CHỦ ĐỀ
PHÂN TÍCH THUẬT TOÁN

GVHD : THS. Nguyễn Thanh Sơn

LỚP: CS112.L21.KHCL

NHÓM 4: Phạm Anh Khoa 19521699
Lê Quang Huy 19521617
Nguyễn Trần Phước Lộc 19521764

NỘI DUNG

PHẦN I: KHÁI NIỆM PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

PHẦN II: TIỆM CẬN VÀ CÁC KÝ HIỆU

PHẦN III: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN PHI ĐỆ QUY

PHẦN IV: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN ĐỆ QUY

PHẦN I
KHÁI NIỆM PHÂN TÍCH THUẬT TOÁN

KHÁI NIỆM THUẬT TOÁN

Thuật toán là gì ?

Thuật toán là **tập hợp hữu hạn** các thao tác để giải quyết một **bài toán (vấn đề)** nào đó.

Ví dụ: Tính diện tích chữ nhật

Input : Chiều dài, chiều rộng hình chữ nhật
Output : Diện tích của hình chữ nhật

KHÁI NIỆM PHÂN TÍCH THUẬT TOÁN

I. KHÁI NIỆM

Phân tích thuật toán là xác định độ phức tạp tính toán của thuật toán, đó là lượng thời gian, lượng lưu trữ hoặc các tài nguyên để thực hiện chúng.

II. VAI TRÒ

Nhờ việc phân tích thuật toán, ta có thể dễ dàng nhận ra được thuật toán nào hiệu quả nhất cho bài toán đã cho.

KHÁI NIỆM PHÂN TÍCH THUẬT TOÁN

III. TỈ SUẤT TĂNG

- Kích thước đầu vào n nhỏ thì các giải thuật phần lớn đều có thời gian thực thi như nhau.

- Nhưng với $n \to \infty$ thì sự khác biệt về thời gian thực thi ngày càng rõ.

KHÁI NIỆM PHÂN TÍCH THUẬT TOÁN

III. TỈ SUẤT TĂNG

n	$\log_2 n$	n^2	n^3	2^n
10	3.3	10^2	10^3	1024
10^2	6.6	10^4	10^6	$1.3*10^{30}$
10^3	10	10^6	10^9	$10.7*10^{300}$
10^4	13	10^8	10^{12}	$1.99*10^{3010}$
10^5	17	10^{10}	10^{15}	$9.99*10^{30102}$

PHẦN II
TIỆM CẬN VÀ CÁC KÝ HIỆU

TIỆM CẬN

Tiệm cận (hay Asymptotic Notations) gồm 3 dạng: **Θ,** O**, Ω**

- Ký hiệu **Θ** (Theta) cận sát – Tight Bound
- Ký hiệu O (big-Oh) cận trên – Tight Upper Bound
- Ký hiệu Ω (big-Omega) cận dưới – Tight Lower Bound

Xác định *tập hợp* các hàm: được sử dụng để so sánh kích thước của 2 hàm.

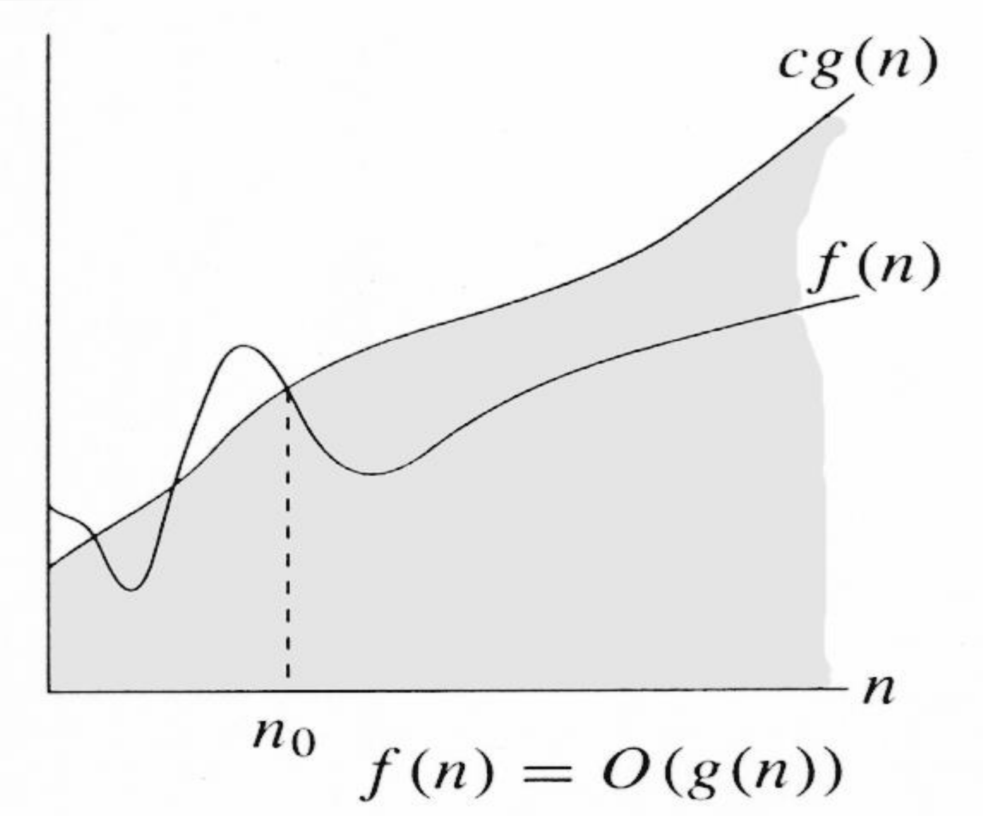
Ký hiệu Big-Oh (O)

