

# CHƯƠNG 1

# KỸ THUẬT

# PHÂN TÍCH THUẬT TOÁN

Bộ môn CÔNG NGHỆ PHẦN MỀM  
Khoa Công nghệ Thông tin & Truyền thông  
ĐẠI HỌC CẦN THƠ



# MỤC TIÊU

- **Sau khi học xong chương này, sinh viên cần:**
  - Hiểu sự cần thiết phải phân tích đánh giá thuật toán.
  - Biết các tiêu chuẩn để đánh giá một thuật toán.
  - Hiểu khái niệm độ phức tạp của thuật toán.
  - Vận dụng các quy tắc để tính độ phức tạp của chương trình không gọi chương trình con, chương trình có gọi các chương trình con không đệ quy.
  - Vận dụng các phương pháp thành lập phương trình đệ quy.
  - Vận dụng các phương pháp giải phương trình đệ quy



# Khái niệm THUẬT TOÁN

- Khái niệm **thuật toán** (Algorithm)
  - Thuật toán là một dãy xác định các thao tác cơ bản áp dụng trên dữ liệu vào nhằm đạt được giải pháp cho một vấn đề.
  - Hai vấn đề :
    - (1) Tìm một **phương pháp** giải quyết vấn đề ?
      - Giải pháp cho  $ax^2 + bx + c = 0$  : rõ ràng và xác định
      - Giải pháp cho  $ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$  : không có giải pháp tổng quát
    - (2) Tìm một giải pháp **hiệu quả** ?
  - Phân biệt **thuật toán** và **chương trình**
    - *Chương trình* là cài đặt *thuật toán* bằng một ngôn ngữ lập trình



# THUẬT TOÁN LÀ GÌ ?

- Thuật toán

- Thủ tục tính toán nhận tập các **dữ liệu vào** (input) và tạo các **dữ liệu ra** (output)



- Thuật toán được gọi là **đúng đắn** (correct), nếu **thuật toán** **dùng** cho **kết quả đúng** với mọi dữ liệu vào



# VÍ DỤ VỀ THUẬT TOÁN ?

- Thuật toán hằng ngày trong cuộc sống
  - Nấu cơm
  - Gọi điện thoại
  - ...
- Thuật toán trong toán học/tin học
  - Nhân hai ma trận
  - Tính tích phân
  - Giải hệ phương trình bậc nhất
  - ...



# Các tính chất của THUẬT TOÁN

- **Tính đúng đắn:** Thuật toán cần đảm bảo cho một kết quả đúng.
- **Tính tổng quát:** Thuật toán phải giải quyết được cho một lớp bài toán tương tự.
- **Tính hữu hạn:** Thuật toán phải dừng sau một số bước xác định.
- **Tính xác định:** Các thao tác trong thuật toán phải được đặc tả chặt chẽ, rõ ràng, cụ thể.
- **Tính hiệu quả:** Thuật toán phải sử dụng hiệu quả nguồn tài nguyên máy tính: Thời gian, Bộ nhớ



# Đặc tả THUẬT TOÁN

– Có nhiều cách đặc tả thuật toán

- **Không hình thức** : Ngôn ngữ tự nhiên

**Ví dụ:** mô tả thuật toán tìm ước số chung lớn nhất của hai số nguyên.

Input: Hai số nguyên  $a, b$ .

Output: Ước số chung lớn nhất của  $a, b$ .

**Thuật toán:**

Bước 1: Nếu  $a=b$  thì  $USCLN(a, b)=a$ .

Bước 2: Nếu  $a > b$  thì tìm  $USCLN$  của  $a-b$  và  $b$ , quay lại bước 1;

Bước 3: Nếu  $a < b$  thì tìm  $USCLN$  của  $a$  và  $b-a$ , quay lại bước 1;



- **Nửa hình thức** : Kết hợp ngôn ngữ tự nhiên và các kí hiệu toán học : Lưu đồ, Sơ đồ khối, ...

```

graph TD
    1([Bắt đầu]) --> 2([Kết thúc])
    3[Câu lệnh] --> 4{Điều kiện}
    4 -- Đ --> Exit(( ))
    4 -- S --> 5[/Nhập xuất dữ liệu/]
    5 --> 3
  
```





# Đặc tả THUẬT TOÁN

- **Hình thức** : Ngôn ngữ giả (pseudocode).  
Ngôn ngữ Z, ngôn ngữ B, ...

**if** Delta > 0 **then begin**

$$x_1 = (-b - \sqrt{\text{delta}}) / (2 * a)$$

$$x_2 = (-b + \sqrt{\text{delta}}) / (2 * a)$$

xuất kết quả : phương trình có hai nghiệm là  $x_1$  và  $x_2$

**end**

**else**

**if** delta = 0 **then**

xuất kết quả : phương trình có nghiệm kép là  $-b / (2 * a)$

**else** {trường hợp delta < 0 }

xuất kết quả : phương trình vô nghiệm



# Sự cần thiết phải phân tích, đánh giá thuật toán

- Cần phải phân tích, đánh giá thuật toán để:
  - Lựa chọn một **thuật toán tốt nhất** trong các thuật toán để cài đặt chương trình giải quyết bài toán đặt ra.
  - Cải tiến thuật toán hiện có để được một thuật toán tốt hơn.



# Tiêu chuẩn đánh giá thuật toán

- Một thuật toán được xem là tốt nếu nó đạt các tiêu chuẩn sau:

## (1) Tính đúng đắn

- Chạy trên dữ liệu thử
- Chứng minh lý thuyết (bằng toán học chẳng hạn)

## (2) Tính đơn giản

## (3) Tính nhanh chóng (thời gian thực thi)

- Rất quan trọng khi chương trình thực thi nhiều lần  
= *Hiệu quả thời gian thực thi*



# Thời gian thực hiện chương trình

- Thời gian thực hiện một chương trình là một hàm của kích thước dữ liệu vào, ký hiệu  **$T(n)$**  trong đó  **$n$**  là *kích thước (độ lớn) của dữ liệu vào*.

**Ví dụ :** Chương trình tính **tổng của  $n$  số** có thời gian thực hiện là  **$T(n) = Cn$**  trong đó  $C$  là một hằng số.

- Thời gian thực hiện chương trình là một hàm không âm, tức là  $T(n) \geq 0, \forall n \geq 0$ .



# Đơn vị đo thời gian thực hiện

- Đơn vị của  **$T(n)$**  :
  - $T(n)$  không phải là đơn vị đo thời gian bình thường như giờ, phút, giây.
  - $T(n)$  xác định bởi *số các lệnh/chỉ thị* được thực hiện trong một *máy tính lý tưởng*.
  - $T$  phụ thuộc vào kích thước dữ liệu nhập và tính chất của dữ liệu.
  - Tính thời gian cần tính trường hợp xấu nhất.
- **Ví dụ:** Khi nói thời gian thực hiện của một chương trình là  **$T(n) = Cn$**  thì có nghĩa là chương trình cần  **$Cn$**  lệnh/chỉ thị thực thi.



