# Yanovich Nikita Olegovich 348001

P.S. считывал программно через таймер. Взял числа меньше(10 20 30 40 50), т.к. для 100 ждать уже более 5 мин. Вывод: фиб числа с фиббоначи с рекурсией пипец долго.

## 1.1 РЕКУРС

#include <iostream>

#include <chrono>

using namespace std;

using namespace chrono;

long long fib\_recursive(int n) {

if (n <= 1) {

return n;

} else {

return fib\_recursive(n - 1) + fib\_recursive(n - 2);

}

}

void measure\_time\_recursive(int n) {

auto start = high\_resolution\_clock::now();

long long result = fib\_recursive(n);

auto end = high\_resolution\_clock::now();

duration<double> elapsed = end - start;

cout << "N: " << n << ", Result: " << result << ", Time: " << elapsed.count() << " seconds" << endl;

}

int main() {

int N\_values[] = {10, 20, 30, 40, 50};

cout << "Recursive Method:" << endl;

for (int N : N\_values) {

measure\_time\_recursive(N);

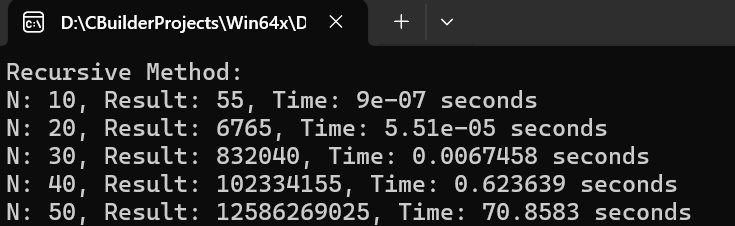
}

cout << "Press Enter to exit...";

cin.get(); // чтобы не закрылась

return 0;

}



## 1.2.1 ПРАВИЛЬНЫЙ

#include <iostream>

#include <chrono>

#include <vector>

using namespace std;

using namespace chrono;

long long fib\_array(int n) {

if (n <= 1) {

return n;

}

vector<long long> F(n + 1);

F[0] = 0;

F[1] = 1;

for (int i = 2; i <= n; ++i) {

F[i] = F[i - 1] + F[i - 2];

}

return F[n];

}

void measure\_time\_array(int n) {

auto start = high\_resolution\_clock::now();

long long result = fib\_array(n);

auto end = high\_resolution\_clock::now();

duration<double> elapsed = end - start;

cout << "N: " << n << ", Result: " << result << ", Time: " << elapsed.count() << " seconds" << endl;

}

int main() {

int N\_values[] = {10, 20, 30, 40, 50};

cout << "Iterative Method:" << endl;

for (int N : N\_values) {

measure\_time\_array(N);

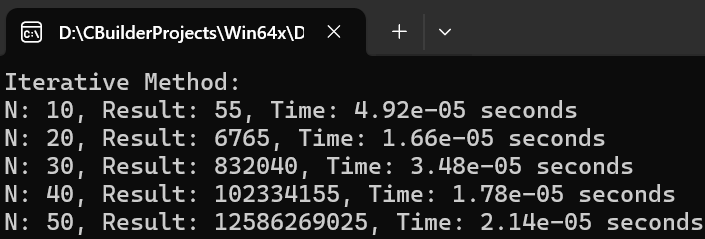
}

cout << "Press Enter to exit...";

cin.get(); // чтобы не закрылась

return 0;

}



Вывод: шустро делает, пробуем для больших:

## 1.2.2 ПРАВИЛЬНЫЙ 2

#include <iostream>

#include <chrono>

#include <vector>

using namespace std;

using namespace chrono;

long long fib\_array(int n) {

if (n <= 1) {

return n;

}

vector<long long> F(n + 1);

F[0] = 0;

F[1] = 1;

for (int i = 2; i <= n; ++i) {

F[i] = F[i - 1] + F[i - 2];

}

return F[n];

}

void measure\_time\_array(int n) {

auto start = high\_resolution\_clock::now();

long long result = fib\_array(n);

auto end = high\_resolution\_clock::now();

duration<double> elapsed = end - start;

cout << "N: " << n << ", Result: " << result << ", Time: " << elapsed.count() << " seconds" << endl;

}

int main() {

int N\_values[] = {100, 250, 500, 1000, 5000};

cout << "Iterative Method:" << endl;

for (int N : N\_values) {

measure\_time\_array(N);

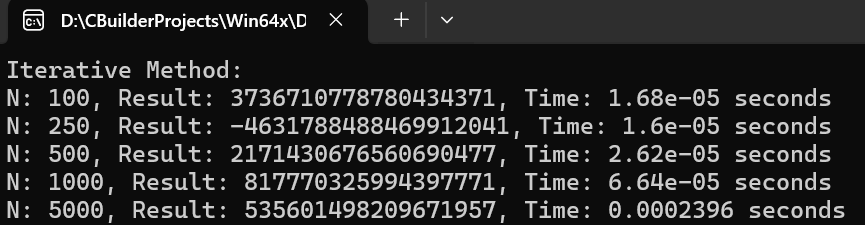
}

cout << "Press Enter to exit...";

cin.get(); // чтобы не закрылась

return 0;

}



Вывод: даже для больших делает шустрей, чем рекурсивная для маленьких.

