

Bài 1

GIỚI THIỆU

CÁC VẤN DỀ



- **Thuật toán**
 Khái niệm
 Đặc trưng
- **Độ phức tạp thuật toán**
 Cơ sở toán học
 Tính toán độ phức tạp thuật toán
- **Tiếp cận giải quyết bài toán**
 Các bước tiếp cận, giải quyết thuật toán
 Xu hướng tiếp cận, giải quyết bài toán

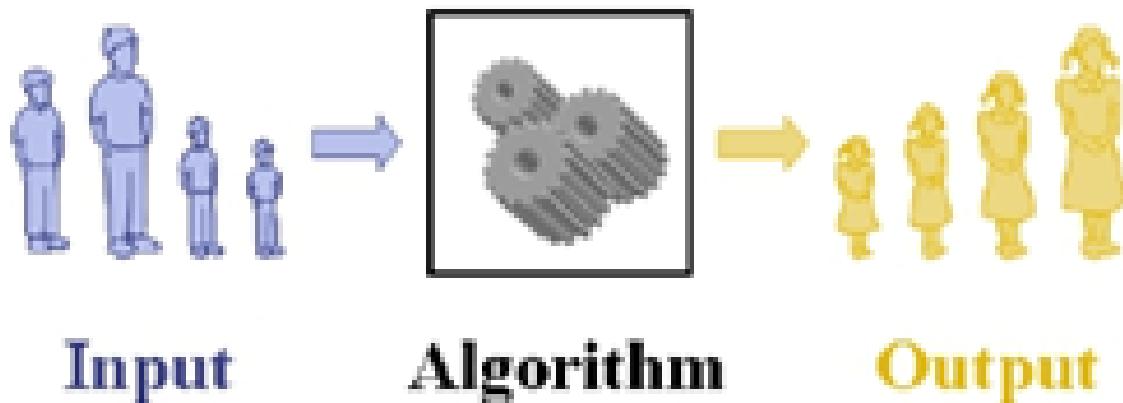
GIẢI THUẬT (THUẬT TOÁN)



- **Khái niệm thuật toán**

Định nghĩa:

Một thuật toán là một bản liệt kê các chỉ dẫn, các quy tắc cần thực hiện theo từng bước xác định nhằm giải quyết một bài toán đã cho trong một khoảng thời gian hữu hạn.



Ví dụ: 2.1 Mô tả thuật toán tìm số lớn nhất trong một dãy hữu hạn các số nguyên.

1. Đặt giá trị cực đại tạm thời bằng số nguyên đầu tiên trong dãy;
2. So sánh số nguyên tiếp theo với giá trị cực đại tạm thời , nếu lớn hơn giá trị cực đại tạm thời thì đặt giá trị cực đại tạm thời bằng số nguyên đó.
3. Lặp lại bước 2) nếu còn các số nguyên trong dãy.
4. Giá trị cực đại tạm thời ở thời điểm này chính là số nguyên lớn nhất trong dãy.



- Mô tả một thuật toán cần chú ý các yếu tố sau:

Dữ liệu đầu vào: Một thuật toán phải mô tả rõ các giá trị đầu vào từ một tập hợp các dữ liệu xác định.

Ví dụ: dãy số nguyên $a(1), a(2), \dots, a(n)$, với $n < \infty$;
hai số nguyên dương a , và b ;...

Dữ liệu đầu ra: Từ một tập các giá trị đầu vào, thuật toán sẽ tạo ra các giá trị đầu ra. Các giá trị đầu ra chính là nghiệm của bài toán.

Ví dụ: số max là phần tử lớn nhất trong $a(1), \dots, a(n)$;
số d là ước chung lớn nhất của a và b ;...

Thuật toán



1. *Tính xác định*: Các bước của thuật toán phải được xác định một cách chính xác, các chỉ dẫn phải rõ ràng, có thể thực hiện được.
2. *Tính hữu hạn*: Thuật toán phải kết thúc sau một số hữu hạn bước.
3. *Tính đúng đắn*: Thuật toán phải cho kết quả đúng theo yêu cầu của bài toán đặt ra.
4. *Tính tổng quát*: Thuật toán phải áp dụng được cho mọi bài toán cùng loại, với mọi dữ liệu đầu vào như đã được mô tả.

Thuật toán



Ta xét thuật toán nêu trong ví dụ trên:
Dữ liệu đầu vào: mảng các số nguyên;
Dữ liệu đầu ra: số nguyên lớn nhất của mảng đầu vào;
Tính xác định: Mỗi bước của thuật toán chỉ gồm các phép gán, mệnh đề kéo theo;
Tính hữu hạn: Thuật toán dừng sau khi tất cả các thành phần của mảng đã được kiểm tra;

Thuật toán



Tính đúng đắn: Sau mỗi bước kiểm tra và so sánh ta sẽ tìm được số lớn nhất trong các số đã được kiểm tra. Rõ ràng, sau lần kiểm tra cuối cùng thì xác định được số lớn nhất trong toàn bộ các số đã được kiểm tra, có nghĩa là toàn bộ dãy.

Tính tổng quát: Thuật toán cho phép tìm số lớn nhất của dãy số nguyên hữu hạn n bất kỳ.

Cấu trúc dữ liệu

- Kiểu dữ liệu trừu tượng (Abstract Data Type)
 - Là mô hình toán học và những phép toán thực hiện trên mô hình toán học này
 - Ví dụ: ADT List
 - Dữ liệu: Các nút
 - Các phép toán:
 - Đầu tiên
 - Loại bỏ một nút
 - Tìm kiếm một nút có giá trị cho trước

Cấu trúc dữ liệu

● Cấu trúc dữ liệu

- Sử dụng để biểu diễn mô hình toán học trong ADT
- Việc cài đặt các kiểu dữ liệu trừu tượng đòi hỏi phải chọn các cấu trúc dữ liệu để biểu diễn
- Liên quan đến cách thức tổ chức và truy nhập các phần tử dữ liệu
- Ví dụ: ADT List
 - Cài đặt sử dụng cấu trúc mảng đơn giản
 - Cài đặt sử dụng cấu trúc con trỏ

Xây dựng chương trình giải bài toán

- Lời giải một bài toán bao gồm
 - Cấu trúc dữ liệu
 - Thuật toán
- Xây dựng chương trình giải bài toán
 - Tương tự như vòng đời của phần mềm
 - Gồm các bước
 - Thu thập yêu cầu: Hiểu rõ đầu vào và kết quả đầu ra
 - Thiết kế : Xây dựng giải thuật, bỏ qua các chi tiết về cách thức cài đặt dữ liệu hay các phương thức, tập trung vào các bước xử lý
 - Phân tích : Tìm, so sánh với giải thuật khác
 - Cài đặt: Xây dựng chương trình, quan tâm đến cách thức tổ chức, biểu diễn và cài đặt các phương thức
 - Kiểm thử : Bao gồm chứng minh tính đúng đắn của chương trình, kiểm thử các trường hợp , tìm, sửa lỗi

Thuật toán và độ phức tạp

- Đánh giá lượng tài nguyên các loại mà một giải thuật đã sử dụng.
 - Giải thuật này thực hiện trong thời gian thế nào → Phân tích về thời gian thực hiện giải thuật
 - Giải thuật này sử dụng bao nhiêu bộ nhớ → Phân tích độ không gian nhớ mà giải thuật (chương trình) cần có.

Phân tích thời gian thực hiện giải thuật

- Mục tiêu của việc xác định thời gian thực hiện một giải thuật:
 - Để ước lượng một chương trình sẽ thực hiện trong bao lâu
 - Để ước lượng kích thước dữ liệu đầu vào lớn nhất có thể cho một giải thuật
 - Để so sánh hiệu quả của các giải thuật khác nhau, từ đó lựa chọn ra một giải thuật thích hợp cho một bài toán
 - Để giúp tập trung vào đoạn giải thuật được thực hiện với thời gian lớn nhất

Phân tích thời gian thực hiện giải thuật

● Cách thức

- Xác định độ phụ thuộc của thời gian tính của thuật toán vào kích thước của dữ liệu đầu vào
- Các phương pháp thực hiện
 - Phương pháp thực nghiệm
 - Phương pháp phân tích dựa trên mô hình lý thuyết

Phân tích thời gian thực hiện giải thuật

- Phương pháp thực nghiệm

- Cài đặt giải thuật bằng ngôn ngữ lập trình
 - Chạy chương trình với các dữ liệu đầu vào khác nhau
 - Đo thời gian thực thi chương trình và đánh giá độ tăng trưởng so với kích thước của dữ liệu đầu vào

- Hạn chế:

- Sự hạn chế về số lượng và chất lượng của mẫu thử
 - Đòi hỏi môi trường kiểm thử (phần cứng và phần mềm) thống nhất, ổn định

Phân tích thời gian thực hiện giải thuật

- Phương pháp lý thuyết

- Có khả năng xem xét dữ liệu đầu vào bất kỳ
- Sử dụng để đánh giá các giải thuật mà không phụ thuộc vào môi trường kiểm thử
- Sử dụng với những mô tả ở mức cao của giải thuật

- Thực hiện phương pháp này cần quan tâm

- Ngôn ngữ mô tả giải thuật
- Xác định độ đo thời gian tính
- Một cách tiếp cận để khái quát hóa độ phức tạp về thời gian

Mô tả giải thuật – Giả ngôn ngữ

- Giả ngôn ngữ (Pseudo-code)

- Mô tả mức khái quát cao được sử dụng trong diễn tả giải thuật

Giả ngôn ngữ = Cấu trúc lập trình + Ngôn ngữ tự nhiên

Algorithm arrayMax(A,n)

Input: Mảng chứa n phần tử là số nguyên

Output: Phần tử lớn nhất trong mảng

Begin

currentMax = A[0]

for i = 1 to n-1 do

 if currentMax < A[i] then currentMax = A[i]

return currentMax

End.

