Phân tích & Thiết kế thuật toán (Algorithms Design & Analysis)

L/O/G/O

GV: HUỲNH THỊ THANH THƯƠNG

Email: thuonghtt@uit.edu.vn

PHÂN TÍCH THUẬT TOÁN

CHƯƠNG 1: TỔNG QUAN

Nội dung

1. Vấn đề và xác định vấn đề
2. Thuật toán
3. Phân tích thuật toán
4. Độ phức tạp

Sửa bài tập tại lớp

a) Phép suy ra bên dưới là đúng hay sai và vì sao?

$$\frac{1}{2}n^2 = O(n^2)$$
$$n^2 + 1 = O(n^2)$$
$$\Rightarrow \frac{1}{2}n^2 = n^2 + 1 \quad ???$$

b) Xét $f(n) = 7n^2$,

$g(n)=n^2 - 80n$,

$h(n)=n^3$

Chứng minh (dùng định nghĩa, không dùng lim):

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

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

$f(n) = O(h(n))$,

$h(n) \neq O(f(n))$

c. Nếu $t1(n) \in O(f(n))$ và $t2(n) \in O(g(n))$ thì $t1(n) + t2(n) \in O(\max\{f(n), g(n)\})$

Hướng dẫn bài tập tại lớp

b. Với mỗi nhóm hàm bên dưới, hãy sắp xếp các hàm số theo thứ tự tăng dần của "order of growth". Ký hiệu: log là log cơ số 2, $\binom{n}{k}$ là tổ hợp chập k của n

Group 1: $f_1(n) = \binom{n}{n-2}$ $f_2(n) = \sqrt[3]{n}(\log n)^2$ $f_3(n) = n^{5(\log n)^2}$ $f_4(n) = 10\log n + 2\log(\log n)$	**Group 2:** $f_5(n) = n2^{n/2}$ $f_6(n) = n^{\log n}$ $f_7(n) = n^{n^{1/5}}$ $f_8(n) = 2^{2^{1000000}}$

Bài tập trên lớp (điểm quá trình): Inclass#05

a. **CM: O(1)=O(C), C là hằng số**

b. $(\boldsymbol{nlogn - 3n}) = \boldsymbol{\Omega(nlogn)}$

c. Hãy sắp xếp 07 hàm số bên dưới theo thứ tự tăng dần của "order of growth", và có giải thích ngắn gọn cách thực hiện. Ký hiệu: log là log cơ số 2.

$f_1(n) = n^2\sqrt{n}$	$f_2(n) = n^{logn}$
$f_3(n) = n^2 * (\log n)^4$	$f_4(n) = 4^{n^4}$
$f_5(n) = \sum_{i=1}^{n}(i^2)$	$f_6(n) = 5^n$
$f_7(n) = n^{n^{1/5}}$	

Đã sửa tại lớp

Chiến lược cho Big-O

- Đôi khi, cách đơn giản nhất để chứng minh $T(n) = O(f(n))$ là chọn c là tổng của các hệ số dương của $f(n)$
- Thường lờ đi các hệ số âm. Vì sao?
- Ví dụ:

Để chứng minh $5n^2 + 3n + 20 = O(n^2)$,

Chọn $c = 5 + 3 + 20 = 28$.

Khi đó, nếu $n \geq n_0 = 1$, $5n^2 + 3n + 20 \leq 5n^2 + 3n^2 + 20n^2 = 28n^2$

Chiến lược cho Big-O

- Cách làm trên không phải luôn dễ dàng
- Ví dụ: chứng minh

  $(\sqrt{2})^{\log n} + \log^2 n + n^4 = O(2^n)$

  $n^2 = O(n^2 - 13n + 23)$

- Chỉ quan tâm đến lũy thừa cao nhất của n
- Bỏ qua các hằng số

Một số lưu ý

- Vài trò của Hằng số trong phân tích

$$2n + \alpha\sqrt{n} + O(\frac{1}{n})$$

α Và β có vai trò như thế nào?

$$\beta\sqrt{n} + O(\frac{1}{n})$$

- Lý thuyết: do n khá lớn nên không đáng kể, không tạo ra sự khác biệt trọng yếu nên có thể bỏ qua
- Thực nghiệm: đôi khi rất quan trọng → cẩn thận

Big-O notation

❖ **Các tính chất:**

- $O(Cf(n)) = O(f(n))$ với C là hằng số
- $O(C) = O(1)$
- $f(n) \in O(g(n))$ và $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$.
- $g(n) \in O(h(n)) \Rightarrow O(g(n)) \subseteq O(h(n))$
- $O(f(n)) = O(g(n)) \Leftrightarrow g(n) \in O(f(n))$ và $f(n) \in O(g(n))$
- ...