# Tính ĐỘ PHỨC TẠP

- Độ phức tạp phụ thuộc vào *dữ liệu* sử dụng bởi thuật toán
- **Trường hợp tốt nhất**
  - Dữ liệu được tổ chức sao cho thuật toán hoạt động hiệu quả nhất
  - Giá trị **nhỏ nhất** của độ phức tạp thực tế
- **Trường hợp xấu nhất**
  - Dữ liệu được tổ chức sao cho thuật toán hoạt động kém hiệu quả nhất
  - Giá trị **lớn nhất** của độ phức tạp thực tế
- **Trường hợp trung bình**
  - Dữ liệu tương ứng với sự tổ chức được xem là trung bình
  - Không đơn giản để xác định
  - Thường có ý nghĩa nhất
  - Không là trung bình của độ phức tạp tốt nhất và độ phức tạp xấu nhất



# Thời gian thực hiện: 3 trường hợp

- Thời gian thực hiện chương trình không chỉ phụ thuộc vào *kích thước* mà còn phụ thuộc vào *tính chất* của dữ liệu vào (cùng kích thước dữ liệu vào nhưng thời gian thực hiện chương trình khác nhau)
- Ví dụ :** *Tìm kiếm tuần tự*

MÔ PHỎNG VỚI  $N = 10$  và DÃY A SAU:

5, 7, 1, 4, 2, 9, 8, 11, 25, 51

$k = 2,$

A	5	7	1	4	2	9	8	11	25	51
i	1	2	3	4	5					



# Thời gian thực hiện trong trường hợp xấu nhất

- (1) Trường hợp tốt nhất : so sánh 1 lần
- (2) Trường hợp trung bình: so sánh  $n/2$  lần
- (3) Trường hợp xấu nhất : so sánh  $n$  lần

•Vì vậy, thường xem  $T(n)$  là thời gian thực hiện chương trình trong **trường hợp xấu nhất** trên dữ liệu vào có kích thước  $n$ .

Hay:  $T(n)$  là **thời gian lớn nhất** để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước  $n$ .





# Tỷ suất tăng của hàm

- Ta nói hàm không âm  **$T(n)$**  có **tỷ suất tăng** (*growth rate*)  **$f(n)$**  nếu tồn tại các hằng số  $C$  và  $N_0$  sao cho  $T(n) \leq Cf(n)$ ,  $\forall n \geq N_0$ .

**Tỷ suất tăng  $f(n)$  = tốc độ tăng của hàm khi  $n$  tăng**

VD :  $\forall n \geq 0$ , hàm  $n^3$  có tốc độ tăng cao hơn  $n^2$  khi  $n$  tăng

- Ta có thể chứng minh được “*Cho một hàm không âm  $T(n)$  bất kỳ, luôn tìm được tỷ suất tăng  $f(n)$  của nó*”.



# Ví dụ về tỷ suất tăng

- **Ví dụ 1:** Xét hàm  $T(n) = (n+1)^2$ . Đặt  $N_0 = 1$  và  $C = 4$  thì  $\forall n \geq 1$ , ta luôn có  $T(n) = (n+1)^2 \leq 4n^2$ , tức là *tỷ suất tăng của  $T(n)$  là  $f(n) = n^2$*
- **Ví dụ 2:** Xét hàm  $T(n) = 3n^3 + 2n^2$ . Cho  $N_0 = 0$  và  $C = 5$ , ta có thể chứng minh  $\forall n \geq 0: 3n^3 + 2n^2 \leq 5n^3$  hay *tỷ suất tăng của  $T(n)$  là  $f(n) = n^3$*
- *Tuy nhiên, rất khó xác định tỷ suất tăng bằng cách như trên mà thường áp dụng quy tắc sau:*

**Quy tắc vận dụng:** Nếu  $T(n)$  là một đa thức của  $n$  thì tỷ suất tăng của  $T(n)$  là  *$n$  với số mũ cao nhất.*



# Khái niệm độ phức tạp của thuật toán

- Giả sử có 2 thuật toán P1 và P2 với *thời gian thực hiện* tương ứng  $T_1(n) = 100n^2$  và  $T_2(n) = 5n^3$ .

- **Vấn đề** : P1 hay P2 nhanh hơn?

- **Cách giải quyết**: So sánh  $T_1(n)$  và  $T_2(n)$

- **Kết quả** : Phụ thuộc vào  $n$

Khi  $n \leq 20$  :  $T_1(n) \geq T_2(n) \rightarrow P_2$  nhanh hơn  $P_1$

Khi  $n > 20$  :  $T_1(n) < T_2(n) \rightarrow P_1 (n^2)$  nhanh hơn ( $<$ )  $P_2 (n^3)$

- Như vậy, một cách hợp lý là nên xét *tỷ suất tăng của hàm thời gian* thực hiện chương trình thay vì xét *thời gian thực hiện*.

Khi đó: P1 thực hiện nhanh hơn P2 vì tỷ suất tăng  $n^2 < n^3, \forall n \geq 0$

**Tỷ suất tăng của hàm thời gian = Độ phức tạp của thuật toán**



# Khái niệm độ phức tạp của thuật toán

- **Ký pháp Ô lớn** (big-O notation): Cho một thuật toán P có *thời gian thực hiện* là hàm  $T(n)$ , nếu  $T(n)$  có tỷ suất tăng là  $f(n)$  thì thuật toán P có độ phức tạp là  $f(n)$  và **ký hiệu thời gian thực hiện  $T(n)$  là  $O(f(n))$**  (đọc là “ô  $f(n)$ ”).
- **Ví dụ:** - Hàm  $T(n) = (n + 1)^2$  có tỷ suất tăng  $f(n)$  là  $n^2$  nên thời gian thực hiện của  $T(n)$  sẽ là  **$O(f(n)) = O(n^2)$** 
  - Hàm  $T(n) = 3n^3 + 2n^2$  có tỷ suất tăng  $f(n)$  là  $n^3$  nên thời gian thực hiện của  $T(n)$  sẽ là  **$O(f(n)) = O(n^3)$**
- **Tính chất:** -  $O(C \cdot f(n)) = O(f(n))$  với  $C$  là hằng số.
  - Đặc biệt  $O(C) = O(1)$

Lưu ý: - Độ phức tạp của thuật toán là một hàm chặn trên của hàm thời gian.  
- Hằng nhân tử  $C$  trong hàm chặn trên thường không có ý nghĩa.