Giả sử g: $N \to R^+$, chúng ta định nghĩa Big-Oh (0):

- O(g(n)) là tập của tất cả các hàm có tỉ suất tang thấp hơn hoặc cùng với g(n) (với bội số không đổi và n $\to\infty$)
- Ví dụ:

n € O(n^2) 2n+1 € O(n^2)

Ký hiệu Big-Oh (O)



$f(n) \in O(g(n))$

$\exists\, c > 0, \exists\, n_0 > 0$ và $\forall n \geq n_0, 0 \leq f(n) \leq c.g(n)$

***g*(*n*) là một tiệm cận chặn trên của *f*(*n*).**

Ví dụ

- Với một ví dụ đơn giản, ta có thể thấy rằng hàm $2n^2 + 5n + 6$ là $O(n^2)$.

- Với mọi $n \geq 1$, ta có:

$$2n^2 + 5n + 6 \leq 2n^2 + 5n^2 + 6n^2 = 13n^2$$

- Vì vậy, ta có **$c = 13$ và $n_o = 1$**.

Ví dụ

Chứng minh rằng: $2n^2 = O(n^3)$

Chứng minh:

Giả sử ta có $f(n) = 2n^2$ và $g(n) = n^3$

$f(n) = O(g(n))$?

Bây giờ chúng ta sẽ tìm giá trị của c và n_0

$f(n) \leq c.g(n) \Rightarrow 2n^2 \leq c.n^3 \Rightarrow 2 \leq c.n$

Nếu $c = 1$ và $n_0 = 2$ *hoặc*

$c = 2$ và $n_0 = 1$ *thì*

$2n^2 \leq c.n^3$

Vì vậy, $f(n) = O(g(n))$, $c = 1$ và $n_0 = 2$

Ví dụ

Chứng minh rằng: $n^2 = O(n^2)$

Chứng minh:

Giả sử ta có: $f(n) = n^2$ và $g(n) = n^2$

$f(n) = O(g(n))$?

Bây giờ chúng ta sẽ tìm giá trị c và n_0

$f(n) \leq c.g(n) \Rightarrow n^2 \leq c.n^2 \Rightarrow 1 \leq c$

Nếu $c = 1$, $n_0 = 1$

Thì

$n^2 \leq c.n^2$

Vì vậy, $n^2 = O(n^2)$, $c = 1$ và $n_0 = 1$

Ví dụ

Chứng minh rằng: $10n + 500 = O(n)$

Chứng minh:

Ta có hàm $y = n$ *không thể lớn hơn* hàm $y = 500 + 10n$ với mọi n không âm.

Tuy nhiên, tồn tại giá trị c_0 và n_0 và

$\underline{500 + 10n \leq c.n}$ với $n \geq n_0$.

Với giá trị của $c > 20$ và $n_0 = 50$ thì bất đẳng thức trên đúng.

Vì vậy, 500 + 10n = O(n).

Ví dụ

Chứng minh hoặc phủ nhận $2^{2n} = O(2^n)$?

- Từ giả thiết ta có:
  - $2^{2n} \leq c.2^n$
  - $2^n 2^n \leq c.2^n$
- Bất đẳng thức chỉ đúng khi:
- $c \geq 2^n$,
- Điều này làm cho c không phải là hằng số.
- **Vì vậy $2^{2n} = O(2^n)$ không đúng.**

