TRƯỜNG CAO ĐẲNG CNTT HỮU NGHỊ ViỆT - HÀN

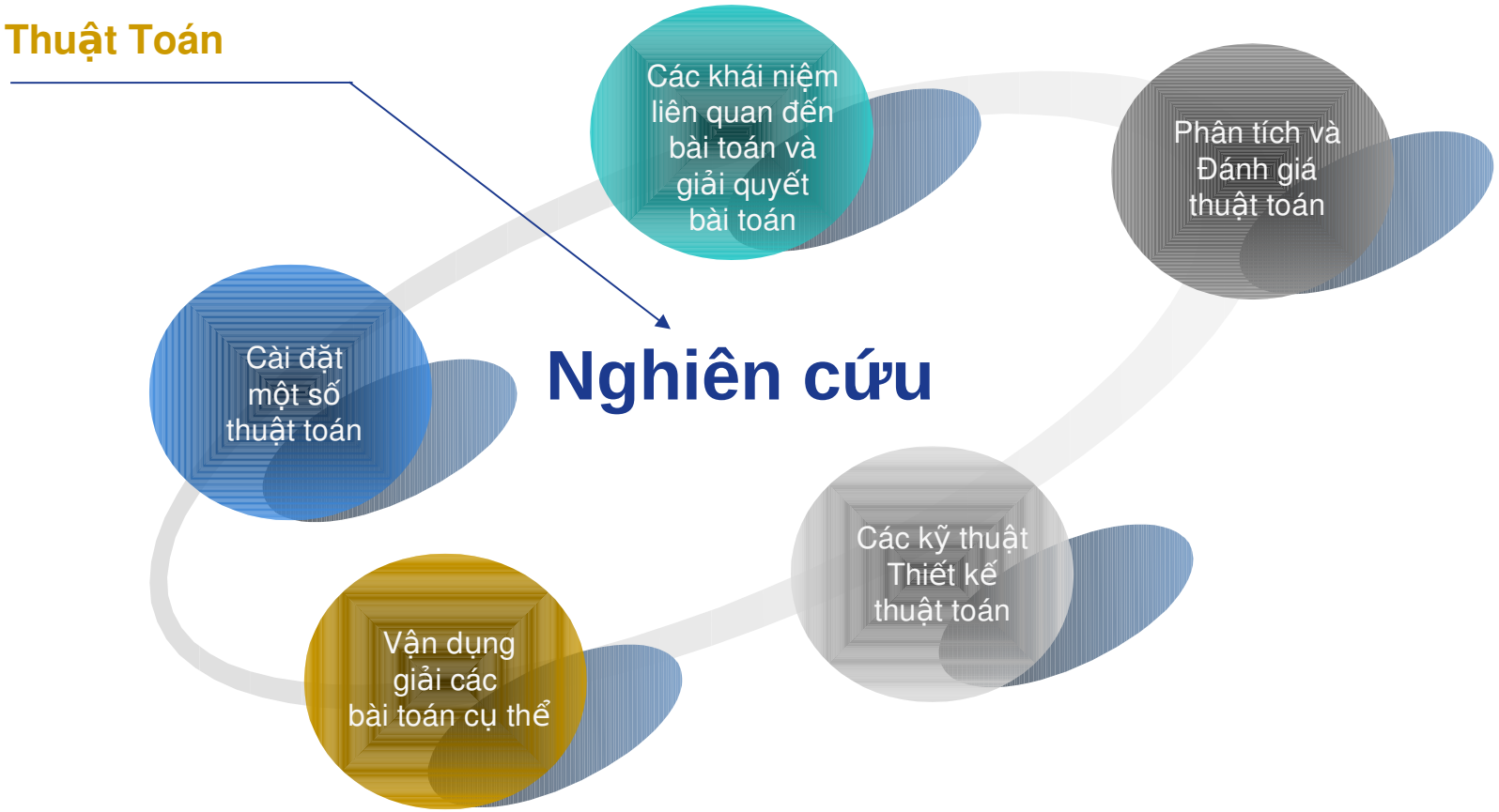
KHOA KHOA HỌC MÁY TÍNH

----------***----------

THUẬT TOÁN

(Algorithms)

Mục đích


Thuật Toán
Các khái niệm liên quan đến bài toán và giải quyết bài toán
Phân tích và Đánh giá thuật toán
Cài đặt một số thuật toán
Nghiên cứu
Các kỹ thuật Thiết kế thuật toán
Vận dụng giải các bài toán cụ thể

Nội Dung

C1 THUẬT TOÁN VÀ ĐỘ PHỨC TẠP

C2 CHIA ĐỂ TRỊ

C3 QUY HOẠCH ĐỘNG

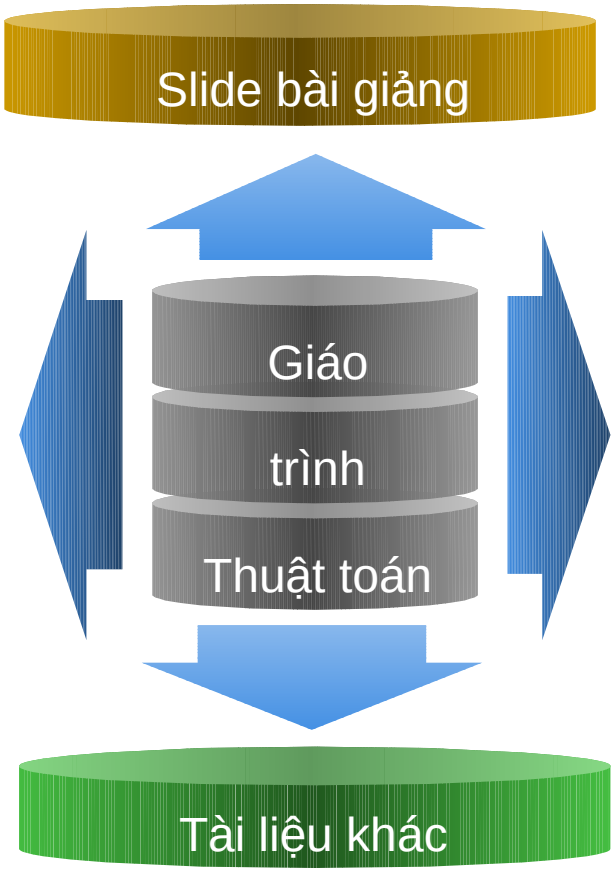
C4 THUẬT TOÁN THAM LAM

C5 THUẬT TOÁN QUAY LUI

Học liệu

Tài liệu trong nước

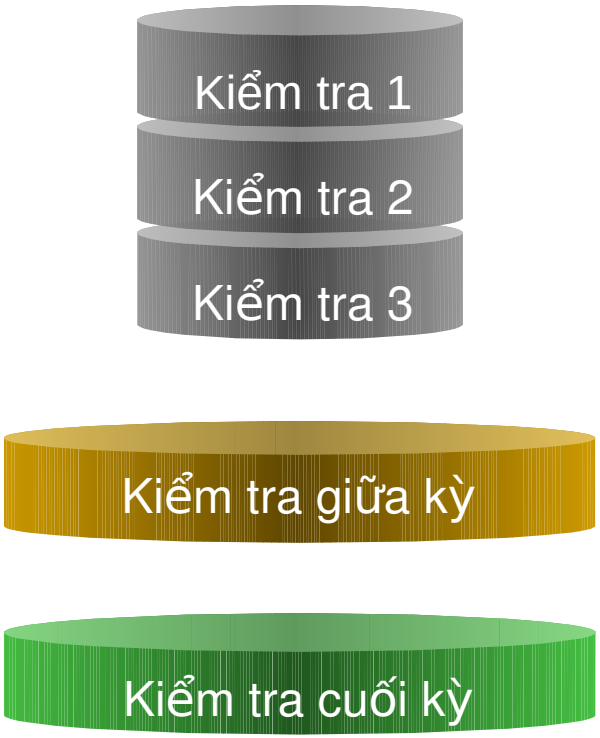
- Vũ Đình Hòa, Đỗ Trung Kiên, Thuật toán và độ phức tạp thuật toán, nhà xuất bản đại học sư phạm 2007
- Đỗ Xuân Lôi, Cấu trúc dữ liệu và giải thuật, NXB Đại học Quốc Gia Hà Nội 2007



Tài liệu nước ngoài

- Richard Neapolitan, Kumarss Naimipour, Foundations of Algorithms Using C++ Pseudocode, Jones and Bartlett Publishers
- Introduction to Algorithms, Second Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein the MIT press

