Performance Analysis

Kuan-Yu Chen (陳冠宇)

2018/09/17 @ TR-212, NTUST

Review

- Algorithm and Program
- Recursive Functions
  - Direct
  - Indirect
  - Tail
  - Compared with non-recursive functions

Space and Time Complexities

- There are many criteria upon which we can judge a program/algorithm
  - Does it do what we want it to do?
  - Dos it work correctly?
  - Is there documentation for the program/algorithm?
  - Is the code readable?

- A reasonable way is to judge programs by considering their **computing time** and **storage requirements**

Definition: The *space complexity* of a program is the amount of memory it needs to run to completion. The *time complexity* of a program is the amount of computer time it needs to run to completion. □

Hard to Imagine?

▼贈TOSHIBA 64GB 隨身碟▼

▼頂級日製商務機●新上市▼

Fujitsu 富士通LIFEBOOK S937-PB722黑
★第七代Core i7-7500U+512GB SSD+掌靜脈辨識器

- 處理器：第七代Intel® Core™ i7-7500U
- 螢幕：13.3"高解析FHD防眩光護眼螢幕
- (解析度1920 x 1080)
- 記憶體：24GB DDR4 2133MHz
- 硬碟：512GB SATAIII M.2 SSD
- 作業系統：Windows 10 Pro 64bits
- USB3.1 x3 (Gen 1; 關機充電技術x1);
- HDMI x1; VGA x1; 擴充底座介面
- 內建富士通專利掌靜脈辨識器
- 起始重量1.13kg起

3期0利率	32家	18期0利率	17家
6期0利率	32家	24期0利率	15家
10期0利率	27家	30期0利率	3家
12期0利率	11家	12期 分期	7家

建議售價 ~~$79999~~
網路價 $78800

VISA ATM 說明
信用卡紅利折抵刷卡金 多家銀行
PChome儲值
LINE Pay Pay G Pay

完售，請參考其他商品

技嘉 AORUS GTX 1080Ti 11G 顯示卡

- 顯示晶片 ：NVIDIA GeForce GTX 1080 Ti
- 記憶體 ：11GB GDDR5X
- 核心時脈 ：1569 MHz
- 記憶體時脈：11010 MHz
- 記憶體介面：352-bit
- 最高解析度：7680x4320
- 輸出端子 ：3x DP / 3x HDMI / DVI
- 電源連結器：2x 8-pin
- 體積(長x寬x高)：29.3 x 14.2 x 5.5cm

Space Complexity

- The space analysis can be classified into two parts
  - Fixed part
    - The instruction space, space for simple variables, space for constants, etc
  - Variable part
    - Space needed by referenced variables
    - The recursion stack space

- The space requirement $S(P)$ of a program P can be defined

$$S(P) = \underbrace{c}_{\text{fixed part, usually a constant}} + \underbrace{S_p}_{\text{variable part, depend on the task}}$$

fixed part
usually a constant

variable part
depend on the task

  - We usually concentrate on S_p

Recursion Stack Space.

- Given an Ackerman's function $A(m, n)$, please calculate $A(1,2)$.

$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

$$A(1,2) = A\big(0, A(1,1)\big)$$

$$A(1,1) = A\big(0, A(1,0)\big)$$

$$A(1,0) = A(0,1)$$

$$A(0,1) = 2$$

Recursion Stack Space..

- Given an Ackerman's function $A(m, n)$, please calculate $A(1,2)$.

$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

$$A(1,2) = A\big(0, A(1,1)\big) = A(0,3) = 4$$

$$A(1,1) = A\big(0, A(1,0)\big) = A(0,2) = 3$$

$$A(1,0) = A(0,1) = 2$$

$$A(0,1) = 2$$

Recursion Stack Space...

- Given an Ackerman's function $A(m, n)$, please calculate $A(1,2)$.

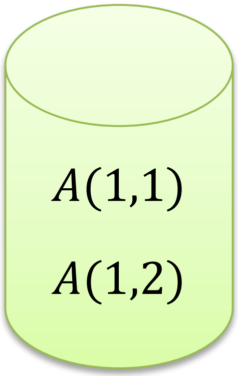
$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

$$A(1,2) = A\big(0, A(1,1)\big)$$

$$A(1,1) = A\big(0, A(1,0)\big)$$

$$A(1,0) = A(0,1)$$

$$A(0,1) = 2$$



Recursion Stack Space....