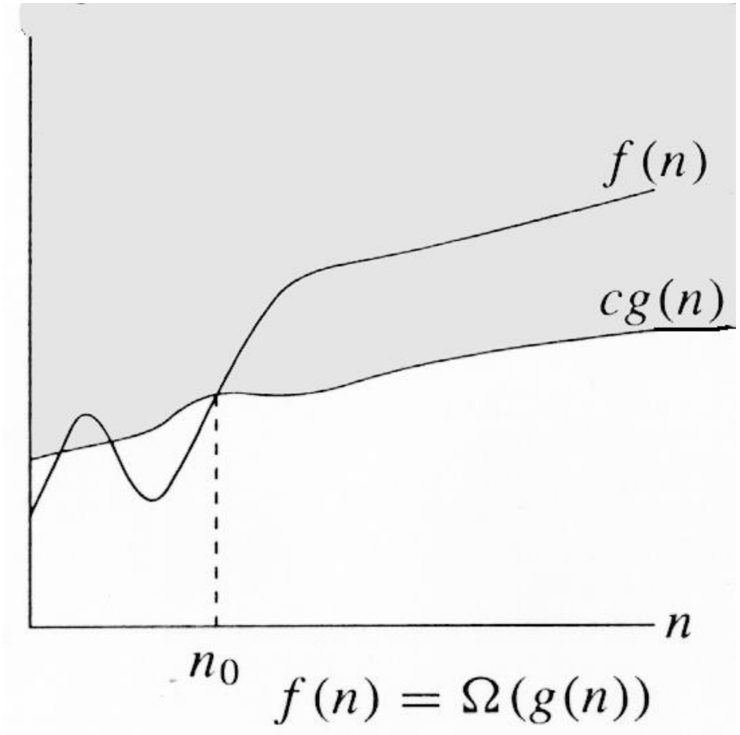
Ký hiệu Big-Omega (Ω)

Với g: $N \rightarrow R^{+}$, chúng ta định nghĩa Big-Omega (Ω)

- Ω(g(n)) là tập của tất cả các hàm có tỉ suất tang cao hơn hoặc bằng với g(n) (với bội số không đổi và $n \rightarrow \infty$)
- Ví dụ:

$50\, n^2 + 10 \in \Omega(n^2)$ $5n^3 \in \Omega(n^2)$

Ký hiệu Big-Omega (Ω)



$f(n) \in \Omega(g(n))$

$\exists\ c > 0, \exists\ n_0 > 0, \forall n \geq n_0, f(n) \geq c.g(n)$

***g*(*n*) được gọi là tiệm cận chặn dưới của *f*(*n*).**

Lưu ý: $t(n) \in \Omega(f(n)) = f(n) \in O(t(n))$

Ví dụ

Chứng minh rằng $5n^2 + 2n - 3 \in \Omega(n^2)$

Chứng minh:

Giả sử $f(n) = 5n^2 + 2n - 3$ và $g(n) = n^2$

$f(n) \in \Omega(g(n))$?

Chúng ta cần xác định giá trị của c và n_0 sao cho:

$c.g(n) \leq f(n) \quad \forall n \geq n_0$

$c.n^2 \leq 5.n^2 + 2n - 3$

Ta có thể lấy c = 5, do đó 2n-3 luôn dương.

Ta có 2n-3 luôn dương với $n \geq 2$. Vì vậy $n_0 = 2$.

Vì vậy $f(n) \in \Omega(g(n))$, với $c = 5$ và $n_0 = 2$

Ví dụ

Chứng minh rằng: $100.n + 5 \notin \Omega(n^2)$

Chứng minh:

Cho $f(n) = 100.n + 5$ và $g(n) = n^2$

Giả sử $f(n) \in \Omega(g(n))$?

Nếu $f(n) \in \Omega(g(n))$ thì tồn tại giá trị c và n_0 sao cho:

$c.g(n) \le f(n)$ với $n \ge n_0$

$c.n^2 \le 100.n + 5$

Để bất đẳng thức trên luôn đúng thì $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ nghĩa là hàm f(n) tăng nhanh hơn hàm g(n).

Nhưng $\lim_{n\to\infty} \frac{100n+5}{n^2} = 0 \neq \infty$ nghĩa là hàm g(n) tăng nhanh hơn hàm f(n)