Đánh giá


Kiểm tra 1
Kiểm tra 2
Kiểm tra 3
Kiểm tra giữa kỳ
Kiểm tra cuối kỳ

THUẬT TOÁN VÀ ĐỘ PHỨC TẠP

1.1 Khái niệm thuật toán

1.2 Thiết kế - Phân tích – Đánh giá thuật toán

1.3 Biểu diễn thuật toán

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

1.5 Đánh giá độ phức tạp thuật toán

1.1 Khái niệm thuật toán

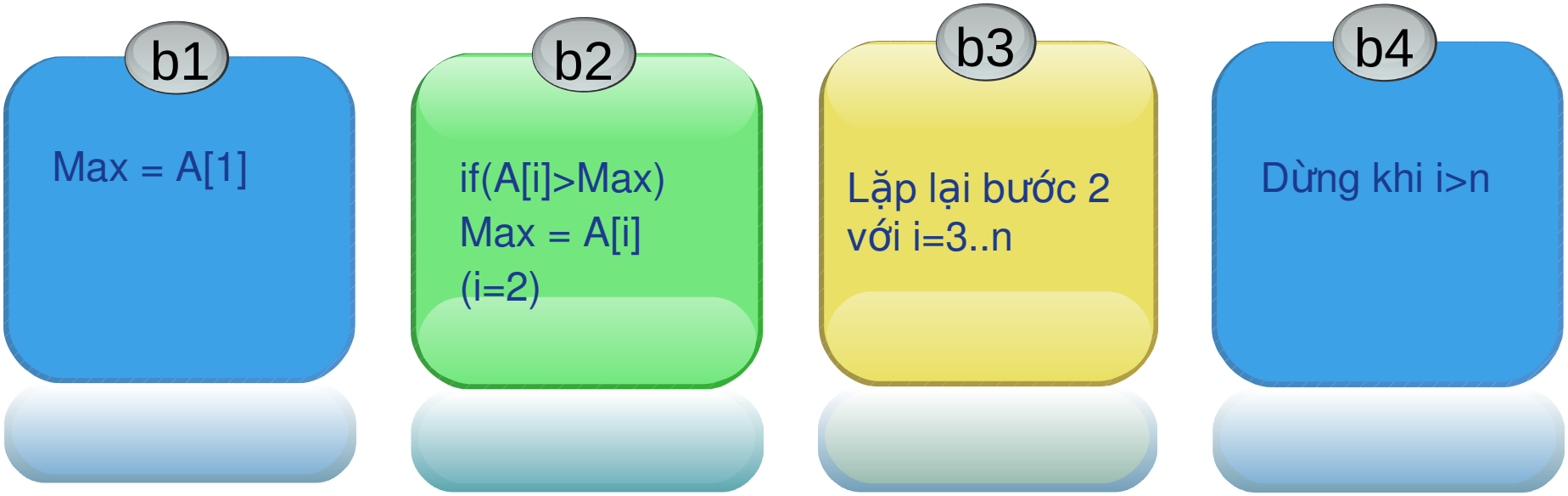
Một ví dụ về thuật toán (1)

- Cho A={a_1, a_2,…,a_n| a_i Z với i N}
- Hãy mô tả các bước để tìm được phần tử lớn nhất trong dãy?

1.1 Khái niệm thuật toán

Một ví dụ về thuật toán (2)

❖ **Ý tưởng**:



Một ví dụ về thuật toán (3)

❖ Nhận xét:

- Sau khi thực hiện trình tự các bước trên, ta sẽ nhận được đáp số của bài toán (đó là phần tử Max của dãy).
- dãy hữu hạn các bước dẫn tới đáp số mong muốn của bài toán được gọi là một thuật toán.

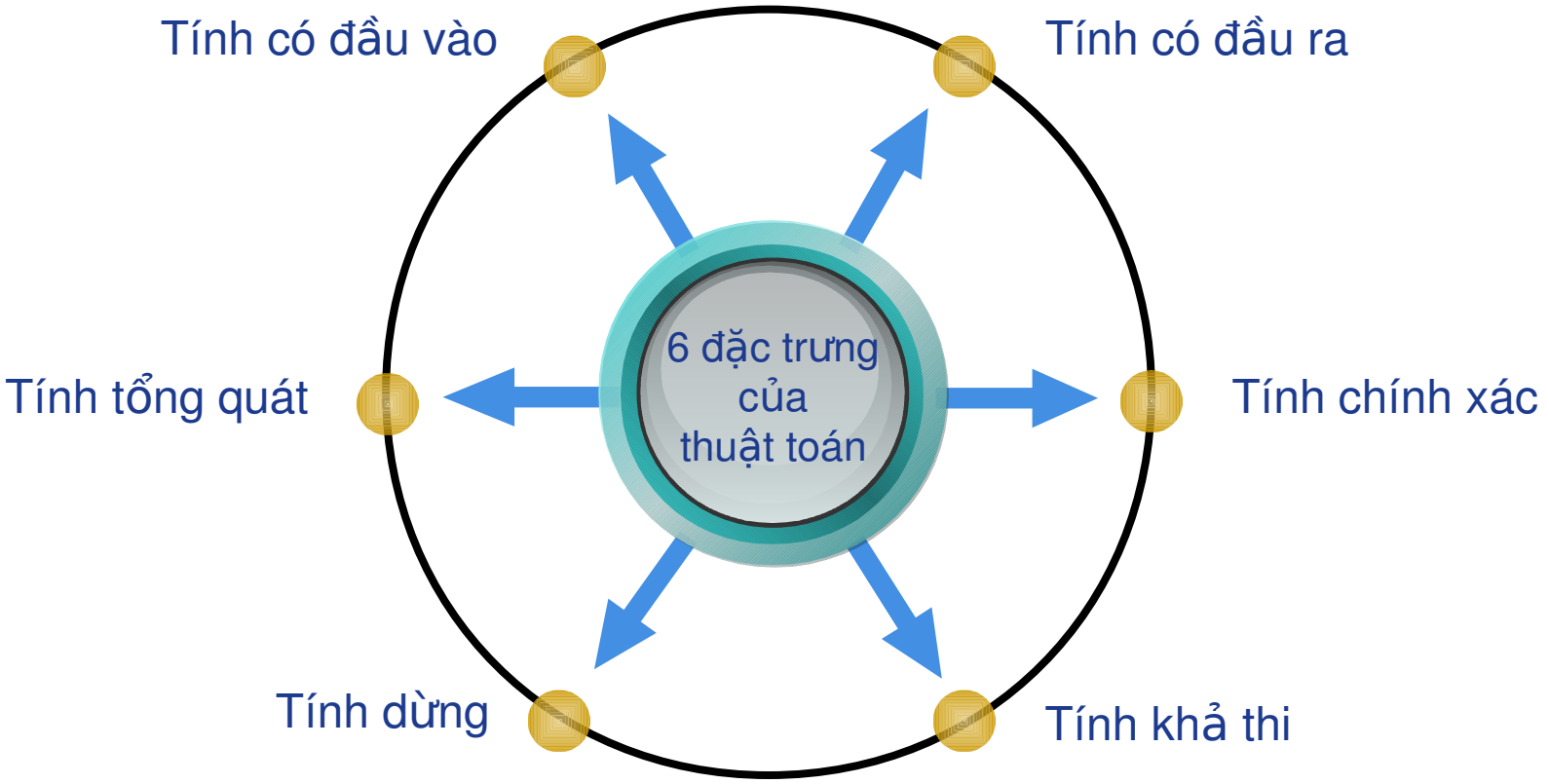
1.1 Khái niệm thuật toán

Khái niệm thuật toán

- Thuật toán (Algorithm) là một dãy hữu hạn các bước, mỗi bước mô tả chính xác các phép toán, hoặc hành động cần thực hiện;
  sau khi thực hiện các bước theo một trình tự xác định, ta được lời giải của bài toán.

