DATA STRUCTURE

Lecture 01: Course Introduction, administration, etc.

Spring 2023

Hsiao-chi Li 李曉祺

ADMINISTRATIVE STUFF

exercise

2 hr / 1 hr

- Who's here? Where? And When?
  - 電機一乙 共同312 at 13:10 – 16:00
- Contact Information
  - 1. Email: hcli@mail.ntut.edu.tw
  - 2. Office: 綜科 518
  - 3. Teaching Assistant –
    - 賴盈翔 herrylai0914@gmail.com
    - 蘇柏凱 winston98321@gmail.com
- Office Hour: Mon. & Tue. 10:00 – 12:00, Wed. 08:00 – 10:00

- Cell Phones should be silenced or turned off
- Texting and web surfing are never appropriate

GRADING, ETC.

- Grading: documentation
  - Homework 15% (Programming in C)
  - Quiz 10%
  - Midterm 30%
  - Final 35%
  - Attendance 10%

- Class Policy:
  - Cooperative learning is wonderful and encouraged!
  - Cooperative testing is terrible and not encouraged!

GRADING, ETC. (CONT'D)

- Grading Policy
  - Late Submission: You can turn in your assignments late for no more than five days. There will be no penalty for your late submission within these three days. Once you used up the 5-day extension, your submission will be counted for no credits.

  - Example: If there are four assignments in this semester, and for these four you turn in your works
    - HW1 – 2 day later
    - HW2 – 3 day later (2+3=5 → no more extension for later assignments)
    - HW3 – on time
    - HW4 – 1 day later (not counted)

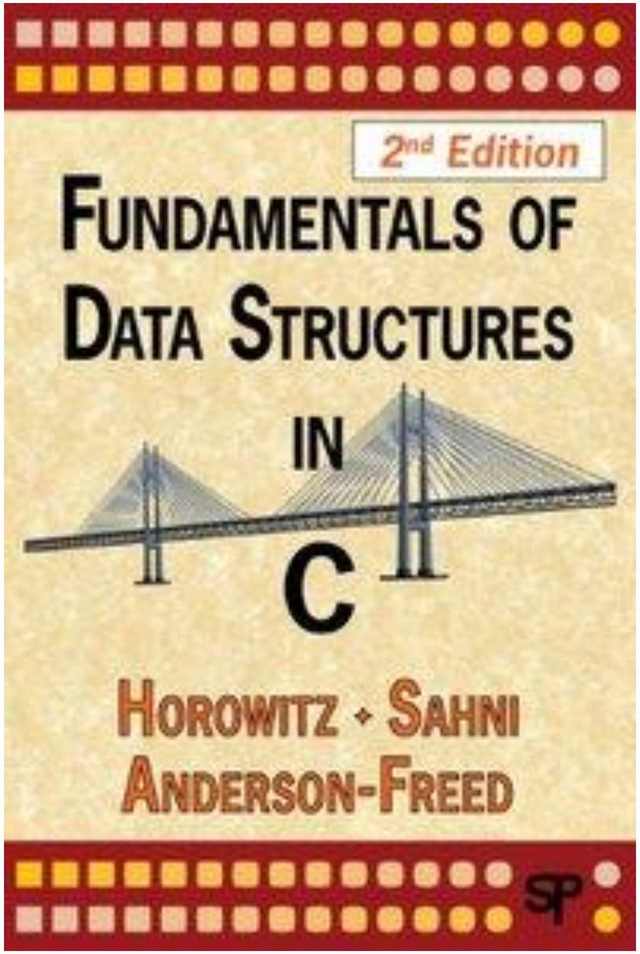
    No credits will be counted for the last assignment.

GRADING, ETC. (CONT'D)

- Policy on Copying
  - Copying is not acceptable. Do your own work.
- Attendance 10% maxima rollcall x 5
  - Absence: 2 points of your final grade will be taking off.

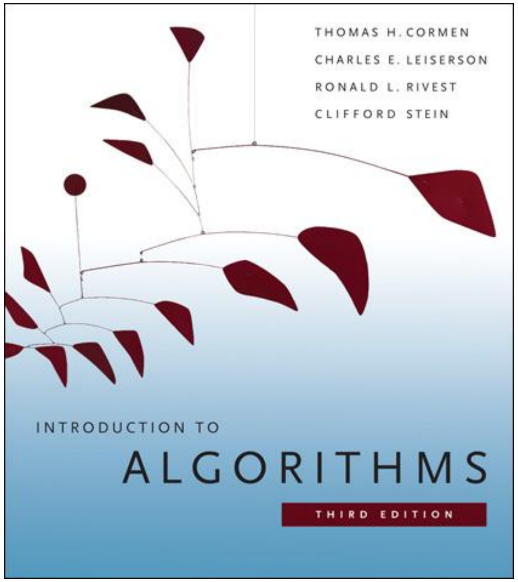
TEXT

- Horowitz, *Fundamentals of Data Structures in C*, 2nd edition, Silicon Press



REFERENCE

- Horowitz, Ellis, S. Sahni, and S. Rajasekaran, *Computer Algorithms / C++*. Summit, NJ: Silicon Press, 2007.
- M.T, Goodrich and R. Tamassia, *Algorithm Design*, John Wiley & Sons.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd Edition, The MIT Press, 2009.



