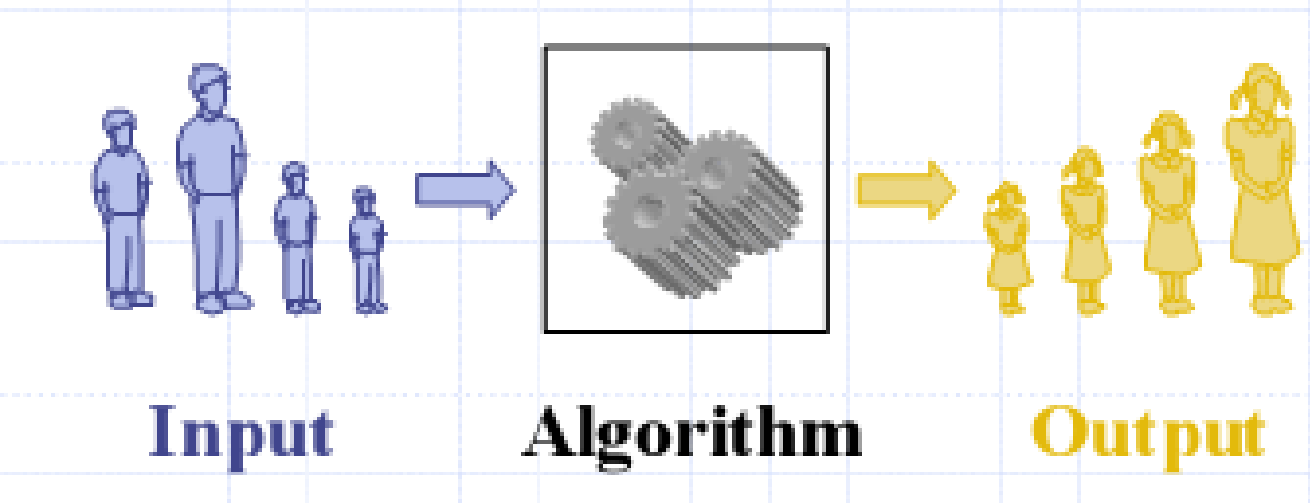
Bài 4. Phân tích các thuật toán
(Analysis of Algorithms)



Thuật toán là một qui trình thực hiện từng bước, từng bước giải quyết một vấn đề trong một khoảng thời gian hữu hạn.

Các khía cạnh cần phân tích

- Bộ nhớ (Space)

  Xác định tổng dung lượng bộ nhớ cần thiết để lưu trữ toàn bộ dữ liệu đầu vào, trung gian và kết quả đầu ra.

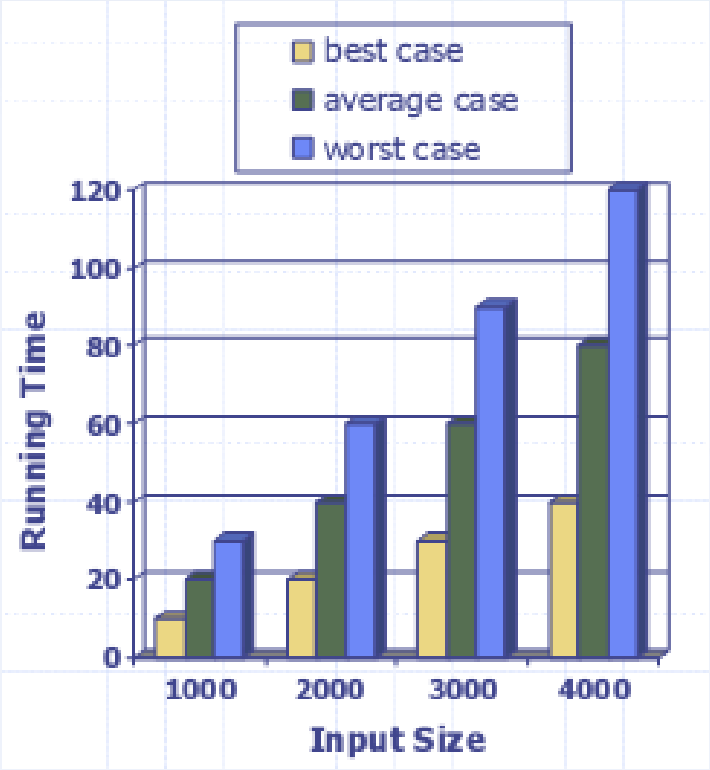
  - Ví dụ: Sắp xếp một dãy n phần tử.

