HUST

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

# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## ET2100

ONE LOVE. ONE FUTURE.

# *Chương I :*
# *Tổng quan về CTDL và Giải thuật*

**Giảng viên: TS. Đỗ Thị Ngọc Diệp**
**Khoa Kỹ thuật Truyền thông – Trường Điện Điện Tử**

# Các nội dung chính

1. Mục đích và nội dung của CTDL
2. Các khái niệm cơ bản về CTDL và giải thuật
3. Ngôn ngữ diễn đạt giải thuật
4. **Thiết kế và Đánh giá giải thuật**

4.1. Thiết kế giải thuật

4.1.1. Phương pháp thiết kế từ trên xuống *(Top-down design)*

4.1.2. Phương pháp tinh chỉnh từng bước *(Refinement)*

4.2. Đánh giá giải thuật

4.2.1. Mục đích

4.2.2. Các vấn đề cần đánh giá

# 4.1. Thiết kế giải thuật

- Thiết kế cấu trúc chương trình cài đặt cho giải thuật.
  - Biến đổi từ *đặc tả giải thuật* thành một *chương trình* được viết bằng một ngôn ngữ lập trình cụ thể mà *có thể chạy tốt* trên máy tính
  - Biến đổi từ *mô tả giải thuật làm cái gì, các bước thực hiện những gì* thành *giải thuật được cài đặt như thế nào*, minh hoạ hoạt động cụ thể của giải thuật.

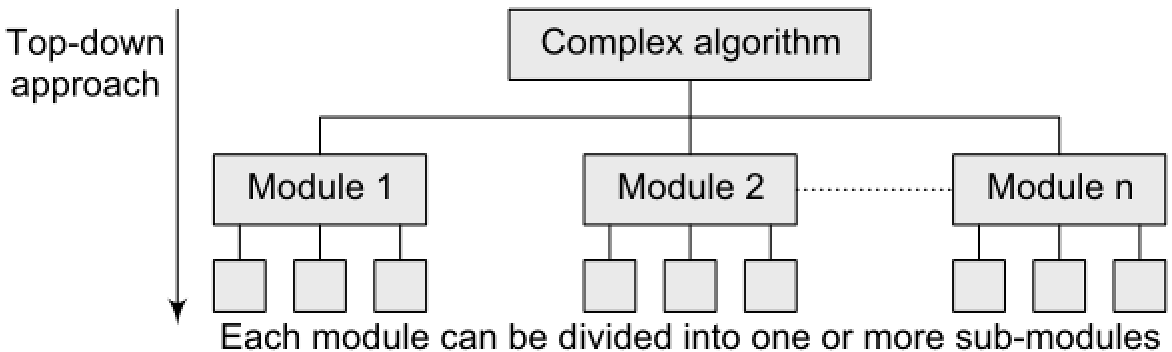
# 4.1. Thiết kế giải thuật

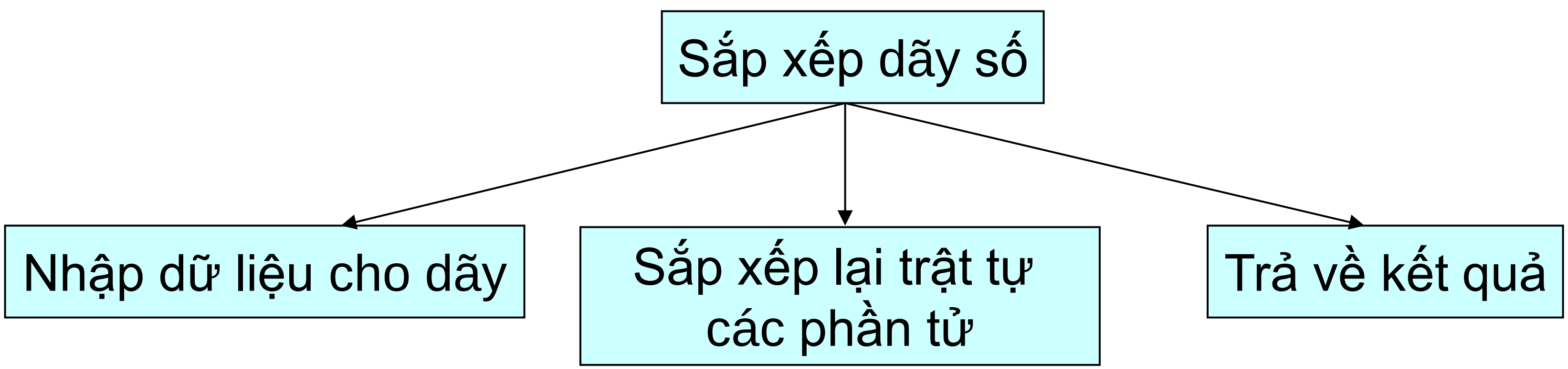
Thường được chia làm hai giai đoạn chính:

- **Thiết kế sơ bộ**:
  - Tìm hiểu các thành phần của giải thuật: gồm bao nhiêu thành phần cơ bản, mỗi thành phần đó làm cái gì, giữa các thành phần đó có mối liên quan gì.
  - Mỗi thành phần cơ bản được gọi là một *mô đun* của giải thuật.
  - Phương pháp thiết kế được sử dụng thường là phương pháp ***thiết kế từ trên xuống***
- **Thiết kế chi tiết**:
  - Cài đặt cụ thể các mô đun bằng một ngôn ngữ lập trình cụ thể.
  - Tiến hành ghép nối các mô dul để tạo thành một chương trình hoàn chỉnh
  - Phương pháp thiết kế sử dụng thường là phương pháp ***tinh chỉnh từng bước***

