Design and Analysis of Algorithms

Lecture 1
Introduction

Lecturer: Nguyen Mau Uyen
uyennm@mta.edu.vn

Nội dung

1. Giới thiệu
2. Thuật toán
   - Khái niệm thuật toán
   - Biểu diễn thuật toán
   - Tính đúng đắn và hiệu quả của TT
3. Đánh giá độ phức tạp TT
   - Độ tăng của hàm
   - Độ phức tạp của TT
   - Đánh giá bằng thực nghiệm

Nội dung

1. **Giới thiệu**
2. Thuật toán
3. Độ phức tạp thuật toán

Mục đích

- Cung cấp kiến thức về việc đánh giá thuật toán
  - Lý thuyết
  - Thực nghiệm
- Thiết kế thuật giải
  - Chia để trị
  - Tham lam
  - Quy hoạch động
  - …

Nội dung môn học

- Tổng quan về thuật toán và độ phức tạp của thuật toán
- Đánh giá thuật toán
- Thiết kế thuật toán
- Phương pháp thiết kế thuật toán
  - Trực tiếp
  - Chia để trị
  - Tham lam …

Hình thức kiểm tra

- 10% Chuyên cần
- 20% Thường xuyên (bài tập, bài kiểm tra)
- 70% Thi cuối kỳ (vấn đáp)
  - Mô tả bài toán
  - Thiết kế thuật toán
  - Đánh giá
  - Cài đặt
  - Báo cáo

Tài liệu tham khảo

- Slide bài giảng.
- Bài giảng Thiết kế và Đánh giá Thuật toán, Trần Xuân Sinh, NXB, ĐHQG, 2010.
- Cẩm nang thuật toán, Robert Sedgewich - Trần Đan Thư dịch (tái bản lần 2), NXB KHKT, 2006.
- Cấu trúc dữ liệu và giải thuật, Đỗ Xuân Lôi, NXB ĐH Quốc Gia, 2006.
- Giải một bài toán trên máy tính như thế nào (3 tập), Hoàng Kiếm, NXB Giáo dục, 2005

Tài liệu tham khảo

- Giải thuật và lập trình (bài giảng chuyên đề), Lê Minh Hoàng, ĐHSP, 2002.
- Computer Algorithms Introduction to Design and Analysis, Addison-Wesley, 1988.
- Algorithms and Complexity, Herbert S. Wilf, University of Pennsylvania, Philadelphia 1999.
- Algorithm Design, Jon Kleinberg, Eva Tardos Pearson, 2006

Nội dung

1. Giới thiệu
2. **Thuật toán**
3. Độ phức tạp thuật toán

Khái niệm

- Một thuật toán là một bản liệt kê các chỉ dẫn, các quy tắc cần thực hiện theo từng bước xác định nhằm giải quyết một bài toán đã cho trong một khoảng thời gian hữu hạn.

- Ví dụ: Mô tả thuật toán giải quyết bài toán tìm phần tử lớn nhất trong dãy có n số cho trước.

Ví dụ 1

1. Chỉ số phần tử lớn nhất tạm thời (LNTT) = chỉ số phần tử đầu tiên;
2. So sánh số tiếp theo với giá trị lớn nhất tạm thời, nếu lớn hơn giá trị lớn nhất tạm thời thì đặt:
   - Chỉ số phần tử LNTT = chỉ số phần tử đó;
3. Lặp lại bước 2) nếu còn phần tử trong dãy.
4. Phần tử lớn nhất tạm thời ở thời điểm này chính là phần tử lớn nhất trong dãy.

Ví dụ 2

- Mô tả dưới dạng giả mã

input: n và a(1), ..., a(n);
output: i_o, chỉ số phần tử lớn nhất trong a;
1) $i_o = 1$;
2) for (i =2; $i \leq n$; i+=)
if $(a(i_o) < a(i))$ $i_o = i$;
3) output i_o;

Tính chất của TT …

1. **Tính chính xác**: để đảm bảo kết quả tính toán hay các thao tác mà máy tính thực hiện được là chính xác.
2. **Tính rõ ràng**: Thuật toán phải được thể hiện bằng các câu lệnh minh bạch; các câu lệnh được sắp xếp theo thứ tự nhất định.