Big-Ω notation (lower bounds)

- $\Omega(g(n)) = \{ f(n) :$ there exist constants $c > 0, n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0 \}$

$$\sqrt{n} = \Omega(\lg n) \quad (c = 1, n_0 = 16)$$

Big-Θ notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

- Ví dụ:

$$\frac{1}{2}n^2 - 2n = \Theta(n^2)$$

Chiến lược cho Big- Ω, Θ

- Chứng minh T(n) = Ω (f(n))
  - Thường chọn c < 1
  - Chiến lược tốt là thử chọn c trước rồi xác định n_o
- Chứng minh T(n) = Θ (f(n))
  - Dùng định nghĩa để tìm c_1, c_2, n_0
  - f(n) = Θ(g(n)) nếu và chỉ nếu f(n) = O(g(n)) và f(n) = Ω(g(n)).

o-notation and ω-notation

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

EXAMPLE: $2n^2 = o(n^3)$ $\quad (n_0 = 2/c)$

$$\lim_{n\to\infty} [f(n) / g(n)] = 0$$

o-notation and ω-notation

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

EXAMPLE: $\sqrt{n} = \omega(\lg n)$ $\quad (n_0 = 1+1/c)$

$$\lim_{n\to\infty} [f(n) / g(n)] = \infty$$

Các lớp độ phức tạp

Dạng O	Tên Phân loại	
O(1)	Hằng	
$O(\log_2(n))$	logarit	
$O(\sqrt{n})$	Căn thức	
$O(\sqrt[3]{n})$		
…		
$O(\sqrt[m]{n})$		
O(n)	Tuyến tính	Đa thức
$O(n^2)$	Bình phương	
$O(n^3)$	Bậc ba	
…		
$O(n^m)$	Đa thức	
$O(c^n)$, với c>1	Mũ	Độ phức tạp lớn
O(n!)	Giai thừa	