1.1 Khái niệm thuật toán

Các đặc trưng của thuật toán



THUẬT TOÁN VÀ ĐỘ PHỨC TẠP

1.1 Khái niệm thuật toán

1.2 Thiết kế - Phân tích – Đánh giá thuật toán

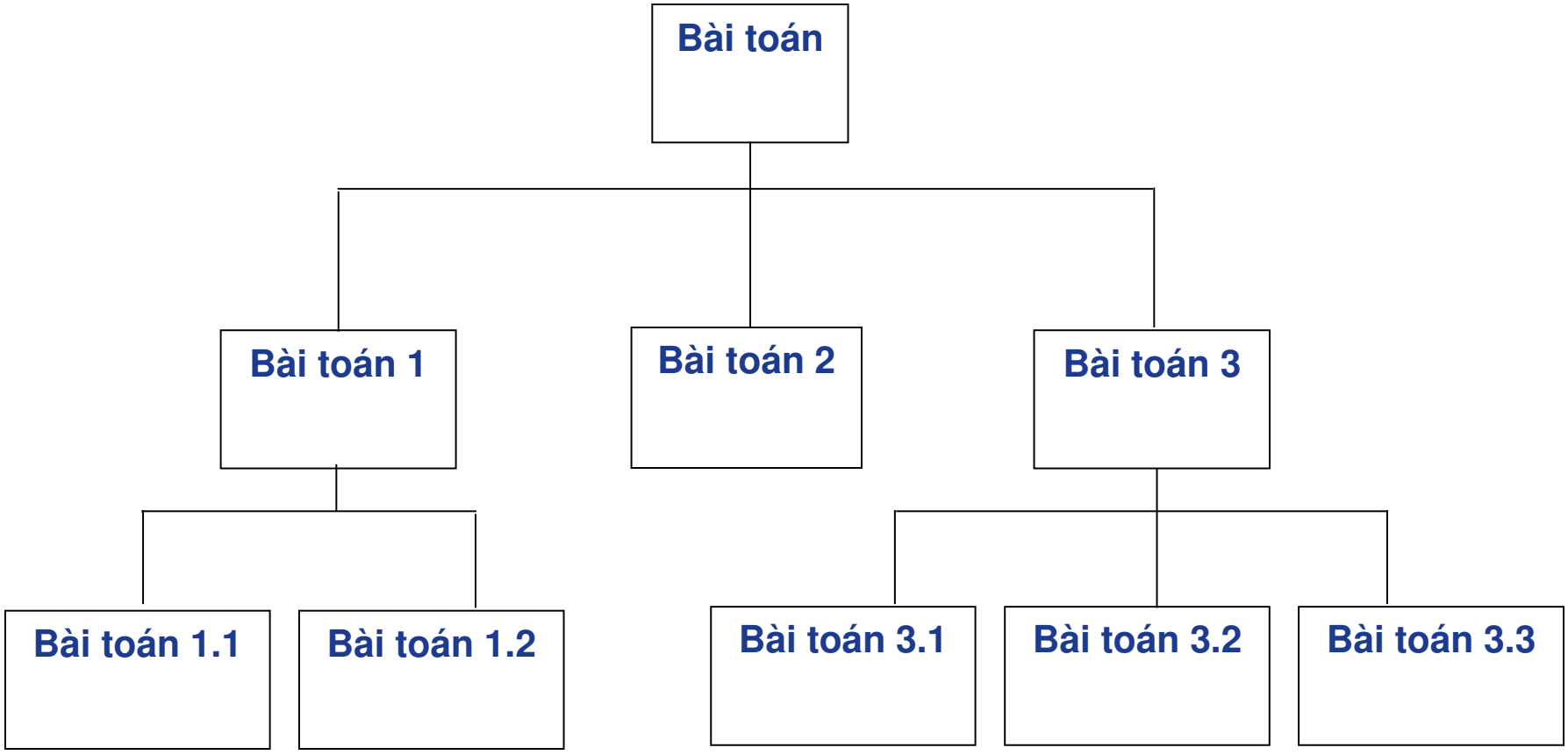
1.3 Biểu diễn thuật toán

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

1.5 Đánh giá độ phức tạp thuật toán

Thiết kế thuật toán (1)

❖ **Mô đun hóa bài toán**:

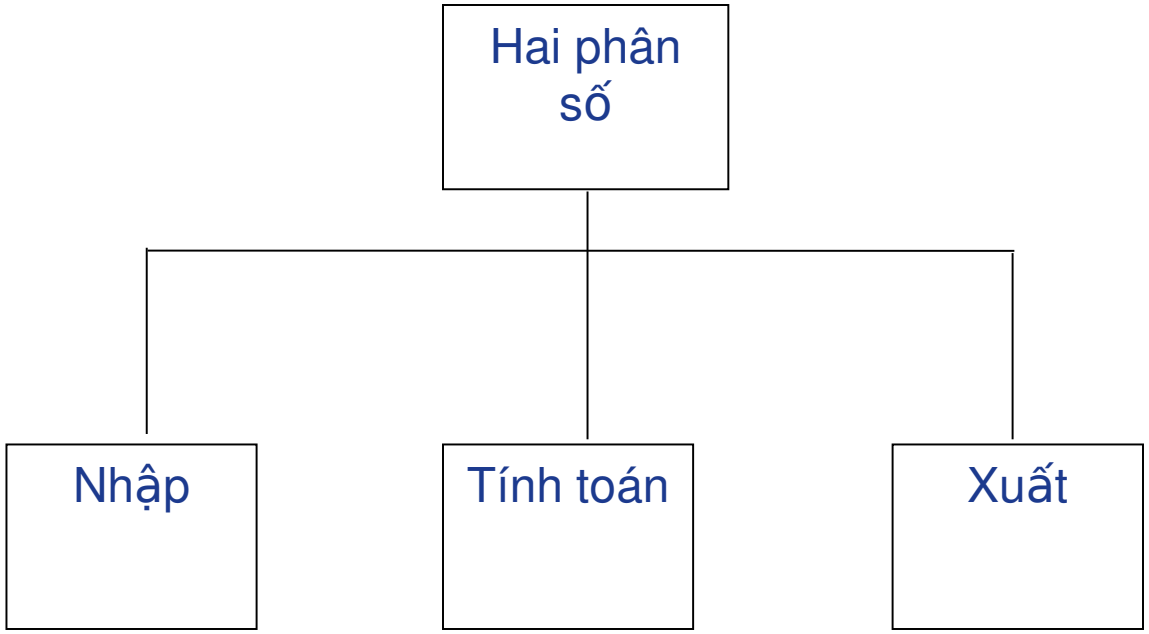


Thiết kế thuật toán (2)

- **Mô đun hóa bài toán**:
  - **Thí dụ**: Viết chương trình thực hiện các phép toán trên hai phân số.

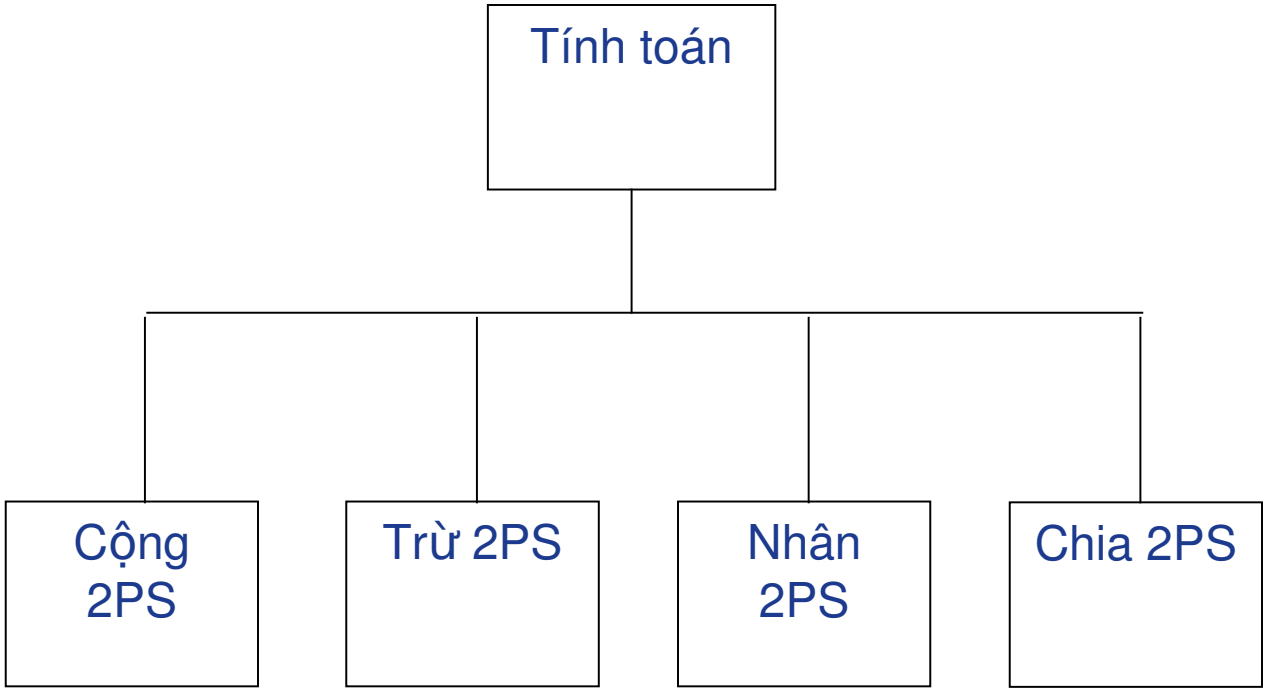
Thiết kế thuật toán (3)

- **Mô đun hóa bài toán**:
  - **Thí dụ**:



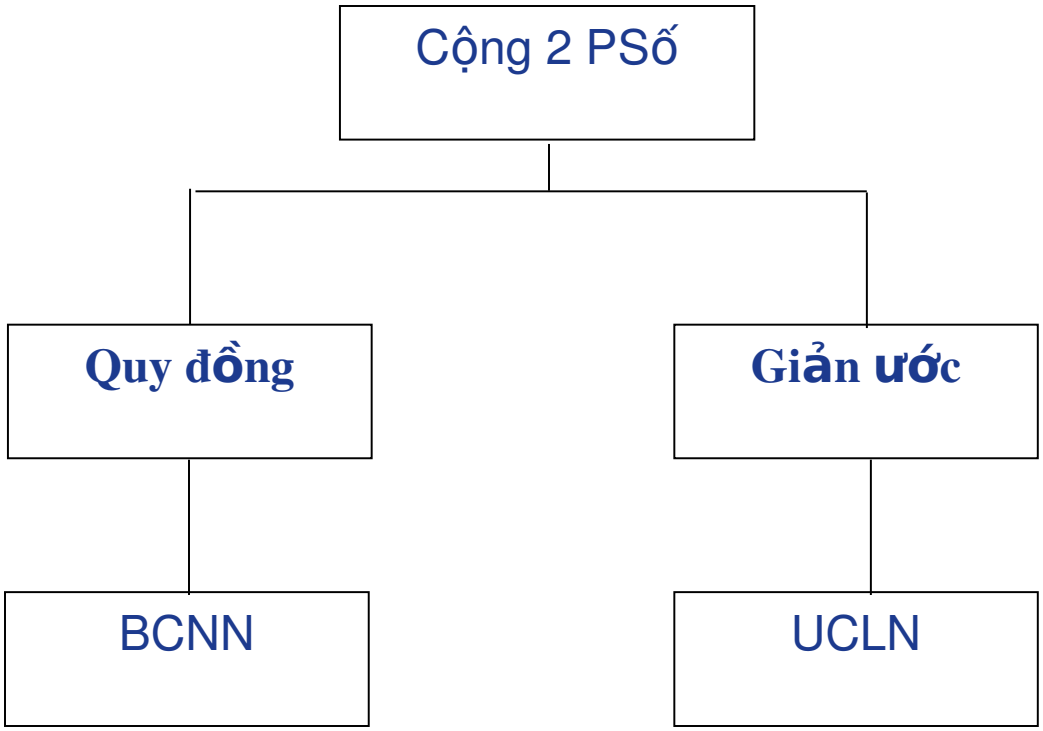
Thiết kế thuật toán (4)