    Bộ nhớ cần cho bài toán là: Bộ nhớ lưu biến n, lưu n phần tử của dãy, lưu các biến i, j, tg (nếu là thuật toán Bubble Sort)

- Thời gian chạy của thuật toán (Running time)

Thời gian chạy (Running time)

- Hầu hết các thuật toán thực hiện biến đổi các đối tượng đầu vào thành các đối tượng đầu ra
- Thời gian chạy của thuật được đặc trưng bởi kích thước của dữ liệu đầu vào
- Chúng ta thường đi đánh giá thời gian chạy của thuật toán trong 3 trường hợp: xấu nhất, trung bình và tốt nhất.
- Thời gian chạy trung bình của thuật toán thường rất khó xác định
- **Chung ta tập chung vào phân tích thời gian chạy trong trường hợp xấu nhất (do dễ phân tích)**



Phương pháp đánh giá

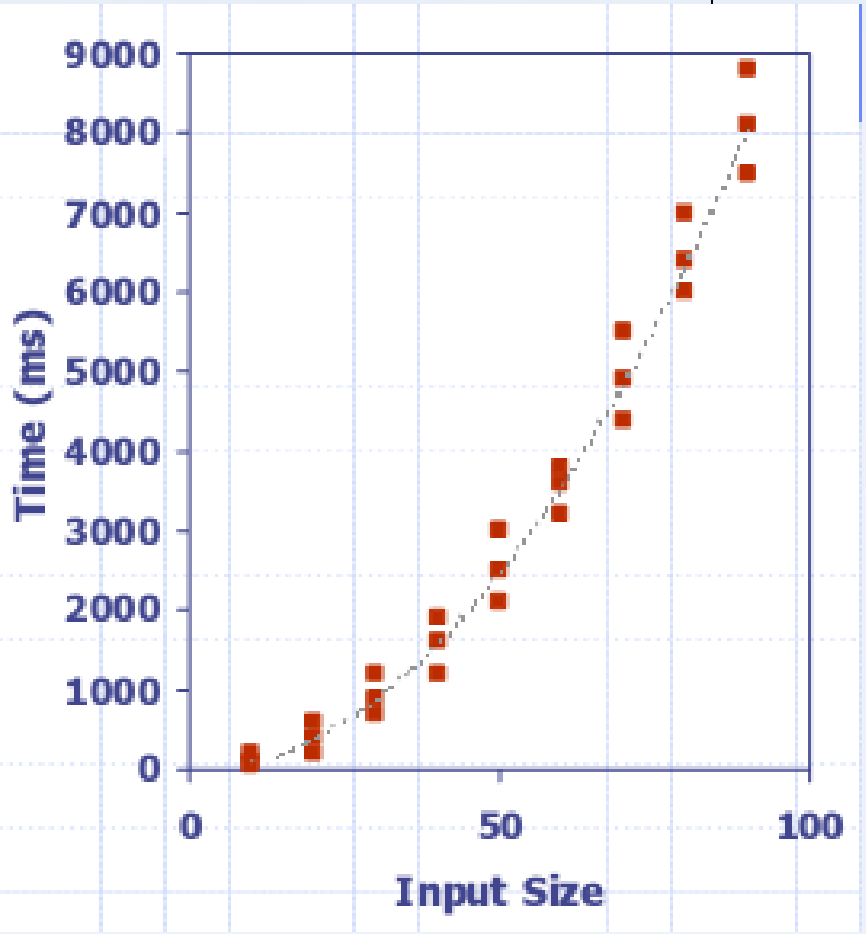
1. **Phương pháp thực nghiệm**

2. **Phương pháp phân tích lý thuyết**

Phương pháp thực nghiệm

Các bước thực hiện:

- Viết một chương trình thể hiện thuật toán
- Chạy chương trình với các bộ dữ liệu đầu vào có kích thước khác nhau và tổng hợp lại.
- Sử dụng một hàm như một đồng hồ để lấy chính xác thời gian chạy của thuật toán.
- Vẽ đồ thị biểu diễn kết quả



Hạn chế của phương pháp thực nghiệm

1. Cần phải cài đặt thuật toán bằng một ngôn ngữ lập trình, nhưng một số thuật toán việc cài đặt là khó.

2. Kết quả thu được không thể biểu thị cho những bộ dữ liệu đầu vào chưa được thực nghiệm

3. Để so sánh giữa hai thuật toán ta cần cài đặt trên các máy có cấu hình giống nhau cả về phần cứng/ môi trường phần mềm.

Phương pháp phân tích lý thuyết

- Sử dụng thuật toán được mô tả ở mức cao (giả mã) thay cho chương trình cài đặt.
- Mô tả thời gian chạy của thuật toán bằng một hàm phụ thuộc vào kích thước của dữ liệu đầu vào, *n*.
- Tính toán tất cả các khả năng của dữ liệu đầu vào
- Cho phép chúng ta đánh giá tốc độ của thuật toán không phụ thuộc vào phần cứng/môi trường phần mềm.

Giả mã (Pseudocode)

- Mô tả thuật toán ở mức trừu tượng cao
- Nhiều cấu trúc hơn ngôn ngữ tự nhiên
- Kém chi tiết hơn chương trình
- Sử dụng nhiều ký hiệu để mô tả

Ví dụ thuật toán tìm Max các phần tử của một mảng

Algorithm ***arrayMax(A,n)***

Input: Mảng A có n số nguyên

Output: Giá trị lớn nhất của A

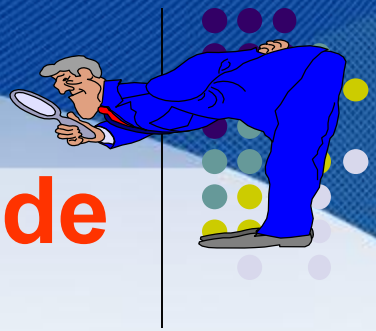
Max ← A[0]

for i ← 1 **to** n-1 **do**

if A[i] > Max **then**

Max ← A[i]

return Max



Những chi tiết mô tả PseudoCode

- **Cấu trúc điều khiển**
  - **If then else**
  - **while do**
  - **For do**
  - Xuống dòng thay cho dấu {, }
- **Khai báo phương thức**

