

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»**

Выполнил:
Степанов Леонид Викторович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение
средств вычислительной
техники и автоматизирование
систем», очная форма обучения

(подпись)

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

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. Написал программу (fib.py), рассчитывающую число Фибоначчи методом нативного алгоритма, рассчитал время выполнения программы, для чисел Фибоначчи от 0 до 9:

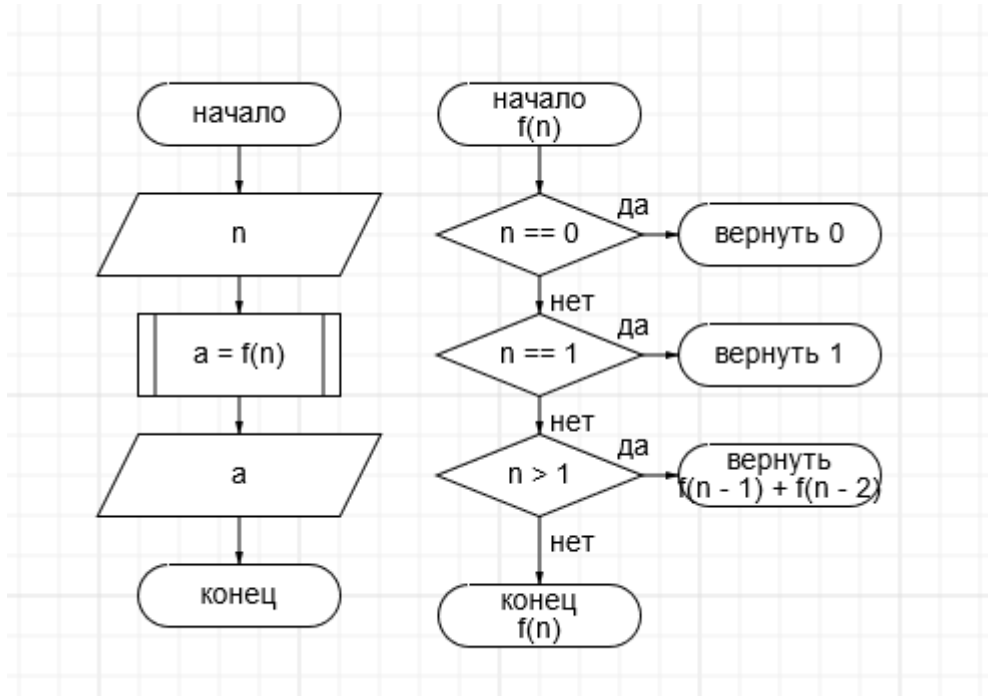


Рисунок 1. Блок-схема нахождения числа Фибоначчи

```
1 def f(n):
2     if n == 0:
3         return 0
4     if n == 1:
5         return 1
6     if n > 1:
7         return f(n - 1) + f(n - 2)
8
9 if __name__ == '__main__':
10     import timeit
11     for i in range(0, 10):
12         a = f(f(i))
13         print(str(f(f(i))) + ", timeit.timeit(stmt=a, setup='from __main__ import f')")
14
```

Run main

```
C:\Users\Neo\PycharmProjects\pythonProject2\venv\bin\python.exe C:\Users\Neo\PycharmProjects\pythonProject2\main.py
f(0) = 0.10087520000161021
f(1) = 0.11326650000046357
f(2) = 0.381085799997097
f(3) = 0.6537223999985144
f(4) = 1.205377000002045
f(5) = 2.016426700000011
f(6) = 3.3407750000005763
f(7) = 5.400266200002079
f(8) = 8.943360099998245
f(9) = 14.466471499999898
```