- **Mô đun hóa bài toán**:
  - **Thí dụ**:



Thiết kế thuật toán (5)

- **Mô đun hóa bài toán**:
  - **Thí dụ**:



1.2 Thiết kế - Phân tích – Đánh giá thuật toán

Ưu điểm
cho phép tách bài toán ra thành các phần độc lập. Việc tìm hiểu cũng như sửa chữa chỉnh lý sẽ dễ dàng hơn.

Hạn chế
Việc phân bài toán thành các bài toán con, là một việc làm không dễ dàng. Thiết kế thuật toán mất nhiều thời gian và công sức.

Thiết kế thuật toán (6)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - Tinh chỉnh từng bước là phương pháp thiết kế thuật toán gắn liền với lập trình.
  - Nó phản ánh tinh thần của quá trình mô-đun hóa bài toán và thiết kế kiểu top-down.

Thiết kế thuật toán (7)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - **Thí dụ**: Viết chương trình sắp xếp một dãy n số nguyên khác nhau theo thứ tự tăng dần.

Thiết kế thuật toán (8)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - Ta có thể phát thảo thuật toán như sau:
  - Chọn số nhỏ nhất, đặt nó vào đầu dãy.
    - phần tử đầu tiên đã cố định vị trí,
    - dãy còn lại (n-1 phần tử) chưa có thứ tự. Ta gọi dãy chưa có thứ tự là dãy nguồn, dãy đã có thứ tự là dãy đích.
  - Tiếp tục tìm phần tử nhỏ nhất trong dãy nguồn và đặt nó vào cuối của dãy đích. Lặp lại quá trình đó cho đến khi dãy nguồn cạn. Ta được dãy có thứ tự.

Thiết kế thuật toán (9)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - Đến đây ta có phát họa thuật toán như sau:

```
{
        Duyệt i từ 1 đến n
        {
        Xét từ a_i tới a_n để tìm số nhỏ nhất a_j
        Đổi chỗ giữa a_i và a_j
        }
}
```

Thiết kế thuật toán (10)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - Phát họa trên chỉ thể hiện những ý cơ bản.
  - Ta thấy có hai nhiệm vụ con cần làm rõ thêm:
    - Tìm số nguyên nhỏ nhất a_j trong các số từ a_i đến a_n
    - Đổi chỗ a_i với a_j

Thiết kế thuật toán (11)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - Tinh chỉnh thứ hai của ý 1 như sau:

```
j = i ;
for k = j +1 to n do
If  a_k < a_j then j = k;
```

Thiết kế thuật toán (12)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - Tinh chỉnh thứ hai của ý 2 như sau:

$tg = a_i;$

$a_i = a_j;$

$a_j = tg;$

Thiết kế thuật toán (13)

- **Tinh chỉnh từng bước (Stepwise refinement)**:
  - Đến đây ta có hàm sắp xếp của bài toán trên như sau:

```
void  sort(int a[]; int n)       //a: là dãy số nguyên, n: là số phần tử của dãy
{ int   i, j, k, tg;
        for (i = 0; i< n; i++)
       {
        j = i;                                  // chọn số nhỏ nhất
        for (k = j+1; k< n; k++)
        if (a[k] <a[j])  j = k;
        {
                                tg = a[i];                      //đổi chỗ
                                a[i] = a[j];
                                a[j] = tg;
        }
      }
}
```

Phân tích thuật toán

❖ Tại sao cần phải phân tích thuật toán ?

- Việc lựa chọn một thuật toán đưa tới kết quả nhanh là một đòi hỏi thực tế
- Thời gian để thực hiện một thuật toán phụ thuộc:
  - tốc độ xử lý của máy tính,
  - ngôn ngữ lập trình,
  - chương trình dịch,
  - kích thước dữ liệu đầu vào của bài toán,…

Phân tích thuật toán

❖ Tại sao cần phải phân tích thuật toán ?

- Gọi n là kích thước dữ liệu đầu vào của bài toán,
- T(n) là hàm xác định thời gian thực hiện thuật toán.
- Giả sử ta có:
  - $T_1(n) = c.n^2$ là hàm chỉ thời gian để thực hiện thuật toán 1,
  - $T_2(n) = k.n$ là hàm chỉ thời gian để thực hiện thuật toán 2 (với c và k là các hằng số tùy ý).
- Khi đó ta thấy với n đủ lớn thì thuật toán 2 là tốt hơn so với thuật toán 1.

Đánh giá thuật toán

- Vì sao phải đánh giá thuật toán? để đánh giá một thuật toán chúng ta dựa vào những tiêu chí nào?
  - Khi giải một bài toán, cần chọn một thuật toán "tốt" nhất .
  - Lựa chọn thuật toán dựa trên cơ sở nào?
  - Thông thường ta dựa trên hai tiêu chuẩn sau đây:
    - 1.Thuật toán đơn giản, dễ hiểu, dễ cài đặt (dễ viết chương trình)
    - 2.Thuật toán sử dụng tiết kiệm nhất các nguồn tài nguyên của máy tính, và đặc biệt chạy nhanh nhất có thể được.

Một thuật toán được xem là hiệu quả nếu thuật toán đó có thời gian chạy ít hơn so với các thuật toán khác.

THUẬT TOÁN VÀ ĐỘ PHỨC TẠP

1.1 Khái niệm thuật toán

1.2 Thiết kế - Phân tích – Đánh giá thuật toán

1.3 Biểu diễn thuật toán

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

1.5 Đánh giá độ phức tạp thuật toán

1.3 Biểu diễn thuật toán

Phương pháp liệt kê từng bước (1)

- Ngôn ngữ liệt kê từng bước có nội dung như sau:
  - *Thuật toán*: Tên thuật toán và chức năng.
  - *Vào (Input)*: Dữ liệu vào với tên kiểu.
  - *Ra (Output)*: Các dữ liệu ra với tên kiểu.
  - *Biến phụ* (nếu có) với tên kiểu.
  - *Hành động:* là các thao tác với các lệnh.

1.3 Biểu diễn thuật toán

Phương pháp liệt kê từng bước (2)

❖ **Thí dụ: Giải phương trình bậc hai $ax^2 + bx + c = 0$:**

- *Bước 1: Xác định các hệ số a, b, c;*
- *Bước 2 :Nếu a = 0 quay lại thực hiện bước 1, ngược lại đến bước 3;*
- *Bước 3: Tính biểu thức = $b^2 - 4ac$;*
- *Bước 4: Nếu < 0 thông báo phương trình vô nghiệm và chuyển sang bước 8;*
- *Bước 5: Nếu = 0, tính và chuyển sang bước 7;*

$$x_1 \quad x_2 \quad \frac{b}{2a}$$

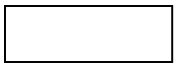
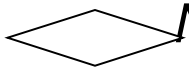
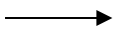
- *Bước 6: Tính ; và chuyển sang bước 7;*

$$x_1 \quad \frac{b}{2a} \qquad x_2 \quad \frac{b}{2a}$$

- *Bước 7: Thông báo các nghiệm x_1, x_2, đến bước 8;*
- *Bước 8: Kết thúc thuật toán.*

