

LAB 2: Phân tích thuật toán

1 Phân tích độ phức tạp thời gian của thuật toán

1.1

a. n^2

(a) Khi tăng gấp đôi kích thước đầu vào:

$$\frac{(2n)^2}{n^2} = 4$$

\Rightarrow Thuật toán chạy chậm hơn **4 lần**.

(b) Khi tăng kích thước đầu vào thêm 1:

$$\frac{(n+1)^2}{n^2} = 1 + \frac{2}{n} + \frac{1}{n^2}$$

Với n lớn $\rightarrow 1 \Rightarrow$ Không đáng kể.

b. n^3

(a) Khi tăng gấp đôi kích thước đầu vào:

$$\frac{(2n)^3}{n^3} = 8$$

\Rightarrow Thuật toán chạy chậm hơn **8 lần**.

(b) Khi tăng kích thước đầu vào thêm 1:

$$\frac{(n+1)^3}{n^3} = 1 + \frac{3}{n} + \frac{3}{n^2} + \frac{1}{n^3}$$

Với n lớn $\rightarrow 1 \Rightarrow$ Không đáng kể.

c. $100n^2$

(a) Khi tăng gấp đôi kích thước đầu vào:

$$\frac{(100(2n))^2}{100n^2} = 4$$

\Rightarrow Thuật toán chạy chậm hơn **4 lần**.

(b) Khi tăng kích thước đầu vào thêm 1:

$$\frac{100(n+1)^2}{n^2} = 100 + \frac{200}{n} + \frac{100}{n^2}$$

Với n lớn $\rightarrow 1 \Rightarrow$ Không đáng kể.

d. $n \log n$

(a) Khi tăng gấp đôi kích thước đầu vào:

$$\frac{(2n) \log(2n)}{n \log n} = 2 + \frac{2 \log 2}{\log n}$$

Với n lớn $\rightarrow 2 \Rightarrow$ Thuật toán chạy chậm hơn **gần 2 lần**.

(b) Khi tăng kích thước đầu vào thêm 1:

$$\frac{(n+1) \log(n+1)}{n \log n} = 1 + \frac{\log(n+1)}{\log n} + \frac{1}{n \log n}$$

Với n lớn $\rightarrow 1 \Rightarrow$ Không đáng kể.

e. 2^n

(a) Khi tăng gấp đôi kích thước đầu vào:

$$\frac{2^{2n}}{2^n} = 2^n$$

Với n lớn \rightarrow rất lớn \Rightarrow Thuật toán chạy chậm hơn **rất nhiều lần**.

(b) Khi tăng kích thước đầu vào thêm 1:

$$\frac{2^{n+1}}{2^n} = 2$$

\Rightarrow Thuật toán chạy chậm hơn **2 lần**.

1.2

a. Phát biểu định nghĩa chính thức của ký pháp Big-O:

$$\exists c > 0, n_0 > 0 \text{ sao cho } T(n) \leq c \cdot f(n), \quad \forall n \geq n_0.$$

b. Cho $f(n)$ và $g(n)$ là các hàm số dương. Sử dụng định nghĩa cơ bản của ký pháp Big-O, chứng minh rằng:

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

Chứng minh: Với mọi n , ta có:

$$\max(f(n), g(n)) \leq f(n) + g(n).$$

Do đó, $\exists c > 0$ sao cho:

$$\max(f(n), g(n)) \leq c \cdot (f(n) + g(n)).$$

Theo định nghĩa:

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

- c. Chứng minh rằng mệnh đề sau là vô nghĩa: "*Thời gian chạy của thuật toán A ít nhất là $O(n^2)$* ".

O là ký pháp mô tả cận trên (upper bound). Do đó, mệnh đề trên không có ý nghĩa vì nó không xác định được cận dưới (lower bound) của thời gian chạy thuật toán A.

1.3

- a. Chứng minh hoặc phản bác mệnh đề sau:

$$2^{n+1} = O(2^n)$$

Chứng minh: Ta có:

$$2^{n+1} = 2 \cdot 2^n \leq c \cdot 2^n, \quad \text{với } c = 2, n_0 = 1.$$

\Rightarrow **đpcm.**

- b. Chứng minh hoặc phản bác mệnh đề sau:

$$2^{2n} = O(2^n)$$

Phản bác: Ta có:

$$2^{2n} = (2^n)^2 \gg c \cdot 2^n, \quad \text{với mọi } c > 0, n_0 > 0.$$

\Rightarrow **Khác hoàn toàn.**

- c. Chứng minh hoặc phản bác mệnh đề sau:

