Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2
дисциплины «Алгоритмизация»
Вариант___

Выполнил:
Репкин Александр Павлович
1 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., канд. техн. наук,
доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. На основе псевдокодов, рассмотренных на лекции, изучена разница между наивными и оптимизированными алгоритмами. Рассмотрены алгоритмы нахождения числа Фибоначчи и Наибольшего Общего Делителя (НОД).

2. Рассмотрены алгоритмы нахождения числа Фибоначчи. Построен график относительно полученной статистики занимаемого времени.

```cpp
#include <iostream>
#include <iomanip>
#include <ctime>

using std::cout;
using std::cin;
using std::time;

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

int main() {
    int needed_number, places;
    for (int i = 30; i < 45; i++) {
        cout << "\nThe needed place of Fibonachi Numbers is " << i << "\n";
        clock_t start_time = clock();
        needed_number = FibRecursive(i);
        clock_t end_time = clock();
        double spent_time = (end_time - start_time) / double(CLOCKS_PER_SEC);
        cout << std::setprecision(8) << "Value on this place is " << needed_number << "\n\nStart time - " << start_time
    }
}
```

Рисунок 1. Наивный код.

```cpp
#include <iostream>
#include <ctime>
#include <iomanip>

using std::cout;
using std::cin;
using std::time;

int FibArray(int n) {
    int F[50] {0};
    F[1] = 1;
    for (int i = 2; i < n; i++) {
        F[i] = F[i - 1] + F[i - 2];
    }
    return F[n-1];
}

int main() {
    int needed_number, places;
    for (int i = 30; i < 45; i++) {
        cout << "\nThe needed place of Fibonachi Numbers is " << i << "\n";
        clock_t start_time = clock();
        needed_number = FibArray(i);
        clock_t end_time = clock();
        double spent_time = (end_time - start_time) / double(CLOCKS_PER_SEC);
        cout << std::setprecision(8) << "Value on this place is " << needed_number << "\n\nStart time - " << start_time
    }
}
```

Рисунок 2. Оптимизированный код.

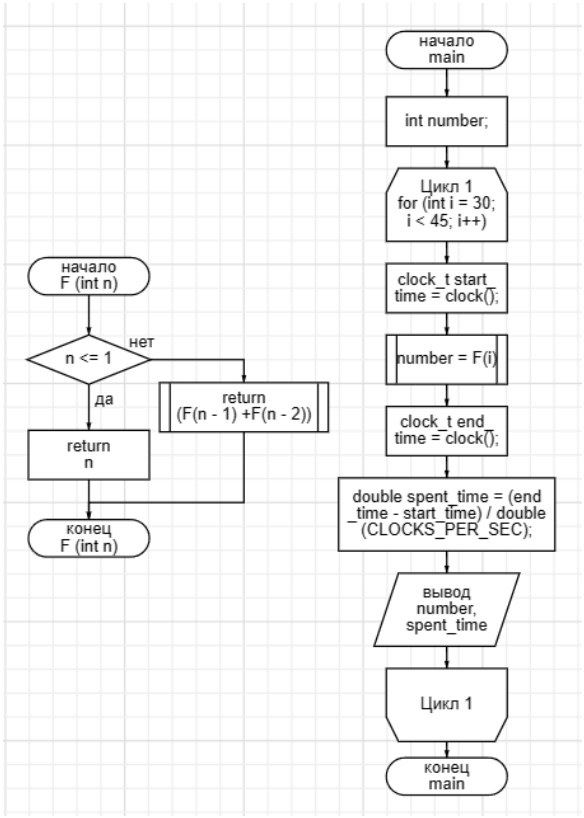

Рисунок 3. Блок-схема Наивного кода.

