

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 14**

Выполнил:  
Касимов Асхаб Арсенович  
2 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. технических  
наук, доцент кафедры  
инфокоммуникаций

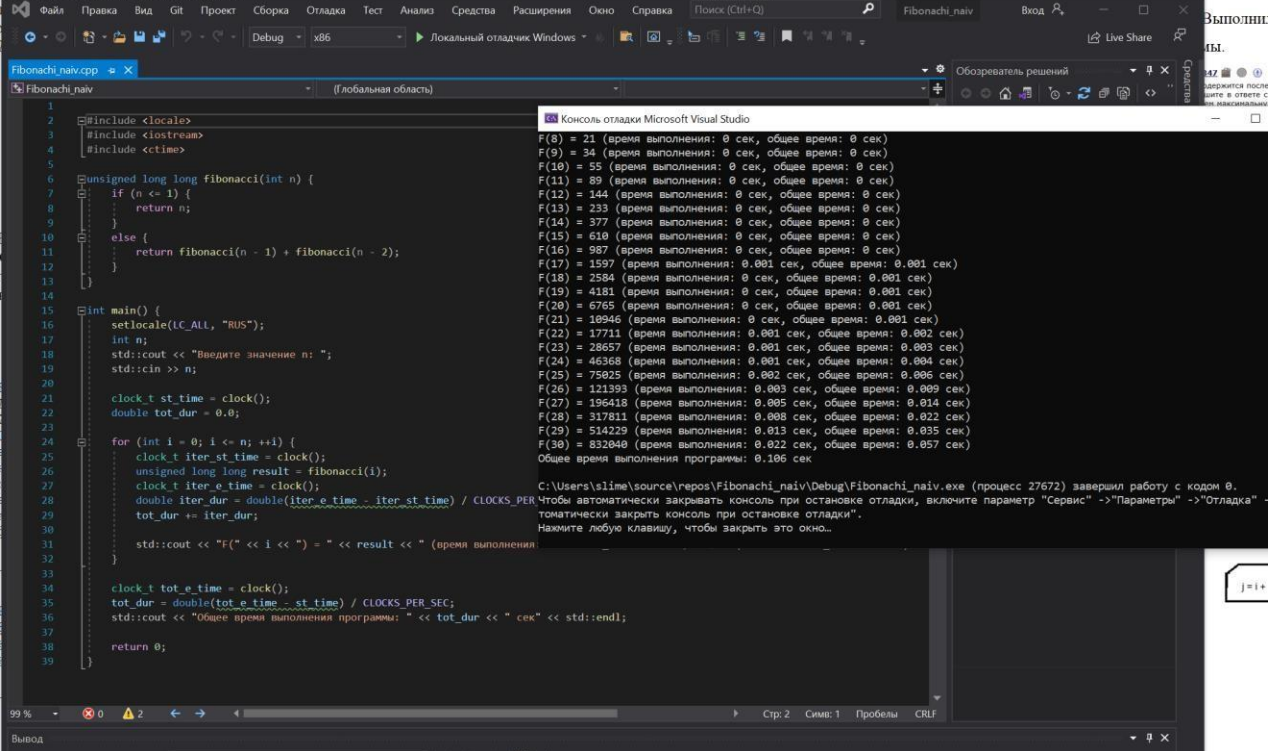
\_\_\_\_\_  
(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