Algorithm ***Phươngthức([Dánh sách đối])***

Input:

output:

- **Gọi hàm, phương thức**

Biến.Phươngthức([Danh sách đối])

- **Trả lại giá trị cho hàm**

return Biểu_thức

- **Các biểu thức**

← Phép gán

= phép so sánh bằng

n^2 Cho phép viết số mũ

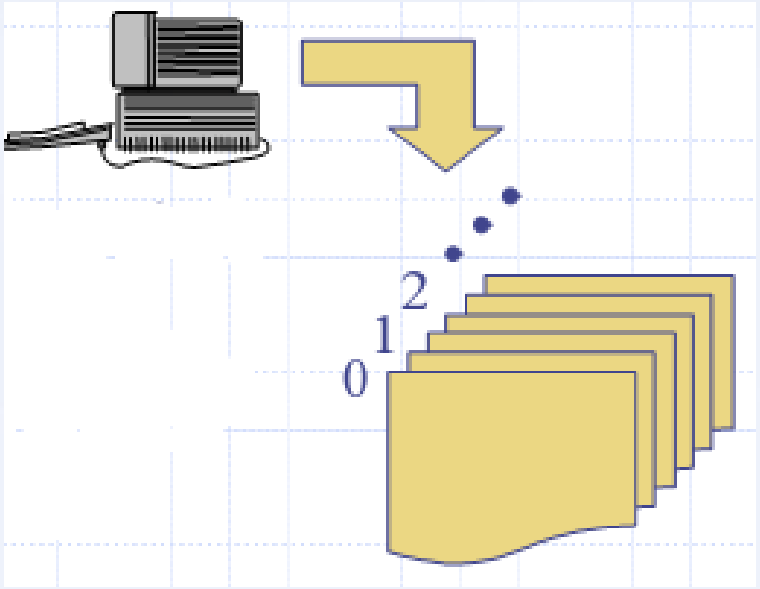
a_i cho phép viết chỉ số

Mô hình máy truy nhập ngẫu nhiên

(Random Access Machine (RAM) Model)

- Một CPU
- Không giới hạn số ô nhớ
- Mỗi ô nhớ có thể lưu một số nguyên hoặc 1 ký tự
- Mỗi ô nhớ được đánh số

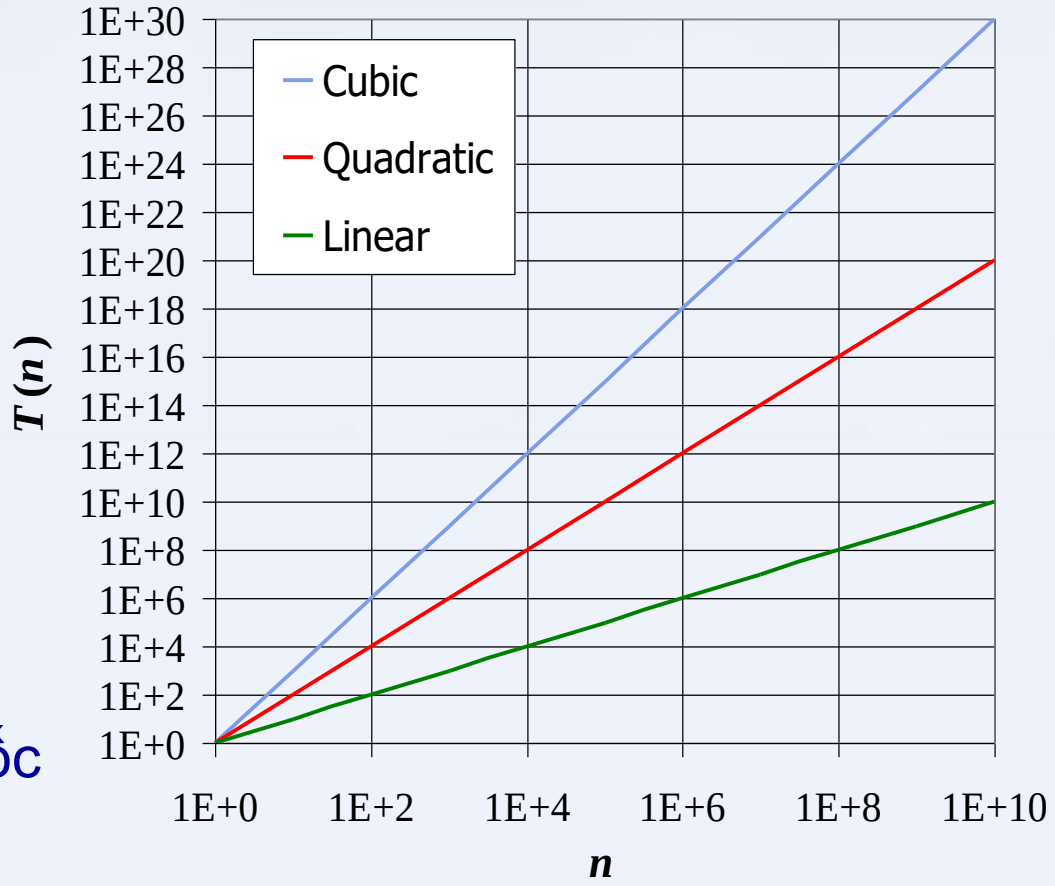
và để truy nhập đến mỗi ô nhớ sẽ mất một đơn vị thời gian