COURSE PLAN (1)

教室一樣，考試時間1:10~3:00，
攜帶有照片知證件

不會有填空題。
會考計算時間複雜度，array的基本概念、操作、ADT、是非題。
還會給一段程式碼，問最後輸出的值。

4/24 手寫only

電機一乙

期中考範圍

週次 Week	日期 Date	主題 Topic
1	02/20	Introduction & Overview
2	02/27	和平紀念日
3	03/06	Algorithms: Analysis, complexity, and the lower bound problem
4	03/13	
5	03/20	Stacks, Queues, Trees, Dictionaries
6	03/27	
7	04/03	兒童節
8	04/10	Graphs
9	04/17	Midterm

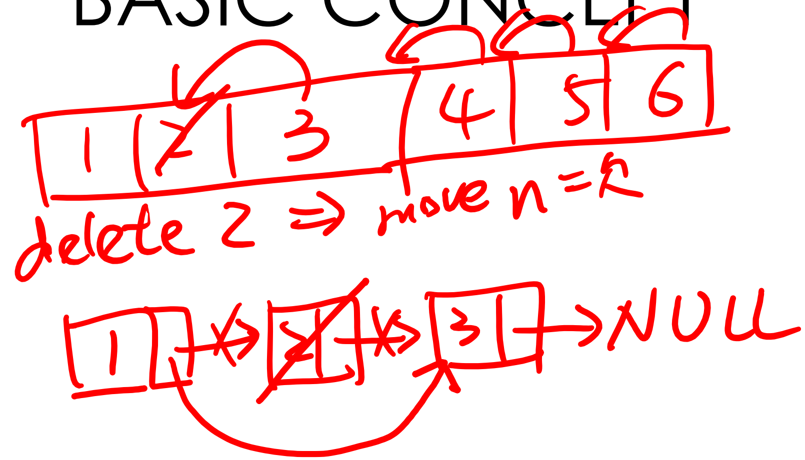
COURSE PLAN (2)

電機一乙

週次 Week	日期 Date	主題 Topic
10	04/24	Heaps, Sets
11	05/01	Sorting
12	05/08	
13	05/15	The Greedy Method; The Divide-and-Conquer Strategy
14	05/22	Tree Searching Strategies; Prune and Search
15	05/29	
16	06/05	Shortest Paths
17	06/12	
18	06/19	Final

- Rough plan (Subject to change!)

BASIC CONCEPT



- What Is Data Structure?
- What Is Algorithm?

- Overview
  - System Life Cycle
  - Data Abstraction and Encapsulation

資料虛擬化

  - Algorithm Specification
  - Performance Analysis and Measurement

SYSTEM LIFE CYCLE

- Requirements: Input and Output

- Analysis: To construct a building
  - Bottom-up
  - Top-down

- Design
  - Data objects: abstract data types
  - Operations: specification & design of algorithms

SYSTEM LIFE CYCLE (CONT'D)

- Refinement & Coding
  - Choose representations for data objects
  - Write algorithms for each operation on data objects

- Verification
  - Correctness proofs: selecting proved algorithms
  - Testing: correctness & efficiency
  - Error removal: well-documented

EVALUATIVE JUDGEMENTS ABOUT PROGRAMS

- Meet the Original Specification?
- Work Correctly?
- Document?
- Use Functions to Create Logical Units?
- Code readable?
- Use Storage Efficiently?
- Running Time Acceptable?

ROLE OF DATA STRUCTURES

- Real problem:
  - Order the students according to their heights
  - Rank the students according to their scores