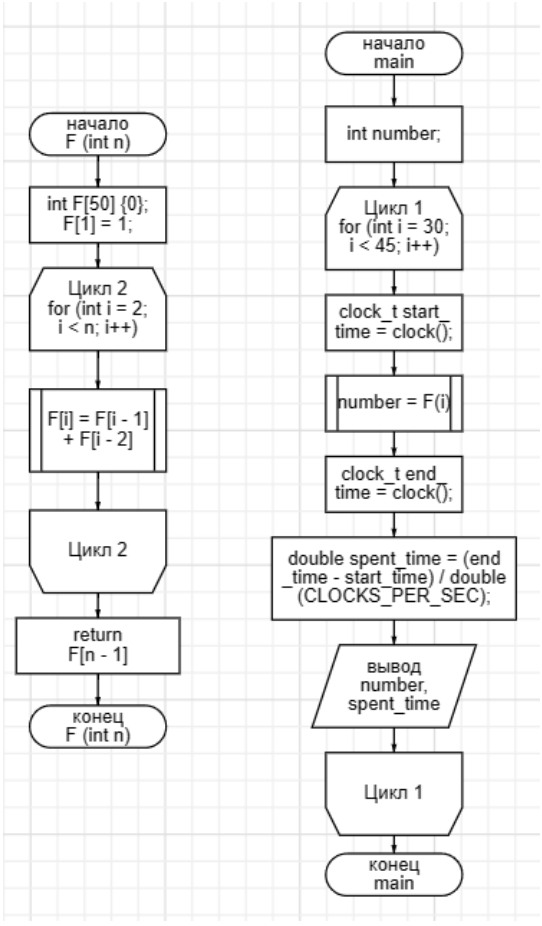

Рисунок 4. Блок-схема Оптимизированного кода.

Значение	Время Наивное	Время Оптмизированное
30	0,01	0
31	0,017	0
32	0,025	0
33	0,042	0
34	0,068	0
35	0,108	0
36	0,18	0
37	0,294	0
38	0,469	0
39	0,747	0
40	1,201	0
41	1,95	0
42	3,142	0
43	5,089	0
44	8,232	0

Рисунок 5. Полученный график.

3. Рассмотрены алгоритмы нахождения НОД. Построен график относительно полученной статистики занимаемого времени.

```cpp
#include <iostream>
#include <iomanip>
#include <ctime>
using std::time;

int NaiveGCD(int a, int b) {
    int gcd = 1;
    for (int d = 2; d < std::max(a, b); d++)
        if (a % d == 0 && b % d == 0) gcd = d;
    return gcd;
}

int main() {
    int needed_number, places;
    for (int i = 1200000; i < 1400000; i += 13837) {
        clock_t start_time = clock();
        needed_number = NaiveGCD(i, 123456789);
        clock_t end_time = clock();
        double spent_time = (end_time - start_time) / double(CLOCKS_PER_SEC);
        std::cout << std::setprecision(8) << "\n\nValues are " << i << " and 5237. Answer is " << needed_number
    }
}
```

Рисунок 6. Наивный код.

```cpp
#include <iostream>
#include <iomanip>
#include <ctime>
using std::time;

int EuclidGCD(int a, int b) {
    if (a == 0) return b;
    if (b == 0) return a;
    if (a >= b) return EuclidGCD(a % b, b);
    return EuclidGCD(a, b % a);
}

int main() {
    int needed_number, places;
    for (int i = 1200000; i < 1400000; i += 13837) {
        clock_t start_time = clock();
        needed_number = EuclidGCD(i, 123456789);
        clock_t end_time = clock();
        double spent_time = (end_time - start_time) / double(CLOCKS_PER_SEC);
        std::cout << std::setprecision(8) << "\n\nValues are " << i << " and 5237. Answer is " << needed_number
    }
}
```

Рисунок 7. Оптимизированный код.