# Các hàm độ phức tạp thường gặp

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
$O(n^2)$	Bình phương
$O(n^3)$	Bậc ba
...	
$O(n^m)$	Đa thức
$O(c^n)$ , với $c > 1$	Mũ
$O(n!)$	Giai thừa

$\sim \log n$

Đa thức

Độ phức tạp lớn

Có thể chấp nhận được

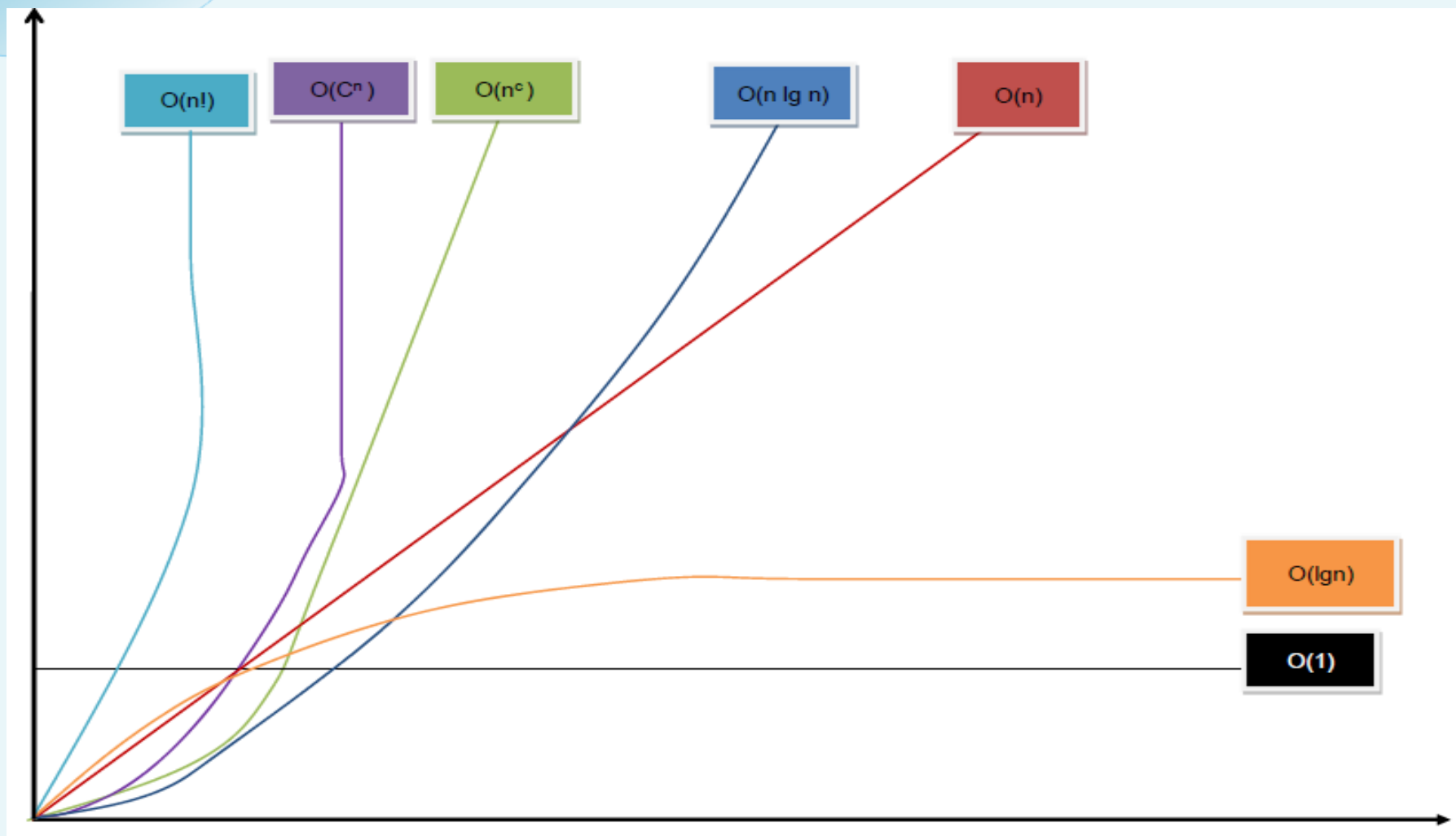
Cải tiến

Phạm Thế Bảo



CANTHO UNIVERSITY

# Đồ thị biến thiên các hàm độ phức tạp thường gặp





# Cách tính độ phức tạp

- Cho 2 đoạn chương trình:
  - P1 có thời gian thực hiện  $T1(n)=O(f1(n))$ .
  - P2 có thời gian thực hiện  $T2(n)=O(f2(n))$ .
- **Quy tắc cộng:**  
Thời gian thực hiện P1 và P2 **nối tiếp** nhau sẽ là:  
 **$T(n) = T1(n) + T2(n) = O(\max(f1(n), f2(n)))$**
- **Quy tắc nhân:**  
Thời gian thực hiện P1 và P2 **lồng nhau** (chẳng hạn vòng lặp lồng nhau) sẽ là:  
 **$T(n) = T1(n) \times T2(n) = O(f1(n) \times f2(n))$**



# Quy tắc tổng quát tính độ phức tạp

- Lệnh đọc (**read, scanf**), lệnh ghi (**write, printf**), lệnh **gán**, lệnh **return**, **định trị biểu thức**, **so sánh**: Thời gian = hằng số hay  **$O(1)$**
- Lệnh **if** :     **if** (điều kiện)  
                  lệnh 1  
                  **else**  
                  lệnh 2;  
- Thời gian =  **$\max(\text{lệnh 1}, \text{lệnh 2}) + \text{điều kiện}$**
- **Vòng lặp**:
  - Thời gian = **Tổng thời gian thực hiện thân vòng lặp**
  - Vòng lặp có số lần lặp xác định (**for**): tính số lần lặp.
  - Vòng lặp có số lần lặp không xác định (**while**): tính số lần lặp trong *trường hợp xấu nhất* (lặp nhiều nhất).





CANTHO UNIVERSITY

# Phương pháp tính độ phức tạp

- Xét phương pháp tính độ phức tạp trong **3 trường hợp**:
  - (1) Chương trình **không gọi** chương trình con.
  - (2) Chương trình **có gọi** chương trình con không đệ quy.
  - (3) Chương trình **đệ quy**.
- Phương pháp thực hiện:
  - Xác định đầu vào: thường ký hiệu là  $n$
  - Cách 1: dùng cho tất cả các loại chương trình
    - Tính thời gian thực hiện  $T(n)$  cho toàn bộ chương trình  $\rightarrow O(f(n))$  từ  $T(n)$
  - Cách 2: không áp dụng cho chương trình đệ quy
    - Chia chương trình nhiều đoạn nhỏ
    - Tính  $T(n)$  và  $O(f(n))$  cho từng đoạn
    - Áp dụng quy tắc cộng, quy tắc nhân để có  $O(f(n))$  cho cả chương trình



# Ví dụ 1.

## Thủ tục sắp xếp “nổi bọt”

(1) Chương trình **không gọi** chương trình con.

```
void BubbleSort(int a[], int n)
{
    int i,j,temp;
1.    for(i= 0; i<=n-2; i++)
2.        for(j=n - 1; j>=i+1;j--)
3.            if (a[j-1] > a[j]) {
4.                temp=a[j-1];
5.                a[j-1] = a[j];
6.                a[j]   = temp;
            }
}
```



