INTRODUCTION

Prof. Michael Tsai

2013/2/19

What is a data structure?

- An organization of information, usually in memory, for better algorithm efficiency.

- Or, a way to store and organize data in order to facilitate access and modifications.

$$2n^5 - 3n^2 + 11n + 27$$

0	1	2	3	4	5	6
27	11	-3	0	0	2	0

- 又可分為:
  - Linear data structure: 必須循序地存取 (如linked list, stack, queue)
  - Non-linear data structure: 可以不循序的存取 (如tree, graph)

What is an algorithm?

- An algorithm is any well-defined **computational procedure** that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.

- An algorithm is a tool for solving a well-specified **computational problem.**
- Computational problem → **input/output relationship**

- The algorithm describes a specific computational procedure for achieving that input/output relationship.

What is an algorithm?

- Example:

- Sorting problem:
- **Input:** A sequence of **n** numbers $\langle a_1, a_2, \dots, a_n \rangle$
- **Output:** A permutation (reordering) $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

- An **instance** of the sorting problem: $\langle 31,41,59,26,41,58 \rangle$
- A sorting algorithm should return as output the sequence $\langle 26,31,41,41,58,59 \rangle$.

What is an algorithm?

- All algorithm must satisfy the following criteria:
  - Input: 外部給的資訊(可以是零個或多個)
  - Output: 產生的結果(至少一個)
  - Definiteness: 每一個指令都是清楚而不模糊的
  - Finiteness: 所有的狀況下(所有的input), 演算法會在有限步驟之後結束
  - Effectiveness: 每一個指令都必須是簡單可以直接執行的 (必須可以執行)

Example

- Statement 1: **"Is n=2 the largest value of n for which there exist positive integers x, y, and z such that $x^n + y^n = z^n$ has a solution?"**
- Statement 2: **"Store 5 divided by zero into x and go to statement ㄎ."**

