Performance Analysis

Kuan-Yu Chen (陳冠宇)

2020/09/21 @ TR-313, NTUST

Review

- Data type determines the set of values that a data item can take and the operations that can be performed on the item

Data Type	Size in Bytes	Range	Use
char	1	-128 to 127	To store characters
int	2	-32768 to 32767	To store integer numbers
float	4	3.4E-38 to 3.4E+38	To store floating point numbers
double	8	1.7E-308 to 1.7E+308	To store big floating point numbers

- Algorithm and Program
  - Algorithms + Data Structures = Programs

- Recursive Functions
  - Direct
  - Indirect
  - Tail
  - Compared with non-recursive functions

Space and Time Complexity

- Analyzing an algorithm means determining the amount of resources (such as time and memory) needed to execute it
  - The **time complexity** of an algorithm is basically the running time of a program as a function of a given input
  - The **space complexity** of an algorithm is the amount of computer memory that is required during the program execution as a function of a given input

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

  - Accordingly, the space requirement $S(P)$ of a program P can be defined

$$S(P) = \underline{c} + \underline{S_p}$$

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 $A(1,2)$

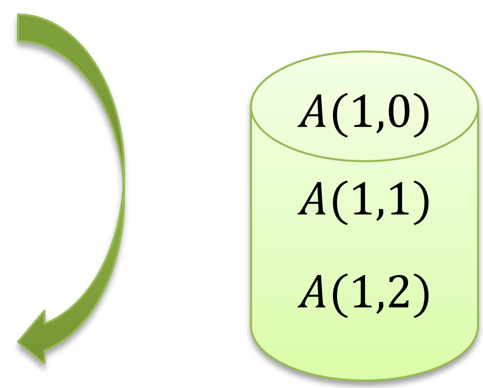
$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

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

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

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

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



Recursion Stack Space….

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

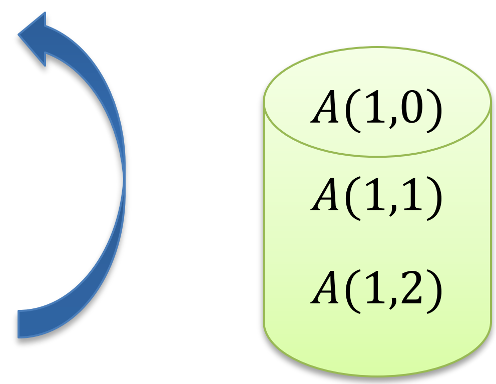
$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