Tính chất của TT …

3. **Tính khách quan**: Một thuật toán dù được viết bởi nhiều người trên nhiều máy tính vẫn phải cho kết quả như nhau.
4. **Tính phổ dụng**: Thuật toán không chỉ áp dụng cho một bài toán nhất định mà có thể áp dụng cho một lớp các bài toán có đầu vào tương tự nhau.
5. **Tính kết thúc**: Thuật toán phải gồm một số hữu hạn các bước tính toán.

Biểu diễn thuật toán

- Có 3 cách biểu diễn thuật toán:
  - Dùng ngôn ngữ tự nhiên
  - Sơ đồ khối và
  - Giả mã.
- Dùng ngôn ngữ tự nhiên: mô tả các bước xử lý bằng ngôn ngữ viết.

Mô tả dữ liệu vào/ra

- *Dữ liệu đầu vào*: Một thuật toán phải mô tả rõ các giá trị đầu vào từ một tập hợp các dữ liệu xác định. Ví dụ, dãy số nguyên a(1), a(2), ..., a(n), với $n<\infty$.
- *Dữ liệu đầu ra*: Từ một tập các giá trị đầu vào, thuật toán sẽ tạo ra các giá trị đầu ra. Các giá trị đầu ra chính là nghiệm của bài toán. Ví dụ, i_o là chỉ số phần tử lớn nhất trong a(1),...,a(n).