- Given an Ackerman's function $A(m, n)$, please calculate $A(1,2)$.

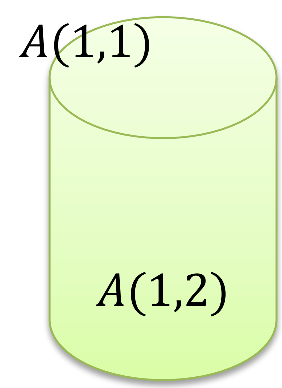
$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m, n-1)\big), & otherwise \end{cases}$$

$$A(1,2) = A\big(0, A(1,1)\big) = A(0,3) = 4$$

$$A(1,1) = A\big(0, A(1,0)\big) = A(0,2) = 3$$

$$A(1,0) = A(0,1) = 2$$

$$A(0,1) = 2$$



Time Complexity

- The time, $T(P)$, taken by a program P is the sum of the **compile time** and the **run (execution) time**
  - We mainly concentrate on the run time of a program

$$T(P) = \underbrace{c}_{\text{compile time}} + \underbrace{T_p}_{\text{run time}}$$

  - There are two ways to determine the run time
    - Measurement

      Execute the program

      Record the CPU time
    - Analysis

      Count only the number of program steps

      Count the number of instructions

Examples

- How many times does the function $call\_fun()$ execute?

```
for( a = 1 ; a <= n ; a++ )
    for( b = 1 ; b <= a ; b++ )
        for( c = 1 ; c <= a ; c++ )
            if( b != c )
                call_fun() ;
```

$$\sum_{a=1}^{n}(a^2 - a) = \sum_{a=1}^{n} a^2 - \sum_{a=1}^{n} a = \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} = \frac{n(n+1)(n-1)}{3}$$

$$\sum_{a=1}^{n} a^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Asymptotic Notations

- We introduce some terminology that will enable is to make **meaningful but inexact** statements about the time and space complexities of a program

Definition [*Big "oh"*]: $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") iff (if and only if) there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all n, $n \geq n_0$. □

Definition: [Omega] $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$. □

Definition: [Theta] $f(n) = \Theta(g(n))$ (read as "f of n is theta of g of n") iff there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$. □

Big-Oh.

Definition [*Big "oh"*]: $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") iff (if and only if) there exist positive constants c and n_0 such that $f(n) \le cg(n)$ for all n, $n \ge n_0$. □

- $f(n) = O(g(n))$ means that $c \times g(n)$ is an **upper bound** on the value of $f(n)$ for all n, where $n \ge n_0$

Example 1.14: $3n + 2 = O(n)$ as $3n + 2 \le 4n$ for all $n \ge 2$. $3n + 3 = O(n)$ as $3n + 3 \le 4n$ for all $n \ge 3$. $100n + 6 = O(n)$ as $100n + 6 \le 101n$ for $n \ge 10$. $10n^2 + 4n + 2 = O(n^2)$ as $10n^2 + 4n + 2 \le 11n^2$ for $n \ge 5$. $1000n^2 + 100n - 6 = O(n^2)$ as $1000n^2 + 100n - 6 \le 1001n^2$ for $n \ge 100$. $6*2^n + n^2 = O(2^n)$ as $6*2^n + n^2 \le 7*2^n$ for $n \ge 4$. $3n + 3 = O(n^2)$ as $3n + 3 \le 3n^2$ for $n \ge 2$. $10n^2 + 4n + 2 = O(n^4)$ as $10n^2 + 4n + 2 \le 10n^4$ for $n \ge 2$. $3n + 2 \ne O(1)$ as $3n + 2$ is not less than or equal to c for any constant c and all n, $n \ge n_0$. $10n^2 + 4n + 2 \ne O(n)$. □

Big-Oh.