- ***Phương pháp mô đun hoá***
  - Dựa trên nguyên tắc *chia để trị*: chia giải thuật ban đầu thành các giải thuật con (mô đun), mỗi giải thuật con sẽ thực hiện một phần chức năng của giải thuật ban đầu
  - Quá trình phân chia này được lặp lại cho các mô đun  con cho đến khi các mô đun là đủ nhỏ để có thể giải trực tiếp
  - Kết quả phân chia này sẽ tạo ra một *sơ đồ phân cấp chức năng*
- Nên dùng khi giải thuật ban đầu quá dài, phức tạp; có một số chức năng độc lập có thể tách nhỏ.
- Lưu ý: điểm dừng phân chia

# Sơ đồ phân cấp chức năng



# 4.1.2. Thiết kế chi tiết - Phương pháp tinh chỉnh từng bước

- Hai nguyên tắc lựa chọn ngôn ngữ diễn đạt giải thuật thường mâu thuẫn, khó có ngôn ngữ nào mà có thể thỏa mãn được cả hai.
- *Phương pháp tinh chỉnh từng bước*: chuyển đổi từ đặc tả giải thuật bằng ngôn ngữ tự nhiên hay lưu đồ sang một đặc tả giải thuật bằng một ngôn ngữ lập trình cụ thể.
  - Quá trình chuyển đổi này gồm nhiều bước
  - Mỗi bước là một đặc tả giải thuật.

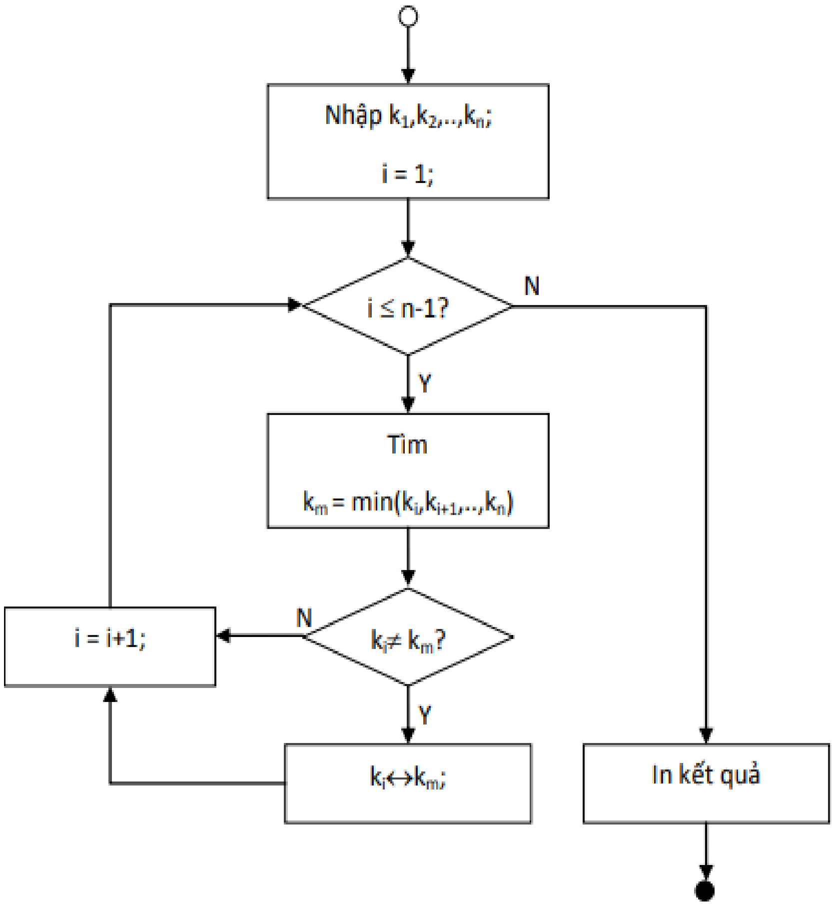
# Phương pháp tinh chỉnh từng bước

- Nguyên tắc:
  - Trong bước đầu tiên, ta có đặc tả giải thuật bằng ngôn ngữ tự nhiên hay lưu đồ giải thuật.
  - Trong các bước sau, tiến hành thay thế dần dần các thành phần được biểu diễn bằng ngôn ngữ tự nhiên của giải thuật bằng các thành phần tương tự được biểu diễn bằng ngôn ngữ lập trình đã chọn.
  - Lặp lại quá trình trên cho đến khi tạo ra một chương trình hoàn chỉnh có thể chạy được, thực hiện giải thuật yêu cầu

# Phương pháp tinh chỉnh từng bước

- Các bước chính:
  - Bước 1: Chọn một ngôn ngữ lập trình thích hợp (C)
  - Bước 2:
    - Tinh chỉnh dữ liệu: chuyển đổi dần từ các thông tin của bài toán sang ***các cấu trúc dữ liệu*** tương ứng, sau đó sang các ***cấu trúc lưu trữ*** thích hợp mà ngôn ngữ lập trình hỗ trợ
    - Tinh chỉnh thao tác: chuyển đổi dần từ mô tả các bước thực hiện trong giải thuật bằng ngôn ngữ tự nhiên sang các ***lệnh*** tương ứng của ngôn ngữ lập trình.
    - Hai quá trình thường được thực hiện một cách song song
  - Bước 2 được thực hiện nhiều lần cho đến khi thu được một chương trình hoàn chỉnh thực hiện giải thuật ban đầu.

