

# Phân tích thuật toán

---

Nguyễn Mạnh Hiễn  
[hiennm@tlu.edu.vn](mailto:hiennm@tlu.edu.vn)

# Nội dung

1. Phân tích thuật toán là gì?
2. Các ký hiệu tiệm cận
3. Tốc độ tăng của các hàm
4. Các ví dụ phân tích thuật toán

# 1. Phân tích thuật toán là gì?

# Phân tích thuật toán

- Nhằm xác định **thời gian chạy** (độ phức tạp) của thuật toán dưới dạng một hàm  $f$  của kích thước đầu vào  $n$ 
  - VD: Thời gian tìm tuần tự một phần tử  $x$  trong một dãy  $n$  phần tử là  $f(n) = n$
- Đơn vị thời gian:
  - Không phải là giờ, phút, giây
  - Mà là **thao tác cơ bản**, VD: cộng, nhân, so sánh
  - Mỗi thao tác cơ bản có thời gian chạy là hằng (một lượng thời gian nhỏ không phụ thuộc vào kích thước đầu vào  $n$ )

# Đếm số thao tác cơ bản

- Nhận diện các thao tác cơ bản trong thuật toán
- Xác định thao tác cơ bản T chiếm nhiều thời gian chạy nhất so với các thao tác cơ bản còn lại
  - Thao tác T này thường xuất hiện trong các vòng lặp
- Đếm số lần thực hiện thao tác T, sẽ thu được hàm thời gian chạy  $f(n)$

# Ví dụ đếm số thao tác cơ bản

**Ví dụ 1:** In các phần tử (C++)

```
for (i = 0; i < n; i++)  
    cout << a[i] << endl;
```

**Số lần in ra màn hình =  $n$**

**Ví dụ 2:** Nhân ma trận tam giác dưới với vectơ (mã giả)

```
for i ← 1 to n  
    ci ← 0  
for i ← 1 to n  
    for j ← 1 to i  
        ci ← ci + aij * bj
```

**Số phép nhân =  $\sum_{i=1}^n i = n(n+1)/2$**

**Ví dụ 3:** Kiểm tra tính sắp xếp (C++)

```
template <class T>  
bool isSorted(T *a, int n)  
{  
    bool sorted = true;  
    for (int i=0; i<n-1; i++)  
        if (a[i] > a[i+1])  
            sorted = false;  
    return sorted;  
}
```

**Số phép so sánh =  $n - 1$**

Có thể cải tiến thuật toán bên trên?

## 2. Các ký hiệu tiệm cận

# Phân tích tiệm cận

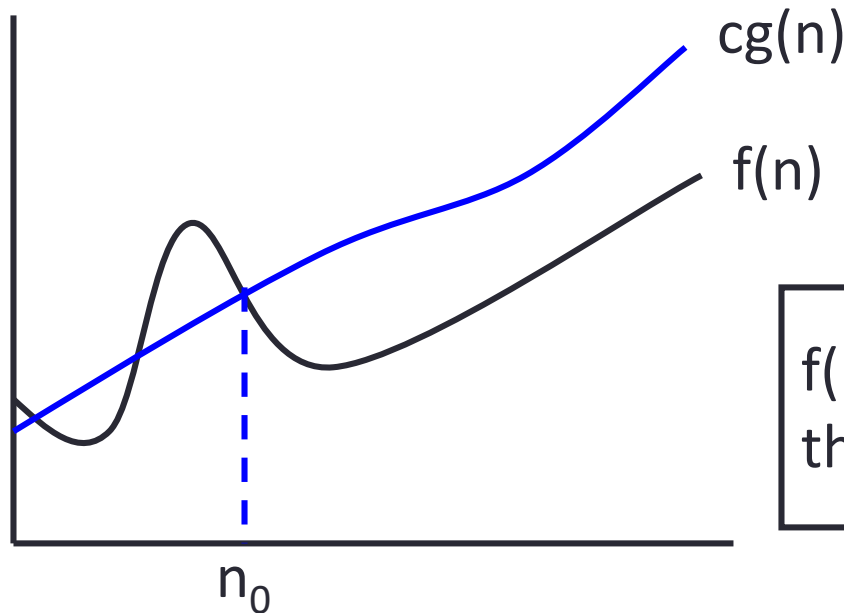
- Nhằm xem xét tốc độ tăng trưởng của hàm  $f(n)$  khi  $n$  dần tới  $+\infty$
- Cho phép quy các dạng hàm  $f(n)$  khác nhau về một vài dạng cơ bản (như  $\log n$ ,  $n$ ,  $n^2$ ), từ đó giúp so sánh thời gian chạy của các thuật toán dễ dàng hơn
- Có 3 cách phân tích tiệm cận tương ứng với ba ký hiệu tiệm cận sau đây:
  - Ô lớn:  $O$   $\rightarrow$  tìm cận trên của  $f(n)$
  - Ô-mê-ga lớn:  $\Omega$   $\rightarrow$  tìm cận dưới của  $f(n)$
  - Tê-ta lớn:  $\Theta$   $\rightarrow$  tìm cận chặt của  $f(n)$



