Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №2 по курсу "Анализ алгоритмов"

Тема Умножение матриц

Студент Золотухин А.В.

Группа ИУ7-54Б

Оценка (баллы)

Преподаватель Волкова Л.Л., Строганов Ю.В.

Москва — 2022 г.

Оглавление

Введение

Матрицы используются в вычислениях почти везде. Особенно умножение. Это довольно трудоемкий процесс даже при небольших размерах матриц, так как требуется большое количество операций умножения и сложения различных чисел. По этой причине человек озадачен проблемой оптимизации умножения матриц и ускорения процесса вычисления. Пусть сегодня и существуют системы, ускоряющие подобные операции аппаратно, это не меняет того факта, что эти операции стоит оптимизировать.

Таким образом, эффективное умножение матриц по времени и затратам ресурсов является актуальной проблемой для науки и техники.

Цель лабораторной работы — изучение трех алгоритмов умножения матриц: классического, алгоритма Винограда и его оптимизации. Для того чтобы добиться этой цели, были поставлены следующие задачи:

- изучить и реализовать классический алгоритм умножения матриц и алгоритм Винограда;
- оптимизировать работу алгоритма Винограда;
- выполнить сравнительный анализ трудоёмкостей алгоритмов;
- сравнить эффективность алгоритмов по времени и памяти.

1 Аналитическая часть

1.1 Классический алгоритм умножения

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица является таблицей, на пересечении строк и столбцов находятся элементы матрицы. Количество строк и столбцов является размерностью матрицы.

Пусть даны две прямоугольные матрицы A и B размерности $m \times n$, $n \times q$ соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nq} \end{bmatrix}$$

Тогда произведением матриц A и B называется матрица C размерностью $m \times q$

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mq} \end{bmatrix},$$

в которой:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \left(\overline{i = 1 \dots m};\ \overline{j = 1 \dots q}\right) \tag{1.1}$$

.

1.2 Алгоритм Винограда

Исходя из равенства 1.1, видно, что каждый элемент матрицы представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее [1].

Рассмотрим два вектора U и V таких, что:

$$U = A_i = (u_1, u_2, \ldots, u_n), \tag{1.2}$$

где $U = A_i$ – i-ая строка матрицы A,
$u_k = a_{ik}, \overline{k = 1 \ldots n}$ – элемент i-ой строки k-ого столбца матрицы A.

$$V = B_j = (v_1, v_2, \ldots, v_n), \tag{1.3}$$

где $V = B_j$ – j-ый столбец матрицы B,
$v_k = b_{kj}, \overline{k = 1 \ldots n}$ – элемент k-ой строки j-ого столбца матрицы B.

По определению их скалярное произведение равно:

$$U \cdot V = u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4. \tag{1.4}$$

Равенство 1.4 можно переписать в виде:

$$U \cdot V = (u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - u_1 u_2 - u_3 u_4 - v_1 v_2 - v_3 v_4. \tag{1.5}$$

В равенстве 1.4 насчитывается 4 операции умножения и 3 операции сложения, в равенстве 1.5 насчитывается 6 операций умножения и 9 операций сложения. Однако выражение $-u_1 u_2 - u_3 u_4$ используются повторно при умножении i-ой строки матрицы A на каждый из столбцов матрицы B, а выражение $-v_1 v_2 - v_3 v_4$ - при умножении j-ого столбца матрицы B на строки матрицы A. Таким образом, данные выражения можно вычислить предварительно для каждых строк и столбцов матриц для сокращения повторных вычислений. В результате повторно будут выполняться лишь 2 операции умножения и 7 операций сложения (2 операции нужны для до-

бавления предварительно посчитанных произведений).

1.3 Оптимизированный алгоритм Винограда

Для оптимизации алгоритма Винограда могут использоваться такие стратегии, как:

- предварительные вычисления повторяющихся одинаковых действий;
- использование более быстрых операций при вычислении (такие, как смещение битов вместо умножения или деления на 2);
- использование аналогичных конструкций, уменьшающих трудоёмкость операций (к примеру, замена сложения с 1 на инкремент).

Вывод

В данном разделе были описаны классический алгоритм умножения матриц, а также алгоритм Винограда и способы его оптимизации.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 представлены схемы алгоритмов трёх алгоритмов умножения матриц.

