CS112.N21.KHCL

CS112
PHÂN TÍCH ĐỘ PHỨC TẠP THUẬT TOÁN KHÔNG ĐỆ QUI

GROUP 5

MEMBER

Tran Quoc An
Mai Anh Quan
Nguyen Hai Dang

NỘI DUNG

- Framework phân tích độ phức tạp
- Các ký hiệu tiệm cận và phân lớp hiệu năng cơ bản
- Phân tích độ phức tạp thuật toán không đệ qui

THUẬT TOÁN LÀ GÌ?

KĨ THUẬT THIẾT KẾ THUẬT TOÁN LÀ GÌ?

LÀM CÁCH NÀO ĐỂ BIỂU DIỄN THUẬT TOÁN?

TẠI SAO PHẢI ĐÁNH GIÁ/PHÂN TÍCH THUẬT TOÁN?

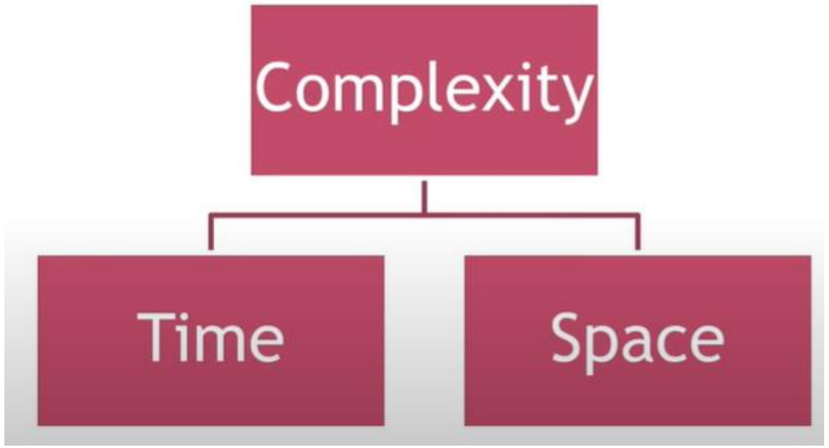
ĐÁNH GIÁ/PHÂN TÍCH MỘT THUẬT TOÁN DỰA TRÊN NHỮNG TIÊU CHÍ NÀO?

1. FRAMEWORK PHÂN TÍCH HIỆU NĂNG

Có 2 loại hiệu năng:

Time efficiency/complexity: Xác định thời gian chạy nhanh hay chậm của thuật toán.

Space efficiency/complexity: Đề cập đến số lượng đơn vị bộ nhớ cần thêm.



1.1 ĐO LƯỜNG KÍCH THƯỚC INPUT

- **Hầu hết các thuật toán sẽ chạy chậm hơn đối với kích thước input lớn.**
- **Do đó, ta có thể khảo sát thuật như một hàm với tham số n nào đó.**

Ví dụ: Sẽ mất thời gian nhiều hơn để sắp xếp các mảng lớn hơn, nhân các ma trận lớn hơn...

```
void insertionSort(int a[], int array_size) {
    int i, j, last;
    for (i=1; i < array_size; i++) {
        last = a[i];
        j = i;
    while ((j > 0) && (a[j-1] > last)) {
        a[j] = a[j-1];
        j = j - 1; }
        a[j] = last;
    } // end for
} // end of isort
```

← Insertion Sort

1.1 ĐO LƯỜNG KÍCH THƯỚC INPUT

- Sự lựa chọn 1 đơn vị đo kích thước phù hợp có thể bị ảnh hưởng bởi cách hoạt động của thuật toán