## Ход работы

1. Написал программу, которая считает числа Фибоначчи через рекурсивную функцию и измерил время за которое она рассчитывает каждое число.



```
1 #include <locale>
2 #include <iostream>
3 #include <ctime>
4
5
6 unsigned long long fibonacci(int n) {
7     if (n <= 1) {
8         return n;
9     }
10    else {
11        return fibonacci(n - 1) + fibonacci(n - 2);
12    }
13 }
14
15 int main() {
16     setlocale(LC_ALL, "RUS");
17     int n;
18     std::cout << "Введите значение n: ";
19     std::cin >> n;
20
21     clock_t st_time = clock();
22     double tot_dur = 0.0;
23
24     for (int i = 0; i <= n; ++i) {
25         clock_t iter_st_time = clock();
26         unsigned long long result = fibonacci(i);
27         clock_t iter_e_time = clock();
28         double iter_dur = double(iter_e_time - iter_st_time) / CLOCKS_PER_SEC;
29         tot_dur += iter_dur;
30
31         std::cout << "F(" << i << ") = " << result << " (время выполнения: " << iter_dur << " сек, общее время: " << tot_dur << " сек) \n";
32     }
33
34     clock_t tot_e_time = clock();
35     tot_dur = double(tot_e_time - st_time) / CLOCKS_PER_SEC;
36     std::cout << "Общее время выполнения программы: " << tot_dur << " сек" << std::endl;
37
38     return 0;
39 }
```

Консоль отладки Microsoft Visual Studio

```
F(8) = 21 (время выполнения: 0 сек, общее время: 0 сек)
F(9) = 34 (время выполнения: 0 сек, общее время: 0 сек)
F(10) = 55 (время выполнения: 0 сек, общее время: 0 сек)
F(11) = 89 (время выполнения: 0 сек, общее время: 0 сек)
F(12) = 144 (время выполнения: 0 сек, общее время: 0 сек)
F(13) = 233 (время выполнения: 0 сек, общее время: 0 сек)
F(14) = 377 (время выполнения: 0 сек, общее время: 0 сек)
F(15) = 610 (время выполнения: 0 сек, общее время: 0 сек)
F(16) = 987 (время выполнения: 0 сек, общее время: 0 сек)
F(17) = 1597 (время выполнения: 0.001 сек, общее время: 0.001 сек)
F(18) = 2584 (время выполнения: 0 сек, общее время: 0.001 сек)
F(19) = 4181 (время выполнения: 0 сек, общее время: 0.001 сек)
F(20) = 6765 (время выполнения: 0 сек, общее время: 0.001 сек)
F(21) = 10946 (время выполнения: 0 сек, общее время: 0.001 сек)
F(22) = 17711 (время выполнения: 0.001 сек, общее время: 0.002 сек)
F(23) = 28657 (время выполнения: 0.001 сек, общее время: 0.003 сек)
F(24) = 46368 (время выполнения: 0.001 сек, общее время: 0.004 сек)
F(25) = 75025 (время выполнения: 0.002 сек, общее время: 0.006 сек)
F(26) = 121393 (время выполнения: 0.003 сек, общее время: 0.009 сек)
F(27) = 196413 (время выполнения: 0.005 сек, общее время: 0.014 сек)
F(28) = 317811 (время выполнения: 0.008 сек, общее время: 0.022 сек)
F(29) = 514229 (время выполнения: 0.013 сек, общее время: 0.035 сек)
F(30) = 832040 (время выполнения: 0.022 сек, общее время: 0.057 сек)
Общее время выполнения программы: 0.106 сек
```

C:\Users\allime\source\repos\Fibonacci\_naiv\Debug\Fibonacci\_naiv.exe (процесс 27672) завершил работу с кодом 0. Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка". Чтобы автоматически закрывать консоль при остановке отладки, нажмите любую клавишу, чтобы закрыть это окно.

Рисунок 1. Результат работы программы

2. Написал программу, считающую числа Фибоначчи с записью их в массив для исключения многочисленных повторяющихся вычислений. Также добавил цикл, делающий 1000000 итераций для каждого числа, чтобы появилась возможность измерить время, поскольку программа работала слишком быстро.