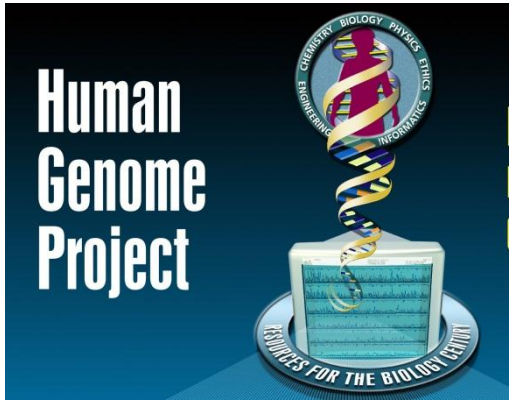
Definiteness

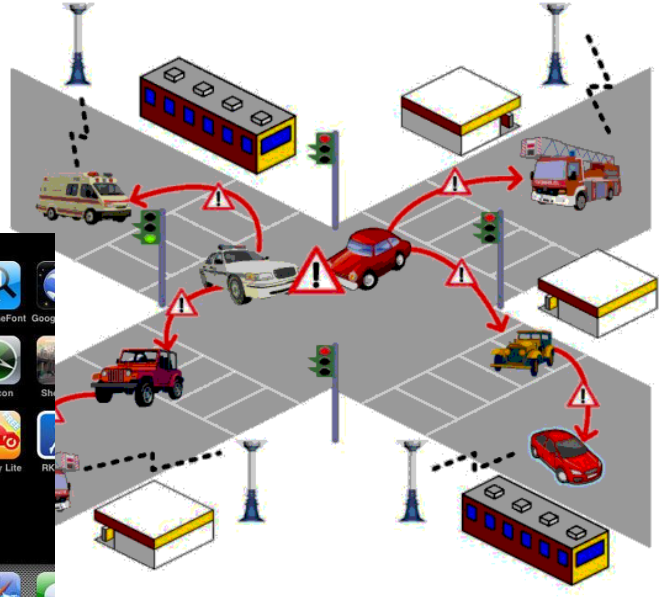
Effectiveness

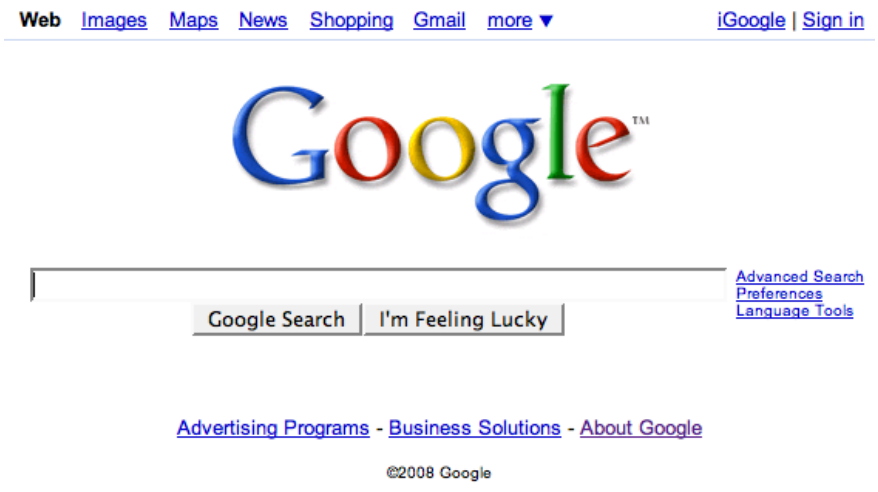
- Which criterion do they violate?
  - Input
  - Output
  - Definiteness
  - Finiteness
  - Effectiveness

Why are algorithms/data structure important?

- 它們被用在生活中的每個層面:







Why are algorithms/data structure important?

- Q: 如果電腦無限快/記憶體免錢，我們還需要研究資料結構與演算法嗎?
- A: Yes. 我們仍然需要確認我們想出來的解法會停止(不會無窮地執行下去)，而且**每次**都產生**正確**的答案。

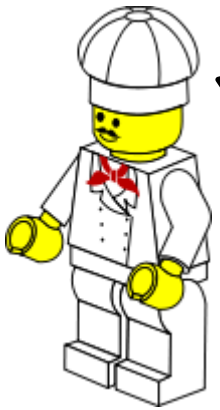
- 在這個假想的世界中，任何**正確**的解法都適用，因此我們通常會選**最容易**實作的方法。

- 但是在真實的世界裡:
  - 電腦不是無限快(計算需要**時間**)
  - 記憶體不是免錢(儲存資料需要**空間**)

- 因此我們需要學習如何好好利用這些**資源**來解決問題!

How do we describe an algorithm?

- Human language (English, Chinese, …)
- Programming language
- A mix of the above