Bẩy hàm quan trọng sử dụng trong phân tích thuật toán

- Hàm hằng ≈ 1
- Hàm Logarit ≈ log n
- Hàm tuyến tính ≈ n
- N-Log-N ≈ n log n
- Hàm bậc 2 ≈ n^2
- Hàm bậc 3 ≈ n^3
- Hàm mũ ≈ 2^n

- Trong biểu đồ log-log, độ nghiêng của đường thẳng tương ứng với tốc độ phát triển của hàm



Các phép toán cơ sở

1. Các phép toán cơ sở được thực hiện bởi thuật toán được xem là như nhau
2. Độc lập với ngôn ngữ lập trình
3. Không cần thiết xác định chính xác số lượng các phép toán
4. Giả thiết mỗi phép toán mất một khoảng thời xác định để thực hiện trong mô hình RAM

Các phép toán cơ sở

- Định giá một biểu thức
- Gán giá trị cho một biến
- Đưa vào/truy cập một phần tử mảng
- So sánh
- Cộng, trừ, nhân, chia
- Gọi hàm
- Trả lại giá trị cho hàm (return)

Xác định số phép toán cơ sở

- **Bằng cách duyệt thuật toán giả mã, chúng ta có thể xác định được số phép tính tối đa mà thuật toán có thể phải thực hiện.**
- **Từ đó ta xây dựng được một hàm thể hiện thời gian chạy của thuật toán phụ thuộc vào kích thước dữ liệu vào.**

Ví dụ:

Algorithm ***arrayMax(A,n)***	***Số phép toán***
Max ← A[0]	2
for i ← 1 **to** n-1 **do**	2+(n-1)
if A[i] > Max **then**	2(n-1)
Max ← A[i]	2(n-1)
return Max	1

Ước lượng thời gian chạy

- Thuật toán ArrayMax thực hiện 5n+1 phép tính cơ bản trong trường hợp xấu nhất
- Định nghĩa:

  a = Khoảng thời gian ngắn nhất cần để thực hiện một phép tính cơ bản

  b = Khoảng thời gian dài nhất cần để thực hiện một phép tính cơ bản
- Ký hiệu T(n) là thời gian chạy trong trường hợp xấu nhất của thuật toán ArrayMax thì:

  a(5n+1)< T(n) <b(5n+1)
- Do đó thời gian chạy T(n) được bao bởi 2 đường tuyến tính

Tốc độ phát triển của thời gian chạy

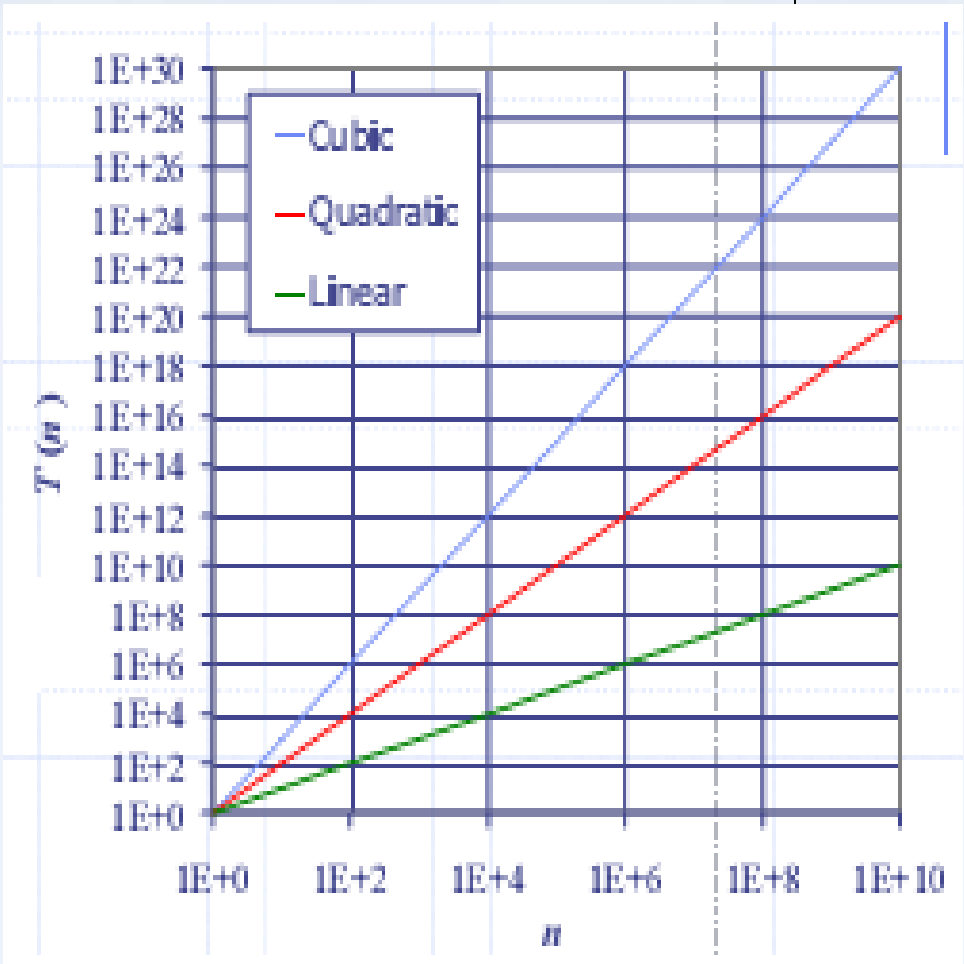
- Khi thay đổi Phần cứng/Môi trường phần mềm