Nếu $f(n) = O(g(n))$ thì

$$\log f(n) = O(\log g(n)), \quad \text{với giả thiết } f(n) > 1, g(n) > 1.$$

Chứng minh: Theo định nghĩa Big-O, ta có:

$$f(n) \leq c \cdot g(n), \quad \forall n \geq n_0.$$

Lấy log hai vế, ta được:

$$\begin{aligned} \log f(n) &\leq \log (c \cdot g(n)) \\ \Leftrightarrow \log f(n) &= \log c + \log g(n) \\ \Leftrightarrow \log f(n) &\leq c' \cdot \log g(n) \end{aligned}$$

$$\text{với } c' = \log c + 1, \quad n > n_0.$$

\Rightarrow **đpcm.**

d. Chứng minh hoặc phản bác mệnh đề sau:

Nếu $f(n) = O(g(n))$ và $g(n) \geq 1$ thì

$$2^{f(n)} = O(2^{g(n)}).$$

Phản bác: Giả sử $f(n) = 2n$, $g(n) = n$. Khi đó:

$$f(n) = O(g(n)) \quad \text{và} \quad g(n) \geq 1.$$

Mà:

$$2^{f(n)} = 2^n \ll 2^{2n} \quad \text{với mọi } c > 0, n > n_0.$$

\Rightarrow **Khác hoàn toàn.**

1.4

Sắp xếp theo thứ tự tăng dần theo tốc độ tăng trưởng (growth rate) của các hàm sau:

Gợi ý: Sử dụng quy tắc L'Hôpital hoặc so sánh logarit để xử lý các hàm phức tạp.

Bài làm

- a. $f_1(n) = n + 10$; $f_2(n) = \sqrt{2n}$; $f_3(n) = n^2 \log(n)$; $f_4(n) = n^{2.5}$; $f_5(n) = 10^n$; $f_6(n) = 100^n$.

Sắp xếp:

$$f_1(n) = O(n)$$

$$f_2(n) = O(\sqrt{n})$$

$$f_3(n) = O(n^2 \log(n))$$

$$f_4(n) = O(n^{2.5})$$

$$f_5(n) = O(10^n)$$

$$f_6(n) = O(100^n)$$

$$\Rightarrow f_2(n) < f_1(n) < f_3(n) < f_4(n) < f_5(n) < f_6(n)$$

b. $g_1(n) = n(\log n)^3$; $g_2(n) = n^{4/3}$; $g_3(n) = 2^n$; $g_4(n) = n^{\log(n)}$; $g_5(n) = 2^{2n}$.

Phân tích:

– So sánh $g_1(n) = n(\log n)^3$ và $g_2(n) = n^{4/3}$:

$$\lim_{n \rightarrow \infty} \frac{n(\log n)^3}{n^{4/3}} \stackrel{L}{=} 0.$$

$$\Rightarrow g_1(n) < g_2(n).$$

– So sánh $g_2(n) = n^{4/3}$ và $g_3(n) = 2^n$:

$$\lim_{n \rightarrow \infty} \frac{n^{4/3}}{2^n} = 0.$$

$$\Rightarrow g_2(n) < g_3(n).$$

– So sánh $g_3(n) = 2^n$ và $g_4(n) = n^{\log n}$:

$$\lim_{n \rightarrow \infty} \frac{2^n}{n^{\log n}} = \lim_{n \rightarrow \infty} 2^{n - (\log n)^2} = \infty.$$

$$\Rightarrow g_3(n) > g_4(n).$$

– So sánh $g_3(n) = 2^n$ và $g_5(n) = 2^{2n}$:

$$\lim_{n \rightarrow \infty} \frac{2^n}{2^{2n}} = \lim_{n \rightarrow \infty} 2^{(\log n)^2 - 2n} = 0.$$

$$\Rightarrow g_3(n) < g_5(n).$$

– So sánh $g_1(n) = n(\log n)^3$ và $g_4(n) = n^{\log n}$:

$$\lim_{n \rightarrow \infty} \frac{n(\log n)^3}{n^{\log n}} = \lim_{n \rightarrow \infty} n^{1 - \log n} (\log n)^3 = 0.$$

$$\Rightarrow g_1(n) < g_4(n).$$

– So sánh $g_4(n) = n^{\log n}$ và $g_2(n) = n^{4/3}$:

$$\lim_{n \rightarrow \infty} \frac{n^{\log n}}{n^{4/3}} = \lim_{n \rightarrow \infty} n^{\log n - 4/3} = \infty.$$

$$\Rightarrow g_4(n) > g_2(n).$$

Kết luận:

$$g_1(n) < g_2(n) < g_4(n) < g_3(n) < g_5(n)$$

1.5

Sử dụng Master Theorem để giải các phương trình đệ quy sau:

a. $T(n) = 2T\left(\frac{n}{2}\right) + n$

b. $T(n) = 3T\left(\frac{n}{2}\right) + n$

c. $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

d. $T(n) = T\left(\frac{n}{2}\right) + n$

e. $T(n) = 2T\left(\frac{n}{2}\right) + 1$

f. $T(n) = 3T\left(\frac{n}{3}\right) + n$

Gợi ý: Xác định rõ các tham số a , b , và $f(n)$ trước khi áp dụng Master Theorem. Nếu Master Theorem không áp dụng được, sử dụng phương pháp thế.