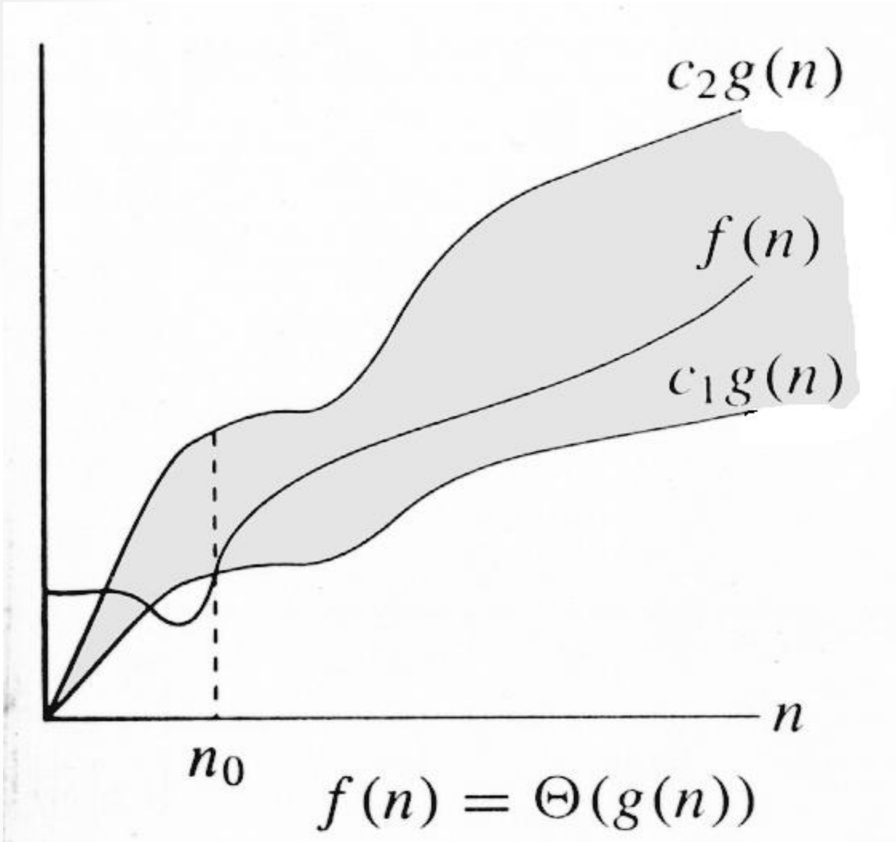
Vì vậy $f(n) \notin \Omega(g(n))$

Ký hiệu Theta (Θ)

Với g: N → R^+, chúng ta định nghĩa Theta (Θ)

- Θ(g(n)) là tập của tất cả các hàm có tỉ suất tăng bằng với g(n) (với bội số không đổi và n →∞)
- Vì vậy, mọi hàm bậc 2 "$xn^2 + yn + z$", với mọi x > 0 đều thuộc Θ(n^2)

Ký hiệu Theta (Θ)



$f(n) \in \Theta(g(n))$

$\exists\ c_1 > 0,\ c_2 > 0,\ \exists\ n_0 > 0,\ \forall\ n \geq n_0,\ c_2.g(n) \leq f(n) \leq c_1.g(n)$

***g*(*n*) được gọi là cận sát theo tiệm cận của *f*(*n*).**

Ví dụ

Chứng minh rằng: $\frac{1}{2}.n^2 - \frac{1}{2}.n = \Theta(n^2)$

Chứng minh:

$f(n) = \frac{1}{2}.n^2 - \frac{1}{2}.n$, and $g(n) = n^2$

$f(n) \in \Theta(g(n))$?

Chúng ta cần tìm giá trị của c_1, c_2 và n_0 sao cho"

$c_1.g(n) \leq f(n) \leq c_2.g(n)$ với $n \geq n_0$

Bởi $\frac{1}{2} n^2 - \frac{1}{2} n \leq \frac{1}{2} n^2$ $(\forall n \geq 0)$ với $c_2 = \frac{1}{2}$ và

$\frac{1}{2} n^2 - \frac{1}{2} n \geq \frac{1}{2} n^2 - \frac{1}{2} n \,.\, \frac{1}{2} n = \frac{1}{4} n^2$ $(\forall n \geq 2)$ với $c_1 = \frac{1}{4}$

Vì vậy $\frac{1}{2} n^2 - \frac{1}{2} n \leq \frac{1}{2} n^2 \leq \frac{1}{2} n^2 - \frac{1}{2} n$

$c_1.g(n) \leq f(n) \leq c_2.g(n)$ $\forall n \geq 2, c_1 = \frac{1}{4}, c_2 = \frac{1}{2}$

Vì vậy $f(n) \in \Theta(g(n)) \Rightarrow \frac{1}{2}.n^2 - \frac{1}{2}.n = \Theta(n^2)$

Ví dụ

Chứng minh rằng: $2.n^2 + 3.n + 6 \notin \Theta(n^3)$

Chứng minh: Cho $f(n) = 2.n^2 + 3.n + 6$ và $g(n) = n^3$

$f(n) = \Theta(g(n))$

Ngược lại giả sử $f(n) \in \Theta(g(n))$ nghĩa là tồn tại các hằng số dương c_1, c_2 và n_0 sao cho:

$c_1.g(n) \leq f(n) \leq c_2.g(n)$

Với c_2:

$f(n) \leq c_2.g(n) \Rightarrow 2n^2 + 3n + 6 \leq 2n^2 + 3n^2 + 6n^2 \leq c_2n^3 \Rightarrow c{=}11$và $n_0 = 1$

Với c_1:

$c_1.g(n) \leq f(n) \Rightarrow c_1n^3 \leq 2n^2 + 3n + 6 \Rightarrow c_1n^3 \leq 2n^2 \leq 2n^2 + 3n + 6$

$c_1.n \leq 2$, điều này là không thể với giá trị của n lớn.

