HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

TUẦN 1 : CÁC KHÁI NIỆM CƠ BẢN

ONE LOVE. ONE FUTURE.

NỘI DUNG

1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được một số khái niệm cơ bản về thuật toán
2. Biết ký hiệu tiệm cận dùng để đánh giá độ phức tạp thuật toán
3. Biết cách phân tích độ phức tạp của thuật toán

NỘI DUNG TIẾP THEO

1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

1. VÍ DỤ MINH HỌA

- Bài toán tìm dãy con lớn nhất:
  - Cho dãy số gồm *n* số: $a_0, a_1, a_2, \dots, a_{n-1}$
  - Dãy gồm liên tiếp các số $a_i, a_{i+1}, \dots, a_j$ với $0 \le i \le j \le n-1$ được gọi là **dãy con** của dãy đã cho và $\sum_{k=i}^{j} a_k$ *được gọi là trọng lượng của dãy con này*
  - **Yêu cầu: Hãy tìm trọng lượng lớn nhất của các dãy con, tức là tìm cực đại giá trị** $\sum_{k=i}^{j} a_k$. Ta gọi dãy con có trọng lượng lớn nhất là **dãy con lớn nhất**.
- **Ví dụ:** Cho dãy số -2, **11, -4, 13**, -5, 2 thì cần đưa ra câu trả lời là 20 (dãy con lớn nhất là 11, -4, 13 với giá trị = 11+ (-4)+13 =20)

1. VÍ DỤ MINH HỌA

- Cách 1: Duyệt toàn bộ
  - Duyệt tất cả các dãy con có thể có của dãy đã cho: a_i, a_{i+1} , …, a_j với $0 \le i \le j \le n-1$, và tính tổng của mỗi dãy con để tìm ra trọng lượng lớn nhất.

```
int maxSum = a[0];
for (int i = 0; i <= n-1; i++) {
   for (int j = i; j <= n-1; j++) {
        int sum = 0;
        for (int k = i; k <= j; k++)  sum += a[k];
        if (sum > maxSum)   maxSum = sum;
    }
}
```

1. VÍ DỤ MINH HỌA

- Cách 1: Duyệt toàn bộ
  - Duyệt tất cả các dãy con có thể có của dãy đã cho: a_i, a_{i+1} , …, a_j với $0 \le i \le j \le n-1$, và tính tổng của mỗi dãy con để tìm ra trọng lượng lớn nhất.
  - **Phân tích thuật toán:** Ta sẽ tính số lượng phép cộng mà thuật toán phải thực hiện, tức là đếm xem dòng lệnh **sum += a[k]** phải thực hiện bao nhiêu lần. Số lượng phép cộng là:

```
int maxSum = a[0];
for (int i = 0; i<=n-1; i++) {
   for (int j = i; j<=n-1; j++) {
       int sum = 0;
       for (int k=i; k<=j; k++)  sum += a[k];
       if (sum > maxSum)   maxSum = sum;
    }
}
```

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)=\sum_{i=0}^{n-1}(1+2+\ldots+(n-i))=\sum_{i=0}^{n-1}\frac{(n-i)(n-i+1)}{2}$$

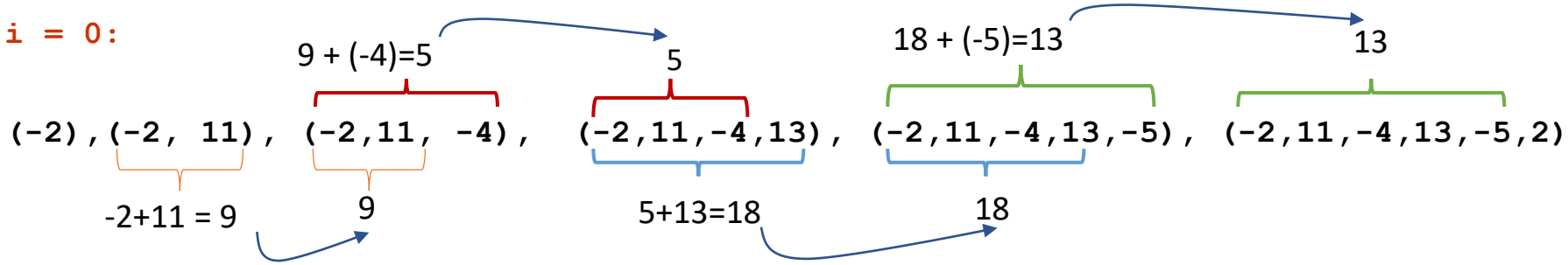
$$=\frac{1}{2}\sum_{k=1}^{n}k(k+1)=\frac{1}{2}\left[\sum_{k=1}^{n}k^2+\sum_{k=1}^{n}k\right]=\frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right]$$