Ví dụ: Thuật toán kiểm tra lỗi chính tả

- **Nếu thuật toán kiểm tra từng kí tự**

→ Ta nên đo kích thước input dựa trên số lượng kí tự

- **Nếu thuật toán xử lý từng từ**

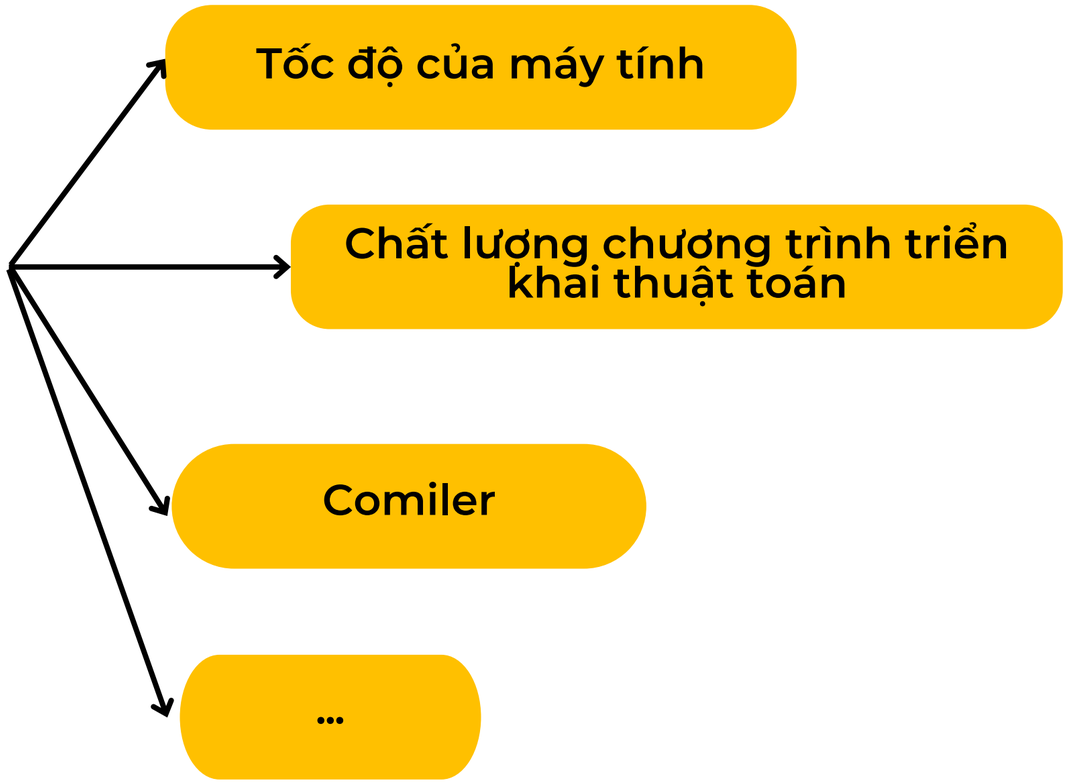
→ Ta nên đo kích thước input dựa trên số lượng từ

1.2 ĐƠN VỊ ĐO THỜI GIAN CHẠY

Q: Đo thời gian chạy của 1 thuật toán như thế nào?

1.2 ĐƠN VỊ ĐO THỜI GIAN CHẠY

- **Ta Không thể dùng đơn vị đo thời gian tiêu chuẩn như second hay milisecond để đo thời gian chạy của thuật toán vì phụ thuộc vào nhiều yếu tố**



1.2 ĐƠN VỊ ĐO THỜI GIAN CHẠY

- Một cách là đếm số lần thuật toán thực thi 1 operation (cách này khó và không cần thiết)

- Điều cần làm là xác định operation quan trọng nhất của thuật toán, gọi là basic operation

- Basic operation: là operation đóng góp nhiều nhất vào thời gian chạy của thuật toán

ALGORITHM *MaxElement*$(A[0..n-1])$

```
//Determines the value of the largest element in a given array
//Input: An array A[0..n − 1] of real numbers
//Output: The value of the largest element in A
maxval ← A[0]
for i ← 1 to n − 1 do
    if A[i] > maxval          ⟶  basic operation
        maxval ← A[i]
return maxval
```

- Số lần thuật toán thực thi 1 basic operation kí hiệu là C(n)

1.3 CẤP ĐỘ TĂNG
(Orders of growth)

- Sự khác nhau về thời gian chạy trên input kích thước nhỏ không thực sự phân biệt được thuật toán hiệu quả và không hiệu quả.