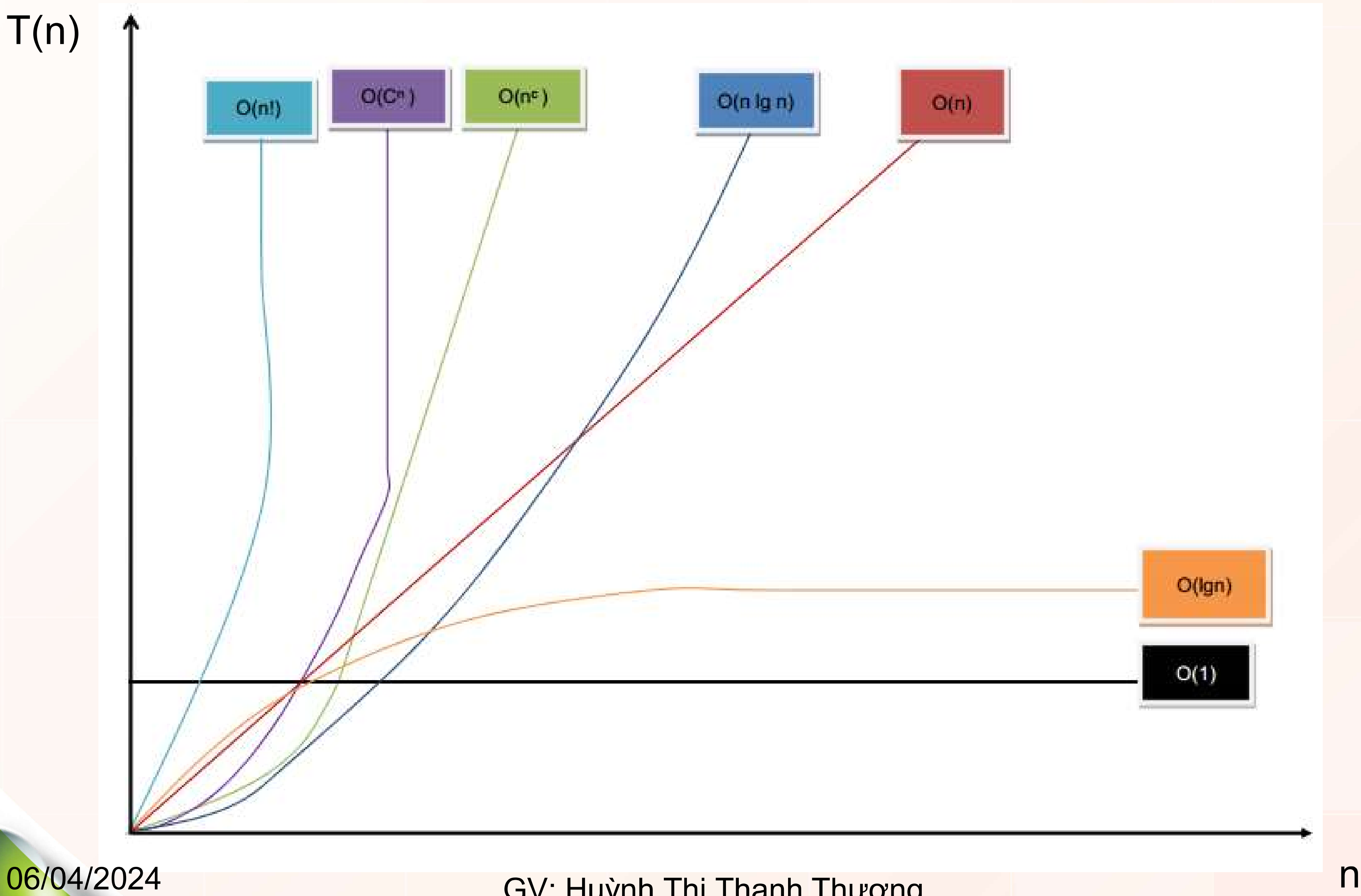
Thường nói đến $O(2^n)$

KHÔNG CHẤP NHẬN ĐƯỢC

Common Rates of Growth

- Constant: $\Theta(k)$, for example $\Theta(1)$
- Linear: $\Theta(n)$
- Logarithmic: $\Theta(\log_k n)$
- $n \log n$: $\Theta(n \log_k n)$
- Quadratic: $\Theta(n^2)$
- Polynomial: $\Theta(n^k)$
- Exponential: $\Theta(k^n)$

So sánh



So sánh

(a)

Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

Big-O notation

- Sử dụng các giới hạn để suy ra quan hệ "O lớn"

Nếu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \neq 0 \ (c < +\infty)$

$\Leftrightarrow$ O(f(n)) = O(g(n)) nghĩa là f(n) = O(g(n)) và g(n) = O(f(n)) (hoặc f(n) = Θ(g(n)))

Big-O notation

Nếu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < +\infty$

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

Nếu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$\Rightarrow O(f(n)) \subset O(g(n)) = O(g(n) \pm f(n))$

nghĩa là $f(n) = O(g(n))$ nhưng $g(n) \neq O(f(n))$

Tóm tắt

- $f(n) = O(g(n)) \Rightarrow f \preceq g$
- $f(n) = \Omega(g(n)) \Rightarrow f \succeq g$
- $f(n) = \Theta(g(n)) \Rightarrow f \approx g$

- Thuật toán có độ phức tạp $O(n^2)$ có thể không mất nhiều thời gian như vậy → có thể thật sự thời gian chạy là tuyến tính $O(n)$

- Thuật toán $\Omega(n\log n)$ có thể thật sự là $\Theta(2^n)$

- Thuật toán $\Theta(n^2)$ thì chính xác thời gian chạy là bậc 2

Tóm tắt

Limits

- $\lim_{n\to\infty} [f(n) / g(n)] = 0 \Rightarrow f(n) \in o(g(n))$
- $\lim_{n\to\infty} [f(n) / g(n)] < \infty \Rightarrow f(n) \in O(g(n))$
- $0 < \lim_{n\to\infty} [f(n) / g(n)] < \infty \Rightarrow f(n) \in \Theta(g(n))$
- $0 < \lim_{n\to\infty} [f(n) / g(n)] \Rightarrow f(n) \in \Omega(g(n))$
- $\lim_{n\to\infty} [f(n) / g(n)] = \infty \Rightarrow f(n) \in \omega(g(n))$
- $\lim_{n\to\infty} [f(n) / g(n)]$ undefined $\Rightarrow$ can't say

Cách đánh giá độ phức tạp

THUẬT TOÁN

KHÔNG ĐỆ QUY

- Kỹ thuật sơ cấp: phép đếm, tổng hữu hạn, xét dấu hàm
- Phụ thuộc tính chất dữ liệu vào: xấu nhất, tốt nhất, trung bình
- Ước lượng nhanh Big-O

ĐỆ QUY

- Tìm phương trình đệ quy
- Khử đệ quy

Một số thuật toán thông dụng

Suy ra Big-O khi biết T(n)

- Quy tắc xác định O: xét thành phần có bậc cao nhất của T(n)
  - $T(n) = 12n - 2 \in O(n)$
  - $T(n) = 3n^2\log(n) - 12n^2 + 19 \quad \in O(n^2\log(n))$
  - $T(n) = 3e^n - 10000n^2 + 2 \quad \in O(e^n)$