Рисунок 2. Результат выполнения программы fib.py

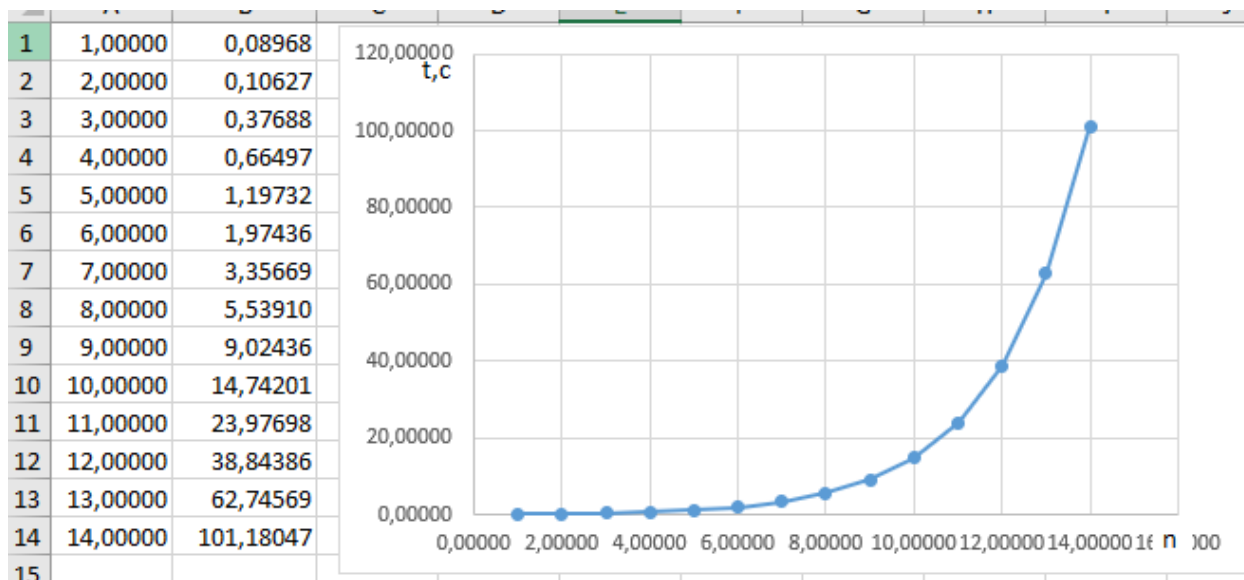


Рисунок 3. Данные и график зависимости порядка числа Фибоначчи от времени нахождения

2. Написал программу (exfib.py), которая рассчитывает число Фибоначчи эффективнее чем программа выше (fib.py), рассчитал время выполнения программы на числа Фибоначчи от 3 до 9

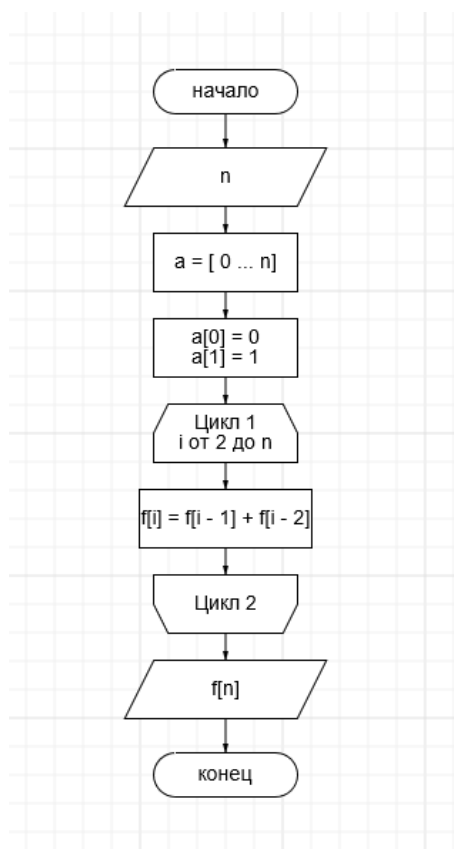


Рисунок 4. Блок-схема нахождения числа Фибоначчи

```

1 if __name__ == '__main__':
2     import time
3     b = 0
4     for n in range(3, 10):
5         for j in range(1, 1000000):
6             start = time.perf_counter()
7             a = [1 * 0 for i in range(n + 1)]
8             a[0] = 0
9             a[1] = 1
10            for i in range(2, n + 1):
11                a[i] = a[i - 1] + a[i - 2]
12            end = time.perf_counter()
13            b += (end - start)
14        b = b / 1000000
15        print(f'{n} ', f'{a[n - 1]} ', f'{b:.8f} ')
16    b = 0

```

Run main

```

C:\Users\Neo\PycharmProjects\pythonProject2\venv\bin\python.exe C:\Users\Neo\PycharmProjects\pythonProject2\main.py
3 1 0.000001515
4 2 0.000001835
5 3 0.000002090
6 5 0.000002349
7 8 0.000002590
8 13 0.000002886
9 21 0.000003121