Giả ngôn ngữ

- Các cấu trúc lập trình trong giả ngôn ngữ
 - Câu lệnh gán: $V = E$ hoặc $V \leftarrow E$
 - Cấu trúc điều khiển:
 - if B then S_1 [else S_2]
 - Case
 - $B_1 : S_1 ;$
 - $B_2 : S_2 ;$
 - ...
 - $B_n : S_n$
 - else S_{n+1}
 - end case;

Giả ngôn ngữ

- Câu lệnh lặp

- Vòng lặp với số lần lặp biết trước
for i = m to n do S hoặc for i = n down to m do S
- Với số lần lặp không biết trước
while B do S hoặc repeat S until B

- Câu lệnh vào ra

- Đọc dữ liệu vào
read (<danh sách biến>);
- Ghi dữ liệu
write (<danh sách biến hoặc dòng ký tự>);

Giả ngôn ngữ

- Khai báo hàm

Function <tên hàm> (<danh sách tham số>)

Begin

<các câu lệnh>

return (giá trị)

End

- Gọi hàm: *Hàm được gọi bằng tên hàm cùng danh sách giá trị tham số thực sự, nằm trong biểu thức*

Giả ngôn ngữ

Function AVERAGE(A,n)

Begin

{A là một mảng gồm n phần tử là số nguyên. Giải thuật trả ra giá trị trung bình của các giá trị trong mảng}

1. sum = 0;
2. {Duyệt mảng} for I = 1 to n do
 sum = sum + A[i];
3. average = sum/n
4. return(average)

End.

Giả ngôn ngữ

- Khai báo thủ tục

Procedure <tên thủ tục> (<danh sách tham số>)

Begin

<các câu lệnh>

End

- Thủ tục được gọi bằng cách sử dụng câu lệnh
Call <tên thủ tục> (<danh sách giá trị tham số>)

Phân tích thời gian thực hiện giải thuật

- Độ đo thời gian tính sử dụng trong phương pháp phân tích lý thuyết
 - Phép toán cơ bản là phép toán có thể được thực hiện với thời gian bị chặn bởi một hằng số không phụ thuộc vào kích thước dữ liệu
- Thời gian tính của giải thuật được xác định bằng cách đếm số phép toán cơ bản mà giải thuật thực hiện

$$T(n) = c_{op} \cdot C(n)$$

Phân tích thời gian thực hiện giải thuật

- Các phép toán cơ bản thường dùng
 - Gán giá trị cho biến số
 - Gọi hàm hay thủ tục
 - Thực hiện các phép toán số học
 - Tham chiếu vào mảng
 - Trả kết quả
 - Thực hiện các phép so sánh

Phân tích thời gian thực hiện giải thuật

- Dòng 1: 2 phép toán cơ bản
- Dòng 2: phép gán giá trị đầu cho i, phép so sánh $i < n$
được thực hiện n lần
 - Thân vòng lặp thực hiện $n-1$ lần, trong thân, tối thiểu phải thực hiện phép so sánh (2 phép toán cơ bản) , tăng i lên 1 (2 phép toán cơ bản) tối đa phải có thêm phép gán (2 phép toán cơ bản)
 - Dòng 3: 1 phép toán cơ bản
Tổng số phép toán cơ bản trong Trường hợp xấu nhất : $7n-2$

Function ARRAY-MAX(A,n)

Đầu vào : mảng A gồm n phần tử.

Đầu ra: phần tử lớn nhất trong mảng

Begin

1. currentMax = A[0]

2. for i = 1 to n-1 do

if currentMax < A[i] then
currentMax = A[i]

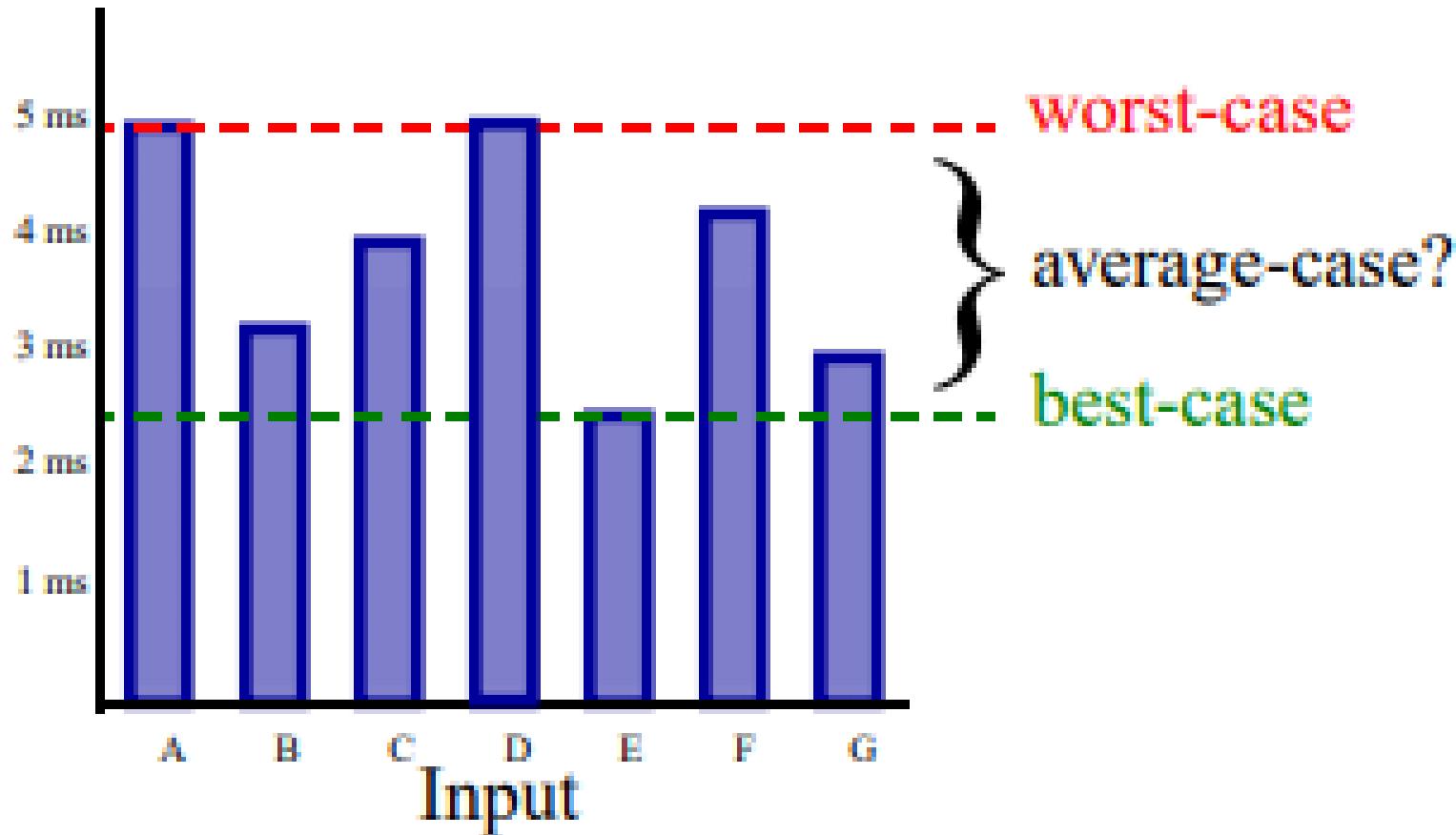
3. return currentMax

End.

Phân tích thời gian thực hiện giải thuật

- Thời gian tính tồi nhất (Worst-case)
 - Thời gian nhiều nhất để thực hiện thuật toán với một bộ dữ liệu vào kích thước n
- Thời gian tính tốt nhất (Best-case)
 - Thời gian ít nhất để thực hiện thuật toán với một bộ dữ liệu cũng với kích thước n
- Thời gian tính trung bình (Average case)
 - Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước n

Phân tích thời gian thực hiện giải thuật



Phân tích thời gian thực hiện giải thuật

- Ví dụ : Tìm kiếm tuần tự một giá trị trên một mảng

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]
4	8	7	10	21	14	22	36	62	91	77	81

- Thời gian xấu nhất : n
- Thời gian tốt nhất : 1
- Thời gian trung bình: $T(n) = \sum i \cdot p_i$
trong đó p_i là xác suất giá trị cần tìm xuất hiện tại $a[i]$. $p_i = 1/n$ thì thời gian sẽ là $(n+1)/2$

Đánh giá độ phức tạp thuật toán



Cho hai hàm số f và g , $f: \mathbb{R} \rightarrow \mathbb{R}$, $g: \mathbb{R} \rightarrow \mathbb{R}$.

Trong phần này bàn đến sự so sánh độ tăng của hai hàm $f(x)$ và $g(x)$ khi $x \rightarrow +\infty$.

1. Định nghĩa

Định nghĩa 1.1. Ta nói rằng $f(x) = o(g(x))$ khi x dần tới dương vô cùng, nếu như $\lim_{x \rightarrow +\infty} f(x)/g(x) = 0$.

Khi này người ta nói rằng $f(x)$ tăng chậm hơn so với $g(x)$ khi x lớn dần đến $+\infty$.

Ví dụ 1.1.

$$x^2 = o(x^5)$$

$$\sin(x) = o(x)$$

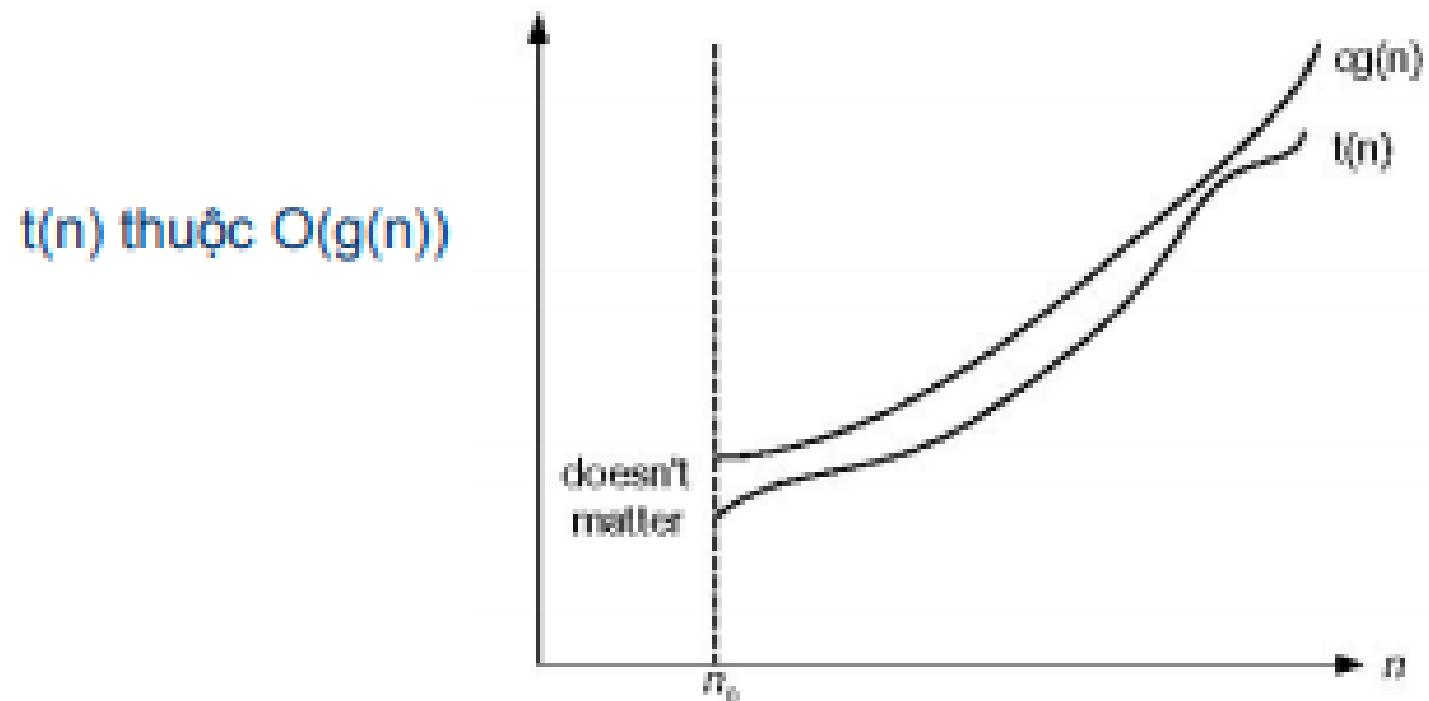
$$1/x = o(1)$$

Ký hiệu tiệm cận

- Khái niệm Big-O

- Cho hàm số $t(n)$ và $g(n)$, ta nói rằng $t(n)$ là $O(g(n))$ nếu tồn tại 2 hằng số nguyên dương c và n_0 sao cho

$$t(n) \leq cg(n) \text{ for } n \geq n_0$$



Đánh giá độ phức tạp thuật toán

Định nghĩa 1.3. Ta nói rằng $f(x)$ tương đương với $g(x)$ khi x dần tới dương vô cùng,

kí hiệu $f(x) \approx g(x)$, nếu như $\lim_{x \rightarrow +\infty} f(x)/g(x) = 1$.

Ví dụ 1.5.

$$1+x+x^2 \approx x^2,$$

$$(2x+4)^2 \approx 4x^2.$$

Đánh giá độ phức tạp thuật toán



Mệnh đề 1.1.

Cho $f(x) = a_0 + a_1x^1 + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$, trong đó a_i , $i=0,1,\dots,n$, là các số thực.

Khi đó $f(x) = O(x^n)$.

Chứng minh:

Kí hiệu $C = |a_0| + |a_1| + |a_2| + \dots + |a_{n-1}| + |a_n|$.

Với $x > 1$ ta có $x^k < x^n$, với $k < n$, suy ra

$$\begin{aligned}|f(x)| &= |a_0 + a_1x^1 + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n| \\&\leq |a_0| + |a_1x^1| + |a_2x^2| + \dots + |a_{n-1}x^{n-1}| + |a_nx^n| = |a_0| \\&\quad + |a_1| \cdot x + |a_2| \cdot x^2 + \dots + |a_{n-1}| \cdot x^{n-1} + |a_n| \cdot x^n \leq (|a_0| + \\&\quad |a_1| + |a_2| + \dots + |a_{n-1}| + |a_n|) \cdot x^n = C \cdot x^n. (\text{đpcm})\end{aligned}$$

Đánh giá độ phức tạp thuật toán



Ví dụ 1.6. Đánh giá tổng n số tự nhiên đầu tiên

$$S(n) = 1 + 2 + \dots + n < n + n + \dots + n = n^2.$$

Vậy $S(n) = O(n^2)$.

Nhận xét:

Số mũ 2 trong $O(n^2)$ đã phải là nhỏ nhất hay chưa?

Cũng như vậy, biểu thức n^2 đã phải là nhỏ nhất hay chưa?

Việc đánh giá hàm trong O -lớn cũng như bậc của hàm càng sát càng tốt.

Ta có nhận xét rằng nếu tồn tại các hằng số N, C_1 và C_2 sao cho bắt đầu từ $x > N$ ta có $C_1 \cdot g(x) \leq f(x) \leq C_2 \cdot g(x)$ thì rõ ràng là đánh giá $O(g(x))$ đối với $f(x)$ được coi là khá chính xác. Trong trường hợp này người ta còn nói rằng $f(x)$ và $g(x)$ là cùng bậc.

Độ tăng của hàm



- Ví dụ: Đánh giá hàm $f(n) = n!$

Quy ước:

$$0! = 1$$

$$1! = 1;$$

$$3! = 1 \cdot 2 \cdot 3 = 6;$$

$$5! = 120;$$

$$10! = 362,880;$$

$$11! = 39,916,800;$$

$$20! = 2,432,902,008,176,640,000$$

Rõ ràng $n! < n^n$. Điều này chứng tỏ $n! = O(n^n)$.

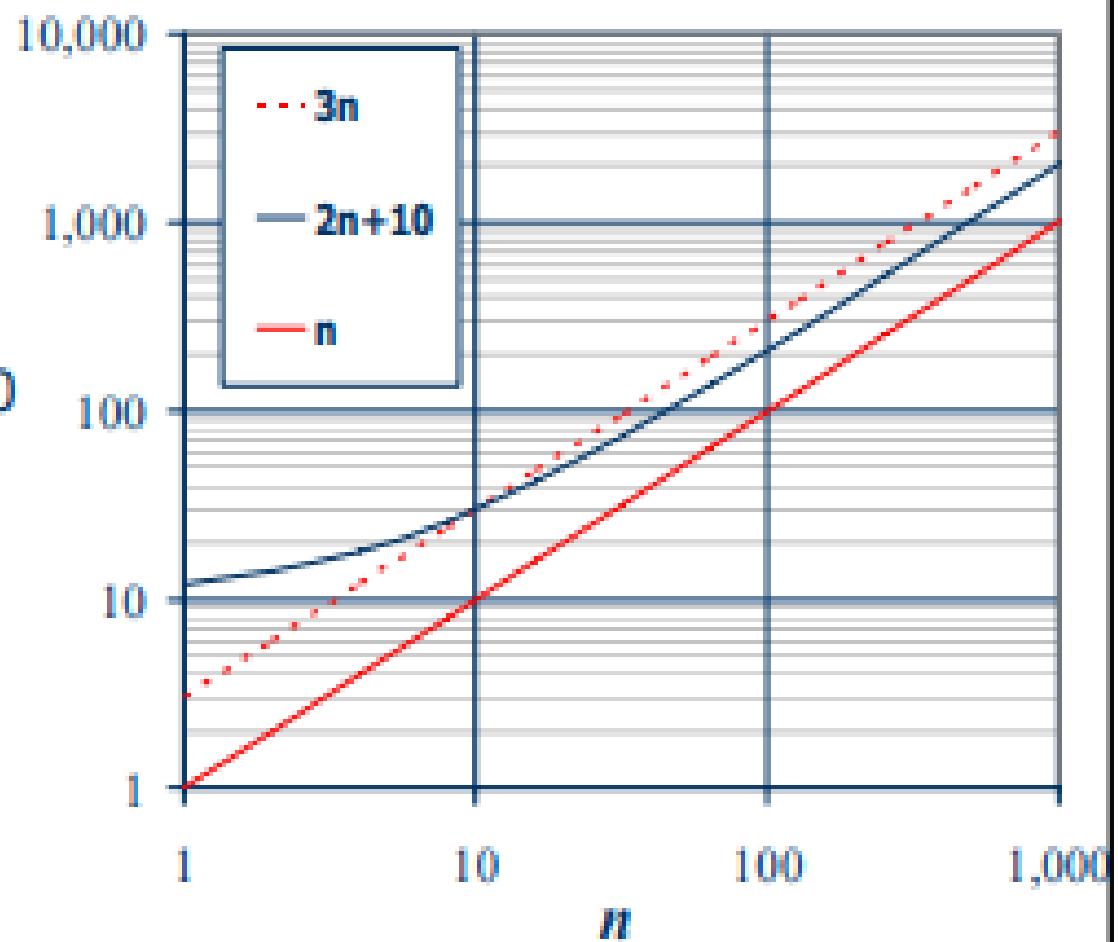
Từ đó suy ra $\log(n!) = O(n \log n)$.

Ký hiệu tiệm cận Big - O

- $7n^2$
 - $7n^2$ là $O(n)$
tìm $c > 0$ và $n_0 \geq 1$ sao cho $7n^2 \leq c^*n$ với $n \geq n_0$
điều này đúng với $c = 7$ và $n_0 = 1$
- $3n^3 + 20n^2 + 5$
 - $3n^3 + 20n^2 + 5$ là $O(n^3)$
tìm $c > 0$ và $n_0 \geq 1$ sao cho $3n^3 + 20n^2 + 5 \leq c^*n^3$ với $n \geq n_0$
điều này đúng với $c = 4$ và $n_0 = 21$
- $3 \log n + 5$
 - $3 \log n + 5$ là $O(\log n)$
cần $c > 0$ và $n_0 \geq 1$ sao cho $3 \log n + 5 \leq c^*\log n$ với $n \geq n_0$
ta xác định được $c = 8$ và $n_0 = 2$

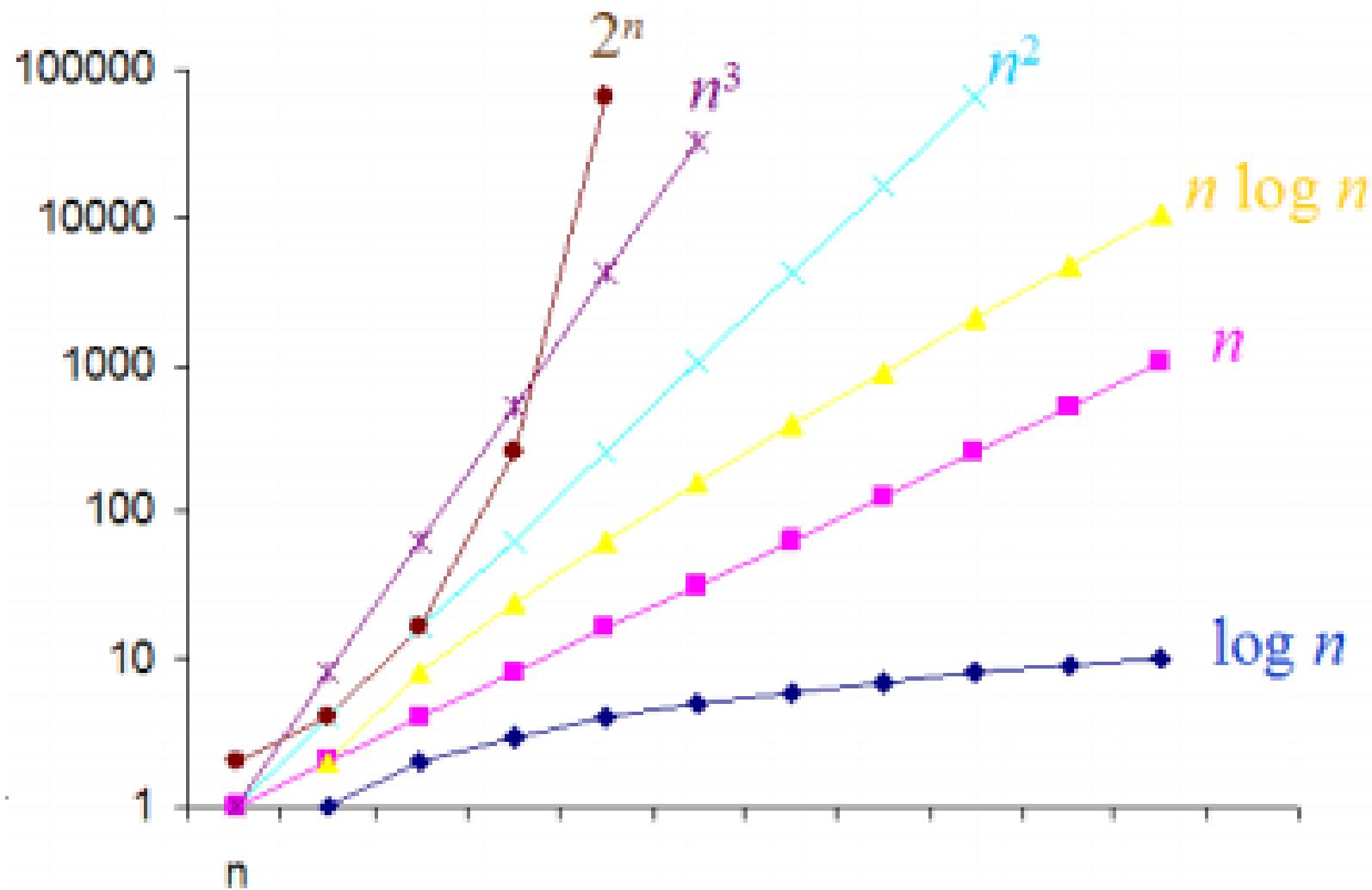
Ký hiệu tiệm cận Big - O

- Ví dụ: Giải thuật có $T(n) = 2n + 10$ thì có độ phức tạp là $O(n)$
 - $2n + 10 \leq cn$
 - $(c - 2)n \geq 10$
 - $n \geq 10/(c - 2)$
 - Lấy $c = 3$ và $n_0 = 10$



Ký hiệu tiệm cận Big - O

- Đồ thị một số hàm cơ bản



Ký hiệu tiệm cận Big - O

- Big-O và độ tăng trưởng

- Big-O là ký hiệu tiệm cận trên của một hàm
- Nếu ta có $T(n)$ là $O(g(n))$ thì độ tăng trưởng của $T(n)$ không vượt quá độ tăng trưởng của $g(n)$

Ký hiệu tiệm cận Big - O

- Qui tắc xác định độ phức tạp về thời gian
 - Hàm thời gian $T(n)$ của một đoạn của thuật toán là đa thức bậc k thì $T(n)$ là $O(n^k)$
 - $n^x = O(a^n)$, với bất kỳ $x > 0$ và $a > 1$
 - $\log n^x = O(\log n)$, với $x > 0$

Ký hiệu tiệm cận Big - O

- Qui tắc xác định độ phức tạp
 - Cấu trúc tuần tự - Qui tắc tổng
 - Cho 2 đoạn của thuật toán P_1 và P_2 với thời gian thực hiện tương ứng là $T_1(n)$ và $T_2(n)$. Thời gian thực hiện P_1 và P_2 kế tiếp nhau là: $T_1(n) + T_2(n)$
 - Độ phức tạp của hai đoạn chương trình P_1 và P_2 liên tục nhau có thể xác định là $O(\max(f(n), g(n)))$ nếu $T_1(n) = O(f(n))$ và $T_2(n) = O(g(n))$.

Xác định độ phức tạp tính toán của giải thuật

❖ Qui tắc bỏ hằng số:

- Đoạn chương trình P có thời gian thực hiện

$$T(n) = O(c_1 \cdot f(n))$$

c_1 : là một hằng số dương

$\Rightarrow P$ có độ phức tạp $O(f(n))$

• **Chứng minh**

$T(n) = O(c_1 \cdot f(n))$ nên $\exists c_0 > 0$ và $\exists n_0 > 0$ để $T(n) \leq c_0 \cdot c_1 \cdot f(n)$

Với $\forall n \geq n_0$

Đặt $C = c_0 \cdot c_1$. Theo đ/n , ta có: $T(n) = O(f(n))$

Xác định độ phức tạp tính toán của giải thuật

❖ Qui tắc lấy max:

- Đoạn chương trình P có thời gian thực hiện

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

$$\Rightarrow P \text{ có độ phức tạp } O(\max(f(n), g(n)))$$

Chứng minh

$T(n) = O(f(n) + g(n))$ nên $\exists c > 0$ và $\exists n_0 > 0$ để

$T(n) \leq c \cdot f(n) + c \cdot g(n)$, với $\forall n \geq n_0$

Mà $T(n) \leq c \cdot f(n) + c \cdot g(n) \leq 2c \cdot (\max(f(n), c \cdot g(n)))$.

Theo đ/n , ta có: $T(n) = O(\max(f(n), c \cdot g(n)))$

Xác định độ phức tạp tính toán của giải thuật 57

❖ Qui tắc lấy công:

- Đoạn chương trình P_1 có thời gian thực hiện
 $T_1(n) = O(f(n))$
- Đoạn chương trình P_2 có thời gian thực hiện
 $T_2(n) = O(g(n))$
⇒ Thời gian thực hiện P_1 rồi đến P_2 sẽ là:

$$T_1(n) + T_2(n) = O(f(n)) + O(g(n))$$

có độ phức tạp $O(\max(f(n), g(n)))$

Chứng minh

$T(n) = O(f(n) + g(n))$ nên $\exists c > 0$ và $\exists n_0 > 0$ để

$T(n) \leq c \cdot f(n) + c \cdot g(n)$, với $\forall n \geq n_0$

Mà $T(n) \leq c \cdot f(n) + c \cdot g(n) \leq 2c \cdot (\max(f(n), g(n)))$.

Theo đ/n, ta có: $T(n) = O(\max(f(n), g(n)))$

Ký hiệu tiệm cận Big - O

- Qui tắc xác định độ phức tạp
 - Cấu trúc lồng - Quy tắc nhân
 - Cho 2 đoạn chương trình P_1 và P_2 với thời gian thực hiện tương ứng là $T_1(n)$ và $T_2(n)$. Thời gian thực hiện P_1 và P_2 lồng vào nhau là: $T_1(n)T_2(n)$
 - Độ phức tạp của hai đoạn chương trình P_1 và P_2 liên tục nhau có thể xác định là $O(f(n)*g(n))$ nếu $T_1(n) = O(f(n))$ và $T_2(n) = O(g(n))$.

Đánh giá độ phức tạp thuật toán



Ví dụ 1.5. Tìm đánh giá hàm

$$f(n) = n \log(n!) + (3n^2 + 2n)\log n.$$

Theo các ví dụ đã nêu ở trên ta có

$\log(n!) = O(n \log n)$, từ đó dễ dàng suy ra rằng

$$n \log(n!) = O(n^2 \log n);$$

Mặt khác ta có $3n^2 + 2n = O(n^2)$, hay suy ra

$$(3n^2 + 2n) \log n = O(n^2 \log n).$$

Cuối cùng ta có: $f(n) = O(n^2 \log n)$

Đánh giá độ phức tạp thuật toán



Ví dụ 1.6. Tìm đánh giá đối với hàm

$$f(n) = (n+3) \log(n^2+4) + 5n^2.$$

Ta có các đánh giá $n+3 = O(n)$

và $\log(n^2+4) = O(\log n)$.

Thật vậy, đánh giá thứ nhất là hiển nhiên.

Xét đánh giá thứ hai. Rõ ràng là với $n > 2$ ta có $\log(n^2+4) < \log(2n^2) < \log 2 + \log n^2 = \log 2 + 2\log n < 3\log n$.

Từ đây suy ra $(n+3) \log(n^2+4) = O(n \log n)$.

Ngoài ra ta có $5n^2 = O(n^2)$. Từ đây suy ra
 $f(n) = O(\max \{ n \log n, n^2 \}) = O(n^2)$.

Đánh giá độ phức tạp thuật toán



Ví dụ 1.7.

Tìm đánh giá tốt nhất của hàm $f(x) = 2^x + 23$.

Hiển nhiên là với mọi $x > 5$ ta có $f(x) < 2 \times 2^x$.

Vì vậy $f(x) = O(2^x)$.

Ta cũng dễ thấy được là $2^x < f(x)$ với mọi $x > 0$.

Vậy $O(2^x)$ là đánh giá tốt nhất đối với $f(x)$

(hay nói cách khác 2^x là cùng bậc với $f(x)$).

Độ phức tạp thuật toán



- Tính hiệu quả của thuật toán thông thường được đo bởi thời gian tính (thời gian được sử dụng để tính bằng máy hoặc bằng phương pháp thủ công) khi các giá trị đầu vào có kích thước xác định. Tính hiệu quả của thuật toán cũng được xem xét theo thước đo dung lượng bộ nhớ đã sử dụng để tính toán khi kích thước đầu vào đã xác định.
- Hai thước đo đã nêu ở trên liên quan đến độ phức tạp tính toán của một thuật toán, được gọi là độ phức tạp thời gian và độ phức tạp không gian (còn gọi là độ phức tạp dung lượng nhớ).

Độ phức tạp thuật toán



- Trong phần đầu, chúng ta sẽ chỉ đề cập đến độ phức tạp thời gian của một thuật toán. Độ phức tạp thời gian của một thuật toán thường được biểu diễn thông qua số phép toán trong khi thực hiện thuật toán khi các giá trị dữ liệu đầu vào có **kích thước** xác định.
- Người ta thường dùng số phép tính làm thước đo độ phức tạp thời gian thay cho thời gian thực của máy tính là vì các máy tính khác nhau thực hiện các phép tính sơ cấp (so sánh, cộng, trừ, nhân, chia các số nguyên) trong những khoảng thời gian khác nhau.

Độ phức tạp thuật toán



- Chúng ta cũng sẽ không thực hiện việc phân rã các phép toán sơ cấp nêu trên thành các phép toán bit sơ cấp mà máy tính sử dụng vì việc này là khá phức tạp.
- Một cách đánh giá được sử dụng ở đây là cách đánh giá khả năng xấu nhất của thuật toán.

Độ phức tạp thuật toán



- Định nghĩa 1
- Một thuật toán được gọi là có **độ phức tạp đa thức**, hay còn gọi là **có thời gian đa thức**, nếu số các phép tính cần thiết khi thực hiện thuật toán không vượt quá $O(n^k)$, với k nguyên dương nào đó, còn n là kích thước của dữ liệu đầu vào.
- Các thuật toán với $O(k^n)$, trong đó n là kích thước dữ liệu đầu vào, còn k là một số nguyên dương nào đó gọi là **các thuật toán có độ phức tạp hàm mũ hoặc thời gian mũ**.

Độ phức tạp thuật toán



Ví dụ 2.4. Xét thuật toán tìm kiếm phần tử lớn nhất trong dãy số nguyên cho trước

Dữ liệu vào (input): $a[1..n]$, a là mảng các số nguyên, $n > 0$ là số các số trong mảng a ;

Dữ liệu ra (output): \max , số lớn nhất trong mảng a ;

```
int TimMax(a: mảng các số nguyên);
```

```
max = a[1];
```

```
for i:2 -> n
```

```
    if (max < a[i] )
```

```
        max = a[i];
```

```
return max;
```

Độ phức tạp thuật toán



- Vì các phép toán được dùng ở đây là các phép toán so sánh sơ cấp như ta đã nêu ở trên nên ta sẽ dùng số các phép toán sơ cấp này để đo độ phức tạp của thuật toán. Ta dễ dàng thấy được số các phép toán so sánh sơ cấp được sử dụng ở đây là $2(n-1)$. Vì vậy ta nói rằng độ phức tạp của thuật toán nói trên là $O(n)$ (còn nói là độ phức tạp tuyến tính).

Độ phức tạp thuật toán



Ví dụ 2.5. Xét độ phức tạp của thuật toán tìm kiếm tuyến tính. Mảng $a[1..n]$, tìm vị trí xuất hiện phần tử x .

```
int TK_TT(a:mảng số nguyên; x: số nguyên)
```

```
    i      =1;
```

```
    While (i<=n and x≠a[i])
```

```
        i= i+1;
```

```
    if (i<=n ) index=i
```

```
    else index=-1;
```

Độ phức tạp thuật toán



- Trong mỗi bước của vòng lặp có 2 phép toán so sánh được thực hiện: một để xem đã tới cuối bảng hay chưa và một để so sánh số nguyên x với một phần tử trong bảng.
- Số bước của thuật toán trong trường hợp xấu nhất là n . Sau cùng là một phép so sánh chỉ số i sau cùng của vòng lặp với số n .
- Vậy tổng số phép so sánh được sử dụng là $2n+1$.
- Độ phức tạp của thuật toán so sánh tuyến tính (tuần tự) là $O(n)$.

Độ phức tạp thuật toán



- Ví dụ 2.6. Xét độ phức tạp của thuật toán tìm kiếm nhị phân.
- Mảng $a[1..n]$ đã được sắp xếp, vị trí xuất hiện x .

```
int TimKiem_NP(a:mảng số nguyên; x: số nguyên)
```

- 1. $first = 1; last = n;$
- 2. $found = false;$
- 3. While ($first \leq last$ and not $found$)
 - $index = (first + last) \text{ div } 2;$
 - If ($x = a(index)$) $found = true$
 - else if ($x < a(index)$) $last = index - 1$
 - else $first = index + 1;$
- 4. If (not $found$) $index := -1;$
- 5. return $index;$

Độ phức tạp thuật toán



- Ví dụ 2.6. Xét độ phức tạp của thuật toán tìm kiếm nhị phân.
- Giả thiết rằng có $n=2^k$ phần tử.
- Ta dễ dàng thấy được số phép toán so sánh tối đa là $2k+1 = 2\log_2 n$.
- Hay độ phức tạp $O(\log n)$, độ phức tạp logarit.

Độ phức tạp thuật toán



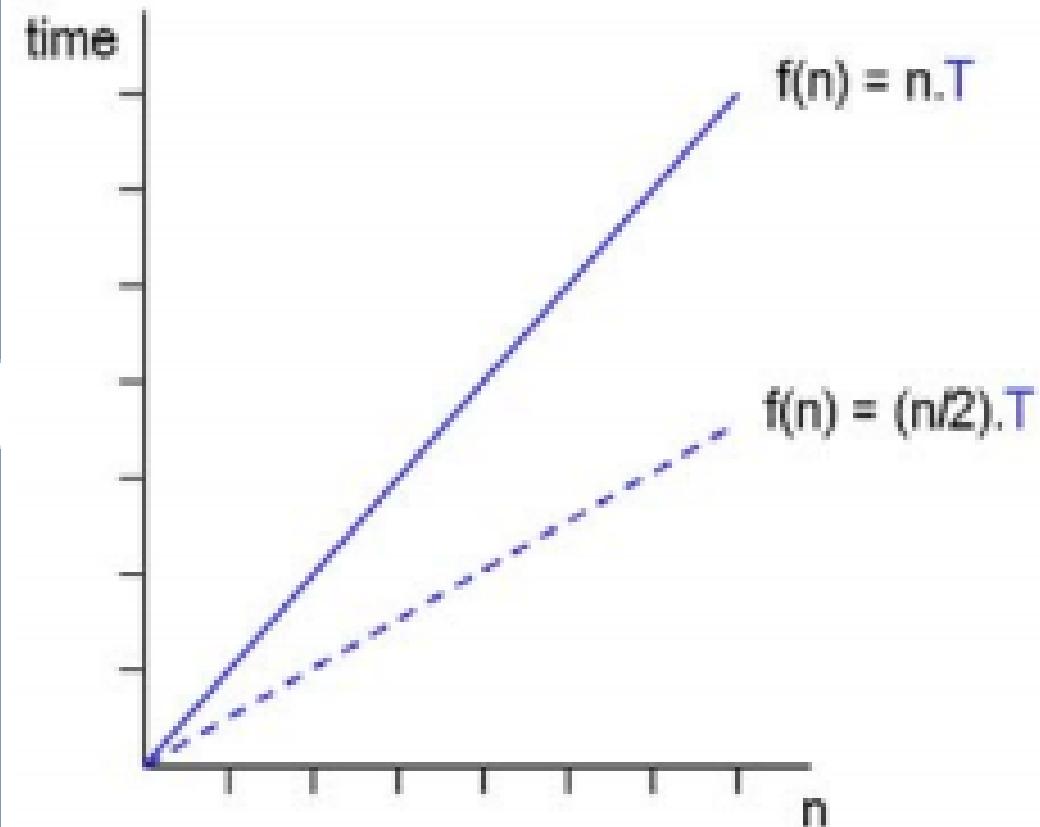
Một vài loại thường gặp

- | | |
|-----------------|--------------------------|
| $O(1)$ | Độ phức tạp hằng số. |
| $O(\log n)$ | Độ phức tạp logarit. |
| $O(n)$ | Độ phức tạp tuyến tính. |
| $O(n^k)$ | Độ phức tạp đa thức. |
| $O(n \log n)$ | Độ phức tạp $n \log n$. |
| $O(b^n), b > 1$ | Độ phức tạp hàm mũ |
| $O(n!)$ | Độ phức tạp giai thừa |

Ký hiệu tiệm cận Big - O

```
for i = 1 to n
begin
    P; {đoạn giải thuật với thời
        gian thực hiện T}
end
```

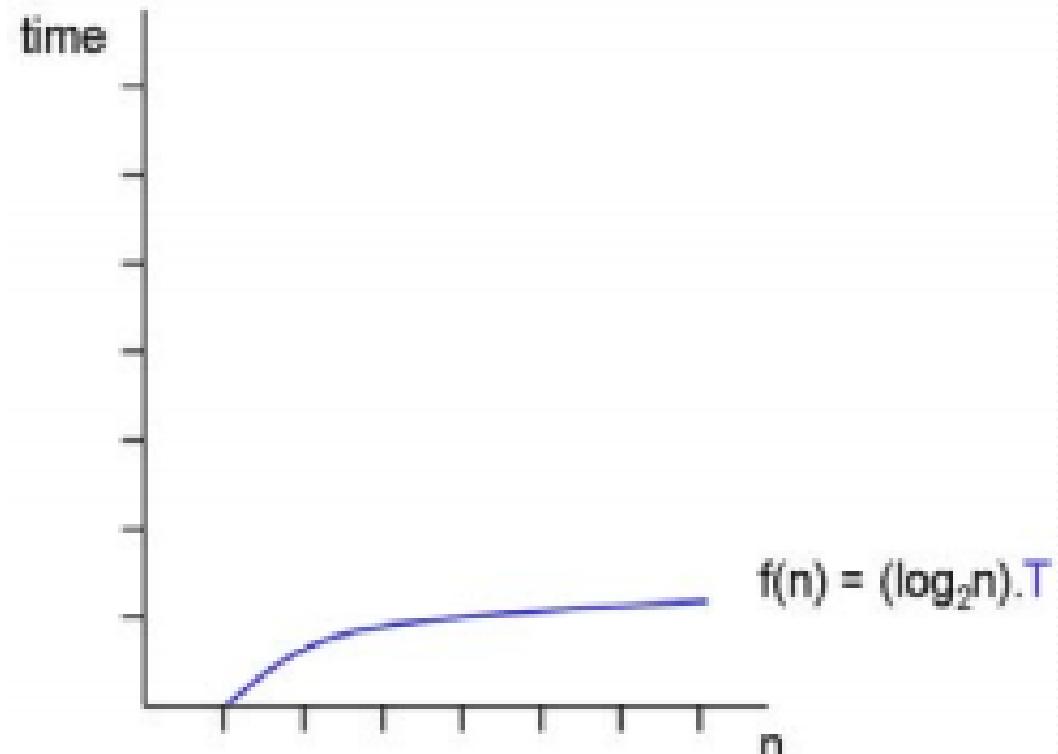
```
i := 1
while (i <= n) do
begin
    P; {đoạn giải thuật với thời
        gian thực hiện T}
    i := i+2;
end
```



Ký hiệu tiệm cận Big - O

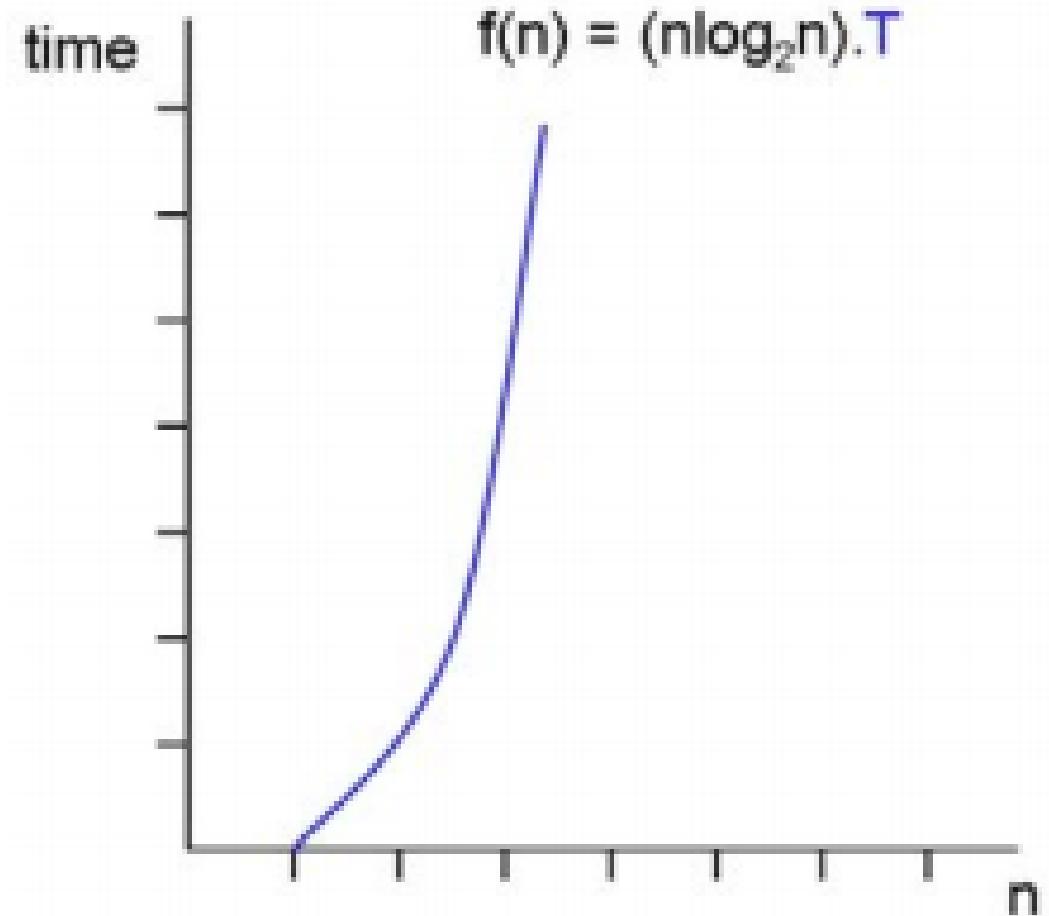
```
i := 1  
while (i <= n) do  
begin  
    P; {đoạn giải thuật với thời  
        gian thực hiện T}  
    i := i * 2;  
end
```

```
i := n  
while (i >= 1) do  
begin  
    P; {đoạn giải thuật với thời  
        gian thực hiện T}  
    i := i / 2  
end
```