Definition [*Big* "*oh*"]: $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") iff (if and only if) there exist positive constants c and n_0 such that $f(n) \le cg(n)$ for all n, $n \ge n_0$. □

- $f(n) = O(g(n))$ means that $c \times g(n)$ is an **upper bound** on the value of $f(n)$ for all n, where $n \ge n_0$

Example 1.14: $3n + 2 = O(n)$ as $3n + 2 \le 4n$ for all $n \ge 2$. $3n + 3 = O(n)$ as $3n + 3 \le 4n$ for all $n \ge 3$. $100n + 6 = O(n)$ as $100n + 6 \le 101n$ for $n \ge 10$. $10n^2 + 4n + 2 = O(n^2)$ as $10n^2 + 4n + 2 \le 11n^2$ for $n \ge 5$. $1000n^2 + 100n - 6 = O(n^2)$ as $1000n^2 + 100n - 6 \le 1001n^2$ for $n \ge 100$. $6*2^n + n^2 = O(2^n)$ as $6*2^n + n^2 \le 7*2^n$ for $n \ge 4$. $3n + 3 = O(n^2)$ as $3n + 3 \le 3n^2$ for $n \ge 2$. $10n^2 + 4n + 2 = O(n^4)$ as $10n^2 + 4n + 2 \le 10n^4$ for $n \ge 2$. $3n + 2 \ne O(1)$ as $3n + 2$ is not less than or equal to c for any constant c and all n, $n \ge n_0$. $10n^2 + 4n + 2 \ne O(n)$. □

Big-Oh.

Definition [*Big "oh"*]: $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") iff (if and only if) there exist positive constants c and n_0 such that $f(n) \le cg(n)$ for all n, $n \ge n_0$. □

- $f(n) = O(g(n))$ means that $c \times g(n)$ is an **upper bound** on the value of $f(n)$ for all n, where $n \ge n_0$

Example 1.14: $3n + 2 = O(n)$ as $3n + 2 \le 4n$ for all $n \ge 2$. $3n + 3 = O(n)$ as $3n + 3 \le 4n$ for all $n \ge 3$. $100n + 6 = O(n)$ as $100n + 6 \le 101n$ for $n \ge 10$. $10n^2 + 4n + 2 = O(n^2)$ as $10n^2 + 4n + 2 \le 11n^2$ for $n \ge 5$. $1000n^2 + 100n - 6 = O(n^2)$ as $1000n^2 + 100n - 6 \le 1001n^2$ for $n \ge 100$. $6*2^n + n^2 = O(2^n)$ as $6*2^n + n^2 \le 7*2^n$ for $n \ge 4$. $3n + 3 = O(n^2)$ as $3n + 3 \le 3n^2$ for $n \ge 2$. $10n^2 + 4n + 2 = O(n^4)$ as $10n^2 + 4n + 2 \le 10n^4$ for $n \ge 2$. $3n + 2 \ne O(1)$ as $3n + 2$ is not less than or equal to c for any constant c and all n, $n \ge n_0$. $10n^2 + 4n + 2 \ne O(n)$. □

Big-Oh.

Definition [*Big* "*oh*"]: $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") iff (if and only if) there exist positive constants c and n_0 such that $f(n) \le cg(n)$ for all n, $n \ge n_0$. □

- $f(n) = O(g(n))$ means that $c \times g(n)$ is an **upper bound** on the value of $f(n)$ for all n, where $n \ge n_0$

Example 1.14: $3n + 2 = O(n)$ as $3n + 2 \le 4n$ for all $n \ge 2$. $3n + 3 = O(n)$ as $3n + 3 \le 4n$ for all $n \ge 3$. $100n + 6 = O(n)$ as $100n + 6 \le 101n$ for $n \ge 10$. $10n^2 + 4n + 2 = O(n^2)$ as $10n^2 + 4n + 2 \le 11n^2$ for $n \ge 5$. $1000n^2 + 100n - 6 = O(n^2)$ as $1000n^2 + 100n - 6 \le 1001n^2$ for $n \ge 100$. $6*2^n + n^2 = O(2^n)$ as $6*2^n + n^2 \le 7*2^n$ for $n \ge 4$. $3n + 3 = O(n^2)$ as $3n + 3 \le 3n^2$ for $n \ge 2$. $10n^2 + 4n + 2 = O(n^4)$ as $10n^2 + 4n + 2 \le 10n^4$ for $n \ge 2$. $3n + 2 \ne O(1)$ as $3n + 2$ is not less than or equal to c for any constant c and all n, $n \ge n_0$. $10n^2 + 4n + 2 \ne O(n)$. □

Big-Oh.