# Ký hiệu O

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

khi và chỉ khi  $\exists c > 0$  và  $n_0 > 0$  sao cho  $f(n) \leq cg(n) \forall n \geq n_0$

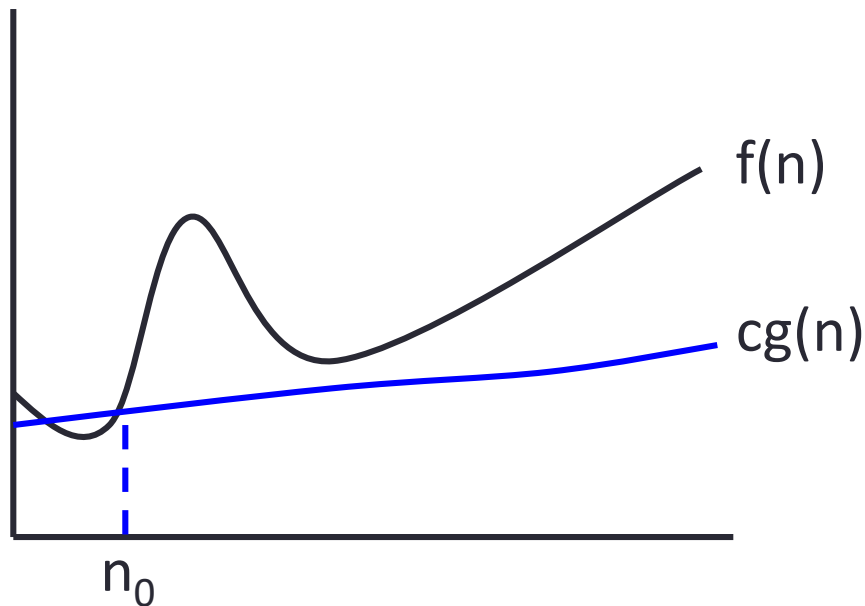


$f(n)$  bị chặn trên bởi  $g(n)$   
theo nghĩa tiệm cận

# Ký hiệu $\Omega$

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

khi và chỉ khi  $\exists c > 0$  và  $n_0 > 0$  sao cho  $cg(n) \leq f(n) \forall n \geq n_0$

