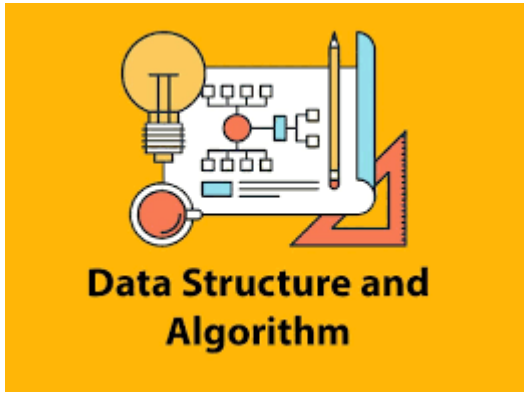
# CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

## NGÀY 1.1: ĐỘ PHỨC TẠP THUẬT TOÁN




Giảng viên: Th.S Bùi Văn Kiên

# Nội dung

- Phân tích và thiết kế giải thuật
  - Thiết kế giải thuật
  - Phân tích giải thuật
- Khái niệm độ phức tạp của thuật toán
- Kí hiệu Big-O Notation
- Các phân lớp giải thuật theo độ phức tạp

# Tổng quan một chương trình

- Hai yếu tố tạo nên một chương trình máy tính
  - Cấu trúc dữ liệu
  - Giải thuật

**Cấu trúc dữ liệu + Giải thuật = Chương trình**

# Đặc trưng của giải thuật

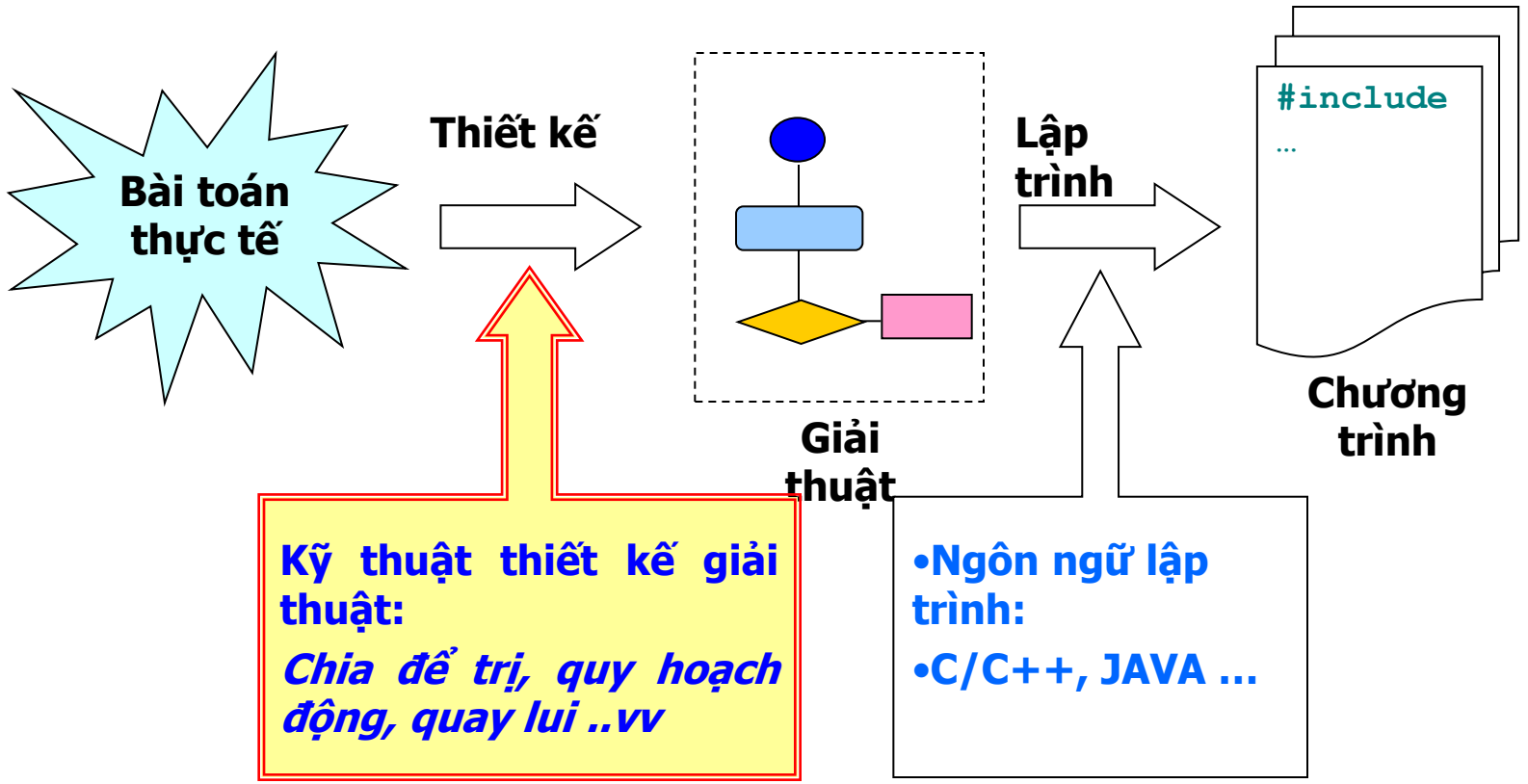
- Định nghĩa: là một trình tự các thao tác trên một số đối tượng nào đó, sao cho sau một số hữu hạn bước thực hiện ta đạt được kết quả mong muốn với input đầu vào.
- Cách trình bày:
  - Giả mã (Pseudo – code)
  - Sơ đồ thuật toán
  - Full code

# Đặc trưng của giải thuật

- Tính xác định
- Tính dừng (hữu hạn)
- Tính đúng đắn
- Tính phổ dụng
- Tính khả thi

# Thiết kế giải thuật

Từ bài toán đến chương trình



# Thiết kế giải thuật

- Với một vấn đề đặt ra, làm thế nào để đưa ra thuật toán giải quyết nó?
- Chiến lược thiết kế:
  - Tham lam (greedy method)
  - Chia-để-trị (divide-and-conquer)
  - Quy hoạch động (dynamic programming)
  - Quay lui (backtracking)
  - …

# Phân tích Giải thuật

- Khi một giải thuật được xây dựng, các yêu cầu đặt ra
  - Yêu cầu về tính đúng đắn của giải thuật
  - Tính đơn giản của giải thuật.
  - Yêu cầu về không gian (bộ nhớ)
  - Yêu cầu về thời gian

# Thời gian chạy của chương trình

Thời gian chạy của chương trình phụ thuộc vào các yếu tố:

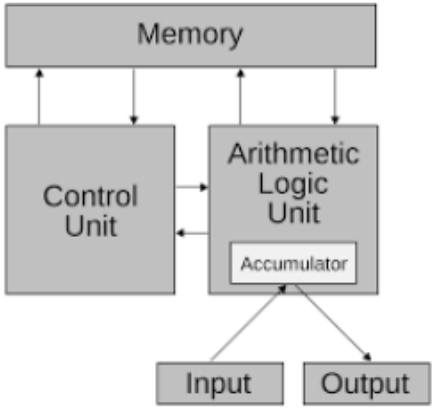
- Độ phức tạp của thuật toán
- Dữ liệu vào
- Tốc độ xử lí của máy tính (phần cứng)
- Trình biên dịch

# Thời gian chạy của chương trình

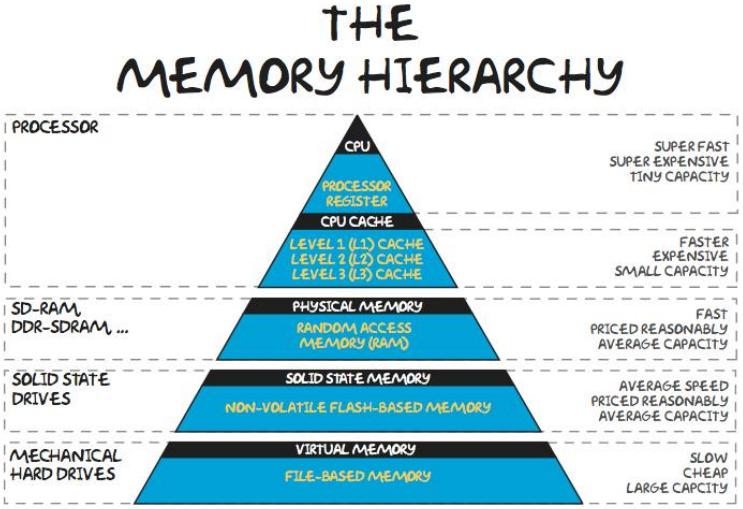
- Tốc độ xử lí của máy tính

# Thời gian chạy của chương trình

- Kiến trúc máy tính




- Phân cấp bộ nhớ



# Thời gian chạy của chương trình

- Trình biên dịch được sử dụng:

# Độ phức tạp thuật toán

**Vấn đề:**

- Tính toán thời gian chạy của chương trình là khó khăn
- Đôi khi phụ thuộc vào những yếu tố ngay cả người lập trình không biết hoặc không kiểm soát được
- Làm thế nào để đo được thời gian chạy mà không bị phụ thuộc những yếu tố này?

# Độ phức tạp thuật toán

**Tính toán xấp xỉ:**

- Tính toán chính xác thời gian chạy của thuật toán là khó khăn vì phụ thuộc nhiều yếu tố.
- Cần tìm giải pháp dễ tính toán hơn, có thể không chính xác tuyệt đối nhưng lại dễ tính toán hơn mà vẫn phản ánh được tốc độ tính toán.
- Tính toán tương đối dựa trên kích thước dữ liệu vào
  - Ví dụ: Thời gian chạy là n so với 1000*n có vẻ chênh lệch nhưng nếu như so sánh với n^2 với input lớn thì 1000*n vẫn cho thời gian tốt hơn n^2.

# Độ phức tạp thuật toán

**Phép tính cơ bản (primitive operations): 1 đơn vị**

- Phép gán một số
- Tăng giá trị của một biến
- So sánh 2 số
- Cộng/trừ 2 số
- Truy cập tới một phần tử của mảng

```
x = 0;
for (int i=0; i<n; i++)
    x = x + 1;
```

Số lượng phép tính cơ bản: 5n+2

- Gán x = 0 → 1 phép tính
- Gán i = 0 → 1 phép tính
- So sánh i < n → n phép tính
- i++ (i = i+1) → 2n phép tính
- x = x + 1 → 2n phép tính

# Độ phức tạp thuật toán

**Ví dụ:** Tìm phần tử lớn nhất của dãy số A[] với n phần tử.

(Chỉ số bắt đầu từ 0, 1, 2, …, n-1)

**Giả mã:**

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

```
int currentMax = A[0];
int i = 1;
while(i < n) {
   if(A[i] > currentMax) {
      currentMax = A[i];
   }
   i = i+1;
}
return currentMax;
```

# Độ phức tạp thuật toán

**Số lượng phép toán cần thực hiện:**

| | Số phép tính |
|---|---|
| `int currentMax = A[0];` | **2** |
| `int i = 1;` | **1** |
| `while(i < n) {` | **n-1** |
| `  if(A[i] > currentMax) {` | **2(n-1)** |
| `  currentMax = A[i];` | **[0, 2(n-1)]** |
| `  }` | |
| `  i = i+1;` | **2(n-1)** |
| `}` | |
| `return currentMax;` | **1** |

Trường hợp tốt nhất: A[0] là phần tử lớn nhất → 5n-1

Trường hợp xấu nhất: A[n-1] là phần tử lớn nhất → 7n-3

# Độ phức tạp thuật toán

**Ví dụ:** Tìm phần tử lớn nhất của dãy số A[] với n phần tử. (Chỉ số bắt đầu từ 0, 1, 2, …, n-1)

Số phép toán cần thực hiện T(n):

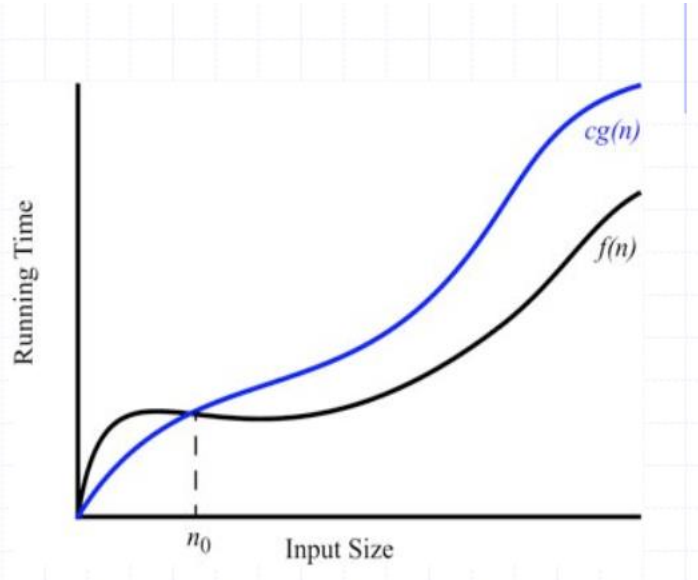
$$5n-1 <= T(n) <= 7n-3$$

→ Cần đưa ra một khái niệm xấp xỉ để tính toán độ phức tạp của thuật toán, đó là Big-O Notation.

# Độ phức tạp thuật toán – Big Oh

- Ký hiệu O (Big-Oh):
  - Cho hàm f(n) và g(n), ta nói:
  - f(n) = O(g(n)), nếu tồn tại các hằng số dương c và $n_0$ sao cho f(n) ≤ cg(n) khi n ≥ $n_0$.

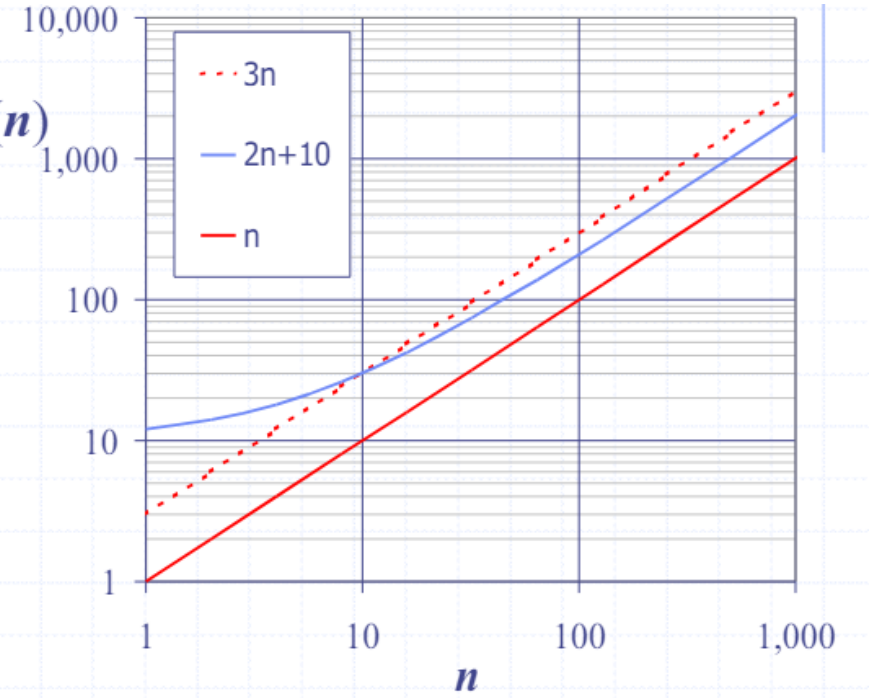
Ký hiệu này dùng để chỉ chặn trên của một hàm.
Ý nghĩa: Tốc độ tăng của hàm f(n) không lớn hơn hàm g(n)



# Độ phức tạp thuật toán – Big Oh

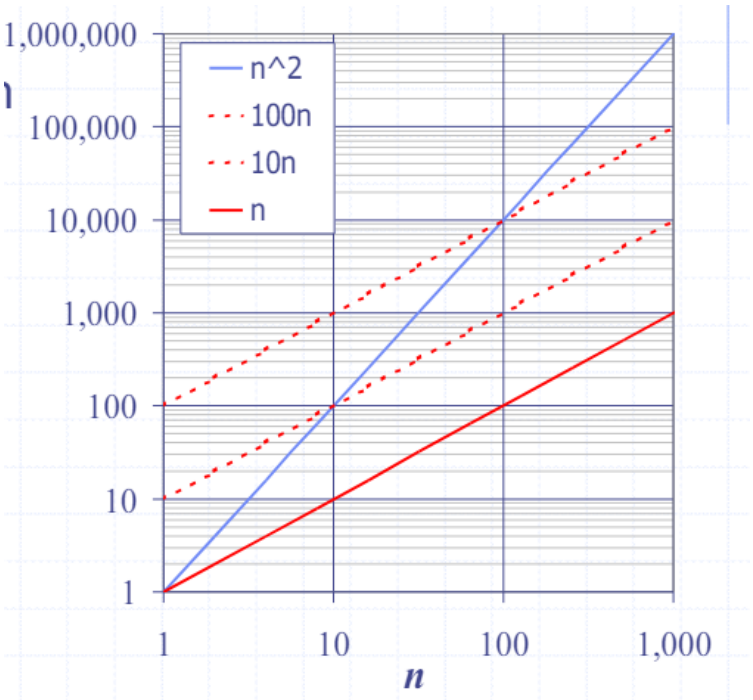
- Ký hiệu O (Big-Oh):
  - f(n) = O(g(n)), nếu tồn tại các hằng số dương c và $n_0$ sao cho f(n) ≤ cg(n) khi n ≥ $n_0$.

- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$



# Độ phức tạp thuật toán – Big Oh

- Ký hiệu O (Big-Oh), ví dụ:
  - Ví dụ hàm số $n^2$ sẽ không phải là O(n) vì không tồn tại hằng số c thỏa mãn $n^2 \leq cn$



# Độ phức tạp thuật toán – Big Oh

- Ký hiệu O (Big-Oh), ví dụ:
  - 7n-2 là O(n)
  - $3n^3 + 20n^2 + 10$ là $O(n^3)$
  - 3 log n + 5 là O(log n)

# Độ phức tạp thuật toán – Big Oh

Gọi $T_1(n)$ và $T_2(n)$ là thời gian thực hiện của hai giai đoạn chương trình $P_1$ và $P_2$ mà $T_1(n) = O(f(n))$; $T_2(n) = O(g(n))$

- **Quy tắc cộng:**

Thời gian thực hiện đoạn $P_1$ rồi $P_2$ tiếp theo là:

$$T1(n) + T2(n) = O(max(f(n), g(n))).$$

- **Quy tắc nhân:**

Thời gian thực hiện $P_1$ và $P_2$ lồng nhau là:

$$T_1(n)T_2(n) = O(f(n)*g(n))$$

# Độ phức tạp thuật toán – Ví dụ

- Các phép gán, đọc, viết, so sánh, if else, switch, return là các phép toán sơ cấp

→ có thời gian thực hiện là: O(1)

- Các vòng lặp for, while, do while, nếu duyệt hết n phần tử thì độ phức tạp là O(n)

# Độ phức tạp thuật toán – Ví dụ

Case1:
```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        sum += i*j;
```
$O(n^2)$

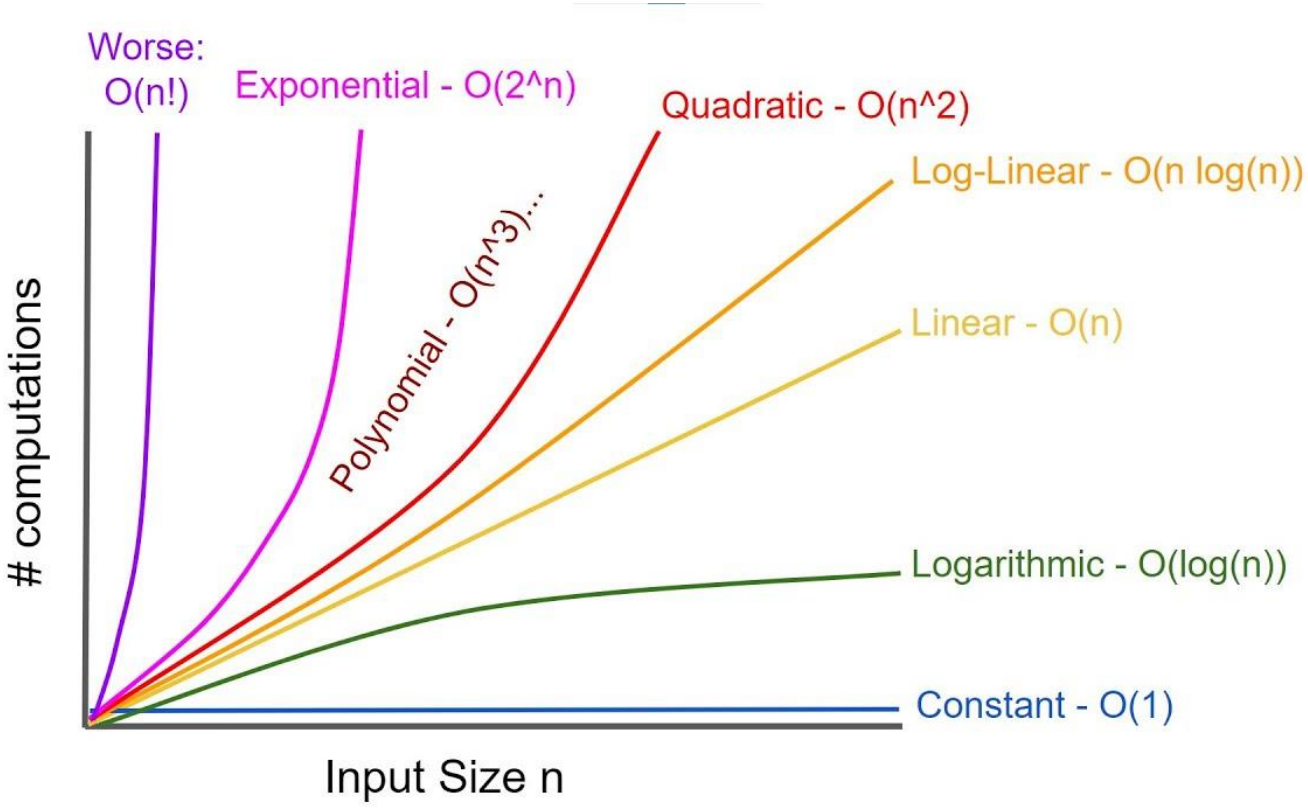
Case 2:
```
for (i=0; i<n; i++)
    for (j=0; i<i; j++)
        for (k=0; k<j; k++)
            ans++;
```
$O(n^3)$

Case 3:
```
for (int i=0; i<n-1; i++)
    for (int j=0; j<i; j++)
        ans += 1;
```
$O(n^2)$

# Độ phức tạp thuật toán – Phân lớp

- Độ phức tạp
  - O(1) độ phức tạp hằng số
  - O(logn) độ phức tạp logarit
  - O(n) độ phức tạp tuyến tính
  - O(nlogn) độ phức tạp n logn
  - O(n^b) độ phức tạp đa thức
  - O(b^n) độ phức tạp mũ
  - O(n!) độ phức tạp giai thừa

# Độ phức tạp thuật toán – Phân lớp



# Độ phức tạp thuật toán – Phân lớp

Thời gian ước tính với máy tính có tốc độ 1Ghz:

| | $n$ | $n \log n$ | $n^2$ | $2^n$ |
|---|---|---|---|---|
| $n = 20$ | 1 sec | 1 sec | 1 sec | 1 sec |
| $n = 50$ | 1 sec | 1 sec | 1 sec | 13 day |
| $n = 10^2$ | 1 sec | 1 sec | 1 sec | $4 \cdot 10^{13}$ year |
| $n = 10^6$ | 1 sec | 1 sec | 17 min | |
| $n = 10^9$ | 1 sec | 30 sec | 30 year | |
| max $n$ | $10^9$ | $10^{7.5}$ | $10^{4.5}$ | 30 |

# Độ phức tạp thuật toán – Đánh giá

- Độ phức tạp tính toán của giải thuật trong các trường hợp
  - Xấu nhất
  - Tốt nhất
  - Trung bình

- Ví dụ: Thuật toán tìm kiếm tuần tự

```
int sequenceSearch(int x, int a[], int n){
        for (int i=0; i<n; i++) {
                if (x==a[i]) return i;
        }
        return -1;
}
```

# Độ phức tạp thuật toán – Đánh giá

- Độ phức tạp
  - Tốt nhất: phần tử đầu tiên là phần tử cần tìm, số lượng phép so sánh là 1

  → T(n) ~ O(1)
  - Xấu nhất: so sánh đến phần tử cuối cùng, số lượng phép so sánh là n

  → T(n) ~ O(n)
  - Trung bình: so sánh đến phần tử thứ i, cần i phép so sánh, vậy trung bình cần:

  (1+2+3+…+n)/n = n*(n+1)/(2*n)= (n+1)/2 = O(n)

Kết luận: T(n) ~ O(n)

# Kinh nghiệm luyện tập

Dựa trên đánh giá độ phức tạp tính toán:

- Với đề bài có giới hạn $N \leq 10^5$ hay $10^6$, nên sử dụng thuật toán có độ phức tạp O(N) hoặc O(N log N).
- Giới hạn $N \leq 2000$ hoặc 5000, có thể sử dụng thuật toán có độ phức tạp $O(N^2)$.
- Giới hạn $N \leq 200$ hoặc 300, có thể sử dụng thuật toán có độ phức tạp $O(N^3)$.
- Với giải thuật có độ phức tạp là O(K*N) với K là tham số nào đó, nếu ước lượng số câu lệnh lớn hơn $10^8$ thì nên suy nghĩ và lựa chọn giải pháp khác.

# QUESTIONS & ANSWERS