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

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

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

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



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

$$T(P) = c + T_p$$

compile time　　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

Example

- 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() ;
```

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

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

Expressing Time and Space Complexity

- The time and space complexities of a given function $f(n)$, where n is a given input for the algorithm, can be expressed by some notations
  - We introduce some terminologies that will enable us 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 $f(n) \le cg(n)$ for all n, $n \ge n_0$. □

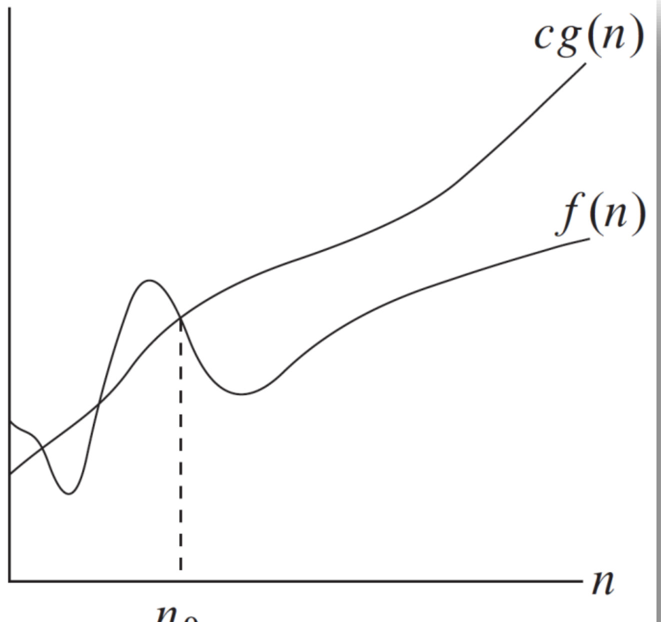
Definition: [Omega] $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \ge cg(n)$ for all n, $n \ge n_0$. □

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 $c_1 g(n) \le f(n) \le c_2 g(n)$ for all n, $n \ge n_0$. □