Vì vậy $f(n) = \Theta(g(n)) \Rightarrow 2.n^2 + 3.n + 6 \notin \Theta(n^3)$

Ví dụ

Chứng minh rằng: $3.n + 2 = \Theta(n)$

Chứng minh: Let $f(n) = 3.n + 2$, and $g(n) = n$

Giả sử rằng: $f(n) \in \Theta(g(n))$ nghĩa là tồn tại các hằng số dương c_1, c_2 và n_0 sao cho:

$$c_1.g(n) \le f(n) \le c_2.g(n)$$

$$\Rightarrow c_1.n \le 3.n + 2 \le c_2.\, n$$

Ta có $c_1 = 3$ thì $3n \le 3n + 2$ với $n_0 = 1$

$3n + 2 \le 3n + 2n \le c_2.\, n \Rightarrow 5n \le c_2.\, n \Rightarrow 5 \le c_2$

$c_1 = 3$, $c_2 = 5$, $n_0 = 1$

Vì vậy $f(n) = \Theta(g(n)) \Rightarrow 3.n + 2 = \Theta(n)$

PHẦN III
PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN PHI ĐỆ QUY

CÁC BƯỚC THỰC HIỆN

BƯỚC 1: Xác định tham số đại diện kích thước đầu vào.

BƯỚC 2: Nhận dạng được **BASIC OPERATION** của thuật toán. (Thông thường thì nó sẽ nằm ở vòng trong cùng.)

BƯỚC 3: Kiểm tra xem số lần thực thi **BASIC OPERATION** của thuật toán có phụ thuộc vào tham số nào khác hay không. Nếu có thì các trường hợp worst-case, average-case phải được tính toán riêng biệt.

BƯỚC 4: Lập một công thức tính tổng số lần **BASIC OPERATION** của thuật toán được thực thi.

BƯỚC 5: Sử dụng các kiến thức đã học về thao tác tính tổng để đơn giản hóa công thức, sau đó ước tính sự gia tăng về mặt thời gian của thuật toán khi kích thước tăng lên.

Ví dụ

ALGORITHM *MaxElement*($A[0..n-1]$)

    //Determines the value of the largest element in a given array
    //Input: An array $A[0..n-1]$ of real numbers
    //Output: The value of the largest element in A
    $maxval \leftarrow A[0]$
    for $i \leftarrow 1$ **to** $n-1$ **do**
        if $A[i] > maxval$
            $maxval \leftarrow A[i]$
    return $maxval$

Đoạn mã giả trên mô phỏng thuật toán tìm phần tử lớn nhất trong mảng bằng phương pháp dò tuyến tính. Phân tích độ phức tạp của thuật toán trên.

```
ALGORITHM MaxElement(A[0..n − 1])
    //Determines the value of the largest element in a given array
    //Input: An array A[0..n − 1] of real numbers
    //Output: The value of the largest element in A
    maxval ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval
```

Bước 1: Xác định tham số quyết định kích thước đầu vào.

Khá đơn giản, kích thước đầu vào là số lượng phần tử trong mảng và được đại diện bởi tham số n.

ALGORITHM $MaxElement(A[0..n-1])$

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

Bước 2: Xác định "BASIC OPERATION" của thuật toán.

Có 2 operation được thực thi trong vòng lặp:
+ Phép so sánh A[i] > maxval.
+ Phép gán maxval := A[i].

Dễ nhận thấy rằng Phép so sánh A[i] > maxval được thực hiện ở mỗi lần lặp còn phép gán thì không.
=> **"BASIC OPERATION"** của thuật toán là phép so sánh.

```
ALGORITHM MaxElement(A[0..n − 1])
    //Determines the value of the largest element in a given array
    //Input: An array A[0..n − 1] of real numbers
    //Output: The value of the largest element in A
    maxval ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval
```

Bước 3: Kiểm tra xem số lần thực thi BASIC OPERATION của thuật toán

Số lần so sánh chỉ phụ thuộc vào **tham số n** - kích thước đầu vào, vì thế trong trường hợp này ta không cần chia best-case, average-case hay là worst-case.

```
ALGORITHM MaxElement(A[0..n − 1])
    //Determines the value of the largest element in a given array
    //Input: An array A[0..n − 1] of real numbers
    //Output: The value of the largest element in A
    maxval ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval
```

Bước 4: Thiết lập công thức tính số lần "BASIC OPERATION" của thuật toán được thực hiện.

Gọi C(n) là số lần phép so sánh được thực thi, ta có công thức :

$$\boxed{C(n) = \sum_{i=1}^{n-1} 1.}$$

PHẦN III: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN PHI ĐỆ QUY

ALGORITHM $MaxElement(A[0..n-1])$

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

Bước 5: Đơn giản hóa công thức và ước tính.

