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

L/O/G/O

Nội dung

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

Đặt vấn đề

- Giả sử bạn đang thiết kế 1 trang web để xử lý số liệu (VD số liệu tài chính). Ta có 2 chương trình có thể xem xét:

$T_1(n)=30n+8$

$T_2(n)=n^2+1$

Ban sẽ chọn chương trình nào nếu muốn phục vụ cho n = 1 000 000 người ?

So sánh 2 thuật toán

- Cách tốt nhất: xác định chính xác thời gian thực hiện T(n) (the exact running time) của 2 GT rồi so sánh chúng
- Thực tế:
  - Rất khó để tính T(n) chính xác

```
int BSearch_Recursion (int list[], int key, int left, int right)
{
   if (left <= right)
   {
         int mid = (left + right)/2;
         if (key == list[mid])
                  return mid;       // trả về vị trí tìm thấy key
         else if (key < list[mid])
                  return BSearch_Recursion (list, key, left, mid-1);
         else return BSearch_Recursion (list, key, mid+1, right);
   }
   return -1;     // không tìm thấy
}
```

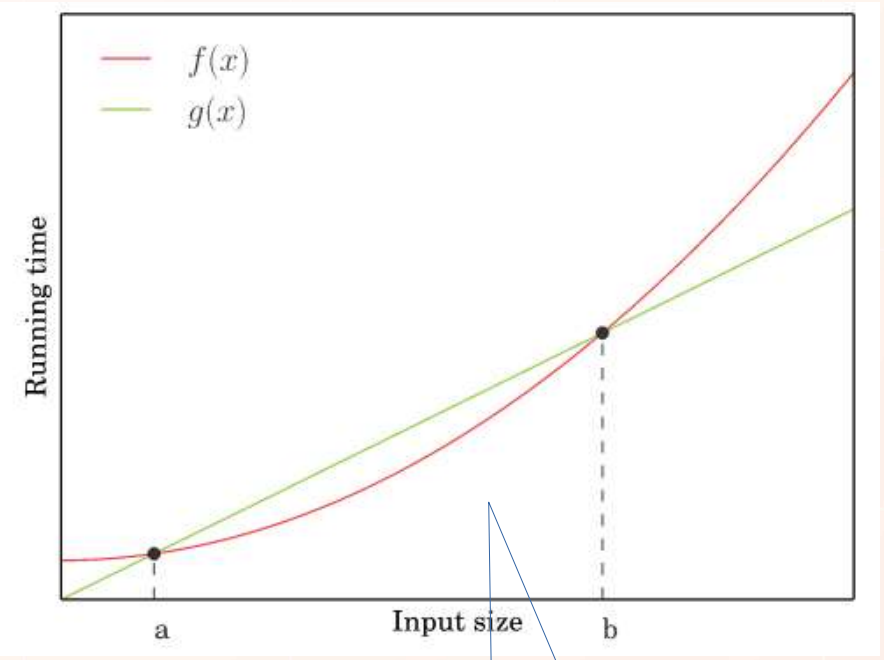
```
1   i = 1; count =0;
2   while ( i ≤ 4n)
3   {   x=(n-i)(i-3n) ;
4       y=i-2n;
5       j=1;
6       while (j  ≤ x )
7       {   count = count - 2;
8           j = j  + 2;
9       }
10      if (x>0)
11          if (y>0)
12              count = count +1;
13      i = i + 1;
14  }
```

Giải phương trình đệ quy để tìm T(n)