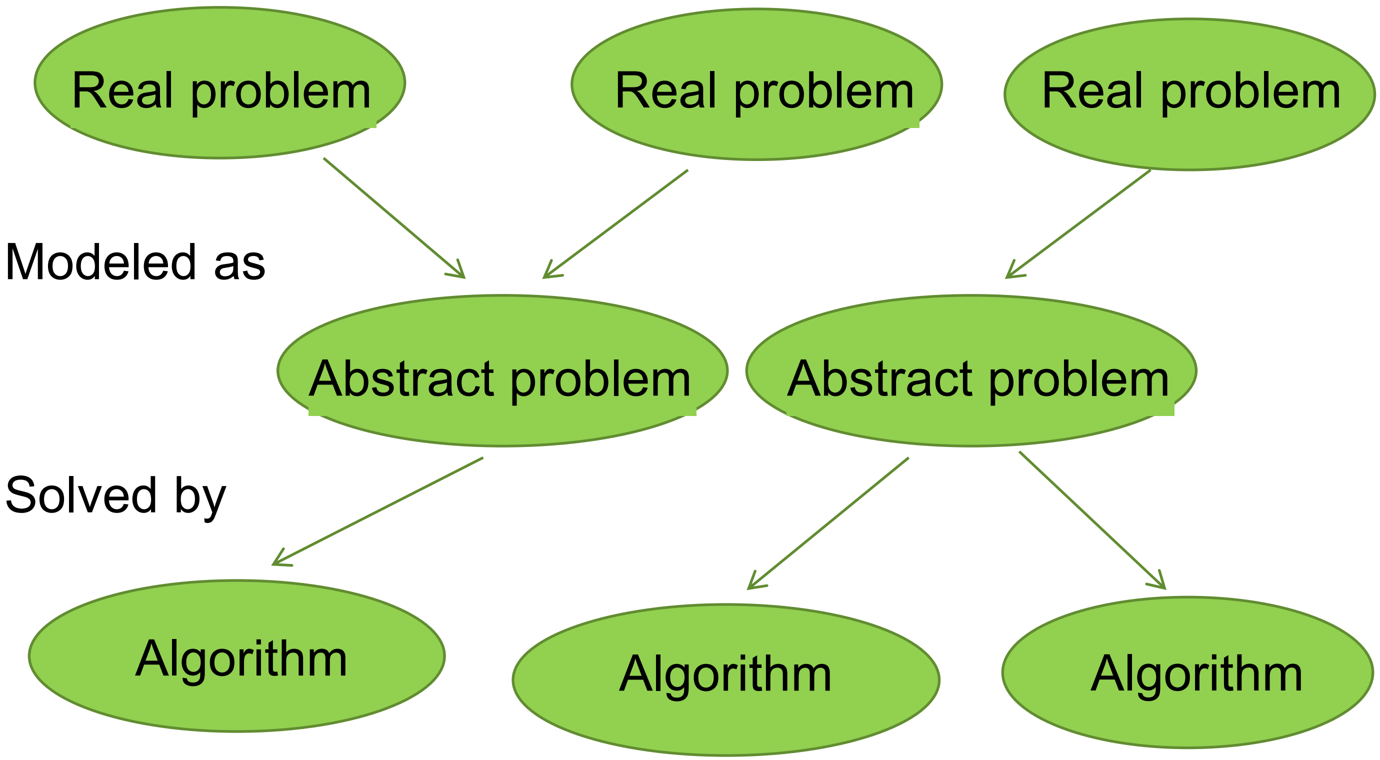
- Abstract problem:
  - Sorting problem

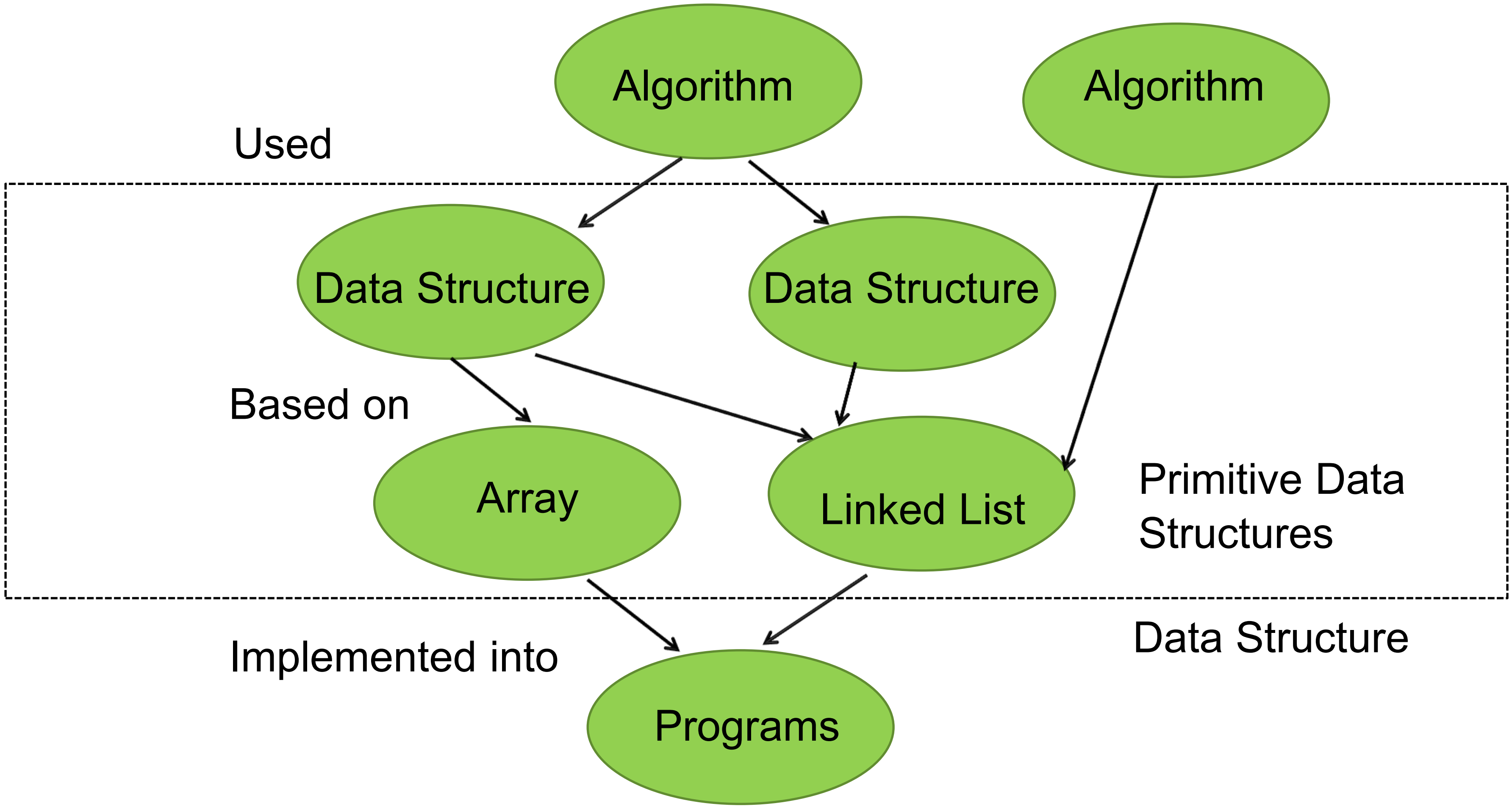
ROLE OF DATA STRUCTURES (CONT'D)

- Algorithm:
  - Bubble sort
  - Quick sort

- Data structure
  - Array
  - Linked list

ROLE OF DATA STRUCTURES (CONT'D)




Algorithm
Algorithm
Used
Data Structure
Data Structure
Based on
Array
Linked List
Primitive Data Structures
Data Structure
Implemented into
Programs

BASIC CONCEPT

- What Is Data Structure?
- What Is Algorithm?

學習資料結構的最大用意在於，了解各抽象化之後的名詞（各種資料結構）所代表的實際意義。
例如：二元樹是一種資料結構，但如果你沒有相關背景知識，就無法了解這些抽象化的名詞。

- Overview
  - System Life Cycle
  - Data Abstraction and Encapsulation 資料抽象與封裝
  - Algorithm Specification
  - Performance Analysis and Measurement

DATA ABSTRACTION AND ENCAPSULATION

- Data encapsulation or information hiding 資料封裝/資訊隱藏
  - is the concealing of the implementation details of a data object from the outside world
  - Example: DVD player
    - Buttons: PLAY, STOP and PAUSE

資料抽象化：抽象化的目的是為了溝通方便，用大家都可以理解的方式去形容，而非描述實際細節上是如何運作。

- Data abstraction
  - is the separation between the specification of a data object and its implementation
  - Example: DVD player
    - How to know the action when the PLAY button is presses.

例如我們不會說我們要搭乘一個包含座椅、引擎、起飛降落逃生的工具前往巴黎，我們會說我們要搭飛機。這就是一種資料抽象化。

- A data type
  - is a collection of objects and a set of operations that act on those objects
  - Example: C
    - char, int, float and double.
    - Short, long, signed and unsigned.

