

# 基礎競賽概論

temmie

1. 賽制介紹

2. 複雜度分析

3. 雜項

## 賽制介紹

- 高中比賽常出現的賽制
- 有部份分（子任務）
- 子任務用來引導參賽者

- 大學比賽常出現的賽制
- 沒有部份分，比團隊的 AC 數量

- 大學比賽常出現的賽制
- 沒有部份分，比團隊的 AC 數量
- 同 AC 數量則以罰時計算
- 罰時的計算與解題時間和 WA 數量有關

# 各種比賽

- APCS（每年三次）
- YTP（7 ~ 8 月）
- 學科能力競賽（9 ~ 12 月）
- NPSC（11 ~ 12 月）

# 複雜度分析



# 複雜度的意義

- 用來客觀衡量一個演算法的標準，包括時間和空間
- 通常用資料的數量當作參數，以函數形式呈現
- 考慮最差的情況

# 為什麼我們需要了解複雜度

- 可以快速判斷你的程式會不會卡到 TLE、MLE
- 比較不同演算法優劣
- 可以透過資料數量猜測預期解法

# 時間複雜度的例子

## 圖書館

身後有很多書架，你可以用多快的速度找到書？

# 時間複雜度的例子

## 圖書館

身後有很多書架，你可以用多快的速度找到書？

- 我想大部分的人可能是從左到右，從上到下找
- 如果有  $n$  本書，則你最差需要找  $n$  次

# 時間複雜度的例子

## 圖書館

身後有很多書架，你可以用多快的速度找到書？

- 我想大部分的人可能是從左到右，從上到下找
- 如果有  $n$  本書，則你最差需要找  $n$  次
- 你知道嗎，在某些特殊的情況下，可以在大概  $\log_2 n$  次甚至是一次就找到！

# 複雜度的表示法

- 通常會用  $O(\dots)$  來表示
- 上面的符號叫做 Big-O，用來表示量級的一種趨近上界

# 複雜度的表示法

- 通常會用  $O(\dots)$  來表示
- 上面的符號叫做 Big-O，用來表示量級的一種趨近上界
- 通常裡面只會丟一種指數量級的參數，例如： $O(n)$ 、 $O(n^2)$
- 如果有較低量級的參數、係數則省略，例如： $O(3n^2 + 2n + 17)$ ，則表示成  $O(n^2)$

# 時間複雜度中的常數

- 只要是固定操作數量的指令都會計成  $O(1)$
- 5 次？10 次？10000 次？

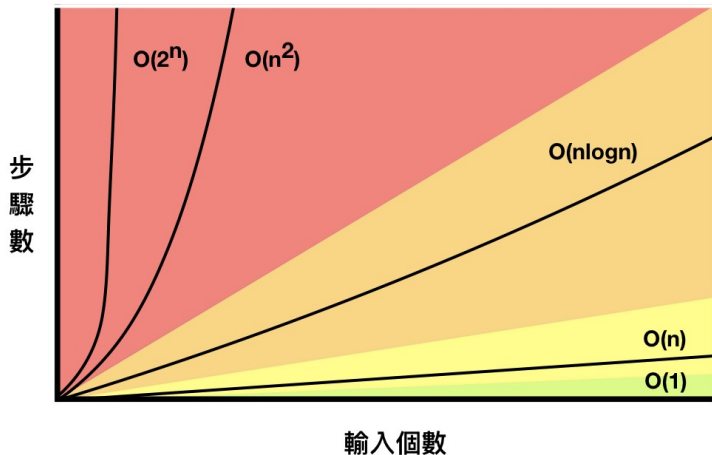


# 時間複雜度中的常數

- 只要是固定操作數量的指令都會計成  $O(1)$
- 5 次？10 次？10000 次？
- 這種東西我們稱作**常數**
- 請注意，即使在時間複雜度上被省略，仍然很重要

# 複雜度的量級差異

常見時間複雜度比較



# 複雜度的量級差異

以下為參考的時間複雜度對照量級，估算為約  $10^8$  操作為一秒

- $O(2^n) : n \approx 25$
- $O(n^2) : n \approx 10^4$
- $O(n \log n) : n \approx 5 \times 10^5$
- $O(n) : n \approx 10^8$
- $O(\log n) : n \approx 10^{18}$