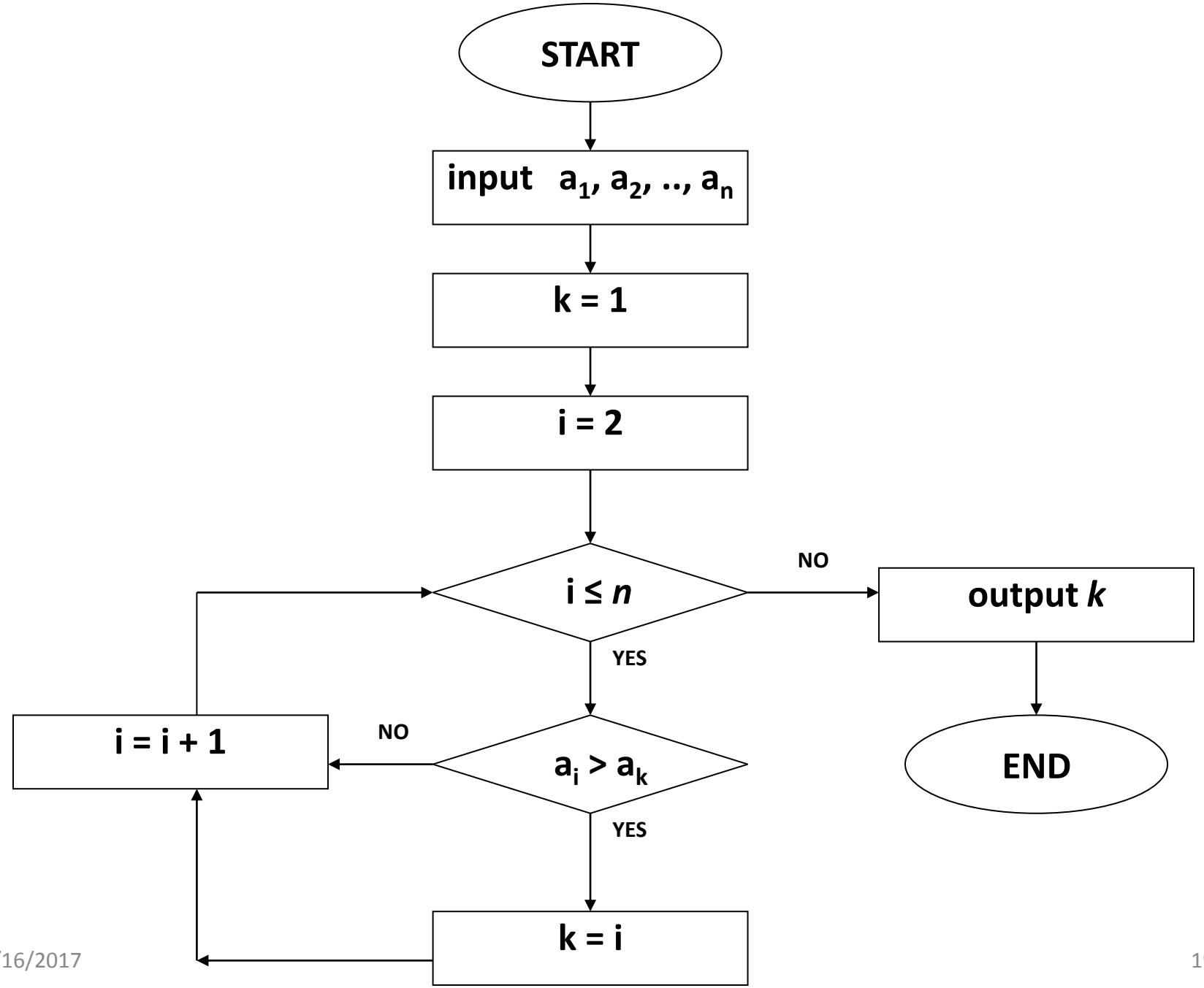
Ví dụ

- Thuật toán tìm số lớn nhất

1) Dữ liệu vào: dãy có n phần tử
2) Dữ liệu ra: Chỉ số phần tử lớn nhất
3) Chỉ số phần tử lớn nhất tạm thời = chỉ số phần tử đầu tiên;
4) So sánh số tiếp theo với giá trị lớn nhất tạm thời GTLNTT, nếu lớn hơn GTLNTT thì đặt
   Chỉ số phần tử lớn nhất tạm thời = chỉ số phần tử đó;
5) Lặp lại bước 2) nếu còn phần tử trong dãy.
6) Phần tử lớn nhất tạm thời ở thời điểm này chính là phần tử lớn nhất trong dãy.

Sơ đồ khối

- Sử dụng bộ kí hiệu các khối để thể hiện thuật toán.
- Bộ kí hiệu:
  - Hộp chữ nhật: Các toán tử gán và phép toán tính toán;
  - Hình thoi: Phép toán so sánh cho kết quả thuộc tập: {đúng, sai}.
  - Đường kẻ + mũi tên: Chỉ thao tác tiếp theo;
  - Hình elip: Biểu thị sự bắt đầu hoặc kết thúc thuật toán


START
input $a_1, a_2, .., a_n$
k = 1
i = 2
i ≤ n
NO
output k
YES
END
i = i + 1
NO
$a_i > a_k$
YES
k = i

Giả mã

- Sử dụng ngôn ngữ tự nhiên kết hợp với một ngôn ngữ lập trình.
- Cần tuân thủ quy tắc của một ngôn ngữ:
  - Có các cấu trúc cơ bản: tuần tự, lặp và rẽ nhánh.
  - Có hệ thống từ khóa, từ điển (phụ thuộc vào bài toán).
- Dễ hiểu, dễ cài đặt.