  - Ảnh hưởng đến T(n) là 1 hằng số, nhưng không làm thay đổi tốc độ phát triển của T(n)

- Tốc độ phát triển tuyến tính của T(n) là bản chất của thuật toán Arraymax.

Tốc độ phát triển TG của thuật toán

- Các hàm thể hiện tốc độ phát triển TG, ví dụ như:
  - Tuyến tính : n
  - Bậc 2 : n^2
  - Bậc 3 : n^3
- Trong biểu đồ, độ nghiêng của các đường thể hiện tốc độ phát triển của các hàm

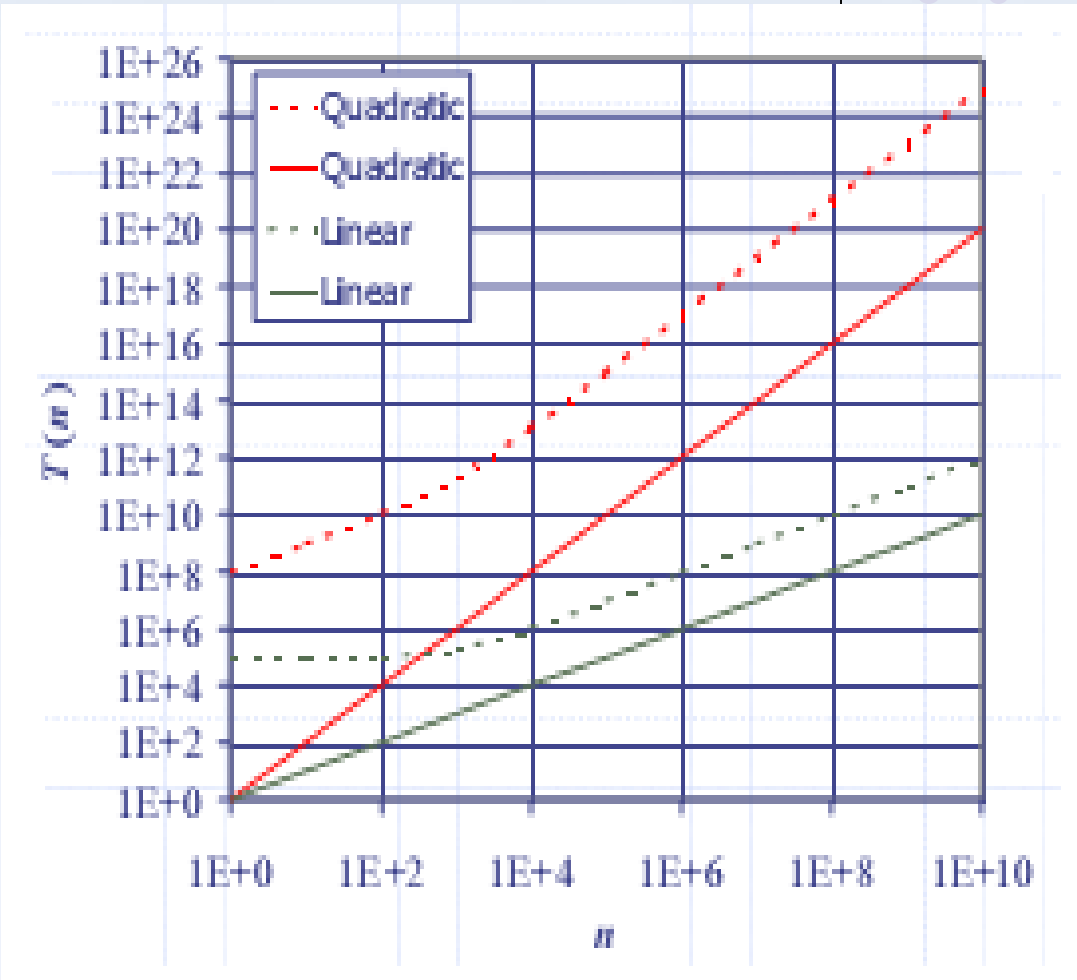


Hệ số hằng

- Tốc độ phát triển của hàm không bị ảnh hưởng bởi:
  - Hệ số hằng và
  - Số hạng bậc thấp
- Ví dụ:

  10^2n+10^5 là hàm tuyến tính

  $10^2n^2+10^5n$ là hàm bậc 2

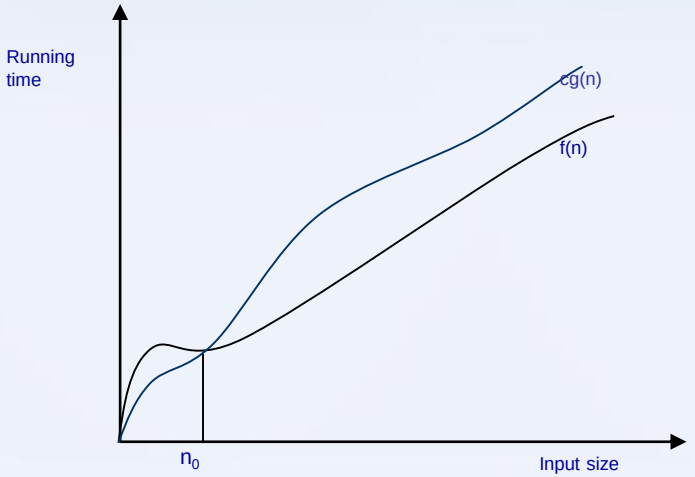


Ký hiệu ô-lớn (Big-Oh)

- Cho hàm f(n) và g(n), chúng ta nói rằng f(n) có ô lớn là g(n), kí hiệu là O(g(n)), nếu tồn tại hằng số dương c và số nguyên n_0 sao cho:

  $f(n) \leq cg(n)$ với mọi $n \geq n_0$

- Ví dụ: 2n +10 là O(n)



Thật vậy: $2n+10 \leq cn$

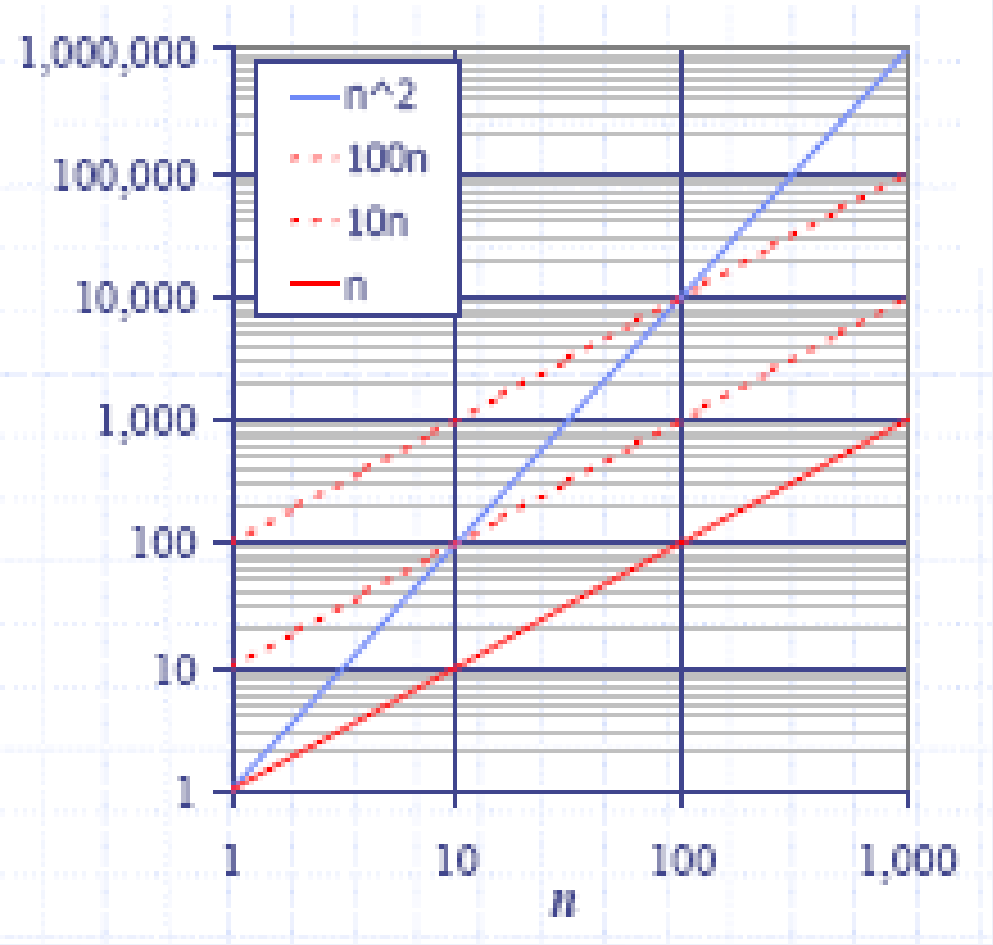
$10 \leq (c-2)n$

$10/(c-2) \leq n$

Chọn c=3 và n_0=10

Ví dụ:

- Hàm n^2 không là O(n) vì:
  - $n^2 \leq cn$
  - $n \leq c$
- Không thể xác định được hằng c số thỏa mãn điều kiện trên



Thêm một số ví dụ về ô-lớn

- 7n-2

  7n-2 là O(n)

  Vì: chọn hằng số c=7 và n_0=1 khi đó 7n-2≤cn $\forall n \geq n_0$

- $3n^3+20n^2+5$

  $3n^3+20n^2+5$ là $O(n^3)$

  Vì nếu chọn c=4 và n_0=21 khi đó $3n^3+20n^2+5 \leq cn^3$ $\forall n \geq n_0$

- 3logn+log logn

  3logn+log logn là O(logn)

  Vì nếu chọn c=4 và n_0=2 khi đó 3logn+log logn ≤ c*logn $\forall n \geq n_0$

Ô-lớn và tốc độ phát triển giá trị

- Ký hiệu Ô-lớn chỉ ra một cận trên của tốc độ phát triển giá trị của một hàm
- Ta nói "f(n) có ô lớn g(n)" có nghĩa là tốc độ phát triển giá trị của f(n) không lớn hơn tốc độ phát triển của g(n).
- Chúng ta có thể sử dụng ký hiệu Ô-lớn để xếp hạng các hàm theo thứ tự tốc độ phát triển giá trị nó.

	f(n) là O(g(n))	g(n) là O(f(n))
Tốc độ g(n) lớn hơn	Đúng	Không
Tốc độ bằng nhau	Đúng	Đúng

Qui tắc xác định Ô-lớn

- **Nếu f(n) là đa thức bậc d thì f(n) là $O(n^d)$**
  - Bỏ qua các số hạng bậc thấp (quy tắc max)
  - Bỏ qua các hệ số hằng (quy tắc bỏ qua hằng số)
- **Sử dụng lớp hàm nhỏ nhất có thể**
  - Ta nói "2n là O(n)" thay cho "2n là $O(n^2)$"
- **Sử dụng lớp hàm đơn giản nhất có thể**

  Ta nói "3n+5 là O(n)" thay cho "3n+5 là O(3n)"

Phân tích tiệm cận

- Việc phân tích thời gian chạy tiệm cận của một thuật toán được xác định bằng ký hiệu Ô-lớn (O)
- **Thực hiện phân tích:**

  - Tìm số phép toán cơ bản cần phải thực hiện trong trường hợp xấu nhất, thể hiện bằng một hàm phụ thuộc vào kích thước của dữ liệu đầu vào.

  - Diễn tả hàm bằng ký hiệu Ô-lớn
- **Ví dụ:**

  - Chúng ta đã xác định thuật toán ArrayMax thực hiện tối đa 5n+1 phép toán cơ bản

  - Chúng ta nói rằng thuật toán ArrayMax chạy trong thời gian O(n)
- Các hệ số hằng và các số hạng bậc thấp bị bỏ qua khi xác đếm số phép toán cơ bản.

Ví dụ: Tính trung bình các phần tử đầu dãy (prefix average)

- Để minh họa phân tích tiệm cận chúng ta phân tích hai thuật toán tính trung bình các phần tử đầu dãy sau:
- Hãy tính trung bình i phần tử đầu của một mảng, với i=1,..,n. Trung bình i phần tử đầu của dãy X là:

  A[i-1]=(X[0]+X[1]+….+X[i-1])/i

Thuật toán độ phức tạp bậc hai

- Thuật toán được định nghĩa như sau:

  Algorithm prefxAverage(X, n)

  Input: mảng X có n số nguyên

  Output: Mảng trung bình các phần tử đầu dãy của X

```
A ← new float[n];                          n+1
for i ←0 to n-1 do                         n+2
        s ← X[0];                          2n
        for j ← 1 to i do                  1 + 2+…+(n-1) + n + n
                s ← s + X[j];              3(1+2+…+(n-1))
        A[i] ← s/(i+1);                    4n
return A;                                  1
```

Thuật toán độ phức tạp bình phương

- Tổng số phép toán tối đa thuật toán thực hiện là:

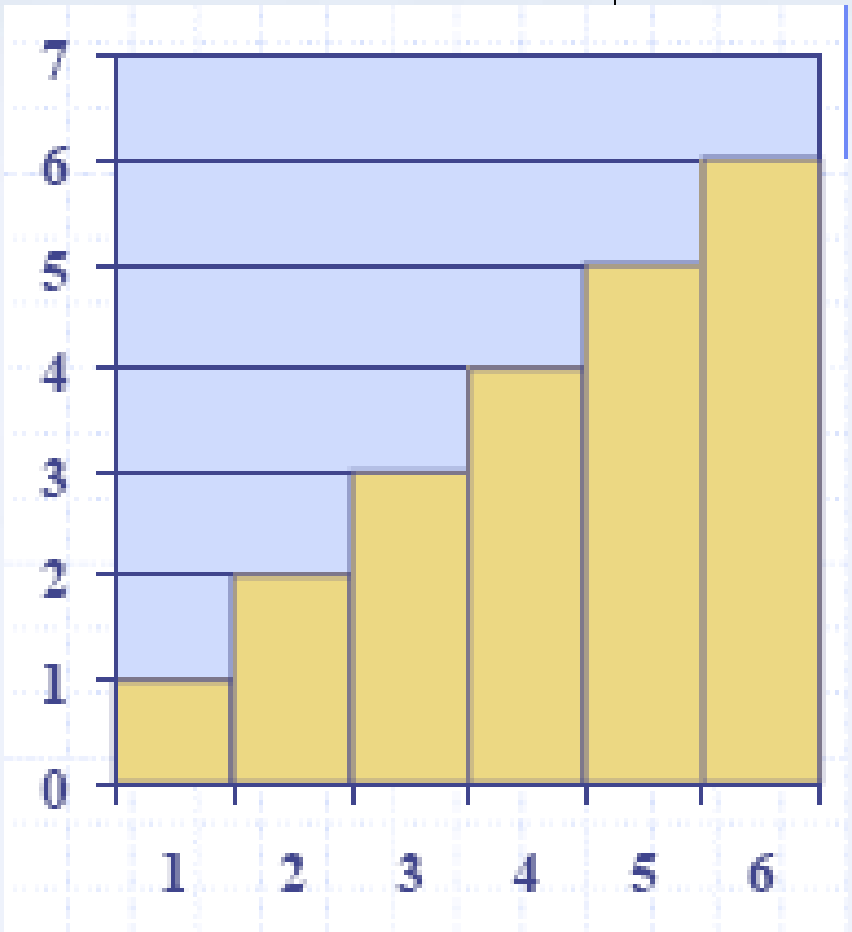
  $T(n) = 4(1+2+\ldots+(n-1))+10n+5$

  $T(n) = 4(1+2+\ldots+n) + 6n+5$

- Tổng của n số nguyên đầu là $n(n+1)/2$

  - Hình bên minh họa tốc độ gia tăng thời gian tnực hiện của thuật toán

$$T(n) = 2n^2+ 8n+5$$



Thời gian chạy của thuật toán

- Thời gian chạy của thuật toán prefixAverages1 là:

  $O(2n^2+ 8n+4)$
- Do đó thuật toán prefixAveragres1 có thời gian chạy là $O(n^2)$

Thuật toán độ phức tạp bậc nhất (tuyến tính)

- Thuật toán được mô tả như sau:

Algorithm prefxAverage(X, n)
Input: mảng X có n số nguyên

Output: Mảng trung bình các phần tử đầu dãy của X

```
A ← new float[n];                    n+1
s ← 0;                               1
for i ←0 to n-1 do                   n+3
      s ← s + X[i];                  3n
      A[i] ← s/(i+1);                4n
return A;                            1
```

- Tổng số phép toán tối đa cần phải thực hiện là
  - T(n) = 9n + 6
- Độ phức tạp tiệm cận của thuật toán prefixAverages2 là O(n)

Qui tắc xác định độ phức tạp của thuật toán

- **Qui tắc cộng và lấy max**

Nếu một thuật toán thực hiện hai đoạn chương trình P1, P2 rời nhau và có độ phức tạp của mỗi đoạn chương trình tương ứng là O(g(n)) và O(f(n)). Khi đó độ phức tạp của thuật toán là: T(n) = O(max{g(n),f(n)}).

- **Ví dụ:**

```
for i = 1 to n do
   input a[i];

Min ← a[0];

 for i=1 to n-1 do
     if a[i]<min then
           min←a[i];
```

P1 có thời gian chạy là O(n)

P2 có thời gian chạy là O(1)

P3 có thời gian chạy là O(n)

Vậy thời gian chạy của cả thuật toán là: T(n) = O(max{1, n, n})=O(n)

Qui tắc xác định độ phức tạp của thuật toán (tiếp)

- **Qui tắc nhân**

Nếu một thuật toán thực hiện hai đoạn chương trình P1, P2, trong đó P2 lồng trong P1và có độ phức tạp của mỗi đoạn chương trình tương ứng là O(g(n)) và O(f(n)). Khi đó độ phức tạp của thuật toán là: T(n) = O(g(n)*f(n)).

- **Ví dụ:**

```
for i = 1 to n do
   input a[i];

for i ← 1 to n-1 do
    for j ← i+1 to n do
        if a[i]<a[j] then
              tg ← a[i];
              a[i] ← a[j];
              a[j] ← tg;
```

P1 có thời gian chạy là O(n)

P2 có thời gian chạy là O(n)

P3 có thời gian chạy là O(n)

Qui tắc xác định độ phức tạp của thuật toán(tiếp)

- Ta thấy đoạn chương trình P3 lồng trong đoạn chương trình P2. Áp dụng qui tắc nhân thì độ phức tạp của đoạn chương trình P2 và P3 là: O(n*n) hay $O(n^2)$.

- Áp dụng qui tắc cộng và lấy max cho đoạn chương trình gồm P1, P2, P3 thì ta được độ phức tạp của thuật toán: $T(n) = O(n^2)$.

Một số hàm sử dụng để đánh giá tốc độ gia tăng thời gian chạy.

Constant	Logarithm	Linear	n-log-n	Quadratic	cubic	exponent
1	logn	n	nlogn	n^2	n^3	a^n