DATA ABSTRACTION

- Specification
  - Name of function
  - Type of arguments
  - Type of result
  - Description of what the function does

Predefined data types
*Struct student { char last_name
int student_id
char grade; }

Data type: object & operation
integer: +,-,,/,%,=,==

- Representation
  - Implementation details
  - e.g. char 1 byte, int 4 bytes

DATA ABSTRACTION

- A abstract data type (ADT)
  - is a data type that
  - is organized in such a way that the specification of the objects, and the specification of the operations on the objects is separated from the representation of the objects and the implementation of the operations
  - Example: C
    - The arithmetic operators: +, -, *, and /.
  - ADT is implementation-independent.

Abstract data type NaturalNumber (p.9)

```
ADT NaturalNumber is
  objects: an ordered subrange of the integers starting at
    zero and ending at the maximum integer (INT_MAX) on
    the computer
  functions:
    for all x, y ∈ Nat_Number; TRUE, FALSE ∈ Boolean
    and where +, -, <, and == are the usual integer
    operations.
    Zero ( ):NaturalNumber          ::=  0
    Is_Zero(x):Boolean              ::= if (x) return FALSE
                                        else return TRUE
    Add(x, y):NaturalNumber         ::= if ((x+y) <= INT_MAX)
                                          return x+y
                                        else return INT_MAX
    Equal(x,y):Boolean              ::= if (x== y) return TRUE
                                        else return FALSE
    Successor(x):NaturalNumber      ::= if (x == INT_MAX)
                                          return x
                                        else return x+1
    Subtract(x,y):NaturalNumber     ::= if (x<y) return 0
                                        else return x-y
end Natural_Number
```

DATA ABSTRACTION AND ENCAPSULATION (CONT'D)

- The advantage of data abstraction and data encapsulation:
  - Simplification of software development
  - Testing and Debugging
  - Reusability
  - Modifications to the representation of a data type

BASIC CONCEPT

- What Is Data Structure?
- What Is Algorithm?

- Overview
  - System Life Cycle
  - Data Abstraction and Encapsulation
  - Algorithm Specification
  - Performance Analysis and Measurement

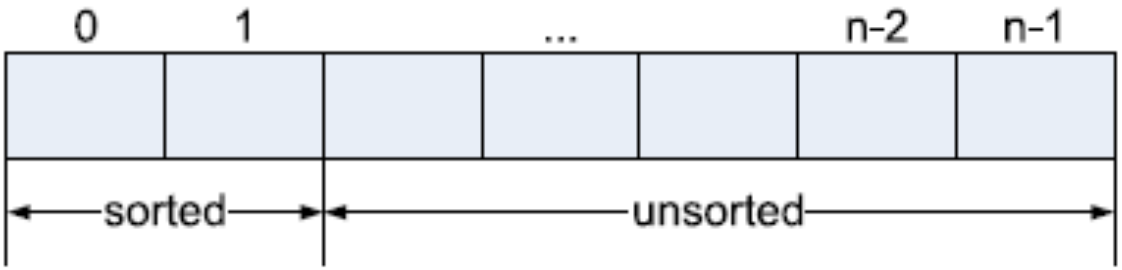
ALGORITHM SPECIFICATION

- Definition
  - An algorithm is a finite set of instructions that accomplishes a particular task.

- Criteria
  - Input
  - Output
  - Definiteness: clear and unambiguous
  - Finiteness: terminate after a finite number of steps
  - Effectiveness: instruction is basic enough to be carried out

3/6 (一)

EXAMPLE 1: SELECTION SORT



- From those integers that are currently unsorted, find the smallest and place it next in the sorted list.

```
for ( i=0; i<n; i++) {
  Examine list[i] to list[n-1] and suppose
    that smallest integer is list[min]
  Interchange list[i] & list[min]
}
```

EXAMPLE 1: SELECTION SORT (CONT.)

```
void sort(int list[ ], int n)
{
  for (i=0; i<n-1; i++)
  {
    int min = i;  假設最小值
    for (j=i+1; j<n; j++)
      if (list[j]<list[min])  如果比 min 小, 就把 min 取代掉, 設成 j
        min=j;
      SWAP(list[i], list[min], temp);
  }
}
```

有6筆資料要排序，要做5次

EXAMPLE OF SELECTION SORT

- Input: 20, 10, 15, 6, 17, 30

min = ~~0~~ ~~1~~ 3

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
Original	20	10	15	6	17	30
i=0 Pass 0	6	10	15	20	17	30
i=1 Pass 1	6	10	15	20	17	30
Pass 2	6	10	15	20	17	30
Pass 3	6	10	15	17	20	30
Pass 4	6	10	15	17	20	30

unsorted

unsorted

sorted

A[min]=6

A[0] ↔ A[min]

SWAP

n-1 次

二元搜尋.

EXAMPLE 2: BINARY SEARCH

找數字, or 找某筆資料

→ 針對 sorted array 去找

index :	0	1	2	3	4	n-2 (5)	n-1 (6)
	3	5	8	9	12	13	14
	left			middle			right

```
while (there are more integers to check)
{
          3 index 索引    0        6
  middle = (left + right) /2;      先切一半, 看要找的數字比較大還小
              5             list[3] = 9
  if (searchnum < list[middle])
    right = middle -1;   → 把一半再切一半, 所以 right 變成 middle -1
  else if (searchnum == list[middle])
    return middle;
  else
    left = middle+1;
}
```

EXAMPLE 2: BINARY SEARCH (CONT.)

```
int compare(int x, int y)
/* return -1 for less than, 0 for equal */
int binsearch(int list[], int searchno, int left,
    int right)
{
  while (left <= right)  {
    middle = (left + right) / 2;
    switch ( COMPARE(list[middle], searchno) )  {
      case -1:
        left = middle +1;
        break;
      case 0:
        return middle;
      case 1:
        right = middle -1;
    }
  }
}
```