Ví dụ 1: Tìm phần tử lớn nhất …

input: n và a(1), …, a(n);

output: i_o, chỉ số phần tử lớn nhất trong a;

1) $i_o = 1$;
2) for (i =2; $i \le n$; i+=)
   if $(a(i_o) < a(i))$ $i_o = i$;
3) output i_o;

Ví dụ 2: Tìm phần tử có giá trị b

- Tìm trong dãy có *n* số cho trước.

input: $a_1, a_2, ..., a_n, b;$

output: chỉ số i_o phần tử đầu tiên có giá trị bằng *b*, nếu không có, $i_o = -1;$

1) $i = 1;$
2) while $(i \le n)$ & $(a_i \ne b)$ i++;
3) if $(i \le n)$ $i_o = i;$
   else $i_o = -1;$
4) output $i_o;$

Chất lượng biểu diễn thuật toán

1. Đúng với ý tưởng đặt ra của bài toán
2. Đơn giản, dễ hiểu
3. Dễ cài đặt

Tính đúng đắn của thuật toán

- PP lý thuyết, PP thực nghiệm
- Phương pháp lý thuyết:
  - Chứng minh thuật toán cho ra kết quả phù hợp với bài toán
  - Thuật toán kết thúc và cho ra kết quả
  - Kết quả phù hợp với yêu cầu của bài toán
  - Ví dụ:
    - Các thuật toán dựa trên qui hoặc động
    - Các thuật toán vét cạn …

Tính đúng đắn của thuật toán

- Phương pháp thực nghiệm:
  - Thuật toán có thể được xây dựng trên một ý tưởng dạng intuitive, là gần đúng hoặc không chứng minh được tính đúng đắn của thuật toán bằng phương pháp lý thuyết.
  - Thuật toán được chương trình hóa và được thực hiện trên một tập dữ liệu đủ lớn, bao hàm các trường hợp có thể xảy ra đối với bài toán ban đầu.

Tính hiệu quả

1. Thời gian: Chi phí cho thời gian tính toán ít hơn so với các thuật toán giải quyết cùng bài toán.
2. Bộ nhớ: Chiếm dụng bộ nhớ ít hơn so với các thuật toán giải quyết cùng bài toán.
3. Độ chính xác: Nếu là cung cấp lời giải gần đúng thì gần với lời giải đúng hơn so với thuật toán giải quyết cùng bài toán

Tính hiệu quả

- Tùy theo bài toán, mục đích mà xét các tiêu chí nói trên.
- Tùy theo chất lượng của thuật toán mà có thể xét trên mọi tập dữ liệu thử nghiệm, hoặc trên một số tập dữ liệu minh họa cho một vài khía cạnh nào đó.

Nội dung

1. Giới thiệu
2. Thuật toán
3. **Độ phức tạp thuật toán**

Độ tăng của hàm

- Cho f và g, f: $R \to R$, g: $R \to R$.
- **Định nghĩa 1**: f(x) = o(g(x)) khi x dần tới dương vô cùng, nếu như $\lim_{x \to +\infty} f(x)/g(x) = 0$.

  f(x) tăng chậm hơn so với g(x) khi x lớn dần đến $+\infty$.
- Ví dụ:
  - $x^2 = o(x^5)$, $\sin(x) = o(x)$
  - $1/x = o(1)$ …

Độ tăng của hàm …

- Cho f và g, f: R$\rightarrow$R, g: R$\rightarrow$R.
- **Định nghĩa 2**: f(x) là O-lớn của g(x) khi x $\rightarrow\infty$, kí hiệu f(x) = O(g(x)), nếu $\exists$ C>0 và N >0 sao cho $\forall$ x > N thì $|f(x)| \leq C.|g(x)|$.
- Ví dụ:
  - $f(x) = x^2+2x+3 = O(x^2)$, vì với mọi $x>1$ ta có $f(x) \leq x^2 + 2x^2 + 3x^2 = 6x^2$. Ngược lại, $x^2 = O(f(x))$ vì hiển nhiên là với mọi $x>0$ ta có $x^2 < f(x)$.

Độ tăng của hàm ...