## 2.1 НАТИВ

#include <iostream>

#include <chrono>

using namespace std;

using namespace chrono;

int naiveGCD(int a, int b) {

int gcd = 1;

for (int d = 2; d <= max(a, b); ++d) {

if (a % d == 0 && b % d == 0) {

gcd = d;

}

}

return gcd;

}

void measure\_time\_naiveGCD(int a, int b) {

auto start = high\_resolution\_clock::now();

int result = naiveGCD(a, b);

auto end = high\_resolution\_clock::now();

duration<double> elapsed = end - start;

cout << "a: " << a << ", b: " << b << ", GCD: " << result << ", Time: " << elapsed.count() << " seconds" << endl;

}

int main() {

pair<int, int> N\_values[] = {

{8723462536745223, 765412221883},

{2973948729384212, 87268364827634},

{1098304287349821, 125367765723},

{2987349872934234, 78238486224}

};

cout << "Naive GCD Method:" << endl;

for (auto p : N\_values) {

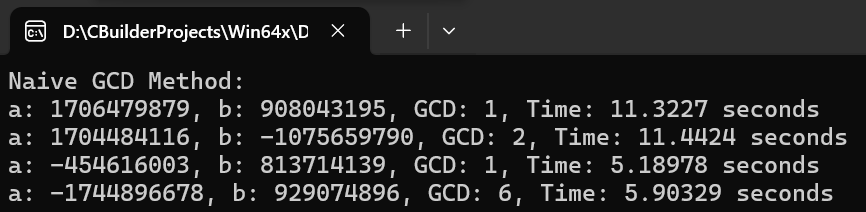
measure\_time\_naiveGCD(p.first, p.second);

}

cin.get(); // чтобы не закрылась

return 0;

}



## 2.2 ЕВКЛИД

#include <iostream>

#include <chrono>

#include <stdexcept>

using namespace std;

using namespace chrono;

long long euclidGCD(long long a, long long b) {

if (a == 0) return b;

if (b == 0) return a;

while (b != 0) {

long long temp = b;

b = a % b;

a = temp;

}

return a;

}

void measure\_time\_euclidGCD(long long a, long long b) {

try {

if (a < 0 || b < 0) throw invalid\_argument("Negative values are not allowed.");

if (a > LLONG\_MAX || b > LLONG\_MAX) throw overflow\_error("Values are too large.");

auto start = high\_resolution\_clock::now();

long long result = euclidGCD(a, b);

auto end = high\_resolution\_clock::now();

duration<double> elapsed = end - start;

cout << "a: " << a << ", b: " << b << ", GCD: " << result << ", Time: " << elapsed.count() << " seconds" << endl;

} catch (const exception& e) {

cerr << "Exception: " << e.what() << endl;

}

}

int main() {

pair<long long, long long> N\_values[] = {

{8723462536745223, 765412221883},

{2973948729384212, 87268364827634},

{1098304287349821, 125367765723},

{2987349872934234, 78238486224}

};

cout << "Euclid GCD Method:" << endl;

for (auto p : N\_values) {

measure\_time\_euclidGCD(p.first, p.second);

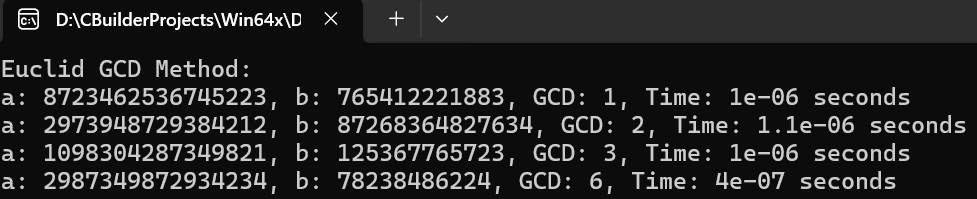
}

cout << "Press Enter to exit...";

cin.get(); // Ожидание ввода от пользователя

return 0;

}



Вывод: Евклид шустрее.

ВЫВОД ПО РАБОТЕ: Для написания программы нужно учитывать время выполнения, занимаемую память и числа, с которыми работаем, чтобы достигнуть максимально эффективного выполнения для частного случая.