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

PHẦN III: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN PHI ĐỆ QUY

Ví dụ

```
ALGORITHM   UniqueElements(A[0..n − 1])
    //Determines whether all the elements in a given array are distinct
    //Input: An array A[0..n − 1]
    //Output: Returns "true" if all the elements in A are distinct
    //          and "false" otherwise
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i] = A[j] return false
    return true
```

Đoạn mã giả trên mô phỏng thuật toán kiểm tra tất cả các phần tử trong mảng có phải là duy nhất trong mảng hay không. Phân tích độ phức tạp của thuật toán trên.

```
ALGORITHM UniqueElements(A[0..n − 1])
    //Determines whether all the elements in a given array are distinct
    //Input: An array A[0..n − 1]
    //Output: Returns "true" if all the elements in A are distinct
    //            and "false" otherwise
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i] = A[j] return false
    return true
```

Bước 1: Xác định tham số quyết định kích thước đầu vào.

Khá đơn giản, kích thước đầu vào là số lượng phần tử trong mảng và được đại diện bởi tham số n.

ALGORITHM *UniqueElements*($A[0..n-1]$)

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
// and "false" otherwise

for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    if $A[i] = A[j]$ **return false**
return true

Bước 2: Xác định "BASIC OPERATION" của thuật toán.

Có 1 operation được thực thi trong vòng lặp:
+ Phép so sánh A[i] = A[j].

Dễ nhận thấy rằng Phép so sánh A[i] = A[j] là phép toán duy nhất

=> **"BASIC OPERATION"** của thuật toán là phép so sánh.

PHẦN III: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN PHI ĐỆ QUY

```
ALGORITHM  UniqueElements(A[0..n − 1])
    //Determines whether all the elements in a given array are distinct
    //Input: An array A[0..n − 1]
    //Output: Returns "true" if all the elements in A are distinct
    //          and "false" otherwise
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i] = A[j] return false
    return true
```

Bước 3: Kiểm tra xem "BASIC OPERATION" của thuật toán còn phụ thuộc vào yếu tố nào khác hay không.

Trong trường hợp này , ngoài kích thước của đầu vào của input **"BASIC OPERATION"** còn phụ thuộc vào việc có tồn tại phần tử xuất hiện nhiều lần trong mảng hay không. Vì thế chúng ta cần tách ra các trường hợp worst-case, best-case và average-case.

Ở đây nhóm sẽ quan tâm đến worst-case .

ALGORITHM *UniqueElements*$(A[0..n-1])$

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
//   and "false" otherwise
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] = A[j]$ **return false**
return true

Bước 4: Thiết lập công thức tính số lần "BASIC OPERATION" của thuật toán được thực hiện.

Gọi C_worst (n) là số lần phép so sánh được thực thi, ta có công thức :

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

PHẦN III: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN PHI ĐỆ QUY

ALGORITHM *UniqueElements*$(A[0..n-1])$

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
// and "false" otherwise
for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    if $A[i] = A[j]$ **return false**
return true

Bước 5: Đơn giản hóa công thức và ước tính.

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$
$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$
$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).$$

Trong trường hợp tệ nhất, thuật toán sẽ thực hiện phép so sánh (n-1)*n/2 lần.
=> Trong trường hợp tệ nhất, thuật toán có độ phức tạp là **Θ(n^2).**

PHẦN III: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN PHI ĐỆ QUY

Ví dụ

ALGORITHM *Binary*(n)

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
$count \leftarrow 1$
while $n > 1$ **do**
    $count \leftarrow count + 1$
    $n \leftarrow \lfloor n/2 \rfloor$
return $count$

Đoạn mã giả trên mô phỏng thuật toán tính số lượng bit nhị phân cần để biểu diễn số thập phân nguyên dương n.

ALGORITHM *Binary*(n)

```
//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
count ← 1
while n > 1 do
    count ← count + 1
    n ← ⌊n/2⌋
return count
```

Basic Operation ở đây thay vì ở trong vòng lặp thì nó lại là phép so sánh n>1. Một điều cần phải để ý nữa là vòng lặp này không lặp liên tục từ 1 -> n. Nên chúng ta cần một cách tính khác để tính số lần Basic Operation được thực thi.

Vì giá trị của n giảm một nửa sau mỗi lần lặp nên tổng số lần phép so sánh được thực thi là $\lfloor \log_2 n \rfloor + 1$.

Và chúng ta cũng có được kết quả này khi **phân tích thuật toán đệ quy.**

Bài tập

Đề bài: Tìm số cách để một số nguyên X có thể biểu diễn được dưới tổng các lũy thừa N^{th} của các số tự nhiên.

Input : số nguyên n là số phần tử mảng và k
Output: có bao nhiêu cặp mà tổng chia hết cho 3
VD :
Input : 5 3
1 3 5 6 8
Output: 3
Giải thích: Các cặp số chia hết cho 3 là (1,5) , (1,8) , (3,6)

PHẦN IV
PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN ĐỆ QUY

CÁC BƯỚC THỰC HIỆN

BƯỚC 1: Xác định tham số đại diện kích thước đầu vào.