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 $f(n) \leq cg(n)$ for all n, $n \geq n_0$. □

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



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

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 $f(n) \leq cg(n)$ for all n, $n \geq n_0$. □

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

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

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 $n \ge 100$. <u>$6*2^n + n^2 = O(2^n)$ as $6*2^n + n^2 \le 7*2^n$ for $n \ge 4$.</u> $3n + 3 = O(n^2)$ 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..

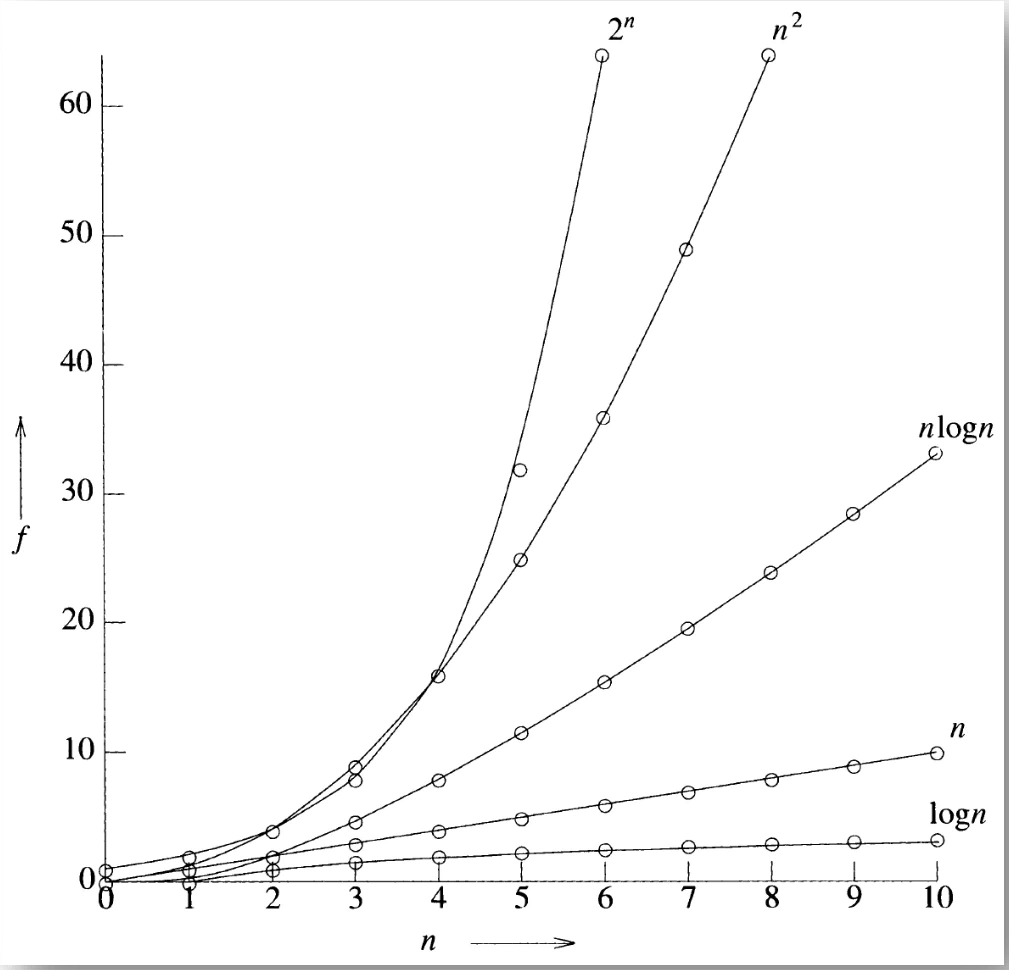
- For the statement $f(n) = O(g(n))$ to be **informative**, $g(n)$ should be as small a function of n as one can come up with
  - $3n + 3 = O(n)$ vs. $3n + 3 = O(n^2)$

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

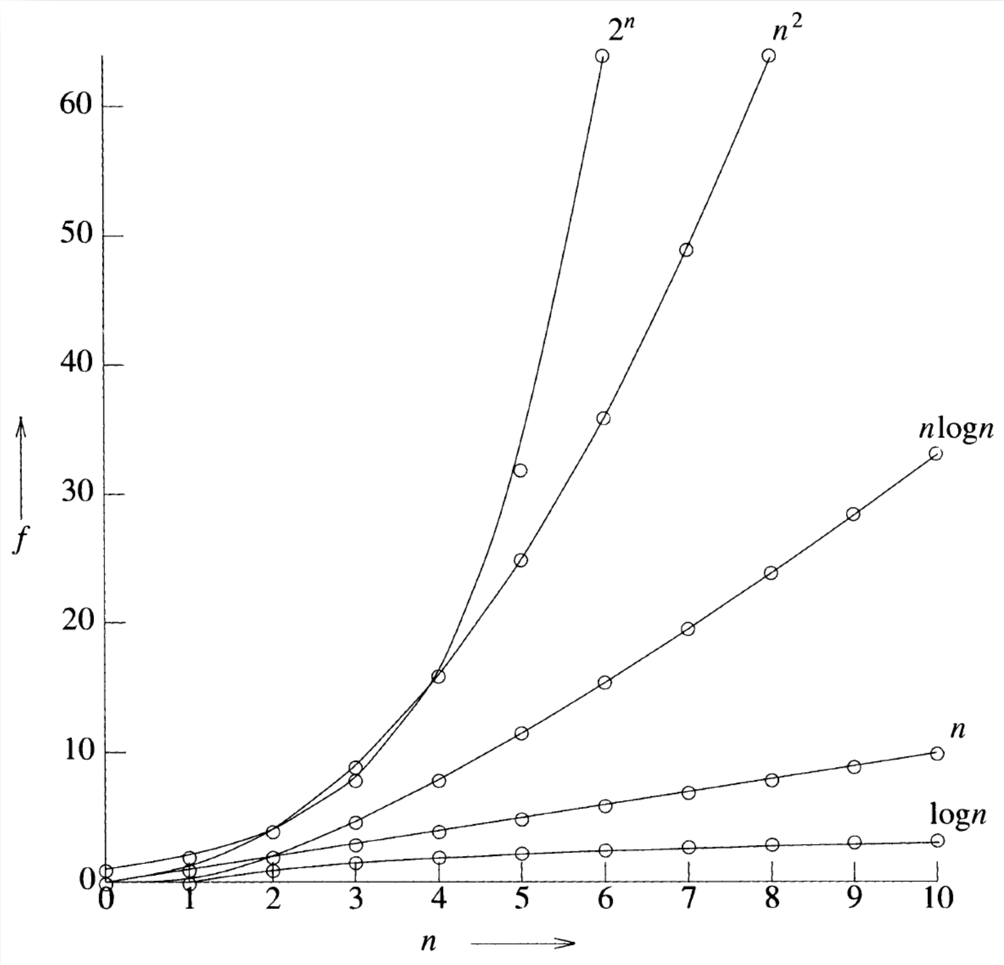
Big-Oh…

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



Big-Oh…

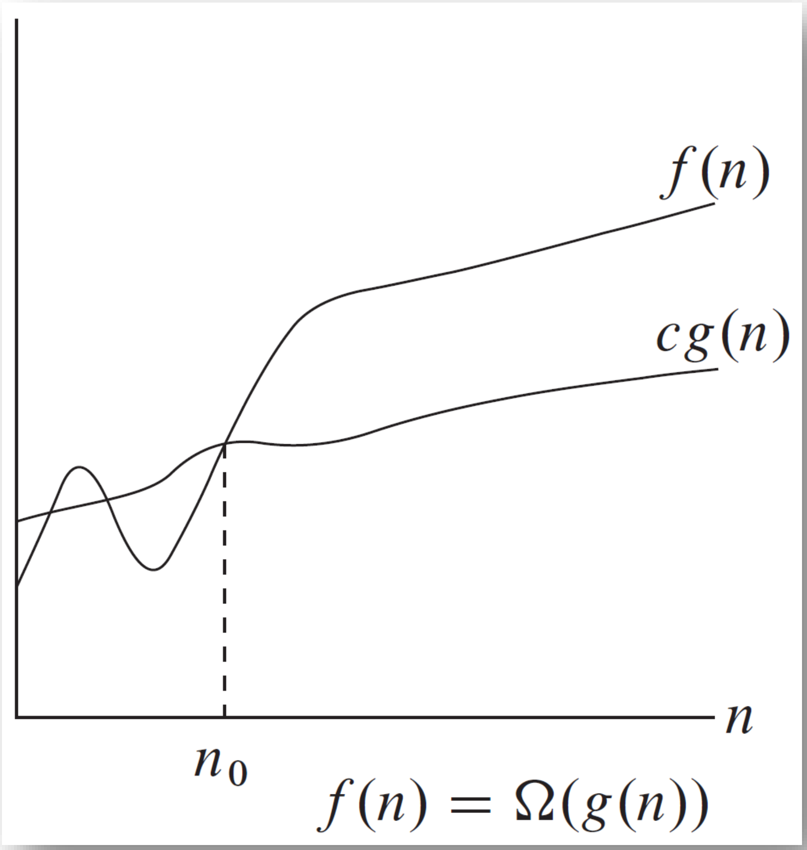
- $O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(n^c) < O(2^n) < O(3^n) < O(c^n) < O(n!) < O(n^n) < O(n^{c^n})$