$$=\frac{n^3}{6}+\frac{n^2}{2}+\frac{n}{3}$$

1. VÍ DỤ MINH HỌA

- Cách 2: Duyệt toàn bộ có cải tiến

Index i	0	1	2	3	4	5
a[i]	-2	11	-4	13	-5	2



- Nhận thấy, ta có thể tính tổng các phần tử từ vị trí *i* đến *j* từ tổng của các phần tử từ *i* đến *j*-1 chỉ bằng 1 phép cộng:

$$\sum_{k=i}^{j} a[k] = a[j] + \sum_{k=i}^{j-1} a[k]$$

Tổng các phần tử từ *i* đến *j* — Tổng các phần tử từ *i* đến *j*-1

1. VÍ DỤ MINH HỌA

- Cách 2: Duyệt toàn bộ có cải tiến

$$\underbrace{\sum_{k=i}^{j} a[k]}_{\text{Tổng các phần tử từ } i \text{ đến } j} = a[j] + \underbrace{\sum_{k=i}^{j-1} a[k]}_{\text{Tổng các phần tử từ } i \text{ đến } j\text{-1}}$$

```
int maxSum = a[0];
for (int i=0; i<=n-1; i++) {
  for (int j=i; j<=n-1; j++) {
      int sum = 0;
      for (int k=i; k<=j; k++) sum += a[k];
      if (sum > maxSum)  maxSum = sum;
   }
}
```

```
int maxSum = a[0];
for (int i=0; i<=n-1; i++) {
   int sum = 0;
   for (int j=i; j<=n-1; j++) {
      sum += a[j];
      if (sum > maxSum) maxSum = sum;
   }
}
```

1. VÍ DỤ MINH HỌA

- Cách 2: Duyệt toàn bộ có cải tiến
  - **Phân tích thuật toán:** Ta sẽ tính số lượng phép cộng mà thuật toán phải thực hiện, tức là đếm xem dòng lệnh **sum += a[j]** phải thực hiện bao nhiều lần. Số lượng phép cộng là:

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

```
int maxSum = a[0];
for (int i=0; i<=n-1; i++) {
    int sum = 0;
    for (int j=i; j<=n-1; j++) {
        sum += a[j];
        if (sum > maxSum) maxSum = sum;
    }
}
```

1. VÍ DỤ MINH HỌA

- Số lượng phép cộng mà mỗi thuật toán cần thực hiện là:
  - Cách 1. Duyệt toàn bộ $\frac{n^3}{6}+\frac{n^2}{2}+\frac{n}{3}$
  - Cách 2. Duyệt toàn bộ có cải tiến $\frac{n^2}{2}+\frac{n}{2}$
- Cùng một bài toán, ta đã đề xuất 2 thuật toán đòi hỏi số lượng phép toán khác nhau, và vì thế sẽ đòi hỏi thời gian tính khác nhau.
- Bảng dưới đây cho thấy thời gian tính của 2 thuật toán trên, với giả thiết: máy tính có thể thực hiện10^8 phép cộng trong một giây