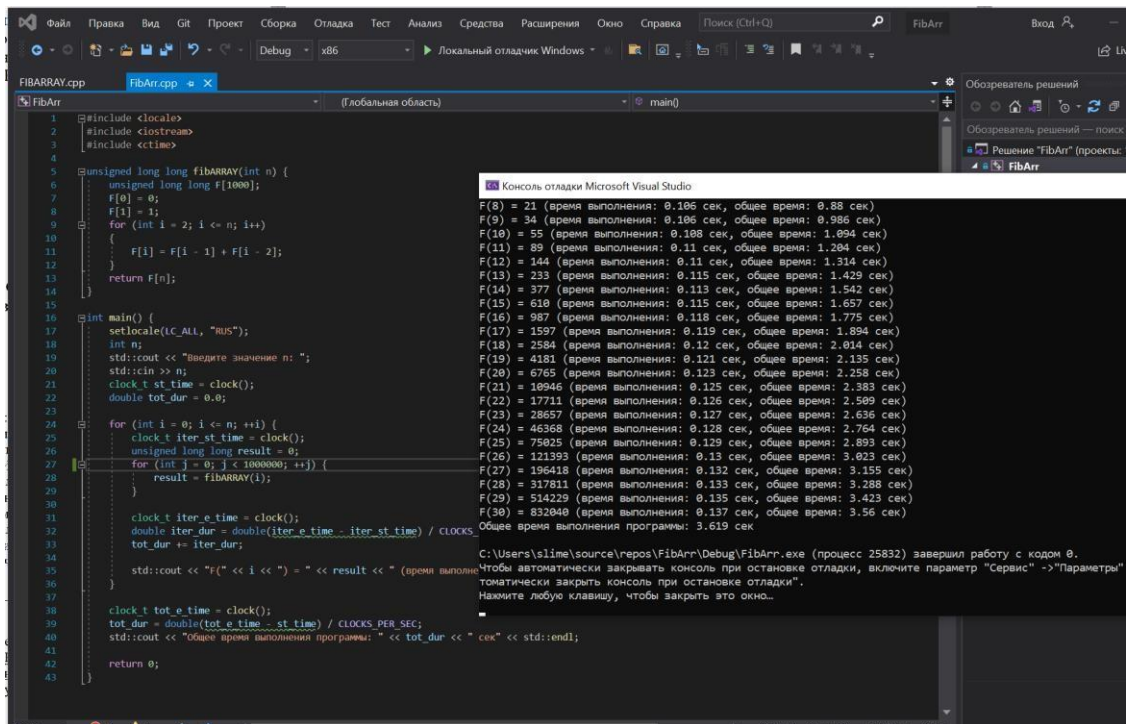


Рисунок 2. Результат программы для первых 30 чисел последовательности

3. Собрал данные и сделал наглядную демонстрацию разницы в эффективности этих двух программ с помощью таблиц excel.

Таблица 1. Время работы алгоритмов вычисления чисел Фибоначчи

n	time (s) NAIV	time (s) ARR
0	0,001	0,000000239
1	0,001	0,000000463
2	0,001	0,000000681
3	0,001	0,000000918
4	0,001	0,000001218
5	0,001	0,000001465
6	0,001	0,000001705
7	0,001	0,000001946
8	0,001	0,000002216
9	0,001	0,000002477
10	0,001	0,000002748
11	0,001	0,000003032
12	0,001	0,000003328
13	0,001	0,000003616
14	0,001	0,000003896
15	0,001	0,000004213
16	0,001	0,000004516
17	0,002	0,000004816
18	0,003	0,000005102
19	0,003	0,000005393
20	0,004	0,000005714
21	0,004	0,000006042

22	0,006	0,000006376
23	0,008	0,000006718
24	0,011	0,000007049
25	0,016	0,000007379
26	0,024	0,000007689
27	0,037	0,000008083
28	0,057	0,000008408
29	0,088	0,000008773
30	0,139	0,000009147
31	0,221	0,000009525
32	0,35	0,000009882
33	0,583	0,000010253
34	0,934	0,000010595
35	1,507	0,000010949
36	2,422	0,000011335
37	3,913	0,000011699
38	6,364	0,000012087
39	10,245	0,000012467
40	16,297	0,000012839
41	26,601	0,000013247
42	43,184	0,000013653
43	69,842	0,000014042
44	112,924	0,000014459
45	182,742	0,000014883
46	295,9	0,000015295
47	479,508	0,000015728
48	782,629	0,000016145
49	1270,29	0,000016537
50	2067,22	0,000016961