Ký hiệu tiệm cận Big - O

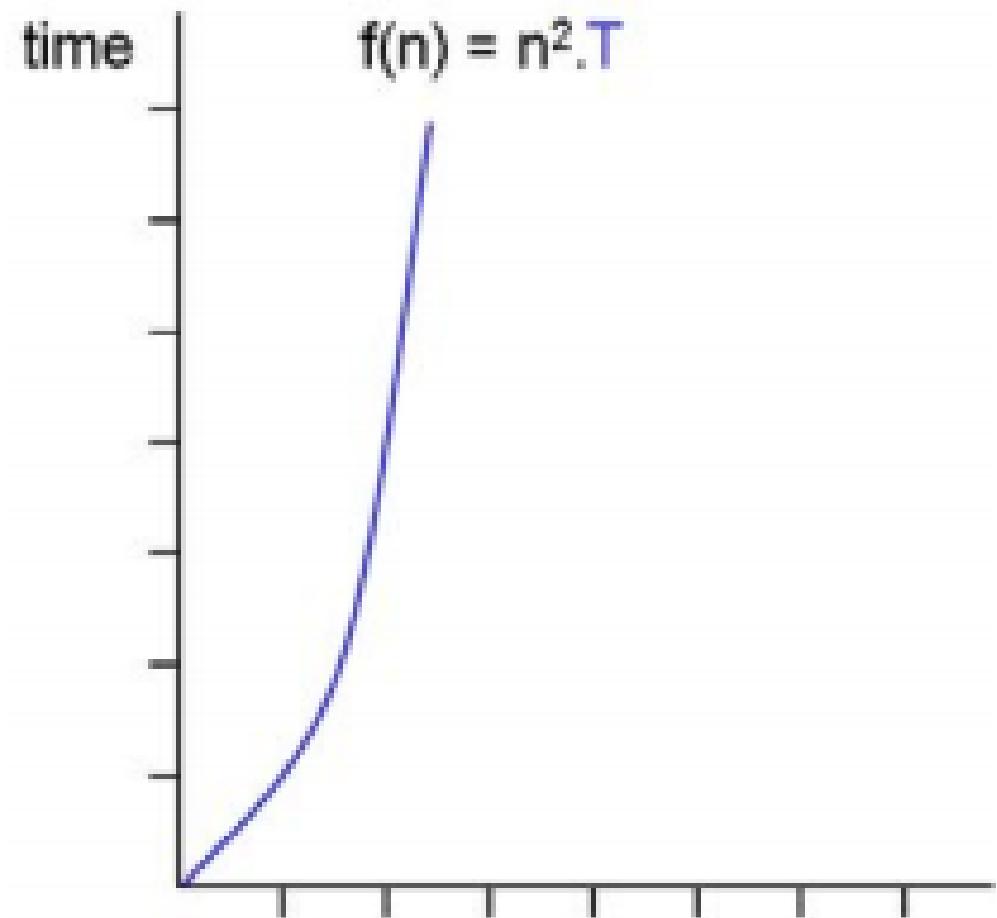
```
i = 1  
while (i <= n) do  
begin  
    j := 1 ;  
    while (j <= n) do  
    begin  
        P : {đoạn giải thuật với  
thời gian thực hiện T}  
        j := j * 2;  
    end  
    i := i + 1;  
end
```



Ký hiệu tiệm cận Big - O

● Ví dụ

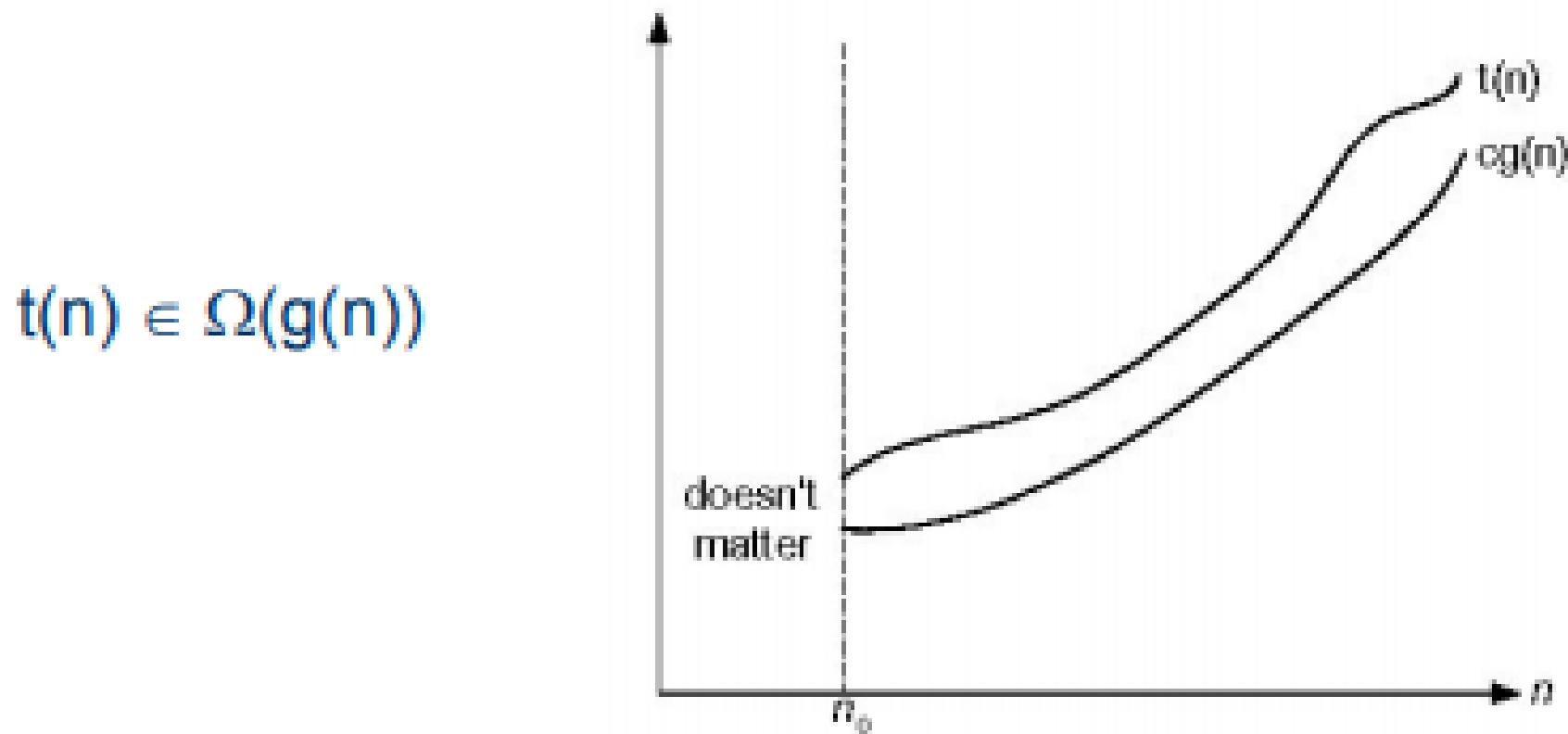
```
i = 1  
while (i <= n) do  
begin  
    j := 1 ;  
    while (j <= n) do  
    begin  
        P;  
        j := j +1;  
    end  
    i := i + 1;  
end
```



Các khái niệm tiệm cận khác

- Big- Omega

- $t(n)$ được coi là $\Omega(g(n))$ nếu tồn tại một hằng số $c > 0$ và một số nguyên $n_0 \geq 1$ sao cho $T(n) \geq c*g(n)$ với mọi $n \geq n_0$