- Cho f và g, $f: R \to R$, $g: R \to R$.
- **Định nghĩa 2**: f(x) là O-lớn của g(x) khi $x \to \infty$, kí hiệu $f(x) = O(g(x))$, nếu $\exists\ C>0$ và $N>0$ sao cho $\forall\ x > N$ thì $|f(x)| \leq C.|g(x)|$.
- Ví dụ:
  - $kx^2 = O(x^3)$ với $k>0$, vì với $x \geq k$ ta có $kx^2 \leq 1.x^3$.
  - $1/(1+x^2) = O(1)$
  - $\sin(x) = O(1)$

Cặp giá trị C và N, nếu tồn tại, rõ ràng không phải là duy nhất

Độ tăng của hàm …

- Cho f và g, f: $R \to R$, g: $R \to R$.
- **Định nghĩa 3**: f(x) tương đương với g(x) khi x dần tới dương vô cùng, kí hiệu $f(x) \approx g(x)$, nếu như $\lim_{x \to +\infty} f(x)/g(x) = 1$.
- Ví dụ:
  - $1+x+x^2 \approx x^2$,
  - $(2x+4)^2 \approx 16x^2$
  - …

Độ tăng của hàm …

- Cho f và g, f: R$\rightarrow$R, g: R$\rightarrow$R.
- **Định nghĩa 4**: Ta nói rằng f(x) = Ω(g(x)) nếu như tồn tại C>0 và dãy $x_1, x_2, x_3, \ldots \rightarrow +\infty$, sao cho với mọi i: $|f(x_i)| > Cg(x_i)$.
- Ví dụ:
  - x = Ω(log(x))
  - …

Độ tăng của hàm …

- Cho f và g, f: R$\rightarrow$R, g: R$\rightarrow$R.
- **Định nghĩa 5**: Ta nói rằng hàm f tăng theo hàm mũ nếu tồn tại c >1 và d sao cho f(x) = $\Omega(c^x)$ và f(x) = $O(d^x)$.
- Ví dụ:
  - $f(x) = e^{2x}$,
  - f(n) = n!

Một số tính chất của O-lớn

- $f(x)=a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n = O(x^n)$
- <u>Chứng minh</u>

Kí hiệu $C = |a_0| + |a_1| + |a_2| + ... + |a_{n-1}| + |a_n|$.

Với $x>1$ ta có $x^k < x^n$, với $k<n$, suy ra

$$|f(x)| < |a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n| \leq$$
$$|a_0| + |a_1x^1| + |a_2x^2| + ... + |a_{n-1}x^{n-1}| + |a_nx^n| \leq$$
$$|a_0| + |a_1|.\ x + |a_2|.\ x^2 + ... + |a_{n-1}|.\ x^{n-1} + |a_n|.\ x^n \leq$$
$$(|a_0| + |a_1| + |a_2|. + ... + |a_{n-1}| + |a_n|).\ x^n \leq C.\ x^n.\ (đpcm)$$

Qui tắc cộng

- $f(x)=O(u(x))$ và $g(x)=O(v(x)) \Rightarrow$
  $(f+g)(x) = O(\max\{u(x),v(x)\})$
- <u>Chứng minh</u>
  - Từ giả thiết suy ra tồn tại C_1, k_1, C_2, k_2 để
    - $x > k_1$ thì $f(x) \leq C_1.u(x)$, và
    - $x > k_2$ thì $g(x) \leq C_2.v(x)$
  - Đặt $k = \max\{ k_1, k_2\}$, và $C=C_1+C_2$
  - Rõ ràng là với mọi $x > k$ thì $f(x) \leq C.u(x)$ và $g(x) \leq C.v(x)$ hay $f(x)+g(x) \leq C \max\{ u(x), v(x) \}$ (đpcm)

Qui tắc nhân

- $f(x) = O(u(x))$ và $g(x) = O(v(x)) \Rightarrow$
  $(f.g)(x) = O(u(x).v(x))$
- Chứng minh
  - Bài tập về nhà

Độ phức tạp của TT

- Tính hiệu quả của thuật toán thông thường được đo bởi:
  - Thời gian tính (thời gian được sử dụng để tính bằng máy hoặc bằng phương pháp thủ công) khi các giá trị đầu vào có kích thước xác định
  - Dung lượng bộ nhớ đã sử dụng để tính toán khi kích thước đầu vào đã xác định.