Dùng kỹ thuật toán "Xét dấu hàm" để xử lý if...else

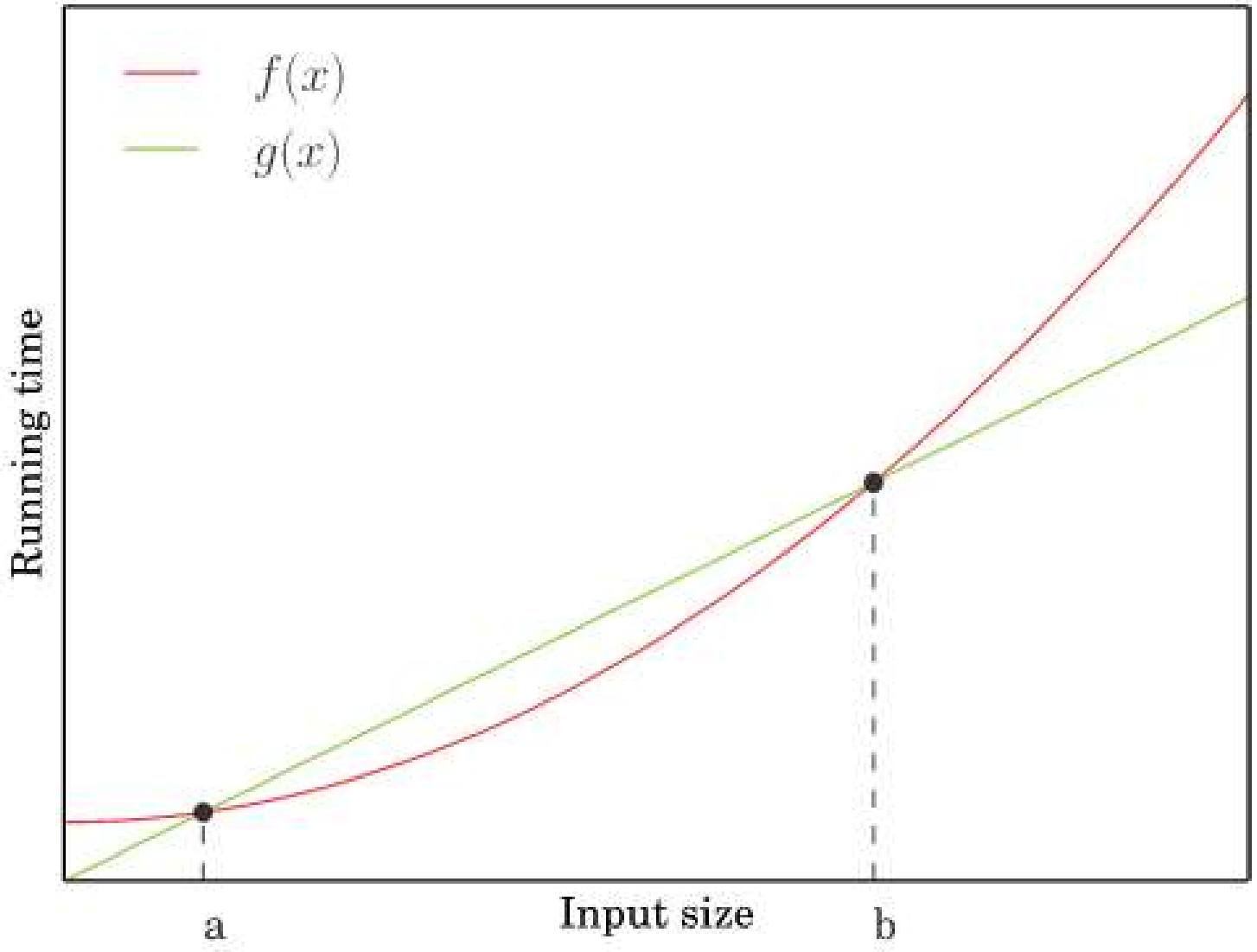
So sánh 2 thuật toán

- Cách tốt nhất: xác định chính xác thời gian thực hiện - T(n) của 2 GT rồi so sánh chúng
- Thực tế:
  - Rất khó để tính T(n) chính xác
  - Nếu tính được T(n) rồi thì có thể gặp khó khăn khác khi so sánh 2 hàm số với nhau



Nếu hàm T(n) đơn giản thì vẽ hình và quan sát hoặc xét dấu hàm

Compare 2 algorithms



$T_2(n)=n^2+1$

$T_1(n)=30n+8$

Đặt vấn đề

$T(n) = 3n^2\log(n) - 12n^2 + 19$

$T(n) = 3e^n - 10000n^2 + 2$

Giải thuật nào nhanh hơn?

- So sánh "độ lớn" của T1, T2 thay vì chính bản thân T1,T2

Compare 2 algorithms

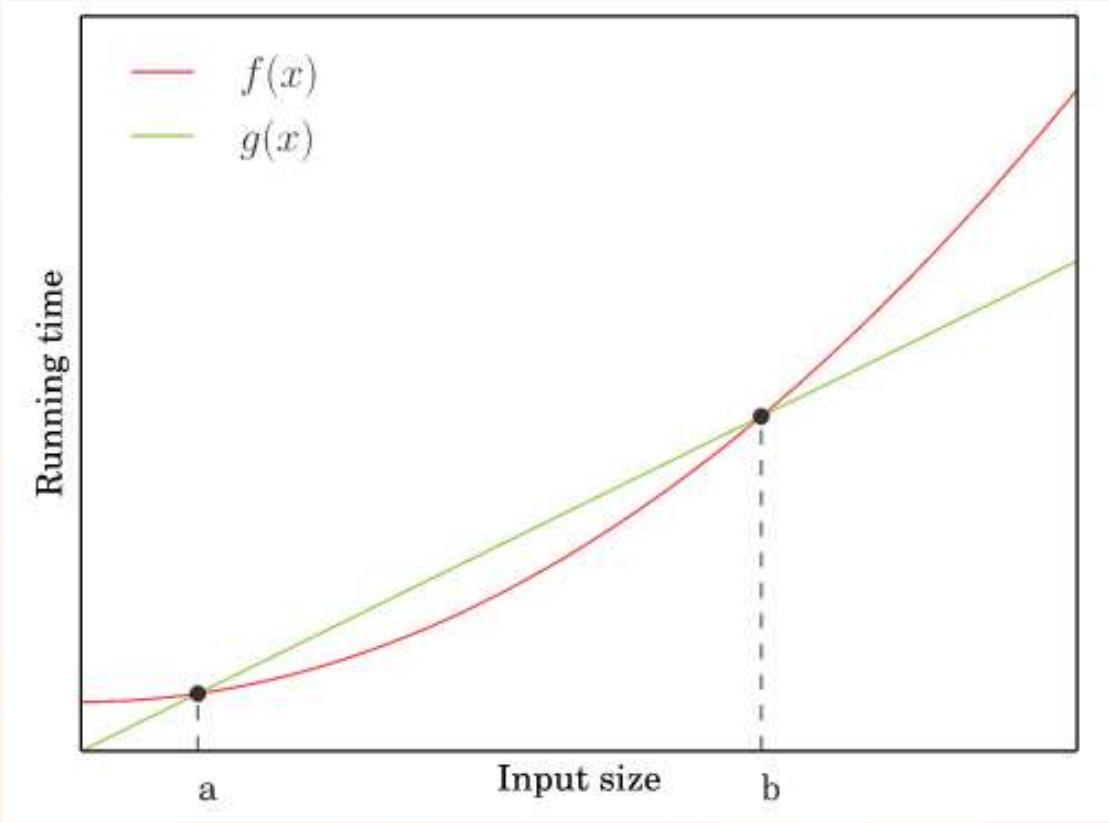
Mục tiêu mới:

- 1. So sánh tương đối: hàm không sai biệt nhiều thì xem như sấp xỉ nhau về độ lớn
- 2. Chỉ quan tâm đến những giá trị n đủ lớn -->lớn như thế nào?
- 3. Khi n càng lớn (tiến tới ∞) thì hàm nào sẽ lớn hơn?

Compare 2 algorithms

- Khi n càng lớn (tiến tới ∞) thì hàm nào sẽ lớn hơn?

- "nếu f (x) tăng nhanh hơn g(x) thì f(x) luôn lớn hơn g(x) ở ∞"



Cách đo tốc độ tăng của hàm số

- Khái niệm:
  - Growth rate: rate of growth of functions (tạm dịch: tỷ suất tăng)
  - Order of growth of the running time of an algorithm (tạm dịch: bậc tăng trưởng)
  - Hàm có bậc tăng trưởng lớn hơn $\rightarrow$ tăng nhanh hơn

Áp dụng trong phân tích TT

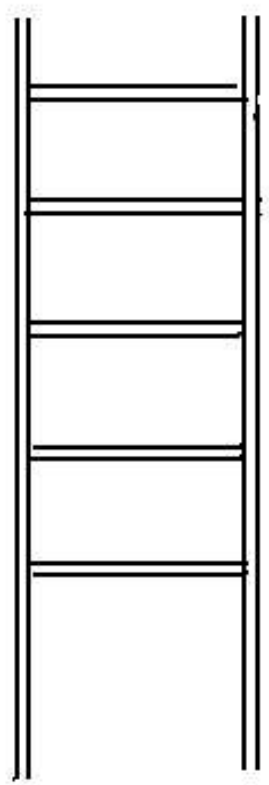
- Thường là quan tâm đến "order of growth" của T(n) thay vì chính xác bản thân T(n)
- Order of growth:

Ước lượng "cấp độ lớn" của T(n) dựa trên những hàm f(n) đã biết

Ví dụ:

$1{,}000{,}001 \approx 1{,}000{,}000$

$3n^2 + 5 \approx n^2$

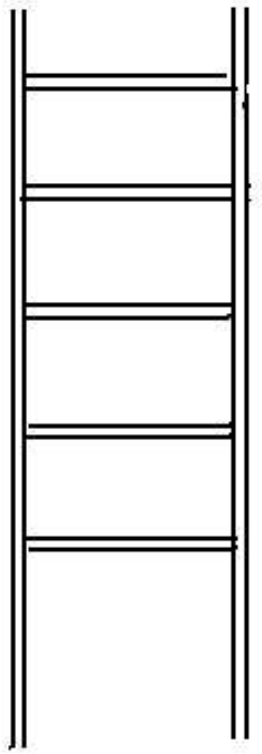


Aymptotic Notation

- Ký hiệu/ký pháp tiệm cận: O, Ω, Θ
  - "Nâng cấp" của bậc tăng trưởng
  - Cho ta 1 phương pháp để phân loại các hàm "chặt chẽ hơn" dựa theo bậc tăng trưởng của chúng