BƯỚC 2: Nhận dạng được **BASIC OPERATION** của thuật toán. (Thông thường thì nó sẽ nằm ở vòng trong cùng.)

BƯỚC 3: Kiểm tra xem số lần thực thi **BASIC OPERATION** của thuật toán có phụ thuộc vào tham số nào khác hay không. Nếu có thì các trường hợp worst-case, average-case phải được tính toán riêng biệt.

BƯỚC 4: Thiết lập hàm đệ quy có điều kiện dừng thích hợp để tính số lần **BASIC OPERATION** được thực thi.

BƯỚC 5: Giải phương trình đệ quy, hoặc ít nhất là ước tính sự gia tăng về mặt thời gian của thuật toán khi kích thước tăng lên.

Ví dụ

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Đoạn mã giả trên mô tả thuật toán tính giai thừa bằng thủ tục đệ quy.

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Bước 1: Xác định tham số quyết định kích thước đầu vào.

Khá là dễ quyết định : n.

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Bước 2: Xác định "BASIC OPERATION" của thuật toán.

Có 1 operation được thực thi trong vòng lặp:
+ Phép nhân F(n-1) * n.

Dễ nhận thấy rằng Phép nhân F(n-1) * n là phép toán duy nhất

=> **"BASIC OPERATION"** của thuật toán là phép Nhân

PHẦN IV: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN ĐỆ QUY

ALGORITHM $F(n)$

```
//Computes n! recursively
//Input: A nonnegative integer n
//Output: The value of n!
if n = 0 return 1
else return F(n − 1) ∗ n
```

Bước 3: Kiểm tra xem "BASIC OPERATION" của thuật toán còn phụ thuộc vào yếu tố nào khác hay không.

Ta thấy thuật toán không phụ thuộc vào yếu tố nào khác.

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Bước 4: Thiết lập hàm đệ quy có điều kiện dừng thích hợp để tính số lần BASIC OPERATION được thực thi.

Gọi M(n) là số lần mà Basic Operation được thực thi trong hàm F(n).

Vì F(n) = F(n-1) * n, n > 0.
Ta có công thức đệ quy tính M(n) như sau.

$$M(n) = \underbrace{M(n-1)}_{\substack{\text{to compute}\\ F(n-1)}} + \underbrace{1}_{\substack{\text{to multiply}\\ F(n-1)\text{ by } n}} \quad \text{for } n > 0.$$

Việc cần làm tiếp theo là xác định Một điều kiện dừng - Một giá trị khởi đầu của M(n). Để xác định được giá trị ban đầu của M(n) ta cần dựa vào điều kiện dừng của F(n) :

if $n = 0$ **return** 1.

Điều kiện này cho ta 2 thông tin.
+ Thứ nhất: với n = 0 thì F(n) dừng lại, vậy giá trị n nhỏ nhất của M(n) là 0.
+ Thứ hai: với n = 0 thì F(n) không thực thi Basic Operation

Từ đó ta có kết luận:

$$M(n) = M(n-1) + 1 \quad \text{for } n > 0,$$
$$M(0) = 0.$$

PHẦN IV: PHƯƠNG PHÁP ƯỚC LƯỢNG ĐỘ PHỨC TẠP THỜI GIAN CHO BÀI TOÁN ĐỆ QUY

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Bước 5: Giải phương trình đệ quy, hoặc ít nhất là ước tính sự gia tăng về mặt thời gian của thuật toán khi kích thước tăng lên.

Trong bài toán này cách giải khá đơn giản :

$$\begin{aligned} M(n) &= M(n-1)+1 && \text{substitute } M(n-1)=M(n-2)+1 \\ &= [M(n-2)+1]+1 = M(n-2)+2 && \text{substitute } M(n-2)=M(n-3)+1 \\ &= [M(n-3)+1]+2 = M(n-3)+3. \end{aligned}$$

Ta có kết luận :

$$M(n) = M(n-1)+1 = \cdots = M(n-i)+i = \cdots = M(n-n)+n = n.$$

M(n) = n => Độ phức tạp của thuật toán là Θ(n^2).

Ví dụ

ALGORITHM *BinRec*(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return $BinRec(\lfloor n/2 \rfloor) + 1$

Đoạn mã trên mô phỏng thuật toán tính số lượng bit nhị phân cần để biểu diễn số thập phân nguyên dương n bằng phương pháp đệ quy.

ALGORITHM *BinRec(n)*

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return $BinRec(\lfloor n/2 \rfloor) + 1$

Bước 1: Xác định tham số quyết định kích thước đầu vào.

k - Số lượng bit vừa đủ để biểu diễn giá trị n trong hệ nhị phân.