Độ phức tạp của TT …

- Hai thước đo đã nêu ở trên liên quan đến độ phức tạp tính toán của một thuật toán, được gọi là:
  - Độ phức tạp thời gian và
  - Độ phức tạp không gian (dung lượng nhớ)
- Trước đây khi kích thước bộ nhớ còn giới hạn người ta quan tâm đến độ phức tạp KG.
- Bây giờ: Độ phức tạp thuật toán được hiểu là độ phức tạp thời gian.

Độ phức tạp thời gian

- Độ phức tạp thời gian của một thuật toán thường được biểu diễn thông qua số phép toán (cơ bản) trong khi thực hiện thuật toán với các giá trị dữ liệu đầu vào có **kích thước** (n) xác định.
- Lý do: Thời gian chạy thực còn phụ thuộc vào máy (chứ không chỉ thuật toán).
- Thường đánh giá trên số lượng các phép toán cơ bản như: so sánh, gán, cộng, trừ, nhân, chia

Ví dụ 1

- Xét thuật toán tìm kiếm phần tử lớn nhất trong dãy số nguyên cho trước như sau:

```
input:   a (1), ..., a(n);
output:  io - chỉ số phần tử lớn nhất trong a;

        io  = 1;
        for  (i = 2; i ≤ n; i++)
            if  (a(io) < a(i) )
                io  = i;
```

Kích thước bài toán: số phần tử - **n**

Số các phép so sánh cần dùng: **2(n-1) = O(n)**

Ví dụ 1…

- Ta nói, thuật toán

```
input:   a (1), ..., a(n);
output:  io - chỉ số phần tử lớn nhất trong a;

        io  = 1;
        for  (i = 2; i ≤ n; i++)
             if  (a(io) < a(i) )
                  io  = i;
```

Có độ phức tạp **O(n)**

Ví dụ 2

- Tính giá trị đa thức $f(x) = a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n$ tại x_o

Thuật toán 1:

```
input: a0, a1, ..., an;     xo;
output:     f(xo);
        x = xo;
        f = a0;
        for ( i = 1; i ≤ n; i++)
                {       x =  x * xo ;
                        f = f  +  a(i)* x ;
                }
        return f;
```

Thuật toán 1:

input: $a_0, a_1, \ldots, a_n$; x_o;
output: $f(x_o)$;

```
x = xo;
f = a0;
for ( i = 1; i ≤ n; i++)
        {       x =  x * xo ;
                f = f  +  a(i)* x ;
        }
return f;
```

- Đánh giá
  - Số phép toán so sánh, cộng, gán, nhân *n*, 2*n*, 2*n*, 2*n*.
  - Nếu x_o là giá trị lớn, giá trị trung gian $x = x_o^n$ có thể sẽ rất lớn

Thuật toán 2 (*Hoorner*):

```
f = a(n) ;
for ( i = 1; i ≤ n; i++)
      f = f * x_o + a(n-i) ;
return f;
```

- Đánh giá
  - Số phép toán so sánh, cộng, gán, nhân là *n*, 2*n*, *n*, *n*.
  - Tránh được phép tính x_o^n.