1.3 Biểu diễn thuật toán

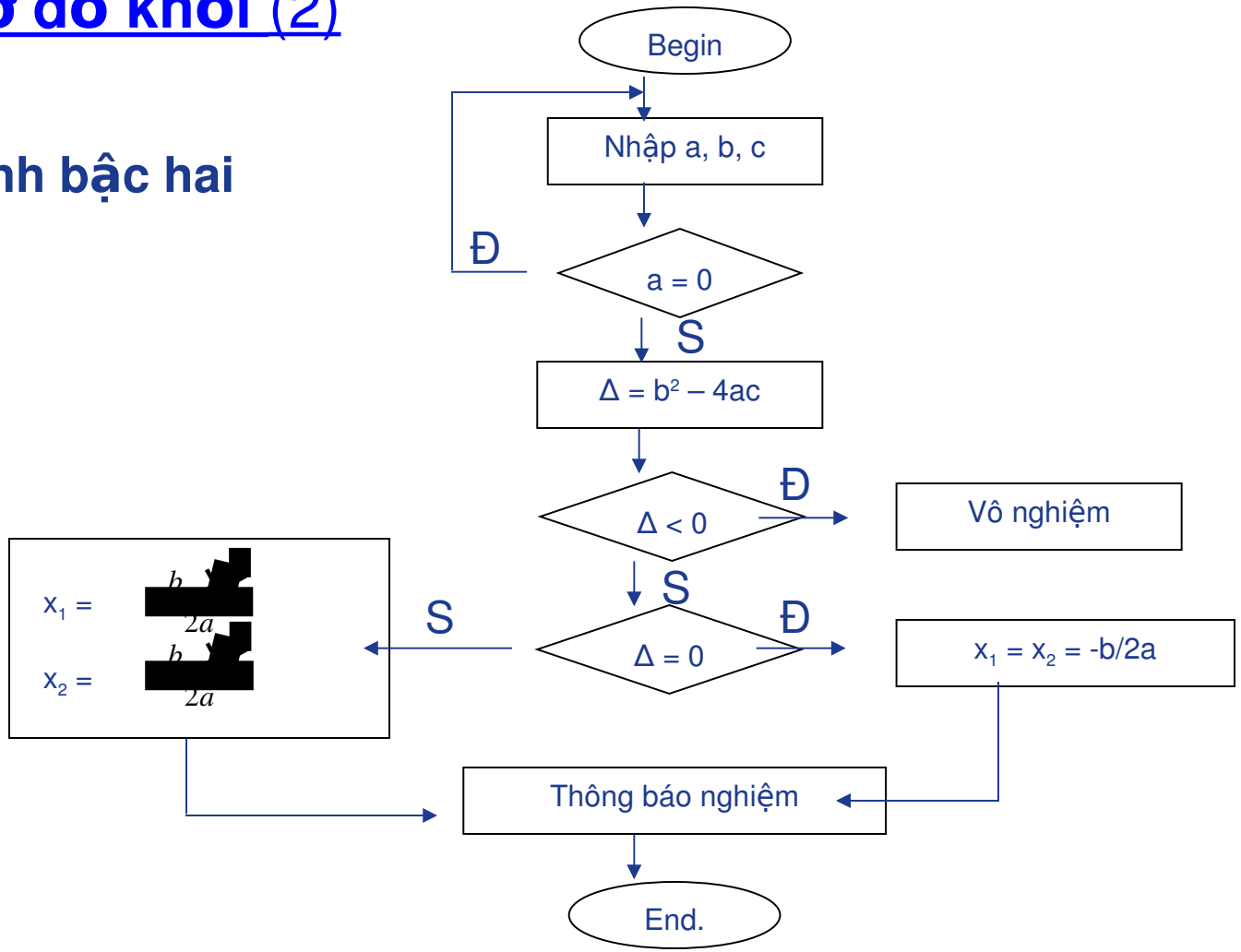
Phương pháp sơ đồ khối (1)

- **Để mô tả thuật toán bằng sơ đồ khối ta cần dựa vào các nút sau đây:**
  -  *Nút thao tác*
  -  *Nút rẽ nhánh*
  -  *Nút khởi đầu, kết thúc*
  -  *Cung*

1.3 Biểu diễn thuật toán

Phương pháp sơ đồ khối (2)

- **Thí dụ:**
  Giải phương trình bậc hai
  $ax^2 + bx + c = 0$.



THUẬT TOÁN VÀ ĐỘ PHỨC TẠP

1.1 Khái niệm thuật toán

1.2 Thiết kế - Phân tích – Đánh giá thuật toán

1.3 Biểu diễn thuật toán

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

1.5 Đánh giá độ phức tạp thuật toán

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Ký tự và biểu thức

- **Giống như trong các ngôn ngữ chuẩn, gồm:**
  - *26 chữ cái la tinh in hoa và in thường*
  - *10 chữ số thập phân*
  - *Các phép toán số học +,-,*,/*
  - *Các phép toán quan hệ <, <=, >, >=*
  - *Các giá trị lôgic True, False*
  - *Các phép toán lôgic: And, Or, Not*
  - *Tên biến: dãy chữ cái và chữ số, bắt đầu bằng chữ cái.*
  - *Biến chỉ số có dạng: A[i], B[i][j], …*
  - *Biểu thức là sự kết hợp giữa hằng, biến và các phép toán.*

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (1)

❖ Câu lệnh gán

- *Có dạng V = E*
- *Với V chỉ tên biến, tên hàm*
- *E chỉ biểu thức*
- *Ví dụ: Variable = exp; A = B = 0.1; Max = a;*
- *x = số lớn nhất trong các số a,b,c…*

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (2)

❖ Câu lệnh ghép

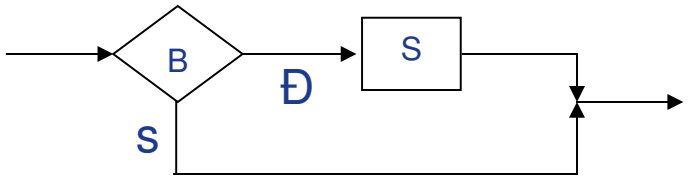
- *Có dạng { S_1; S_2;…; S_n; }*
- *Với S_i (i = 1, 2,…,n) là các lệnh.*
- *Nó cho phép ghép nhiều câu lệnh để được một câu lệnh.*
- *Ví dụ.*

```
{
    Câu lệnh 1;
    Câu lệnh 2;
    ..............
    Câu lệnh n;
}
```

Một số câu lệnh chính (3)

❖ Câu lệnh rẽ nhánh

- *Dạng 1: If (B) S;*
- *Với B là biểu thức lôgic, S là các lệnh (lệnh đơn, lệnh ghép hay lệnh rỗng).*
- *Ý nghĩa: nếu điều kiện B đúng thì lệnh S được thực hiện.*
- *Có thể biểu diễn bởi sơ đồ:*

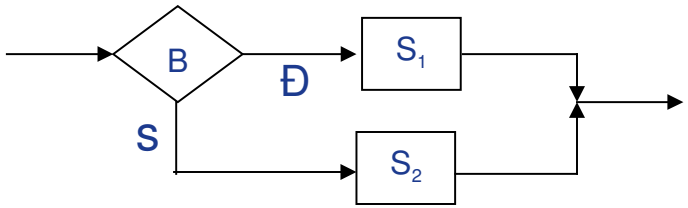


1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (4)

❖ Câu lệnh rẽ nhánh

- *Dạng 2: If (B) S_1;*
- *else S_2;*
- *B: là biểu thức lôgic, S_1, S_2: là các lệnh*
- *Ý nghĩa: nếu điều kiện B đúng thì lệnh S_1 được thực hiện, ngược lại thì lệnh S_2 được thực hiện.*
- *Sơ đồ:*



1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (5)

❖ Câu lệnh tuyển

switch (B) {

case B_1: S_1 ;

case B_2 : S_2 ;

….

case B_n : S_n ;

[default: S_{n+1} ;]

}

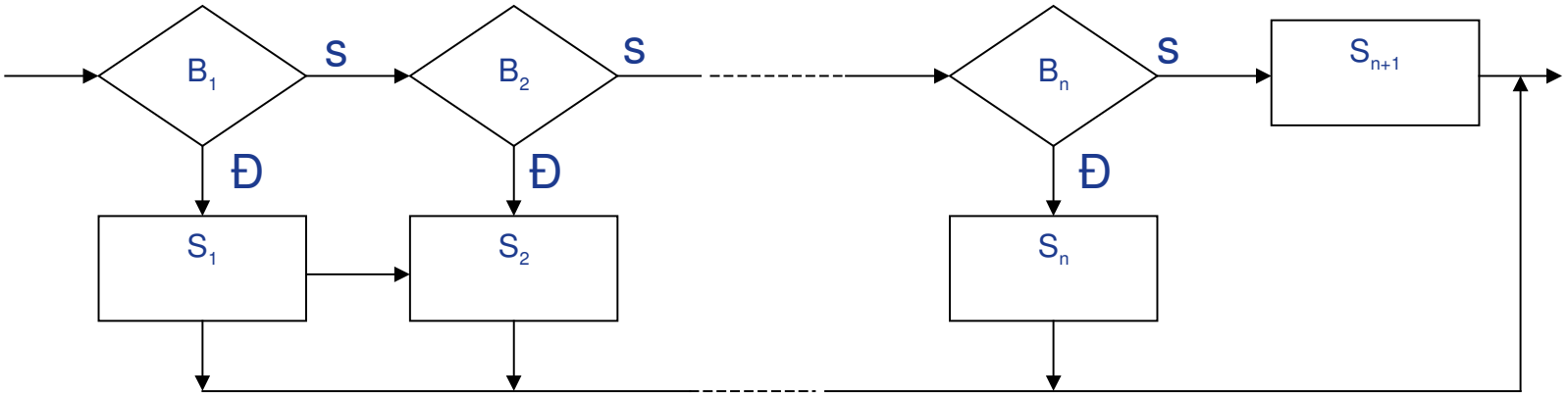
- *Với B_i (i = 1, 2, …, n) là các hằng*
- *S_i (i = 1, 2, …, n) là các lệnh.*
- *B là biểu thức lôgic.*

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (6)

❖ Câu lệnh tuyển

- *Có thể diễn tả bởi sơ đồ:*



1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (7)