# Tính độ phức tạp của thủ tục sắp xếp “nổi bọt”

- Chương trình sử dụng các vòng lặp xác định. Toàn bộ chương trình chỉ gồm một lệnh lặp {1}, lồng trong lệnh {1} là lệnh lặp {2}, lồng trong lệnh {2} là lệnh {3} và lồng trong lệnh {3} là 3 lệnh nối tiếp nhau {4}, {5} và {6}.
- 3 lệnh gán {4}, {5} và {6} đều tốn  $O(1)$  thời gian, việc so sánh  $a[j-1] > a[j]$  cũng tốn  $O(1)$  thời gian, do đó lệnh {3} tốn  $O(1)$  thời gian.
- Vòng lặp {2} thực hiện  $(n-i-1)$  lần, mỗi lần  $O(1)$  do đó vòng lặp {2} tốn  $O((n-i-1) \times 1) = O(n-i-1)$ .
- Vòng lặp {1} có  $i$  chạy từ 0 đến  $n-2$  nên thời gian thực hiện của vòng lặp {1} cũng là độ phức tạp của thuật toán:

$$T(n) = \sum_{i=0}^{n-2} (n-i-1) = \frac{n(n-1)}{2} = O(n^2)$$



# Ví dụ 2.

## Thủ tục tìm kiếm tuần tự

(1) Chương trình **không gọi** chương trình con.

```
1.  int tim_kiem(int x, int a[], int n) {  
2.      int found, i;  
3.      found = 0;  
4.      i = 0;  
5.      while (i < n && !found)  
6.          if (a[i] == x)  
7.              found = 1;  
8.          else  
9.              i = i+1;  
10.     return i;  
11. }
```



# Tính độ phức tạp của hàm tìm kiếm tuần tự

- Các lệnh  $\{3\}$ ,  $\{4\}$ ,  $\{5\}$  và  $\{10\}$  nối tiếp nhau, do đó độ phức tạp của hàm Search là độ phức tạp lớn nhất trong 4 lệnh này.
- 3 lệnh  $\{3\}$ ,  $\{4\}$  và  $\{10\}$  đều có độ phức tạp  $O(1)$ , do đó độ phức tạp lớn nhất trong 4 lệnh này là độ phức tạp của lệnh  $\{5\}$ . Lồng trong lệnh  $\{5\}$  là lệnh  $\{6\}$ , lệnh  $\{6\}$  có độ phức tạp  $O(1)$ .
- Lệnh  $\{5\}$  là một vòng lặp không xác định, do đó cần tính số lần lặp trong trường hợp xấu nhất (khi tất cả các phần tử của mảng  $a$  đều khác  $x$ ): xét tất cả các  $a[i]$ ,  $i$  có các giá trị từ 0 đến  $(n - 1)$  hay vòng lặp  $\{3\}$  thực hiện  $n$  lần, tốn  $O(n)$  thời gian.

Vậy  $T(n) = O(n)$



# Bài tập

**Bài 1:** Tính thời gian thực hiện của các đoạn chương trình sau:

a) Tính tổng của các số

```
{1} Sum := 0;  
{2} for i:=1 to n do begin  
{3}     readln(x);  
{4}     Sum := Sum + x;  
end;
```

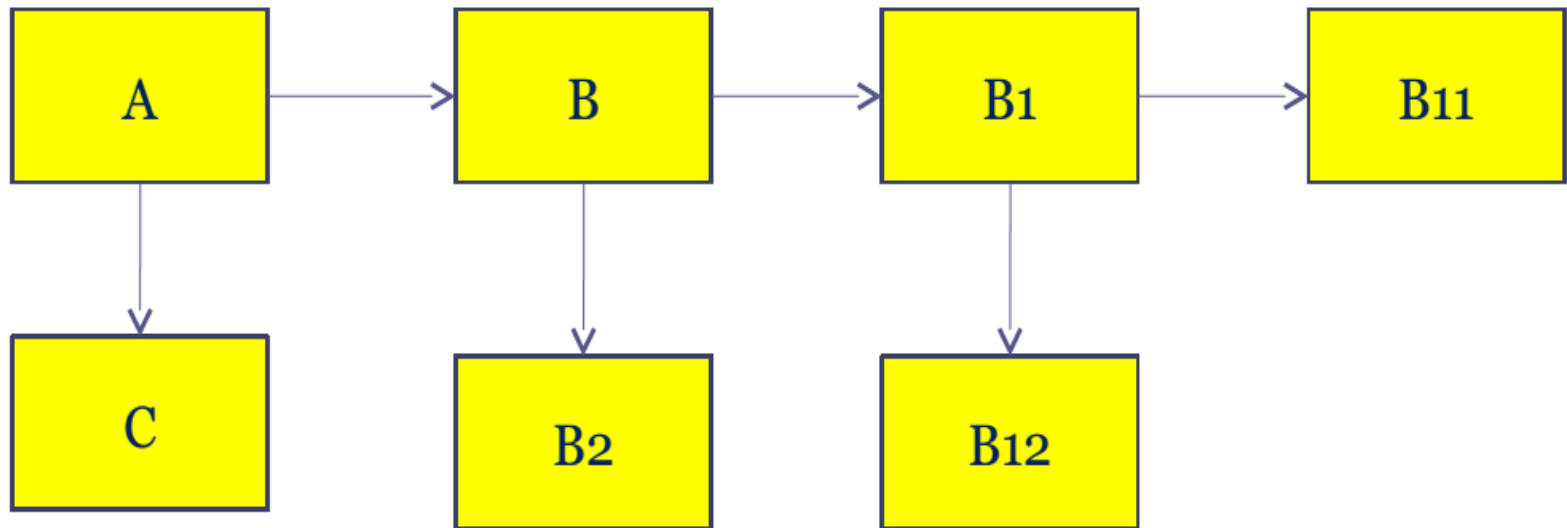
b) Tính tích hai ma trận vuông cấp  $n$   $C = A * B$ :

```
{1} for i := 1 to n do  
{2}     for j := 1 to n do begin  
{3}         c[i,j] := 0;  
{4}         for k := 1 to n do  
{5}             c[i,j] := c[i,j] + a[i,k] * b[k,j];  
end;
```



# Độ phức tạp của chương trình có gọi chương trình con không đệ quy

- Quy tắc: tính từ trong ra ngoài



- C, B2, B12, B11
- B1
- B
- A



## Ví dụ 3.

# Sắp xếp “nổi bọt” gọi chương trình con

## (2) Chương trình **có gọi** chương trình con không đệ quy

```
1. void Swap (int &a,int &b) {  
2.     int temp;  
3.     temp = x;  
4.     x = y;  
5.     y = temp; }  
  
6. void BubbleSort(int a[],int n) {  
7.     for(i= 0; i<=n-2; i++)  
8.         for(j=n - 1; j>=i+1;j--)  
9.             if (a[j-1] > a[j]) {  
10.                Swap(a[j-1],a[j]); }  
}
```