Ví dụ 2 …

- Tính giá trị đa thức $f(x) = a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n$ tại x_o
- **Thuật toán 1**: Số phép toán so sánh, cộng, gán, nhân *n*, 2*n*, 2*n*, 2*n*.
- **Thuật toán 2**: Số phép toán so sánh, cộng, gán, nhân là *n*, 2*n*, *n*, *n*

Kích thước bài toán: số phần tử - **n**

Độ phức tạp thuật toán: **O(n)**

Một số đánh giá thường dùng

- Độ phức tạp thuật toán thường được đưa về một trong các dạng độ phức tạp sau đây

$O(1)$	Độ phức tạp hằng số
$O(\log n)$	Độ phức tạp logarit
$O(n)$	Độ phức tạp tuyến tính.
$O(n^k)$	Độ phức tạp đa thức
$O(n\log n)$	Độ phức tạp $n\log n$
$O(b^n), b>1$	Độ phức tạp hàm mũ
$O(n!)$	Độ phức tạp giai thừa

Một số đánh giá thường dùng

- Độ phức tạp thuật toán thường được đưa về một trong các dạng độ phức tạp sau đây

$O(1)$

$O(\log n)$

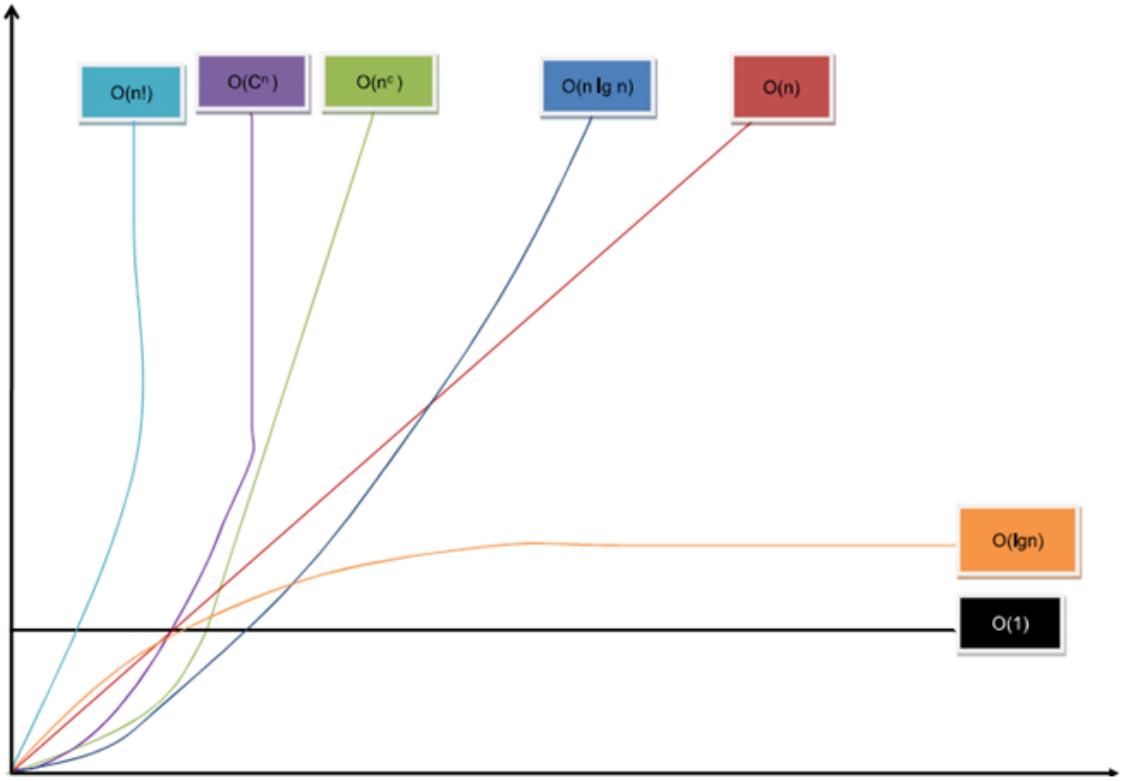
$O(n)$

$O(n^k)$

$O(n \log n)$

$O(b^n), b>1$

$O(n!)$



Lớp P và NP

a. *Một thuật toán được gọi là có độ phức tạp đa thức, hay còn gọi là có thời gian đa thức, nếu số các phép tính cần thiết khi thực hiện thuật toán không vượt quá* $O(n^k)$, *với k nguyên dương nào đó, còn n là kích thước của dữ liệu đầu vào.*
b. *Các thuật toán với* $O(b^n)$, *trong đó n là kích thước dữ liệu đầu vào, còn k là một số nguyên dương nào đó gọi là các thuật toán có độ phức tạp hàm mũ hoặc thời gian mũ.*