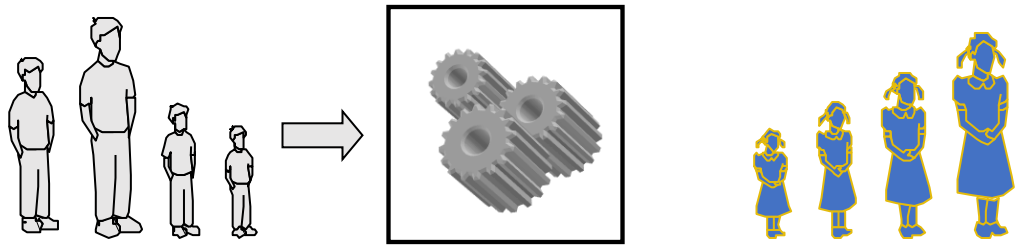
Độ phức tạp	n=10	Thời gian	n=100	Thời gian	$n=10^4$	Thời gian	$n=10^6$	Thời gian
n^3	10^3	10^{-5} giây	10^6	10^{-2} giây	10^{12}	2.7 giờ	10^{18}	115 ngày
n^2	100	10^{-6}giây	10000	10^{-4} giây	10^8	1 giây	10^{12}	2.7 giờ

NỘI DUNG

1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Thuật toán (Algorithm) giải bài toán đặt ra là một thủ tục xác định bao gồm một dãy hữu hạn các bước cần thực hiện để **thu được đầu ra (output)** từ **một đầu vào cho trước (input)** của bài toán.



Input **Algorithm** **Output**

- Một số đặc trưng cơ bản của thuật toán:
  - Chính xác
  - Hữu hạn
  - Đơn trị
  - Tổng quát

2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Độ phức tạp của thuật toán:
  - Khi nói đến hiệu quả của một thuật toán, ta quan tâm đến chi phí cần dùng để thực hiện nó:
    1) Dễ hiểu, dễ cài đặt, dễ sửa đổi ?
    2) Thời gian sử dụng CPU ? **THỜI GIAN**
    3) Tài nguyên bộ nhớ ? **BỘ NHỚ**

2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Độ phức tạp của thuật toán:
  - Làm thế nào để đo được thời gian tính?
    - Thời gian tính của thuật toán phụ thuộc vào dữ liệu vào (kích thước tăng, thì thời gian tăng).
    - Vì vậy, người ta tìm cách đánh giá thời gian tính của thuật toán bởi một hàm của độ dài dữ liệu đầu vào. Tuy nhiên, trong một số trường hợp, kích thước dữ liệu đầu vào là như nhau, nhưng thời gian tính lại rất khác nhau.
      - Ví dụ: Để tìm số nguyên tố đầu tiên có trong dãy: ta duyệt dãy từ trái sang phải

        Dãy 1: 3 9 8 12 15 20 (thuật toán dừng ngay khi xét phần tử đầu tiên)

        Dãy 2: 9 8 3 12 15 20 (thuật toán dừng khi xét phần tử thứ ba)

        Dãy 3: 9 8 12 15 20 3 (thuật toán dừng khi xét phần tử cuối cùng)

        ➔ 3 loại thời gian tính

2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Các loại thời gian tính của thuật toán:
  - **Thời gian tính tốt nhất (Best-case)**

    T(*n*): thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước *n*.

  - **Thời gian tính trung bình (Average-case)**

    T(*n*): thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước *n*.

  - **Thời gian tính tồi nhất (Worst-case)**

    T(*n*): thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước *n*.

2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Có hai cách để đánh giá thời gian tính:
  - **Từ thời gian chạy thực nghiệm:**
    - cài đặt thuật toán, rồi chọn các bộ dữ liệu đầu vào thử nghiệm
    - chạy chương trình với các dữ liệu đầu vào kích thước khác nhau
    - sử dụng hàm clock( ) để đo thời gian chạy chương trình

```
clock_t startTime = clock();
doSomeOperation();
clock_t endTime = clock();
clock_t clockTicksTaken = endTime - startTime;
double timeInSeconds = clockTicksTaken / (double) CLOCKS_PER_SEC;
```

  - **Lý thuyết: khái niệm xấp xỉ tiệm cận**

NỘI DUNG

1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

3. KÝ HIỆU TIỆM CẬN

- Các ký hiệu tiệm cận (asymptotic notation):

$$\Theta, \Omega, O, \omega$$

  - Được sử dụng để mô tả thời gian tính của thuật toán, mô tả tốc độ tăng của thời gian chạy phụ thuộc vào kích thước dữ liệu đầu vào.
  - Ví dụ, khi nói thời gian tính của thuật toán cỡ $\Theta(n^2)$, tức là, thời gian tính tỉ lệ thuận với n^2 cộng thêm các đa thức bậc thấp hơn.