Master Theorem: Let $a \geq 1$ and $b > 1$ be constants, and let $f(n)$ be an asymptotically positive function. Consider the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$

Bài làm

a. $T(n) = 2T\left(\frac{n}{2}\right) + n$

$$a = 2, \quad b = 2, \quad f(n) = n.$$

Ta có:

$$n^{\log_b a} = n^{\log_2 2} = n.$$

Do đó, theo trường hợp 2 của Master Theorem:

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

b. $T(n) = 3T\left(\frac{n}{2}\right) + n$

$$a = 3, \quad b = 2, \quad f(n) = n.$$

Ta có:

$$n^{\log_b a} = n^{\log_2 3} \approx n^{1.585}$$

Do đó, theo trường hợp 1 của Master Theorem:

$$\Rightarrow f(n) = O(n^{\log_b a - \epsilon}) \text{ với } \epsilon \approx 0.585.$$

c. $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

$$a = 4, \quad b = 2, \quad f(n) = n^2.$$

Ta có:

$$n^{\log_b a} = n^{\log_2 4} = n^2.$$

Do đó, theo trường hợp 2 của Master Theorem:

$$T(n) = \Theta(n^2 \log n).$$

d. $T(n) = T\left(\frac{n}{2}\right) + n$

$$T(n) = T(n/4) + n/2 + n$$

$$T(n) = T(n/8) + n/4 + n/2 + n$$

$$T(n) = T(n/16) + n/8 + n/4 + n/2 + n$$

$$\vdots$$

$$T(n) = T(1) + n(1 + 1/2 + 1/4 + \dots)$$

$$T(n) = T(1) + 2n$$

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

e. $T(n) = 2T\left(\frac{n}{2}\right) + 1$

$$a = 2, \quad b = 2, \quad f(n) = 1.$$

Ta có:

$$n^{\log_b a} = n^{\log_2 2} = n.$$

Do đó, theo trường hợp 1 của Master Theorem:

$$\Rightarrow f(n) = O(n^{\log_b a - \epsilon}) \text{ với } \epsilon = 1.$$

f. $T(n) = 3T\left(\frac{n}{3}\right) + n$

$$a = 3, \quad b = 3, \quad f(n) = n.$$

Ta có:

$$n^{\log_b a} = n^{\log_3 3} = n.$$

Do đó, theo trường hợp 2 của Master Theorem:

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

1.6

Phân tích độ phức tạp thời gian của các thuật toán sau trong các trường hợp (Worst case, Average case, Best case):

- a. Tìm kiếm nhị phân (Binary Search)
- b. Kiểm tra số nguyên tố
- c. Thuật toán Euclid tính Ước số chung lớn nhất (GCD)
- d. Đếm số bit 1 trong biểu diễn nhị phân của số nguyên

Bài làm

- a. Tìm kiếm nhị phân (Binary Search)

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

Giải phương trình đệ quy trên ta được:

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

- **Best case:** Phần tử cần tìm nằm ngay ở giữa mảng \Rightarrow chỉ cần 1 phép so sánh.

$$T_{\text{best}}(n) = O(1)$$

- **Worst case:** Phần tử cần tìm không tồn tại hoặc nằm ở biên, mỗi lần thu hẹp tìm kiếm còn một nửa, lặp cho đến khi còn 1 phần tử.

$$T_{\text{worst}}(n) = O(\log n)$$

- **Average case:** Trung bình cũng phải thực hiện số lần so sánh tương tự như worst case.

$$T_{\text{avg}}(n) = O\left(\frac{\log n}{2}\right) = O(\log n)$$

- b. Kiểm tra số nguyên tố

$$T(n) = O(\sqrt{n})$$

- **Best case:** Số n là số chẵn lớn hơn 2 \Rightarrow chỉ cần 1 phép chia.

$$T_{\text{best}}(n) = O(1)$$

- **Worst case:** Số n là số nguyên tố \Rightarrow phải kiểm tra tất cả các số từ 2 đến \sqrt{n} .

$$T_{\text{worst}}(n) = O(\sqrt{n})$$

- **Average case:** Trung bình cũng phải thực hiện số phép chia tương tự như worst case.

$$T_{\text{avg}}(n) = O(\sqrt{n})$$

c. Thuật toán Euclid tính Ước số chung lớn nhất (GCD)

$$T(n) = T(n \log n) + O(1)$$

- **Best case:** Hai số a và b bằng nhau \Rightarrow chỉ cần 1 phép chia lấy dư.

