Advanced Analysis

Kuan-Yu Chen (陳冠宇)

Review

- Space and Time complexity
  - Big-Oh
  - Omega
  - Theta

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, $n \geq n_0$. □

Online Link

- Webex: https://ctld.webex.com/meet/kychen

Little-Oh

$o(g(n)) = \{ f(n) :$ for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le f(n) < cg(n)$ for all $n \ge n_0 \}$.

- The value of n_0 may depend on c

- The definitions of O-notation and o-notation are similar

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, $n \ge n_0$. □

- $f(n) = \mathrm{O}\big(g(n)\big)$, the bound $0 \le f(n) \le cg(n)$ holds for ***some*** constant $c > 0$
- $f(n) = \mathrm{o}\big(g(n)\big)$, the bound $0 \le f(n) < cg(n)$ holds for ***all*** constants $c > 0$

- Examples:
  - $2n = \mathrm{o}(n^2)$
  - $2n^2 \neq \mathrm{o}(n^2)$

Little-Omega

$$\omega(g(n)) = \{f(n) : \text{ for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \le cg(n) < f(n) \text{ for all } n \ge n_0\} .$$

- By analogy, ω-notation is to Ω-notation as o-notation is to O-notation

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, $n \ge n_0$. □

- Examples:
  - $\frac{n^2}{2} = \omega(n)$
  - $\frac{n^2}{2} \neq \omega(n^2)$

Summary

$$
\begin{aligned}
f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) &\text{ imply } f(n) = \Theta(h(n)) \\
f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) &\text{ imply } f(n) = O(h(n)) \\
f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) &\text{ imply } f(n) = \Omega(h(n)) \\
f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) &\text{ imply } f(n) = o(h(n)) \\
f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) &\text{ imply } f(n) = \omega(h(n))
\end{aligned}
$$

$$
\begin{aligned}
f(n) &= \Theta(f(n)) \\
f(n) &= O(f(n)) \\
f(n) &= \Omega(f(n))
\end{aligned}
$$

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n))$$

$$
\begin{aligned}
f(n) = O(g(n)) &\text{ if and only if } g(n) = \Omega(f(n)) \\
f(n) = o(g(n)) &\text{ if and only if } g(n) = \omega(f(n))
\end{aligned}
$$

Example – 1.

- Given a recursive function, which is used to calculate the sum, the function $T(n)$ denotes the number of elementary operations performed by the function call $sum(n)$. Please an give an asymptotic estimate of the time complexity.

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

$$
\begin{aligned}
sum(n) &= n + sum(n-1) \\
&= n + \{(n-1) + sum(n-2)\} \\
&= n + \{(n-1) + [(n-2) + sum(n-3)]\} \\
&= \cdots
\end{aligned}
$$

Example – 1..

- Given a recursive function, which is used to calculate the sum, the function $T(n)$ denotes the number of elementary operations performed by the function call $sum(n)$. Please an give an asymptotic estimate of the time complexity.

$$\begin{aligned} sum(n) &= n + sum(n-1) \\ &= n + \{(n-1) + sum(n-2)\} \\ &= n + \{(n-1) + [(n-2) + sum(n-3)]\} \\ &= \cdots \end{aligned}$$

$$\begin{aligned} T(n) &= y + T(n-1) \\ &= y + \{y + T(n-2)\} \\ &= y + \{y + [y + T(n-3)]\} \\ &= y + \{y + [y + \cdots + T(1)]\} \\ &= y + \{y + [y + \cdots + x]\} \end{aligned}$$

  - If $n > 1$, the function will perform a fixed number of operations and will make a recursive call, so it can be calculated by $T(n) = y + T(n-1)$
  - $sum(1)$ is computed using a fixed number of operations, so we specify $T(1) = x$

Example – 1…

- Given a recursive function, which is used to calculate the sum, the function $T(n)$ denotes the number of elementary operations performed by the function call $sum(n)$. Please an give an asymptotic estimate of the time complexity.
  - The constants x and y are set to 1, since we are only looking for an asymptotic estimate of the time complexity