```

Process finished with exit code 0

Рисунок 5. Результат выполнения программы fib.py

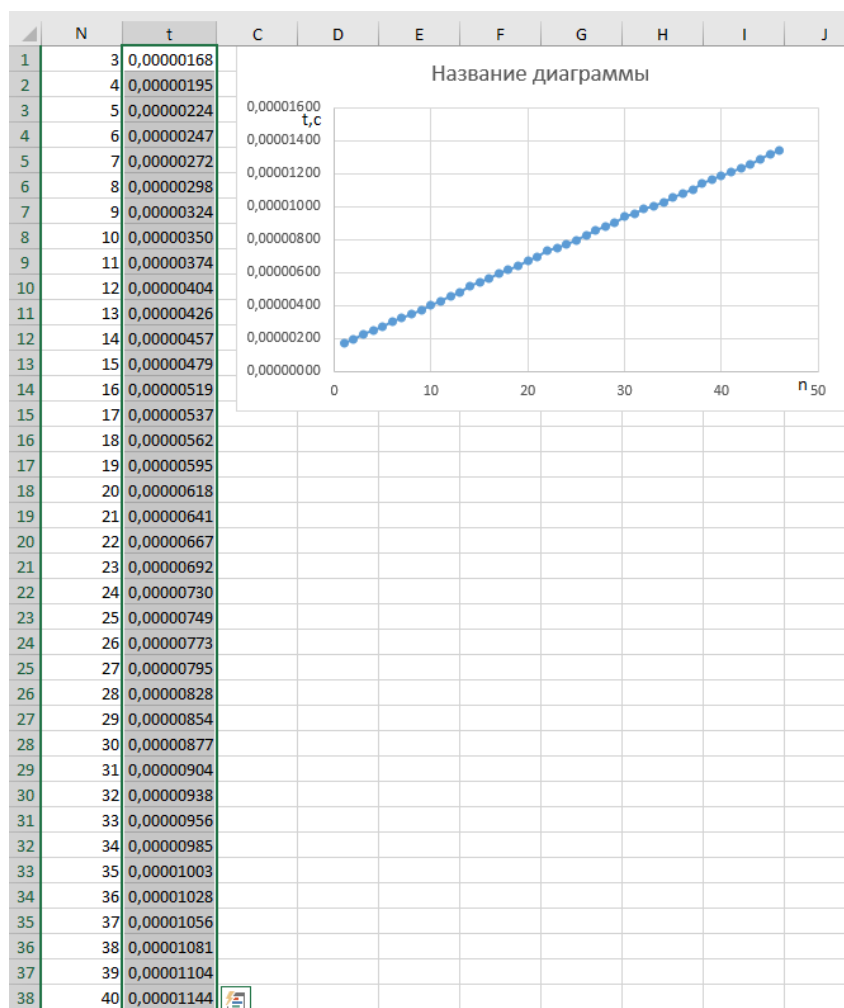


Рисунок 6. Данные и график зависимости порядка числа Фибоначчи от времени нахождения

3. Написал программу (nod.py), которая ищет наибольший общий делитель среди 2-х чисел, она рассчитывает НОД для числа 3918848 и числа от 208 до этого числа с шагом 200. Этот алгоритм рассчитывает методом перебора всех значений от максимального в паре до 2 ищет число одновременно делящееся на друг друга

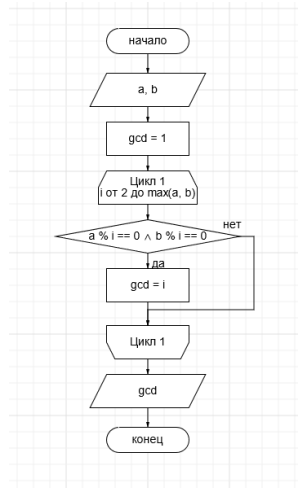


Рисунок 7. Блок-схема нахождения НОД двух чисел

```
1 def nod(a, b):
2     gcd = 1
3     for i in range(2, max(a, b)):
4         if a % i == 0 and b % i == 0:
5             gcd = i
6     return gcd
7
8 if __name__ == '__main__':
9     import time
10    b = 0
11    for i in range(208, 3918848, 200):
12        for j in range(10):
13            start = time.perf_counter()
14            a = nod(3918848, i)
15            end = time.perf_counter()
16            b += end - start
17            if j == 10-1:
18                print(f"{i}; {b/(10-1)}")
19                b = 0
```

Run main

```
208; 0.27707328889098587
408; 0.27752186666637296
608; 0.2817346333322348
808; 0.27800492222175105
1008; 0.2802437222198848
1208; 0.2842912111098283
1408; 0.2813731555571495
1608; 0.27929400000236154
1808; 0.28576175555483335
2008; 0.2798719111063595
```

