Chương 1

Khái niệm cơ bản

Nội dung

- Thuật toán/ giải thuật
- Cấu trúc dữ liệu
- Xây dựng và biểu diễn thuật toán
- Độ chính xác
- Tính hiệu quả
- Mô hình RAM
- Mô hình O-lớn
- Bài tập

Khái niệm cơ bản

Thuật toán/ giải thuật

Vấn đề (trong thực tế)	Bài toán (vấn đề giải quyết bằng máy tính)
Một khó khăn cần được giải quyết.	Là một vấn đề cần được giải quyết
Phương án giải quyết vấn đề: là việc tìm ra một giải pháp cho câu hỏi rắc rối, phức tạp, khó hiểu	**Thuật toán/giải thuật**: Là một phương án để giải quyết bài toán (bằng máy tính)

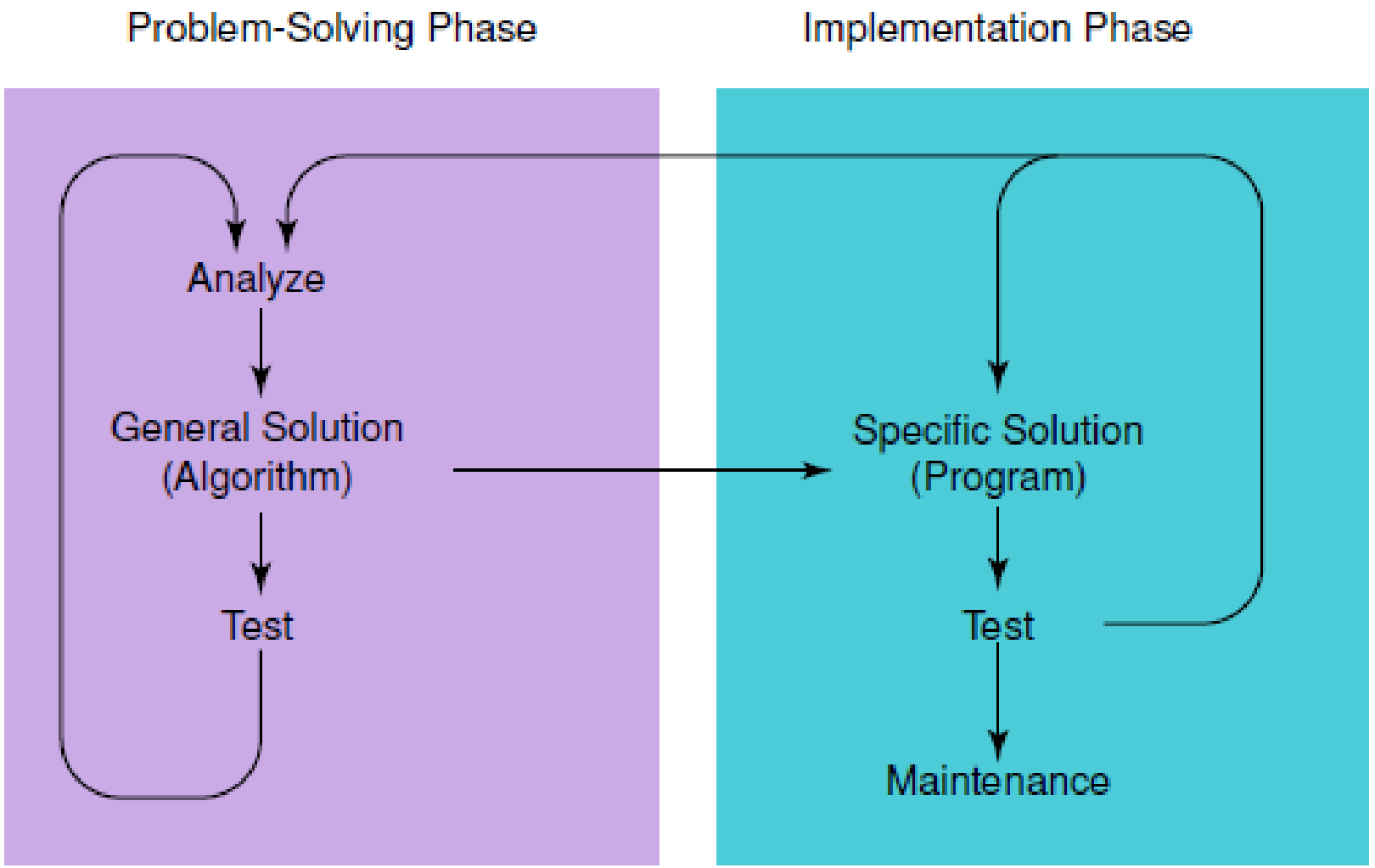
- Phương pháp giải quyết vấn đề thông thường (trong cuộc sống) : **4 bước**
  - Bước 1: **Hiểu vấn đề**: cái gì chưa biết, cái gì là dữ liệu, cái gì là điều kiện
  - Bước 2: **Đưa ra một phương án**: tìm mối quan hệ giữa dữ liệu và những thứ chưa biết, có thể tham khảo từ cách giải quyết các vấn đề tương tự
  - Bước 3: **Thực hiện phương án**
  - Bước 4: **Kiểm tra lại lời giải thu được**

Thuật toán

Giải quyết vấn đề bằng máy tính

- **Giai đoạn phát triển thuật toán**
  - **Phân tích:** hiểu vấn đề
  - **Đề xuất thuật toán:** đưa ra các bước tuần tự giải bài toán
  - **Kiểm tra thuật toán:** theo các bước để kiểm tra lại thuật toán
- **Giai đoạn triển khai**
  - **Code:** chuyển thuật toán thành chương trình
  - **Kiểm tra:** thực hiện trên máy tính, kiểm tra kết quả và sửa đổi nếu cần
- **Giai đoạn bảo trì**
  - **Sử dụng:** Dùng chương trình
  - **Bảo trì:** sửa đổi chương trình cho phù hợp yêu cầu mới hoặc để sửa lỗi.

Thuật toán



Pha giải quyết vấn đề ***Pha triển khai, cài đặt***

Ví dụ

- Bài toán tìm đoạn con có tổng lớn nhất
- Đầu vào:
  - Dãy N các số thực $a_1, a_2, .., a_N$
  - Đoạn con là 1 dãy liên tiếp các phần tử được lấy từ dãy ban đầu $a_i, a_{i+1}, .., a_j$ là đoạn con từ vị trí i tới j trong dãy ban đầu
- Đầu ra
  - Đưa ra 1 đoạn con có tổng lớn nhất

Đầu vào	Đầu ra
3, 5, -9, 6, 4, 2, -8, -9, 1, 4	6,4,2

Ví dụ

- Thuật toán 1. Vét cạn
  - Duyệt hết tất cả các đoạn con có thể
  - Có $^{N*(N-1)}/_{2}$ đoạn con có thể
  - Không tốn bộ nhớ phụ (ngoại trừ các biến tạm đơn lẻ)
- Thuật toán 2. Tìm kết hợp với đánh dấu vị trí
  - Mỗi phần tử chỉ cần duyệt qua 1 lần
  - Tốn bộ nhớ phụ để đánh dấu

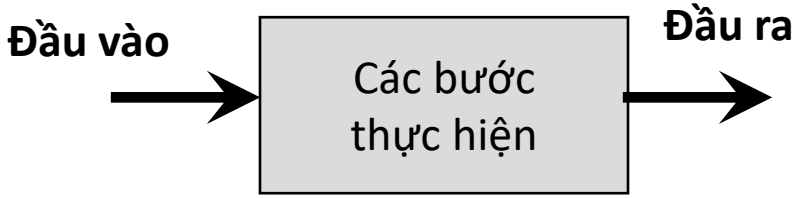
Thuật toán/ giải thuật

Chương trình = CTDL + Thuật toán

- Chương trình (phần mềm máy tính) : giải quyết 1 hoặc 1 vài bài toán bằng máy tính
- Chương trình (phần mềm máy tính) là cài đặt của một hoặc nhiều thuật toán khác nhau bằng ngôn ngữ lập trình cụ thể
- Hiệu quả của chương trình?
  - Quyết định bởi thuật toán
  - Và CTDL được lựa chọn
- Vậy thuật toán/giải thuật và CTDL là gì?

Thuật toán là gì ?

- **Thuật toán**:
  - thủ tục để thực hiện một nhiệm vụ cụ thể
  - ý tưởng nằm sau các chương trình máy tính.
- Thuật toán phải giải quyết bài toán tổng quát, và được định nghĩa rõ ràng.
- Một thuật toán 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** cho một **đầu vào** cho trước của bài toán.