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

$$\because sum(n) = n + sum(n-1)$$
$$= n + \{(n-1) + sum(n-2)\}$$
$$= n + \{(n-1) + [(n-2) + sum(n-3)]\}$$
$$= \cdots$$

$$T(1) = 1$$
$$T(n) = 1 + T(n-1)$$
$$\therefore T(n) = 1 + T(n-1)$$
$$= 1 + \{1 + T(n-2)\}$$
$$= 1 + \{1 + [1 + T(n-3)]\}$$
$$= \cdots$$
$$= k + T(n-k)$$
$$= \cdots$$
$$= (n-1) + T(1) = (n-1) + 1 = n = \Theta(n)$$

Example – 2

- Given a recursive function and its time complexity can be denoted by $T(n) = 2T\left(\frac{n}{2}\right) + n$, 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 $2^x = n$

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

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

Master Method.

- The master method provides a "cookbook" method for solving recurrences of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is a positive function

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Method..

- The master method theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

– Take $T(n) = 9T\left(\frac{n}{3}\right) + n$ for example

- $a = 9, b = 3, f(n) = n$, and $n^{\log_b a} = n^{\log_3 9} = n^2$
- $f(n) = n = O\left(n^{\log_b a-\epsilon}\right) = O\left(n^{\log_3 9-\epsilon}\right)$, where $\epsilon = 1$
  - ✓ Case1
- $T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_3 9}\right) = \Theta\left(n^2\right)$

Master Method...

- The master method theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

– Take $T(n) = T\left(\frac{2n}{3}\right) + 1$ for example

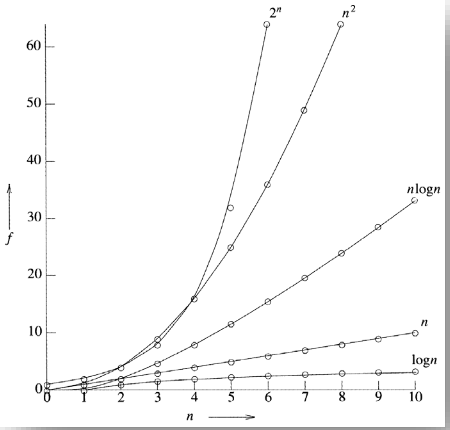
- $a = 1, b = \frac{3}{2}, f(n) = 1$, and $n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$
- $f(n) = 1 = \Theta\left(n^{\log_b a}\right) = \Theta(1)$
  - ✓ Case2
- $T(n) = \Theta\left(n^{\log_b a} \log_2 n\right) = \Theta(\log_2 n)$

Master Method....

- The master method theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$



– Take $T(n) = 3T\left(\frac{n}{4}\right) + n\log_2 n$ for example

- $a = 3, b = 4,\ f(n) = n\log_2 n$, and $n^{\log_b a} = n^{\log_4 3}$
- $f(n) = \Omega(n^{\log_4 3 + \epsilon})$
- $af\left(\frac{n}{b}\right) = 3f\left(\frac{n}{4}\right) = 3\frac{n}{4}\log_2\frac{n}{4} = 3\frac{n}{4}(\log_2 n - \log_2 4)$

  $= \frac{3}{4}n\log_2 n - \frac{3}{2}n \leq cf(n) = cn\log_2 n$, when $c = \frac{3}{4}$

  ✓ Case3
- $T(n) = \Theta\big(f(n)\big) = \Theta(n\log_2 n)$

Master Method.....

- The master method theorem

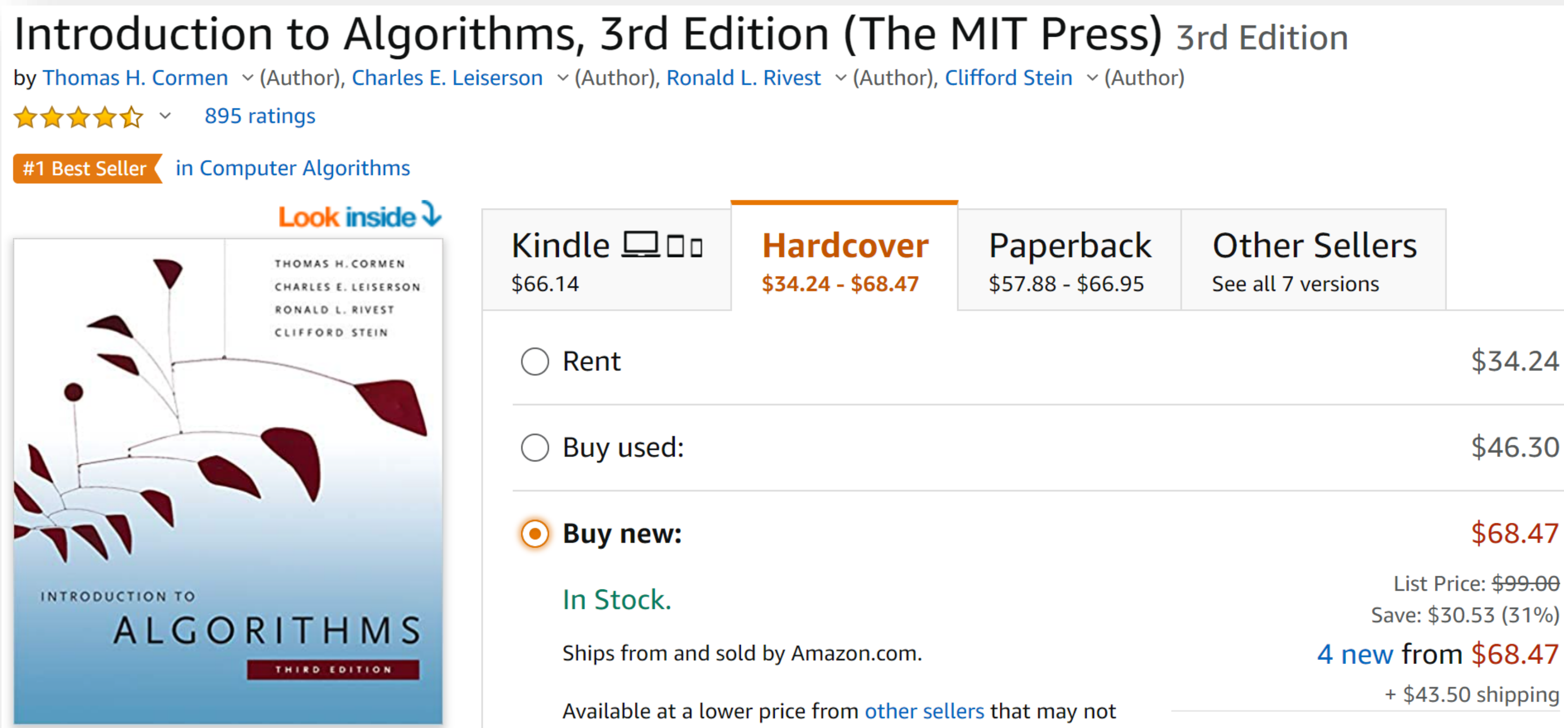
$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

- In each of the three cases, we compare $f(n)$ with $n^{\log_b a}$
- In case1, $n^{\log_b a}$ is larger than $f(n)$, thus $T(n) = \Theta(n^{\log_b a})$
- In case3, $f(n)$ is larger than $n^{\log_b a}$, thus $T(n) = \Theta(f(n))$
- In case2, $f(n)$ and $n^{\log_b a}$ are the same size, we multiply by a logarithmic factor, and the solution is $T(n) = \Theta(n^{\log_b a} \log_2 n) = \Theta(f(n) \log_2 n)$

Master Method……

- Check out the proof from:



Questions?

kychen@mail.ntust.edu.tw