Độ phức tạp

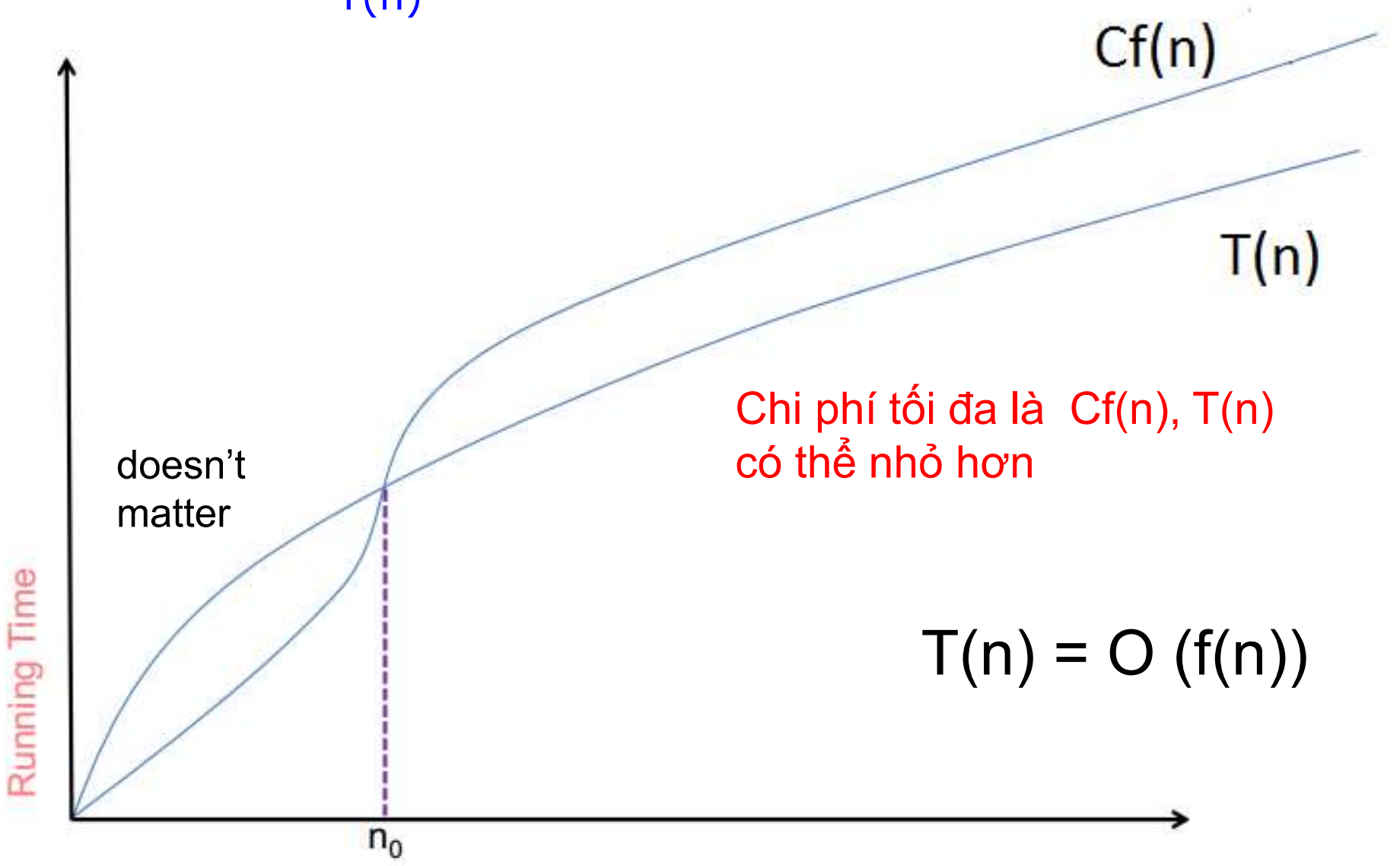
- Ý nghĩa chung:
  - Phân lớp "cấp độ lớn" của hàm T(n) khi n đủ lớn
  - GT nào có độ phức tạp ở phân lớp thấp hơn thì hiệu quả hơn



- Độ phức tạp của GT: được xác định thông qua các ký hiệu tiệm cận O, Ω, Θ --> có 3 loại độ phức tạp phổ biến