EXAMPLE OF BINARY SEARCH

index 0 1 2 3 4 5 6 7 8

- Input: 1,4,7,10,12,13,17,23,32

① left = 0, right = 8, middle = 4

$10 < A[4] = 12 \Rightarrow right = middle - 1 = 3$

$middle = \frac{0+3}{2} = 1$ (type 是 int，故為 1)

$10 > A[1] = 4 \Rightarrow left = middle + 1 = 2$

$middle = \frac{(left + right)}{2} = \frac{2+3}{2} = 2$

$10 > A[2] = 7 \Rightarrow left = middle + 1 = 3$

$middle = \frac{3+3}{2} = 3$

$A[3] = 10$

- ① Search for 10
- Search for 15

EXAMPLE 3: SELECTION PROBLEM

- Selection problem: select the k^{th} largest among N numbers

  - Approach 1 要比 $\frac{n(n+1)}{2}$
    - Read N numbers into an array → 一次讀 N 個數字
    - Sort the array in decreasing order
    - Return the element in position k

EXAMPLE 3: SELECTION PROBLEM (CONT.)

只需要比 $3(n-3)$ 次

- Approach 2
  - Read *k* elements into an array → 一次只讀k個數字
  - Sort them in decreasing order → 從大排到小
  - For each remaining elements, read one by one
  - Ignored if it is smaller than the k^{th} element
  - Otherwise, place in correct place and bumping one out of array
- Which is better?
- Efficiency?

EXAMPLE OF SELECTION PROBLEM

- Input: 20, 9, 15, 6, 17, 30

- Find the third largest number

遞迴

RECURSIVE ALGORITHMS

Iteractive

- Recursion is usually used to solve a problem in a *divided-and-conquer* manner
- Direct Recursion 直接遞迴
  - Functions that call themselves
- Indirect Recursion 間接遞迴
  - Functions that call other functions that invoke calling function again
- $C\binom{n}{m} = \frac{n!}{m!(n-m)!}$ Binomial coefficient
  - $C\binom{n}{m} = C\binom{n-1}{m} + C\binom{n-1}{m-1}$
- Boundary condition for recursion

程式可以變精簡，但佔空間，也會多花時間

什麼時候可以寫recursive?

當一個程式是不斷重複的時候就可以寫成遞迴。

factorial (n) = n!

一定會有終止條件

Stop criterion

```
int S ; // sum

 s=0;

for ( i=1 ; i<=n, i++ ){
     s=s+i ;
 }
```

```
int rsum(n){
 if n=0  return 0;
  return  rsum(n-1)+n;
```

RECURSIVE SUMMATION

- sum(1,*n*) = sum(1,*n*-1)+*n*
- sum(1,1) = 1

1+2+3

遞迴的寫法

```
int sum(int n)
{
   if (n==1)
      return (1);
   else
      return(sum(n-1)+n);
}
```

3

boundary condition

3代入

$\boxed{\text{sum (2)}}$ +n

sum(2)

2≠1

$\boxed{\text{sum(1)}}$ + 2 = 3

1代入

sum(1)

1=1

return 1

RECURSIVE FACTORIAL

- $n! = n\ (n\text{-}1)!$
- factorial(n) = n×factorial(n-1)
- 0! = 1

```
int fact(int n)
{
  if ( n== 0)
    return (1);
  else
  return (n*fact(n-1));
}
```

RECURSIVE MULTIPLICATION

乘法可以用遞迴寫

- $a \times b = a \times (b-1) + a$

```
int mult(int a, int b)
{
   if ( b==1)
      return (a);
   else
      return(mult(a,b-1)+a);
}
```

BINARY SEARCH

```
int compare(int x, int y)
/* return -1 for less than, 0 for equal */
int binsearch(int list[], int searchno, int left,
    int right)
{
  while (left <= right)  {
    middle = (left + right) / 2;
    switch ( COMPARE(list[middle], searchno) )  {
      case -1:
        left = middle +1;
        break;
      case 0:
        return middle;
      case 1:
        right = middle -1;
    }
  }
}
```

RECURSIVE BINARY SEARCH

```
int binsearch(int list[], int searchno, int left,
    int right)
{
  if (left <= right) {
    middle = (left + right)/2;
    switch (COMPARE(list[middle], searchno) ) {
      case -1:
        return binsearch(list, searchno, middle+1,
                right)
      case 0:
        return middle;
      case 1:
        return binsearch(list, searchno, left,
                middle-1);
    }
  }
  return -1;
}
```