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 $f(n)$



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 $6*2^n + n^2 = \Omega(1)$. □

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 $6*2^n + n^2 = \Omega(1)$. □

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 $f(n)$

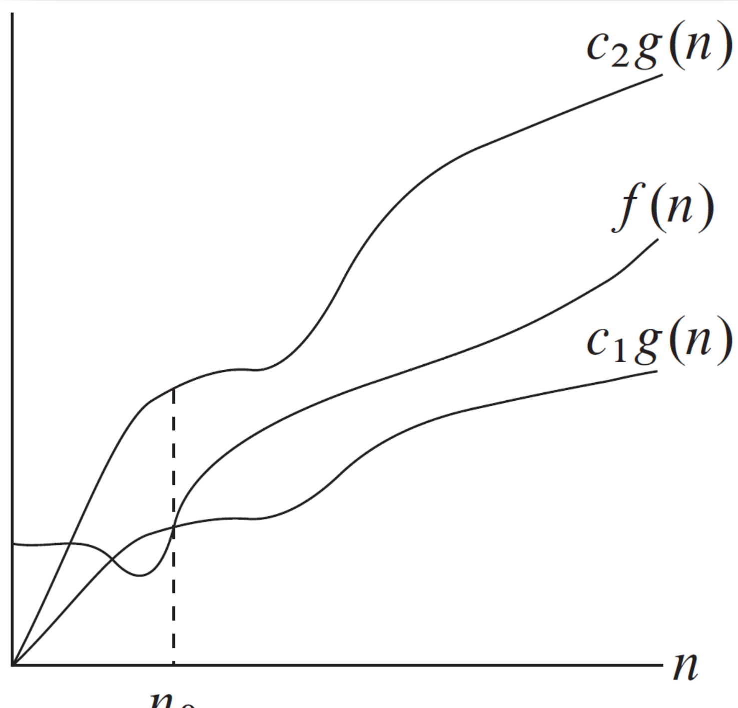
Example 1.15: $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for $n \geq 1$ (actually the inequality holds for $n \geq 0$, but the definition of Ω requires an $n_0 > 0$). $3n + 3 = \Omega(n)$ as $3n + 3 \geq 3n$ for $n \geq 1$. $100n + 6 = \Omega(n)$ as $100n + 6 \geq 100n$ for $n \geq 1$. $10n^2 + 4n + 2 = \Omega(n^2)$ 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 $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$. □

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



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

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
  - We assume n is a power of 2 for simplification
    - That is $2^x = n$

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

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

Questions?

kychen@mail.ntust.edu.tw