# Tính độ phức tạp của sắp xếp “nổi bọt” có gọi chương trình con

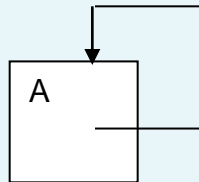
- Chương trình BubbleSort gọi chương trình con Swap, do đó để tính độ phức tạp của BubbleSort trước tiên cần tính độ phức tạp của Swap.
- Swap bao gồm 3 lệnh gán 3 lệnh gán {3}, {4} và {5} đều tốn  $O(1)$ , do đó độ phức tạp của Swap là  $O(1)$ .
- Trong BubbleSort: lệnh {10} gọi Swap nên tốn  $O(1)$ ; lệnh {9} có điều kiện so sánh  $a[j] < a[j-1]$  cũng tốn  $O(1)$ ; vòng lặp {8} thực hiện  $(n - i - 1)$  lần, mỗi lần tốn  $O(1)$  nên vòng lặp {8} tốn  $O((n-i-1) \times 1) = O(n-i-1)$ ; Vòng lặp {7} có  $i$  chạy từ 0 đến  $n-1$  nên thời gian thực hiện của vòng lặp {7} cũng là độ phức tạp của thuật toán:

$$T(n) = \sum_{i=0}^{n-2} (n-i-1) = \frac{n(n-1)}{2} = O(n^2)$$

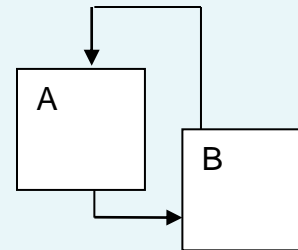


# Phân tích các chương trình đệ quy

- 2 dạng chương trình đệ quy:



**Đệ quy trực tiếp**



**Đệ quy gián tiếp**

- Tính độ phức tạp chương trình đệ quy:
  - Thành lập phương trình đệ quy  $T(n)$
  - Giải phương trình đệ quy tìm nghiệm
  - Suy ra tỷ suất tăng  $f(n)$  hay  $O(f(n))$ .



# Chương trình đệ quy

- Chương trình đệ quy giải bài toán kích thước  $n$ , phải có ít nhất một *trường hợp dừng* ứng với một  $n$  cụ thể và *lời gọi đệ quy* giải bài toán kích thước  $k$  ( $k < n$ )
- Ví dụ :** Chương trình đệ quy tính  $n!$

```
1.  int giai_thua(int n) {  
2.      if (n == 0)  
3.          return 1;  
4.      else  
5.          return n * giai_thua(n - 1);  
6.  }
```

- Trường hợp dừng*  $n = 0$  và *lời gọi đệ quy*  $k = n - 1$ .



# Thành lập phương trình đệ quy

- **Phương trình đệ quy** là phương trình biểu diễn mối liên hệ giữa  $T(n)$  và  $T(k)$ , trong đó  $T(n)$  và  $T(k)$  là thời gian thực hiện chương trình có kích thước dữ liệu nhập tương ứng là  $n$  và  $k$ , với  $k < n$ .
- Để thành lập được phương trình đệ quy, phải căn cứ vào chương trình đệ quy.
  - **Khi đệ quy dừng**: xem xét khi đó chương trình làm gì và tốn hết bao nhiêu thời gian (thông thường thời gian này là hằng số  $C(n)$ )
  - **Khi đệ quy chưa dừng**: xem xét có bao nhiêu lời gọi đệ quy với kích thước  $k$  thì sẽ có bấy nhiêu  $T(k)$ .
- Ngoài ra, còn phải xem xét thời gian phân chia bài toán và tổng hợp các lời giải (chẳng hạn gọi thời gian này là  $d(n)$ ).



# Dạng phương trình đệ quy

- Dạng tổng quát của một phương trình đệ quy sẽ là:

$$T(n) = \begin{cases} C(n) \\ F(T(k)) + d(n) \end{cases}$$

- **C(n)**: thời gian thực hiện chương trình ứng với trường hợp đệ quy dừng.
- **F(T(k))**: hàm xác định thời gian theo T(k).
- **d(n)**: thời gian phân chia bài toán và tổng hợp các kết quả.



# Ví dụ 1. Phương trình đệ quy của chương trình đệ quy tính $n!$

- Gọi  $T(n)$  là thời gian tính  $n!$ .
- Thì  $T(n-1)$  là thời gian tính  $(n-1)!$ .
- Trong trường hợp  $n = 0$  thì chương trình chỉ thực hiện một lệnh *return 1*, nên tốn  $O(1)$ , do đó ta có  $T(0) = C_1$ .
- Trong trường hợp  $n > 0$  chương trình phải gọi đệ quy *Giai\_thua(n-1)*, việc gọi đệ quy này tốn  $T(n-1)$ .
- Sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với  $n$  và trả về tích số. Thời gian để thực hiện phép nhân và trả kết quả về là một hằng  $C_2$ .
- Vậy ta có phương trình đệ quy như sau:

$$T(n) = \begin{cases} C_1 & \text{nếu } n=0 \\ T(n-1) + C_2 & \text{nếu } n>0 \end{cases}$$



# Thuật toán MergeSort

- Ý tưởng thuật toán :

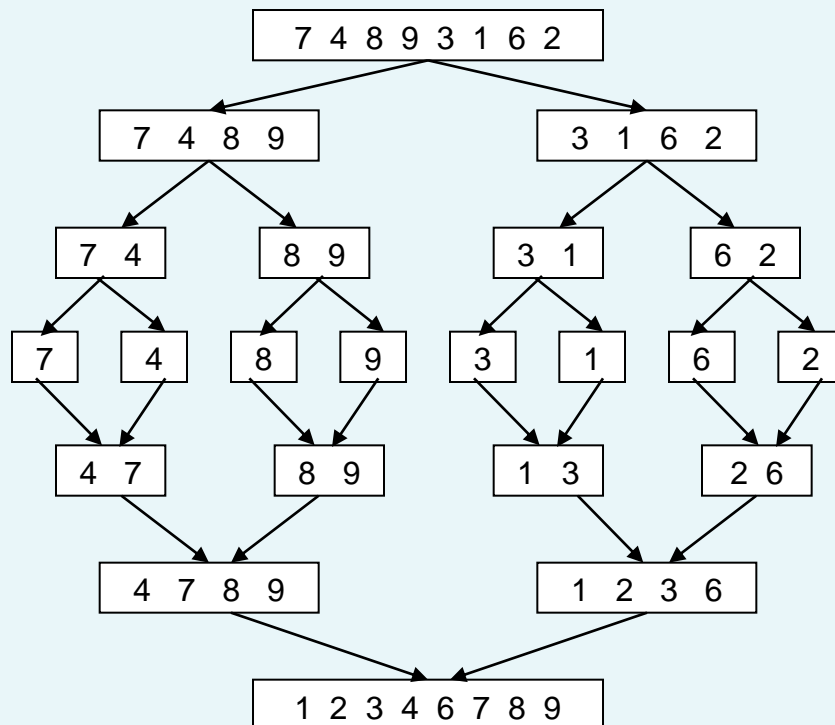
```
FUNCTION MergeSort (L:List; n:Integer):List;  
VAR  L1,L2:List;  
BEGIN  
    IF n=1 THEN RETURN(L)  
    ELSE BEGIN  
        Chia đôi L thành L1 và L2, với độ dài n/2;  
        RETURN (Merge (MergeSort (L1,n/2) ,MergeSort (L2,n/2) ) ) ;  
    END;  
END;
```

- **Hàm MergeSort** nhận danh sách có độ dài n và trả về danh sách đã được sắp xếp.
- **Thủ tục Merge** nhận 2 danh sách đã được sắp thứ tự L1 và L2, mỗi danh sách có độ dài n/2, trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự.