Рисунок 8. Результат выполнения программы nod.py

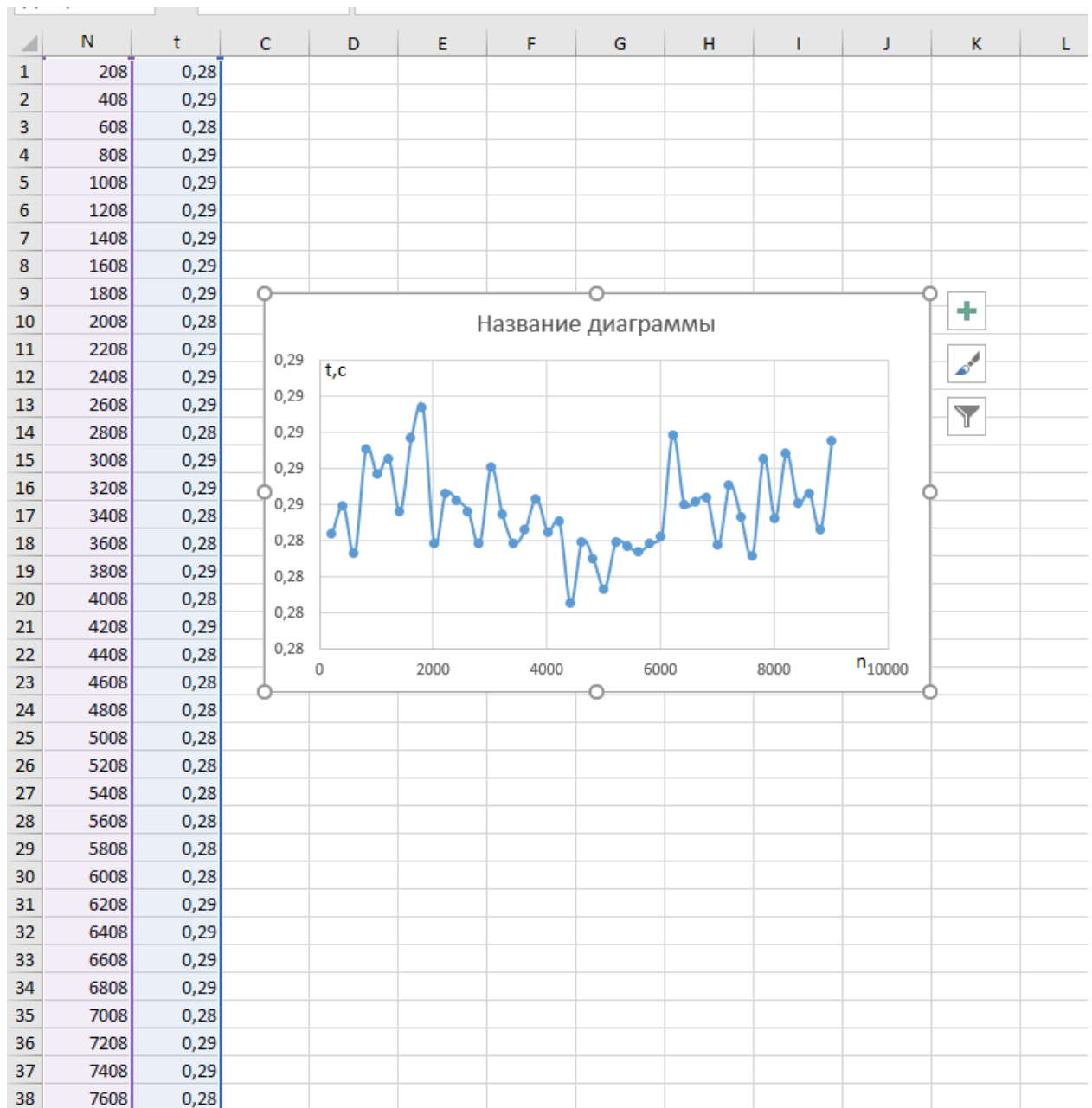


Рисунок 9. Данные и график зависимости второго числа от времени

4. Написал программу (exnod.py), которая ищет наибольший общий делитель среди 2-х чисел, она рассчитывает НОД для числа 3918848 и числа от 200 до этого числа с шагом 208. Этот алгоритм основан на лемме, которая говорит о том НОД элементов a и b , где $a > b$ равен НОД из остатка от деления a на b и b