Thuật toán

- Đặc điểm
  - Tính tổng quát: áp dụng trên mọi trường hợp có thể có của đầu vào bài toán
  - Tính đơn trị: Kết quả chỉ phụ thuộc vào giá trị đầu vào và kết quả trung gian
  - Tính hữu hạn: Phải dừng và trả về kết quả sau một thời gian
  - **Tính chính xác(*)**: Giá trị đầu ra thu được phải đúng với giá trị đầu vào
  - **Tính hiệu quả(*)**: chương trình phải hiệu quả, chạy với lượng thời gian và bộ nhớ chấp nhận được

Cấu trúc dữ liệu

- Cấu trúc dữ liệu:
  - Là cách lưu trữ và biểu diễn dữ liệu của bài toán, kết quả trung gian của quá trình tính toán trên máy tính
- Một số khái niệm liên quan
  - **Kiểu dữ liệu – Data type**: tập các giá trị và các phép toán được thực hiện trên các giá trị đó.
  - **Kiểu dữ liệu trừu tượng – Abstract Data Type**: là mô tả về kiểu dữ liệu (tập giá trị, các phép toán), nhưng chưa đề cập đến cách biểu diễn cụ thể trên máy
  - **Kiểu dữ liệu dựng sẵn – Built-in Data Type**: Là các kiểu dữ liệu đã được biểu diễn (cài đặt) sẵn trong một ngôn ngữ lập trình cụ thể

Cấu trúc dữ liệu

- Định nghĩa một cấu trúc dữ liệu gồm
  - Các kiểu dữ liệu
  - Và mối quan hệ giữa các kiểu dữ liệu
- Khi lựa chọn cấu trúc dữ liệu
  - Khả năng biểu diễn (dải giá trị,…)
  - Các thao tác (phép toán,..) mà nó hỗ trợ
  - Cài đặt cụ thể của cấu trúc dữ liệu đó trên máy

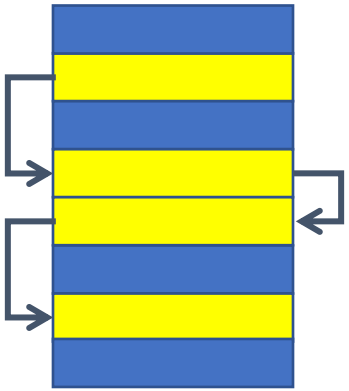
Cấu trúc dữ liệu

- Chương trình = CTDL + Giải thuật
- Thay đổi cấu trúc dữ liệu **không làm thay đổi tính chính xác** của chương trình. Tuy nhiên nó sẽ làm thay đổi **hiệu quả** của chương trình.
- Tốt nhất nên chọn cấu trúc dữ liệu cho hiệu quả cao thiết kế chương trình!

Cấu trúc liên tục VS liên kết

- Các cấu trúc dữ liệu có thể được chia thành liên tục (*contiguous*) hoặc liên kết(*linked*), tùy vào việc nó được cài đặt dựa trên mảng hay con trỏ.

Cấu trúc được cấp phát liên tục: được cấp phát thành vùng bộ nhớ liên tục. VD mảng, ma trận, đống (heap), và bảng băm



Cấu trúc dữ liệu liên kết: gồm các đoạn(chunk) trong bộ nhớ (không nằm liên tục) và được liên kết với nhau thông qua con trỏ. VD, danh sách, cây, và đồ thị danh sách kề.

Một số ví dụ

- VD 1. Bài toán nhân 2 số nguyên lớn
  - Đầu vào: 2 giá trị số nguyên
  - Đầu ra: Kết quả phép tính +, -, *, / và % của 2 giá trị số nguyên

- Vấn đề:
  - Số nguyên nhỏ: có thể dùng các kiểu dữ liệu dựng sẵn của NNLT như char, int, long, long long, (Big Int – trong java). Các phép toán đã được hỗ trợ sẵn.
  - Số nguyên lớn: Vd tới 1000 chữ số (trong bài toán mã hóa bảo mật) các số nguyên biểu diễn thế nào? Các phép toán cài đặt ra sao?

Một số ví dụ

- VD 2. Biểu diễn 1 vector n chiều (hoặc ma trận nxm) với các phép toán cơ bản như +, -, *, /, tích descartes (chuyển vị, nghịch đảo)
- Vấn đề
  - Số lượng chiều n lớn hay nhỏ
  - Vector thưa hay dày (thưa là có quá nửa thành phần bằng 0)
  - Thời gian tính toán cho phép
  - Tối ưu bộ nhớ

Một số ví dụ

- VD 3. Bài toán tìm kiếm xem 1 giá trị khóa k có xuất hiện trong dãy n phần tử đã cho hay không
  - Đầu vào: dãy n phần tử với khóa tìm kiếm, và giá trị khóa cần tìm k
  - Đầu ra: trả lời câu hỏi k có xuất hiện (vị trí xuất hiện) hay không
- Vấn đề
  - Kiểu giá trị của khóa: số, xâu ký tự hay object
  - Kích thước danh sách: 100, 100000 hay lớn hơn
  - Danh sách được lưu trữ trong RAM toàn bộ hay phải lưu trữ trên bộ nhớ ngoài?
  - Thời gian tìm kiếm cho phép?
  - Thuật toán tìm kiếm?

Một số ví dụ

- VD 4. Xây dựng chương trình lưu trữ và tra cứu danh sách tiêm chủng covid theo CCCD (chưa tiêm, đã tiêm 1 mũi, 2 mũi,..)
  - Đầu vào: danh sách thông tin tiêm chủng (CCCD, họ tên, SDT, địa chỉ, …) và giá trị CCCD cần tra cứu
  - Đầu ra: trả lời cho câu hỏi CCCD đó có trong danh sách tiêm hay chưa (trả về thông tin thêm nếu có)
- Vấn đề:
  - Thông tin chi tiết về 1 người đăng ký tiêm gồm những gì?
  - Lưu trữ thông tin đó thế nào?
  - Số lượng người dự kiến (số lượng tối đa) có biết trước?
  - Lưu trữ đủ trong RAM hay phải trên bộ nhớ ngoài
  - Thuật toán tìm kiếm?

Một số ví dụ

- VD 5. Cho danh sách thông tin của n thí sinh đăng ký nguyện vọng vào BKHN, hãy đưa ra điểm chuẩn đầu vào nếu chỉ tiêu tuyển sinh năm nay là k
  - Đầu vào: danh sách thông tin hồ sơ thí sinh(mã hồ sơ, họ tên, địa chỉ, điểm,..)
  - Đầu ra: Danh sách thông tin thí sinh đạt (được sắp theo thứ tự điểm giảm dần)
- Vấn đề:
  - Số lượng phần tử trong danh sách (n=1000, 10000, 100000000,…)
  - Thông tin hồ sơ được lưu trữ trong máy tính thế nào (dùng CTDL gì, lưu hết trong RAM hay phải lưu cả trên bộ nhớ ngoài)
  - Thời gian sắp xếp cho phép (dưới 1s, 10s, hay 1 giờ, 1 ngày, 1 tháng,…)
  - Số lượng chỉ tiêu k nhỏ và có nhiều thí sinh bằng điểm? Ưu tiên ai?

Thuật toán

Thuật toán

- Làm thế nào để xây dựng được thuật toán giải bài toán ban đầu?
  - Với các bài toán đơn giản: như tìm kiếm, sắp xếp trên danh sách nhỏ ta có thể dễ dạng tìm được thuật toán có sẵn
  - Với các bài toán phức tạp (như ví dụ 4 hoặc 5) sẽ KHÔNG có thuật toán nào có thể giải ngay được.
  - Giải pháp là chia nhỏ bài toán ban đầu và cần nhiều thuật toán khác nhau để giải các bài toán con khác nhau.
    - Chia như thế nào?
    - Bài toán cỡ nào được coi là nhỏ?
    - Tổng hợp lời giải ra sao?
  - Giải pháp khác: theo cách tiếp cận hướng đối tượng
    - Xác định các đối tượng và tương tác giữa các đối tượng (properties & methods)

Thuật toán

- Bài toán tìm điểm chuẩn đầu ra cho danh sách n hồ sơ nguyện vọng vào BKHN và chỉ tiêu là k
- Bài toán có thể chia nhỏ thành
  - Nhập danh sách hồ sơ nguyện vọng vào máy tính: nếu hồ sơ là giấy, hoặc bản điện tử cách xử lý sẽ khác nhau
  - Sắp xếp danh sách hồ sơ theo tổng điểm giảm dần: Tổng điểm đã được tính cả điểm cộng
  - Đưa ra ngưỡng điểm sàn cho số lượng chỉ tiêu là k: chọn sao cho số lượng hồ sơ trùng điểm sàn là ít nhất?

Thuật toán

Liệu với mỗi bài toán luôn phải nghĩ ra thuật toán mới?

- Với các bài toán thông thường thì các thuật toán đã có sẵn (trong thư viện, trong Lib, hoặc trên mạng,…)
- Các bài toán mới thường sẽ kế thừa (có sửa đổi) từ các thuật toán đã có
- Để tiến lên phía trước "Chúng ta luôn đứng trên vai người khổng lồ chứ không đi phát minh lại bánh xe"

Vậy áp dụng CTDL&TT thế nào?