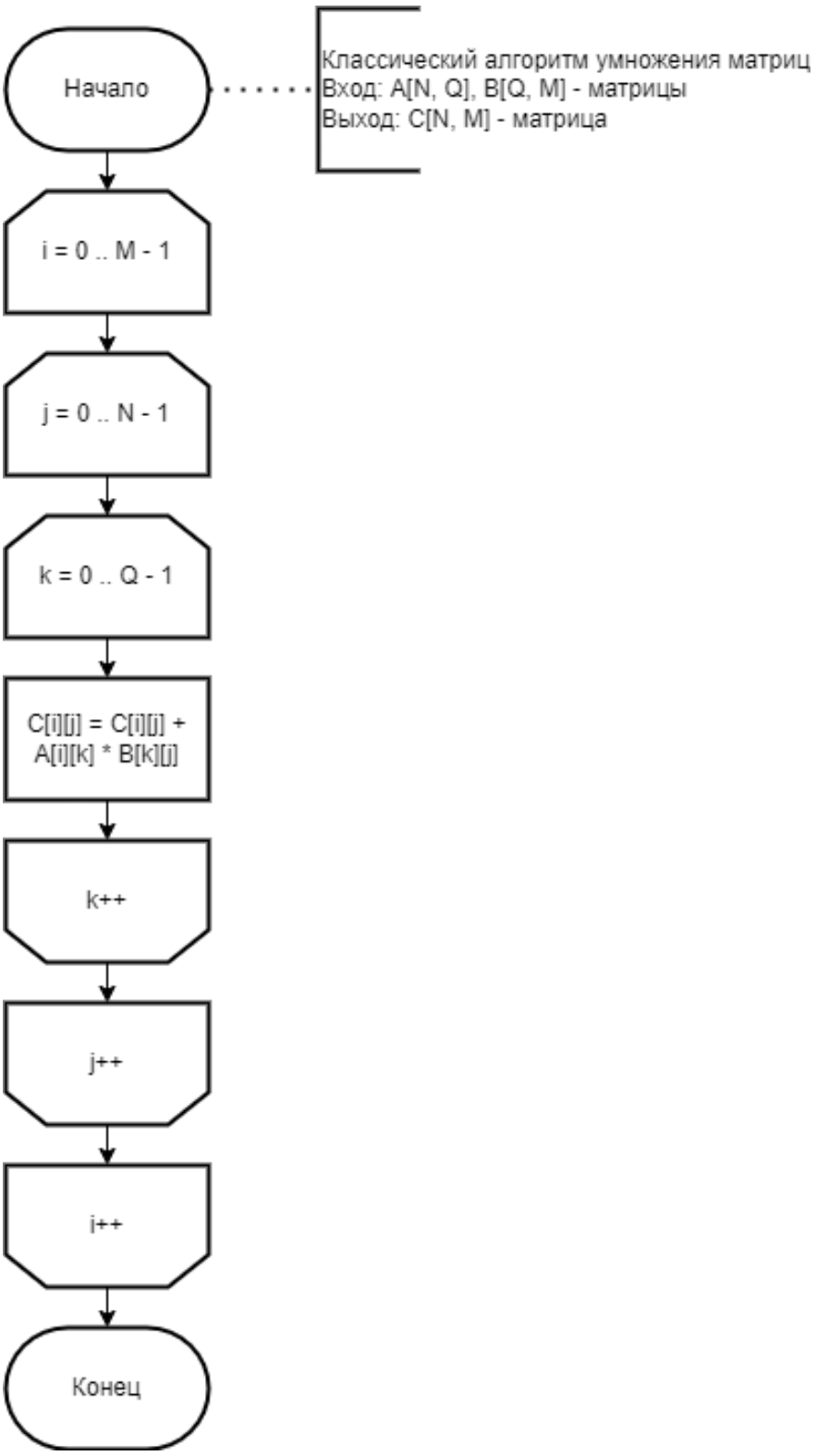

Рисунок 2.1 – Классический алгоритм

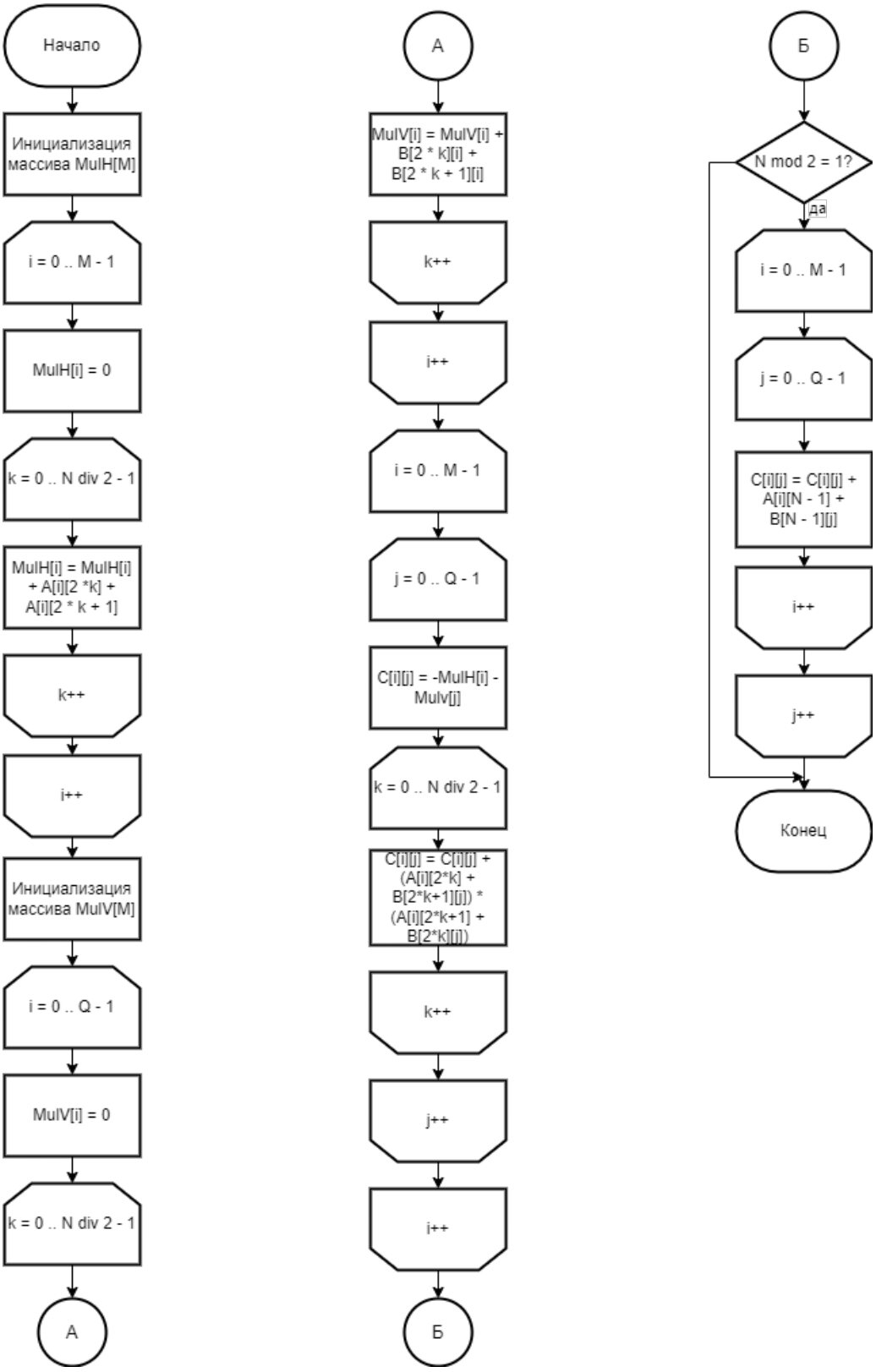

Рисунок 2.2 – Алгоритм Винограда

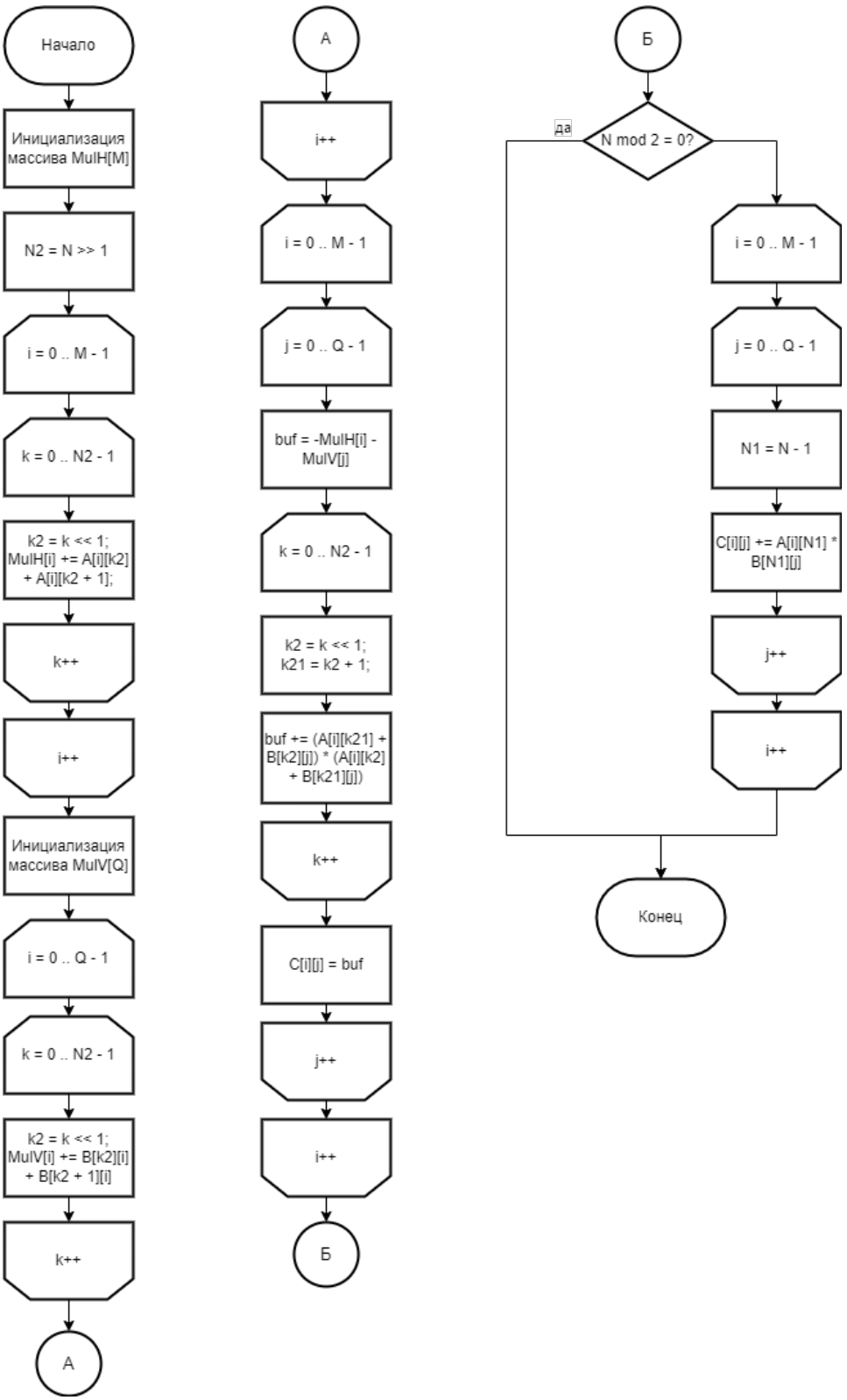

Рисунок 2.3 – Оптимизированный алгоритм Винограда

2.2 Оценка трудоёмкости

Пусть даны две матрицы A и B размерностью $M \times N$ и размерностью $N \times Q$ соответственно. Рассмотрим трудоемкость трёх алгоритмов умножения матриц.

2.2.1 Классический алгоритм

Данный алгоритм был рассмотрен на семинарских занятиях, его трудоемкость равна:

$$f_{classic} = 13MNQ + 4MQ + 4M + 1 \tag{2.1}$$

2.2.2 Алгоритм Винограда

Трудоёмкости составных частей алгоритма:

- цикл создания вектора MulH $f_I = \frac{15}{2}MN + 4M + 2$;
- цикл создания вектора MulV $f_{II} = \frac{15}{2}NQ + 4Q + 2$;
- основной цикл $f_{III} = 13MNQ + 4MQ + 4M + 2$;
- условный переход при чётном N $f_{IV} = 2$;
- условный переход и цикл при нечётном N $f_{IV} = 17MQ + 4M + 4$.

Общая трудоемкость алгоритма.
При чётном N.

$$f_{vin} = f_I + f_{II} + f_{III} + f_{IV} = 13MNQ + \frac{15}{2}MN + \frac{15}{2}NQ + 4MQ + 8M + 4Q + 8 \tag{2.2}$$

При нечётном N.