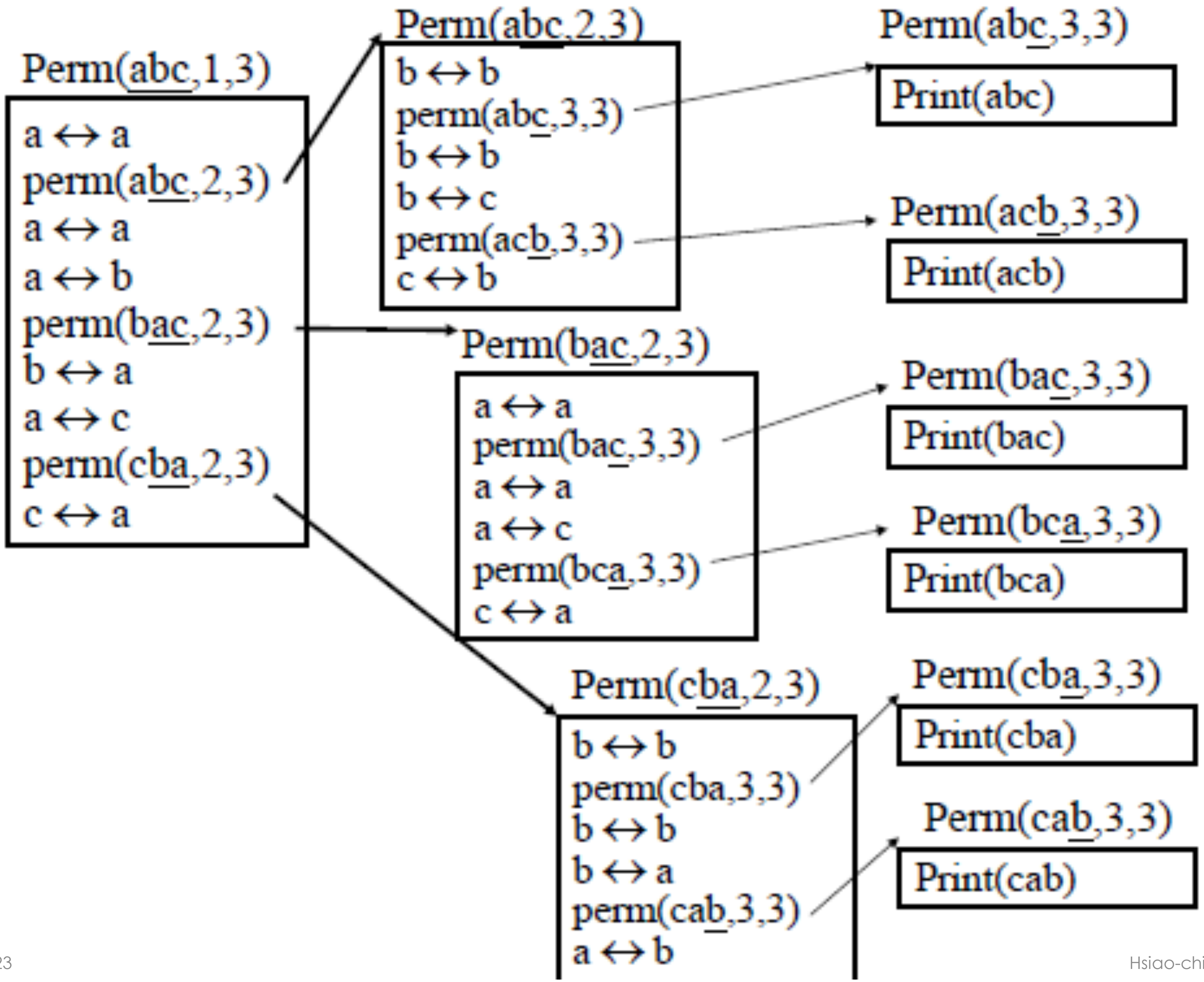
3/13 說

RECURSIVE PERMUTATION

- Permutation of {a, b, c}
  - (a, b, c), (a, c, b)
  - (b, a, c), (b, c, a)
  - (c, a, b), (c, b, a)
- Recursion?
  - a+Perm({b,c})
  - b+Perm({a,c})
  - c+Perm({a,b})

code parsing/pracing

3/13 說



3/13 說

RECURSIVE PERMUTATION (CONT'D.)

```
void perm(char *list, int i, int n)
{
   if (i==n) {
      for (j=0; j<=n; j++)
         printf("%c", list[j]);
   }
   else {
      for (j=i; j<= n; j++) {
         SWAP(list[i], list[j], temp);
         perm(list, i+1, n);
         SWAP(list[i], list[j], temp);
      }
   }
}
```

上半部在做初始化

BASIC CONCEPT

- What Is Data Structure?
- What Is Algorithm?

- Overview
  - System Life Cycle
  - Data Abstraction and Encapsulation
  - Algorithm Specification
  - Performance Analysis and Measurement

PERFORMANCE EVALUATION

- Criteria
  - Is it correct?
  - Is it readable?
- Performance analysis
  - Machine Independent

在評斷演算法好壞時，不可以用機器效能來評斷.

要在同一個環境下評斷.

- Performance measurement
  - Machine dependent

PERFORMANCE ANALYSIS

- ★ Complexity theory
- Space Complexity
  - Amount of memory
- Time Complexity 不是用時間(秒數)來算,是用演算法執行的步驟數
  - Amount of computing time

SPACE COMPLEXITY

試験に出ない.

- $S(P) = c+S_p(I)$
  - c: fixed space (instruction, simple variables, constants)
  - $S_p(I)$: depends on characteristics of instance I
    - Characteristics: number, size, values of I/O associated with I
- If n is the only characteristic, $S_p(I) \rightarrow S_p(n)$

SPACE COMPLEXITY (CONT'D.)

```
float abc(float a, float b, float c)
{
	return a+b+b*c+(a+b-c)/(a+b)+4.00;
}
```

$\boxed{S_{abc}(I)=0}$

SPACE COMPLEXITY (CONT'D.)

```
float rsum(float list[ ], int n)
{
  if (n)
    return rsum(list, n-1) + list[n-1];
  return 0;
 }
```

$\boxed{S_{sum}(I)=S_{sum}(n)=6n}$

Assumptions:

Space needed for one recursive call of the program

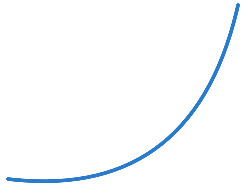
Type	Name	Number of bytes
Parameter: float	list[ ]	2 → 用 list 會用到 2個 bytes
Parameter: integer	n	2
Return address: (used internally)		2 (unless a far address)
Total		6

TIME COMPLEXITY

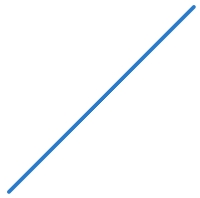
計算有多少步驟被執行

- $T(P) = c + T_p(I)$
  - c: compile time
  - $T_p(I)$: program execution time
    - Depends on characteristics of instance *I*

- Predict the growth in run times as the instance characteristics change



exponential
指數成長



linear

TIME COMPLEXITY (CONT'D.)

- Compile time (c)
  - Independent of instance characteristics
- Run (execution) time T_P
  - Real measurement
  - Analysis: counts of program steps
- Definition

  A program step is a syntactically or semantically meaningful program segment whose execution time is independent of the instance characteristics.