Lớp P và NP …

- P (Polynomial time) là lớp các bài toán giải được với thời gian đa thức.
- NP (Nondeterministic Polynomial time) là lớp các bài toán mà lời giải của nó có thể kiểm tra với thời gian đa thức.

Bài toán P so với NP

- Bài toán **P so với NP** là một bài toán mở quan trọng trong lĩnh vực khoa học máy tính.
- Nó đặt ra vấn đề là: bất kì bài toán nào có lời giải có thể được kiểm chứng với thời gian đa thức (lớp NP) cũng có thể được giải quyết với thời gian đa thức (lớp P) hay không? hay

P = NP ???

P = NP ???

- Được Stephen Cook đưa ra năm 1971 trong bài báo nổi tiếng "The complexity of theorem proving procedures”
- Là một trong số bảy bài toán của giải thiên niên kỷ được chọn bởi Viện Toán học Clay.
- Mỗi bài trong số bảy bài này có giải thưởng US$1,000,000 cho lời giải đúng đầu tiên.

Bài toán tổng tập hợp con

- Cho một tập hợp các số nguyên, tìm một tập hợp con khác rỗng có tổng bằng 0. Ví dụ, có tập hợp con nào của {−2, −3, 15, 14, 7, −10} có tổng bằng 0?
  - Lời giải "có, vì {−2, −3, −10, 15} có tổng bằng 0" có thể được kiểm chứng dễ dàng bằng cách cộng các số đó lại.
  - Tuy nhiên, hiện chưa có thuật toán nào để tìm ra một tập hợp như thế trong thời gian đa thức.

Bài toán tổng tập hợp con

- Cho một tập hợp các số nguyên, tìm một tập hợp con khác rỗng có tổng bằng 0. Ví dụ, có tập hợp con nào của {−2, −3, 15, 14, 7, −10} có tổng bằng 0?
  - Có một thuật toán đơn giản thực thi trong thời gian hàm mũ là kiểm tra tất cả 2^n-1 tập hợp con khác rỗng.
  - Bài toán này nằm trong **NP** (kiểm chứng nhanh chóng) nhưng chưa biết có nằm trong **P** (giải nhanh chóng) hay không

NP-Đầy đủ (NP-Complete)

- Bài toán A $\in$ NP được gọi là NP-đầy đủ nếu thỏa mãn tính chất: A giải được với thời gian đa thức thì mọi bài toán khác trong NP giải được bằng thời gian đa thức.
- Khi đó ta có **P = NP**

Một số bài toán NP-Đầy đủ

- Bài toán xếp ba lô
- Bài toán người bán hàng
- Chu trình Hamilton
- Bài toán tô màu đồ thị
- …

Bài tập

1. Nêu định nghĩa, tính chất và các cách thức biểu diễn thuật toán.
2. Cho các bài toán sau:
   a) Tính nghiệm phương trình bậc 2: $ax^2+bx+c=0$, $a\neq0$.
   b) Tính tổng bình phương của *n* số tự nhiên đầu tiên.
   c) Tìm số có giá trị *x* trong dãy $x_1, x_2,\ldots,x_n$.
   d) Tìm số có giá trị lớn nhất trong dãy $x_1, x_2,\ldots,x_n$.

   Hãy tìm thuật toán để giải bài toán trên, mô tả các thuật toán sử dụng ngôn ngữ tự nhiên và chỉ ra các tính chất của thuật toán đó.

Bài tập

3. Mô tả các thuật toán trong bài 2 dạng sơ đồ khối.

4. Mô tả các thuật toán trong bài 2 dạng giả mã