$$T_{\text{best}}(n) = O(1)$$

- **Worst case:** Hai số a và b là hai số Fibonacci liên tiếp đến khi dư = 0 \Rightarrow phải thực hiện nhiều phép chia lấy dư.

$$T_{\text{worst}}(n) = O(\log n)$$

- **Average case:** Trung bình cũng phải thực hiện số phép chia tương tự như worst case.

$$T_{\text{avg}}(n) = O(\log n)$$

d. Đếm số bit 1 trong biểu diễn nhị phân của số nguyên

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

- **Best case:** Số $n = 0 \Rightarrow$ không có bit 1.

$$T_{\text{best}}(n) = O(1)$$

- **Worst case:** Số n có tất cả các bit là 1 \Rightarrow phải kiểm tra tất cả các bit.

$$T_{\text{worst}}(n) = O(\log n)$$

- **Average case:** Trung bình số bit 1 trong biểu diễn nhị phân của số nguyên có độ dài k là $k/2 \Rightarrow$ phải kiểm tra khoảng nửa số bit.

$$T_{\text{avg}}(n) = O(\log n)$$

1.7

Phân tích độ phức tạp thời gian của các thuật toán sau:

a. Dãy Fibonacci**- Độ quy đơn giản:**

Ý tưởng: Gọi đệ quy trực tiếp theo công thức:

$$F(n) = F(n-1) + F(n-2)$$

Tính toán lặp lại nhiều lần.

$$T(n) = T(n-1) + T(n-2) + O(1) \implies T(n) = O(2^n)$$

- Độ quy có nhớ (Memoization):

Ý tưởng: Lưu kết quả đã tính vào mảng nhớ, lần sau cần thì lấy ra, không tính lại. Mỗi giá trị chỉ tính một lần, do đó:

$$T(n) = O(n)$$

- Lặp (Iteration):

Ý tưởng: Tính tuần tự từ $F(0), F(1)$ rồi dùng vòng lặp đến $F(n)$. Chỉ cần duyệt từ 1 đến n , nên:

$$T(n) = O(n)$$

b. Tính tổng mảng bằng phương pháp chia để trị

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

Áp dụng định lý Master, ta có:

$$T(n) = O(n)$$

c. Tính lũy thừa a^n **- Naïve:** Nhân a với nhau n lần:

$$T(n) = O(n)$$

- Chia để trị (Exponentiation by Squaring):

$$T(n) = T\left(\frac{n}{2}\right) + O(1) \implies T(n) = O(\log n)$$

d. Tháp Hà Nội (đệ quy)

$$T(n) = 2T(n-1) + O(1) \implies T(n) = O(2^n)$$

2 Lập trình và Đo lường thời gian thực thi thuật toán

2.1

Cài đặt và so sánh thời gian thực thi của từng thuật toán trong 1.6

Yêu cầu:

- Đo thời gian thực thi của thuật toán với kích cỡ input: $n < 10^6$.
- Vẽ biểu đồ so sánh thời gian thực thi của thuật toán.
- So sánh kết quả thực nghiệm với dự đoán lý thuyết từ phần 1 và giải thích sự khác biệt (nếu có).
- Thiết kế các bộ input khác nhau để kiểm tra các trường hợp (Worst case, Average case, Best case).
- Xác định điểm giao (giá trị n mà tại đó thuật toán A có độ phức tạp lý thuyết tốt hơn nhưng thực tế lại chậm hơn thuật toán B).

Bài làm

- Đo thời gian thực thi của thuật toán với kích cỡ input: $n < 10^6$.

| Input Size | BinarySearch | LinearSearch | Ratio (B/A) |
|------------|--------------|--------------|-------------|
| 2 | 0.0000 | 0.0000 | — |
| 4 | 0.0000 | 0.0000 | — |
| 8 | 0.0000 | 0.0000 | — |
| 16 | 0.0000 | 0.0000 | — |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 65536 | 0.0000 | 0.9990 | — |
| 131072 | 0.0000 | 1.8250 | — |
| 262144 | 0.0000 | 3.9990 | — |
| 524288 | 0.0000 | 9.7710 | — |

Bảng 1: So sánh thời gian thực thi giữa Binary Search và Linear Search

| Input Size | Prime Optimized (ms) | Prime Naive (ms) | Ratio |
|------------|----------------------|------------------|-------|
| 2 | 0.0000 | 0.0000 | — |
| 4 | 0.0000 | 0.0000 | — |
| 8 | 0.0000 | 0.0000 | — |
| 16 | 0.0000 | 0.0000 | — |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 32768 | 0.0000 | 0.1950 | — |
| 65536 | 0.0000 | 0.2000 | — |
| 131072 | 0.0000 | 0.2240 | — |
| 262144 | 0.0000 | 0.9980 | — |
| 524288 | 0.0000 | 4.0020 | — |