❖ Câu lệnh lặp

- ***Lặp với số lần lặp biết trước:***

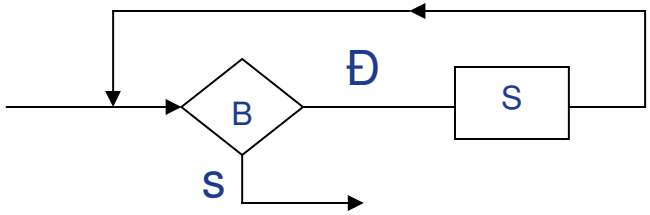
for (i = m ; i<= n; i++) S;

Khi thực hiện lệnh S, i lấy giá trị từ m đến n (m <= n), với bước nhảy tăng 1;

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (8)

- **Câu lệnh lặp**
  - ***Lặp với số lần lặp không biết trước:***
  - *Vòng while: While (B) S;*

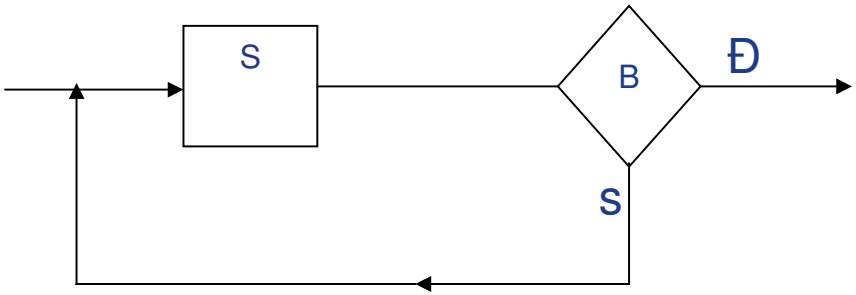


Ý nghĩa: Trong khi điều kiện B còn đúng thì lệnh S thực hiện.

Một số câu lệnh chính (9)

❖ Câu lệnh lặp

- ***Lặp với số lần lặp không biết trước:***
- *Vòng do while: do (S) while B;*



Ý nghĩa: Lặp lại lệnh S cho đến khi điều kiện B đúng

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Một số câu lệnh chính (10)

❖ Câu lệnh vào ra

- *Có dạng:*
- *cin>>danh sách biến>>…;*
- *cout<<danh sách biến hoặc dòng ký tự<<…;*

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

Chương trình con (thủ tục và hàm)

- ***Hàm:***

Kiểu <tên hàm>(<danh sách tham số>)

{ S1; S2;….Sn;

Return;

}

- ***Hàm không kiểu:***

void <tên hàm>(<danh sách tham số>)

{

S1; S2, …Sn ;

}

Sự khác nhau cơ bản giữa chúng là hàm không kiểu không trả lại kết quả, hàm trả lại kết quả thông qua tên hàm.

THUẬT TOÁN VÀ ĐỘ PHỨC TẠP

1.1 Khái niệm thuật toán

1.2 Thiết kế - Phân tích – Đánh giá thuật toán

1.3 Biểu diễn thuật toán

1.4 Ngôn ngữ diễn đạt thuật toán (tựa c)

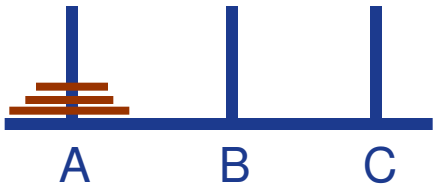
1.5 Đánh giá độ phức tạp thuật toán

1.5 Đánh giá độ phức tạp thuật toán

Tại sao lại cần thuật toán có hiệu quả?

❖ ***Ví dụ:*** **Bài toán tháp Hà Nội** (1)

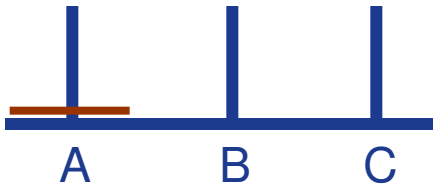
- *Có 3 cọc A, B, C. Lúc đầu, ở cọc A có n đĩa được lồng theo thứ tự nhỏ trên lớn dưới. Yêu cầu chuyển n đĩa từ cọc A sang cọc B với điều kiện:*
- *Mỗi lần chỉ được chuyển một đĩa*
- *Không có tình huống đĩa to ở trên đĩa nhỏ (dù là tạm thời)*
- *Được phép sử dụng cọc C làm trung gian.*

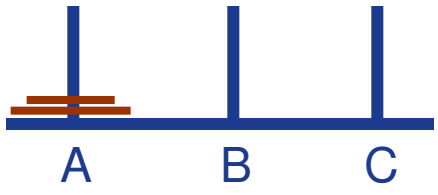


1.5 Đánh giá độ phức tạp thuật toán

Tại sao lại cần thuật toán có hiệu quả?

- ***Ví dụ:*** **Bài toán tháp Hà Nội** (2)
  - *Trường hợp một đĩa:*
    - *Chuyển đĩa từ cọc A sang cọc B.*
  - *Trường hợp 2 đĩa:*
    - *Chuyển đĩa thứ nhất từ cọc A sang cọc C;*
    - *Chuyển đĩa thứ hai từ cọc A sang cọc B;*
    - *Chuyển đĩa thứ nhất từ cọc C sang cọc B.*

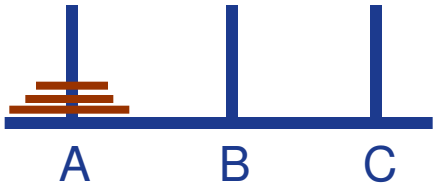




1.5 Đánh giá độ phức tạp thuật toán

Tại sao lại cần thuật toán có hiệu quả?

- ***Ví dụ:*** **Bài toán tháp Hà Nội** (3)
  - *Ta thấy với trường hợp n đĩa (n>2) nếu ta xem (n-1) đĩa ở trên đóng vai trò như đĩa thứ nhất thì có thể xử lý như trường hợp hai đĩa, nghĩa là:*
    - *Chuyển (n-1) đĩa trên từ A sang C*
    - *Chuyển đĩa thứ n từ A sang B*
    - *Chuyển (n-1) đĩa từ C sang A.*



1.5 Đánh giá độ phức tạp thuật toán

Tại sao lại cần thuật toán có hiệu quả?

- ***Ví dụ:*** **Bài toán tháp Hà Nội** (4)
  - *Nếu gọi F(n) là số lần chuyển đĩa.*
  - *Người ta chứng minh được* $F(n) = 2^n - 1$
  - Với n = 64, ta có $F(64) = 2^{64} - 1$ lần chuyển. Giả sử mỗi lần chuyển 1 đĩa từ cọc này sang cọc kia, cần 1 giây. Khi đó để thực hiện $2^{64} - 1$ lần chuyển cần 5.10^{11} năm. Nếu tuổi của vũ trụ là 10 tỉ năm , ta cần 50 lần tuổi của vũ trụ để chuyển 64 đĩa!
  - Qua thí dụ trên ta thấy, tuy bài toán tháp Hà Nội tồn tại thuật toán giải nhưng với n đủ lớn thì thuật toán không khả thi

1.5 Đánh giá độ phức tạp thuật toán

Đánh giá thời gian thực hiện thuật toán

- ***Có hai cách tiếp cận để đánh giá thời gian thực hiện của một thuật toán:***
  - *phương pháp thử nghiệm*
  - *phương pháp lý thuyết*

Thời gian chạy chương trình phụ thuộc vào các nhân tố sau đây:

1. *Các dữ liệu vào*
2. *Chương trình dịch để chuyển chương trình thành mã máy.*
3. *Tốc độ thực hiện các phép toán của máy tính được sử dụng để chạy chương trình .*

1.5 Độ phức tạp thuật toán

O(f(x)) và đánh giá thời gian thực hiện thuật toán

- ***Giả sử n là số nguyên không âm. T(n) và f(n) là các hàm thực không âm. Ta viết T(n) = O(f(n)) (đọc T(n) là ô lớn của f(n)), nếu và chỉ nếu tồn tại các hằng số dương c và n_0 sao cho***

  $T(n) <= c.f(n)$, với mọi $n >= n_0$.

  - ***Ví dụ*. Giả sử $T(n) = 3n^2 + 4n + 5$.**

  Ta có : $3n^2 + 4n + 5 <= 3n^2 + 4n^2 + 5n^2 = 12n^2$, với mọi $n >= 1$.

  - Vậy $T(n) = O(n^2)$. thuật toán có thời gian thực hiện cấp n^2, hoặc thuật toán có thời gian thực hiện bình phương.