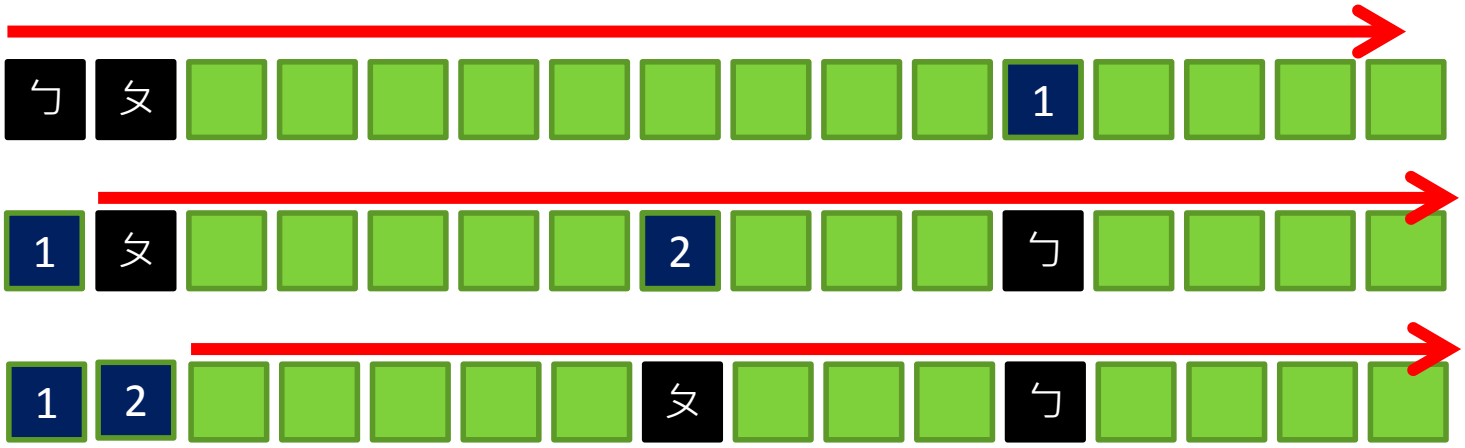


1. 拿平底鍋
2. 拿沙拉油
    1. 我們有油嗎?
        1. 有的話, 倒一茶匙的沙拉油到鍋子裡
        2. 沒有的話, 我們想要買油嗎?
            1. 是的話, 就去全聯買一罐沙拉油
            2. 如果不想的話, 只好先不煮了.
3. 打開火爐, …

Example: Selection Sort

Sorting problem:
Input: A sequence of **n** numbers $\langle a_1, a_2, \ldots, a_n \rangle$
Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

- Integers are stored in an array, list. The i-th integer is stored in list[i], 0<i<n.
- Solution: From those integers that are currently unsorted, find the smallest and place it next in the sorted list.



Example: Selection Sort

- First attempt:

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

        Interchange list[i] and list[min];
}
```

Task 1

Task 2

Task 2

Task 2

```
void swap(int *x, int *y) {
        int temp = *x;
        *x=*y;
        *y=temp;
}
```

Or

```
#define SWAP(x,y,t) ((t)=(x), (x)=(y), (y)=(t))
```

Task 1

Task 1

```
min=i;
for(j=i;j<n;++j)
        if (list[j]<list[min])
                min=j;
```

Final program

```
#include <stdio.h>
#include <math.h>

#define MAX-SIZE 101
#define SWAP(x,y,t) ((t) = (x), (x) = (y), (y) = (t))

void sort(int [],int); /*selection sort */
void main(void)
{
        int i,n;
        int list[MAX-SIZE];

        printf("Enter the number of numbers to generate: ");
        scanf (" %d", &n) ;
        if( n < 1 I In> MAX-SIZE) {
                fprintf(stderr, "Improper value of n\n");
                exit(EXIT_FAILURE);
        }

        for (i = 0; i < n; i++) {/*randomly generate numbers*/
                list[i] = rand() % 1000;
                printf("%d ",list[i]);
        }
```

```
        sort(list,n);
        printf("\n Sorted array:\n ");
        for (i = 0; i < n; i++) /* print out sorted numbers */
                printf("%d ",list[i]);
        printf("\n");
}


void sort(int list[],int n)
{
        int i, j, min, temp;