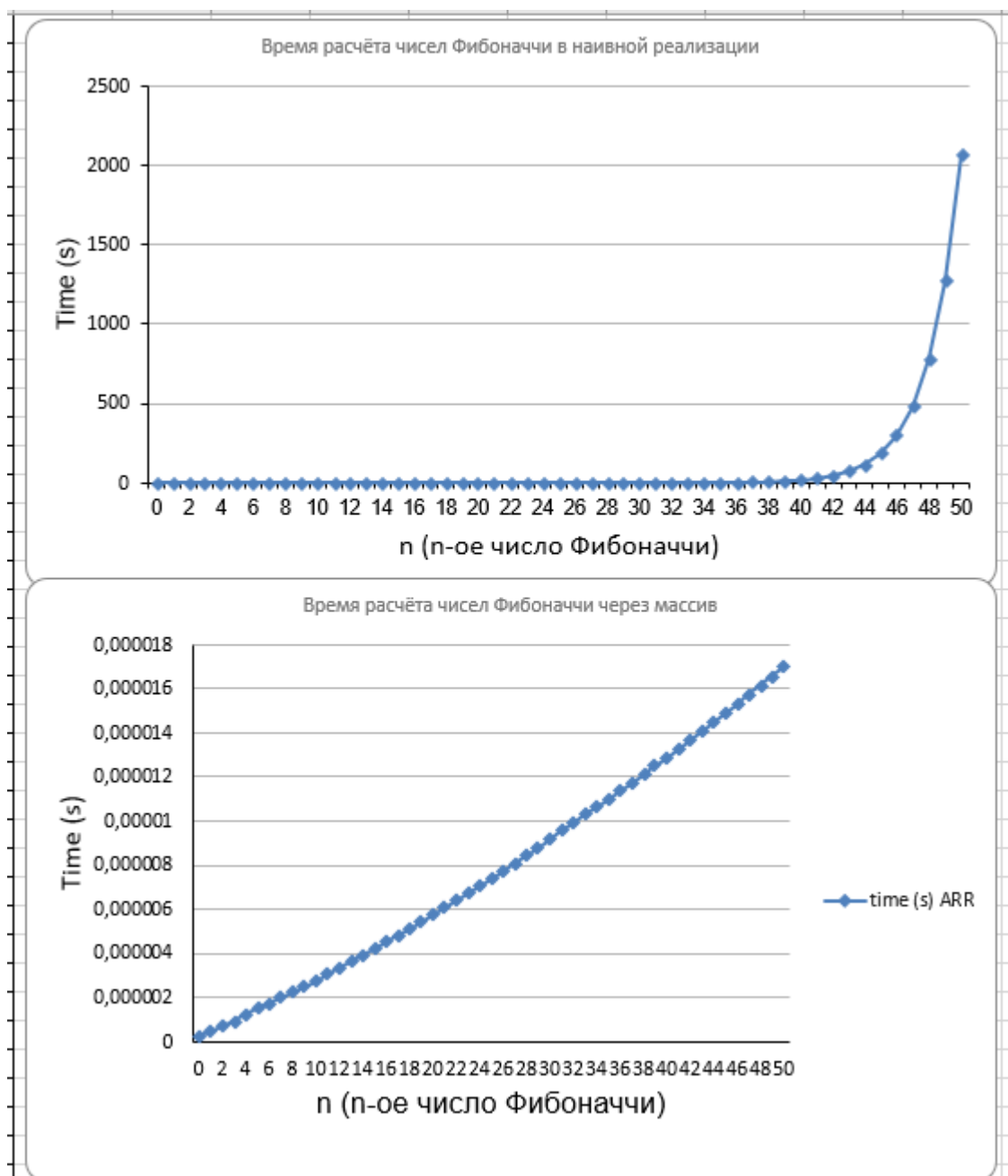


Рисунок 3. Графики для чисел Фибоначчи

4. Написал программу, вычисляющую наибольший общий делитель для двух целых чисел через цикл с перебором всех возможных целых чисел, которые меньше максимального из пары данных.