ALGORITHM $BinRec(n)$

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
if $n = 1$ **return** 1
else return $BinRec(\lfloor n/2 \rfloor) + 1$

Bước 2: Xác định "BASIC OPERATION" của thuật toán.

Phép cộng: $\boxed{BinRec(\lfloor n/2 \rfloor) + 1}$

=> **"BASIC OPERATION"** của thuật toán là phép Cộng.

ALGORITHM *BinRec*(n)

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
if $n = 1$ **return** 1
else return $BinRec(\lfloor n/2 \rfloor) + 1$

Bước 3: Kiểm tra xem "BASIC OPERATION" của thuật toán còn phụ thuộc vào yếu tố nào khác hay không.

Ta thấy thuật toán không phụ thuộc vào yếu tố nào khác.

ALGORITHM $BinRec(n)$

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return $BinRec(\lfloor n/2 \rfloor) + 1$

Bước 4 : Thiết lập hàm đệ quy có điều kiện dừng thích hợp để tính số lần BASIC OPERATION được thực thi.

Gọi A(n) là số lần phép cộng được thực thi trong hàm BinRec.

Ta có công thức :

$$\boxed{A(n) = A(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1.}$$

Dựa vào điều kiện dừng của hàm BinRec :

$$\boxed{\textbf{if } n = 1 \textbf{ return } 1}$$

Ta xác định được giá trị khởi đầu của A(n): A(1) = 0;

ALGORITHM *BinRec(n)*

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
if $n = 1$ **return** 1
else return $BinRec(\lfloor n/2 \rfloor) + 1$

Bước 5: Giải phương trình đệ quy, hoặc ít nhất là ước tính sự gia tăng về mặt thời gian của thuật toán khi kích thước tăng lên.

Do công thức **[n/2]** xuất hiện trong chương trình, thế nên phương pháp thay thế lùi ở ví dụ 1 không còn hiệu quả khi áp dụng với n không là lũy thừa của 2.

Vì thế ta sẽ giải phương trình A(n) với n là lũy thừa của 2 trước, sau đó áp dụng **smoothness rule** để kết luận với mọi n.

Với n = 2^k, k > 0. Ta có :

$$A(2^k) = A(2^{k-1}) + 1 \quad \text{for } k > 0,$$
$$A(2^0) = 0.$$

Áp dụng kĩ thuật thay thế lùi vào phương trình đã cho, ta có :

$$\begin{aligned}
A(2^k) &= A(2^{k-1}) + 1 && \text{substitute } A(2^{k-1}) = A(2^{k-2}) + 1\\
&= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2 && \text{substitute } A(2^{k-2}) = A(2^{k-3}) + 1\\
&= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3 && \dots\\
&\dots\\
&= A(2^{k-i}) + i\\
&\dots\\
&= A(2^{k-k}) + k.
\end{aligned}$$

Sau khi viết gọn lại, ta được :

$$\boxed{A(2^k) = A(1) + k = k,}$$

Vì n = 2^k => k = log2(n) nên ta có:

$$\boxed{A(n) = \log_2 n \in \Theta(\log n).}$$

Kết hợp với **smoothness rule**, ta có thể kết luận :

Hàm BinRec có độ phức tạp Θ(log2(n))

Bài tập

Đề bài: Cho mảng arr gồm các số nguyên dương. Xác định số lượng cặp (a,b) (trong đó a < b) sao cho arr[a]+arr[b] chia hết cho số nguyên k cho trước.

Input : số nguyên X và N(2<=N<=10)
Output: có bao nhiêu cách kết hợp lũy thừa mũ N để tổng bằng X

VD: Input : 10 2
Output : 1

Giải thích: 10 = 1^2+3^2

BÀI TẬP VỀ NHÀ

BÀI 1:

Tìm siêu số của của số nguyên X(super digit)
*Nếu chỉ có 1 chữ số thì số đó là super digit
*Ngược lại super digit X bằng super digit của tổng các chữ số của X
Vd: super_digit(5789)   5+7+8+9 = 29
super_digit(29)   2+9 = 11
super_digit(11)   1+1 = 2

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN TRÊN LÀ BAO NHIÊU ?

BÀI TẬP VỀ NHÀ

BÀI 2:

```
s = 0;
for (i=0; i<=n;i++){
        p =1;
        for (j=1;j<=i;j++)
                p=p * x / j;
        s = s+p;
}
```

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN TRÊN LÀ BAO NHIÊU ?

BÀI TẬP VỀ NHÀ

BÀI 3:

```
for (i= 1;i<=n;i++)
        for (j= 1;j<=n;j++)
                A[i]=A[j];
```

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN TRÊN LÀ BAO NHIÊU ?

TÀI LIỆU THAM KHẢO

- **Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition by Thomas H. Cormen**
- **Introduction to the Design and Analysis of Algorithms, 3rd Edition by Anany Levitin, Villanova University**