        for (i = 0; i < n-1; i++) {
                min = i;
                for (j = i+1; j < n; j++)
                        if (list[j] < list[min])
                                min = j;
                SWAP(list[i],list[min],temp);
        }
}
```

How do we prove that it is correct?

- **[Theorem]** Function sort(list,n) correctly sorts a set of n≥1 integers. The result remains in list[0], …, list[n-1] such that $list[0] \leq list[1] \leq \cdots \leq list[n-1]$.

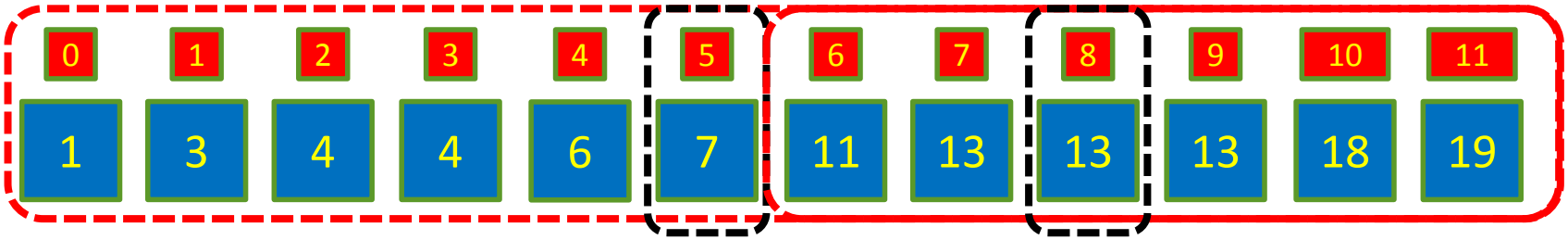
- Proof:
  When the outer for loop completes its iteration for i=q, we have $list[q] \leq list[r], q < r < n$. Further, on subsequent iterations, i>q and list[0] through list[q] are unchanged. Hence following the last iteration of the outer for loop (i.e., i=n-2), we have $list[0] \leq list[1] \leq \cdots \leq list[n-1]$.

Example: Binary Search

- Input:
  - *searchnum*: the number to be found
  - *list:* sorted array, size n*, and* $list[0] \leq list[1] \leq \cdots \leq list[n-1]$
- Output:
  - -1 if *searchnum* is not found in *list*
  - the index of *searchnum* in list[] if *searchnum* is found

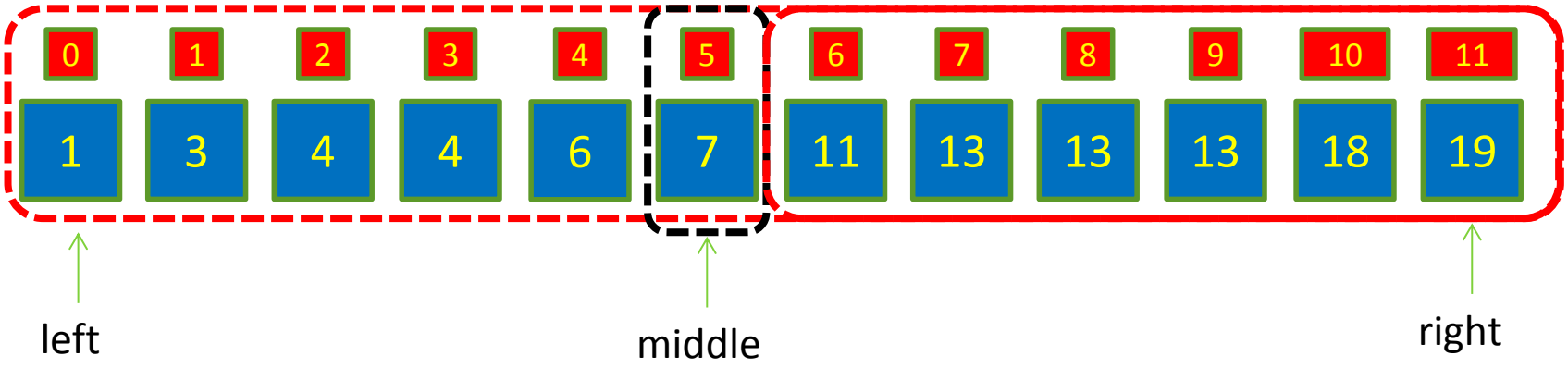
Example:

```
searchnum=13;
```



Example:

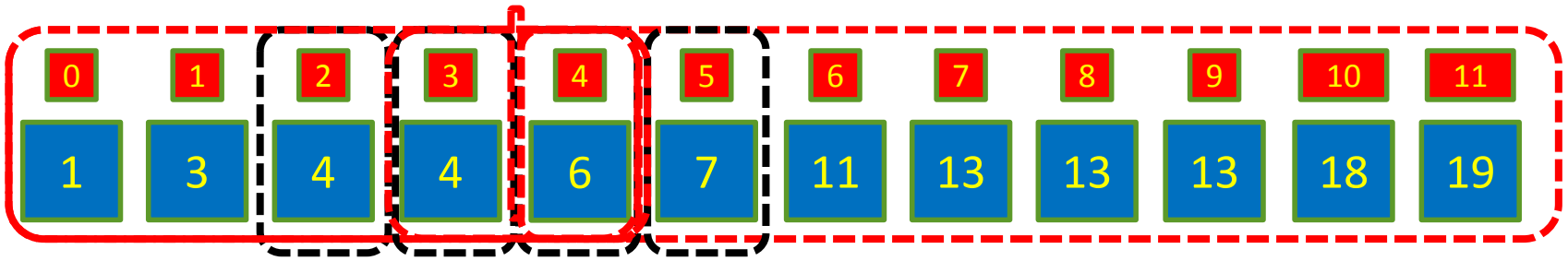
```
searchnum=13;
```