3. KÝ HIỆU TIỆM CẬN

3.1. Ký hiệu tiệm cận theta Θ

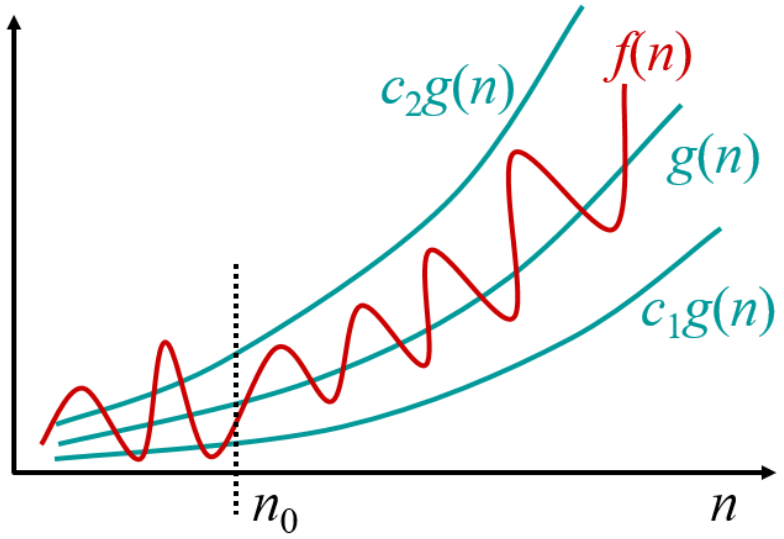
- Đối với hàm $g(n)$ cho trước, ta kí hiệu $\Theta(g(n))$ là tập các hàm:

$$\Theta(g(n)) = \{f(n): \text{tồn tại các hằng số } c_1, c_2 \text{ và } n_0 \text{ sao cho:}$$

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n), \text{ với mọi } n \ge n_0 \}$$

(tập tất cả các hàm có cùng tốc độ tăng với hàm $g(n)$)

- Khi ta nói một hàm là theta của hàm khác, nghĩa là không có hàm nào đạt tới giá trị vô cùng nhanh hơn.



3. KÝ HIỆU TIỆM CẬN

3.1. Ký hiệu tiệm cận theta Θ

- Ví dụ: Chứng minh rằng $10n^2 - 3n = \Theta(n^2)$

Ta cần chỉ ra với những giá trị nào n_0, c_1, c_2 thì bất đẳng thức trong định nghĩa của kí hiệu theta là đúng:

$$c_1 n^2 \leq f(n) = 10n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

Gợi ý: lấy c_1 nhỏ hơn hệ số của số hạng với số mũ cao nhất, và c_2 lấy lớn hơn.

➔ Chọn: $c_1 = 1$, $c_2 = 11$, $n_0 = 1$ thì ta có

$n^2 \leq 10n^2 - 3n \leq 11n^2$, *với* $n \geq 1$

➔ $\forall n \geq 1$: $10n^2 - 3n = \Theta(n^2)$

Chú ý: Với các hàm đa thức: để so sánh tốc độ tăng, ta cần nhìn vào số hạng có số mũ cao nhất

3. KÝ HIỆU TIỆM CẬN

3.2. Ký hiệu tiệm cận O lớn O

- Đối với hàm $g(n)$ cho trước, ta kí hiệu $O(g(n))$ là tập các hàm:

$$O(g(n)) = \{f(n): \text{tồn tại các hằng số dương } c \text{ và } n_0 \text{ sao cho:}$$

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

(tập tất cả các hàm có **tốc độ tăng nhỏ hơn hoặc bằng** tốc độ tăng của g(n))