$$f_{vin} = f_I + f_{II} + f_{III} + f_{IV} = 13MNQ + \frac{15}{2}MN + \frac{15}{2}NQ + 21MQ + 12M + 4Q + 10 \tag{2.3}$$

2.2.3 Оптимизированный алгоритм Винограда

Трудоёмкости составных частей алгоритма:

- цикл создания вектора MulH $f_I = \frac{13}{2}MN + 2M + 3$;
- цикл создания вектора MulV $f_{II} = \frac{13}{2}NQ + 2Q + 3$;
- основной цикл $f_{III} = \frac{19}{2}MNQ + 11MQ + 3M + 1$;
- дополнительный цикл при нечётном N $f_{IV} = 11MQ + 2M + 3$;
- дополнительный цикл при чётном N $f_{IV} = 2$.

Общая трудоемкость алгоритма.
При чётном N.

$$f_{opt} = f_I + f_{II} + f_{III} + f_{IV} = \frac{19}{2}MNQ + \frac{13}{2}MN + \frac{13}{2}NQ + 11MQ + 5M + 2Q + 9 \quad (2.4)$$

При нечётном N.

$$f_{opt} = f_I + f_{II} + f_{III} + f_{IV} = \frac{19}{2}MNQ + \frac{13}{2}MN + \frac{13}{2}NQ + 22MQ + 7M + 2Q + 10 \quad (2.5)$$

2.3 Список оптимизаций алгоритма Винограда

Алгоритм Винограда был оптимизирован с помощью следующих модификаций.

1. Замена конструкций вида a = a + b (трудоёмкость = 2) на a += b (трудоёмкость = 1).
2. Предвычисление некоторых слагаемых.
3. Замена умножения на 2 (трудоёмкость 2) на побитовый сдвиг (трудоёмкость 1).

Вывод

Были разработаны схемы всех трех алгоритмов умножения матриц. Для каждого из них были рассчитаны трудоемкости.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к программному обеспечению

На вход реализация алгоритма должна принимать две матрицы.

На выход реализация алгоритма должна выдавать матрицу, которая является результатом умножения двух входных матриц.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования C# [2].

Язык C# является полностью объектно-ориентированным. Все необходимые библиотеки для реализации поставленной задачи являются стандартными.

Время работы алгоритмов было замерено с помощью функции clock().

3.3 Сведения о модулях программы

Программа состоит из пяти модулей.

1. Programm.cs - главный файл программы, в котором располагается код меню;
2. BaseAlgo.cs - файл с базовым классом алгоритмов.
3. Classic.cs, Vinograd.cs, OptimizedVinograd.cs - файлы с кодами алгоритмов

3.4 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3 представлены реализации алгоритмов умножения матриц (классический, Винограда и оптимизированный по Винограду).