Big-O

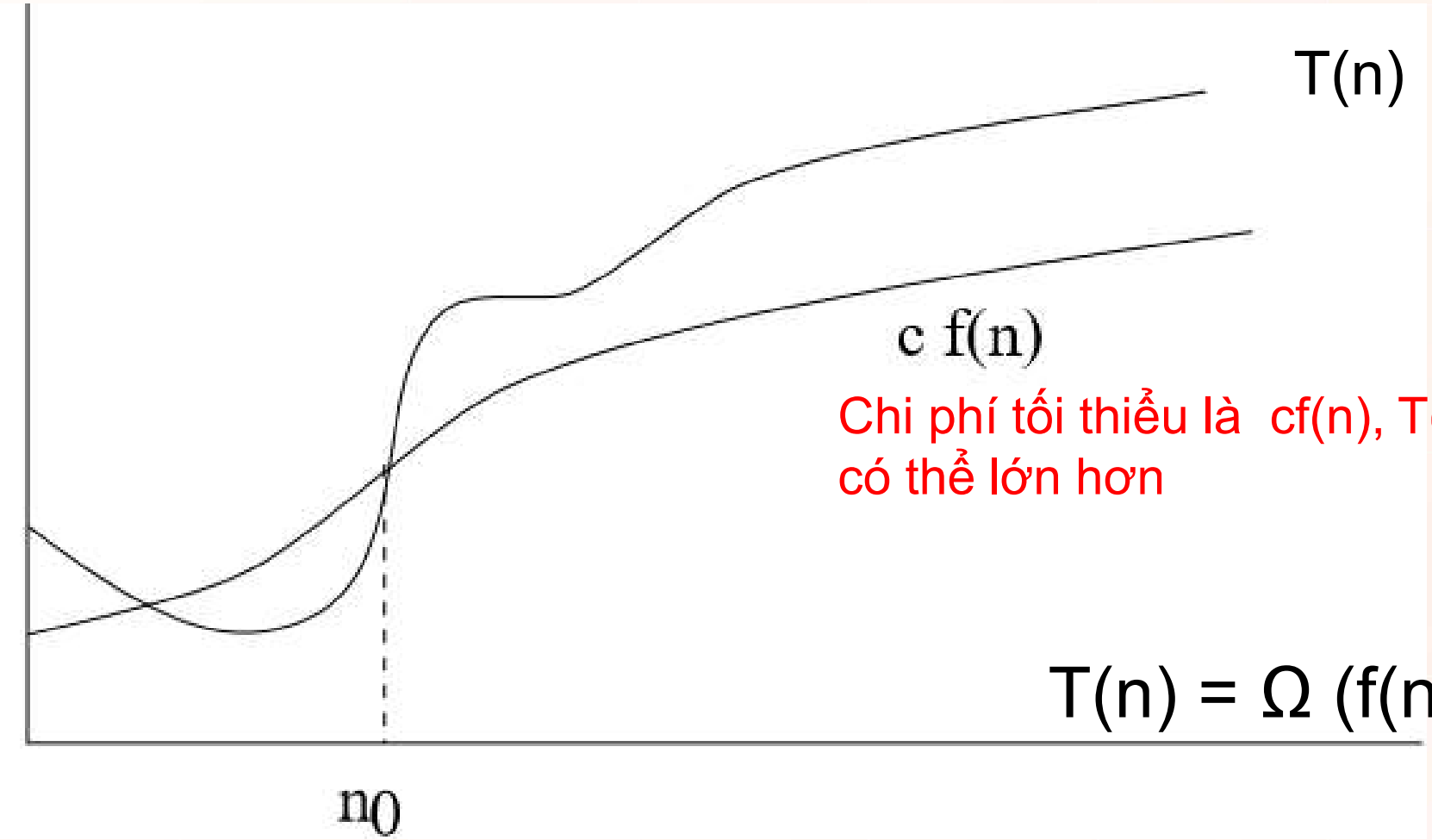
- Khi n tăng, T(n) tăng không nhanh hơn Cf(n), Cf(n) là chặn trên (asymptotic upper bound) của T(n)



Chi phí tối đa là Cf(n), T(n) có thể nhỏ hơn

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

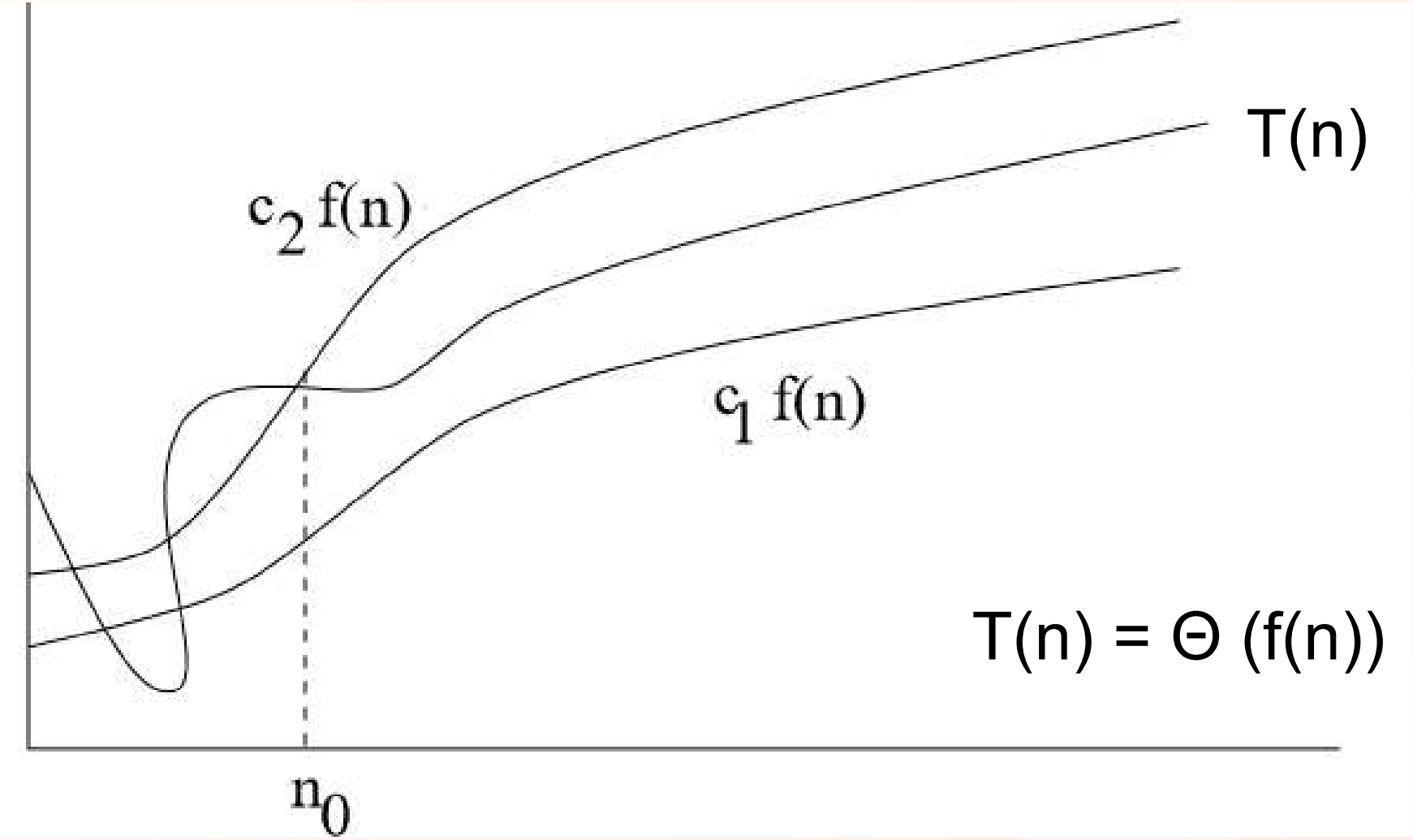
Big- Ω



Chi phí tối thiểu là cf(n), T(n) có thể lớn hơn

$T(n) = \Omega(f(n))$

Big-Θ



Big-O notation (upper bound)

- Định nghĩa (phân tích GT):
  - T(n) = O(f(n)) nếu tồn tại hằng số dương $c \in \mathbb{R}^+$ và $n_0 \in \mathbb{N}$ sao cho:

$$T(n) \leq cf(n) \text{ với } \forall\ n \geq\ n_0$$

Ta nói: T(n) có bậc tăng trưởng là f(n) , T(n) có độ phức tạp là O của f(n)

Big-O notation

- Ký hiệu: T(n) = O(f(n)) . Bản chất: $T(n) \in O(f(n))$
- Chú ý:
  - Dấu = chỉ là ký hiệu hình thức
  - O(f(n)) là tập hợp

$$O(f(n)) = \{t : N^* \rightarrow N^* \mid \exists\, c \in R^+ , \exists n_0 \in N, \forall\, n \geq n_0 , t(n) \leq cf(n)\}$$

O(f(n)) = {t(n) : ∃ positive constants c and n_0, such that ∀n ≥ n_0, we have 0 ≤ t(n) ≤ cf(n) }

Big-O notation

- Lưu ý:
  - Có thể đánh giá ĐPT bằng ký hiệu tiệm cận khác như Ω, Θ → nên kèm theo ký hiệu khi nói đến ĐPT. VD: "GT có ĐPT $O(n^2)$"
  - f(n) chỉ là 1 hàm chặn trên của T(n), vẫn có thể có cách ước lượng chặt hơn
  - Luôn tìm được f(n) và cần tìm f(n) nhỏ nhất có thể

Big-O notation

- Ta thấy: Với $T(n) = 10n$
  - $T(n) = O(n)$, $T(n) = O(n^2)$, $T(n) = O(n^3)$
  - $10n \in O(n)$? $O(n^2)$? $O(n^3)$ (Hỏi: dùng ký hiệu gì thay cho ? là đúng)

- Ví dụ 1: Xét $T(n) = 3n^2 + 5n + 4$. Tìm f(n) để $T(n) = O(f(n))$?

Ta có:

$T(n) = 3n^2 + 5n + 4 \leq 3n^2 + 5n^2 + 4n^2$, $\forall\ n \geq 1$

$\Rightarrow T(n) \leq 12n^2\ \forall\ n \geq 1.$

Chọn $c=12$, $n_0 = 1$, theo định nghĩa của Big-O, ta có dccm, $T(n) = O(n^2)$

Big-O notation

- Ví dụ 2: $n^2 + n = O(n^3)$

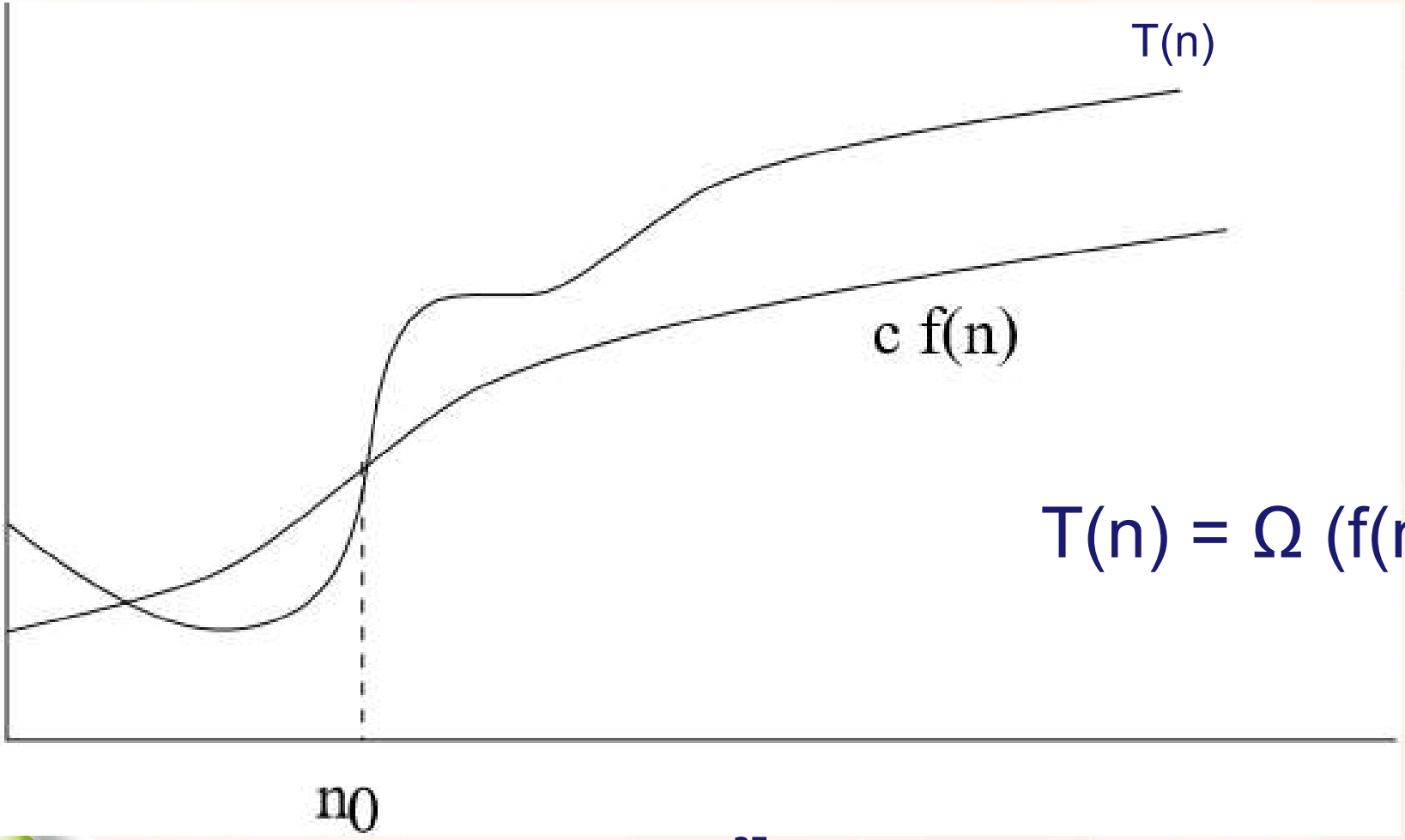
Chứng minh:

- Dễ thấy rằng, khi $n \geq 1$ thì $n \leq n^3$ và $n^2 \leq n^3$

  (Tổng quát, nếu $a \leq b$, thì $n^a \leq n^b$ khi $n \geq 1$)

- Vì vậy, $n^2 + n \leq n^3 + n^3 = 2n^3$
- Nên, $n^2 + n \leq 2n^3$ với mọi $n \geq 1$
- Theo định nghĩa của Big-O, xét $c = 2$ và $n_o = 1$, ta có đccm

Big- Ω notation



$T(n) = \Omega(f(n))$

Big-Ω notation

- Định nghĩa:

$T(n) = \Omega(f(n))$ nếu và chỉ nếu tồn tại các hằng số dương $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$ sao cho:

$$T(n) \geq c.f(n) \text{ với } \forall\ n \geq n_0$$

Ký hiệu Tiệm cận Ω

Định nghĩa:

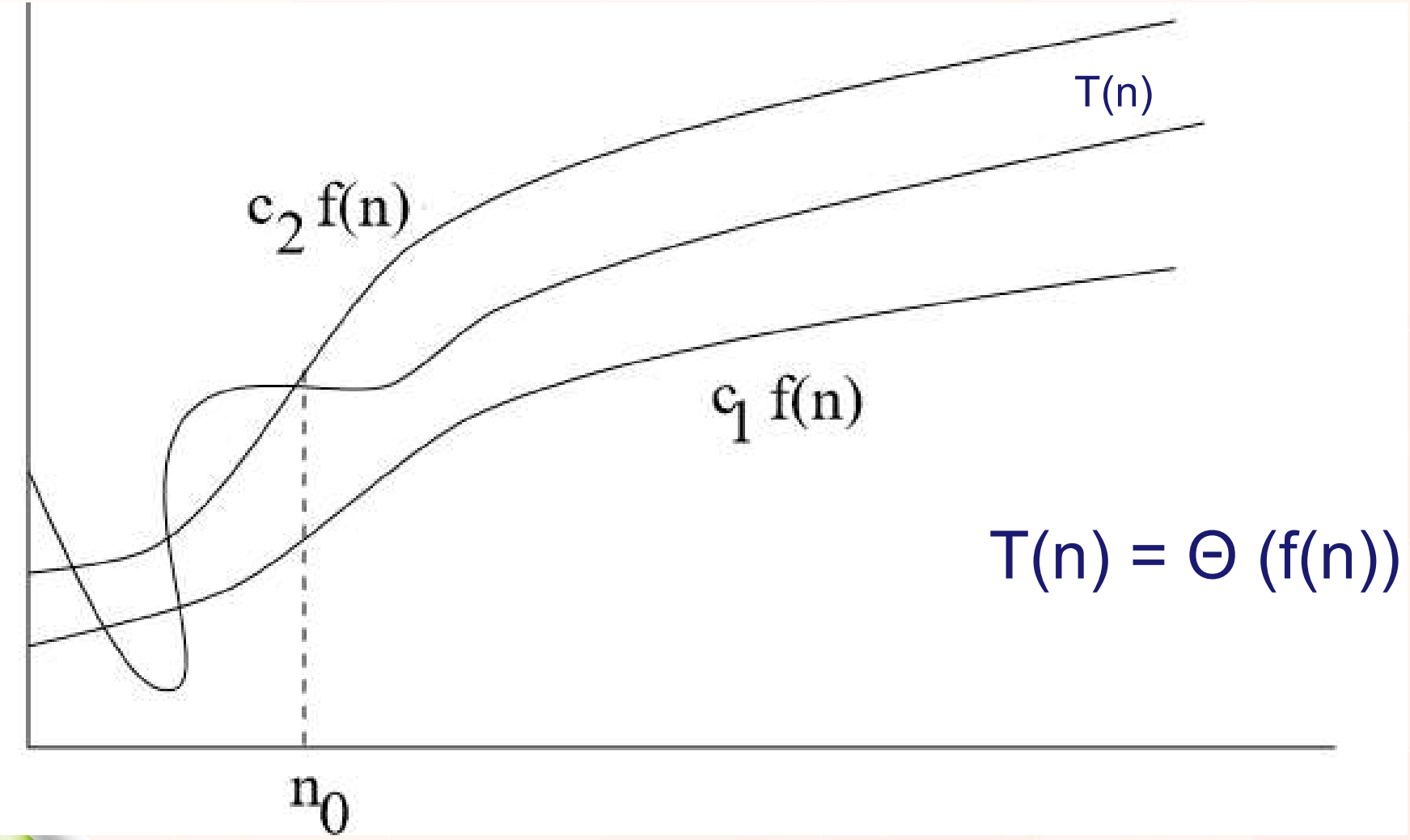
$$\Omega(f(n))=\{t: N \rightarrow N^* \mid \exists c \in R^+ , \exists n_0 \in N, \forall n \geq n_0 , t(n) \geq cf(n)\}$$

Tính chất:

Cho f, g : $N \rightarrow N^*$, Ta có:

$$f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega (f(n))$$

Big-Θ notation



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

Big-Θ notation

- Định nghĩa:

$T(n) = \Theta(f(n))$ nếu và chỉ nếu tồn tại các hằng số dương $c_1, c_2 \in \mathbb{R}^+$ và $n_0 \in \mathbb{N}$ sao cho:

$$c_1 f(n) \leq T(n) \leq c_2 f(n) \text{ với } \forall n \geq n_0$$

Ký hiệu Tiệm cận Θ

Định nghĩa:

$f(n) = \Theta(g(n))$ nếu và chỉ nếu

$f(n) = O(g(n))$ và $g(n) = O(f(n))$

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

o-notation and ω-notation

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

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

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

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

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

Tóm tắt

Asymptotic notations

$O(g(n))$= $\{f(n)$: there **exist** positive constants c and n_0 such that $0 \le f(n) \le c \cdot g(n)$ for **all** $n \ge n_0\}$, e.g. $8 \lg n = O(n)$.

$\Omega(g(n))$= $\{f(n)$: there **exist** positive constants c and n_0 such that $0 \le c \cdot g(n) \le f(n)$ for **all** $n \ge n_0\}$, e.g. $26n^7 + 2013 = \Omega(n^5)$.

$\Theta(g(n))$= $\{f(n)$: there **exist** positive constants c_1, c_2 and n_0 such that $0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$ for **all** $n \ge n_0\}$, e.g. $5 \cdot 2^n + n^6 - 10^{10} = \Theta(2^n)$.

$o(g(n))$= $\{f(n)$: for **any** positive constant c, there **exists** a constant n_0 such that $0 \le f(n) < c \cdot g(n)$ for **all** $n \ge n_0\}$.
If $g(n) > 0$ for large n, this means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, e.g. $10^5 \cdot 2^n = o(3^n)$.

$\omega(g(n))$= $\{f(n)$: for **any** positive constant c there **exists** a constant n_0 such that $0 \le cg(n) < f(n)$ for **all** $n \ge n_0\}$.
If $g(n) > 0$ for large n, this means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, e.g. $2^n = \omega(n^3)$.

Dùng lim để suy ra quan hệ

- Sử dụng các giới hạn để suy ra quan hệ

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)))

Dùng lim để suy ra quan hệ

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

Dùng lim để suy ra quan hệ

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

Classification of algorithms

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 Chấp nhận được
$O(n^2)$	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 $O(2^n)$

Vét cạn, lớn hơn rất nhiều

KHÔNG CHẤP NHẬN ĐƯỢC

So sánh

(a)

	n					
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}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

So sánh