- Học để nắm được ưu nhược điểm của các thuật toán giải quyết các bài toán cơ bản ➔ biết càng nhiều bài toán, thuật toán càng tốt
- So sánh, đánh giá được thuật toán để chọn thuật toán phù hợp nhất với bài toán hiện tại

Thuật toán

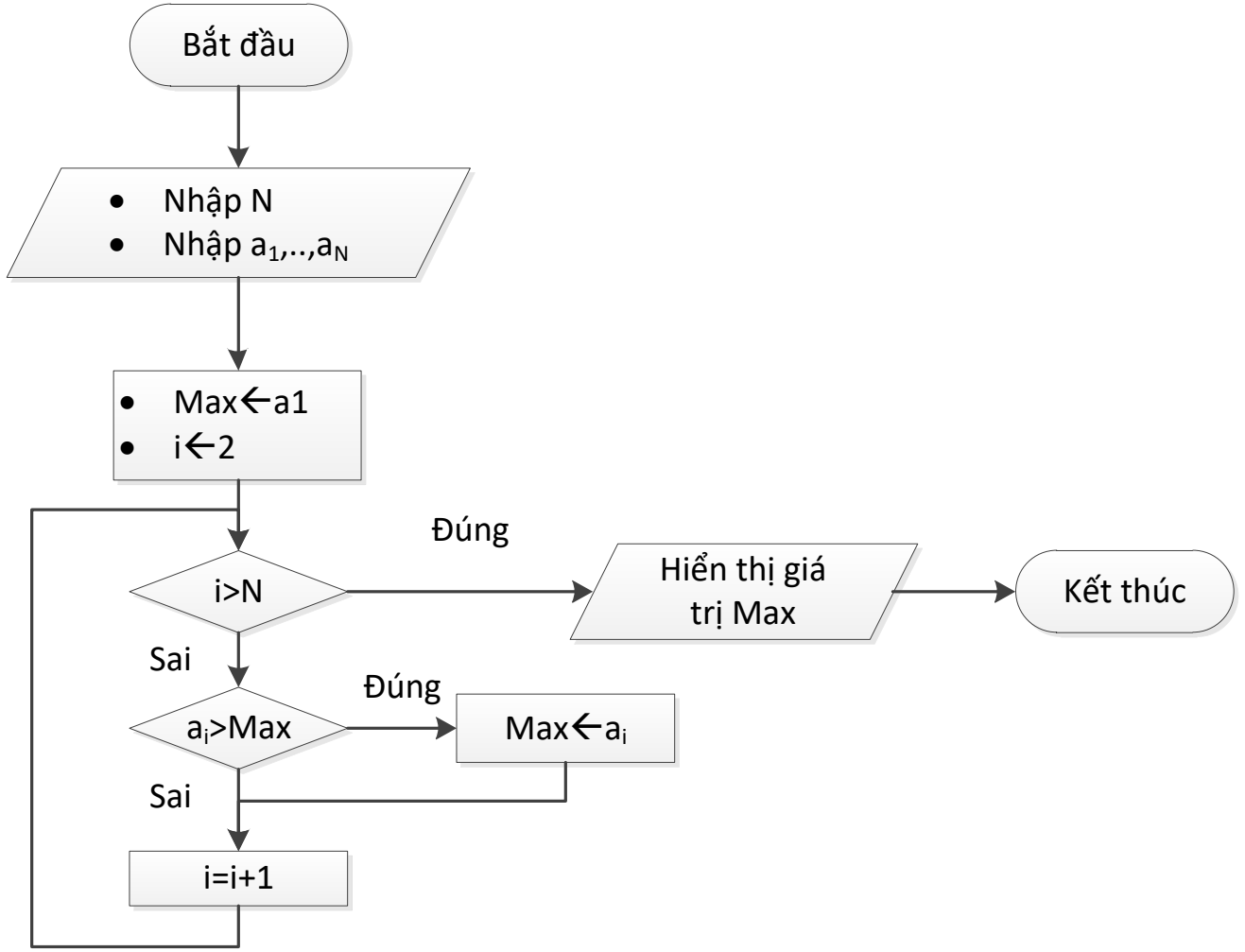
- **Biểu diễn thuật toán**: Để diễn đạt thuật toán cho người khác hiểu
  - Biểu diễn bằng ngôn ngữ tự nhiên: thường qua các Bước (hay nhập nhằng nếu thuật toán phức tạp)
  - Dùng sơ đồ khối: diễn tả rõ luồng thực thi (không phù hợp với bài toán phức tạp)
  - Dùng giả ngôn ngữ: có thể dễ dạng triển khai sang một NNLT cụ thể (đủ dễ hiểu và chặt trẽ)
  - Dùng một ngôn ngữ lập trình cụ thể: chặt trẽ, tuy nhiên sẽ khó hiểu nếu thuật toán phức tạp

Thuật toán

- Bài toán tìm giá trị lớn nhất trong dãy n số nguyên

Thuật toán: Tìm giá trị lớn nhất trong dãy số nguyên
B1: Max ← a_1, i ← 2.
B2: Nếu i > N, Chuyển qua bước 6
B3: Nếu a_i > Max, gán Max bằng a_i .
B4: Tăng i lên 1 đơn vị.
B5: Quay lên B2.
B6: In ra Max (là giá trị lớn nhất cần tìm)

Ngôn ngữ tự nhiên



Sơ đồ khối

Thuật toán

- Bài toán tìm giá trị lớn nhất trong dãy n số nguyên

```
max = A[0]
for i=1 to n do
    if(max<A[i]) max = A[i]
write(max)
```

Mã giả

```
int max = A[0];
for (int i = 1; i < n; i++)
     if (max < A[i]) max = A[i];
printf(" %d", max);
```

NNLT C

Mã giả

- Mô tả thuật toán đơn giản, gần gũi, ngắn gọn và không phụ thuộc vào cú pháp ngôn ngữ lập trình cụ thể

Assignment

```
x = <expression>;
x ← <expression>;
```

Condition

```
if a < b then {
  . . .
}
```

For loop

```
for i = 1 to n do{
   . . .
}
```

Procedures, funtions

```
proc(a,b,x){
   . . .
   return ans;
}
```

While loop

```
while i ≠ 100 do{
   . . .
}
```

```
max(a[1..n]){
  ans = a[1];
  for i = 2 to n do
    if ans < a[i] then
      max = a[i];
  return ans;
}
```

Mã giả

- Một bài toán (ví dụ sắp xếp) có thể có nhiều thuật toán giải quyết

```
selectionSort(a[1..n]){
   for k = 1 to n do{
    min = k;
    for j = k+1 to n do{
      if a[min] > a[j] then
        min = j;
    }
    swap(a[k],a[min]);
  }
}
```

```
insertionSort(a[1..n]){
   for k = 2 to n do{
     last = a[k];
     j = k;
   while(j > 1 and a[j-1] > last){
     a[j] = a[j-1];
     j--;
   }
   a[j] = last;
  }
}
```

Thuật toán

- Cài đặt thuật toán:
  - Cài đặt dùng vòng lặp: thường dài
  - Cài đặt dùng đệ quy: thường ngắn gọn nhưng kém hiệu quả hơn (tại sao?)
- VD. Tìm giá trị lớn nhất trong dãy n phần tử số nguyên

```
int findMax_loop(int A[], int n)
{
   int max = A[0];
   for(int i=1; i<n; i++)
      if(max<A[i]) max = A[i];
   return max;
}
```

```
int findMax_rec(int *A, int n)
{
    if(n=1) return A[0];
    return MAX(A[n-1], findMax_rec(A, n-1));
}
```

Thuật toán

- Với bài toán tương tự bài toán đã có
  - Dùng thuật toán đã có sẵn để giải (các thuật toán này đã được CM tính chính xác)
  - Có nhiều thuật toán để cùng giải bài toán → phải chọn lựa thuật toán phù hợp với yêu cầu bài toán hiện tại
- Với 1 bài toán mới
  - Đề xuất thuật toán để giải (thường sẽ phải dựa vào thuật toán giải các bài toán gần giống)
  - Làm thế nào để chứng minh được thuật toán vừa đưa ra là chính xác?
  - Đánh giá hiệu quả của thuật toán đề xuất?

Tính chính xác của thuật toán

Thuật toán

Tính chính xác của thuật toán

- Thuật toán phải cho đầu ra mong muốn ứng với bất cứ đầu vào hợp lệ nào của bài toán.
- Tính chính xác không phải lúc nào cũng dễ thấy!

Chỉ ra thuật toán sai:

- Chỉ cần đưa ra ví dụ trường hợp kết quả sai khi áp dụng thuật toán (phản ví dụ)

Chứng minh thuật toán đúng: phức tạp hơn nhiều

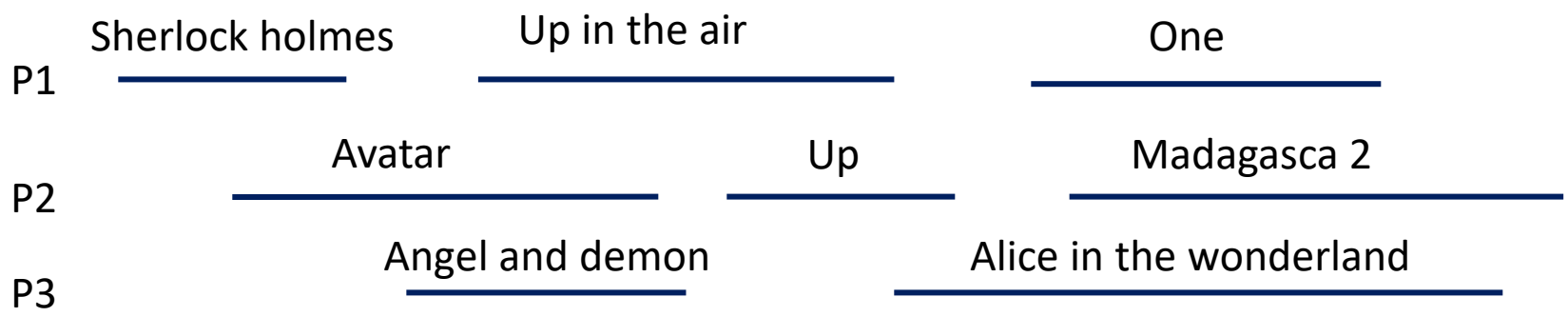
- Chứng minh bằng toán học: chặt trẽ nhưng mất nhiều thời gian
- Dùng kiểm thử: xây dựng các test cases
  - Có thể không xét được hết các trường hợp có thể có của đầu vào → bỏ sót
  - Chương trình vẫn có thể lỗi khi triển khai, mặc dù đã qua được hết các test cases

Thuật toán

- Ví dụ 1. Bài toán các đoạn thẳng không giao nhau

Đưa ra phương án chọn lịch xem phim

  - **Đầu vào**: Một tập L gồm thời gian chiếu trong ngày của n bộ phim
  - **Đầu ra**: Tập con của L chứa số bộ phim lớn nhất có thể xem (không được chồng nhau về thời gian)



Tính chính xác

- **Thuật toán 1**. Chọn bộ phim sớm nhất trong L mà không trùng với các bộ phim đã chọn trước đó. Lặp lại cho đến khi không thể chọn thêm.

- **Thuật toán 2**. Chọn bộ phim có thời gian chiếu ngắn nhất trong L mà không trùng với các bộ phim đã chọn trước. Lặp lại cho đến khi không chọn thêm được.

Tính chính xác

- **Thuật toán 3**. **Vét cạn - Duyệt toàn bộ**: duyệt 2^n tập con của n bộ phim trong L. Chọn ra tập con nào có số lượng phần tử lớn nhất.
  - Đảm bảo thu được kết quả tối ưu
  - Thuật toán chạy rất chậm khi đầu vào lớn, vd n =20 thì số tập con là 2^{20}

- **Thuật toán 4**. **Thuật toán tối ưu**:
  - sắp xếp các lịch chiếu phim theo thứ tự không giảm thời gian kết thúc.
  - Lần lượt xem xét các phim trong danh sách đã sắp xếp, bổ sung vào danh sách xem bộ phim đang xét nếu nó **không trùng với các bộ phim** đã có trong danh sách xem.

- **Có những bài toán không tồn tại thuật toán chính xác với thời gian đa thức!**

Bài toán cái túi

- **Đầu vào**: n đồ vật, mỗi đồ vật *i* có một trọng lượng w_i và một giá trị c_i. Một cái túi có thể chứa các đồ vật với trọng lượng tối đa là *b*
- **Đầu ra:** Cách chất các đồ vật vào túi sao cho trọng lượng tối đa không vượt quá b, và tổng giá trị các đồ vật trong túi là lớn nhất.

$$W = \sum_{i=1}^{k} w_i \leq b \qquad C = \sum_{i=1}^{k} c_i \rightarrow max$$

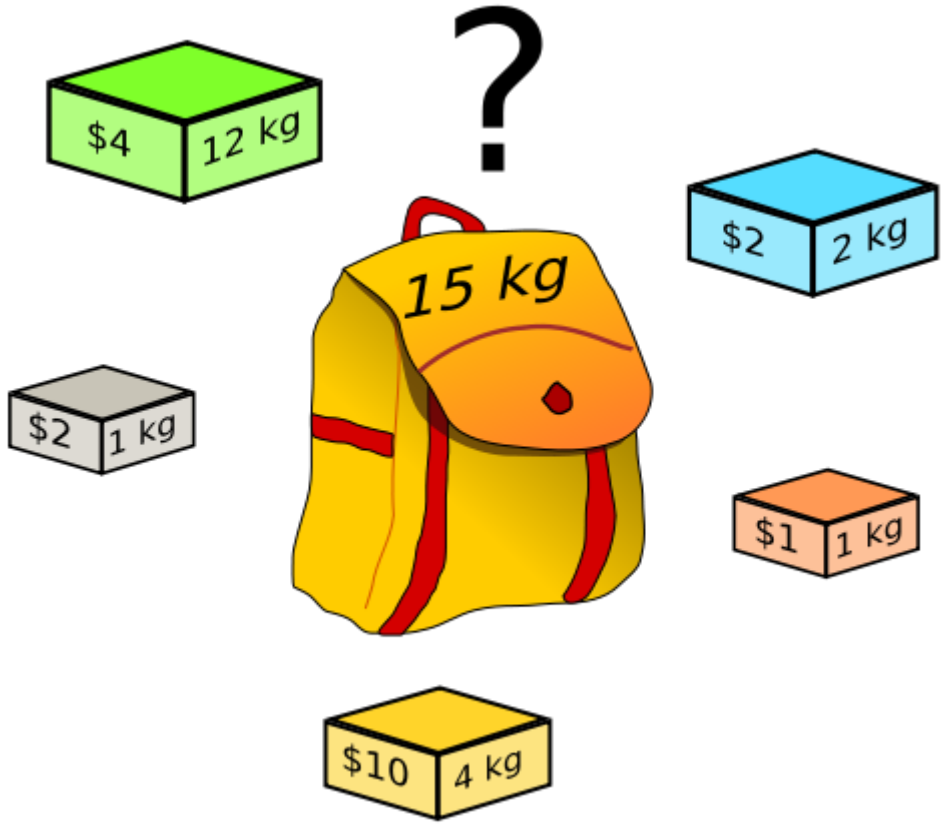
- Xây dựng thuật toán chất các đồ vào túi ?

Thuật toán 1. Chọn đồ vật có giá trị cao trước

- Sắp xếp các đồ vật theo thứ tự giảm về giá trị.
- Lần lượt xét các đồ theo thứ tự này, cho đồ vật đang xét vào túi nếu nó còn có thể chứa thêm được

Thuật toán 2. Chọn đồ vật trọng lượng nhỏ trước

- Sắp xếp các đồ vật theo thứ tự tăng trọng lượng
- Lần lượt xét các đồ vật theo thứ tự này,
  chọn đồ vật đang xét vào túi
  nếu nó vẫn có thể chứa thêm



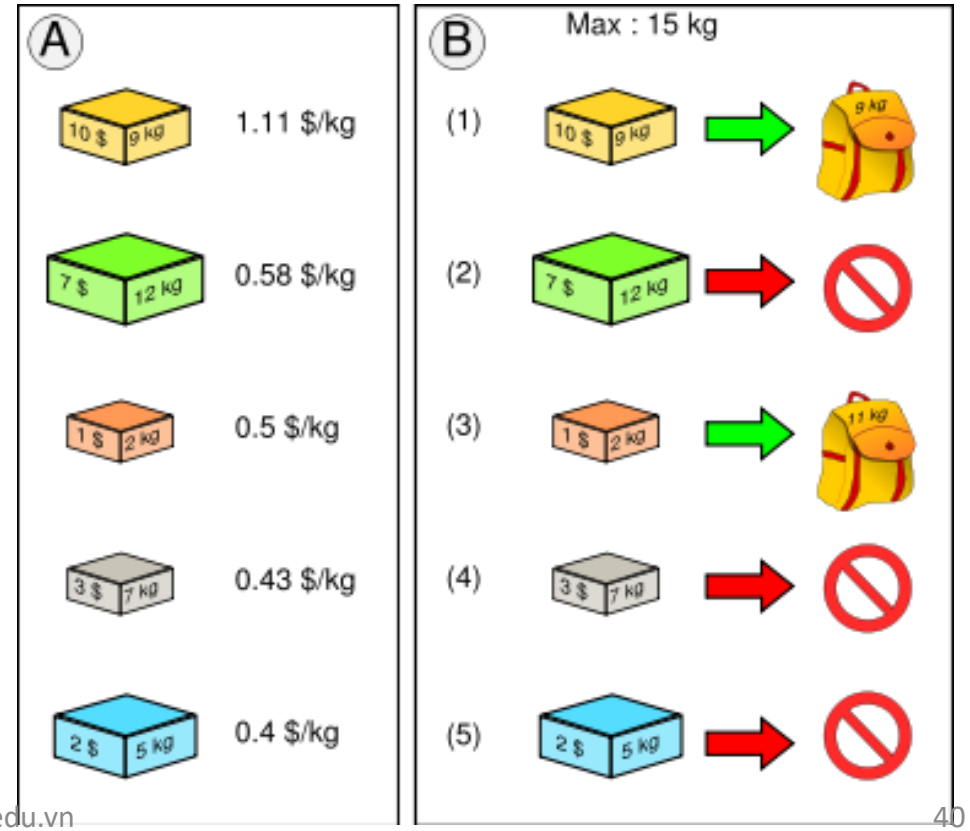
Thuật toán 3. Chọn đồ vật theo tỉ lệ c_i/w_i

- Sắp xếp các đồ vật theo thứ tự giảm của tỉ lệ giá trị/ trọng lượng

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \frac{c_3}{w_3} \ldots \geq \frac{c_n}{w_n}$$

  - Lần lượt xét các đồ vật theo thứ tự này, chọn đồ vật đang xét vào túi nếu nó vẫn có thể chứa thêm

- Thuật toán chính xác?
  - Vét cạn với số đồ vật nhỏ
  - Với số đồ vật lớn, giải gần đúng (VD. thuật toán nhánh cận)



Thuật toán

- Tìm phản ví dụ ?
  - Tìm trong các trường hợp dữ liệu nhỏ
  - Các ví dụ mà sát với các tiêu chuẩn lựa chọn của thuật toán
  - Các ví dụ của các trường hợp cực trị (lớn nhất, nhỏ nhất …)

  Không tìm được phản ví dụ không có nghĩa thuật toán là đúng!
- Chứng minh thuật toán đúng
  - Quy nạp, phản chứng
  - Dùng logic
  - Chứng minh bằng thực nghiệm (chạy trên bộ test)

Chứng minh tính đúng đắn

- **Thuật toán được định nghĩa đệ quy**: Thuật toán được định nghĩa lại bằng chính nó (với kích thước bài toán nhỏ hơn)

$$n! = \begin{cases} 1 & nếu\ n = 0 \\ n \times (n-1)! & nếu\ n > 0 \end{cases}$$

- Chứng minh tính đúng đắn của thuật toán đệ quy bằng phương pháp quy nạp

Tính hiệu quả của thuật toán

Tính hiệu quả

- Tại sao cần thuật toán hiệu quả khi mà chỉ cần dùng máy tính mạnh hơn nếu muốn chạy nhanh hơn?
  - Liệu máy tính có thể nhanh mãi được?
  - Chi phí về kinh tế?
- Đánh giá thuật toán: Dự đoán lượng tài nguyên cần
  - Tài nguyên? Bộ nhớ trong, thời gian CPU, băng thông mạng, …
- Làm thế nào để đánh giá hiệu quả của một thuật toán? (tạm thời bỏ qua ảnh hưởng của CTDL)
  - Đánh giá gián tiếp thông qua đánh giá hiệu quả chương trình máy tính tương ứng?
  - Vậy hiệu quả của chương trình máy tính đo bằng gì?
  - Cách đo vậy liệu có đủ tin cậy khi so sánh các thuật toán với nhau?
  - Việc so sánh liệu có dễ dàng thực hiện?

Đánh giá gián tiếp

- VD. bài toán sắp xếp danh sách hồ sơ nguyện vọng nộp vào DHBKHN với số lượng hồ sơ tầm 10000 và theo thứ tự giảm dần về điểm
  - Thuật toán 1. Chạy mất thời gian 1.5s
  - Thuật toán 2. Chạy mất thời gian 32.5s
- Liệu có thể kết luận thuật toán 1 tốt hơn thuật toán 2? (về thời gian)
  - Cả 2 thuật toán có cùng chạy trên máy cấu hình tương đương?
  - Có cùng cài dặt dùng ngôn ngữ lập trình tương đương?
  - Thời gian chạy đo như thế nào?
  - Bộ nhớ trống của RAM tại thời điểm chạy?
  - Có bị ảnh hưởng bởi ứng dụng nào chạy đồng thời trên máy?
  - Người lập trình có kỹ năng tương đương?

Đánh giá gián tiếp

- Đánh giá các thuật toán bằng cách gián tiếp
  - Phải cài đặt các thuật toán dùng các NNLT tương đương
  - Cùng chạy trên 1 bộ test đầu vào giống nhau
  - Cấu hình phần cứng và phần mềm khi đánh giá phải tương đồng (tốt nhất trên cùng 1 máy)
  - Thường không dùng thời gian khi chạy trên máy khác, cài đặt bởi người khác để so sánh ➔ không tận dụng được kết quả cũ
  - Nếu muốn so sánh với các thuật toán khác ➔ phải cài đặt và chạy lại toàn bộ nên tốn nhiều thời gian
  - Bị ảnh hưởng nhiều bởi phần cứng, NNLT và kỹ năng lập trình

Tính hiệu quả

- Đánh giá hiệu quả thuật toán trực tiếp?
  - Đánh giá dựa trên mô tả thuật toán (mô tả nên bằng giả ngôn ngữ hoặc NNLT cụ thể để tránh nhập nhằng)
  - Không phụ thuộc vào phần cứng, phần mềm hoặc NNLT cụ thể nên kết quả đánh giá đủ tin cậy khi so sánh các thuật toán với nhau
  - Khi đánh giá thuật toán mới, không cần đánh giá lại thuật toán cũ (chỉ cần dùng lại kết quả)
- Thời gian CPU là tài nguyên quan trọng nhất
  - Đánh giá hiệu quả thường hiểu là đánh giá độ phức tạp về thời gian thực hiện
- Phương pháp đánh giá trực tiếp
  - Mô hình RAM – Random Access Machine
  - Mô hình O-lớn

Đánh giá hiệu quả thuật toán

Mô hình RAM

- Thực hiện thuật toán trên một máy tính giả định gọi là *Random Access Machine* hoặc RAM.
  - Mỗi phép tính đơn giản (+, *, –, =, if,..) thực hiện trong 1 đơn vị thời gian (hoặc 1 bước).
  - Vòng lặp, hàm, thủ tục: là kết hợp của nhiều phép tính đơn lẻ
  - Mỗi bước truy cập bộ nhớ mất 1 đơn vị thời gian
  - Luôn có đủ bộ nhớ cần thiết để thực hiện thuật toán
- Đánh giá thời gian thực hiện thuật toán bằng cách đếm số đơn vị thời gian cần.
  - Thuật toán nào càng cần nhiều thời gian đơn vị thì càng tồi
  - Chỉ dùng để đánh giá tài nguyên thời gian CPU

Mô hình RAM

- VD1. Tính tổng các phần tử trong dãy n phần tử

```
1: int sumKeys(int* A, int n)
2: {
3:          int i;
4:          int sum = 0;
5:          for (i = 0; i < n; i++)
6:                    sum=sum+A[i];
7:          return sum;
8: }
```

Dòng	Số lần thực hiện	Thời gian	Ghi chú
3	1	1	
4	1	1	
5	n	n	Lệnh lặp
6	n	n	Lệnh lặp
7	1	1	

Thời gian thực hiện phụ thuộc vào kích thước đầu vào (số lượng phần tử, số bit, kích thước bộ nhớ biểu diễn đầu vào …)

$$T(n) = 2n + 3$$

Mô hình RAM

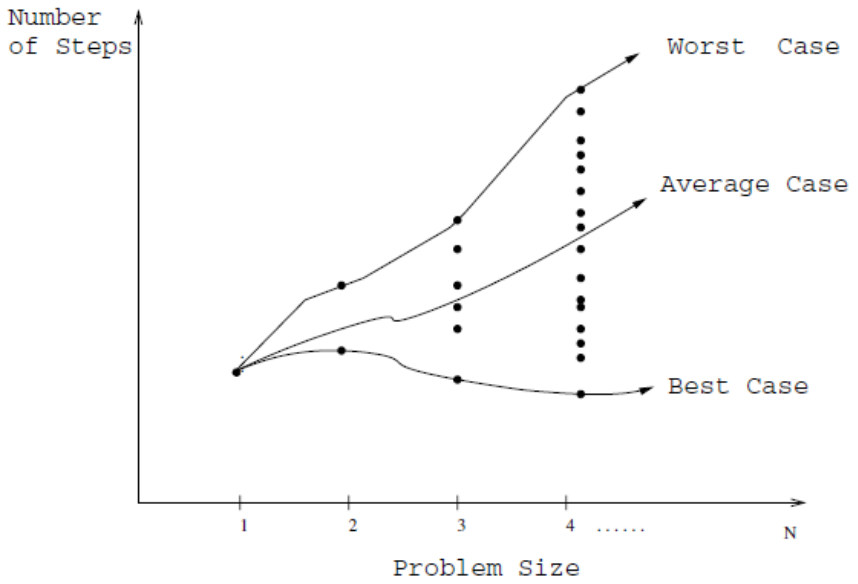
- VD 2. Thuật toán tìm kiếm xem khóa k có xuất hiện trong dãy n phần tử số nguyên cho trước hay không

```
1: int searchKey(int* A, int n, int key)
2: {
3:          int i;
4:          for (i = 0; i < n; i++)
5:                    if (A[i] == key) return i;
6:          return -1;
7: }
```

Dòng	Số lần thực hiện	Thời gian	Ghi chú
3	1	1	
4	n	n	Lệnh lặp
5	1	1	Nếu A[0]==key
5	n	n	Nếu ko giá trị nào bằng key
6	1	1	Thực hiện nếu như ko thực hiện return i

Thời gian thực hiện còn phụ thuộc vào giá trị cụ thể của đầu vào. Thời gian thực hiện được chia thành tốt nhất, tồi nhất và thời gian trung bình

Tốt nhất, tồi nhất và trung bình



- **Trường hợp tồi nhất (*worst-case complexity*):** Là số lượng bước ***lớn nhất*** thuật toán cần thực hiện với bất cứ đầu vào kích thước *n* nào.
- **Trường hợp tốt nhất (*best-case complexity*):** Là số lượng bước ***nhỏ nhất*** thuật toán cần thực hiện với bất cứ đầu vào kích thước *n* nào.
- **Thời gian trung bình (*average-case complexity*):** Là số lượng bước ***trung bình*** thuật toán cần thực hiện trên tất cả các trường hợp đầu vào kích thước *n*.

Với VD 2 thì
T(n) = 1+1+1 = 3 trong TH tốt nhất
T(n) = 2n + 2 trong TH tồi nhất

- Mỗi độ phức tạp là một hàm ***thời gian*** và ***kích thước*** đầu vào dạng

$$T(n) = 120n^3 + 12.4n^2 - 43nlogn + 9n$$

Tốt nhất, tồi nhất và trung bình

- Trường hợp tốt nhất và tồi nhất có thể rất hiếm khi xảy ra
- Thời gian trung bình với đa số bài toán rất khó tìm
  - Vì phải xét hết tất cả các trường hợp có thể có của đầu vào
- Thời gian trong nào mang ý nghĩa thực tiễn nhiều nhất?

"Hoping for the best, prepared for the worst, and unsurprised by anything in between."

— Maya Angelou, I Know Why the Caged Bird Sings

Read more quotes from Maya Angelou

Tốt nhất, tồi nhất và trung bình

- Cần hiểu thế nào về đánh giá thuật toán (về thời gian)
  - Là đánh giá độ phức tạp về thời gian của thật toán trong trường hợp tồi nhất
  - Ước lượng lượng thời gian lớn nhất mà thuật toán cần để đưa ra kết quả
- Dùng đánh giá này thế nào cho đúng
  - VD. Có 2 thuật toán A và B cùng giải bài toán P với độ phức tạp về thời gian trong trường hợp tồi nhất lần lượt là $T_A(n) = 100n^2+20$ và $T_B(n) = n^3+2n^2+n$.
    - Kết luận thuật toán A luôn nhanh hơn B?
    - Trong thực tế cần chọn thuật toán nào? A hay B?
  - Trong trường hợp tổng quát, cần chọn thuật toán có độ phức tạp về thời gian nhỏ hơn

Mô hình RAM

```
1.  insertionSort(a[1..n]){
2.    for j = 2 to n do{
3.      key = a[j];
4.      i = j-1;
5.      while i > 0 and a[i] > key do{
6.         a[i+1] = a[i];
7.         i = i – 1;
8.      }
9.      a[i+1] = key;
10.   }
11. }
```

Dòng	Thời gian	Số lần
2	c_2	*n-1*
3	c_3	*n*-1
4	C_4	*n*-1
5	C_5	$\sum_{j=2}^{n} t_j$
6	C_6	$\sum_{j=2}^{n} (tj - 1)$
7	C_7	$\sum_{j=2}^{n} (tj - 1)$
9	c_9	*n*-1

Ký hiệu t_j: số lần điều kiện của vòng lặp while (dòng 5) được thực hiện ứng với 1 giá trị *j* (vòng lặp bên ngoài)

Thời gian thực hiện $T(n) = c_2 n + c_3(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (tj - 1) + c_7 \sum_{j=2}^{n} (tj - 1) + c_9(n-1)$

Mô hình RAM

Thời gian tính $T(n) = c_2 n + c_3(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(tj-1) + c_7\sum_{j=2}^{n}(tj-1) + c_9(n-1)$

- Tình huống tốt nhất: dãy đã được sắp xếp, $t_j = 1$ (j = 2,…,n)

  → $T(n)$ có dạng $an + b$ (tuyến tính)
- Tình huống tồi nhất: dãy được sắp xếp theo thứ tự ngược lại, $t_j = j$ (j = 2,…, n)

  → $T(n)$ có dạng $an^2 + bn + c$ (bình phương)

- Nhận xét:
  - Mô hình RAM phải thống kê thời gian thực hiện của tất cả các câu lệnh đơn → quá chi ly và mất nhiều thời gian, nhất là trong thuật toán phức tạp
  - Độ lớn thời gian quyết định bởi lệnh được lặp nhiều nhất (lệnh cơ sở)
  - Trong thực tế, thường quan tâm nhiều đến tốc độ tăng thời gian hơn là thời gian cụ thể của thuật toán

Mô hình RAM

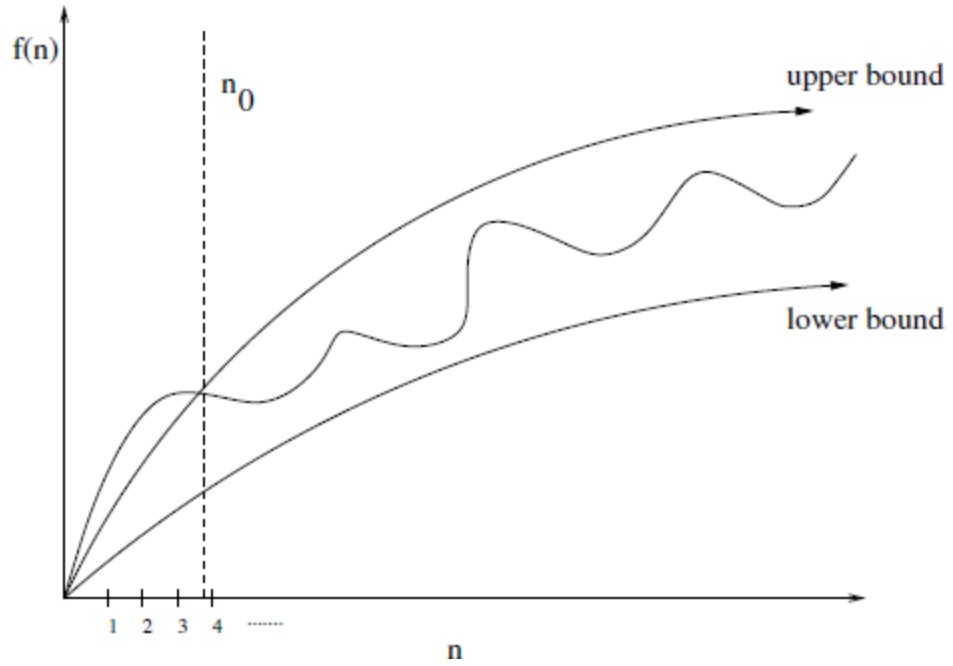
- Tốc độ tăng thời gian thực hiện là gì?
  - “Khi kích thước đầu vào tăng lên gấp đôi thì thời gian thực hiện của thuật toán tăng lên gấp mấy lần?”
- Đánh giá theo tốc độ tăng thuật tiện hơn
  - Chỉ cần quan tâm đến lệnh được lặp nhiều nhất
  - Trong chương trình thì đó là lệnh của vòng lặp trong cùng, hoặc lệnh của hàm được gọi đệ quy nhiều lần nhất → lệnh cơ sở
  - Số lượng lệnh cơ sở chỉ 1 vài lệnh → xác định nhanh hơn
  - Các thuật toán có cùng tốc độ tăng được xếp chung vào cùng 1 lớp
  - Trong công thức thời gian T(n) của mô hình RAM, tốc độ tăng quyết định bởi hệ số mũ lớn nhất của n

  → Mô hình tiệm cận, hoặc mô hình O-lớn

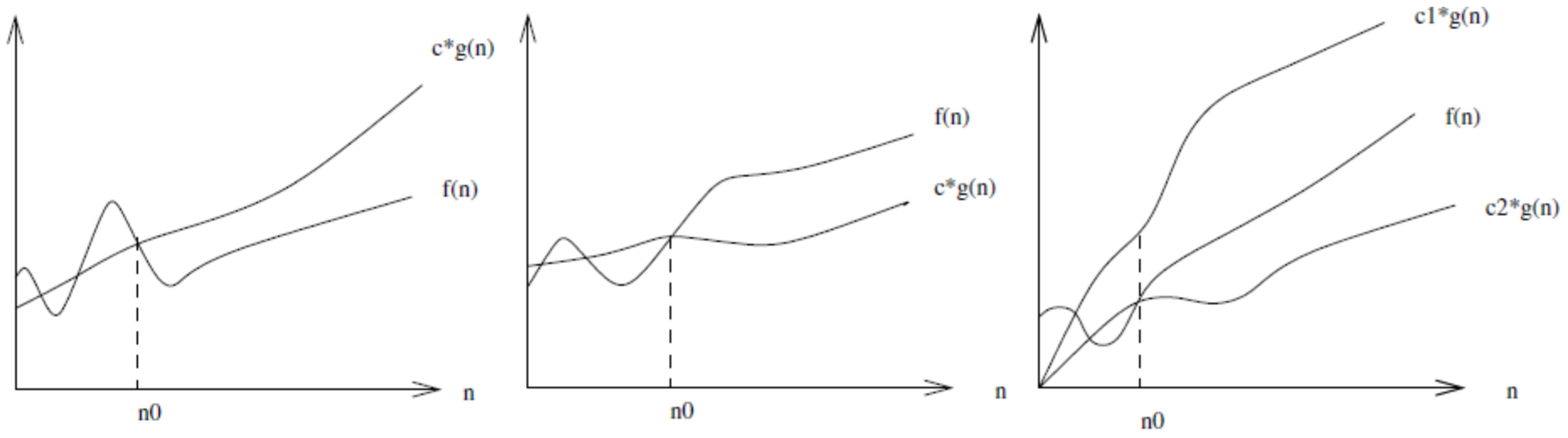
Mô hình O-lớn

- Dùng bộ ký hiệu tiệm cận (O, Θ, Ω)
- Định nghĩa O lớn chính thức:
  - $f(n) = O(g(n))$: ($c.g(n)$ *là giới hạn trên của* $f(n)$) nếu tồn tại hằng số c và n_0 sao cho $f(n) \leq c.g(n)$ luôn đúng với mọi $n \geq n_0$
  - $f(n) = \Omega(g(n))$: *($c.g(n)$ là giới hạn dưới của $f(n)$)* nếu tồn tại hằng số c và n_0 sao cho $f(n) \geq c.g(n)$ luôn đúng với mọi $n \geq n_0$
  - $f(n) = \Theta(g(n))$: ($c_1.g(n)$ *là giới hạn trên, và* $c_2.g(n)$ *là giới hạn dưới của* $f(n)$) nếu tồn tại hằng số c_1,c_2 và n_0 sao cho $f(n) \leq c_1.g(n)$ và $f(n) \geq c_2.g(n)$ luôn đúng với mọi $n \geq n_0$, $g(n)$ là giới hạn chặt của $f(n)$

Với c, c_1, c_2 là các hằng số dương không phụ thuộc vào n, và $n_0 > 0$



Cận trên, cận dưới để đánh giá cho độ phức tạp của hàm



(a) O lớn (b) Ω lớn (c) Θ lớn

Ký hiêu O lớn

Ví dụ

- $\boldsymbol{2n^2 - 4n + 5 = O(n^2)}$ vì chọn $c = 2$ thì $2n^2 > 2n^2 - 4n + 5$
- $\boldsymbol{2n^2 - 4n + 5 = O(n^3)}$ vì chọn $c = 1$ thì $n^3 > 2n^2 - 4n + 5$ khi $n > 1$
- $\boldsymbol{2n^2 - 4n + 5 \neq O(n)}$ vì với bất kỳ hằng số c nào thì $cn < 2n^2 - 4n + 5$ khi $n > c$

Omega Lớn

- $\boldsymbol{2n^2 - 4n + 5 = \Omega(n^2)}$ vì chọn $c = 1.5$ thì
  $1.5n^2 < 2n^2 - 4n + 5$ khi $n > 50$

- $\boldsymbol{2n^2 - 4n + 5 \neq \Omega(n^3)}$ vì với bất kỳ giá trị c thì
  $cn^3 > 2n^2 - 4n + 5$ khi n đủ lớn ($n > 100c$ nếu $c > 1$, $n > 10/c$ nếu $c < 1$)

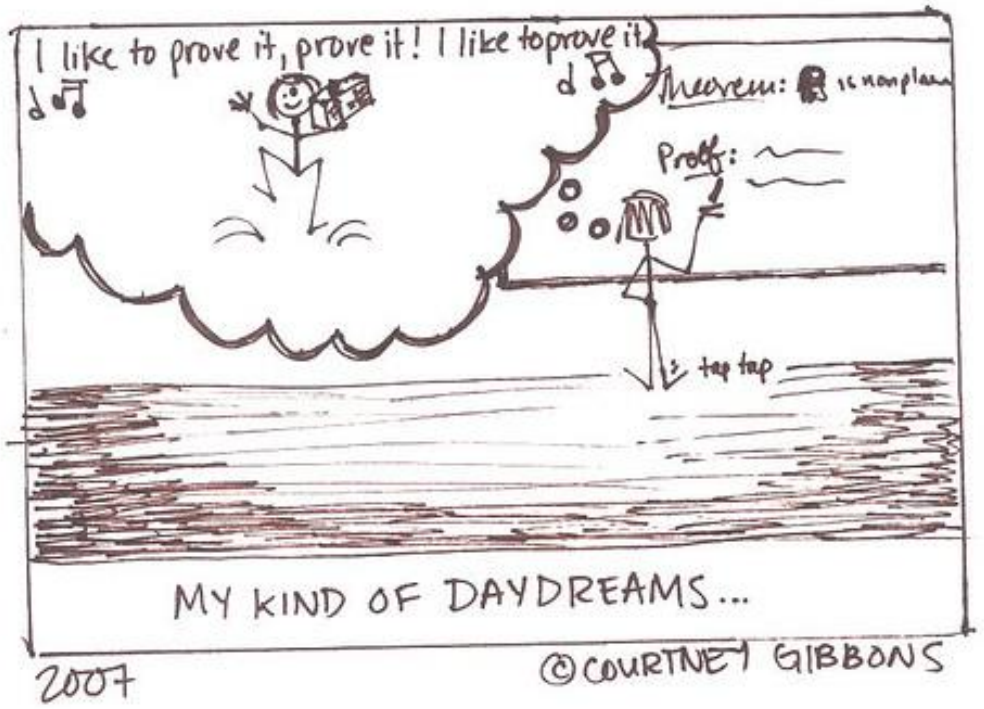
- $\boldsymbol{2n^2 - 4n + 5 = \Omega(n)}$ vì với bất kỳ hằng số c nào thì
  $cn < 2n^2 - 4n + 5$ khi $n > 5c$

Theta lớn

- $\boldsymbol{2n^2 - 4n + 5 = \Theta(n^2)}$ vì cả O, Ω đều đúng
- $\boldsymbol{2n^2 - 4n + 5 \neq \Theta(n^3)}$ vì chỉ O đúng
- $\boldsymbol{2n^2 - 4n + 5 \neq \Theta(n)}$ vì chỉ Ω đúng

Để chứng minh $f(n) = \Theta(g(n))$ thì cần chỉ ra

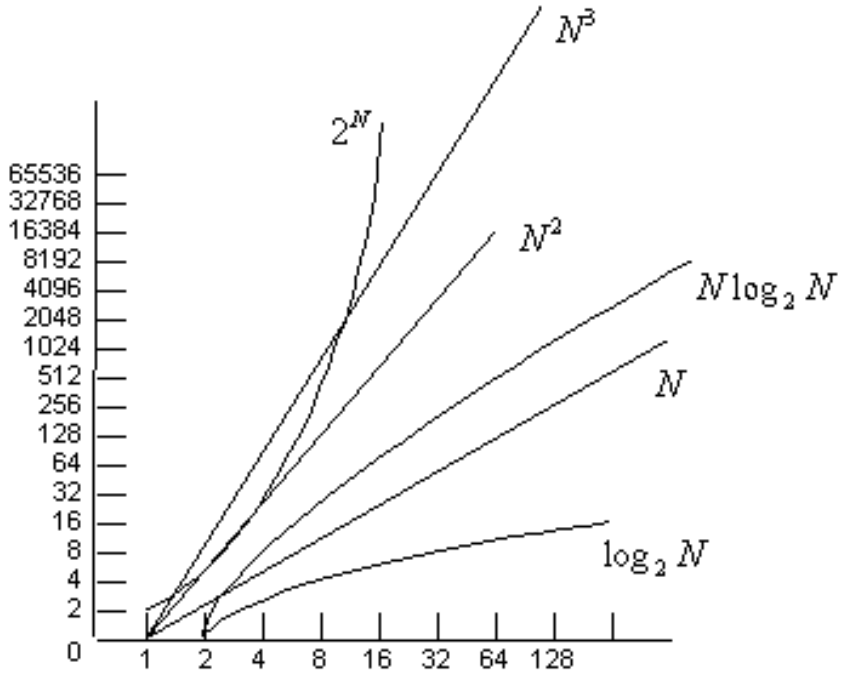
- $f(n) = O\big(g(n)\big)$
- $f(n) = \Omega(g(n))$



Ví dụ

Khẳng định sau đúng hay sai? Tại sao ?

- $2^{n+3} = \Theta(2^n)$
- $3^{2n} = \Theta(3^n)$
- $(x + y)^2 = O(x^2 + y^2)$



Tốc đô tăng và quan hê thống trị

- **Tốc độ tăng của một số hàm thông dụng**

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Tốc đô tăng và quan hê thống trị

- O lớn nhóm các hàm cùng tốc độ tăng thành các ***lớp hàm***.
  $n + 4$ và $100n - 3$ là thuộc lớp hàm $\Theta(n)$
- Hai hàm f, g thuộc hai lớp khác nhau có quan hệ theo các ký hiệu tiệm cận Ω, O khác nhau
- Các lớp hàm thông dụng:
  - Hàm hằng $f(n) = 1$. Thời gian thực hiện là hằng số VD hàm tính tổng 2 số
  - Hàm loga $f(n) = logn$. VD tìm kiếm nhị phân
  - Hàm tuyến tính $f(n) = n$. VD Tìm giá trị lớn nhất trong dãy số
  - Hàm siêu tuyến tính $f(n) = nlogn$. VD QuickSort, MergeSort
  - Hàm bậc hai $f(n) = n^2$. VD Sắp xếp nổi bọt (bubble sort)
  - Hàm bậc ba $f(n) = n^3$.
  - Hàm mũ $f(n) = c^n$, c là hằng số >1.
  - Hàm giai thừa $f(n) = n!$

Tốc đô tăng và quan hê thống trị

- Quan hệ thống trị:

$$n! \gg c^n \gg n^3 \gg n^2 \gg n\log n \gg n \gg \log n \gg 1$$

- Giới hạn và quan hệ thống trị của các hàm
  - $f(n)$ thống trị $g(n)$ nếu $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$

VD.

$f(n) = n^2$ không thống trị $g(n) = 3n^2 + 5$ vì $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 3$

$f(n) = n^2$ thống trị $g(n) = n^{1.999999}$ vì $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$

$$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n\log n \gg n \gg \sqrt{n} \gg \log^2 n \gg \log n \gg \log n/\log\log n \gg \log\log n \gg 1$$

Mô hình O-lớn

- Chú ý:
  - Hai hàm trong cùng lớp hàm (cùng tốc độ tăng) vẫn khác nhau về thời gian thực hiện nếu tính chi ly
  - Khi lựa chọn, nên chọn thuật toán có tốc độ tăng chậm nhất có thể
- Mối quan hệ giữa các ký hiện tiệm cận O-lớn và lim nếu *f* và *g* là các hàm tiệm cận dương (từ *N* đến *R)*
  - Nếu $f(n) = \Theta(g(n))$, thì $f(n) = \Omega(g(n))$ và $f(n) = O(g(n))$
  - Nếu $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} = C < \infty$, thì $f(n) = \Theta(g(n))$
  - Nếu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, thì $f(n) = O(g(n))$
  - Nếu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, thì $f(n) = \Omega(g(n))$

Một số ví dụ

Đánh giá độ phức tạp về thời gian của thuật toán sắp xếp lựa chọn theo O-lớn trong trường hợp tồi nhất

Đánh giá trong trường hợp tồi nhất

- Mọi điều kiện với giá trị dữ liệu đầu vào luôn đúng để dẫn về trường hợp tồi nhất
- Nếu có nhiều lệnh return, break,.. cần quan tâm tới lệnh nào sẽ làm chương trình phải thực hiện lâu nhất

Thuật toán sắp xếp lựa chọn – Selection Sort

```
selection_sort(int s[], int n)
{
    int i,j; /* counters */
    int min; /* index of minimum */
    for (i=0; i<n; i++) {
       min=i;
       for (j=i+1; j<n; j++)
          if (s[j] < s[min]) min=j;
       swap(&s[i],&s[min]);
    }
}
```