Листинг 3.1 – Класс с реализацией классического алгоритма

```
internal class Classic : BaseAlgo
{
    public override int[][] Multiply(int[][] A, int[][] B)
    {
        int Ar = A.Length;
        int Br = B.Length;

        if (Ar == 0 || Br == 0)
        return null;

        int Ac = A[0].Length;
        int Bc = B[0].Length;

        if (Ac != Bc)
        return null;

        int[][] C = new int[Ar][];
        for (int i = 0; i < Ar; i++)
        C[i] = new int[Bc];

        for (int i = 0; i < Ar; i++)
        for (int j = 0; j < Bc; j++)
        for (int k = 0; k < Ac; k++)
        C[i][j] += A[i][k] * B[k][j];

        return C;
    }
}
```

Листинг 3.2 – Класс с реализацией алгоритма Винограда

```
internal class Vinograd : BaseAlgo
{
    public override int[][] Multiply(int[][] A, int[][] B)
```

```
4       {
5           int Ar = A.Length;
6           int Br = B.Length;
7
8           if (Ar == 0 || Br == 0)
9           return null;
10
11          int Ac = A[0].Length;
12          int Bc = B[0].Length;
13
14          if (Ac != Br)
15          return null;
16
17          int[] mulH = new int[Ar];
18          int[] mulV = new int[Bc];
19
20          int[][] C = new int[Ar][];
21          for (int i = 0; i < Ar; i++)
22          C[i] = new int[Bc];
23
24          for (int i = 0; i < Ar; i++)
25          for (int j = 0; j < Ac / 2; j++)
26          mulH[i] = mulH[i] + A[i][j * 2] * A[i][j * 2 + 1];
27
28          for (int i = 0; i < Bc; i++)
29          for (int j = 0; j < Br / 2; j++)
30          mulV[i] = mulV[i] + B[j * 2][i] * B[j * 2 + 1][i];
31
32          for (int i = 0; i < Ar; i++)
33          for (int j = 0; j < Bc; j++)
34          {
35              C[i][j] = -mulH[i] - mulV[j];
36              for (int k = 0; k < Ac / 2; k++)
37              C[i][j] = C[i][j] + (A[i][2 * k + 1] + B[2 * k][j]) *
                    (A[i][2 * k] + B[2 * k + 1][j]);
38          }
39
40          if (Ac % 2 == 1)
41          for (int i = 0; i < Ar; i++)
42          for (int j = 0; j < Bc; j++)
43          C[i][j] = C[i][j] + A[i][Ac - 1] * B[Ac - 1][j];
```

```
44
45          return C;
46      }
47  }
```

Листинг 3.3 – Класс с реализацией оптимизированного алгоритма Винограда

```
1  internal class OptimizedVinograd : BaseAlgo
2  {
3      public override int[][] Multiply(int[][] A, int[][] B)
4      {
5          int Ar = A.Length;
6          int Br = B.Length;
7
8          if (Ar == 0 || Br == 0)
9          return null;
10
11         int Ac = A[0].Length;
12         int Bc = B[0].Length;
13
14         if (Ac != Br)
15         return null;
16
17         int[] mulH = new int[Ar];
18         int[] mulV = new int[Bc];
19
20         int[][] C = new int[Ar][];
21         for (int i = 0; i < Ar; i++)
22         C[i] = new int[Bc];
23
24         int Ac2 = Ac >> 1;
25         for (int i = 0; i < Ar; i++)
26         for (int j = 0; j < Ac2; j++)
27         {
28             int j2 = j << 1;
29             mulH[i] += A[i][j2] * A[i][j2 + 1];
30         }
31
32         for (int i = 0; i < Bc; i++)
33         for (int j = 0; j < Ac2; j++)
34         {
```

```
35              int j2 = j << 1;
36              mulV[i] += B[j2][i] * B[j2 + 1][i];
37          }
38
39          for (int i = 0; i < Ar; i++)
40          for (int j = 0; j < Bc; j++)
41          {
42              int buf = -mulH[i] - mulV[j];
43              for (int k = 0; k < Ac2; k++)
44              {
45                  int k2 = k << 1;
46                  int k21 = k2 + 1;
47                  buf += (A[i][k21] + B[k2][j]) * (A[i][k2] +
                        B[k21][j]);
48              }
49              C[i][j] = buf;
50          }
51
52
53          if (Ac % 2 == 1)
54          for (int i = 0; i < Ar; i++)
55          for (int j = 0; j < Bc; j++)
56          {
57              int Ac1 = Ac - 1;
58              C[i][j] += A[i][Ac1] * B[Ac1][j];
59          }
60
61          return C;
62      }
63  }
```

3.5 Модульные тесты

В таблице 3.1 приведены тесты для методов, реализующих классический алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда. Тесты пройдены успешно.

Таблица 3.1 – Тестирование методов

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 1 & 1 \\ 5 & 5 & 5 \\ 2 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 15 \\ 6 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$
$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$
$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	Неверный размер

Вывод

В данном разделе были реализованы все три алгоритма умножения матриц, а именно: классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование.