Definition [*Big "oh"*]: $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") iff (if and only if) there exist positive constants c and n_0 such that $f(n) \le cg(n)$ for all n, $n \ge n_0$. □

- $f(n) = O(g(n))$ means that $c \times g(n)$ is an **upper bound** on the value of $f(n)$ for all n, where $n \ge n_0$

Example 1.14: $3n + 2 = O(n)$ as $3n + 2 \le 4n$ for all $n \ge 2$. $3n + 3 = O(n)$ as $3n + 3 \le 4n$ for all $n \ge 3$. $100n + 6 = O(n)$ as $100n + 6 \le 101n$ for $n \ge 10$. $10n^2 + 4n + 2 = O(n^2)$ as $10n^2 + 4n + 2 \le 11n^2$ for $n \ge 5$. $1000n^2 + 100n - 6 = O(n^2)$ as $1000n^2 + 100n - 6 \le 1001n^2$ for $n \ge 100$. $6*2^n + n^2 = O(2^n)$ as $6*2^n + n^2 \le 7*2^n$ for $n \ge 4$. $3n + 3 = O(n^2)$ as $3n + 3 \le 3n^2$ for $n \ge 2$. $10n^2 + 4n + 2 = O(n^4)$ as $10n^2 + 4n + 2 \le 10n^4$ for $n \ge 2$. $3n + 2 \ne O(1)$ as $3n + 2$ is not less than or equal to c for any constant c and all n, $n \ge n_0$. $10n^2 + 4n + 2 \ne O(n)$. □

Big-Oh..

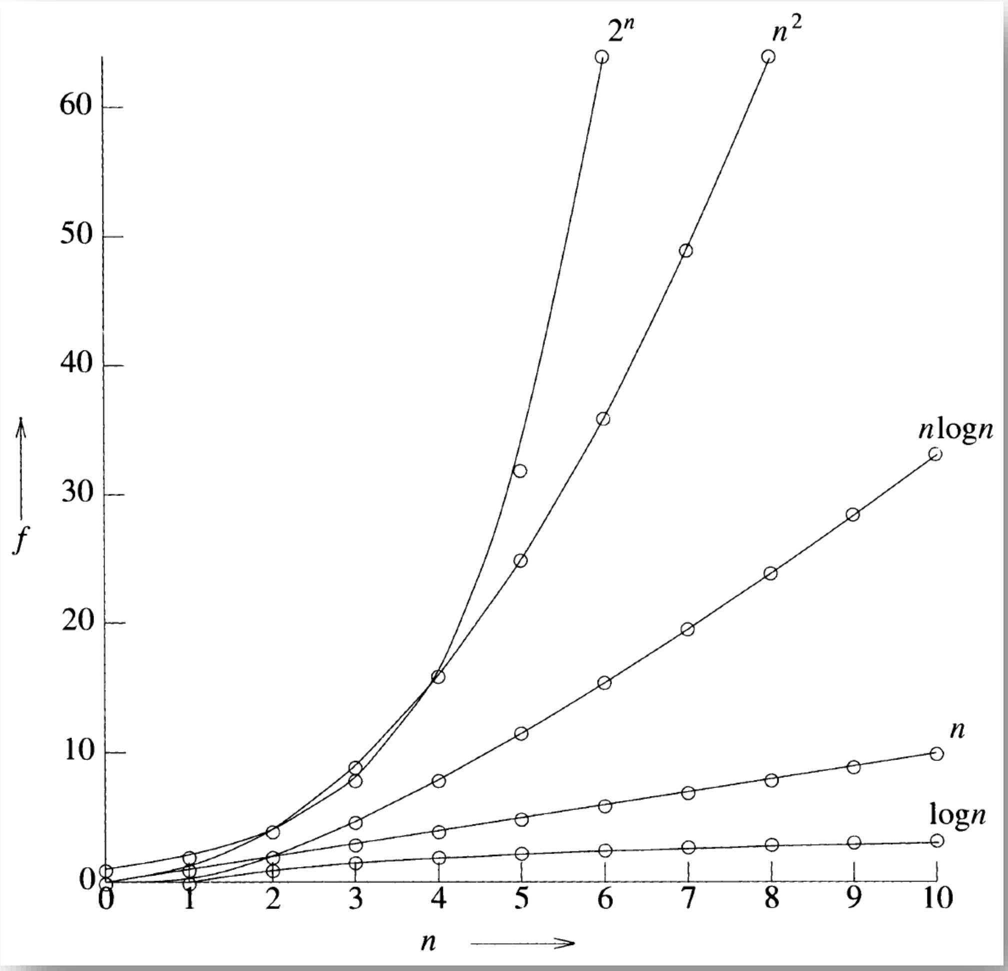
- For the statement $f(n) = O(g(n))$ to be **informative**, $g(n)$ should be as small a function of n as one can come up with
  - $3n + 3 = O(n)$ vs. $3n + 3 = O(n^2)$