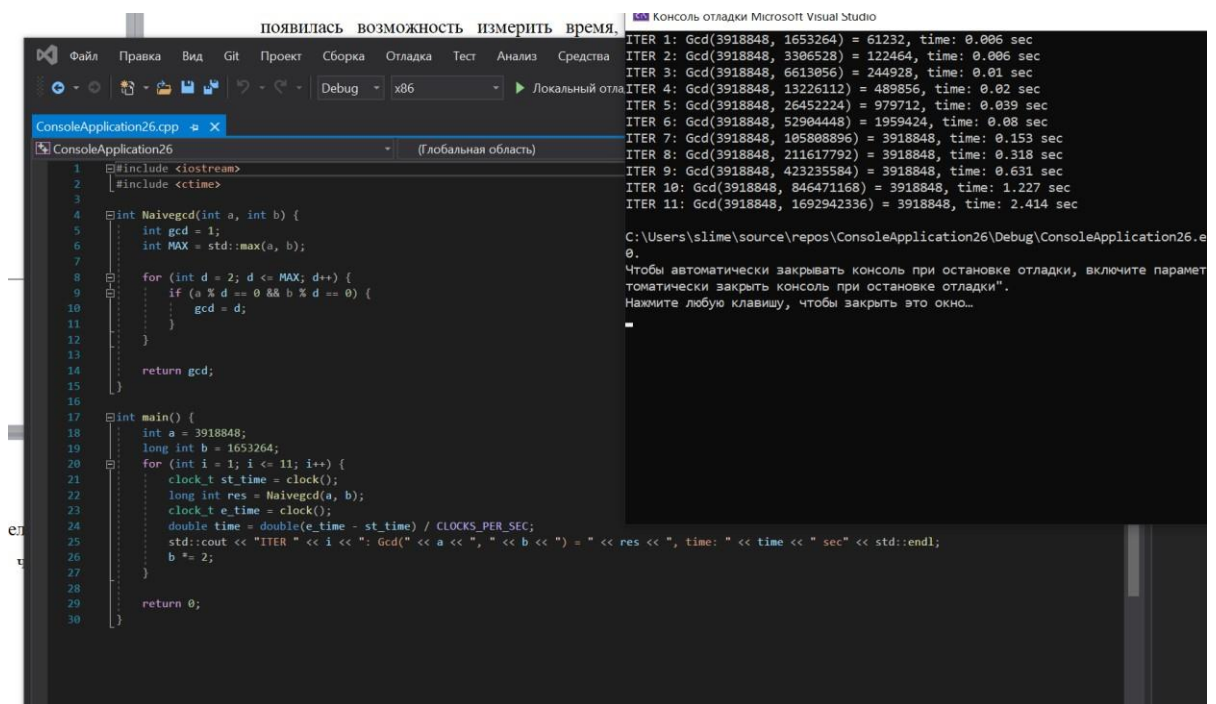


Рисунок 4. Результат работы программы (НОД через перебор)

5. Разработал программу, выполняющую расчёт НОД для двух целых чисел с помощью алгоритма Евклида.