METHODS TO COMPUTE THE STEP COUNT

- Introduce variable count into programs
- Tabular method
- Determine the total number of steps contributed by each statement

  step per execution × frequency
- Add up the contribution of all statements

TIME COMPLEXITY (CONT'D.)

```
float sum(float list[ ], int n)
{
  float tempsum = 0;
  count++;          /* for assignment */
  int i;
  for (i=0; i<n; i++) {
    count++;        /* for the for loop */
    tempsum += list[i];
    count++;        /* for assignment */
  }
  count++;          /* last execution of for */
  return tempsum;
  count++;          /* for return */
}
```

1次

i=i+1

n+1次

2n+1次

n次

上下交換

1次

2n+3 steps

∴ Θ(n) #

TIME COMPLEXITY (CONT'D.)

```
float rsum(float list[ ], int n)
{
  count++;
  /* for if conditional */
  if (n<=0) {
    count++; // for return
    return 0
  }
  else {
    count++; // for return
    return rsum(list, n-1) + list[n-1];
  }
  count++;
  return list[0];
}
```

Tn

T(n)
=2+T(n-1)
=2+2+T(n-2)
...
=2n+T(0) 2
=2n+2

$\{1, 2, 3\}$ $T(3)$

$\{1, 2\} + 3$ $T(2) + 2$

$\{1\} + 2 + 3$ $T(1) + 2 + 2$

$\{\ \} + 1 + 2 + 3$ $T(0) + 2 + 2 + 2$

$T(0) = 2$, $n = 3$

$T(3) = 2n + 2 = 8$ #

3/13 說

TABULAR METHOD

Table 1.1: Step count table for Program 1.13 (p.40)

Statement	s/e	Frequency	Total steps
`float sum(float list[ ], int n)`			
`{`	0	1	0
`float tempsum = 0;`	1	1	1
`for(int i=0; i <n; i++)`	1	n+1	n+1
`tempsum += list[i];`	1	n	n
`return tempsum;`	1	1	1
`}`	0	1	0
Total			2n+3

s/e: steps per execution

TIME COMPLEXITY (CONT'D.)

- Difficult to determine the exact step counts

- What a step stands for is inexact
  eg. x := y versus x := y + z + (x/y) + …

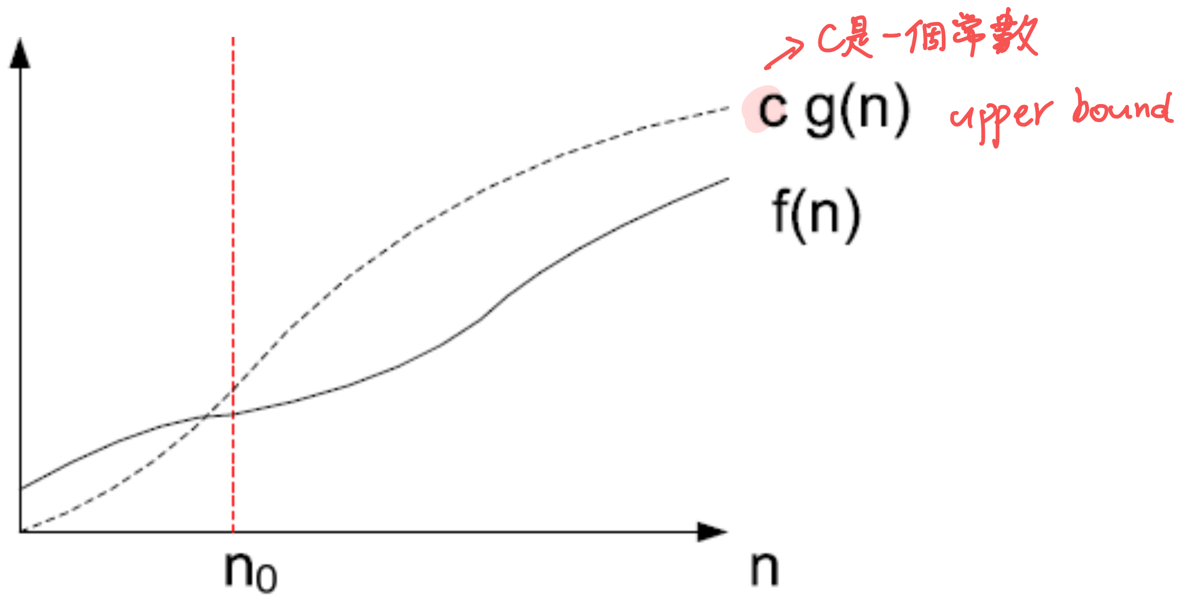
- Exact step count is not useful for comparison

- Step count doesn't tell how much time step takes

- Just consider the growth in run time
  Best case/Worst case/ Average case

ASYMPTOTIC NOTATION – BIG "OH"

- $f(n) = O(g(n))$ iff
  - $\exists$ a real constant $c > 0$ and an integer constant $n_0 \geq 1$, s.t. $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$



漸近式符號

ASYMPTOTIC NOTATION – BIG "OH" (CONT'D.)