Một số ví dụ

```
selection_sort(int s[], int n)
{
    int i,j; /* counters */
    int min; /* index of minimum */
    for (i=0; i<n; i++) {
       min=i;
       for (j=i+1; j<n; j++)
          if (s[j] < s[min]) min=j;
       swap(&s[i],&s[min]);
    }
}
```

- Lệnh được lặp lại nhiều nhất (lệnh cơ sở):
  - là lệnh gán min=j (giả sử đk của lệnh if luôn đúng)
  - Đếm số lần thực hiện lệnh cơ sở

Phân tích trong trường hợp tồi nhất

- Vòng lặp ngoài lặp n lần (từ 0 tới n-1)
- Vòng lặp trong lặp n lần ứng với mỗi lần lặp của vòng ngoài
- Vậy số lượng bước (thời gian) cần thực hiện trong trường hợp tồi nhất là $n \times n \rightarrow \mathrm{O}(n^2)$
- Hàm có tốc độ tăng bậc 2: ***Khi kích thước đầu vào tăng gấp 2 lần thì thời gian thực hiện tăng cỡ 4 lần***

Một số ví dụ

```
selection_sort(int s[], int n)
{
    int i,j; /* counters */
    int min; /* index of minimum */
    for (i=0; i<n; i++) {
       min=i;
       for (j=i+1; j<n; j++)
          if (s[j] < s[min]) min=j;
       swap(&s[i],&s[min]);
    }
}
```

- **Phân tích chi tiết hơn**
  - $i = 0$ lệnh if lặp $n - 1$ lần (từ 1 tới n-1)
  - $i = 1$ lệnh if lặp $n - 2$ lần (từ 2 tới n-1)

    …
  - $i = n - 2$ lệnh if lặp 1 lần (từ n-1 tới n-1)
  - $i = n - 1$ lệnh if lặp 0 lần
- Số lần lặp của if sẽ là

$$T(n) = (n-1) + (n-2) + \cdots + 1 + 0 = (n-1)\,n/2$$

- Vậy $T(n) = \Theta(n^2)$

Một số ví dụ

- Đánh giá độ phức tạp về thời gian của thuật toán **sắp xếp chèn – Insertion Sort** trong trường hợp tồi nhất theo O-lớn

```
for (i=1; i<n; i++) {
    j=i;
    while ((j>0) && (s[j] < s[j-1])) {
        swap(&s[j],&s[j-1]);
        j = j-1;
    }
}
```

- Lệnh SWAP có thời gian thực hiện là hằng số: O(1)
- Lệnh được lặp nhiều nhất là 2 lệnh bên trong while : lệnh cơ sở
- ***Đếm cả 2 lệnh cơ sở thì có thay đổi tốc độ tăng thời gian, hoặc dạng O-lớn?***

Một số ví dụ

```
for (i=1; i<n; i++) {
    j=i;
    while ((j>0) && (s[j] < s[j-1])) {
        swap(&s[j],&s[j-1]);
        j = j-1;
    }
}
```

- **Phân tích trong trường hợp tồi nhất**
  - $i = 1$ lệnh cơ sở lặp 1 lần
  - $i = 2$ lệnh cơ sở lặp 2 lần

    …
  - $i = n - 2$ lệnh cơ sở lặp $n - 2$ lần
  - $i = n - 1$ lệnh cơ sở lặp $n - 1$ lần
- Số lần lặp của lệnh cơ sở sẽ là

$$T(n) = 1 + 2 + \cdots + (n-2) + (n-1) = (n-1)n/2$$

Vậy $T(n) = \mathrm{O}(n^2)$

- ***Nếu đếm lệnh cơ sở là 2 lần dạng O-lớn cũng KHÔNG đổi → chỉ cần đếm 1 lần cho gọn***

Một số công thức hay dùng

- $\sum_{a}^{b} 1 = b - a + 1$
- $\sum_{i=1}^{n} i = 1 + 2 + .. + n = \frac{n(n+1)}{2}$
- $\sum_{j=i}^{n} j = i + (i+1) + .. + n = \sum_{i=1}^{n} i - \sum_{j=1}^{i-1} j = \frac{(n^2+n-i^2+i)}{2}$
- $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + .. + n^2 = \frac{(2n^3+3n^2+n)}{6}$
- $a^0 + a^1 + .. + a^n = \frac{(a^{n+1}-1)}{(a-1)}$ với $a \neq 1$
- $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ nếu $|x| < 1$
- $\sum_{i=1}^{n} ic^i = c + 2c^2 + .. + nc^n = \frac{\left((n-1)c^{n+1} - nc^n + c\right)}{(c-1)^2}$

Một số tính chất của hàm mũ, loga, giai thừa

- Hàm mũ:

$$a = b^{\log_b a} ; \quad \log_b a = 1/(\log_a b)$$

- Do đó, trong ký hiệu tiệm cận cơ số của log là không quan trọng:

$$O(\lg n) = O(\ln n) = O(\log n)$$

- Công thức Stirling:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

- Giai thừa và hàm mũ:

$$2^n < n! < n^n \quad \text{với } n > 5 ;$$

$$\log n! = \Theta(n \log n).$$

Một số tính chất của hàm log và hàm mũ

Geometric series

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

Harmonic series

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \log n$$

Other important formulae

$$\sum_{k=1}^{n} \log k \approx n \log n$$

$$\sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p \approx \frac{1}{p+1} n^{p+1}$$

$$\log xy = \log x + \log y \qquad \log^k n = (\log n)^k$$

$$\log \log n = \log(\log n) \qquad \log \frac{x}{y} = \log x - \log y$$

$$a^{\log_b^x} = x^{\log_b^a} \qquad \log_b^x = \frac{\log_a^x}{\log_a^b}$$

Bài tập

Bài 1. Sắp xếp các hàm sau theo thứ tự tăng dần độ phức tạp

a) $\sqrt{n+3}, \frac{n}{5}, \frac{3n}{17}, n\log\sqrt[3]{n+n^2}, n\log\log n, {}^{n^3}/_{(100+n)}, 2n\ln(n^2+3n), 5n+\sqrt[3]{6n+7}$

b) $\frac{1}{2n}, \frac{5n}{3n^2+1}, \frac{6+7n}{\sqrt[3]{n^2+n}}, \frac{n}{\log n+2}, \log(n+34), 2n+\frac{3n^2}{6n+\log n}, \sqrt{n+\log 3n}$

Bài tập

Bài 2. Tìm các hàm $g(n)$ đơn giản mà $f(n) = \Theta(g(n))$ cho các hàm $f(n)$ sau đây

a) $f(n) = 2^n + n^2$

b) $f(n) = n^2 + n\sqrt{n} + \log n$

c) $f(n) = \log 20^n + n^2$

d) $f(n) = (\frac{2}{3})^n + n^3$

Bài tập

Bài 3. Cho đoạn chương trình in ra xâu “Hello” sau

```
for (i=1; i<=n; i++)
   for (j=i; j<=2*i; j++)
      printf("Hello");
```

Đánh giá thời gian thực hiện của đoạn chương trình trên theo O-lớn trong trường hợp tồi nhất

Bài tập

- Đánh giá thời gian thực hiện của đoạn chương trình trên theo O-lớn trong trường hợp tồi nhất

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

```
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;
    }
}
```

```
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;
    }
}
```

Bài tập

Bài 4. Bạn có 25 con ngựa. Tại mỗi lần đua thì chỉ có thể cho tối đa 5 con ngựa tham gia. Bạn phải xác định 3 con ngựa là nhanh nhất, nhì và ba trong 25 con ngựa. Hãy tìm số lượng vòng đua tối thiểu để có thể thực hiện việc này.

Bài 5. Viết hàm thực hiện phép chia số nguyên mà không dùng các toán tử / hoặc *. Hãy tìm các thực hiện nhanh nhất.

Bài 6. Nhân hai số nguyên: Để nhân 2 số nguyên a, b ta thực hiện theo cách nhân lần lượt a với các chữ số trong b (có tính trọng số) sau đó tính tổng.
VD, $a = 120, b = 142$ thì $a \times b = 120 \times 100 + 120 \times 40 + 120 \times 2 = 17{,}040$. Phân tích độ phức tạp của thuật toán trên khi áp dụng để nhân hai số có n, m chữ số. Ở đây ta giả sử phép cộng hoặc nhân từng chữ số có thời gian thực hiện là hằng số(tức là $O(1)$)

Bài tập

Bài 7. Đánh giá độ phức tạp hàm sau theo O-lớn

```
int findmatch(char* p, char* t)
{
    int i, j; /* counters */
    int m, n; /* string lengths */
    m = strlen(p);
    n = strlen(t);
    for (i = 0; i <= (n - m); i = i + 1) {
        j = 0;
        while ((j < m) && (t[i + j] == p[j]))
            j = j + 1;
        if (j == m) return(i);
    }
    return(-1);
}
```

Bài tập

- Bài 8. Bài toán đoạn con trọng số lớn nhất
  - Cho dãy $a = (a_1, a_2, \ldots, a_n)$.
  - Một đoạn con của a được định nghĩa là dãy gồm một số phần tử liên tiếp $a_i, a_{i+1},\ldots,a_j$.
  - Trọng số của dãy con là tổng các phần tử của nó.
  - Tìm đoạn con có trọng số lớn nhất
- Ví dụ: a = **2, 11, -4, 13, -5, 2** khi đó dãy **2, 11, -4, 13** là dãy con lớn nhất

- Thuật toán vét cạn? $O(n^2)$
- Thuật toán tốt hơn? $O(n^2)$ và $O(n)$