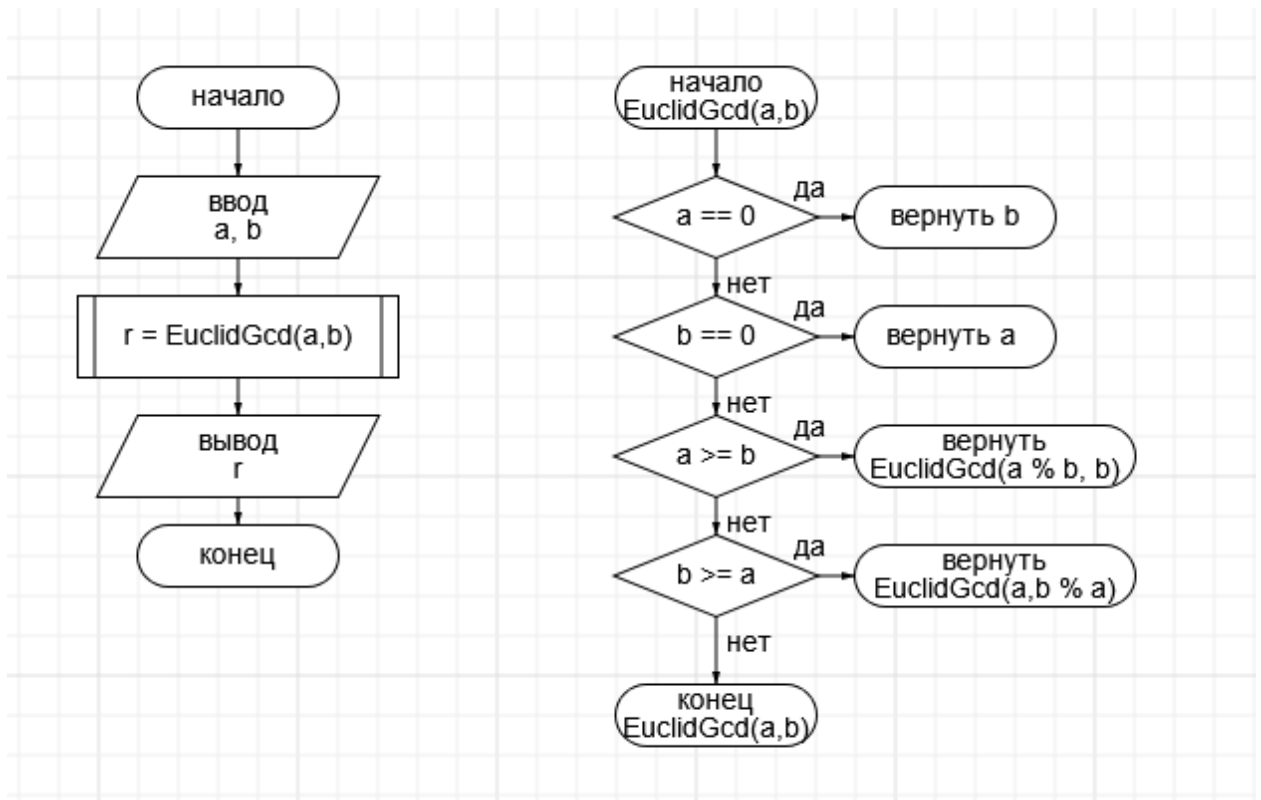


Рисунок 10. Блок схема алгоритма

```

1 3 usages
2 def EuclidGcd(a, b):
3     if a == 0:
4         return b
5     if b == 0:
6         return a
7     if a >= b:
8         return EuclidGcd(a % b, b)
9     if b >= a:
10        return EuclidGcd(a, b % a)
11
12 if __name__ == '__main__':
13     import time
14     b = 0
15     for i in range(208, 3918848, 200):
16         for j in range(100000):
17             start = time.perf_counter()
18             a = EuclidGcd(3918848, i)
19             end = time.perf_counter()
20             b += end - start
21             if j == 100000 - 1:
22                 print(f'{i}; {(b / (100000 - 1)):.8f}')
23                 b = 0
24
25 if __name__ == '__main__':
26     for i in range(208, 3918848, 200):
27         for j in range(100000):
28             start = time.perf_counter()
29             a = EuclidGcd(3918848, i)
30             end = time.perf_counter()
31             b += end - start
32             if j == 100000 - 1:
33                 print(f'{i}; {(b / (100000 - 1)):.8f}')
34                 b = 0

```

Run main

C:\Users\leo\PycharmProjects\pythonProject2\venv\bin\python.exe C:\Users\leo\PycharmProjects\pythonProject2\main.py

