**Анализ алгоритмов вычисления ряда Фибоначчи.**

Ionasco Gheorghe

FAF-193

**1.Рекурсия**

Прямая рекурсивная реализация

#include <iostream>

#include <time.h>

using namespace std;

void printTimeSince(clock\_t tStart)

{

printf("Time taken: %.2fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

}

int fib(int n)

{

if (n <= 1)

return n;

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

}

int main ()

{

int n;

cout<<"n=";

cin>>n;

clock\_t tStart = clock();

cout << fib(n)<<endl;

printf("Time taken: %.9fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

return 0;

}

**Метод 2**

Существует более эффективное решение данной задачи с помощью быстрого возведения матрицы в степень. Оно основано на следующем [тождестве](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8):

fibonaccimatrix

#include <bits/stdc++.h>

using namespace std;

void multiply(int F[2][2], int M[2][2]);

void power(int F[2][2], int n);

int fib(int n)

{

int F[2][2] = {{1, 1}, {1, 0}};

if (n == 0)

return 0;

power(F, n - 1);

return F[0][0];

}

void power(int F[2][2], int n)

{

if(n == 0 || n == 1)

return;

int M[2][2] = {{1, 1}, {1, 0}};

power(F, n / 2);

multiply(F, F);

if (n % 2 != 0)

multiply(F, M);

}

void multiply(int F[2][2], int M[2][2])

{

int x = F[0][0] \* M[0][0] + F[0][1] \* M[1][0];

int y = F[0][0] \* M[0][1] + F[0][1] \* M[1][1];

int z = F[1][0] \* M[0][0] + F[1][1] \* M[1][0];

int w = F[1][0] \* M[0][1] + F[1][1] \* M[1][1];

F[0][0] = x;

F[0][1] = y;

F[1][0] = z;

F[1][1] = w;

}

void printTimeSince(clock\_t tStart)

{

printf("Time taken: %.2fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

}

int main()

{

int n;

cout<<"n=";

cin>>n;

clock\_t tStart = clock();

cout << fib(n)<<endl;

printf("Time taken: %.9fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

return 0;

}

**Метод 3 – Формула Бине**

Fn=φn−ψn√5,

#include<iostream>

#include<cmath>

#include <time.h>

using namespace std;

int fib(int n) {

double phi = (1 + sqrt(5)) / 2;

return round(pow(phi, n) / sqrt(5));

}

void printTimeSince(clock\_t tStart)

{

printf("Time taken: %.18fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

}

int main ()

{

int n;

cout<<"n=";

cin>>n;

clock\_t tStart = clock();

cout << fib(n)<<endl;

printf("Time taken: %.18fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

return 0;

}

**Метод 4 O(Log n) Time**

If n is even then k = n/2:

F(n) = [2\*F(k-1) + F(k)]\*F(k)

If n is odd then k = (n + 1)/2

F(n) = F(k)\*F(k) + F(k-1)\*F(k-1)

----------------------------------------------------------------------------------------------

#include <iostream>

#include <time.h>

using namespace std;

const int MAX = 10000;

int f[MAX] = {0};

void printTimeSince(clock\_t tStart)

{

printf("Time taken: %.18fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

}

int fib(int n)

{

if (n == 0)

return 0;

if (n == 1 || n == 2)

return (f[n] = 1);

if (f[n])

return f[n];

int k = (n & 1)? (n+1)/2 : n/2;

f[n] = (n & 1)? (fib(k)\*fib(k) + fib(k-1)\*fib(k-1))

: (2\*fib(k-1) + fib(k))\*fib(k);

return f[n];

}

int main()

{

int n;

cout<<"n=";

cin>>n;

clock\_t tStart = clock();

cout << fib(n)<<endl;

printf("Time taken: %.18fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

return 0;

}

Вывод:

Лучший способ реализовать такие алгоритмы - использовать формулы, а не прямые вычисления, такой как рекурсия, потому что, как показывают результаты, для вычисления больших чисел требуется больше времени, просто потому, что программа выполняет много вычислений, чтобы вернуть только один номер. Лучиший метод –> это формула Бине,так как она требует меньше всего памяти и времени,и для всего вычисления требуется лишь 1 формула. Но в то же время 2 3 4 способ близки по времени поэтому 2 и 4 также можно использовать.