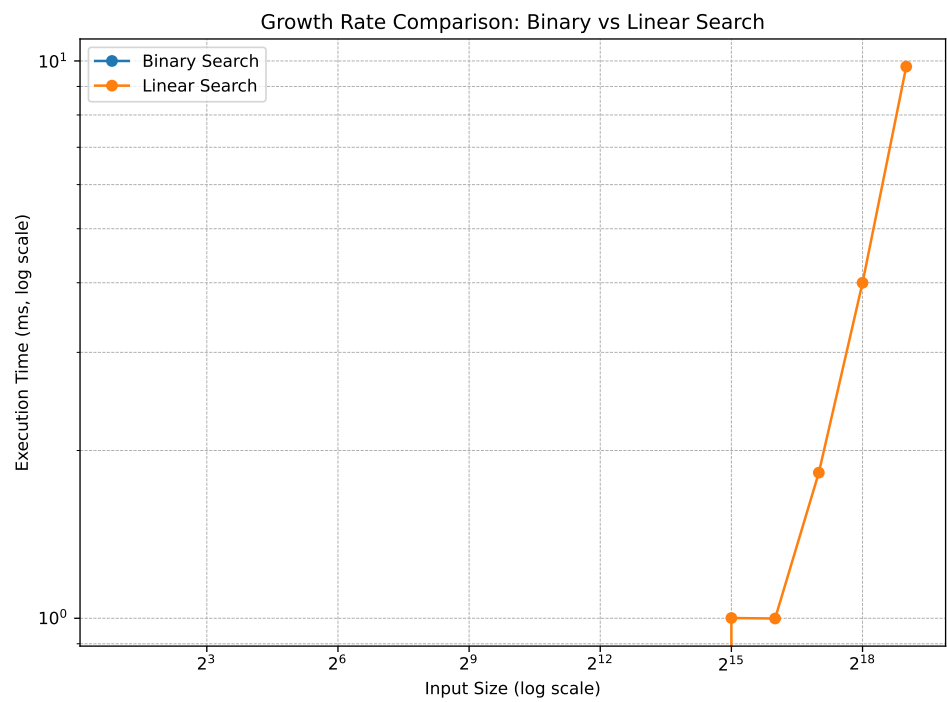
Bảng 2: So sánh thời gian chạy kiểm tra số nguyên tố: Optimized vs Naive

| Input Size | Prime Optimized (ms) | Prime Naive (ms) | Ratio |
|------------|----------------------|------------------|-------|
| 2 | 0.0000 | 0.0000 | — |
| 4 | 0.0000 | 0.0000 | — |
| 8 | 0.0000 | 0.0000 | — |
| 16 | 0.0000 | 0.0000 | — |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 32768 | 0.0000 | 0.0000 | — |
| 65536 | 0.0000 | 0.0000 | — |
| 131072 | 0.0000 | 0.0000 | — |
| 262144 | 0.0000 | 0.0000 | — |
| 524288 | 0.0000 | 0.0000 | — |

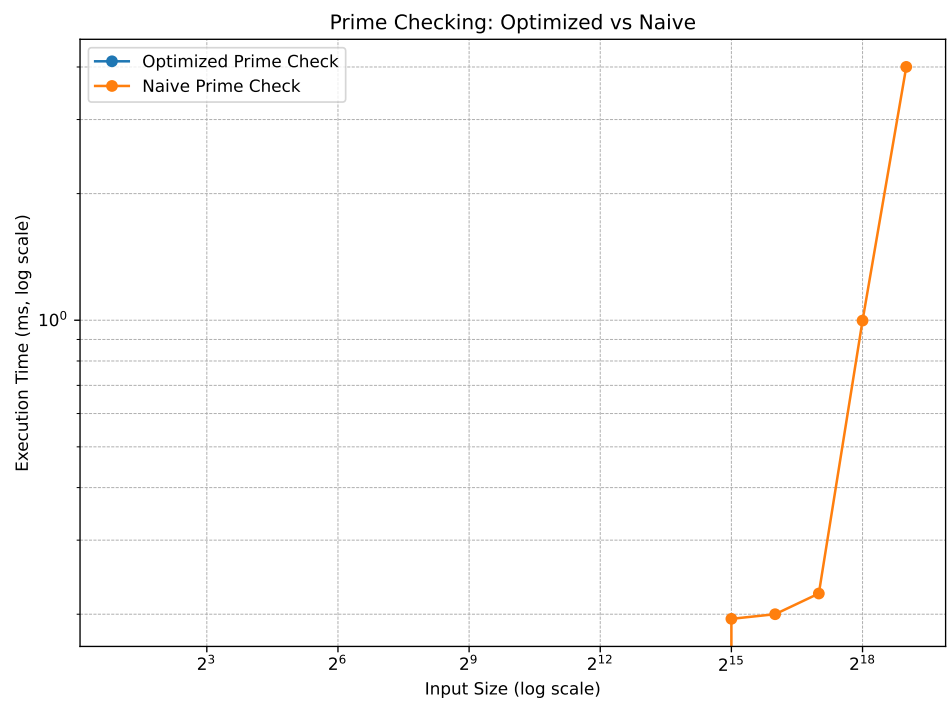
Bảng 3: So sánh thời gian chạy kiểm tra số nguyên tố: Optimized vs Naive

Bit count (tương tự như GCD) \Rightarrow không thay đổi quá nhiều.