- $f(n) = O(g(n))$ iff
  - $\exists$ a real constant $c > 0$ and an integer constant $n_0 \geq 1$, s.t. $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$
  - eg. 係數不重要，重點是指數
    - $3n + 6 = O(n)$
    - $4n^2 + 2n - 6 = O(n^2)$
    - $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$
      $f(n) = O(n^m)$

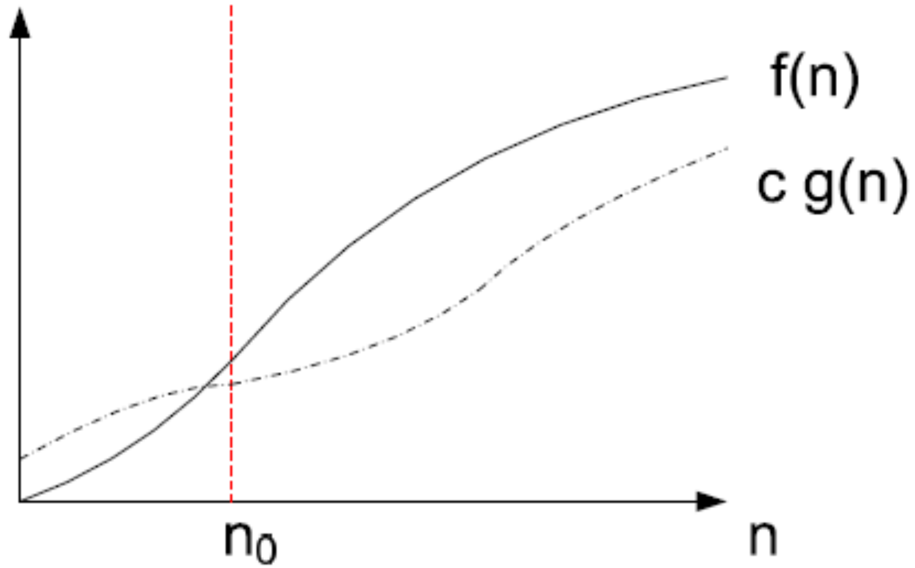
$4n^2 + 2n - 6 = f(n)$ $\exists\ C = 5$, $g(n) = n^2$ $C \cdot g(n) = 5n^2 \geq 4n^2 + 2n - 6$, $n \geq n_0 = 1$

- $g(n)$ should be a least upper bound. $O(n^2) = f(n)$:

ASYMPTOTIC NOTATION - OMEGA

- $f(n) = \Omega(g(n))$ iff
  - $\exists$ a real constant $c > 0$ and an integer constant $n_0 \geq 1$, s.t. $f(n) \geq c \cdot g(n)$, $\forall n \geq n_0$

- $g(n)$ should be a most lower bound.



ASYMPTOTIC NOTATION – OMEGA (CONT'D)

- eg.
  - $3n+3 = \Omega(n)$
  - $3n^2+4n-8 = \Omega(n^2)$
  - $6*2^n+n^2 = \Omega(2^n)$

$3n+3 \geq 2n$, $n \geq n_0 = 1$

$\downarrow$

$\Omega(n)$

3/6 (一) Quiz

(1) $5n^2+6n=\Theta(n^2)$
- 可以找到大於等於 … $6n^2$
- 可以找到小於等於 … $5n^2$

(2) $n! = O(n^2)$ --- $n^2 \geq n!$，所以會被 $O(n^2)$ bound住

(3) $n^2+1000n^2=\Theta(n^3)$

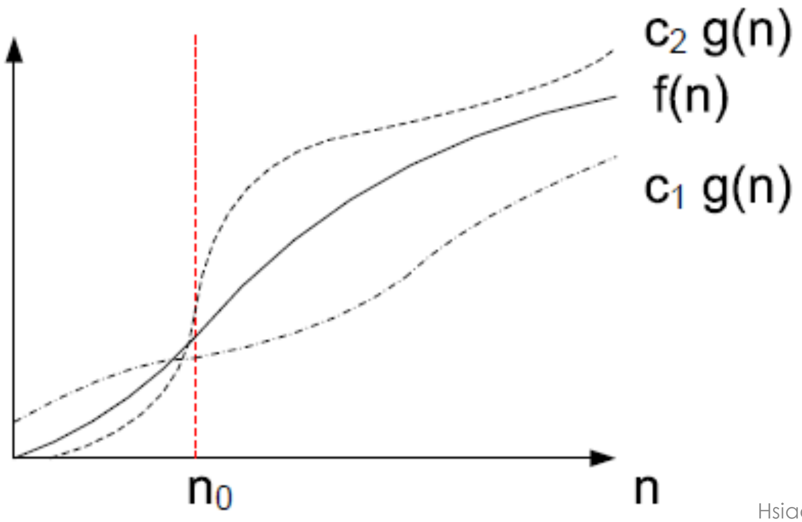
(4) $1^2+2^2+\cdots+n^2=\Theta(n^3)$

(5) $n^k+n+(n^k)\times\log n=\Theta(n^k\cdot\log n)$, for all $k\geq 1$

ASYMPTOTIC NOTATION - THETA

- $f(n) = \Theta\big(g(n)\big)$ iff
  - $\exists$ real constants c_1 and $c_2 > 0$ and an integer constant $n_0 \geq 1$, s.t. $c_1 g(n) \leq f(n) \leq c_2 g(n)$, $\forall n \geq n_0$

- $g(n)$ should be both upper bound and lower bound. It is called precise bound.



ASYMPTOTIC NOTATION – THETA (CONT'D)

- eg.
  - f(n) = $3n^2+4n-8$
  - f(n) = log(n!)

- $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

資料結構 繳交作業規則

Data Structure - About Submission

1. Submit your work in .pdf file. Name the file as DS_Assignment(#)_(StudentID).
   報告一律以 **PDF** 繳交，檔名為 Assignment#(編號)_學號，請依照繳交期限繳交。原則上，週一上課者，隔週星期五截止；週三上課者，隔週星期日截止。
   - Ex: DS_Assignment01_11031xxx0.pdf

2. Report contents include:
   - Assignment#, student ID, and name shall be shown in cover page.
   - You must include your C code in the assigned format with proper comments. Using Pygments to convert your code in the assigned format.
   - Your running results must be logged or screen-captured.
   - State your findings, conclusions, or answer questions in discussion.

   報告內容主要部份為：
   - 首頁需註明 Assignment 編號、學號、姓名
   - 程式碼與註解，務必使用 Pygments 將程式碼編輯為 C 語言的格式呈現
   - 執行結果，製作實驗紀錄或截圖
   - 實驗討論、結論或回答問題

3. If you have any questions, feel free to email your TA: (Please indicate your class in subject title.)
   若有問題請聯繫助教信箱 (請在主旨處說明你的信件來意與班級)