```
middle=(left+right)/2;
left=middle+1;
```

Example:

```
searchnum=5;
```




```
return -1;
```

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

list: 存sort好數字的array
searchnum: 要找的數字
left, right: 正在找的範圍左邊和右邊邊界

What is a data Type?

- A data type is a collection of objects and a set of operations that act on those objects.
- 每種data type 有所占的記憶體大小及可表示的資料數值範圍
- Data types in C
  - char, int, float, long, double (unsigned, signed, …)
  - Array
  - Structure (User-defined)

```
int iarray[16];

struct {
        int a;
        int b;
       char str[16];
        int * iptr;
} blah;
```

Operations of Data Types

- Operations
  - +, -, *, /, %, ==
  - =, +=, -=
  - ? :
  - sizeof, - (negative)
  - giligulu(int a, int b)

Data Type

- **Representation** of the objects of the data type
- Example: char
- char blah='A'; ('A': ASCII code is 65(dec), or 0x41 (hex))

1 byte of memory: 01000001

Q: The maximum number which can be represented with a char variable?

A: 255.

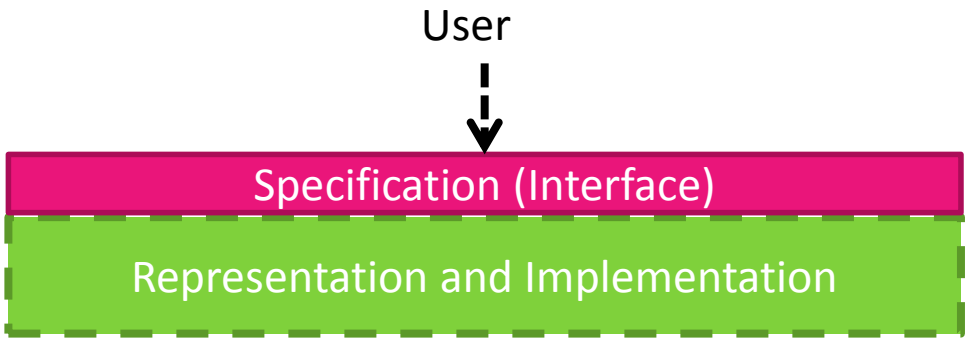
- How about char, int, long, float?

Data Type

- **Q:** 我們需要知道data type的representation嗎?
- A: 不一定.
  - 知道representation可能可以設計出更有效率的algorithm.
    - 但是當data type的representation被修改以後，程式可能必須重新確認、修正、或**完全重寫**。. 囧
    - 移植到不同的平台上 (x86, ARM, embedded system, …)
    - 改變program或library的specification(規格) (ex. 16-bit int → 32-bit long)

Abstract Data Type

- "Abstract Data Type" (ADT):
- Separate the specifications from the representation and the implementation



Abstract Data Type

- Specifications:
  - Operations:
    - Name of the function and the description of what the function does
    - The type of the argument(s)
    - The type of the result(s) (return value)
  - Data (usually hidden)
- Function categories:
  - Creator/constructor
  - Transformers
  - Observer/reporter

Example

```
ADT NaturalNumber is
        objects:
                an ordered subrange of the integers starting at zero
and ending at the maximum integer (lNT-MAX) on the computer
        functions:
                for all x, Y E NaturalNumber; TRUE, FALSE E Boolean
        and where +, -, <, and == are the usual integer operations