1. Операционная система: Windows 10 Корпоративная, Версия 21H1, Сборка ОС 19043.2006.
2. Память: 8 ГБ.
3. Процессор: AMD Ryzen 5 4600H с видеокартой Radeon Graphics 3.00 ГГц [3].

Исследование проводилось на ноутбуке, включенном в сеть электропитания. Во время исследования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой.

4.2 Демонстрация работы программы

На рисунках 4.1, 4.2, 4.3 представлены результаты работы реализаций алгоритмов.

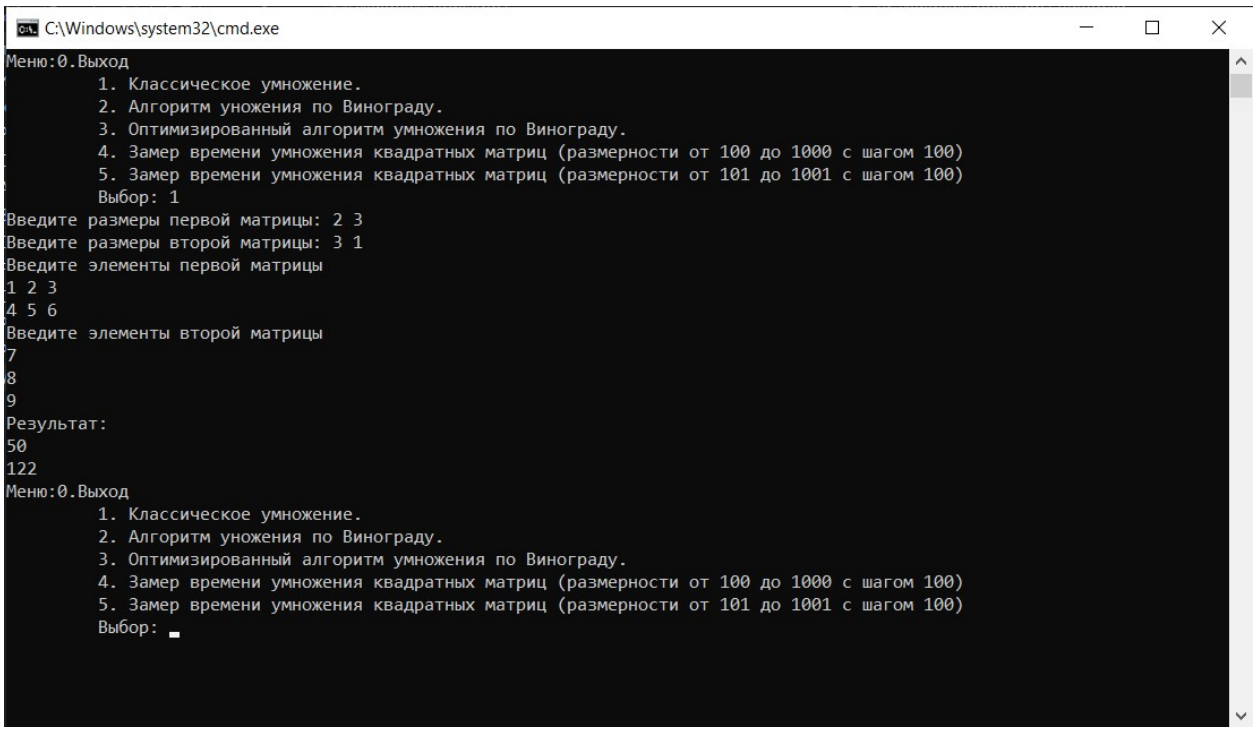

Рисунок 4.1 – Демонстрация корректной работы реализации классического алгоритма умножения матриц