# Mô hình minh họa Mergesort

- Sắp xếp danh sách L gồm 8 phần tử : **7, 4, 8, 9, 3, 1, 6, 2**







## Ví dụ 2. Phương trình đệ quy của thuật toán MergeSort

- $T(n)$  là thời gian thực hiện **MergeSort** danh sách  $n$ ,  $T(n/2)$  là thời gian thực hiện **MergeSort** một danh sách  $n/2$  phần tử.
- Khi  $L$  có độ dài 1 ( $n = 1$ ): chương trình chỉ làm một việc duy nhất là  $return(L)$ , việc này tốn  $O(1) = C_1$  thời gian.
- Khi  $L$  có độ dài  $n > 1$ : chương trình gọi đệ quy **MergeSort** 2 lần cho  $L_1$  và  $L_2$  với độ dài  $n/2$ , do đó thời gian để gọi 2 lần đệ quy này là  $2T(n/2)$ .
- Ngoài ra, còn phải tốn thời gian chia danh sách  $L$  thành 2 nửa bằng nhau và trộn hai danh sách kết quả (Merge): cần thời gian  $O(n) = nC_2$ .
- Vậy ta có phương trình đệ quy như sau:

$$T(n) = \begin{cases} C_1 & \text{nếu } n=1 \\ 2T(\frac{n}{2}) + nC_2 & \text{nếu } n>1 \end{cases}$$



# Giải phương trình đệ quy

- Có 3 phương pháp giải phương trình đệ quy:

## **(1) Phương pháp truy hồi.**

- Triển khai  $T(n)$  theo  $T(n-1)$ , rồi  $T(n-2)$ , ... cho đến  $T(1)$  hoặc  $T(0)$
- Suy ra nghiệm

## **(2) Phương pháp đoán nghiệm.**

- Dự đoán nghiệm  $f(n)$
- Áp dụng định nghĩa tỷ suất tăng và chứng minh  $f(n)$  là tỷ suất tăng của  $T(n)$

## **(3) Lời giải tổng quát của một lớp phương trình đệ quy.**



# Phương pháp truy hồi

- Dùng đệ quy để thay thế  $T(m)$  với  $m < n$  (vào phía phải phương trình) cho đến khi tất cả  $T(m)$  với  $m > 1$  được thay thế bởi biểu thức của  $T(1)$  hoặc  $T(0)$ .

Triển khai  $T(n)$

$T(n-1)$

$T(n-2)$

...

cho đến  $T(1)$

theo  
rồi đến  
tiếp đến

- Vì  $T(1)$  và  $T(0)$  là hằng số nên công thức  $T(n)$  chứa các số hạng chỉ liên quan đến  $n$  và hằng số.
- Từ công thức đó suy ra nghiệm của phương trình.



# Ví dụ 1. Giải phương trình đệ quy bằng phương pháp truy hồi

**Hàm tính Giai thừa N !**

$$T(n) = \begin{cases} C_1 & \text{nếu } n=0 \\ T(n-1) + C_2 & \text{nếu } n>0 \end{cases}$$

Ta có :

$$T(n) = T(n-1) + C_2$$

$$T(n) = [T(n-2) + C_2] + C_2 = T(n-2) + 2C_2$$

$$T(n) = [T(n-3) + C_2] + 2C_2 = T(n-3) + 3C_2$$

.....

$$T(n) = T(n-i) + iC_2$$

- Quá trình truy hồi kết thúc khi  $n - i = 0$  hay  $i = n$ .
- Khi đó ta có  $T(n) = T(0) + nC_2 = C_1 + nC_2 = \mathbf{O(n)}$



# Một số tổng thông dụng

$$S = 1 + 2 + 3 + \dots + n = n(n+1)/2 \approx n^2/2$$

$$S = 1 + 2^2 + 3^2 + \dots + n^2 = n(n+1)(2n+1)/6 \approx n^3/3$$

$$S = 1 + a + a^2 + a^3 + \dots + a^n = (a^{n+1} - 1)/(a - 1)$$

Nếu  $0 < a < 1$  thì

$$S \leq 1/(1-a)$$

và khi  $n \rightarrow \infty$  thì

$S$  tiến về  $1/(1-a)$

$$S = 1 + 1/2 + 1/3 + \dots + 1/n = \ln(n) + \gamma$$

Hằng số Euler  $\gamma \approx 0.577215665$

$$S = 1 + 1/2 + 1/4 + 1/8 + \dots + 1/2^n + \dots \approx 2$$



# Một số công thức Logarit

**10. Logarit :**  $0 < N_1, N_2, N$  và  $0 < a, b \neq 1$  ta có

$$\log_a N = M \Leftrightarrow N = a^M \quad \log_a \left( \frac{N_1}{N_2} \right) = \log_a N_1 - \log_a N_2$$

$$\log_a a^M = M \quad \log_a N^\alpha = \alpha \log_a N$$

$$a^{\log_a N} = N \quad \log_{a^\alpha} N = \frac{1}{\alpha} \log_a N$$

$$N_1^{\log_a N_2} = N_2^{\log_a N_1} \quad \log_a N = \frac{\log_b N}{\log_b a}$$

$$\log_a (N_1 \cdot N_2) = \log_a N_1 + \log_a N_2 \quad \log_a b = \frac{1}{\log_b a}$$



# Lời giải tổng quát cho một lớp các phương trình đệ quy

- Trong mục này, ta sẽ nghiên cứu các phần sau:
  - Bài toán đệ quy tổng quát.
  - Thành lập phương trình đệ quy tổng quát.
  - Giải phương trình đệ quy tổng quát.
  - Các khái niệm về *nghiệm thuần nhất, nghiệm riêng* và *hàm nhân*.
  - Nghiệm của phương trình đệ quy tổng quát khi  **$d(n)$  là hàm nhân**.
  - Nghiệm của phương trình đệ quy tổng quát khi  **$d(n)$  không phải là hàm nhân**.



# Bài toán đệ quy tổng quát

- **Giải thuật Chia để trị:**

- Phân rã bài toán kích thước  $n$  thành  $a$  bài toán con có kích thước  $n/b$ .
- Giải các bài toán con và tổng hợp kết quả để được kết quả của bài toán đã cho.

- Với các bài toán con, tiếp tục áp dụng giải thuật *Chia để trị* cho đến khi các bài toán con kích thước 1. Kỹ thuật này sẽ dẫn đến một *thuật toán đệ quy*.

- **Giả thiết :**

- Bài toán con kích thước 1 lấy **một** đơn vị thời gian
- Thời gian chia bài toán và tổng hợp kết quả để được lời giải của bài toán ban đầu là  **$d(n)$** .