- Fantastic names
  - $O(1)$ mean a computing time that is a constant
  - $O(n)$ is called linear
  - $O(n^2)$ is called quadratic
  - $O(n^3)$ is called cubic
  - $O(2^n)$ is called exponential

- Ordering
  - $O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

Big-Oh…

- $O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(n^c) < O(2^n) < O(3^n) < O(c^n) < O(n!) < O(n^n) < O\left(n^{c^n}\right)$



Omega

Definition: [Omega] $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$. □

- The function $g(n)$ is a lower bound on $f(n)$

Example 1.15: $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for $n \geq 1$ (actually the inequality holds for $n \geq 0$, but the definition of Ω requires an $n_0 > 0$). $3n + 3 = \Omega(n)$ as $3n + 3 \geq 3n$ for $n \geq 1$. $100n + 6 = \Omega(n)$ as $100n + 6 \geq 100n$ for $n \geq 1$. $10n^2 + 4n + 2 = \Omega(n^2)$ as $10n^2 + 4n + 2 \geq n^2$ for $n \geq 1$. $6*2^n + n^2 = \Omega(2^n)$ as $6*2^n + n^2 \geq 2^n$ for $n \geq 1$. Observe also that $3n + 3 = \Omega(1)$; $10n^2 + 4n + 2 = \Omega(n)$; $10n^2 + 4n + 2 = \Omega(1)$; $6*2^n + n^2 = \Omega(n^{100})$; $6*2^n + n^2 = \Omega(n^{50.2})$; $6*2^n + n^2 = \Omega(n^2)$; $6*2^n + n^2 = \Omega(n)$; and $6*2^n + n^2 = \Omega(1)$. □

- For the statement $f(n) = \Omega(g(n))$ to be informative, $g(n)$ should be as large a function of n as possible
  - $3n + 3 = \Omega(n)$ vs. $3n + 3 = \Omega(1)$
  - $6 \times 2^n + n^2 = \Omega(2^n)$ vs. $6 \times 2^n + n^2 = \Omega(1)$

Omega

Definition: [Omega] $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$. □

- The function $g(n)$ is a lower bound on $f(n)$

Example 1.15: $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for $n \geq 1$ (actually the inequality holds for $n \geq 0$, but the definition of Ω requires an $n_0 > 0$). $3n + 3 = \Omega(n)$ as $3n + 3 \geq 3n$ for $n \geq 1$. $100n + 6 = \Omega(n)$ as $100n + 6 \geq 100n$ for $n \geq 1$. $10n^2 + 4n + 2 = \Omega(n^2)$ as $10n^2 + 4n + 2 \geq n^2$ for $n \geq 1$. $6*2^n + n^2 = \Omega(2^n)$ as $6*2^n + n^2 \geq 2^n$ for $n \geq 1$. Observe also that $3n + 3 = \Omega(1)$; $10n^2 + 4n + 2 = \Omega(n)$; $10n^2 + 4n + 2 = \Omega(1)$; $6*2^n + n^2 = \Omega(n^{100})$; $6*2^n + n^2 = \Omega(n^{50.2})$; $6*2^n + n^2 = \Omega(n^2)$; $6*2^n + n^2 = \Omega(n)$; and $6*2^n + n^2 = \Omega(1)$. □