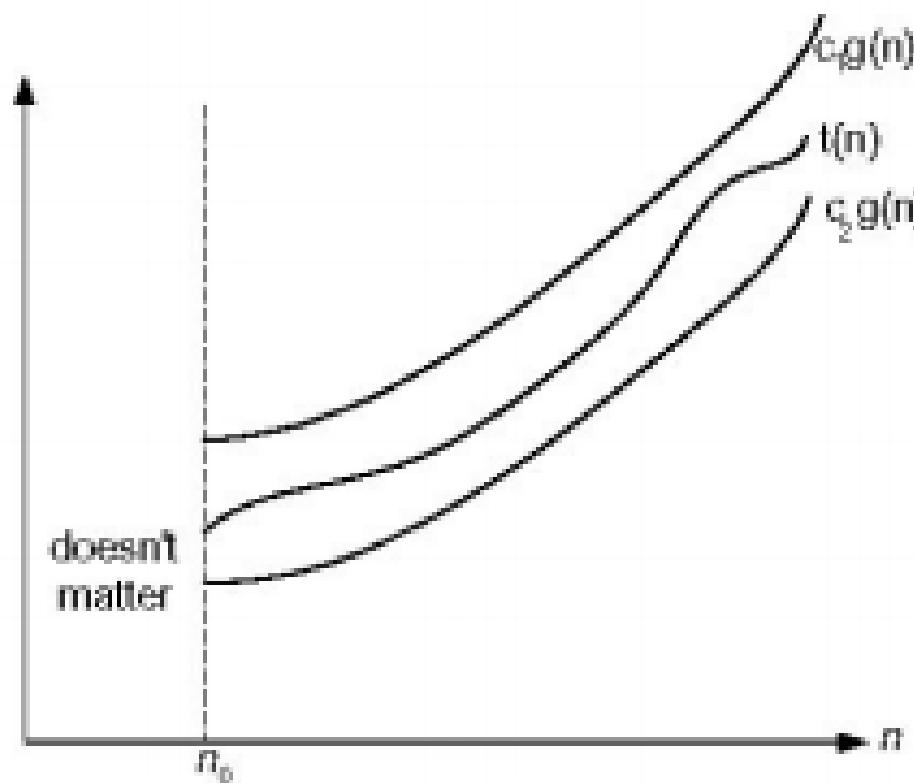


Các khái niệm tiệm cận khác

- Big-Theta

- $t(n)$ được coi là $\Theta(g(n))$ nếu tồn tại hai hằng số $c' > 0$ và $c'' > 0$ và một số nguyên $n_0 \geq 1$ sao cho $c'g(n) \leq T(n) \leq c''g(n)$ với mọi $n \geq n_0$

$t(n) \in \Theta(g(n))$



PHÉP TOÁN TÍCH CỰC (BEST PROXY)

- Trong một thuật toán, ta chú ý đặc biệt đến một phép toán gọi là phép toán tích cực. Đó là phép toán mà số lần thực hiện không ít hơn các phép toán khác
- Ví dụ 1:

```
s = 0;  
for (i=0; i<=n;i++){  
    p =1;  
    for (j=1;j<=i;j++)  
        p=p * x / j;  
    s = s+p;  
}
```

Phép toán tích cực là $p = p * x / j$

Ví dụ 1:

Số lần thực hiện phép toán này là

$$0+1+2+\dots+n = n(n-1)/2 = n^2/2 - n/2$$

=> Vậy độ phức tạp là $O(n^2/2 - n/2)$

$= O(n^2/2)$ sử dụng quy tắc lấy max

$= O(n^2)$ sử dụng quy tắc bỏ hằng số

Ví dụ 2:

```
1. s = 1; p = 1;  
2. for (i=1; i<=n; i++) {  
3.         p = p * x / i;  
4.         s = s + p;  
5. }
```

Phép toán tích cực là $p = p * x / i$

Số lần thực hiện phép toán là n

=> Vậy độ phức tạp của thuật toán là
 $O(n)$

Ví dụ 3:

```
1. for (i= 1; i<=n; i++)  
2.     for (j= 1; j<=n; j++)  
3.         // Lệnh
```

=> Dùng quy tắc nhân ta có $O(n^2)$

Ví dụ 4:

```
1. for (i= 1; i<=n; i++)  
2.     for (j= 1; j<=m; j++)  
3.         // Lệnh
```

=> Dùng quy tắc nhân ta có $O(n*m)$

Ví dụ 5:

```
1. for (i= 1; i<=n; i++)  
2.           // lệnh 1  
3. for (j= 1; j<=m; j++)  
4.           // lệnh 2
```

=> Dùng quy tắc cộng và quy tắc lấy max, sẽ có $O(\max(n,m))$

Ví dụ 7:

```
1. for (i= 1; i<=n; i++)  
2.   for (j= 1; j<=m; j++) {  
3.     for (k= 1; k<=x; k++)  
4.       //lệnh  
5.     for (h= 1; h<=y; h++)  
6.       //lệnh  
7. }
```

- => Dùng quy tắc cộng và quy tắc lấy max, sẽ có $O(n*m*\max(x,y))$

Ví dụ 8:

```
1. for (i= 1; i<=n; i++)  
2.   for (u= 1; u<= m; u++)  
3.     for (v= 1; v<=n; v++)  
4.       //lệnh  
5.   for (j= 1; j<=x; j++)  
6.     for (k= 1; k<=z; k++)  
7.       //lệnh
```

- => Dùng quy tắc cộng và quy tắc lấy max, sẽ có $O(\max(m*n^2, x*z))$

• Ví dụ 11: Đoạn chương trình tính tổng 2 đa thức

$$P(x) = x_m x_m + a_{m-1} x_{m-1} + \dots + a_1 x + a_0$$

$$Q(x) = b_n x_n + b_{n-1} x_{n-1} + \dots + b_1 x + b_0$$

```
1. if (m<n) p = m; else p = n;  
2. for (i=0; i<=p; i++)  
3.     c[i] = a[i] + b[i];  
4. if (p<m)  
5.     for (i=p+1; i<=m; i++) c[i] = a[i];  
6. else  
7.     for (i=p+1; i<=n; i++) c[i] = b[i];  
8. while (p>0 && c[p] = 0) p = p-1;
```

=> Độ phức tạp: O(max(m,n))

• Ví dụ 11: Đoạn chương trình tính tích 2 đa thức

$$P(x) = x_m x_m + a_{m-1} x_{m-1} + \dots + a_1 x + a_0$$

$$Q(x) = b_n x_n + b_{n-1} x_{n-1} + \dots + b_1 x + b_0$$

```
1. p = m+n;  
2. for (i=0; i<=p; i++) c[i] = 0;  
3. for (i=0; i<=m; i++)  
4.     for (j=0; j<=n; j++)  
5.         c[i+j] = c[i+j] + a[i] + b[j];
```

=> Độ phức tạp: O(m.n)

Tiếp cận thuật toán



Bước 1: Xác định bài toán. Nhiệm vụ của giai đoạn này là làm rõ yêu cầu đặt ra của bài toán. Có thể đặt ra và trả lời một số câu hỏi: Bài toán đặt ra vấn đề gì phải giải quyết. Giải quyết những vấn đề này trong phạm vi nào? Xếp thứ tự về tầm quan trọng các yêu cầu đặt ra. Ví dụ, bài toán đặt ra là “Giải hệ phương trình đại số tuyến tính có số phương trình bằng số ẩn”. Yêu cầu của bài toán này là “tìm nghiệm của hệ phương trình đại số tuyến tính n phương trình, n ẩn”. Yêu cầu quan trọng nhất là “độ chính xác của nghiệm”.

Tiếp cận thuật toán



- **Bước 2: Phân tích bài toán và lựa chọn cách giải quyết.** Với một bài toán đặt ra thường sẽ có nhiều cách giải quyết. Chúng có thể khác nhau về thời gian thực hiện, dung lượng bộ nhớ cần thiết, độ chính xác có thể đạt được. Tuỳ theo yêu cầu và điều kiện mà lựa chọn cách tiếp cận cho thích hợp. Ví dụ, với bài toán trên có thể lựa chọn chỉ giải quyết trường hợp hệ có một nghiệm duy nhất.

Tiếp cận thuật toán



- **Bước 3:** Xây dựng thuật toán (thuật giải). Trên cơ sở mô hình đã xây dựng được từ hai bước trước (chính là yêu cầu và các điều kiện) chi tiết hóa các bước thực hiện hoặc lựa chọn một trong các thuật toán đã biết phù hợp với bài toán.
- **Bước 4:** Viết chương trình theo thuật toán (thuật giải) đã lựa chọn. Dựa vào thuật toán (thuật giải) đã được xây dựng, lựa chọn ngôn ngữ lập trình phù hợp với thuật toán và dữ liệu để thiết kế chương trình.
- **Bước 5:** Xây dựng chương trình, thử nghiệm, triển khai

Tiếp cận thuật toán



- Ví dụ: tìm số có giá trị lớn thứ 2 trong mang a[0..n-1]. Ví dụ 1, 2, 5, 3, 7, 8, 3, 5, 8: thì số cần tìm là 7.
- Giải quyết:
 - Yêu cầu bài toán: Tìm số có giá trị lớn thứ 2. Đầu vào mảng có n phần tử. (Xác định chính xác tham số đầu ra)
 - Số có giá trị thứ 2:
 - Sắp xếp, sau đó tìm phần tử có giá trị lớn thứ 2
 - Dùng vòng lặp thứ nhất tìm phần tử lớn thứ nhất, vòng lặp thứ 2 tìm phần tử thứ 2
 - Dùng một vòng lặp, số lớn nhất sẽ là số đầu tiên, tìm số thứ khác số thứ nhất so sánh để lấy số thứ 2. duyệt đến cuối, nếu số lớn hơn số lớn nhất thì số lớn nhất hiện tại là số lớn thứ 2, ngược lại và lớn hơn số thứ 2 thì cập nhật số thứ 2.

Tiếp cận thuật toán



- Giải quyết (t):
 - int secondmax(int a[]){
 - M1=a[0];
 - For(i=1;a[i]==m1 && i<n;i++)
 - If(i<n)
 - If(a[i]>m1)
 - M2=m1; m1=a[i]
 - Else
 - M2=a[i];
 - For(j=i+1;j<n;j++)
 - If(a[j]>m1)
 - M2=m1; m1=a[j]
 - Else
 - If(a[j]>m2)
 - » m[2=a[j]
 - Return m2