# Thành lập phương trình đệ quy tổng quát

- Gọi  $T(n)$  là thời gian để giải bài toán kích thước  $n$ , thì  $T(n/b)$  là thời gian để giải bài toán con kích thước  $n/b$ .
- *Khi  $n = 1$* : theo giả thiết trên thời gian giải bài toán kích thước 1 là 1 đơn vị, tức là  $T(1) = 1$ .
- *Khi  $n > 1$* : ta phải giải đệ quy  $a$  bài toán con kích thước  $n/b$ , mỗi bài toán con tốn  $T(n/b)$  nên thời gian cho  $a$  lời giải đệ quy này là  $aT(n/b)$ .
- Ngoài ra, còn phải tốn thời gian để phân chia bài toán và tổng hợp các kết quả, thời gian này theo giả thiết trên là  $d(n)$ .
- Vậy ta có phương trình đệ quy:

$$T(1) = 1$$

$$T(n) = aT(n/b) + d(n)$$



# Ví dụ MergeSort

```
FUNCTION MergeSort (L:List; n:Integer):List;  
VAR  L1,L2:List;  
BEGIN  
  IF n=1 THEN RETURN (L)  
  ELSE BEGIN  
    Chia đôi L thành L1 và L2, với độ dài n/2;  
    RETURN (Merge (MergeSort (L1,n/2) ,MergeSort (L2,n/2)) );  
  END;  
END;
```

- *Khi  $n = 1$ : tốn  $C_1$*
- *Khi  $n > 1$ :  $a = b = 2$  (Giải đệ quy 2 bài toán con kích thước  $n/2$ , mỗi bài toán con tốn  $T(n/2)$  nên cần  $2T(n/2)$ ).*
- *Thời gian phân chia bài toán và tổng hợp kết quả:  $d(n) = nC_2$ .*
- *Vậy phương trình đệ quy:*

$$T(n) = \begin{cases} C_1 & \text{nếu } n=1 \\ 2T(\frac{n}{2}) + nC_2 & \text{nếu } n>1 \end{cases}$$



# Giải phương trình đệ quy tổng quát

$$T(n) = \begin{cases} 1 & \text{neu } n = 1 \\ aT\left(\frac{n}{b}\right) + d(n) & \text{neu } n > 1 \end{cases}$$

$$T(n) = aT\left(\frac{n}{b}\right) + d(n)$$

$$T(n) = a\left[aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b}\right)\right] + d(n) = a^2T\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$\begin{aligned} T(n) &= a^2\left[aT\left(\frac{n}{b^3}\right) + d\left(\frac{n}{b^2}\right)\right] + ad\left(\frac{n}{b}\right) + d(n) \\ &= a^3T\left(\frac{n}{b^3}\right) + a^2d\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n) \end{aligned}$$

.....

$$T(n) = a^iT\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j d\left(\frac{n}{b^j}\right)$$



# Giải phương trình đệ quy tổng quát

$$T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j d\left(\frac{n}{b^j}\right)$$

Giả sử  $n = b^k$ , quá trình suy rộng trên sẽ kết thúc khi  $i = k$ .  
Khi đó ta được:

$$T\left(\frac{n}{b^i}\right) = T\left(\frac{n}{b^k}\right) = T\left(\frac{b^k}{b^k}\right) = T(1) = 1$$

Thay vào trên ta có:

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$



# Nghiệm thuần nhất và nghiệm riêng

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$

Nghiệm  
thuần nhất

Nghiệm riêng

**Nghiệm thuần nhất** là nghiệm chính xác khi  $d(n)=0$ ,  $\forall n$  : biểu diễn thời gian giải tất cả bài toán con.

**Nghiệm riêng** phụ thuộc hàm tiến triển, số lượng và kích thước bài toán con: biểu diễn thời gian tạo bài toán con và tổng hợp kết quả.



# Nghiệm thuần nhất và nghiệm riêng

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$

Nghiệm  
thuần nhất

$$a^k = n^{\log_b a}$$

Nghiệm riêng

NTN: Do  $n=b^k$  nên  $n^{\log_b a} = (b^k)^{\log_b a} = b^{\log_b a^k} = a^k$

Nghiệm của phương trình là: **MAX(NTN, NR)**.



# Hàm nhân

- Hàm  $f(n)$  là **hàm nhân** (multiplicative function) nếu:  **$f(m.n) = f(m).f(n)$**   $\forall m, n$  nguyên dương
- **Ví dụ:**
  - Hàm  **$f(n) = 1$**  là hàm nhân vì:  $f(m.n) = 1.1 = 1 = f(m).f(n)$
  - Hàm  **$f(n) = n$**  là hàm nhân vì:  $f(m.n) = m.n = f(m).f(n)$
  - Hàm  **$f(n) = n^k$**  là hàm nhân vì:  
$$f(m.n) = (m.n)^k = m^k.n^k = f(m).f(n).$$
  - Hàm  **$f(n) = \log n$**  *không phải* là hàm nhân vì:  
$$f(m.n) = \log(m.n) = \log m + \log n \neq \log m . \log n = f(m).f(n)$$



# Tính nghiệm riêng khi $d(n)$ là hàm nhân

- Khi  $d(n)$  là **hàm nhân**, ta có:

$$d(b^{k-j}) = d(b.b.b\dots b) = d(b).d(b)\dots d(b) = [d(b)]^{k-j}$$

$$NR = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} a^j [d(b)]^{k-j} = [d(b)]^k \sum_{j=0}^{k-1} \left[ \frac{a}{d(b)} \right]^j = [d(b)]^k \frac{\left[ \frac{a}{d(b)} \right]^k - 1}{\frac{a}{d(b)} - 1}$$

$$\text{Hay } NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$





$$NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

## Ba trường hợp

- **Trường hợp 1:  $a > d(b)$**

Trong công thức trên:  $a^k > [d(b)]^k$

Theo quy tắc tính độ phức tạp:

$$NR \text{ là } O(a^k) = O(n^{\log_b a}) = \text{NTN}.$$

Do đó  **$T(n) = O(n^{\log_b a})$** .

Nhận xét:  $T(n)$  chỉ phụ thuộc vào  $a, b$  mà không phụ thuộc hàm tiến triển  $d(n) \rightarrow$  Cải tiến thuật toán = Giảm  $a$  : giảm số bài toán con.



$$NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

## Ba trường hợp

- **Trường hợp 2:  $a < d(b)$**

Trong công thức trên:  $[d(b)]^k > a^k$

Theo quy tắc tính độ phức tạp :

NR là  $O([d(b)]^k) = O(n^{\log_b d(b)}) > NTN$ .

Do đó  **$T(n) = O(n^{\log_b d(b)})$** .

Nhận xét :  $T(n)$  phụ thuộc vào  $b$  và hàm tiến triển  $d(b) \rightarrow$

Cải tiến thuật toán = Giảm  $d(b)$  : cải tiến việc phân chia bài toán và tổng hợp kết quả.