- For the statement $f(n) = \Omega(g(n))$ to be informative, $g(n)$ should be as large a function of n as possible
  - $3n + 3 = \Omega(n)$ vs. $3n + 3 = \Omega(1)$
  - $6 \times 2^n + n^2 = \Omega(2^n)$ vs. $6 \times 2^n + n^2 = \Omega(1)$

Omega

Definition: [Omega] $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$. □

- The function $g(n)$ is a lower bound on $f(n)$

Example 1.15: $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for $n \geq 1$ (actually the inequality holds for $n \geq 0$, but the definition of Ω requires an $n_0 > 0$). $3n + 3 = \Omega(n)$ as $3n + 3 \geq 3n$ for $n \geq 1$. $100n + 6 = \Omega(n)$ as $100n + 6 \geq 100n$ for $n \geq 1$. $10n^2 + 4n + 2 = \Omega(n^2)$ as $10n^2 + 4n + 2 \geq n^2$ for $n \geq 1$. $6*2^n + n^2 = \Omega(2^n)$ as $6*2^n + n^2 \geq 2^n$ for $n \geq 1$. Observe also that $3n + 3 = \Omega(1)$; $10n^2 + 4n + 2 = \Omega(n)$; $10n^2 + 4n + 2 = \Omega(1)$; $6*2^n + n^2 = \Omega(n^{100})$; $6*2^n + n^2 = \Omega(n^{50.2})$; $6*2^n + n^2 = \Omega(n^2)$; $6*2^n + n^2 = \Omega(n)$; and $6*2^n + n^2 = \Omega(1)$. □

- For the statement $f(n) = \Omega(g(n))$ to be informative, $g(n)$ should be as large a function of n as possible
  - $3n + 3 = \Omega(n)$ vs. $3n + 3 = \Omega(1)$
  - $6 \times 2^n + n^2 = \Omega(2^n)$ vs. $6 \times 2^n + n^2 = \Omega(1)$

Omega

Definition: [Omega] $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$. □

- The function $g(n)$ is a lower bound on $f(n)$

Example 1.15: $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for $n \geq 1$ (actually the inequality holds for $n \geq 0$, but the definition of Ω requires an $n_0 > 0$). $3n + 3 = \Omega(n)$ as $3n + 3 \geq 3n$ for $n \geq 1$. $100n + 6 = \Omega(n)$ as $100n + 6 \geq 100n$ for $n \geq 1$. $10n^2 + 4n + 2 = \Omega(n^2)$ as $10n^2 + 4n + 2 \geq n^2$ for $n \geq 1$. $6*2^n + n^2 = \Omega(2^n)$ as $6*2^n + n^2 \geq 2^n$ for $n \geq 1$. Observe also that $3n + 3 = \Omega(1)$; $10n^2 + 4n + 2 = \Omega(n)$; $10n^2 + 4n + 2 = \Omega(1)$; $6*2^n + n^2 = \Omega(n^{100})$; $6*2^n + n^2 = \Omega(n^{50.2})$; $6*2^n + n^2 = \Omega(n^2)$; $6*2^n + n^2 = \Omega(n)$; and $6*2^n + n^2 = \Omega(1)$. □

- For the statement $f(n) = \Omega(g(n))$ to be informative, $g(n)$ should be as large a function of n as possible
  - $3n + 3 = \Omega(n)$ vs. $3n + 3 = \Omega(1)$
  - $6 \times 2^n + n^2 = \Omega(2^n)$ vs. $6 \times 2^n + n^2 = \Omega(1)$

Theta

Definition: [Theta] $f(n) = \Theta(g(n))$ (read as "f of n is theta of g of n") iff there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all n, $n \ge n_0$. □