1.5 Độ phức tạp thuật toán

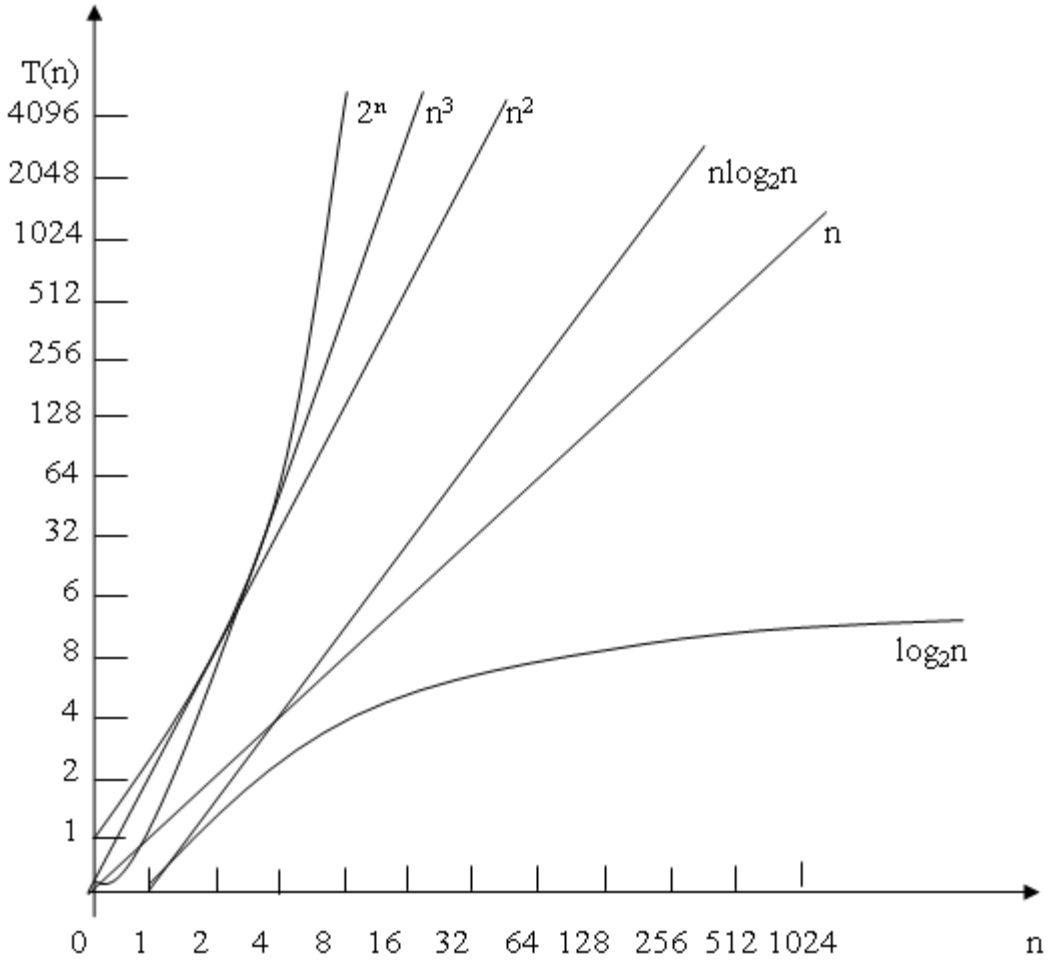
O(f(x)) và đánh giá thời gian thực hiện thuật toán

Tên gọi	Logarit	Tuyến tính	n log n	Bình phương	Lập phương	mũ	Giai thừa	n mũ n
f(n) / n	$\log_2 n$	N	$n\log_2 n$	n^2	n^3	2^n	n!	n^n
1	0	1	0	1	1	2	1	1
2	1	2	2	4	8	4	2	4
4	2	4	8	16	64	16	24	256
8	3	8	24	64	512	256	40320	134217728
16	4	16	64	256	4096	65536	$\sim 21.10^{12}$	$\sim 18.10^{18}$
32	5	32	160	1024	32768	2.147.483.648	$\sim 26.10^{61}$	$\sim 1.5x10^{56}$
64	6	64	384	4096	262144	$\sim 1.8x10^{27}$	$\sim 1.3x10^{97}$	~

Bảng phân cấp thời gian thực hiện thuật toán

1.5 Độ phức tạp thuật toán

O(f(x)) và đánh giá thời gian thực hiện thuật toán



1.5 Độ phức tạp thuật toán

O(f(x)) và đánh giá thời gian thực hiện thuật toán

- ***Xét ví dụ**: Giả sử một bài toán nào đó, có hai thuật toán giải là A và B.*
- *Thuật toán A có thời gian thực hiện là $T_A(n) = O(n^2)$*
- *Thuật toán B có thời gian thực hiện là $T_B = O(n.logn)$.*
- *Với n = 1024 thuật toán A cần 1048576 phép toán sơ cấp,*
- *thuật toán B đòi hỏi 10240 phép toán sơ cấp.*
- *Nếu cần một micro-giây cho một phép toán sơ cấp thì thuật toán A cần khoảng 1,05 giây trong khi đó thuật toán B cần khoảng 0,01 giây.*
- *Nếu n = 2048 , thì thuật toán A đòi hỏi khoảng 4,2 giây, trong khi thuật toán B chỉ đòi hỏi khoảng 0,02 giây.*
- *Vì vậy nếu một thuật toán có thời gian thực hiện $O(n^2)$ mà ta tìm ra được một thuật toán khác cho bài toán đó với thời gian O(n.logn) thì đó là một kết quả đáng kể, rất có ý nghĩa.*

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

- ***Qui tắc tổng :***

  Giả sử $T_1(n)$ và $T_2(n)$ là thời gian thực hiện của hai đoạn chương trình P_1 và P_2 mà

  $T_1(n) = O(f_1(n))$ *và*

  $T_2(n) = O(f_2(n))$

  thì thời gian thực hiện P_1 và P_2 kế tiếp nhau sẽ là:

  $T_1(n) + T_2(n) = O(\max(f_1(n), f_2(n)))$.

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

- ***Quy tắc nhân :***

  Nếu tương ứng với P_1 và P_2 là

  $T_1(n) = O(f_1(n))$,

  $T_2(n) = O(f_2(n))$

  thì thời gian thực hiện P_1 và P_2 lồng nhau sẽ là :

  $T_1(n).T_2(n) = O(f_1(n).f_2(n))$.

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

- ***Thí dụ**: Câu lệnh gán : x = x +1 có thời gian thực hiện bằng c (hằng số) nên được đánh giá là O(1).*
- *Câu lệnh for (i = 1;i<= n; i++) x = x +1 có thời gian thực hiện O(n.1) = O(n).*

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

- *Câu lệnh for (i = 1; i<=n; i ++)*

  for (j = 1;j<=n; j++) x = x +1 có thời gian được đánh giá là O(n.n) = O(n^2)

- *Cũng có thể thấy O(c.f(n)) = O(f(n)) chẳng hạn O(n^2/2) = O(n^2).*

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

- *1.Thời gian thực hiện các câu lệnh đơn*
- 2. Thời gian thực hiện câu lệnh điều kiện
- 3. Thời gian thực hiện lệnh Case
- 4. Thời gian thực hiện câu lệnh for
- 5. Thời gian thực hiện câu lệnh While
- 6. Thời gian thực hiện câu lệnh do while
- 7. Đánh giá độ phức tạp chương trình có chứa lời gọi hàm

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(1)Thời gian thực hiện các câu lệnh đơn

Lệnh đơn gồm: Phép gán, các câu lệnh đọc, viết, câu lệnh goto.

Thời gian thực hiện lệnh đơn: O(1) tức là thời gian thực hiện không đổi.

***Thí dụ**: xét đoạn chương trình sau:*

```
(1)     for (i = 1; i<= n-1; i++) {
(2)     small = i;
(3)     for (j = i +1; j<=n; j++)
(4)     if (A[j] < A[small])
(5)     small = j; {
(6)     tg = A[small];
(7)     A[small] = A[i];
(8)     A[i] := tg; }}
```

Các phép gán ở dòng (2), (5), (6), (7) và (8) tất cả đều là O(1).

Chú ý: Lệnh { S1, S2, ... ,Sn } là lệnh ghép.

Thời gian thực hiện lệnh ghép được xác định bởi luật tổng. Ở thí dụ trên thì (6), (7), (8) là lệnh ghép và theo luật tổng thì có O(1).

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(2) Thời gian thực hiện câu lệnh điều kiện

Câu lệnh if: if (B) S1; else S2;

B: điều kiện

S1, S2: lệnh.

Nếu điều kiện đúng thì lệnh S1 được thực hiện, điều kiện sai lệnh S2 thực hiện (S2 có thể không có).

Đánh giá thời gian thực hiện các lệnh if: Giả sử thời gian thực hiện các lệnh S1, S2, là O(f1(n)) và O(f2(n)) tương ứng. Khi đó thời gian thực hiện lệnh if là: O(max(f1(n),f2(n))).

Thí dụ:

```
(4)      if (A[j] < A[small])
(5)    small = j;
```

Điều kiện có cận là O(1). Phần thân là O(1), còn phần else là rỗng (bằng 0). Vậy câu lệnh if này có cận là O(1).

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(3) Thời gian thực hiện lệnh switch

```
switch   (B)
      {case v1:   S1;
      case v2:   S2;
      .............
      case vn :   Sn;
}
```

Đánh giá thời gian thực hiện lệnh switch như lệnh if.

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(4) Thời gian thực hiện câu lệnh for

for (i= m ; i<= n; i++) S

Với m, n nguyên và m <= n.

Đánh giá thân vòng for:

- lệnh gán,
- lệnh tăng chỉ số lặp,
- lệnh kiểm tra điều kiện dừng,.. đều là lệnh đơn nên thường có cận là O(1) = c.