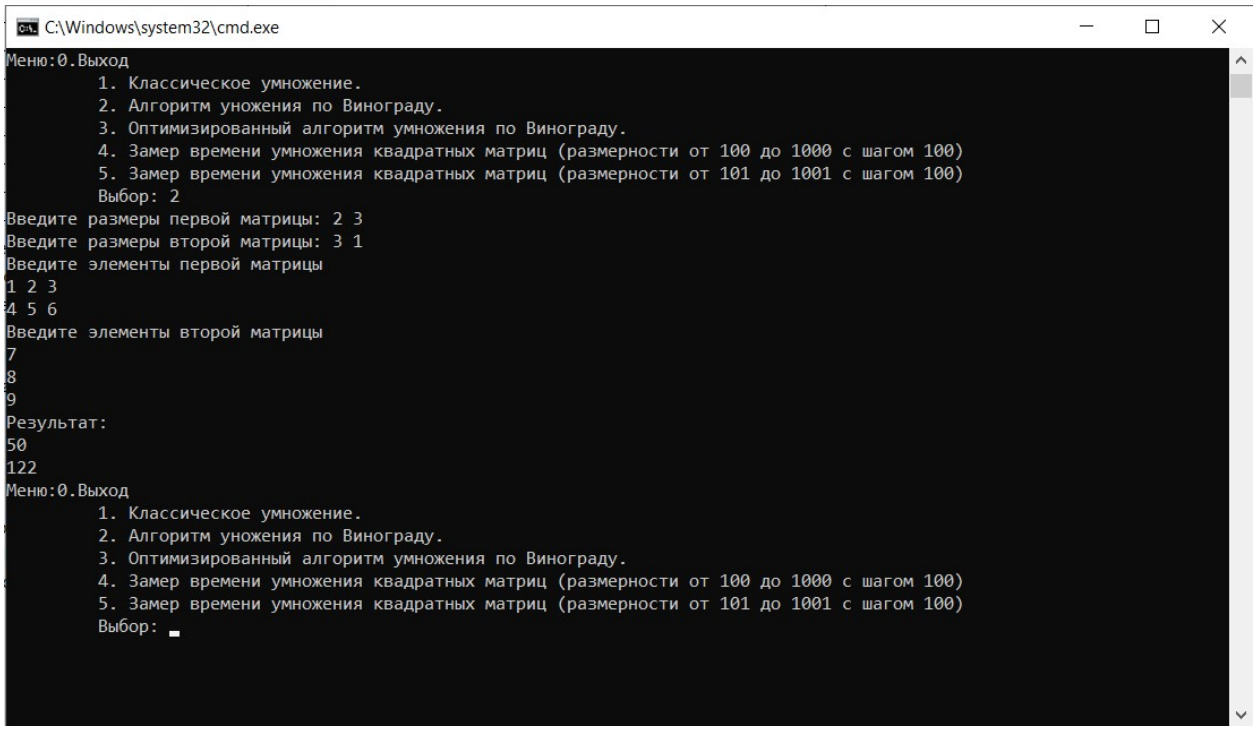

Рисунок 4.2 – Демонстрация корректной работы реализации алгоритма Винограда

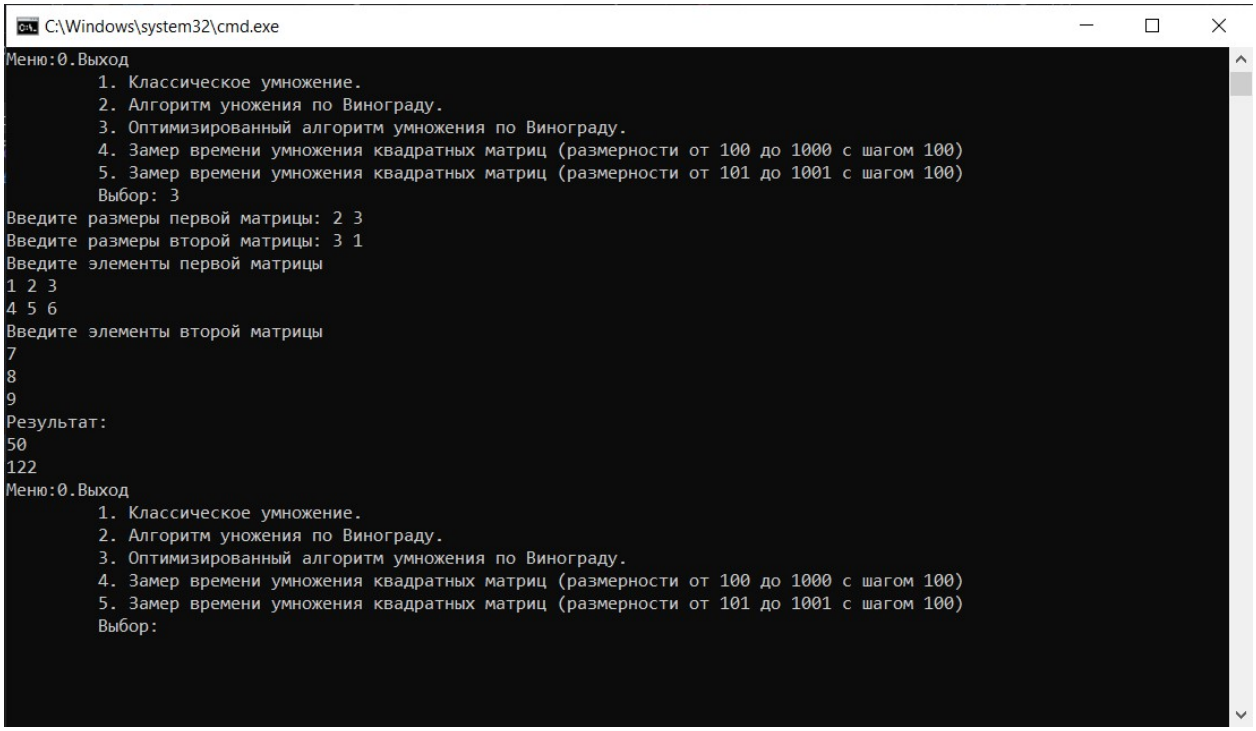

Рисунок 4.3 – Демонстрация корректной работы реализации оптимизированного алгоритма Винограда

4.3 Время выполнения реализаций алгоритмов

Время выполнения реализаций алгоритмов было замерено при помощи функции clock() [4]. Данная функция всегда возвращает значения времени, а именно сумму системного и пользовательского процессорного времени текущего процесса, значения типа float — время в тиках.