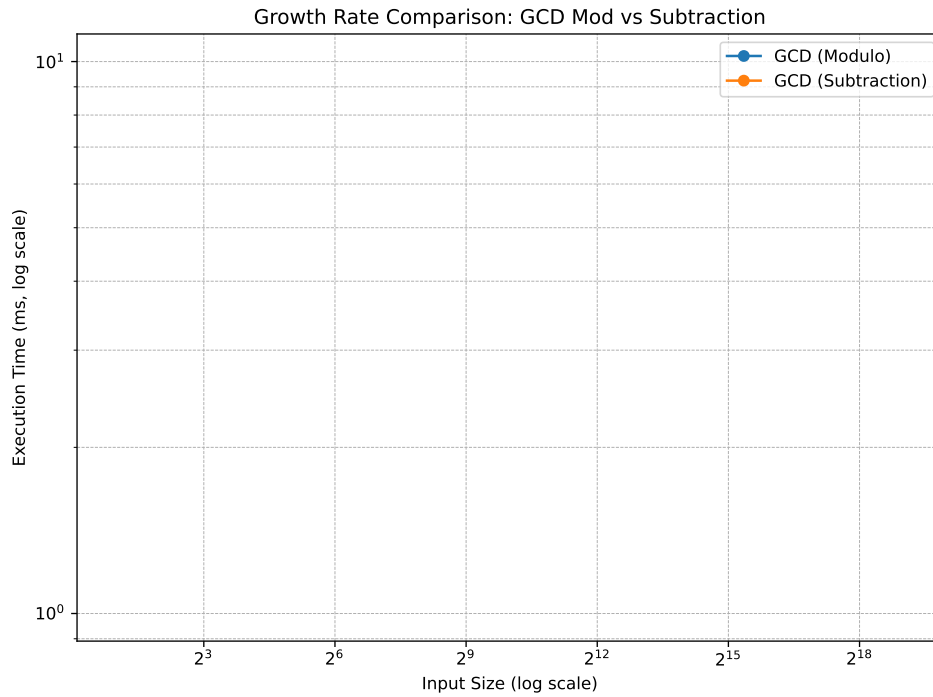
- b. Vẽ biểu đồ so sánh thời gian thực thi của thuật toán.



Hình 1: So sánh thời gian thực thi giữa Binary Search và Linear Search



Hình 2: So sánh thời gian chạy kiểm tra số nguyên tố: Optimized vs Naive



Hình 3: So sánh thời gian chạy tính GCD: Modulo vs Subtraction

Bit count (tương tự như GCD) \Rightarrow không thay đổi quá nhiều.

e. Xác định điểm giao:

- **Binary vs Linear:** Điểm giao khoảng $n \in [20, 50]$, nơi Binary bắt đầu nhanh hơn. Mặc dù về lý thuyết Binary tốt hơn ngay từ đầu, nhưng hằng số ẩn làm nó chậm hơn ở n nhỏ.
- **Prime Optimized vs Naive:** Optimized luôn nhanh hơn, không có điểm giao nơi A chậm hơn.
- **GCD Mod vs Sub:** Điểm giao ở n nhỏ ($n < 10$), Subtraction nhanh hơn do phép trừ rẻ hơn modulo.
- **Bit Kern vs Loop:** Kernel luôn nhanh hơn hoặc bằng, không có điểm giao.

2.2

Phân tích độ phức tạp thời gian của các thuật toán sau:

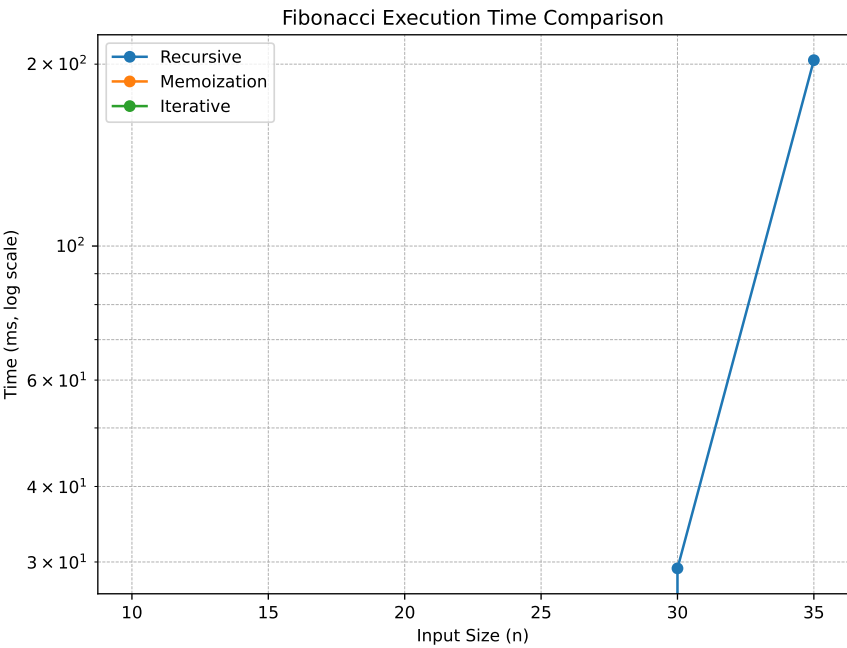
- a. Dãy Fibonacci (đệ quy đơn giản, đệ quy có nhớ, lặp)
- b. Tính tổng mảng bằng phương pháp chia để trị
- c. Tính lũy thừa a^n (Phương pháp naive và chia để trị)
- d. Tháp Hà Nội (đệ quy)

Bài làm

a. Dãy Fibonacci (đệ quy đơn giản, đệ quy có nhớ, lặp)

| Input Size | Recursive (ms) | Memoization (ms) | Iterative (ms) |
|------------|----------------|------------------|----------------|
| 10 | 0.0000 | 0.0000 | 0.0000 |
| 15 | 0.0000 | 0.0000 | 0.0000 |
| 20 | 0.0000 | 0.0000 | 0.0000 |
| 25 | 0.0000 | 0.0000 | 0.0000 |
| 30 | 0.0000 | 0.0000 | 0.0000 |
| 35 | 127.1480 | 0.0000 | 0.0000 |

Bảng 4: So sánh thời gian thực thi Fibonacci: Recursive vs Memoization vs Iterative

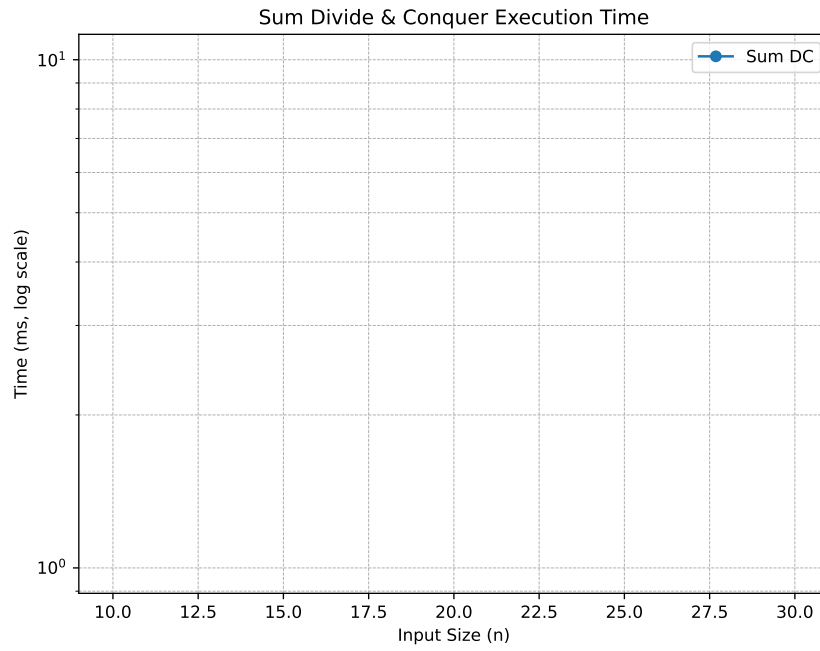


Hình 4: So sánh thời gian thực thi giữa các phương pháp Fibonacci

b. Tính tổng mảng bằng phương pháp chia để trị

| Input Size | Sum Divide & Conquer (ms) |
|------------|---------------------------|
| 10 | 0.0000 |
| 20 | 0.0000 |
| 30 | 0.0000 |

