ĐỘ PHỨC TẠP THUẬT TOÁN KHÔNG ĐỆ QUY

1. Thuật toán

Thuật toán:

ĐN1: Một dãy hữu hạn các chỉ thị có thể thi hành để đạt mục tiêu đề ra nào đó.

ĐN2: Một chuỗi các bước tính toán được định nghĩa rõ ràng để giải quyết một vấn đề

Đặc điểm:

* Nhận vào một tập các giá trị đầu vào
* Trả về một tập các giá trị đầu ra
* Biểu diễn bằng: mã nguồn, mã giả

Tính chất:

* Tính đúng đắn (cần thiết): kết quả trả về phản ánh đúng mong muốn của thông tin nhận vào
* Tính hiệu quả (quan trọng): độ tin cậy, tốc độ xử lý, tài nguyên sử dụng

Một thuật toán hiệu quả: Chi phí cần sử dụng tài nguyên thấp: Bộ nhớ, thời gian sử dụng CPU, …

2. Độ phức tạp

Hai phương pháp đánh giá độ phức tạp của thuật toán:

* Phương pháp thực nghiệm:
* Cài thuật toán rồi chọn các bộ dữ liệu thử nghiệm.
* Thống kê các thông số nhận được khi chạy các bộ dữ liệu đó.
* *Ưu điểm*: Dễ thực hiện.
* *Nhược điểm*:
* Chịu sự hạn chế của ngôn ngữ lập trình.
* Ảnh hưởng bởi trình độ của người lập trình.
* Chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: khó khăn và tốn nhiều chi phí.
* Phụ thuộc vào phần cứng.
* Phương pháp xấp xỉ toán học:
* Đánh giá giá thuật toán theo hướng tiệm xấp xỉ tiệm cận qua các khái niệm O().
* *Ưu điểm*: Ít phụ thuộc môi trường cũng như phần cứng hơn.
* *Nhược điểm*: Phức tạp.

Đánh Giá:

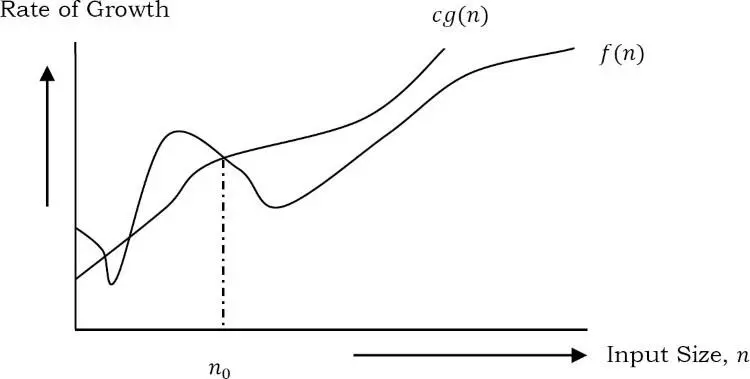
* Không gian: dung lượng bộ nhớ sử dụng
* Thời gian: thời gian chạy của thuật toán
  + Đánh giá thời gian chạy thuật toán
    - *T*(n) = số lượng phép toán sơ cấp cần thực hiện (số học, logic, so sánh)
    - Mỗi phép toán sơ cấp thực hiện trong khoảng thời gian cố định
    - Chỉ quan tâm với tốc độ tăng của hàm *T*(n)
    - Ví dụ: *T*(n) =
  + Độ Tăng Của Hàm
    - Với n là độ lớn dữ liệu đầu vào
    - Tỷ lệ tăng trưởng (chính xác):
  + Bậc tăng trưởng (xấp xỉ):
    - => bậc
    - => bậc
    - => bậc

3. Các ký hiệu tiệm cận

Ký hiệu Big-O

Ký hiệu Big-O cung cấp **upper bound** (giới hạn/tiệm cận trên) của hàm đã cho có thể được định nghĩa là **O(g(n)) = f(n): tồn tại các hằng số dương c và *n0* sao cho 0 ≤ f(n) ≤ c \* g(n) với mọi n ≥ *n0*.**

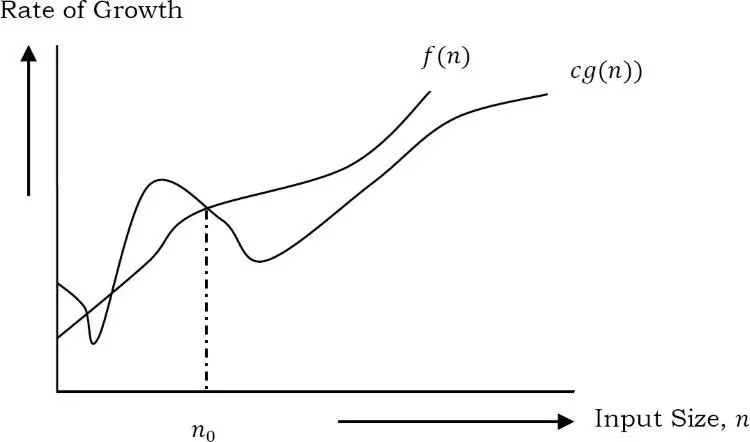
Nghĩa là, với các số n có giá trị lớn, tiệm cận trên của f(n) là g(n).



Kí hiệu Omega-Ω

Kí hiệu Ω cung cấp lower bound (tiệm cận dưới) của một thuật toán, có thể được định nghĩa là **Ω(g(n)) = f(n): tồn tại các hằng số dương c và *n0* sao cho 0 ≤ cg(n) ≤ f(n) với mọi n ≥ *n0*.**

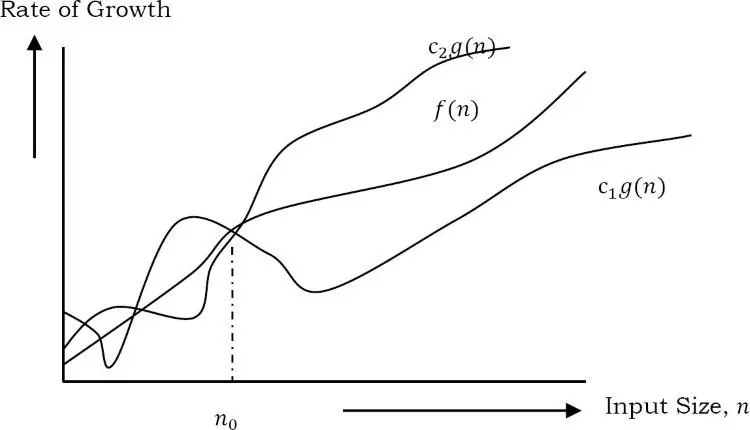
Nghĩa là, với các số n có giá trị lớn, cận dưới của f(n) là g(n).



Theta-Θ Notation

Ký hiệu Θ (giúp quyết định xem upper và lower bounds của một hàm (thuật toán) đã cho có giống nhau hay không) được định nghĩa là **Θ(g(n)) = f(n): tồn tại *c1*, *c2* và *n0* sao cho 0 ≤ *c1* \* g(n) ≤ f(n) ≤ *c2* \* g(n) với mọi n ≥ *n0*.**

Thời gian chạy trung bình của một thuật toán luôn nằm giữa giới hạn dưới và giới hạn trên.



Tổng kết

Để phân tích (best case, worst case and average), chúng ta đã có upper bound (O), lower bound (Ω) và average running time (Θ).  
Chúng ta sẽ **tập trung vào upper bound (O)** vì biết lower bound (Ω) của một thuật toán không có tầm quan trọng trong thực tế, **chúng ta sẽ sử dụng ký hiệu Θ nếu upper bound (O) và lower bound (Ω) là giống nhau.**

Sử dụng kí hiệu Big-O

Hằng số: O(c)

logN: O(logN)

N: O(N)

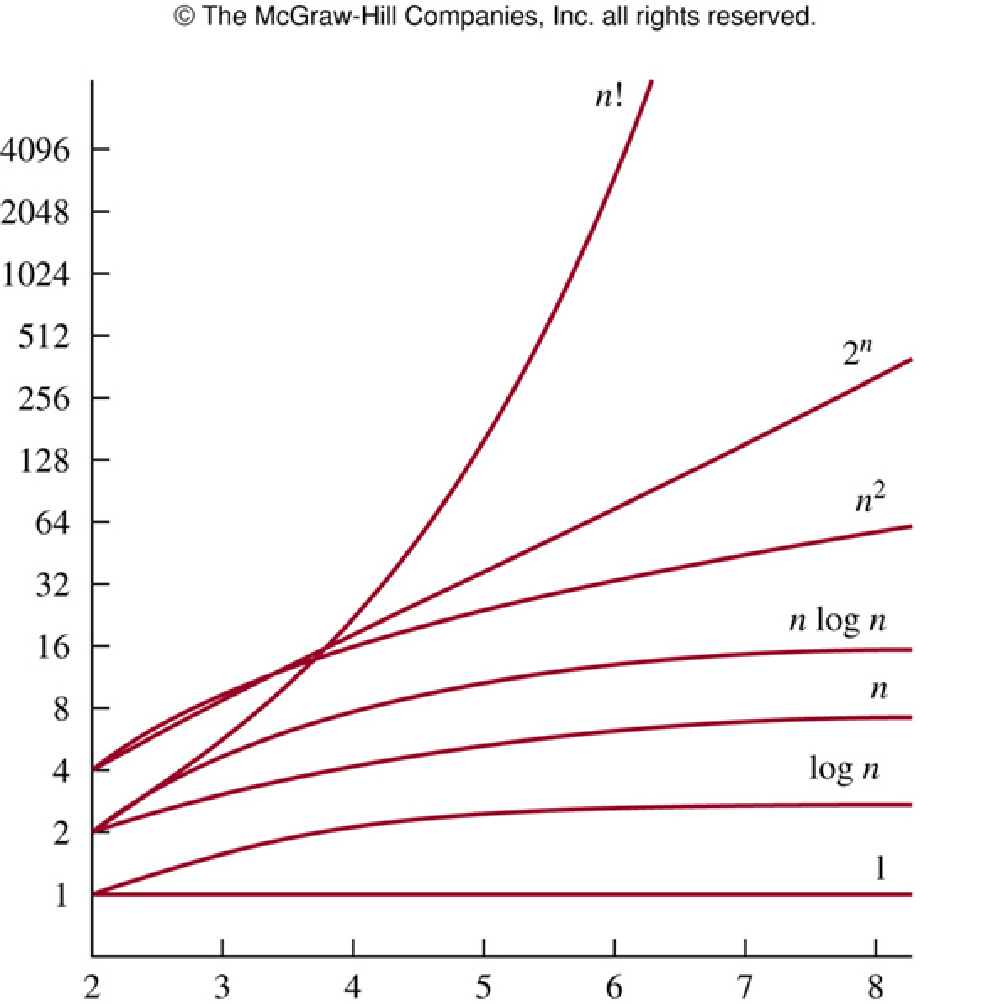
NlogN: O(NlogN)

N2: O(N2)

N3: O(N3)

2N: O(2N)

N!: O(N!)



4. Kỹ Thuật Đánh Giá Thời Gian Chạy

* + Lệnh gán

X=<biểu thức>

Thời gian chạy của lệnh gán bằng thời gian thực hiện biểu thức

* + Lệnh lựa chọn

if (điều kiện) *T0*(n)

lệnh 1 *T1*(n)

else

lệnh 2 *T2*(n)

Thời gian: *T0*(n) + min(*T1*(n), *T2*(n))

* + Lệnh lặp: for, while, do-while

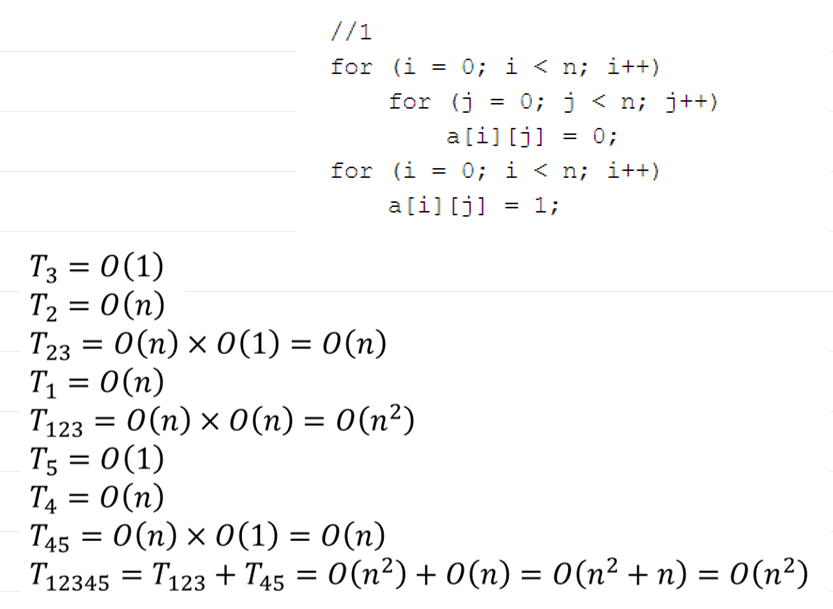
*X*(n): số vòng lặp

*T0*(n): điều kiện lặp

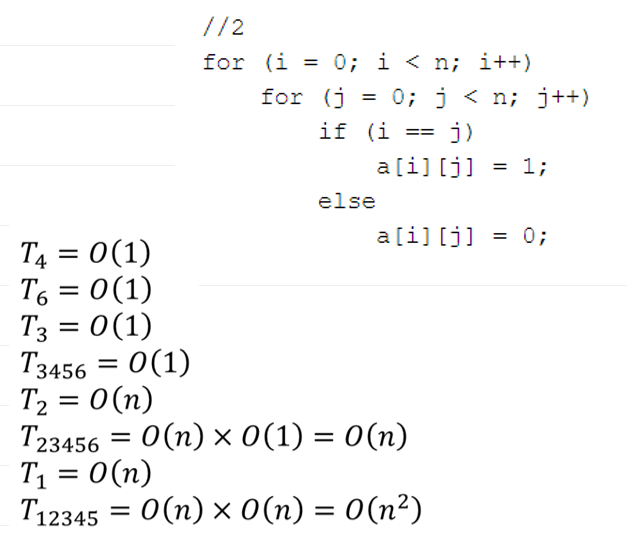
*Ti*(n): thời gian thực hiện vòng lặp thứ i

Áp dụng 1: Đếm xấp xỉ

Ví dụ 1:



Ví dụ 2:



Áp dụng 2: Đếm chính xác

Đếm chính xác từng dòng lệnh để có được hàm số phép gán (Gan(n)) và số phép so sánh (Sosanh(n)) từ đó tính được T(n) = Gan(n) + Sosanh(n).

**Một số công thức cần nhớ:**

Cấp số cộng

* Cấp số cộng: dãy số thảo mãn điều kiện: 2 phần tử liên tiếp nhau sai khác nhau một hằng số (công sai)
* Số hạng thứ n:

**an = a1 + (n-1)d**

* Tổng của n số hạng đầu:

**Sn = a1 + a2 + … + an = =**

(a1: phần tử đầu, d: công sai )

Cấp số nhân

* Cấp số nhân: dãy số thoả mãn điều kiện: tỷ số của hai phần tử liên tiếp là hằng số (công bội)
* Số hạng thứ n:

an = arn-1

* Tổng các phần tử của cấp số nhân:

(a: phần tử đầu, r: công bội)

**Tổng hữu hạn**

Tính chất:

**Sum Manipulation Rules**

Một số công thức cần nhớ:

**Important Summation Formulas**

**Cận trên, cận dưới**

Một số công thức cần nhớ:

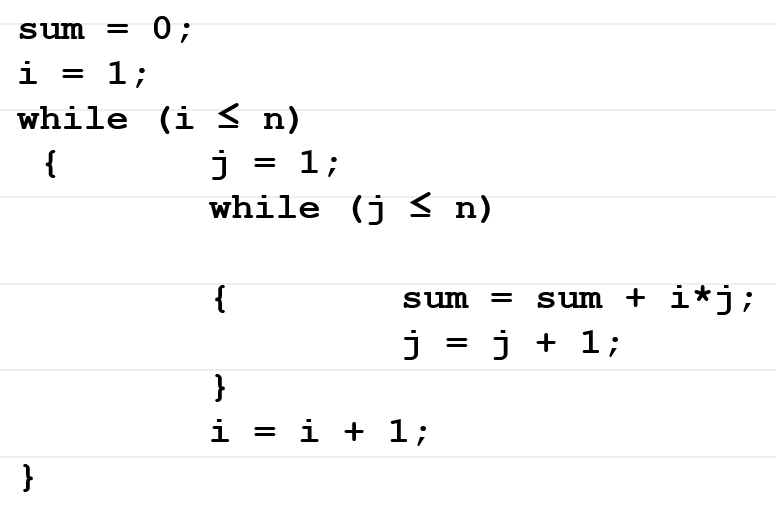
**Floor and Ceiling Formulas**

Floor of a real number x, denoted ⎿x⏌,⎿-3.8⏌=-4, ⎿3.8⏌=3, ⎿3⏌=3

Ceiling of a real number x, denoted ⎾x⏋, ⎾3.8⏋=4 ,⎾-3.8⏋=-3, ⎾3⏋=3

1. x-1 < ⎿x⏌
2. ⎿x+n⏌=⎿x⏌+n and ⎾x+n⏋=⎾x⏋+n for real x and integer n
3. ⎿n/2⏌+⎿n/2⏌=n
4. ⎾lg(n+1)⏋= ⎿lg n⏌+1

**Ví dụ 1:**



**Lời giải:**

sum = 0; {1 g}

i = 1; {1 g}

while (i n) {n+1 ss}

{ j = 1; {n g}

while (j n)

{ sum = sum + i\*j;

j = j + 1; While ngoài lặp n lần

}

i = i +1; {n g}

}

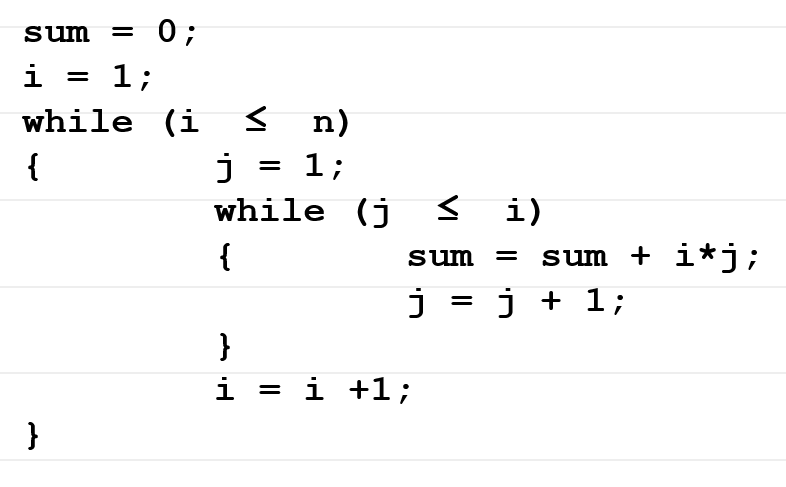
Cứ 1 lần thực hiện while trong sẽ tốn chi phí là 2n phép gán và n+1

phép so sánh

T(n) = Gán (n)+SS(n)

T(n) = 3n2 + 4n + 3

**Ví dụ 2:**

****

**Lời giải:**

sum = 0; {1 g}

i = 1; {1 g}

while (i n) {n+1 ss}

{ j = 1; {n g}

while (j i)

{ sum = sum + i\*j;

j = j + 1;

}

i = i +1; {n g}

}