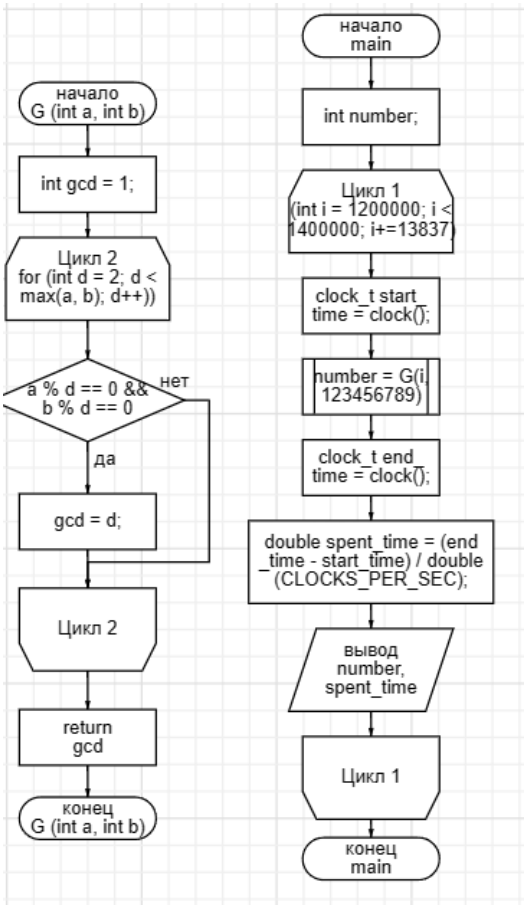

Рисунок 8. Блок-схема Наивного кода.

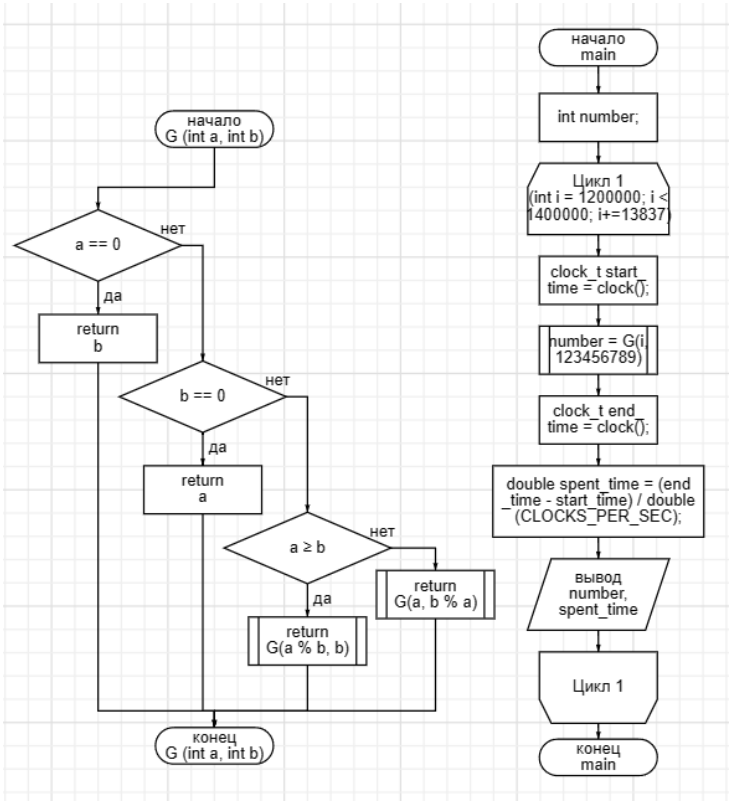

Рисунок 9. Блок-схема Оптимизированного кода.

Значение	Время Наивное	Время Оптмизированное
1200000	0,513	0
1213837	0,517	0
1227674	0,496	0
1241511	0,491	0
1255348	0,491	0
1269185	0,492	0
1283022	0,496	0
1296859	0,496	0
1310696	0,498	0
1324533	0,491	0
1338370	0,491	0
1352207	0,491	0
1366044	0,493	0
1379881	0,492	0
1393718	0,492	0

Рисунок 10. Полученный график.

Вывод: в ходе выполнения практической работы, была рассмотрена разница в эффективности между примерами наивных и оптимизированных кодов. Из полученных данных получено понимание важности оптимизации.