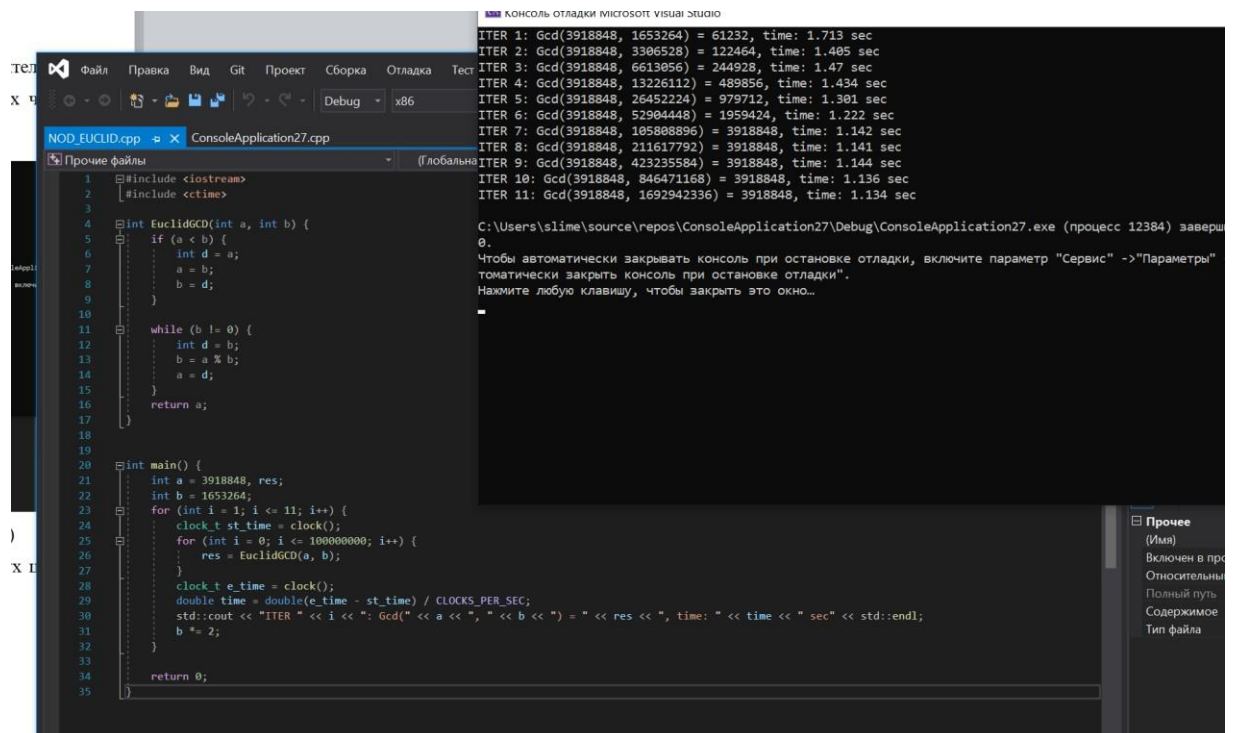


Рисунок 5. Результат работы программы (НОД через алгоритм Евклида)

6. Собрал данные из консоли и сделал графическую демонстрацию превосходства второго метода над первым.

Таблица 2. Время работы алгоритмов вычисления НОД

a	b	time (s) Naiv	time (s) Euclid
3918848	1653264	0,006	0,00000001683
3918848	3306528	0,007	0,00000001389
3918848	6613056	0,01	0,00000001447
3918848	13226112	0,02	0,00000001434
3918848	26452224	0,038	0,00000001305
3918848	52904448	0,076	0,00000001225
3918848	105808896	0,175	0,00000001175
3918848	211617792	0,311	0,0000000114
3918848	423235584	0,623	0,00000001149
3918848	846471168	1,21	0,00000001155
3918848	1692942336	2,413	0,0000000114