- Nội dung cơ bản của GT
  - Dữ liệu:
    - Đầu vào: một dãy *n* số $k_1, k_2, ..., k_n$ hữu hạn
    - Đầu ra: một hoán vị của dãy ban đầu và được sắp xếp theo trật tự tăng dần.
  - Thao tác:
    - Nhập dãy vào.
    - Sắp xếp dãy số theo trật tự tăng dần: sử dụng giải thuật sắp xếp chọn
    - Đưa kết quả ra.

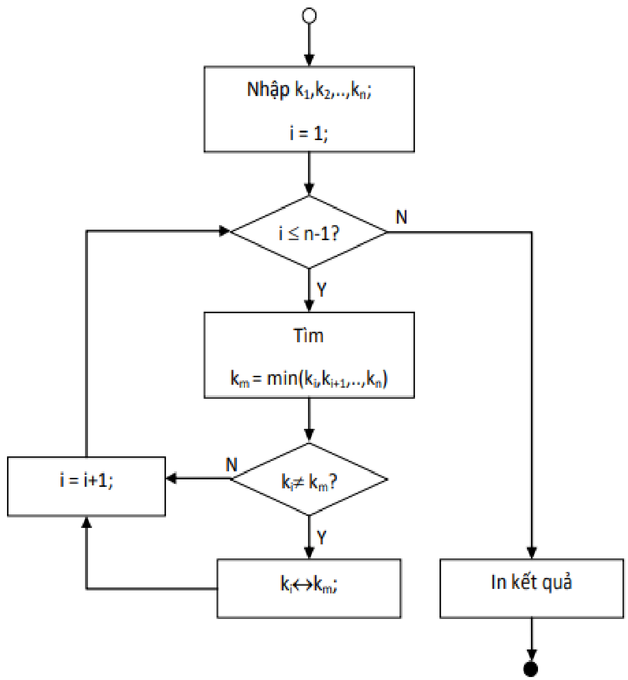


- Tinh chỉnh dữ liệu

| TT | Thông tin của giải thuật | Dữ liệu trong chương trình |
|---|---|---|
| 1 | Dãy số $k_1, k_2, \ldots, k_n$; | Mảng số thực: `float k[N] ;`<br>Trong đó N là một hằng số cần phải được khai báo trước. |
| 2 | Đếm số bước $i$; | Biến chạy kiểu nguyên: `int i ;` |
| 3 | Lưu lại chỉ số phần tử nhỏ nhất $m$; | Biến kiểu nguyên: `int m ;` |

# Bài toán sắp xếp một dãy số theo trật tự tăng dần

- Tinh chỉnh thao tác




| TT | Thao tác của giải thuật | Lệnh trong chương trình |
|---|---|---|
| 1 | Nhập dãy số $k_1, k_2, ..., k_n$; | Lệnh lặp cho N lần nhập: `for (i=0; i<N; i++)` `scanf("%f", &k[i]);` |
| 2 | Tìm $k_m = min(k_i, k_{i+1}, .., k_n)$; | Thêm biến chạy j: `int j;` Sau đó là đoạn xử lý tìm min: m = i; `for (j=i+1; j<N; j++)` `if (k[j]<k[m]) m = j;` |
| 3 | Hoán đổi $k_i$ và $k_m$; | Sử dụng thêm biến phụ: `float t;` Rồi hoán đổi: `t = k[i];` `k[i] = k[m];` `k[m] = t;` |
| 4 | Sắp xếp dãy $k_1, k_2, .., k_n$; | Lặp lại N-1 lần quá trình Tìm min và Hoán đổi ở giai đoạn 2 và 3 ở trên. `for (i=0; i<N; i++){` Tìm $k_m = min(k_i, k_{i+1}, .., k_n)$; Hoán đổi $k_i$ và $k_m$; `}` |
| 5 | In kết quả dãy số $k_1, k_2, .., k_n$ sau khi sắp xếp | Lệnh lặp cho N lần in ra màn hình: `for (i=0; i<N; i++)` `printf("%f", k[i]);` |

- Chương trình hoàn chỉnh

```
#include <stdio.h>
#define  N  5
void main()
{
    float  k[N] ;
    int i;
    int m;
    for (i=0; i<N; i++)
    {
        printf("k[%d]=",i);
        scanf("%f", &k[i]);
    }

    for (i=0; i<N; i++)
    {
        int j;
        m = i;
        for (j=i+1; j<N; j++)
            if (k[j]<k[m]) m = j;
        float t = k[i];
        k[i]=k[m];
        k[m]=t;
    }
    for (i=0; i<N; i++)
        printf("%f ", k[i]);
}
```

## 4.2.1. Mục đích:

- Tìm hiểu tính đúng đắn của giải thuật:
  - Giải thuật có đúng đắn hay không? Tức là nó cho ra kết quả đúng đối với mọi tập dữ liệu vào hay không.
- Tìm hiểu mức độ tài nguyên mà giải thuật sử dụng:
  - Tài nguyên tính toán gồm thời gian chạy và dung lượng bộ nhớ
  - => đánh giá tính thực tế của giải thuật.

# 4.2.2. Các vấn đề cần đánh giá

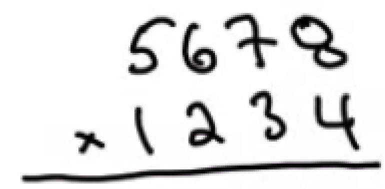
- Xác định tính đúng đắn của GT:
  - Chứng minh bằng quy nạp (Proof by Induction)
  - Chứng minh bằng phản ví dụ (Counterexample)
- Ước lượng kích thước bộ nhớ (độ phức tạp bộ nhớ)
  - Thường dựa trên kích thước dữ liệu của giải thuật
- Ước lượng thời gian thực hiện (độ phức tạp tính toán)
  - Thủ công: dùng đồng hồ đo
    - Sử dụng đồng hồ của máy tính: phụ thuộc vào cấu hình, cần chạy thực, tốn kém thời gian tài nguyên nếu GT không tốt
  - Lý thuyết: xác định độ phức tạp của GT

## Xác định độ phức tạp của GT

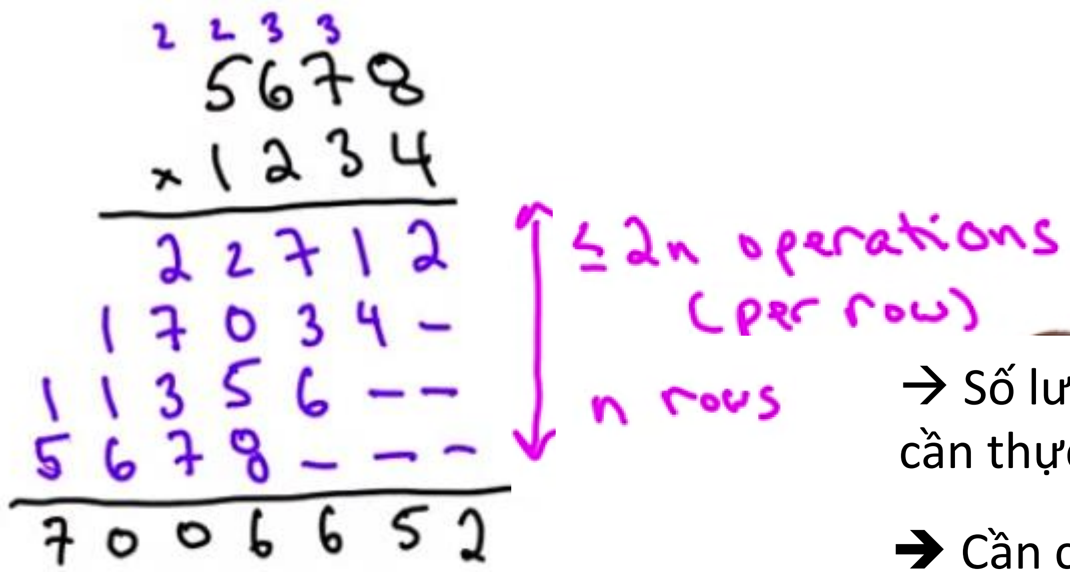
- Đánh giá về thời gian thực hiện
  - Đánh giá thời gian thực hiện giải thuật chỉ phụ thuộc vào *dữ liệu vào* chứ không phụ thuộc vào môi trường cài đặt
  - Giả sử giải thuật được thực thi trên một máy tính trừu tượng mà thời gian thực hiện mỗi thao tác/câu lệnh đơn là như nhau và bằng 1 đơn vị thời gian

=> Thời gian thực hiện của giải thuật = tổng số lệnh mà CT thực hiện

# Ví dụ




- Bài toán nhân 2 số nguyên
  - Đầu vào: 2 số nguyên *x*, *y* có *n* chữ số
  - Đầu ra: kết quả *x* * *y*
  - Cách giải:
    - Sử dụng các toán tử cơ bản: cộng 2 số và nhân hai số có 1 chữ số
    - => tính số lượng các phép toán cơ bản cần thực hiện ?




→ Số lượng các phép toán cơ bản cần thực hiện <= 2 * $n^2$ + 1 !

➔ Cần các thuật toán tốt hơn

# Xác định độ phức tạp của GT

- Đánh giá về thời gian thực hiện:
  - Quy kết quả tính toán thời gian thực hiện một giải thuật A nào đó về một hàm có dạng $T_A(n)$, với *n* đại diện cho kích thước *dữ liệu vào* của giải thuật *A* (kí hiệu ngắn gọn là T(n))

```
for(i=0;i<100;i++)
      statement      ;
```

T(n) = 1+(1+1+1)*n+1 = 3*n+2

```
for(i=0;i<100;i+=2)
      statement      ;
```

T(n)=1+(1+1+1)*n/2+1=3*n/2+2

```
for(i=0;i<10;i++)
      for(j=0; j<10;j++)
            statement      ;
```