Bảng 5: Thời gian thực thi thuật toán tính tổng mảng bằng Divide & Conquer

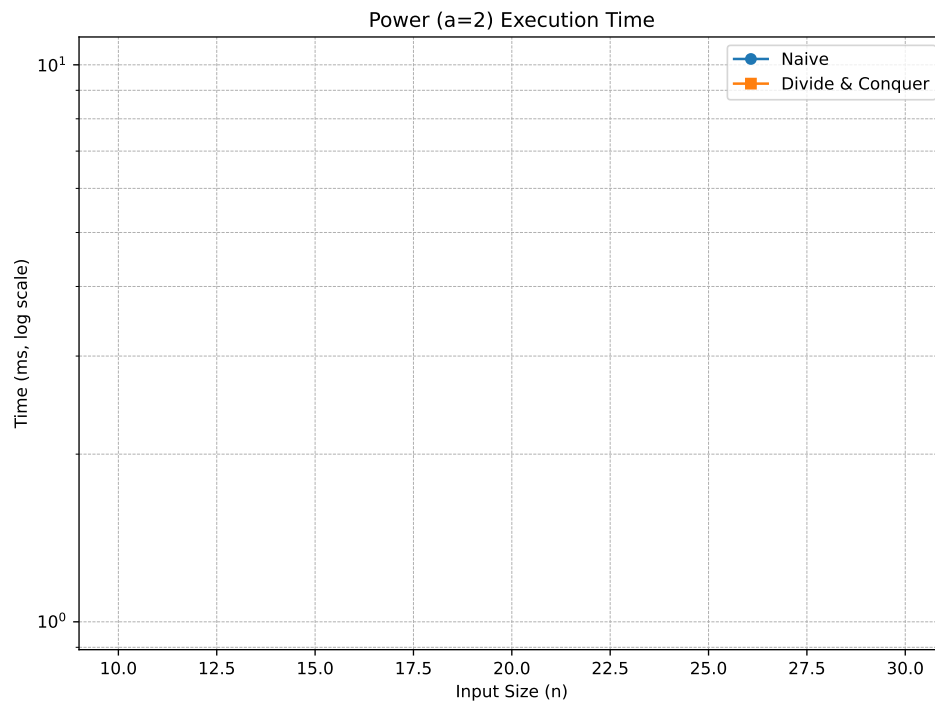


Hình 5: Thời gian thực thi thuật toán tính tổng bằng Divide & Conquer

c. Tính lũy thừa a^n (Phương pháp naive và chia để trị)

| Input Size | a^n Phương pháp naive và chia để trị |
|------------|--|
| 10 | 0.0000 |
| 20 | 0.0000 |
| 30 | 0.0000 |

Bảng 6: Thời gian thực thi thuật toán tính lũy thừa a^n (Phương pháp naive và chia để trị)

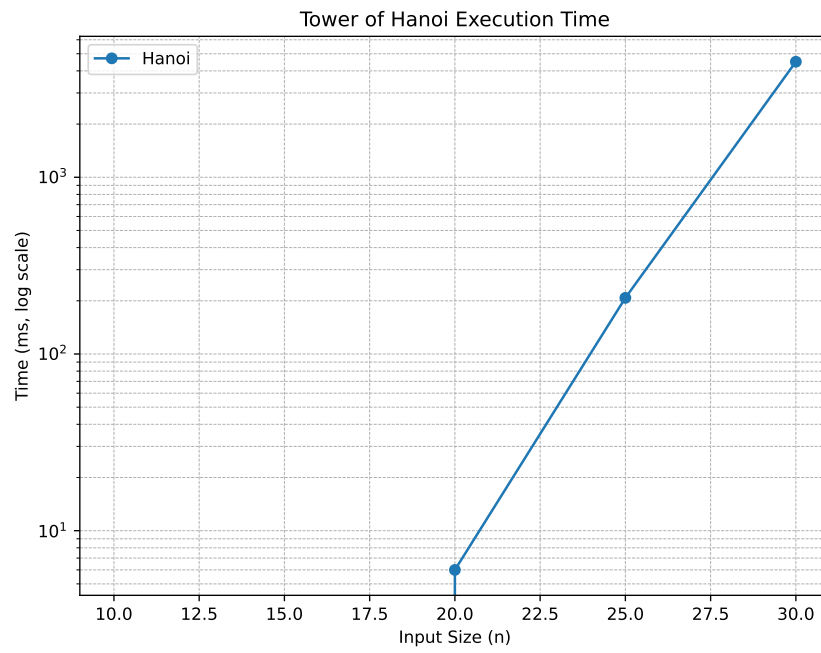


Hình 6: Thời gian thực thi thuật toán tính lũy thừa a^n (Phương pháp naive và chia để trị)

d. Tháp Hà Nội (đệ quy)

| Input Size | Hanoi (ms) |
|------------|------------|
| 10 | 0.0000 |
| 15 | 0.0000 |
| 20 | 5.0910 |
| 25 | 130.7070 |
| 30 | 4550.7800 |

Bảng 7: Thời gian thực thi bài toán Tháp Hà Nội



Hình 7: Thời gian thực thi thuật toán Tháp Hà Nội