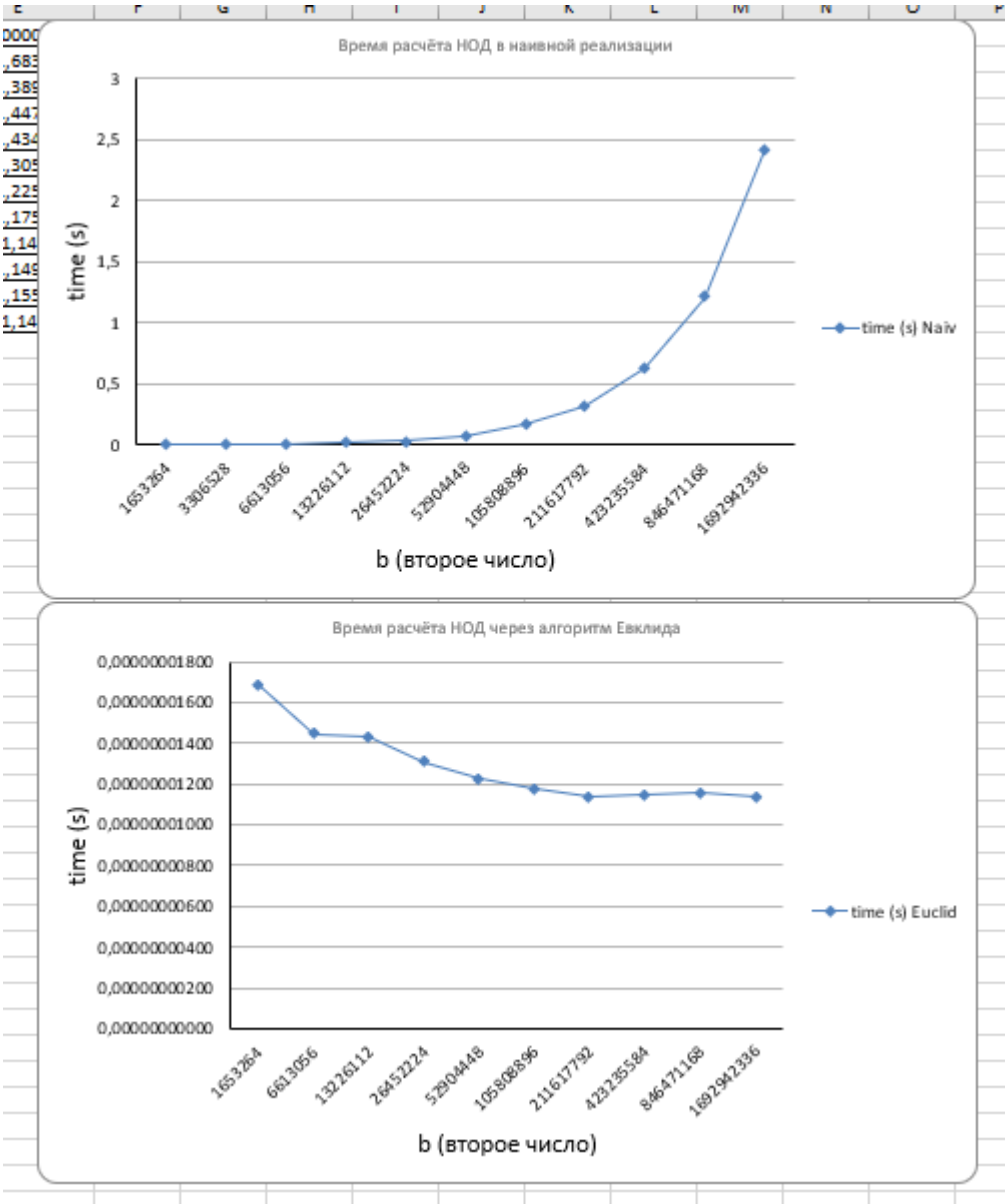


Рисунок 6. Графики для НОД

Вывод: в ходе выполнения лабораторной работы был проведен сравнительный анализ наивных и оптимизированных алгоритмов для решения задач. Из полученных результатов можно сделать следующий вывод: наивные реализации алгоритмов зачастую показывают значительно меньшую производительность. На примере алгоритма поиска НОД для двух целых чисел и вычисления чисел Фибоначчи оптимизированный алгоритм показал эффективность в десятки тысяч раз превосходящую таковую у наивной реализации. Таким образом, использование наивных алгоритмов является нецелесообразным, что значит, что при разработке программы необходимо обращать внимание на оптимизацию и доработку.