \in S_1 \text{ and } f_2(n) \in S_2\}$$

可以被理解成

- 把  $=$  解成  $\subseteq$
- 把  $g_1(n)$  理解成  $\{g_1\}$
- $O(g_1(n))$  解為成長率不超過  $g_1$  的成長率的所有函數所組成的集合

**Remark.** 減乘除應被理解成與剛剛加法類似的模式，而無法被理解為集合的運算

**Definition 1.4.3** (Big-Oh  $-$ ,  $\cdot$ ,  $/$ ). (Take  $-$  as the example) The equality

$$O(g_1(n)) - O(g_2(n)) = O(h(n))$$

signifies the equality

$$f_1(n) - f_2(n) = O(h(n))$$

holds for any functions  $f_1(n)$  and  $f_2(n)$  with

$$f_1(n) = O(g_1(n))$$

$$f_2(n) = O(g_2(n))$$

**Question.** Proof or disproof:

$$O(n)^{O(\log_2 n)} = O(2^n)$$

**Answer.** First, we take log on both sides:

$$\text{LHS} = O(\log n) \cdot O(\log n) = (O(\log n))^2$$

$$\text{RHS} = O(n)$$

LHS grows slower than RHS, therefore the original statement is true. ⊛

**Remark.**  $\log$  的底數不影響成長率，因此可忽略。

**Definition 1.4.4** (Big-Oh 套 Big-Oh). The equality

$$O(O(g(n))) = O(h(n))$$

signifies that the equality

$$O(f(n)) = O(h(n))$$

holds for any function  $f$  with

$$f(n) = O(g(n))$$

i.e.  $f(n) = O(g(n))$  implies  $O(f(n)) = O(h(n))$ .

**Theorem 1.4.1.**  $g(n) = O(h(n))$  if and only if  $O(O(g(n))) = O(h(n))$

**Proof.** Consider the two directions separately.

- For the  $(\Rightarrow)$  case:

As previously seen (Definition 1.3.1).

$$g(n) = O(h(n)) \implies \exists c_0, n_0 > 0, \forall n \geq n_0, 0 \leq g(n) \leq c_0 \cdot h(n)$$

$f(n) = O(O(g(n)))$  signifies that for  $c_1, c_2, n_1, n_2 > 0$

$$\forall n \geq n_1, 0 \leq f(n) \leq c_2 \cdot u(n); \quad \forall n \geq n_2, 0 \leq u(n) \leq c_1 \cdot g(n)$$

Get all together, we have

$$0 \leq f(n) \leq c_2 \cdot (c_1 \cdot g(n)) \leq c_2 c_1 c_0 \cdot h(n) \implies f(n) = O(h(n))$$

Thus, we can conclude that

$$O(O(g(n))) = O(h(n))$$

- We can easily proof that

$$g(n) \subseteq O(g(n)) \subseteq O(O(g(n)))$$

Then we can get

$$g(n) = O(O(g(n))) = O(h(n))$$

Hence,  $g(n) = O(h(n)) \Leftrightarrow O(O(g(n))) = O(h(n))$  ■

**Theorem 1.4.2 (Rules of Computation in Big-Oh).** The following statements hold for functions  $f, g : \mathbb{N} \rightarrow \mathbb{R}$  such that there is a constant  $n_0$  such that  $f(n)$  and  $g(n)$  for any integer  $n \geq n_0$ :

- **Rule 1:**  $f(n) = O(f(n))$ .
- **Rule 2:** If  $c$  is a positive constant, then  $c \cdot f(n) = O(f(n))$ .
- **Rule 3:**  $f(n) = O(g(n))$  if and only if  $O(f(n)) = O(g(n))$ .
- **Rule 4:**  $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$ .
- **Rule 5:**  $O(f(n) \cdot g(n)) = f(n) \cdot O(g(n))$