208; 0.00000144
408; 0.00000074
608; 0.00000088
808; 0.00000113
1008; 0.00000129
1208; 0.00000150
1408; 0.00000110
1608; 0.00000159
1808; 0.00000117
2008; 0.00000211
2208; 0.00000146

Рисунок 11. Результат выполнения exnod.py

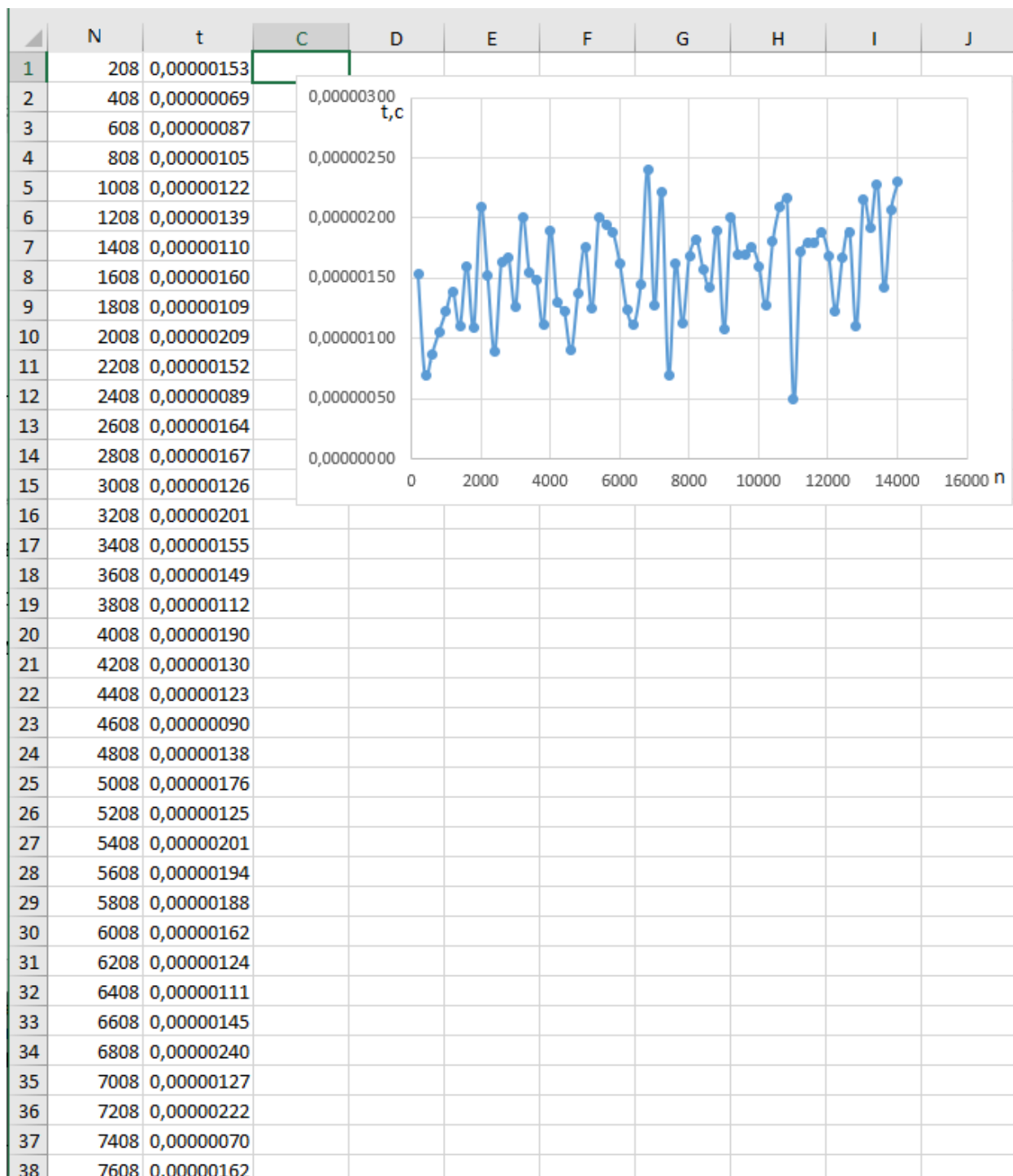


Рисунок 12. График зависимости второго элемента от времени
выполнения алгоритма

Выводы: в ходе выполнения работы было выявлено, что время выполнения программы зависит от алгоритма, сравнили нативные алгоритмы с их улучшенными версиями