Ước lượng nhanh Big-O

Cho T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2

T1(n) = O(f(n))T2(n) = O(g(n))

– **Qui tắc cộng**: Thời gian thực hiện của hai đoạn chương trình P1 và P nối tiếp nhau là

$$T(n) = O(\max(f(n), g(n)))$$

– **Qui tắc nhân**: Thời gian thực hiện của hai đoạn chương trình P1 và P2 lồng nhau là

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

```
sum = 0;

for(i= 0; i < n; i++)
        sum = sum + i;

return sum;
```

P_1

P_2 Nối tiếp nhau

P_3

$$T(n) = O(max(f(n), g(n), h(n)))$$

```
for(i= 0; i < n; i++)
    if (a[max] < a[i])
        max = i;
```

P_1

P_2

Lồng nhau

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

Ví dụ 4:

sum = 0;	P_1	$T_1(n) = O(1)$
for(i= 0; i < n; i++) sum = sum + i;	P_2	$T_3(n) = O(n.1) = O(n)$
return sum;	P_3	$T_3(n) = O(1)$

$$T(n) = O(\max(1, n, 1)) = O(n)$$

Ước lượng nhanh Big-O

- Cách tìm T(n) ?

```
if (<điều kiện>)
{
        Thực hiện…;
}
else
{
        Thực hiện …;
        }
```

Ví dụ

```
/*1*/ s = 0;
/*2*/ i = 1;
/*3*/ while (i ≤ n) {
/*4*/        j  = n – i;
/*5*/        while (j ≥ 1) {
/*6*/                  s = s + 1;
/*7*/                  j = j – 1;
             }
/*8*/    i = i +1;
      }
```

Bài tập trên lớp: xác định O(?)

-

```
int Fifth_Element(int A[],int n) {
    return A[5];
}
int Partial_Sum(int A[],int n) {
    int sum=0;
    for(int i=0;i<42;i++)
        sum=sum+A[i];
    return sum;
}
```

T(n) = O (?)

Bài tập: xác định O(?)

-
```
void sum_first_n(int n) {
    int i,sum=0;
    for (i=1;i<=n;i++)
        sum = sum + i;
  }
void m_sum_first_n(int n) {
    int i,k,sum=0;
    for (i=1;i<=n;i++)
        for (k=1;k<7;k++)
            sum = sum + i;
  }
```

Bài tập: xác định O(?)

•

```
int *compute_sums(int A[], int n) {
    int M[n][n];
    int i,j;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            M[i][j]=A[i]+A[j];
    return M;
    }
```

Master Theorem (1): Dạng đơn giản

❖ Định lý Master cho phép ước lượng nghiệm của các phương trình đệ quy có dạng (*đơn giản*):

$$\begin{aligned} T(n) &= aT(\tfrac{n}{b}) + f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1, b \geq 2, c > 0$. *If* $f(n) \in \Theta(n^d)$ *where* $d \geq 0$, *then*

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Master Theorem (1)

❖ Ví dụ:

$$T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$$

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

a = 1, b = 2, d = 2

→ Vì 1 < 2^2 TH1 được áp dụng

$$T(n) \in \Theta(n^d) = \Theta(n^2)$$

Master Theorem: Ưu nhược điểm

❖ Nhược điểm:

- Chỉ áp dụng cho phương trình đệ quy có dạng như trên
- Không thể dùng định lý Master (dạng 1) nếu:

- T(n) không đơn điệu. VD: $T(n) = sin(n)$

- f(n) không phải hàm đa thức. VD: $T(n) = 2T\left(\frac{n}{2}\right) + 2^n$

- b không thể biểu diễn như 1 hằng số.

Master Theorem (2): Dạng tổng quát

❖ The Master Theorem applies to recurrence of the following form (general - tổng quát hơn 1):

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants

$f(n)$ is an asymptotically positive function

Master Theorem (2)

❖ Có 3 trường hợp:

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 $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k \geq 0$,

   then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$,

   and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$

Regularity condition:

$af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Master Theorem (2)

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 $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k \geq 0$,
   then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$,
   and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$

❖ Ví dụ: $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

a = 4, b = 2, và $n^{log_b a} = n^{log_2 4} = n^2$

Ta thấy, $f(n) = n^3 = \Omega(n^{2+\epsilon}), \epsilon = 0.5$

và $af(n/b) \leq cf(n)$

(vì $4\left(\frac{n}{2}\right)^3 = \frac{n^3}{2} \leq cn^3, c = 1/2$)

→áp dụng (2.3) ta được

$$T(n) = \Theta\big(f(n)\big) = \Theta(n^3)$$