Kết quả đo thời gian thực thi các thuật toán

n	Fib đệ quy (ms)	Fib có nhớ(ms)	Fib lặp (ms)	Sum (ms)	Pow Naive (ms)	Pow chia để trị (ms)	Hanoi (ms)
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0
18	0	0	0	0	0	0	1.002
19	0	0	0	0	0	0	0
20	0	0	0	1.121	0	0	1.385
21	0	0	0	0	0	0	2.308

22	1.392	0	0	0	0	0	6.3
23	0	0	0	0	0	0	14.113
24	0	0	0	0	0	0	28.219
25	0	0	0	0	0	0	55.126
26	1.017	0	0	0	0	0	105.901
27	1.216	0	0	0	0	0	220.228
28	1.003	0	0	0	0	0	431.116
29	2.034	0	0	0	0	0	854.852
30	3.063	0	0	0	0	0	1763.14
31	5.972	0	0	0	0	0	3906.35
32	11.891	0	0	0	0	0	7458.05
33	15.576	0	0	0	0	0	17591.3
34	51.169	0	0	0	0	0	67380.2

+ Giá trị n lớn nhất mà phiên bản đệ quy đơn giản có thể chạy là : n chạy khoảng 34, còn nếu n=40 thì chạy mất khoảng 17 phút.

+ Bảng trên cho thấy sự so sánh giữa dự đoán lý thuyết và kết quả thực nghiệm của bài đo thời gian các thuật toán.
Về xu hướng tăng, lý thuyết dự đoán rằng thời gian thực thi của thuật toán Fibonacci đệ quy sẽ tăng theo hàm mũ ($O(2^n)$), và thực nghiệm cũng cho thấy thời gian tăng gần gấp đôi mỗi khi n tăng thêm 1, chứng tỏ phù hợp với lý thuyết.
Về giới hạn giá trị n, cả hai bên đều cho kết quả tương đồng, khi chương trình bắt đầu chạy chậm rõ rệt từ khoảng n = 34, điều này khớp với dự đoán lý thuyết.
Cuối cùng, nguyên nhân sai lệch chủ yếu đến từ các yếu tố như độ trễ trong việc đo thời gian, các tối ưu của compiler, hoặc caching của CPU, tuy nhiên những sai lệch này không ảnh hưởng đáng kể đến kết luận tổng thể — rằng xu hướng và độ phức tạp của thuật toán hoàn toàn phù hợp với lý thuyết.

CODE :

```
#include <bits/stdc++.h>
```

```
using namespace std;
using namespace chrono;

long long fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}

long long fib_memo(int n, vector<long long> &dp) {
    if (n <= 1) return n;
    if (dp[n] != -1) return dp[n];
    return dp[n] = fib_memo(n - 1, dp) + fib_memo(n - 2, dp);
}

long long fib_iterative(int n) {
    if (n <= 1) return n;
    long long a = 0, b = 1, c;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}
long long sum_divide(vector<int> &arr, int l, int r) {
    if (l == r) return arr[l];
```

```
    int mid = (l + r) / 2;
    return sum_divide(arr, l, mid) + sum_divide(arr, mid + 1, r);
}

long long power_naive(long long a, int n) {
    long long res = 1;
    for (int i = 0; i < n; i++)
        res *= a;
    return res;
}

long long power_divide(long long a, int n) {
    if (n == 0) return 1;
    long long half = power_divide(a, n / 2);
    if (n % 2 == 0) return half * half;
    else return half * half * a;
}

void hanoi(int n, char from, char to, char aux) {
    if (n == 1) return;
    hanoi(n - 1, from, aux, to);
    hanoi(n - 1, aux, to, from);
}

template <typename Func>
double measureTime(Func f) {
```

```
    auto start = high_resolution_clock::now();
    f();
    auto end = high_resolution_clock::now();
    duration<double, milli> elapsed = end - start;
    return elapsed.count();
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cout << left << setw(5) << "n"
         << setw(18) << "Fib de quy (ms)"
         << setw(18) << "Fib co nho (ms)"
         << setw(18) << "Fib lap (ms)"
         << setw(18) << "Sum chia de tri (ms)"
         << setw(18) << "Pow Naive (ms)"
         << setw(18) << "Pow chia de tri (ms)"
         << setw(18) << "Hanoi (ms)"
         << "\n";

    for (int n = 1; n < 40; n++) {
        vector<int> arr(n, 1);
        vector<long long> dp(n + 1, -1);

        double t1 = measureTime([&]() { fib_recursive(n); });
```

```
        double t2 = measureTime([&]() { fib_memo(n, dp); });
        double t3 = measureTime([&]() { fib_iterative(n); });
        double t4 = measureTime([&]() { sum_divide(arr, 0, n - 1); });
        double t5 = measureTime([&]() { power_naive(2, n); });
        double t6 = measureTime([&]() { power_divide(2, n); });
        double t7 = measureTime([&]() { hanoi(n, 'A', 'C', 'B'); });

        cout << setw(5) << n
             << setw(18) << t1
             << setw(18) << t2
             << setw(18) << t3
             << setw(18) << t4
             << setw(18) << t5
             << setw(18) << t6
             << setw(18) << t7
             << "\n";
    }
    return 0;
}
```