**Proof.** For **Rule 5:** By the Definition 1.3.1,  $u(n) = O(f(n) \cdot g(n))$  signifies that there exist positive constants  $c_1$  and  $n_1$  such that the inequality

$$\exists c_0, n_0 > 0, \forall n \geq n_0, 0 \leq u(n) \leq c_0 \cdot f(n) \cdot g(n)$$

the definition of  $u(n) = f(n) \cdot O(g(n))$  is

$$\exists c_1, n_1 > 0, \forall n \geq n_1, 0 \leq u(n) \leq f(n) \cdot c_1 \cdot g(n)$$

which are equivalence to each other. ■

## 1.5 More Asymptotic Notation

**Definition 1.5.1 (Little-oh).** For any function  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ , we write

$$f(n) = o(g(n))$$

to signify that for any constant  $c > 0$ , there is a positive constant  $n_0(c)$  such that

$$0 \leq f(n) < c \cdot g(n)$$

holds for each integer  $n \geq n_0(c)$

**Note.**  $n_0(c)$  is a function of  $c$ . When we  $n_0(c)$  is a constant, we means that it does not depend on  $n$ .

白話來說  $f(n) = o(g(n))$  的定義是說，不管是多小的常數  $c$ ，要  $n$  夠大 (i.e.,  $n \geq n_0(c)$ )，

$$0 \leq f(n) < c \cdot g(n)$$

都還是成立。

**Example.**

$$n = o(n^2)$$

Observe that for any positive constant  $c$ , as long as  $n > \frac{1}{c}$ , we have

$$0 \leq n < c \cdot n^2$$

Therefore, we may let  $n_0(c) = \frac{1}{c} + 1$  and have  $n = o(n^2)$  proved.

**Definition 1.5.2 (Other notation).** The other notation can be defined via  $O$  and  $o$  notation:

- We write  $f(n) = \Omega(g(n))$  if

$$g(n) = O(f(n)).$$

- We write  $f(n) = \Theta(g(n))$  if

$$f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

- We write  $f(n) = \omega(g(n))$  if

$$g(n) = o(n)$$

Limit notation 可以幫我們判斷各種 Asymptotic Notation:

- If

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

, the we can guess  $f(n) = o(g(n))$ .

- If

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$$

, the we can guess  $f(n) = \Theta(g(n))$ .

- If

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

, then we can guess  $f(n) = \omega(g(n))$ .

然而，極限不一定應可以推至 Asymptotic Notation:

- Let  $f(n) = g(n) = (-1)^n$ . We have

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1,$$

but  $f(n) \neq O(g(n))$ ,  $f(n) \neq \Omega(g(n))$ , and  $f(n) \neq \Theta(g(n))$ .

- Let  $f(n) = (-1)^n$  and  $g(n) = n \cdot (-1)^n$ . We have

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0,$$

but  $f(n) \neq o(g(n))$ .

- Let  $f(n) = 2 + (-1)^n$  and  $g(n) = 2 - (-1)^n$ . We have

$$f(n) = \Theta(g(n)),$$

but  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$  does not exist.

**Question.** Can we just use  $\leq$  instead of  $<$  in the definition of  $o$  ?

**Answer.** In most part of it will be right. However there will be a special situation:

$$o(0) = 0$$

which is definitely wrong. ⊗

**Question.** 為何不都用  $\exists c_0, n_0$  或都用  $\forall c, n_0(c)$  ?

**Answer.** 如果都用  $\exists c_0, n_0$ ，那  $o$  就會退化，變成  $O$  而已，並且  $<$ ,  $\leq$  是沒有太大差別的

**Proof.** Suppose that  $\hat{n}_0(c)$  is the constant ensured by the  $\leq$ -version. We simply let

$$n_0(c) = \max(m_0, \hat{n}_0(c/2)).$$

As a result, for any positive constant  $c$ , if  $n \geq n_0(c)$ , we have  $g(n) > 0$  and thus

$$\begin{aligned} 0 < f(n) &\leq \frac{c}{2} \cdot g(n) \\ &< c \cdot g(n). \end{aligned}$$

■

證畢，由此可知符號並無太大影響，不可讓  $o$  退化 ⊗