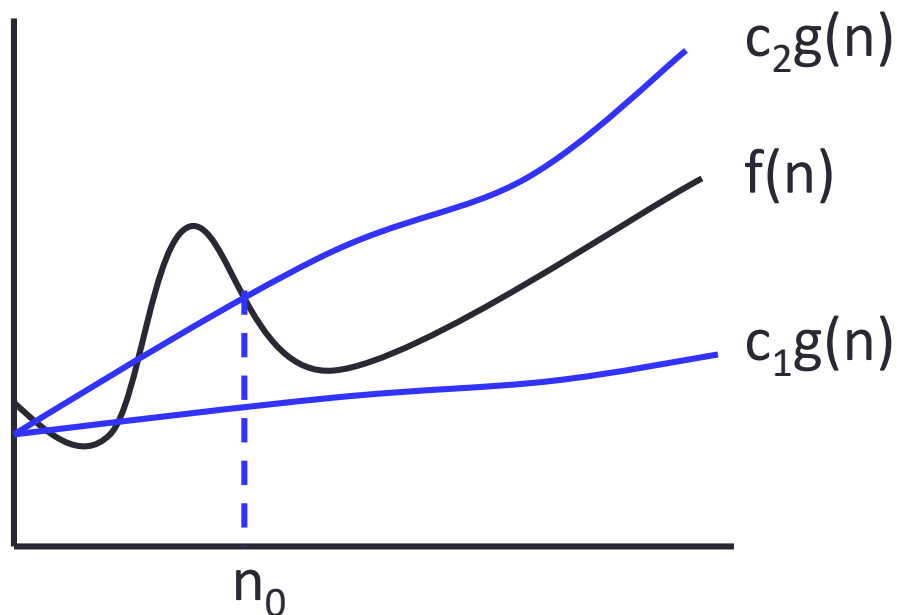


$f(n)$  bị chặn dưới bởi  $g(n)$   
theo nghĩa tiệm cận

# Ký hiệu $\Theta$

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

khi và chỉ khi  $\exists c_1 > 0, c_2 > 0$  và  $n_0 > 0$  sao cho  
 $c_1g(n) \leq f(n) \leq c_2g(n) \quad \forall n \geq n_0$



$f(n)$  có cùng tốc độ tăng  
với  $g(n)$  theo nghĩa tiệm  
cận

# Ví dụ phân tích tiệm cận

$$f(n) = 3n^2 + 17$$

- $\Omega(1), \Omega(n), \Omega(n^2)$  → cận dưới
- $O(n^2), O(n^3), \dots$  → cận trên
- $\Theta(n^2)$  → cận chặt

Hãy điền vào dấu chấm hỏi sau đây !

$$f(n) = 1000 n^2 + 17 + 0,001 n^3$$

- $\Omega(?)$  → cận dưới
- $O(?)$  → cận trên
- $\Theta(?)$  → cận chặt

$$\begin{aligned} &1, n^2, n^3 \\ &cg(n) = c \cdot n^3 \leq 0,001 n^3 + 1000 n^2 \\ &(c - 0,001)n \leq 1000 \end{aligned}$$

# Tính chất bắc cầu

- Nếu  $f(n) = O(g(n))$  và  $g(n) = O(h(n))$   
 $\rightarrow f(n) = O(h(n))$
- Nếu  $f(n) = \Omega(g(n))$  và  $g(n) = \Omega(h(n))$   
 $\rightarrow f(n) = \Omega(h(n))$
- Nếu  $f(n) = \Theta(g(n))$  và  $g(n) = \Theta(h(n))$   
 $\rightarrow f(n) = \Theta(h(n))$

# Một số quy tắc

- Nếu  $f(n) = a_0 + a_1n + \dots + a_kn^k$  ( $a_k > 0$ )  
 $\rightarrow f(n) = O(n^k)$
- $\log^k n = O(n)$  với  $k$  là một hằng số  
(hàm lôgarit tăng chậm hơn hàm tuyến tính)

*Chú ý: Trong môn học này, khi viết hàm lôgarit mà không chỉ rõ cơ số, ta ngầm hiểu cơ số đó là 2*

### 3. Tốc độ tăng của các hàm

# Tốc độ tăng của một số hàm cơ bản



Hàm	Tên
c	Hằng
$\log n$	Lôgarit
$\log^2 n$	Lôgarit bình phương
n	Tuyến tính
$n \log n$	
$n^2$	Bậc hai
$n^3$	Bậc ba
$2^n$	Hàm mũ



# Hàm nào tăng chậm hơn?

- $f(n) = n \log n$  và  $g(n) = n^{1,5}$
- Lời giải:  $f(n)$ 
  - Chú ý rằng  $g(n) = n^{1,5} = n * n^{0,5}$
  - Vì vậy, chỉ cần so sánh  $\log n$  và  $n^{0,5}$
  - Tương đương với so sánh  $\log^2 n$  và  $n$
  - Tham khảo bảng trong slide trước:  $\log^2 n$  tăng chậm hơn  $n$
  - Suy ra  $f(n)$  tăng chậm hơn  $g(n)$