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

1.4 HIỆU NĂNG TRƯỜNG HỢP XẤU NHẤT, TỐT NHẤT VÀ TRUNG BÌNH
(Worst-case, best-case and average-case efficiencies)

- Nhiều thuật toán mà thời gian chạy của nó không chỉ phụ thuộc vào kích thước input mà còn phụ thuộc vào các chi tiết cụ thể của một input cụ thể

Ví dụ: ta có thuật toán tìm kiếm tuần tự

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element in A that matches K
//        or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```

Thời gian chạy của thuật toán có thể khác nhau trên cùng 1 mảng có n phần tử

Trường hợp xấu nhất: phần tử cần tìm nằm ở cuối mảng hoặc không tồn tại

1.4 HIỆU NĂNG TRƯỜNG HỢP XẤU NHẤT, TỐT NHẤT VÀ TRUNG BÌNH
(Worst-case, best-case and average-case efficiencies)

- Hiệu năng trong trường hợp xấu nhất: $C_{worst}(n)$.
- Hiệu năng trong trường hợp tốt nhất: $C_{best}(n)$.

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element in A that matches K
//        or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```

- $C_{worst}(n) = n$.
- $C_{best}(n) = 1$.

1.4 HIỆU NĂNG TRƯỜNG HỢP XẤU NHẤT, TỐT NHẤT VÀ TRUNG BÌNH
(Worst-case, best-case and average-case efficiencies)

Tuy nhiên, phân tích trường xấu nhất và tốt nhất không thể hiện được hành vi của thuật toán trong trường hợp input mang tính " **đặc trưng** " hay " **ngẫu nhiên**“.

⟶ **Hiệu năng trong trường hợp trung bình:** $C_{avg}(n)$.

Q: Làm cách nào để xác định $C_{avg}(n)$?

1.4 HIỆU NĂNG TRƯỜNG HỢP XẤU NHẤT, TỐT NHẤT VÀ TRUNG BÌNH
(Worst-case, best-case and average-case efficiencies)

- **Để phân tích hiệu năng trường hợp trung bình của thuật toán, ta cần phải đưa ra các giả định về các loại input khả thi có kích thước là n.**

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element in A that matches K
//        or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```

Giả định:
- Xác suất tìm kiếm thành công là p $(0 \le p \le 1)$ (phần tử cần tìm có tồn tại).
- Xác suất tìm thấy tại vị trí thứ i là như nhau với mọi i.

- Trong trường hợp tìm kiếm thành công, xác suất tìm thấy tại vị trí thứ i là p/n với mọi i.

- Trong trường hợp tìm kiếm không thành công, số lần thực hiện so sánh là n với xác suất là $(1-p)$.

$$C_{avg}(n) = \left[1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}\right] + n \cdot (1-p)$$
$$= \frac{p}{n}[1 + 2 + \cdots + i + \cdots + n] + n(1-p)$$
$$= \frac{p}{n}\frac{n(n+1)}{2} + n(1-p) = \frac{p(n+1)}{2} + n(1-p)$$

1.5 TÓM TẮT

- Cả hiệu năng thời gian và không gian đều được đo như các hàm của kích thước input của thuật toán

- Hiệu năng của thuật toán có thể thay đổi một cách đáng kể trên các kiểu input khác nhau có cùng kích thước. Đối với những thuật toán như thế, ta cần phải phân biệt hiệu năng của các trường hợp xấu nhất, tốt nhất và trung bình.

- Hiệu năng của thuật toán có thể thay đổi 1 cách đáng kể trên các kiểu input khác nhau có cùng kích thước

- Trọng tâm của framework nằm ở cấp độ tăng của thời gian chạy của thuật toán khi kích thước input tiến đến vô cùng.

2. CÁC KÝ HIỆU TIỆM CẬN VÀ PHÂN LỚP HIỆU NĂNG

Để so sánh và xếp hạng các cấp độ tăng (orders of growth), người ta dùng 3 ký hiệu tiệm cận:

- O (big-oh)
- Ω (big-omega)
- Θ (big-theta)

2.1 Ký hiệu tiệm cận O (chặn trên)

ĐỊNH NGHĨA. Ta nói hàm $t(n)$ thuộc $O\big(g(n)\big)$, ký hiệu $t(n) \in O\big(g(n)\big)$, nếu $t(n)$ bị chặn trên bởi bội số không đổi của $g(n)$ với mọi n lớn. Tức là tồn tại một hằng số c và số nguyên không âm n_0 sao cho:

$$t(n) \leq cg(n) \quad \text{với mọi } n \geq n_0.$$

2.1 Ký hiệu tiệm cận O (chặn trên)

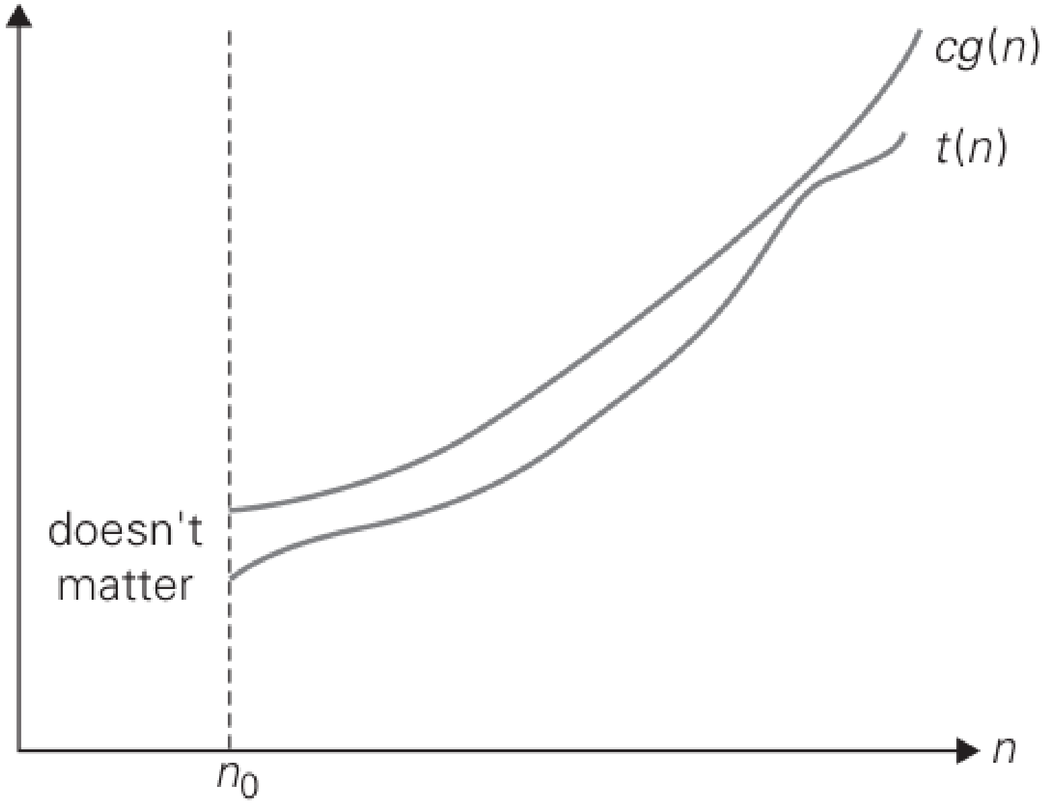


FIGURE 2.1 Big-oh notation: $t(n) \in O(g(n))$.

2.1 Ký hiệu tiệm cận O (chặn trên)

Ví dụ: chứng minh $100n + 5 \in O(n^2)$.

$$100n + 5 \leq 100n + n = 101n \leq 101n^2 \quad \text{với mọi } n \geq 5.$$

Như vậy, ta có thể chọn $c = 100$ và $n_0 = 5$. Lưu ý rằng, định nghĩa cho ta rất nhiều tự do trong việc lựa chọn các hằng số c và n_0. Ta có thể làm như sau:

$$100n + 5 \leq 100n + 5n = 105n \quad \text{với mọi } n \geq 1.$$

và hoàn tất chứng minh với $c = 5$ và $n_0 = 1$.

2.2 Ký hiệu tiệm cận Ω (chặn dưới)

ĐỊNH NGHĨA. Ta nói hàm $t(n)$ thuộc $\Omega\big(g(n)\big)$, ký hiệu $t(n) \in \Omega\big(g(n)\big)$, nếu $t(n)$ bị chặn dưới bởi bội số không đổi của $g(n)$ với mọi n lớn. Tức là tồn tại một hằng số c và số nguyên không âm n_0 sao cho:

$$t(n) \geq cg(n) \quad \text{với mọi } n \geq n_0.$$

2.2 Ký hiệu tiệm cận Ω (chặn dưới)

Q: Có trường hợp cấp độ tăng của $t(n)$ lớn hơn $cg(n)$ hay không?