T(n) = 1+(1+(3*n+2)+1)*n+1
= $3*n^2+4*n+2$

```
for(i=1;i<1000;i*=2)
      statement      ;
```

T(n) ~ $\log_2(n)$

```
for(i=0;i<10;i++)
      for(j=1; j<10;j*=2)
            statement      ;
```

T(n) ~ $n.\log_2(n)$

- Thông thường, chỉ xét 3 trường hợp T(n):
  - T/h tốt nhất $T_{tn}(n)$: trường hợp giải thuật yêu cầu số thao tác *ít nhất* ứng với kích thước dữ liệu *n*
  - T/h xấu nhất $T_{xn}(n)$: trường hợp giải thuật yêu cầu số thao tác *nhiều nhất* ứng với kích thước dữ liệu *n*
  - T/h trung bình $T_{tb}(n)$: trung bình cộng tất cả các trường hợp ứng với kích thước dữ liệu *n*

$$T_{tn}(n) \le T_{tb}(n) \le T_{xn}(n)$$

# Xác định độ phức tạp của GT

- Sử dụng
  - Khi muốn dự đoán tốc độ tăng của độ phức tạp khi kích thước đầu vào của bài toán tăng lên.
  - Khi có nhiều thuật toán, cần tìm thuật toán hiệu quả nhất.
- Khi *n* rất lớn, có thể lựa chọn thành phần lớn nhất trong biểu thức của T(n) để biểu diễn cho độ lớn của GT (*magnitude of algorithm*)
  - Thành phần này được gọi là O lớn

*"what happens for very large values of n"*

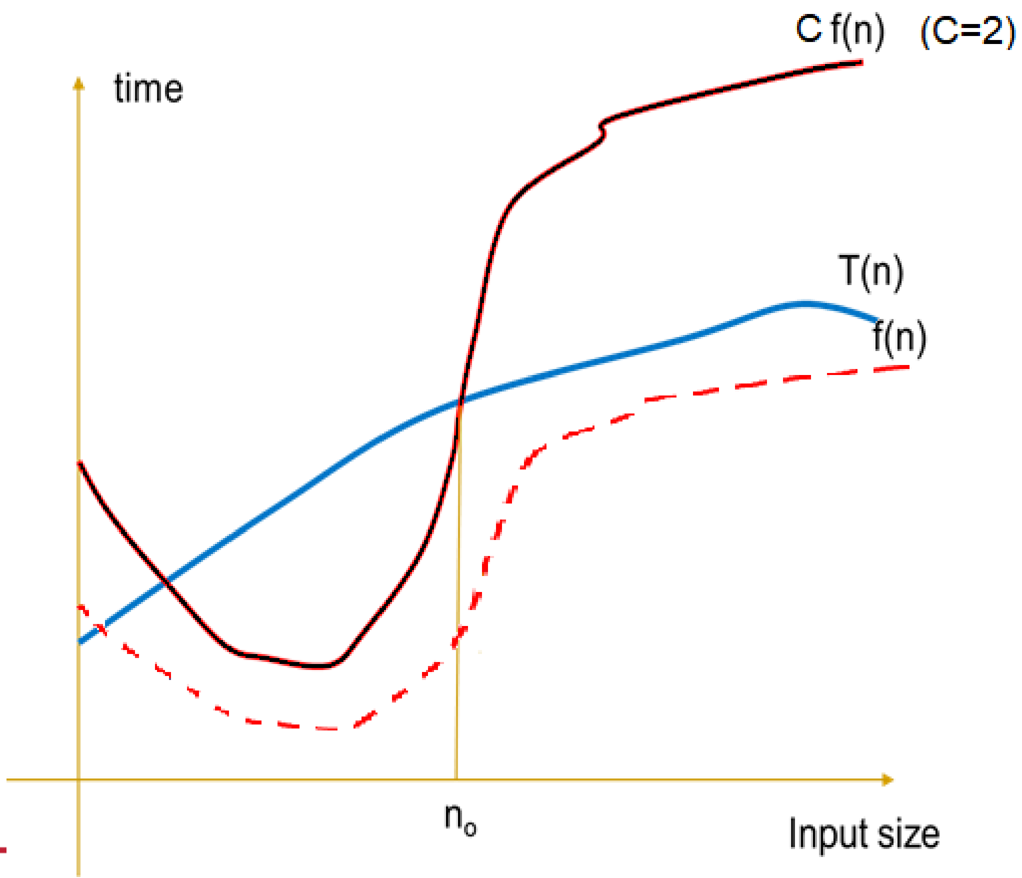
# Khái niệm O (ô lớn)

- Cho n là một số nguyên không âm, T(n), f(n) ≥ 0

  Ta nói **T(n) là O (f(n))** nếu và chỉ nếu

  $\exists$ hằng số C dương và $n_0$: $\forall\ n \geq n_0$ thì $T(n) \leq C.f(n)$

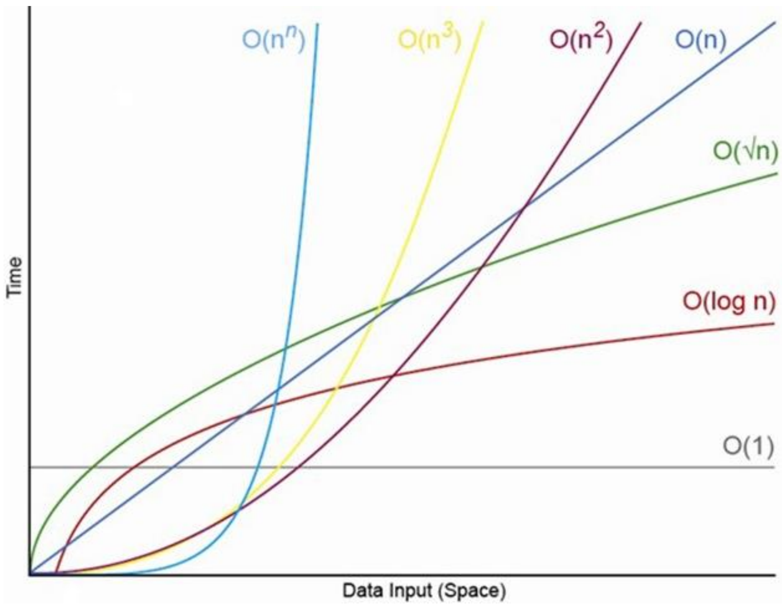
- Ý nghĩa:
  - Với lượng lớn dữ liệu, T(n) có tăng cũng không vượt quá C.f(n)
  - với $\forall\ n \geq n_0$, C.f(n) là tiệm cận trên của T(n).
  - Hằng số C phụ thuộc vào ngôn ngữ, CPU, bộ nhớ v.v.



f(n) thường đưa về các dạng sau:

$1, \log_2(n), n, n.\log_2(n), n^2, n^3, n^k, 2^n, k^n$.

- Hằng số: O(1)
- Tuyến tính: $O(n)$
- Bậc hai: $O(n^2)$
- Đa thức: $O(n^k)$, $k \geq 1$
- Hàm mũ: $O(a^n)$, $n > 1$
- Logarith: $O(\log n)$
- Giai thừa: $O(n!)$




$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2)$
$< O(n^3) < O(2^n) < O(3^n) < O(n!) < O(n^n)$

- Ví dụ: cho **T(n) = 3n**
  - Tìm được 1 vài hàm f(n)
  - f(n) = n: **T(n) = O (n)**
    vì với C=3 và $n_0$=0, ta có $\forall$ n $\geq$ 0 → 3n $\leq$ 3.n
  - f(n) = $n^2$: **T(n) = O ($n^2$)**
    vì với C=1 và $n_0$=3, ta có $\forall$ n $\geq$ 3 → 3n $\leq$ 1.$n^2$

  - Chọn f(n) nhỏ nhất và đơn giản nhất thỏa mãn → f(n) = n
  - Khi đó f(n) được gọi là **hàm độ lớn** hay **độ phức tạp** (hay cấp độ so sánh, hay cấp độ thời gian thực hiện) → Ký hiệu **T(n) = O (n)**

# Tính chất của O lớn

- $O(Cf(n)) = O(f(n))$, C: hằng số
- $O(C) = O(1)$, C: hằng số

- Tính chất 1:

  nếu $T(n) = O(f(n))$ và $f(n) = O(g(n))$ thì $T(n) = O(g(n))$

- Tính chất 2:

  nếu $T(n) = T1(n) + T2(n)$

  và $T1(n) = O(f1(n))$, $T2(n) = O(f2(n))$

  thì $T(n) = O(\max(f1(n), f2(n)))$

- Tính chất 3:

  nếu $T(n) = T1(n).T2(n)$

  và $T1(n) = O(f1(n))$, $T2(n) = O(f2(n))$

  thì $T(n) = O(f1(n).\ f2(n))$

# Ví dụ

- $T(n)=10$ => $T(n)=O(1)$
- $T(n)=3n+5$ => $T(n)=O(n)$
- $T(n)=3n^2+5n+4$ => $T(n)=O(n^2)$
- $T(n)=n+\sqrt{n}$ => $T(n)=O(n)$
- $T(n)=2^n+n^2$ => $T(n)=O(2^n)$
- $T(n)=8n^2\log n+5n^2+n$ => $T(n)=O(n^2\log n)$

*"Drop lower-order terms and constant factors"*

# Hệ quả

- Hệ quả 1: Độ phức tạp của một lệnh rẽ nhánh bằng độ phức tạp của nhánh có độ phức tạp cao nhất.
- Hệ quả 2: Độ phức tạp của một lệnh tuần tự bằng tổng độ phức tạp của các lệnh thành phần.
- Hệ quả 3: Độ phức tạp của một lệnh lặp bằng tổng độ phức tạp của tất cả các lần lặp

# Ví dụ

```
for(i=0;i<100;i++)
        statement block;
```

$T(n)=O(n)$

```
for(i=0;i<100;i+=2)
        statement block;
```

$T(n)=O(n)$

```
for(i=0;i<10;i++)
        for(j=0; j<10;j++)
                statement block;
```

$T(n)=O(n^2)$

```
for(i=1;i<1000;i*=2)
        statement block;
```

$T(n) = O(\log_2 n)$

```
for(i=0;i<10;i++)
        for(j=1; j<10;j*=2)
                statement block;
```

$T(n) = O(n.\log n)$

# Cách tính độ phức tạp

- Xác định đầu vào: thường ký hiệu là *n*
- Cách 1: dùng cho tất cả các loại chương trình
  - Tính thời gian thực hiện T(n) cho toàn bộ chương trình
  - Xác định O(f(n)) từ T(n)
- Cách 2: không áp dụng cho chương trình đệ quy
  - Chia chương trình nhiều đoạn nhỏ
  - Tính T(n) và O(f(n)) cho từng đoạn
  - Áp dụng quy tắc cộng, quy tắc nhân để có O(f(n)) cho cả chương trình

# VÍ DỤ:
## Xác định độ phức tạp của giải thuật tìm kiếm đơn giản

- Bài toán: Cho mảng *A* gồm *n* phần tử số nguyên, 1 giá trị nguyên *t*. Kiểm tra xem mảng *A* có chứa giá trị nguyên *t* không ?

**Algorithm 1**

```
for i = 1 to n do
    if A[i] == t then
        Return TRUE
Return FALSE
```

Thời gian thực hiện ?

A) O(1)

B) O(log n)

C) O(n)

D) O($n^2$)

THtn: O(1)

THxn: O(N)

=> THtb: O(N)

# Xác định độ phức tạp của giải thuật tìm kiếm đơn giản

- Bài toán: Cho mảng *A*, *B* gồm *n* phần tử số nguyên, 1 giá trị nguyên *t*. Kiểm tra xem mảng *A*, *B* có chứa giá trị nguyên *t* không ?

**Algorithm 2**

```
for i = 1 to n do
    if A[i] == t then
        Return TRUE
for i = 1 to n do
    if B[i] == t then
        Return TRUE
Return FALSE
```

Thời gian thực hiện ?

A) O(1)

B) O(log n)

C) O(n)

D) O($n^2$)

THtn: O(1)

THxn: O(N)

=> THtb: O(N)

# VÍ DỤ:
## Xác định độ phức tạp của giải thuật tìm kiếm đơn giản

- Bài toán: Cho mảng *A*, *B* gồm *n* phần tử số nguyên. Kiểm tra xem mảng *A*, *B* có chứa giá trị nào chung không ?

**Algorithm 3**

```
for i = 1 to n do
    for j = 1 to n do
        if A[i] == B[j] then
            Return TRUE
Return FALSE
```

Thời gian thực hiện ?

A) O(1)

B) O(log n)

C) O(n)

D) $O(n^2)$

THtn: O(1)

THxn: $O(N^2)$

=> THtb: $O(N^2)$

# VÍ DỤ:
# Xác định độ phức tạp của giải thuật sắp xếp

```
01. #include <stdio.h>
02. #define  N  5
03. void main()
04. {
05.     float  k[N] ;
06.     int i;
07.     int m;
08.     for (i=0; i<N; i++)
09.     {
10.         printf("k[%d]=",i);
11.         scanf("%f", &k[i]);
12.     }
13.     for (i=0; i<N; i++)
14.     {
15.         int j;
16.         m = i;
17.         for (j=i+1; j<N; j++)
18.             if (k[j]<k[m]) m = j;
19.         float t = k[i];
20.         k[i]=k[m];
21.         k[m]=t;
22.     }
23.     for (i=0; i<N; i++)
24.         printf("%f ", k[i]);
25. }
```

- Coi như thời gian thực hiện mỗi lệnh đơn là như nha

| | |
|---|---|
| Dòng lệnh từ 5-7 (tuần tự) | 3 |
| Dòng lệnh từ 8-12 (lặp *for*) | 2.N |
| Dòng lệnh từ 23-24 (lặp *for*) | N |

- Dòng 13-22:
  - Lệnh *17-18*

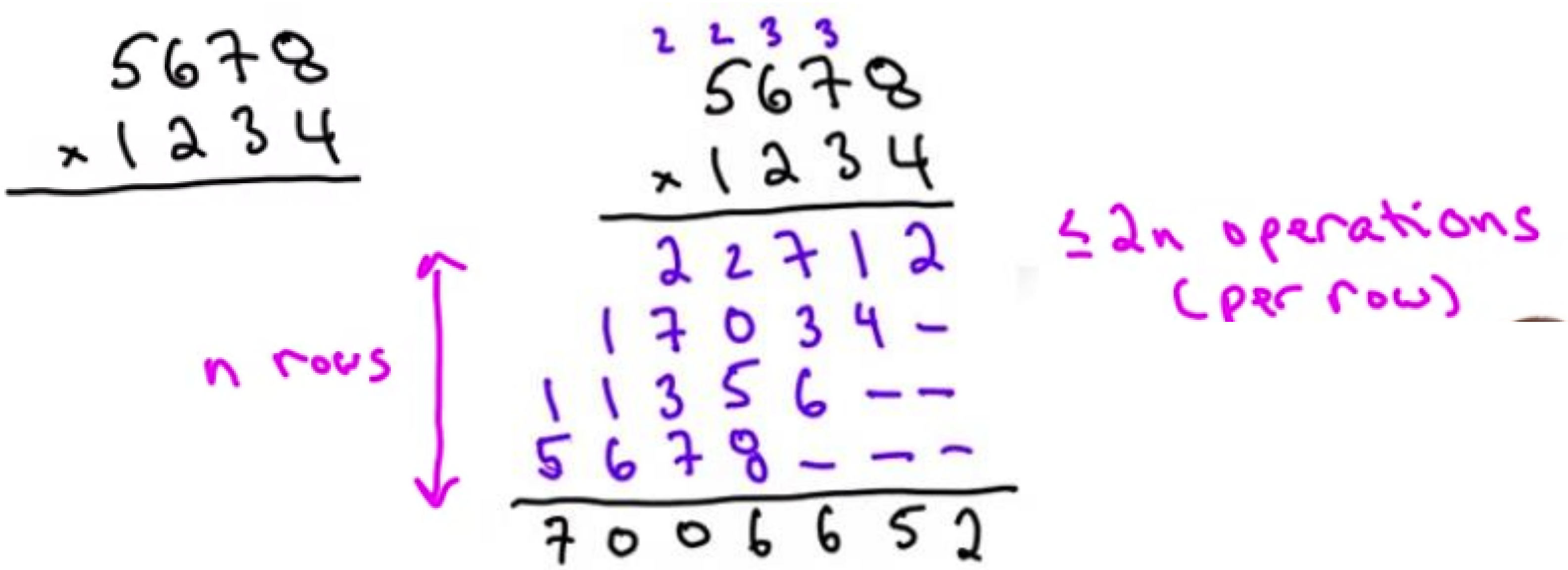
| Độ pt | for trong | kết hợp for ngoài |
|---|---|---|
| $TH_{tn}$ | N-i-1 | $\sum_{i=0}^{N-1}(N-i-1) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$ |
| $TH_{xn}$ | 2(N-i-1) | $N(N-1)$ |
| $TH_{tb}$ | 3(N-i-1)/2 | 3N(N-1)/4 |

- Đoạn lệnh 13-22
  - THtn: $N(N-1)/2+5N = N(N+9)/2 = O(N^2)$
  - THxn: $N(N-1)+5N = N(N+4) = O(N^2)$
  - THtb: $3N(N-1)/4+5N = O(N^2)$

# Bài tập

- Bài tập 1:
  - Trình bầy một giải thuật tìm phần tử lớn nhất trong một dãy số.
  - Xác định độ phức tạp của giải thuật sử dụng khái niệm O (hãy xét cả ba trường hợp: tốt nhất, xấu nhất và trung bình).
  - Viết một chương trình cài đặt cho giải thuật trên
- Bài tập 2:
  - Trình bầy một giải thuật kiểm tra một số nguyên N cho trước có phải là số nguyên tố hay không.
  - Xác định độ phức tạp của giải thuật sử dụng khái niệm O.
  - Viết một chương trình cài đặt cho giải thuật trên.

# Ví dụ




Số lượng các phép toán cơ bản cần thực hiện <= C * $n^2$

CAN WE DO BETTER?

# Ví dụ

- Thuật toán nhân nhanh Karatsuba

$$x = \underbrace{56}_{a}\underbrace{78}_{b}$$
$$y = \underbrace{12}_{c}\underbrace{34}_{d}$$

Step 1: compute $a \cdot c = 672$

Step 2: compute $b \cdot d = 2652$

Step 3: compute $(a+b)(c+d) = 134 \cdot 46 = 6164$

Step 4: compute ③ − ② − ① $= 2840$

Step 5:

$$\begin{array}{r} 6720000 \\ 2652 \\ 284000 \\ \hline 7006652 \end{array}$$

# Ví dụ

- Thuật toán nhân nhanh Karatsuba

Write $x = 10^{\frac{n}{2}} a + b$ and $y = 10^{\frac{n}{2}} c + d$

where a, b, c, d are $\frac{n}{2}$-digit numbers.

[example: $a = 56, b = 78, c = 12, d = 34$]

Then: $x \cdot y = (10^{\frac{n}{2}} a + b) \cdot (10^{\frac{n}{2}} c + d)$

$$= 10^n ac + 10^{\frac{n}{2}} (ad + bc) + bd \qquad (*)$$

Idea: recursively compute ac, ad, bc, bd, then compute (*) in the straightforward way.

- Thuật toán nhân nhanh Karatsuba

$$x = \underbrace{56}_{a}\,\underbrace{78}_{b}$$
$$y = \underbrace{12}_{c}\,\underbrace{34}_{d}$$

Step 1: compute $a \cdot c = 672$

Step 2: compute $b \cdot d = 2652$

Step 3: compute $(a+b)(c+d) = 134 \cdot 46 = 6164$

Step 4: compute ③ − ② − ① $= 2840$

Step 5:

$$\begin{array}{r} 6720000 \\ 2652 \\ 284000 \\ \hline 7006652 \end{array}$$

# Ví dụ

- Thuật toán nhân nhanh Karatsuba

  Write $x = 10^{n/2}a + b$ and $y = 10^{n/2}c + d$

  Where a,b,c,d are n/2-digit numbers.

  [example: a=56, b=78, c=12, d=34]

  Then $x.y = (10^{n/2}a + b)(10^{n/2}c + d)$

  $$= (10^n ac + 10^{n/2}(ad + bc) + bd \qquad (*)$$

- Lặp lại thuật toán nhân nhanh cho *ac*, *ad*, *bc*, *bd*, rồi tính (*)
  - Hoặc tính ac, bd, (a+b)(c+d)
- TH cơ sở: các toán hạng nhân chỉ còn 1 chữ số.