- $O(g(n))$ là tập các hàm đạt tới giá trị vô cùng không nhanh hơn $g(n)$.
- Ví dụ: Chứng minh rằng $2\boldsymbol{n} + 10 = \boldsymbol{O}(\boldsymbol{n})$

➔ $f(n) = 2n+10$, $g(n) = n$

- Cần tìm hằng số c và n_0 sao cho:

$2\boldsymbol{n} + 10 \leq \boldsymbol{cn}$ với mọi $n \geq n_0$

➔ $(\boldsymbol{c} - 2)\,\boldsymbol{n} \geq 10$

➔ $\boldsymbol{n} \geq 10/(\boldsymbol{c} - 2)$

➔ chọn $\boldsymbol{c} = 3$ và $\boldsymbol{n_0} = 10$

3. KÝ HIỆU TIỆM CẬN

3.2. Ký hiệu tiệm cận O lớn O

- Chú ý: Có $f(n) = 50n^3 + 20n + 4$ là $O(n^3)$
  - Cũng đúng khi nói $f(n)$ là $O(n^3+n)$
    - Không hữu ích, vì n^3 có tốc độ tăng lớn hơn rất nhiều so với n, khi n lớn
  - Cũng đúng khi nói f(n) là $O(n^5)$
    - Đúng, nhưng g(n) nên có tốc độ tăng càng gần với tốc độ tăng của f(n) càng tốt, thì đánh giá thời gian tính mới có giá trị
- Quy tắc đơn giản: Bỏ qua các số hạng có số mũ thấp hơn và các hằng số
  - Ví dụ:
    - Tất cả các hàm sau đều là $O(n)$: n, $3n$, $61n + 5$, $22n – 5$, …
    - Tất cả các hàm sau đều là $O(n^2)$: n^2, $9\ n^2$, $18\ n^2 + 4n – 53$, …
    - Tất cả các hàm sau đều là $O(n \log n)$: $n(\log n)$, $5n(\log 99n)$, $18 + (4n – 2)(\log (5n + 3))$,...

3. KÝ HIỆU TIỆM CẬN

3.3. Ký hiệu tiệm cận Omega Ω

- Đối với hàm $g(n)$ cho trước, ta kí hiệu $\Omega(g(n))$ là tập các hàm:

$$\Omega(g(n)) = \{f(n): \text{tồn tại các hằng số dương } c \text{ và } n_0 \text{ sao cho:}$$

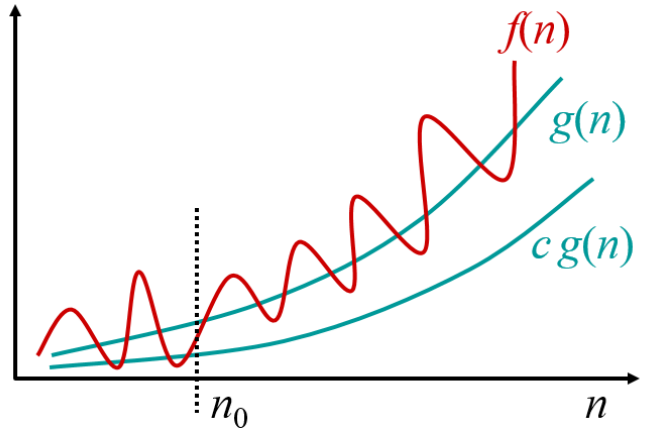
$$cg(n) \leq f(n) \text{ với mọi } n \geq n_0\}$$

(tập tất cả các hàm có **tốc độ tăng lớn hơn hoặc bằng** tốc độ tăng của $g(n)$)

- $\Omega(g(n))$ là tập các hàm đạt tới giá trị vô cùng không chậm hơn $g(n)$.
- Ví dụ: Chứng minh rằng $5n^2 = \Omega(n)$

  Cần tìm c và n_0 sao cho $\boldsymbol{cn} \leq 5n^2$ với $n \geq n_0$

  Bất đẳng thức đúng với $c = 1$ và $n_0 = 1$



NỘI DUNG

1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- **Cấu trúc tuần tự:**

  Thời gian tính của chương trình "**P; Q**",với P và Q là hai đoạn chương trình thực thi một thuật toán, P thực hiện trước, rồi đến Q là: $Time(P; Q) = Time(P) + Time(Q)$ hoặc ta có thể dùng kí hiệu tiệm cận theta: $Time(P; Q) = \Theta(\max(Time(P), Time(Q))$

  với $Time(P)$, $Time(Q)$ là thời gian tính của P và Q.