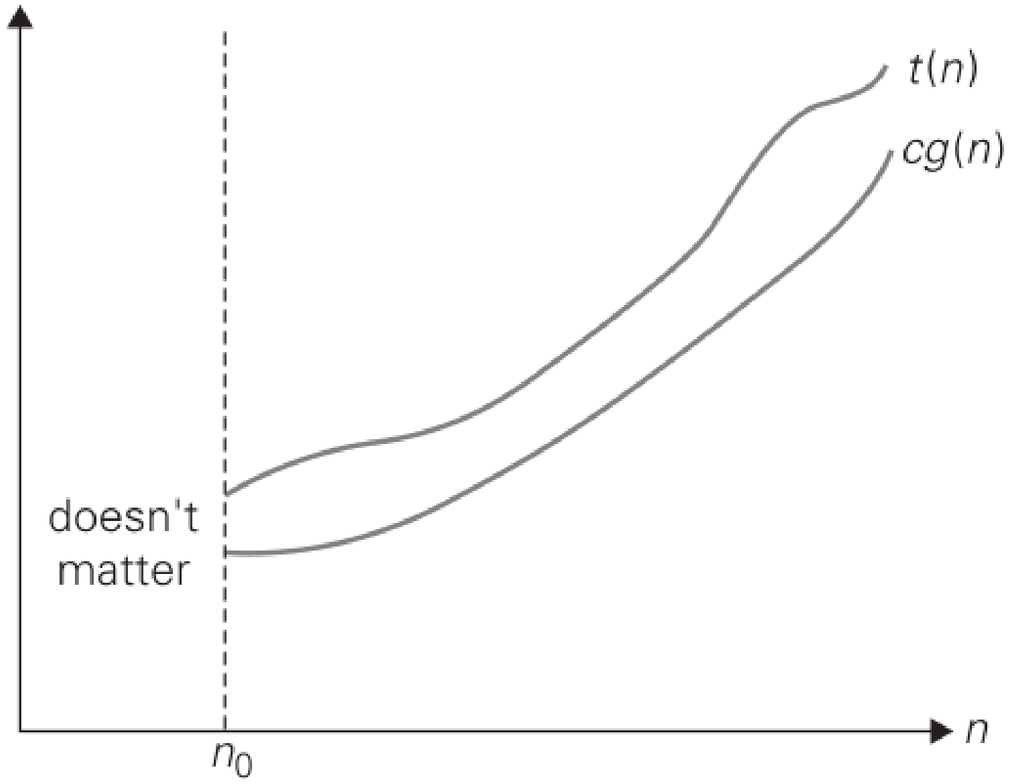


FIGURE 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$.

2.2 Ký hiệu tiệm cận Ω (chặn dưới)

Ví dụ: chứng minh rằng $n^3 \in \Omega(n^2)$.

$$n^3 \geq n^2 \quad \text{với mọi } n \geq 0.$$

Vậy ta có thể chọn $c = 1$ và $n_0 = 0$.

2.3 Ký hiệu tiệm cận Θ (chặn trên và chặn dưới)

ĐỊNH NGHĨA. Ta nói hàm $t(n)$ thuộc $\Theta(g(n))$, ký hiệu $t(n) \in \Theta(g(n))$, nếu $t(n)$ bị chặn dưới bởi bội số không đổi của $g(n)$ với mọi n lớn. Tức là tồn tại hằng số c_1 và c_1 và số nguyên không âm n_0 sao cho:

$$c_2 g(n) \leq t(n) \leq c_1 g(n) \quad \text{với mọi } n \geq n_0.$$

2.3 Ký hiệu tiệm cận Θ (chặn trên và chặn dưới)

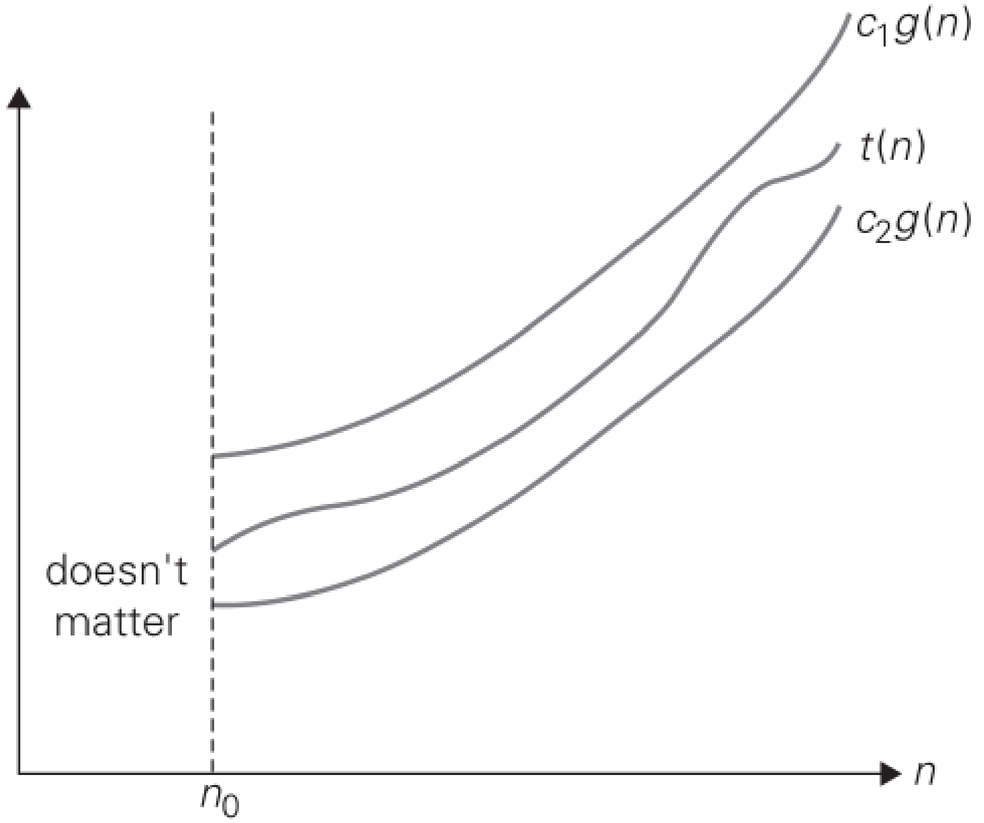


FIGURE 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$.

2.3 Ký hiệu tiệm cận Θ (chặn trên và chặn dưới)

Ví dụ: chứng minh rằng $\frac{1}{2}n(n-1) \in \Theta(n^2)$.

Đầu tiên, ta sẽ chứng minh chặn trên:

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \qquad \forall n \geq 0.$$

Tiếp theo, ta sẽ chứng minh chặn dưới:

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n = \frac{1}{4}n^2 \qquad \forall n \geq 2.$$

Như vậy, ta có thể chọn $c_1 = \frac{1}{4}$, $c_1 = \frac{1}{2}$ **và** $n_0 = 2$.

2.4 Tính chất liên quan đến các ký hiệu tiệm cận

ĐỊNH LÝ. Nếu $t_1(n) \in O\big(g_1(n)\big)$ và $t_2(n) \in O\big(g_2(n)\big)$, thì

$$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\}).$$

Điều này cũng đúng với các ký hiệu tiệm cận Ω và Θ.

2.5 Dùng giới hạn để so sánh cấp độ tăng

Ta có thể so sách cấp độ tăng giữa 2 hàm bằng cách tính giới hạn tỷ lệ của chúng.

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\begin{cases}0\\c\\\infty\end{cases}$$

Trong đó:

- 0 nghĩa là $t(n)$ có cấp độ tăng nhỏ hơn $g(n)$
- Hằng số c nghĩa là $t(n)$ có cùng cấp độ tăng với $g(n)$
- ∞ nghĩa là $t(n)$ có cấp độ tăng lớn hơn $g(n)$

2.5 Dùng giới hạn để so sánh cấp độ tăng

Ta có thể tận dụng quy tắc L'Hôspital:

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \lim_{n\to\infty} \frac{t'(n)}{g'(n)}$$

và công thức/xấp xỉ Stirling (đối với giá trị n lớn):

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

2.5 Dùng giới hạn để so sánh cấp độ tăng

Ví dụ 1. So sánh cấp độ tăng của $\frac{1}{2}n(n-1)$ và n^2.

$$\lim_{n\to\infty}\frac{\frac{1}{2}n(n-1)}{n^2}=\frac{1}{2}\lim_{n\to\infty}\frac{n^2-n}{n^2}=\frac{1}{2}\lim_{n\to\infty}\left(1-\frac{1}{n}\right)=\frac{1}{2}.$$

Vậy cấp tăng của 2 hàm là như nhau. Có thể viết $\frac{1}{2}n(n-1)\in\Theta(n^2)$.

2.5 Dùng giới hạn để so sánh cấp độ tăng

Ví dụ 2. So sánh cấp độ tăng của $\log_2 n$ và $\sqrt{n}$.

$$\lim_{n\to\infty}\frac{\log_2 n}{\sqrt{n}}=\lim_{n\to\infty}\frac{(\log_2 n)'}{(\sqrt{n})'}=\lim_{n\to\infty}\frac{\frac{1}{n\ln 2}}{\frac{1}{2\sqrt{n}}}=\lim_{n\to\infty}\frac{\log_2 e}{n}2\sqrt{n}=2\log_2 e\lim_{n\to\infty}\frac{1}{\sqrt{n}}=0.$$

Vậy $\log_2 n$ **có cấp độ tăng nhỏ hơn** $\sqrt{n}$.

2.5 Dùng giới hạn để so sánh cấp độ tăng

Ví dụ 3. So sánh cấp độ tăng của $n!$ và 2^n. Sử dụng xấp xỉ Stirling:

$$\lim_{n\to\infty}\frac{n!}{2^n}=\lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2^n}=\lim_{n\to\infty}\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n=\infty.$$

2^n tăng rất nhanh nhưng $n!$ tăng còn nhanh hơn.

Ta có thể viết là : $n!\in\Omega(2^n)$ nhưng vẫn không biết được cấp độ tăng của $n!$ có lớn hơn 2^n hay không. Ta chỉ thấy được điều đó khi tính giới hạn.

2.6 Phân lớp hiệu năng

1	Constant
$\log(n)$	Logarithmic
n	Linear
$n\log(n)$	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
$n!$	Factorial

2.7 Bài tập tại lớp

1. Các mệnh đề sau đúng hay sai?

a. $\frac{n(n+1)}{2} \in O(n^3)$

b. $\frac{n(n+1)}{2} \in O(n^2)$

c. $\frac{n(n+1)}{2} \in \Theta(n^3)$

d. $\frac{n(n+1)}{2} \in \Omega(n^3)$

2.7 Bài tập tại lớp

2. Xác định lớp hiệu năng $\Theta(g(n))$ của các hàm sau:

a. $(n^2 + 1)^{10}$

b. $\sqrt{10n^2 + 7n + 3}$

c. $2n \lg(n+1)^2 + (n+1)^2 \lg \frac{n}{2}$ **(**$\lg n = \log_2 n$**)**

d. $2^{n+1} + 3^{n-1}$

(Lưu ý, $g(n)$ **phải đơn giản nhất có thể)**

2.7 Bài tập tại lớp

3. Sắp xếp cấp độ tăng từ nhỏ đến lớn:

$(n-2)!$, $5\lg(n+100)^{10}$, 2^{2n}, $0.001n^4+3n^3+1$, $\ln^2 n$, $\sqrt[3]{n}$, 3^n.

3. Phân tích độ phức tạp thuật toán không đệ quy

Q: Thế nào là thuật toán không đệ quy?

3.1 Hướng chung để phân tích độ phức tạp thời gian

- Xác định một hay nhiều tham số biểu thị kích thước của input
- Xác định basic operation của thuật toán. (Về nguyên tắc, nó sẽ nằm ở vòng lặp trong cùng)
- Kiểm tra xem số lần thuật toán thực thi basic operation có chỉ phụ thuộc vào kích thước input hay không. Nếu nó còn phụ thuộc vào một vài yếu tố khác thì ta cần phải khảo sát riêng từng trường hợp xấu nhất, trung bình và trường hợp tốt nhất nếu cần thiết
- Thiết lập một tổng thể hiện số lần thuật toán thực thi basic operation
- Dùng các công thức và quy tắc biến đổi tổng để cho ra một công thức rút gọn thể hiện được số lần thuật toán thực thi basic operation hoặc ít nhất là thiết lập được cấp độ tăng của thuật toán

3.1 Hướng chung để phân tích độ phức tạp thời gian

2 quy tắc của tổng thường sử dụng:

$$\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i \,.$$

$$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i + \sum_{i=l}^{u} b_i \,.$$

Và 2 công thức tổng:

$$\sum_{i=l}^{u} 1 = u - l + 1.$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2).$$

3.1 Hướng chung để phân tích độ phức tạp thời gian

Ví dụ 1. Tìm giá trị lớn nhất trong mảng có n phần tử

ALGORITHM *MaxElement*$(A[0..n-1])$
//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

- Kích thước input là n , số phần tử trong mảng
- Basic operation là phép so sánh A[i] > maxval, vì nó luôn được thực thi sau mỗi lần lặp, còn phép gán thì không
- Số lần thực thi basic operation đối với mọi mảng có kích thước n là như nhau. Do đó, không cần phải phân biệt các trường hợp xấu nhất, tốt nhất và trung bình
- Thuật toán lặp từ chỉ số 1 đến chỉ số cuối của mảng. Do đó:

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

3.1 Hướng chung để phân tích độ phức tạp thời gian

Ví dụ 2. Kiểm tra các phần tử trong mảng có đôi 1 khác nhau không

ALGORITHM *UniqueElements*$(A[0..n-1])$

```
//Determines whether all the elements in a given array are distinct
//Input: An array A[0..n − 1]
//Output: Returns "true" if all the elements in A are distinct
//          and "false" otherwise
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i] = A[j] return false
return true
```

- Kích thước input là n , số phần tử trong mảng
- Basic operation là phép so sánh A[i] = A[j] ở vòng lặp trong cùng
- Số lần thực thi basic operation không chỉ phụ thuộc vào mà còn phụ thuộc có hay không các phần tử bằng nhau, nếu có thì vị trí các phần tử đó ở đâu. Ta sẽ chỉ khảo sát trường hợp xấu nhất, $C_{worst}(n)$.

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).$$

3.2 Bài tập tại lớp

Tính các tổng sau:

a. $\sum_{i=3}^{n+1} i$

b. $\sum_{i=0}^{n-1} i(i+1)$

c. $\sum_{j=1}^{n} 3^{j+1}$

d. $\sum_{i=1}^{n} \sum_{j=1}^{n} ij$

e. $\sum_{i=1}^{n} 1/i(i+1)$

4. Bài tập về nhà

1. Cho thuật toán sau:

ALGORITHM *Mystery*(n)

```
//Input: A nonnegative integer n
S ← 0
for i ← 1 to n do
    S ← S + i * i
return S
```

a. **Output của thuật toán này là gì?**
b. **Basic operation của thuật toán này là gì?**
c. **Tính số lần thực thi basic operation? (Tính $C(n)$)**
d. **Lớp hiệu năng của thuật toán?**
e. **Cải thiện hoặc đề xuất một thuật toán tốt hơn và xác định lớp hiệu năng? (Nếu có)**

4. Bài tập về nhà

2. Cho thuật toán sau:

ALGORITHM *Secret*$(A[0..n-1])$

//Input: An array $A[0..n-1]$ of n real numbers

$minval \leftarrow A[0]$; $maxval \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n-1$ **do**

  if $A[i] < minval$

    $minval \leftarrow A[i]$

  if $A[i] > maxval$

    $maxval \leftarrow A[i]$

return $maxval - minval$

a. Output của thuật toán này là gì?
b. Basic operation của thuật toán này là gì?
c. Tính số lần thực thi basic operation? (Tính $C(n)$)
d. Lớp hiệu năng của thuật toán?
e. Cải thiện hoặc đề xuất một thuật toán tốt hơn và xác định lớp hiệu năng? (Nếu có)

4. Bài tập về nhà

3. Cho thuật toán sau:

ALGORITHM *Enigma*$(A[0..n-1, 0..n-1])$

```
//Input: A matrix A[0..n − 1, 0..n − 1] of real numbers
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i, j] ≠ A[j, i]
            return false
return true
```

a. Output của thuật toán này là gì?
b. Basic operation của thuật toán này là gì?
c. Tính số lần thực thi basic operation? (Tính $C(n)$)
d. Lớp hiệu năng của thuật toán?
e. Cải thiện hoặc đề xuất một thuật toán tốt hơn và xác định lớp hiệu năng? (Nếu có)

4. Bài tập về nhà

4. Cho thuật toán sau:

ALGORITHM $GE(A[0..n-1, 0..n])$

```
//Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        for k ← i to n do
            A[j, k] ← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i]
```

a. Xác định lớp hiệu năng thời gian của thuật toán này?
b. Xác định sự kém hiệu quả trong thuật toán này và làm sao để tăng tốc độ của thuật toán?