        NaturalNumber Zero()            ::=0
        Boolean IsZero(x)               ::= if (x) return FALSE
                                            else return TRUE
        Boolean Equal(x, y)             ::= if (x == y) return TRUE
                                            else return FALSE
        NaturalNumber Successor(x)      ::= if (x == INT-MAX) return x
                                            else return x + 1
        NaturalNumber Add(x, y)         ::= if ((x + y) <= INT-MAX)
                                            return x + y
                                          else return INT-MAX
        NaturalNumber Subtract(x, y)     ::= if (x < y) return 0
                                             else return x-y
end NaturalNumber
```

怎麼評估一個程式寫得好不好?

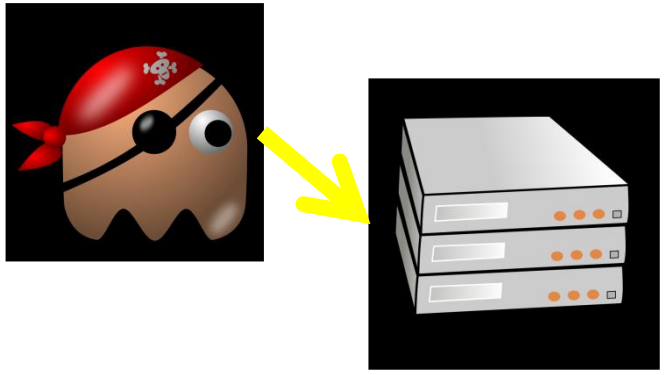
1. Does the program meet the original specifications of the task?
2. Does it work correctly?
3. Does the program **contain documentation** that shows how to use it and how it works?
4. Does the program effectively use **functions** to create **logical units**?
5. Is the program's code **readable**?

怎麼評估一個程式寫得好不好?

6. Does the program efficiently use primary and secondary storage?
   - Primary storage: memory?
   - Secondary storage: Hard drive, flash disk, etc.

7. Is the program's running time acceptable for the task?

   Example: Network intrusion detection system



(1) 99.8% detection rate, 50 minutes to finish analysis of a minute of traffic

(2) 85% detection rate, 20 seconds to finish analysis of a minute of traffic

怎麼評估一個程式寫得好不好?

6. 程式是否有效地使用主要及次要的儲存?

Space complexity

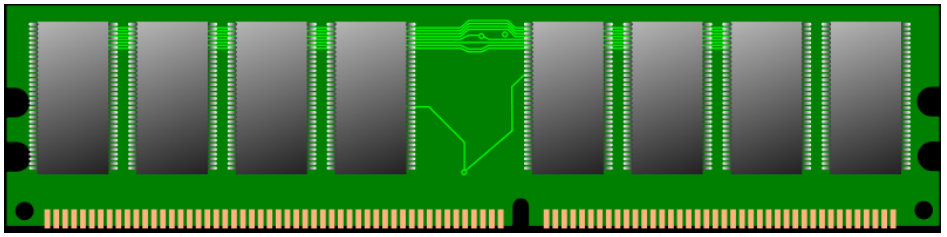
7. 程式的執行時間是否適合所需解決的工作內容?

Time complexity

空間及時間複雜度

- 程式的空間複雜度:
  - 程式執行完畢所需使用的所有空間(記憶體)
- 程式的時間複雜度:
  - 程式執行完畢所需使用的(執行)時間
- Goal: 找出執行時間/使用空間”如何”隨著input size變長(成長的有多快)
- 什麼是input size?
- 問題給的input的”元素數量”, 如:
  - Array大小
  - 多項式最高項的次方
  - 矩陣的長寬
  - 二進位數的位元數目

空間複雜度



- 程式所需空間:
  1. 固定的空間
     - 和input/output的大小及內容無關
  2. 變動的空間
     - 和待解問題P的某個input instance I(某一個input)有關
     - 跟recursive function會使用到的額外空間有關
- $S(P) = c + S_P(I)$

時間複雜度

- 一個程式*P*所需使用的時間:
  - Compile所需時間
  - 執行時間 (execution time or run time)

- Compile時間: 固定的. (例外?)
  - C (and other compiled programming languages)
    → One Compilation → Multiple Executions

- Run time: T_P
  - 和input instance的特性有關!

Reading materials for this lecture

- Cormen Chapter 1
- Horowitz Chapter 1.3-1.4