Gọi f(n) là số lần thực hiện lệnh S. Khi đó thời gian thực hiện lệnh for là:
O(c.f(n)).

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(5) Thời gian thực hiện câu lệnh While

Câu lệnh: While (B) S

B: điều kiện, S: lệnh.

Thời gian thực hiện lệnh while được đánh giá: Giả sử thời gian thực hiện lệnh S (thân của lệnh while) là O(f(n)). Giả sử g(n) là số tối đa các lần thực hiện lệnh S khi thực hiện lệnh while. Khi đó thời gian thực hiện lệnh while là **O(f(n).g(n)).**

Thí dụ: xét đoạn chương trình:

(1) i = 1;

(2) while (x<>A[i])

(3) i = i +1;

Hai câu lệnh gán (1), (3) đều là O(1). Vòng lặp while ở hai dòng (2) và (3) có mục đích đi tìm vị trí của phần tử có giá trị bằng x trên mảng A (giả thiết x có mặt trên mảng). Như vậy tối đa số lần lặp là n (với n là số phần tử của mảng A). Từ đó suy ra thời gian thực hiện của đoạn chương trình trên là O(n).

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(6) Thời gian thực hiện câu lệnh do while

```
do
{
    S1, S2, .., Sn
}
while   (B);
```

B: điều kiện, Si (i = 1, 2,… , n) các câu lệnh.

Giả sử thời gian thực hiện khối begin S1, S2,…Sn end là O(f(n)). Giả sử g(n) là số tối đa các lần lặp. Khi đó thời gian thực hiện lệnh do while là **O(f(n).g(n)).**

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(7) Đánh giá độ phức tạp chương trình có chứa lời gọi hàm

❖ Chương trình không đệ quy

- Để đánh giá thời gian chạy của các chương trình (hay đoạn chương trình) có chứa lời gọi hàm nhưng trong đó không có lời gọi đệ quy.
  - Ta đánh giá từng hàm void một ở trong đó theo trật tự từ dưới lên
  - bắt đầu từ các hàm không có lời gọi hàm,
  - rồi lên dần qua các hàm mà các hàm được gọi trong đó đều đã được đánh giá,
  - cho đến khi đánh giá được chương trình chính.

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(7) Đánh giá độ phức tạp chương trình có chứa lời gọi hàm

❖ Chương trình không đệ quy

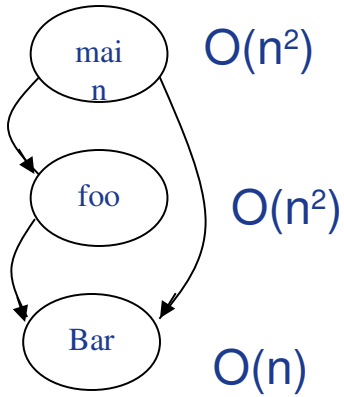
- Thí dụ: Hãy phân tích chương trình sau, trong đó có các lời gọi không đệ quy:

```
 int bar(int x, n)
 {int i;
(1)     For (i = 1 ;i<= n ;i++)
(2)     x = x +i;
(3)     bar = x;
 }
void foo(int  x, n)
{int i;
(4)     for (i = 1; i<= n; i++)
(5)     x = x + bar(i,n);
 }
 Main(input,output)
{int a,n;
(6)     cin>>n;
(7)     a = 0;
(8)     foo(a,n);
(9)     cout<<bar(a,n);
 }
```

$$bar(x,n) \quad x \quad \sum_{i\ 1}^{n} i \quad x \quad \frac{n(n\ \ 1)}{2}$$

$$\sum_{i\ 1}^{n} bar(i,n) \quad \sum_{i\ 1}^{n} (i \quad \frac{n(n\ \ 1)}{2}) \quad \frac{n^3 \quad 2n^2 \quad n}{2}$$

$$(n^3 + 2n^2 + n)/2$$



1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(7) Đánh giá độ phức tạp chương trình có chứa lời gọi hàm

❖ Chương trình có đệ quy

- Lời gọi đệ quy là lời gọi trong một thủ tục (hay hàm) P tới chính P.
- khi phân tích ta thay mỗi lời gọi bởi chi phí của thủ tục được gọi.

1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(7) Đánh giá độ phức tạp chương trình có chứa lời gọi hàm

❖ Chương trình có đệ quy

- Thí dụ 1:

Hãy phân tích và đánh giá chương trình tính giai thừa của n, như sau :

```
int giaithua(int n)
{
    If (n <= 1)
    Giaithua =1;
    Else
    Giaithua = n*giaithua(n-1)
}
```

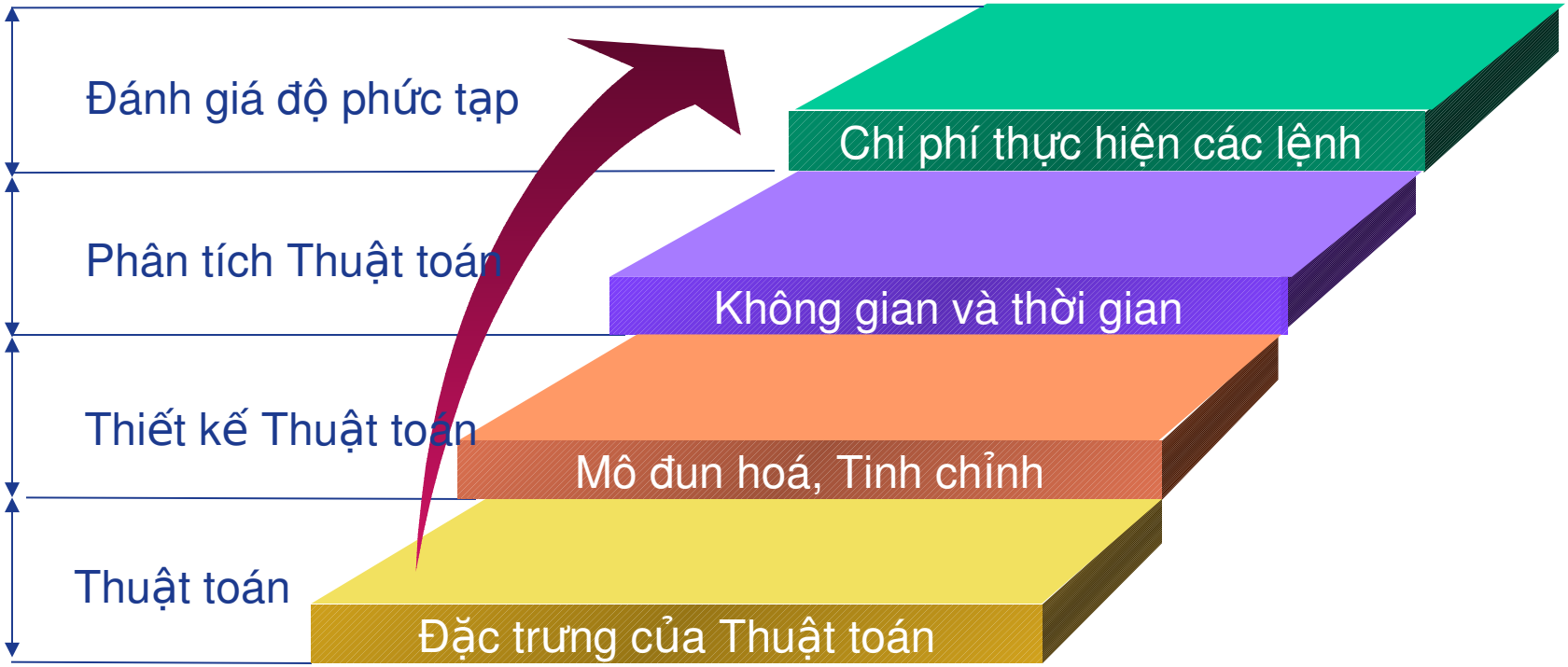
1.5 Độ phức tạp thuật toán

Các quy tắc đánh giá độ phức tạp về thời gian thực hiện thuật toán

(7) Đánh giá độ phức tạp chương trình có chứa lời gọi hàm

- **Chương trình có đệ quy**
  - Cơ sở: T(1) = O(1).
  - Quy nạp: T(n) = O(1) + T(n-1) với n >1.
  - Hóa giải các biểu thức O lớn, bằng cách đưa vào các hằng số a, b, ta có:
  - Cơ sở: T(1) = a.
  - Quy nạp: T(n) = b + T(n-1).
  - Từ đó ta có thể viết các phương trình liên tiếp theo giá trị tăng của n:
  - T(1) = a
  - T(2) = b +T(1)
  - T(3) = b +T(2)
  - …..
  - T(n-1) = b +T(n-2)
  - T(n) = b +T(n-1).
  - Cộng các phương trình này vế theo vế và loại bỏ các hạng thức chung ở hai vế ta có:
  - T(n) = a +(n-1)b = bn +(a-b)
  - Vậy T(n) = O(n).

Tổng kết chương



Bài tập

Bài tập:

Bài tập 1 đến bài tập 6 của chương 1

Nội dung nghiên cứu trước

Nghiên cứu trước chương 2

TRƯỜNG CAO ĐẲNG CNTT HỮU NGHỊ ViỆT - HÀN

KHOA KHOA HỌC MÁY TÍNH

----------***----------

Chúc các b n thành công !