- **Vòng lặp FOR**

  for i =1 **to** m **do** P(i);

  Giả sử thời gian thực hiện $P(i)$ là $t(i)$, khi đó thời gian thực hiện vòng lặp for là $\sum_{i=1}^{m} t(i)$

- **Vòng lặp FOR lồng nhau**

```
for i =1 to n do
    for j =1 to m do P(j);
```

Giả sử thời gian thực hiện *P*(*j*) là *t*(*j*), khi đó thời gian thực hiện vòng lặp lồng nhau này là:

- **Cấu trúc If/Else**

```
if (điều_kiện) then P;
else  Q;
endif;
```

Thời gian thực hiện câu lệnh if/else

= thời gian kiểm tra (điều_kiện) + max (Time(P), Time (Q))

- Ví dụ

Case1: for (i=0; i<n; i++)
for (j=0; j<n; j++)
k++;

$O(n^2)$

$O(n^2)$

$O(n^2)$

- **Câu lệnh đặc trưng:** *là câu lệnh được thực hiện thường xuyên ít nhất là cũng như bất kỳ câu lệnh nào trong thuật toán.*
- Nếu giả thiết thời gian thực hiện mỗi câu lệnh là bị chặn bởi hằng số thì thời gian tính của thuật toán sẽ cùng cỡ với số lần thực hiện câu lệnh đặc trưng. Do đó, để đánh giá thời gian tính, người ta đếm số lần thực hiện câu lệnh đặc trưng.
- Ví dụ 1: Hàm tính số Fibonacci $f_0=0; f_1=1; f_n= f_{n-1} + f_{n-2}$

```
function Fibiter(n)
    i=0;
    j=1;
    for k=1 to n-1 do
        j = i + j;
        i = j – i;
    return j;
```

Số lần thực hiện câu lệnh đặc trưng là n

➔ Thời gian chạy Fibiter là $O(n)$

- Ví dụ 2: Bài toán dãy con lớn nhất
  - Thuật toán 1: Duyệt toàn bộ

```
int maxSum = a[0];
for (int i=0; i<n; i++) {
   for (int j=i; j<n; j++) {
      int sum = 0;
      for (int k=i; k<=j; k++)
          sum += a[k];
      if (sum > maxSum)
          maxSum = sum;
   }
}
```

Chọn câu lệnh đặc trưng là **sum+=a[k]**

➔ Thời gian tính của thuật toán: $O(n^3)$

- Ví dụ 2: Bài toán dãy con lớn nhất
  - Thuật toán 2: Duyệt toàn bộ có cải tiến

```
int maxSum = a[0];
for (int i=0; i<n; i++) {
   int sum = 0;
   for (int j=i; j<n; j++) {
      sum += a[j];
      if (sum > maxSum)
          maxSum = sum;
   }
}
```

Chọn câu lệnh đặc trưng là **sum+=a[j]**

➔ Thời gian tính của thuật toán: $O(n^2)$

- Ví dụ 3: Đưa ra đánh giá tiệm cận O lớn cho thời gian tính $T(n)$ của đoạn chương trình sau:

a)
```
int x = 0;
for (int i = 1; i <= n; i *= 2)  x=x+1;
```

b)
```
int x = 0;
for (int i = n; i > 0; i /= 2) x=x+1;
```

HUST

hust.edu.vn fb.com/dhbkhn

TỔNG KẾT VÀ GỢI MỞ

1. Bài học đã trình bày **các khái niệm cơ bản về thuật toán và độ phức tạp thuật toán**
2. Tiếp sau bài này, người học sẽ được học về **đệ quy – sơ đồ chung và một số ví dụ**

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !