



# THUẬT TOÁN ỨNG DỤNG

---

Thuật toán và Phân tích Thuật toán



1. Thông tin chung về môn học
2. Thuật toán
3. Ví dụ đầu tiên
  1. Duyệt toàn bộ
  2. Duyệt toàn bộ nhưng tối ưu hơn
  3. Chia để trị
  4. Quy hoạch động
4. Phân tích thuật toán
  1. Độ tăng trưởng
  2. Phân tích thực nghiệm
5. Bài tập



Phần 1

# Thông tin chung về môn học



- Tên môn: Thuật toán Ứng dụng
  - Tiếng Anh: Application of Algorithms
- Số tín chỉ: 3 (30 lý thuyết + 15 thực hành)
- Nội dung chính:
  - Giới thiệu cấu trúc dữ liệu và thuật toán
  - Đệ quy, quay lui và nhánh cận
  - Các cách tiếp cận: tham lam, chia để trị, quy hoạch động
  - Đồ thị
  - \*Xử lý chuỗi
- Giảng viên: Trương Xuân Nam, khoa CNTT
- Email: [namtx@wru.vn](mailto:namtx@wru.vn) / [truongxuannam@gmail.com](mailto:truongxuannam@gmail.com)

# Tài liệu môn học và phần mềm học tập



- Tài liệu chính: bài giảng của giáo viên
- Phần mềm học tập: C/C++/Java/Python
  - Dùng ngôn ngữ lập trình nào cũng được, miễn là minh họa đúng tính chất của bài giải
  - Chấm tự động bằng phần mềm FineTest hoặc dịch vụ online
- Bài giảng, bài tập, mã nguồn, điểm số,... sẽ được đưa lên site <https://txnam.net> mục **BÀI GIẢNG**
  - Bài giảng và bài tập sẽ được đưa lên site trước giờ học
  - Trong giờ thực hành, FineTest tự động gửi bài cho sinh viên để làm, giáo viên sẽ không gửi cho lớp
  - Điểm quá trình sẽ được công bố trên website



- Lập trình được với C/C++, Java, Python hoặc C#
  - Vì chúng ta sẽ áp dụng kiến thức đó vào môn học
  - Lập trình được tức là có thể viết chương trình với ngôn ngữ đó dựa trên mô tả thuật toán
- Đã học:
  - Ngôn ngữ Lập trình
  - Cấu trúc Dữ liệu và Giải thuật
- Biết sử dụng email
  - Nộp bài tập vào email của thầy giáo: cần ghi rõ tên sinh viên, bài nộp là bài nào, của buổi bài tập số mấy
  - Có thể email cho thầy giáo để hỏi thêm các vấn đề về môn học
- **Chú ý: copy bài của bạn khác để nộp sẽ bị cấm thi**

# Đánh giá kết quả



- Điểm môn học:
  - Điểm quá trình: **50%**
  - Điểm thi cuối kỳ: **50%**
- Điểm quá trình:
  - Chuyên cần
  - Bài làm trên lớp, trong phòng lab
  - Bài tập về nhà (nộp qua email)
  - Bài kiểm tra
- Thi cuối kỳ:
  - Thi trên máy tính bằng FineTest
  - Học gì thi nấy, không hỏi ngoài môn học
  - Không có giới hạn nội dung thi



# Mục tiêu của môn học này



- Nâng cao kỹ năng thực hành của sinh viên trong việc giải quyết các vấn đề thuật toán
- Nhấn mạnh vào quy trình “phân tích – thiết kế – cài đặt – tối ưu” thuật toán trong quá trình phát triển phần mềm
- Có thể trả lời được các câu hỏi phỏng vấn khi tìm việc làm ở các công ty tốt
  - Tất cả các công ty công nghệ hàng đầu đều phỏng vấn ứng viên lập trình về thuật toán (Google, Facebook, Apple, Microsoft, Grab, Shopee, Zalo, FPT, Viettel,...)
- Làm đẹp hồ sơ xin việc





Phần 2

# Thuật toán



- Thuật toán = Các bước để giải quyết vấn đề
- Thuật toán = Phương thức tính toán được cài đặt trên máy tính
- Đặc trưng:
  - Có đầu vào: một tập giá trị
  - Có đầu ra: kết quả tính toán, là một tập giá trị khác
  - Tính rõ ràng: xác định, không hiểu sai
  - \*Tính tổng quát: giải một lớp các bài toán
  - \*Tính dừng: kết thúc sau một số bước hữu hạn
  - \*Tính đúng: đưa ra kết quả đúng



## ■ Cơ bản

- Giới thiệu về Thuật toán
- Tìm kiếm & Sắp xếp
- Cấu trúc dữ liệu cơ bản
- Đệ quy, Quay lui và Nhánh cận
- Thuật toán tham lam
- Chia để trị
- Quy hoạch động
- Thuật toán về đồ thị và ứng dụng

## ■ Nâng cao

- Cấu trúc dữ liệu nâng cao và ứng dụng
- Thuật toán với chuỗi ký tự và ứng dụng



Phần 3

# Ví dụ đầu tiên

# Đoạn con có tổng lớn nhất



- Tìm đoạn con có tổng lớn nhất của một dãy số cho trước
- Cho một dãy số  $s = (a_1, \dots, a_n)$
- Đoạn con:  $s(i, j) = (a_i, \dots, a_j), 1 \leq i \leq j \leq n$
- Tổng đoạn con:  $w(i, j) = w(s(i, j)) = a_i + a_{i+1} + \dots + a_j$
- Tìm đoạn con có tổng lớn nhất:  $X = \max(w(i, j))$ 
  - Tìm cặp  $(p, q)$  để  $s(p, q)$  có tổng lớn nhất
  - $(p, q) = \operatorname{argmax}(w(i, j))$
- Ví dụ:
  - Dãy số: -2, 11, -4, 13, -5, 2
  - Dãy con có tổng lớn nhất là:  $s(2, 4) = (11, -4, 13)$ 
    - Tổng của dãy con này là 20

# Các cách tiếp cận



1. Duyệt toàn bộ các cặp  $(i, j)$
2. Duyệt toàn bộ nhưng tính tối ưu hơn
3. Chia để trị
4. Quy hoạch động

## 3.1 Duyệt toàn bộ các cặp (i, j)

- Duyệt tất cả các đoạn = duyệt mọi cặp (i, j)
- Tính tổng của các cặp và lưu lại đoạn lớn nhất

```
// Phương pháp 1: duyệt toàn bộ cặp (i,j)
int tong_max(vector<int> & a, int n) {
    int m = a[1];
    for (int i = 1; i <= n; i++)
        for (int j = i; j <= n; j++) {
            int s = 0;
            for (int k = i; k <= j; k++) s += a[k];
            if (s > m) m = s;
        }
    return m;
}
```

## 3.2 Duyệt toàn bộ nhưng tính tối ưu hơn



- Duyệt tất cả các đoạn = duyệt mọi cặp (i, j)
- Tính tổng của các cặp và lưu lại đoạn lớn nhất
- Tận dụng tính chất:  $w(i, j) = w(i, j-1) + a[j]$

// Phương pháp 2: duyệt nhưng tận dụng lại tổng cũ

```
int tong_max2(vector<int> & a, int n) {  
    int m = a[1];  
    for (int i = 1; i <= n; i++) {  
        int s = 0;  
        for (int j = i; j <= n; j++) {  
            s += a[j];  
            if (s > m) m = s;  
        }  
    }  
    return m;  
}
```



## 3.3 Chia để trị



- Định nghĩa đệ quy
- Đoạn con tổng lớn nhất của  $S$ , từ vị trí **đầu** đến vị trí **cuối**
- Tìm đoạn con tổng lớn nhất của  $S$ :
  - Nếu  $S$  chỉ có 1 phần tử: đoạn cần tìm chính là toàn bộ  $S$
  - Nếu  $S$  nhiều hơn 1 phần tử, ta chia đôi  $S = S1 + S2$
  - Đoạn con cần tìm sẽ rơi vào một trong ba tình huống
    - Trái: đoạn nằm trong  $S1$
    - Phải: đoạn nằm trong  $S2$
    - Giữa: đầu đoạn nằm trong  $S1$ , cuối đoạn nằm trong  $S2$
  - Tính giá trị của ba tình huống và trả về giá trị lớn nhất

## 3.3 Chia để trị



```
// Phương pháp 3: chia để trị
int tong_max3(vector<int> & a, int dau, int cuoi) {
    // xét trường hợp độ dài là 1
    if (dau == cuoi) return a[dau];

    // chia đôi dãy: (dau, mid) + (mid+1, cuoi)
    int mid = (dau + cuoi) / 2;

    // tính max trong 3 trường hợp
    int m_trai = tong_max3(a, dau, mid);
    int m_phai = tong_max3(a, mid + 1, cuoi);
    int m_giua = tong_max3(a, dau, mid) + tong_max3(a, mid + 1, cuoi);

    // trả về max của 3 trường hợp
    return max(m_giua, max(m_trai, m_phai));
}
```

## 3.3 Chia để trị



```
// tìm đoạn max phía trái (kết thúc ở cuối)
int trai(vector<int> & a, int dau, int cuoi) {
    int m = a[cuoi], s = 0;
    for (int i = cuoi; i >= dau; i--) {
        s += a[i];
        if (s > m) m = s;
    }
    return m;
}

// tìm đoạn max phía phải (bắt đầu từ dau)
int phai(vector<int> & a, int dau, int cuoi) {
    int m = a[dau], s = 0;
    for (int i = dau; i <= cuoi; i++) {
        s += a[i];
        if (s > m) m = s;
    }
    return m;
}
```

## 3.4 Quy hoạch động



- Xét các dãy con kết thúc tại  $k$ , đặt  $w(k)$  là tổng lớn nhất
  - $w(k) = \max(w(i, k))$
- Dễ thấy  $X = \max(w(i, j)) = \max(w(k))$
- Nếu ta tính được mọi  $w(k)$  thì dễ dàng tính  $X$
- Ta có thể tính nhanh  $w(k)$  dựa trên  $w(k-1)$ :
  - $w(1) = a_1$
  - $w(k) = a_k$  hoặc  $w(k-1) + a_k$ , ứng với hai tình huống
    - Dãy chỉ gồm  $a_k$
    - Dãy gồm  $a_k$  và nối với phần tử phía trước  $a_{k-1}$



## 3.4 Quy hoạch động

```
// Phương pháp 4: quy hoạch động
int tong_max4(vector<int> & a, int n) {
    vector<int> w(n+1);
    w[1] = a[1];
    for (int k = 2; k <= n; k++)
        w[k] = max(a[k], w[k-1] + a[k]);
    int m = w[1];
    for (int i = 2; i <= n; i++)
        if (m < w[i]) m = w[i];
    return m;
}
```



## 3.4 Quy hoạch động

// Phương pháp 4.1: quy hoạch động với mã tối ưu hơn

```
int tong_max4_1(vector<int> & a, int n) {  
    int w = a[1], m = a[1];  
    for (int k = 2; k <= n; k++) {  
        if (w > 0) w = w + a[k];  
        else w = a[k];  
        if (w > m) m = w;  
    }  
    return m;  
}
```



Phần 4

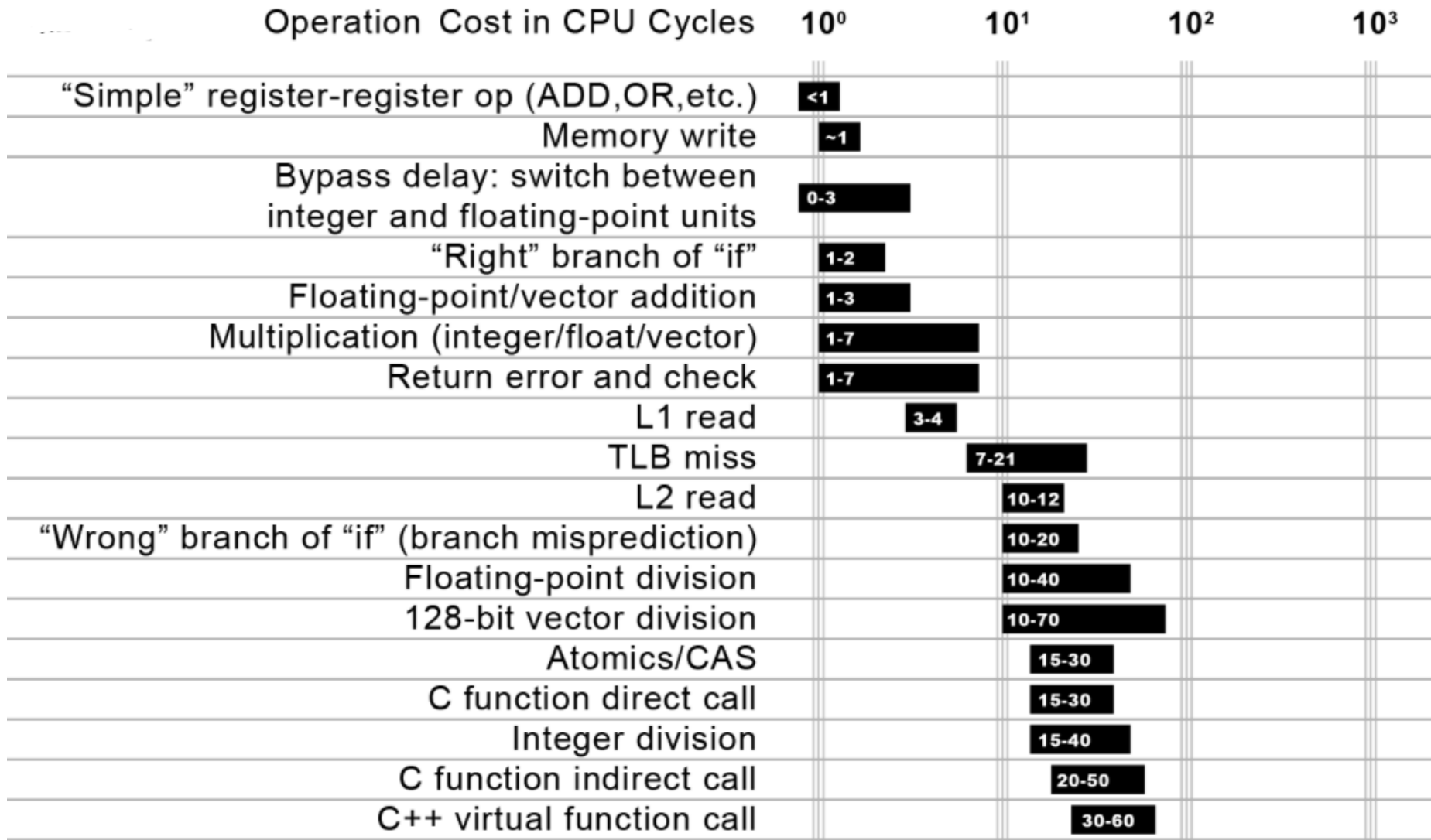
# Phân tích thuật toán



- Giúp hiểu được sơ bộ “chi phí” khi thực hiện thuật toán
- Các khía cạnh quan tâm:
  - Kích thước đầu vào
  - Kích thước đầu ra
  - Tài nguyên để chạy thuật toán
    - Bộ nhớ
    - Số lượng phép toán cần thực hiện
- Số lượng phép toán  $\approx$  thời gian tính toán
  - Phép so sánh: 1 (CPU cycle)
  - Phép cộng/trừ: 1 (CPU cycle)
  - Phép nhân: 10 (CPU cycles)
  - Phép chia: 66-80 (CPU cycles)



# Phân tích thuật toán



# Phân tích thuật toán: bài tập ví dụ



- Thuật toán 1:
  - Bộ ba  $(i, k, j)$  sắp thứ tự  $\approx n^3/6$
  - $O(n^3)$
- Thuật toán 2:
  - Cặp  $(i, j)$  sắp thứ tự  $\approx n^2/2$
  - $O(n^2)$
- Thuật toán 3:
  - Dãy độ dài  $n$ , chia đôi cho đến khi còn 1 phần tử
    - Độ sâu  $k$  thỏa mãn  $n/2^k \approx 1$
    - Nên  $k \approx \log_2 n$
  - $O(n \log_2 n)$
- Thuật toán 4:  $O(n)$



- **Thuật toán ổn định:** thời gian chạy không đổi, chỉ phụ thuộc vào số lượng dữ liệu đầu vào
- **Thuật toán thích ứng:** thời gian chạy thay đổi tùy thuộc vào giá trị của dữ liệu đầu vào
  - **Thời gian chạy tốt nhất:** thời gian thực thi ngắn nhất với dữ liệu đầu vào cỡ  $n$
  - **Thời gian chạy tồi nhất:** thời gian thực thi dài nhất với dữ liệu đầu vào cỡ  $n$
  - **Thời gian chạy trung bình:** thời gian thực thi trung bình của mọi lần chạy
    - Rất khó tính toán thực tế (vì phải thử mọi trường hợp)
    - Tính toán ước lượng dựa trên số lượng phép toán phải thực hiện
    - Dựa trên giải thiết xác suất của dữ liệu đầu vào

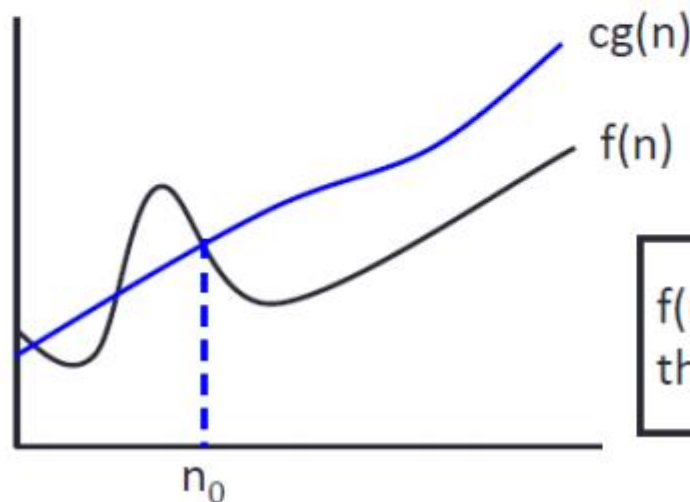
# Phân tích độ tăng trưởng



- Độ tăng trưởng cho ta ước lượng những tình huống chưa thử nghiệm thực tế
- Chặn trên của  $g(n)$ :  $O(g(n))$

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

khi và chỉ khi  $\exists c > 0$  và  $n_0 > 0$  sao cho  $f(n) \leq cg(n) \forall n \geq n_0$



$f(n)$  bị chặn trên bởi  $g(n)$   
theo nghĩa tiệm cận

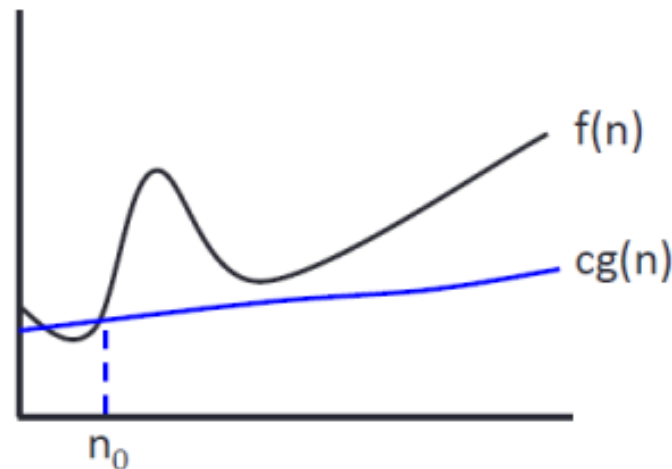
# Phân tích độ tăng trưởng



- Chặn dưới của  $g(n)$ :  $\Omega(g(n))$
- Ô-mê-ga của hàm có thể không cùng bậc với hàm đó

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

khi và chỉ khi  $\exists c > 0$  và  $n_0 > 0$  sao cho  $cg(n) \leq f(n) \forall n \geq n_0$



$f(n)$  bị chặn dưới bởi  $g(n)$   
theo nghĩa tiệm cận

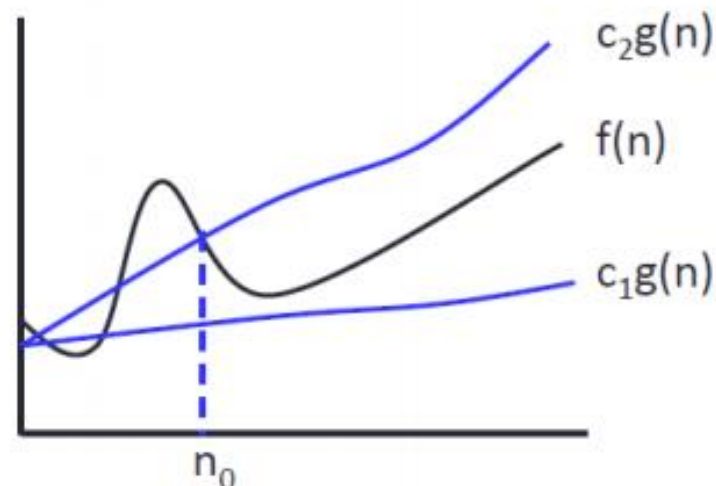
# Phân tích độ tăng trưởng



- Tiệm cận của  $g(n)$ :  $\theta(g(n))$
- Thê-ta của hàm thường là cùng bậc với hàm đó

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

khi và chỉ khi  $\exists c_1 > 0, c_2 > 0$  và  $n_0 > 0$  sao cho  
 $c_1g(n) \leq f(n) \leq c_2g(n) \quad \forall n \geq n_0$



$f(n)$  có cùng tốc độ tăng với  $g(n)$  theo nghĩa tiệm cận



- Cài đặt thuật toán
- Chạy chương trình rất nhiều lần
  - Trên cùng một máy tính
  - Dữ liệu đầu vào có kích thước khác nhau
  - Dữ liệu đầu vào có phân bố giá trị khác nhau
- Đo thời gian chạy thực tế
- Biểu diễn, so sánh kết quả



- Điểm yếu:
  - Cài đặt thuật toán đôi khi dựa vào năng lực của người viết mã
  - Phải thử rất nhiều tình huống thì kết quả mới ổn định và có tính đại diện cao
  - Không thể ngoại suy ra kết quả trong các tình huống mới
- Điểm mạnh:
  - Trực quan, rõ ràng
  - Tin cậy





Phần 5

# Bài tập

# Viết mã và phân tích độ phức tạp tính toán



1. Nhập 2 số nguyên  $a$  và  $b$ , hãy tính  $a^b$ . In ra 9 chữ số cuối cùng nếu giá trị tìm được lớn hơn  $10^9$ .

2. Dãy số Fibonacci được định nghĩa như sau:

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \text{ nếu } n > 1$$

Viết hàm  $F(n)$  trả về giá trị của số Fibonacci thứ  $n$ .

3. Cho dãy  $A = (a_0, a_1, \dots, a_{N-1})$ , nhập số  $K$  ( $0 \leq K < N$ ). Tìm phần tử nhỏ thứ  $K$  trong dãy  $A$ , in ra giá trị phần tử đó.

4. Nhập 2 số nguyên  $a$  và  $b$ , hãy tính  $a^b$ . In ra 9 chữ số đầu tiên nếu giá trị tìm được lớn hơn  $10^9$ .