# Ba trường hợp

$$NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

## Trường hợp 3: **a = d(b)**

Công thức trên không xác định nên phải tính trực tiếp nghiệm riêng:

$$NR = [d(b)]^k \sum_{j=0}^{k-1} \left[ \frac{a}{d(b)} \right]^j = a^k \sum_{j=0}^{k-1} 1 = a^k k \quad (\text{do } a = d(b))$$

Do  $n = b^k$  nên  $k = \log_b n$  và  $a^k = n^{\log_b a}$ .

Vậy NR là  $n^{\log_b a} \log_b n > \text{NTN}$ .

Do đó  **$T(n) = O(n^{\log_b a} \log_b n)$**



# Tính nghiệm riêng khi $d(n)$ không phải là hàm nhân

- Trong trường hợp hàm tiến triển  $d(n)$  không phải là hàm nhân thì không thể áp dụng các công thức ứng với ba trường hợp nói trên mà chúng ta phải **tính trực tiếp**  $NR$ , sau đó so sánh với  $NTN$  và lấy  **$MAX(NR, NTN)$** .



# Quy tắc chung để giải phương trình đệ quy

- **Lưu ý khi giải một phương trình đệ quy cụ thể:**
  - (1) Xem phương trình có thuộc **dạng phương trình tổng quát** không ?
  - (2) Nếu có, xem hàm tiến triển  $d(n)$  có **dạng hàm nhân** không?
    - a) Nếu **có**: xác định  $a, d(b)$ ; so sánh  $a, d(b)$  để vận dụng một trong ba trường hợp trên.
    - b) Nếu **không** : tính trực tiếp nghiệm để so sánh.
  - (3) Suy ra nghiệm của phương trình  $= \text{Max}(\text{NTN}, \text{NR})$



## Ví dụ 1. GPT với $T(1) = 1$ và

$$1/ \quad T(n) = 4T\left(\frac{n}{2}\right) + n$$

- Phương trình đã cho có dạng phương trình tổng quát.
- $d(n)=n$  là hàm nhân.
- $a = 4$  và  $b = 2$ .
- $d(b) = b = 2 < a$  (Trường hợp 1)  
 $\rightarrow T(n) = O(n^{\log_b a}) = O(n^{\log^4}) = O(n^2).$



## Ví dụ 2. GPT với $T(1) = 1$ và

$$2/ \quad T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

- Phương trình đã cho có dạng phương trình tổng quát.
  - $d(n)=n^2$  là hàm nhân.
  - $a = 4$  và  $b = 2$ .
  - $d(b) = b^2 = 4 = a$  (Trường hợp 3)
- $T(n) = O(n^{\log_b a} \log_b n) = O(n^{\log^4} \log n) = (n^2 \log n).$



## Ví dụ 3. GPT với $T(1) = 1$ và

$$3/ \quad T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

- Phương trình đã cho có dạng phương trình tổng quát.
- $d(n)=n^3$  là hàm nhân.
- $a = 4$  và  $b = 2$ .
- $d(b) = b^3 = 8 > a$  (Trường hợp 2)  
 $\rightarrow T(n) = O(n^{\log_b d(b)}) = O(n^{\log 8}) = O(n^3).$





## Ví dụ 4. GPT với $T(1) = 1$ và

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

- PT thuộc dạng phương trình tổng quát nhưng  $d(n) = n \log n$  *không phải là một hàm nhân*.
- $a = 2$  và  $b = 2$
- $NTN = n^{\log_b a} = n^{\log 2} = n$
- Do  $d(n) = n \log n$  không phải là hàm nhân nên ta phải tính nghiệm riêng bằng cách xét trực tiếp



## Ví dụ 4. (tt)

$$\begin{aligned}NR &= \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j 2^{k-j} \log 2^{k-j} \\&= 2^k \sum_{j=0}^{k-1} (k-j) = 2^k \frac{k(k+1)}{2} = O(2^k k^2)\end{aligned}$$

- Theo cách giải phương trình đệ quy tổng quát thì  $n = b^k$  nên  $k = \log_b n$ , ở đây do  $b = 2$  nên  $2^k = n$  và  $k = \log n$ ,
- $NR = O(2^{\log n} \log^2 n) = O(n \log^2 n) > NTN = n$   
 $\rightarrow T(n) = O(n \log^2 n).$



## Bài tập 4-1. GPT với $T(1) = 1$ và

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- Phương trình đã cho có dạng phương trình tổng quát.
- $d(n)=1$  là hàm nhân.
- $a = 1$  và  $b = 2$ .
- $d(b) = 1 = a$ .

$$\rightarrow T(n) = O(n^{\log_b a} \log_b n) = O(n^{\log^1 1} \log n) = O(\log n).$$



## Bài tập 4-2. GPT với $T(1) = 1$ và

$$T(n) = 2T\left(\frac{n}{2}\right) + \log n$$

- Phương trình đã cho có dạng phương trình tổng quát.
- $d(n) = \log n$  **không phải** là hàm nhân.
- $a = 2$  và  $b = 2$
- $NTN = O(n^{\log_b a}) = O(n^{\log 2}) = O(n)$ .
- Tính trực tiếp nghiệm riêng.



## Bài tập 4-2. (tt)

$$NR = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j \log 2^{k-j}$$

$$NR = \sum_{j=0}^{k-1} 2^j (k - j) = \sum_{j=0}^{k-1} k 2^j - \sum_{j=0}^{k-1} j 2^j$$

$$NR = O(k \sum_{j=0}^{k-1} 2^j) = O(k \frac{2^k - 1}{2 - 1})$$

$$NR = O(k 2^k) = O(n \log n) > n = NTN$$

$$T(n) = O(n \log n)$$



## Bài tập 8.

$$C_n^k = \begin{cases} 1 & \text{neu } k = 0 \text{ hoac } k = n \\ C_{n-1}^{k-1} + C_{n-1}^k \end{cases}$$

- Gọi  $T(n)$  là thời gian để tính  $C_n^k$
  - Thì thời gian để tính  $C_{n-1}^k$  là  $T(n-1)$
  - Khi  $n=1$  thì  $k=0$  hoặc  $k=1 \Rightarrow$  chương trình trả về giá trị 1, tốn  $O(1) = C_1$
  - Khi  $n>1$ , trong trường hợp xấu nhất, chương trình phải làm các việc:
    - Tính  $C_{n-1}^k$  và  $C_{n-1}^{k-1}$  : tốn  $2T(n-1)$ .
    - Thực hiện phép cộng và trả kết quả: tốn  $C_2$
- a) Ta có phương trình:  $T(1)=C_1$  và  $T(n)=2T(n-1) + C_2$



## Bài tập 8. (tt)

b) Giải phương trình  $T(n) = 2T(n-1) + C_2$

$$\bullet T(n) = 2[2T(n-2) + C_2] + C_2$$

$$= 4T(n-2) + 3C_2$$

$$= 4[2T(n-3) + C_2] + 3C_2$$

$$= 8T(n-3) + 7C_2$$

.....

$$= 2^i T(n-i) + (2^i - 1) C_2$$

$$\bullet T(n) = 2^{n-1} C_1 + (2^{n-1} - 1) C_2$$

$$= (C_1 + C_2) 2^{n-1} - C_2 = O(2^n)$$