# Ví dụ về tốc độ tăng của các hàm

- Xét một chiếc máy tính thực hiện được 1.000.000 thao tác cơ bản trong một giây
- Khi thời gian chạy vượt quá  $10^{25}$  năm, ta viết "very long"

	$n$	$n \log_2 n$	$n^2$	$n^3$	$1.5^n$	$2^n$	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	$10^{25}$ years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	$10^{17}$ years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

## 4. Các ví dụ phân tích thuật toán

# Vòng lặp

```
1  for (i = 0; i < n; i++)
2  {
3      x = a[i]/2;
4      a[i] = x + 1;
5  }
```

- Có 4 thao tác cơ bản ở các dòng 3 và 4 gồm 2 phép gán, 1 phép chia và 1 phép cộng
- Cả 4 thao tác đó được lặp lại  $n$  lần
- Thời gian chạy:  $t(n) = 4n = O(n)$

*Chú ý: Ở đây, ta bỏ qua 3 thao tác cơ bản điều khiển quá trình lặp ở dòng 1. Kết quả phân tích thuật toán sẽ không thay đổi nếu tính thêm cả 3 thao tác đó.*

# Vòng lặp có câu lệnh break

```
1  for (i = 0; i < n; i++)
2  {
3      x = a[i]/2;
4      a[i] = x + 1;
5      if (a[i] > 10) break;
6  }
```

- Có 5 thao tác cơ bản ở các dòng 3, 4, 5 gồm 2 phép gán, 1 phép chia, 1 phép cộng và 1 phép so sánh
- Không thể đếm chính xác số lần thực hiện 5 thao tác đó vì ta không biết khi nào điều kiện  $a[i] > 10$  xảy ra
- Trong trường hợp tồi nhất, tức là điều kiện  $a[i] > 10$  xảy ra ở bước lặp cuối cùng hoặc không xảy ra, cả 5 thao tác cơ bản được lặp lại  $n$  lần
- Thời gian chạy (trong trường hợp tồi nhất):  $t(n) = 5n = O(n)$

# Các vòng lặp tuần tự

```
for (i = 0; i < n; i++)  
{  
    ... // 3 thao tác cơ bản ở đây  
}  
for (i = 0; i < n; i++)  
{  
    ... // 5 thao tác cơ bản ở đây  
}
```

- Chỉ cần cộng thời gian chạy của các vòng lặp
- Thời gian chạy tổng thể:  $t(n) = 3n + 5n = 8n = O(n)$

# Các vòng lặp lồng nhau

```
for (i = 0; i < n; i++)  
{  
    ... // 2 thao tác cơ bản ở đây  
    for (j = 0; j < n; j++)  
        ... // 3 thao tác cơ bản ở đây  
}
```

- Phân tích các vòng lặp từ trong ra ngoài:
  - Vòng lặp bên trong thực hiện  $3n$  thao tác cơ bản
  - Mỗi bước lặp của vòng lặp bên ngoài thực hiện  $2 + 3n$  thao tác cơ bản
- Thời gian chạy tổng thể:  $t(n) = (2 + 3n)n = 3n^2 + 2n = O(n^2)$

# Câu lệnh if-else

```
1  if (x > 0)
2      i = 0;
3  else
4      for (j = 0; j < n; j++)
5          a[j] = j;
```

- Có 3 thao tác cơ bản:
  - Phép so sánh ở dòng 1 được thực hiện 1 lần
  - Phép gán ở dòng 2 được thực hiện 0 hoặc 1 lần
  - Phép gán ở dòng 5 được thực hiện 0 hoặc n lần
- Trong trường hợp tồi nhất (điều kiện  $x > 0$  sai), phép gán ở dòng 5 chạy n lần
- Thời gian chạy:  $t(n) = 1 + n = O(n)$



# Hàm đệ quy

```
1 long factorial(int n)
2 {
3     if (n <= 1)
4         return 1;
5     else
6         return n * factorial(n - 1);
7 }
```