- The theta is more precise than both big-oh and omega
  - $g(n)$ is both an upper and lower bound on $f(n)$

Example 1.16: $3n + 2 = \Theta(n)$ as $3n + 2 \ge 3n$ for all $n \ge 2$, and $3n + 2 \le 4n$ for all $n \ge 2$, so $c_1 = 3$, $c_2 = 4$, and $n_0 = 2$. $3n + 3 = \Theta(n)$; $10n^2 + 4n + 2 = \Theta(n^2)$; $6*2^n + n^2 = \Theta(2^n)$; and $10*\log n + 4 = \Theta(\log n)$. $3n + 2 \ne \Theta(1)$; $3n + 3 \ne \Theta(n^2)$; $10n^2 + 4n + 2 \ne \Theta(n)$; $10n^2 + 4n + 2 \ne \Theta(1)$; $6*2^n + n^2 \ne \Theta(n^2)$; $6*2^n + n^2 \ne \Theta(n^{100})$; and $6*2^n + n^2 \ne \Theta(1)$. □

Theta

Definition: [Theta] $f(n) = \Theta(g(n))$ (read as "f of n is theta of g of n") iff there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all n, $n \ge n_0$. □

- The theta is more precise than both big-oh and omega
  - $g(n)$ is both an upper and lower bound on $f(n)$

Example 1.16: $3n + 2 = \Theta(n)$ as $3n + 2 \ge 3n$ for all $n \ge 2$, and $3n + 2 \le 4n$ for all $n \ge 2$, so $c_1 = 3$, $c_2 = 4$, and $n_0 = 2$. $3n + 3 = \Theta(n)$; $10n^2 + 4n + 2 = \Theta(n^2)$; $6*2^n + n^2 = \Theta(2^n)$; and $10*\log n + 4 = \Theta(\log n)$. $3n + 2 \ne \Theta(1)$; $3n + 3 \ne \Theta(n^2)$; $10n^2 + 4n + 2 \ne \Theta(n)$; $10n^2 + 4n + 2 \ne \Theta(1)$; $6*2^n + n^2 \ne \Theta(n^2)$; $6*2^n + n^2 \ne \Theta(n^{100})$; and $6*2^n + n^2 \ne \Theta(1)$. □

Example

- Given a recursive function $T(n) = 2T\left(\frac{n}{2}\right) + n$, where $T(1) = 0$, please write down the time complexity in big-oh for the function.

$$
\begin{aligned}
T(n) &= 2 \times T\left(\frac{n}{2}\right) + n \\
&= 2 \times \left[2 \times T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4 \times T\left(\frac{n}{4}\right) + 2 \times n \\
&= 4 \times \left[2 \times T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2 \times n = 8 \times T\left(\frac{n}{8}\right) + 3 \times n \\
&= \cdots \\
&= n \times T\left(\frac{n}{n}\right) + (\log_2 n) \times n = n \log_2 n
\end{aligned}
$$

$$\therefore T(n) = O(n \log_2 n)$$

Data Type

- All programming languages provide a set of predefined **data types**, and they also have the ability to construct new, and/or user-defined types

Definition: A *data type* is a collection of *objects* and a set of *operations* that act on those objects. □

- A data type should consider two aspects
  - Objects
  - Operations

Data Type	Objects	Operations
integer	$0, \pm 1, \pm 2, \pm 3, \cdots$	$+, -, \times, \div, \cdots$

Abstract Data Type

Definition: An *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. □

- ADT is implementation-independent
- Some programming languages provide explicit mechanisms to support the distinction between specification and implementation
  - Class in C++
  - Standard Template Library (STL) in C++
    - Map, Vector, List, …
    - http://www.cplusplus.com/reference/stl

fx **Member functions**

(constructor)	Construct map (public member function)
(destructor)	Map destructor (public member function)
operator=	Copy container content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return container size (public member function)
max_size	Return maximum size (public member function)

Element access:

operator[]	Access element (public member function)
at C++11	Access element (public member function)

Modifiers:

insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Observers:

key_comp	Return key comparison object (public member function)
value_comp	Return value comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific key (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

Questions?

kychen@mail.ntust.edu.tw