# 複雜度的量級差異

以下為參考的時間複雜度對照量級，估算為約  $10^8$  操作為一秒

- $O(2^n) : n \approx 25$
  - $O(n^2) : n \approx 10^4$
  - $O(n \log n) : n \approx 5 \times 10^5$
  - $O(n) : n \approx 10^8$
  - $O(\log n) : n \approx 10^{18}$
- 
- 背不起來？經驗多就知道了！

# 判斷時間複雜度

## 判斷時間複雜度的方法

- 計算 for 迴圈的層數
- 用規律去找
- 透過經驗 / 背公式

# 例題

## 複雜度分析練習-1

請試著分析下面程式碼的時間複雜度

```
for (int i=0 ; i<n ; i++){  
    for (int j=0 ; j<n ; j++){  
        // O(1) 操作  
    }  
}
```

# 例題

## 複雜度分析練習-1

請試著分析下面程式碼的時間複雜度

```
for (int i=0 ; i<n ; i++){  
    for (int j=0 ; j<n ; j++){  
        // O(1) 操作  
    }  
}
```

- 我們可以發先第一個 for 迴圈重複  $n$  次，並且第二個也重複  $n$  次
- 答案為  $O(n^2)$

# 例題

## 複雜度分析練習-2

請試著分析下面程式碼的時間複雜度

```
void f(int n){  
    if (n==1){  
        // O(1) 操作  
    }else{  
        f(n-1);  
        f(n-1);  
    }  
}
```



# 例題

## 複雜度分析練習-2

請試著分析下面程式碼的時間複雜度

```
void f(int n){  
    if (n==1){  
        // O(1) 操作  
    }else{  
        f(n-1);  
        f(n-1);  
    }  
}
```

- 每次遞迴都會增加兩倍的數字，實際上會有  $2^{n-1}$  次操作
- 答案為  $O(2^n)$

- 回家把這些影片看完，練習看看裡面的題目吧:D
- AA 競程 Level 1 公開課 713

## 雜項

# 競程常見數學符號

- $\sum_{i=1}^n i$ : 連加符號
- $\prod_{i=1}^n i$ : 連乘符號

# 競程常見數學符號

- $\sum_{i=1}^n i$  : 連加符號
- $\prod_{i=1}^n i$  : 連乘符號
- $\lceil n \rceil$  : 無條件進位
- $\lfloor n \rfloor$  : 無條件捨去

# 競程常見數學符號

- $a \bmod b$  :  $a \div b$  的餘數
- $a \equiv b \pmod{m}$  :  $a \bmod m = b \bmod m$
- $a \mid b$  :  $a$  整除  $b$

# 萬用標頭檔

- 只要在使用 `#include<bits/stdc++.h>` 就可以引入 95% 會用到的標頭檔

- 不想學指標？把所有東西都丟進全域吧
- 可以開更大的陣列
- 可以自動初始化
- (還是要學會指標啦)



- 先試試看這題吧

- 先試試看這題吧
- 如此簡單的題目怎麼過不了？

- 先試試看這題吧
- 如此簡單的題目怎麼過不了？
- 這是因為輸入、輸出太多導致超時
- 可以透過加上 `cin.tie(0)`、`ios::sync_with_stdio(0)` 和不使用 `endl` 改用 `'\n'` 加速

- 先試試看這題吧
- 如此簡單的題目怎麼過不了？
- 這是因為輸入、輸出太多導致超時
- 可以透過加上 `cin.tie(0)`、`ios::sync_with_stdio(0)` 和不使用 `endl` 改用 `'\n'` 加速
- 想了解原因？私訊我吧

- 總是因為沒開 `long long` 而 WA 感到很煩躁？

# #define

- 總是因為沒開 long long 而 WA 感到很煩躁？
- 我們可以用 #define 這個函式把所有 int 定義成 long long

# #define

- 總是因為沒開 long long 而 WA 感到很煩躁？
- 我們可以用 #define 這個函式把所有 int 定義成 long long
- 請記得將原先的 main 改成 signed main(void)，否則無法使用

# 函式改值

- 丟進函式裡面的值沒有辦法被更改，但又不想用很毒的全域變數？
- 你可以在宣告變數的時候加上 `&`，這樣就可以在函式裡面改值囉

```
void f(int a){
    a=a+1;
}

int main(){
    int a=1;
    f(a);
    cout << a << '\n'; // 輸出:1
    return 0;
}
```

```
void f(int &a){
    a=a+1;
}

int main(){
    int a=1;
    f(a);
    cout << a << '\n'; // 輸出:2
    return 0;
}
```