- Nếu  $n = 1$ , chỉ mất 1 phép so sánh ở dòng 3
- Nếu  $n > 1$ :
  - Dòng 3 có 1 phép so sánh
  - Dòng 6 có 1 phép nhân, 1 phép trừ và 1 lời gọi hàm đệ quy tốn thời gian  $t(n-1)$

# Hàm đệ quy (tiếp)

- Suy ra thời gian chạy của thuật toán:

$$t(1) = 1 \quad (\text{với } n = 1)$$

$$t(n) = 3 + t(n - 1) \quad (\text{với } n > 1)$$

$$= 3 + 3 + t(n - 2)$$

$$= 3 + 3 + 3 + t(n - 3)$$

...

$$= 3k + t(n - k)$$

- Chọn  $k = n - 1$ , khi đó:

$$t(n) = 3(n - 1) + t(1) = 3n - 2 = O(n)$$

# Tìm kiếm tuần tự

```
for (i = 0; i < n; i++)  
{  
    if (a[i] == x) return i;  
}  
return -1;
```

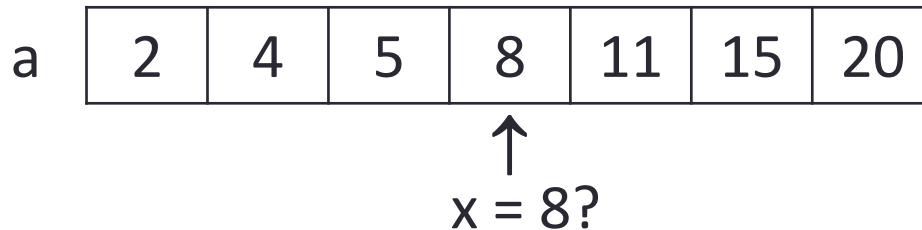
- Trong trường hợp tồi nhất (x nằm ở cuối mảng hoặc x không có trong mảng), ta phải thực hiện n phép so sánh  $a[i] == x$
- Thời gian chạy:  $t(n) = n = O(n)$

# Tìm kiếm nhị phân

- Cho mảng  $a$  đã sắp xếp tăng dần
- Tìm  $x$  trong mảng  $a$ :
  - So sánh  $x$  với phần tử ở chính giữa mảng  $a[\text{mid}]$
  - Nếu  $x < a[\text{mid}]$ , tìm  $x$  ở nửa bên trái của mảng
  - Nếu  $x > a[\text{mid}]$ , tìm  $x$  ở nửa bên phải của mảng
  - Nếu  $x = a[\text{mid}]$ , báo cáo vị trí tìm được  $x$  là  $\text{mid}$
  - Nếu không còn phần tử nào để xét, báo cáo không tìm được  $x$

# Tìm kiếm nhị phân – ví dụ

Giả sử  $x = 11$  và ta phải tìm  $x$  trong mảng  $a$  bên dưới



# Tìm kiếm nhị phân – thuật toán

```
function binarySearch(a, n, x)
{
    left ← 1, right ← n
    while (left ≤ right)
    {
        mid ← (left + right) / 2
        if      (x < a[mid])  right ← mid - 1
        else if (x > a[mid])  left  ← mid + 1
        else                  return mid
    }
    return -1
}
```

# Tìm kiếm nhị phân – phân tích

- Nếu  $n = 1$ , chỉ mất một phép so sánh  $x$  với phần tử duy nhất của mảng
- Nếu  $n > 1$ , mất một phép so sánh  $x$  với phần tử chính giữa mảng, sau đó là mất thời gian tìm  $x$  trong một nửa (trái hoặc phải) của mảng
- Suy ra thời gian chạy của thuật toán:

$$t(1) = 1 \quad (\text{với } n = 1)$$

$$t(n) = 1 + t(n/2) \quad (\text{với } n > 1)$$

$$= 1 + 1 + t(n/4)$$

$$= 1 + 1 + 1 + t(n/8)$$

...

$$= k + t(n/2^k)$$

- Chọn  $k = \log n$ , khi đó:

$$t(n) = \log n + t(1) = \log n + 1 = O(\log n)$$