Замеры времени для каждой длины массива проводились 100 раз. В качестве результата взято среднее время работы алгоритма на данном размере матрицы.

4.4 Сравнение времени выполнения реализаций алгоритмов

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 100 до 1000 с шагом

100. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.1 и на рисунке 4.4.

Таблица 4.1 – Время работы алгоритмов сортировки на отсортированных данных (тики)

Размер	Classic	Vinograd	Optimized Vinograd
100	4,6	4,4	3,1
200	35,5	34,5	23,9
300	119,3	115,7	80
400	286,5	276,7	195,7
500	578,2	538,7	404,6
600	955,1	935,2	700,8
700	1588,1	1488,5	1115,8
800	2663,8	2254,7	2060,6
900	3584,3	3348,4	3033,1
1000	5246	5006,7	3747,1

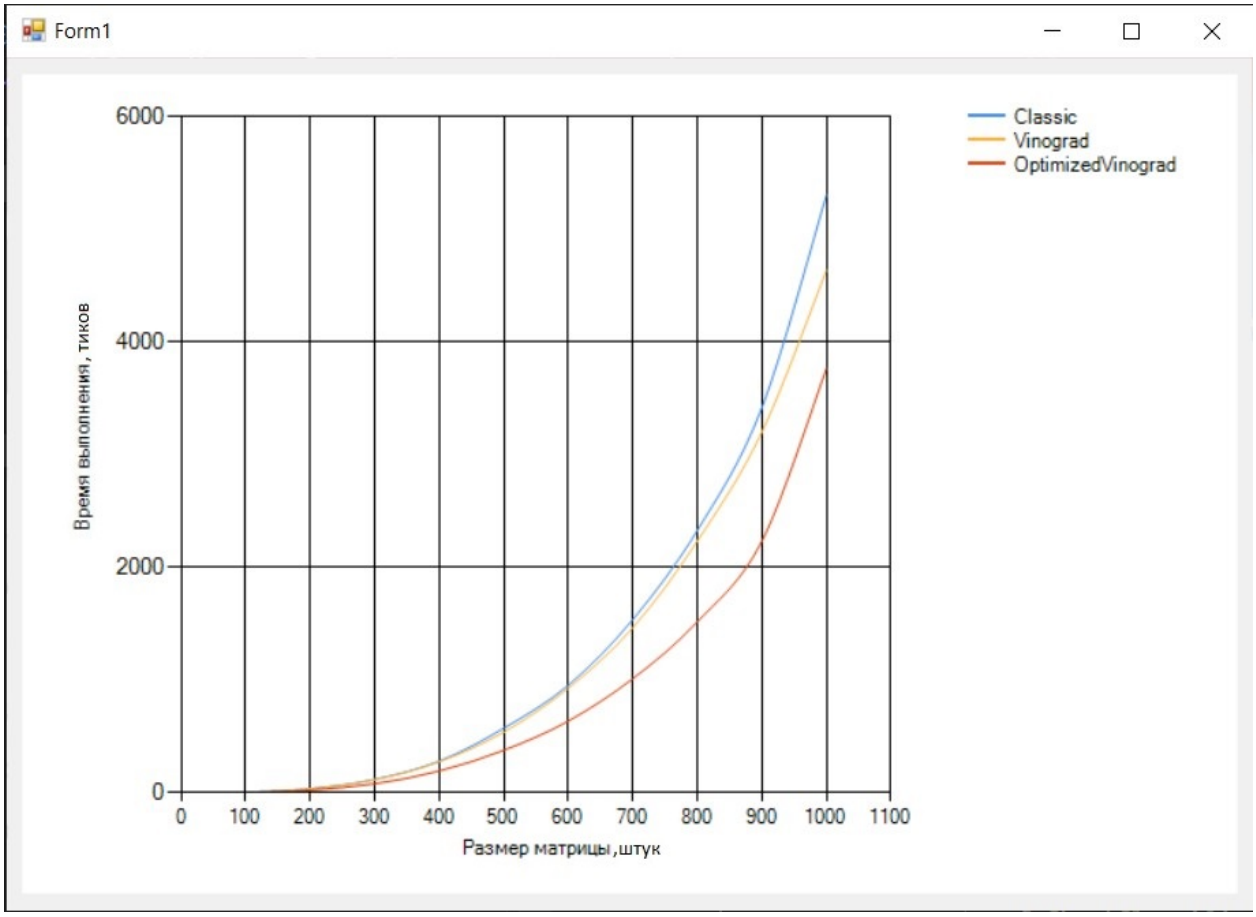

Рисунок 4.4 – Зависимость времени работы алгоритма от размера (четного) матрицы (время в тиках)

Из результатов экспериментов можно сделать вывод о том, что алгоритм Винограда эффективнее по времени чем классический алгоритм умножения матриц в среднем на 7%. Оптимизированный алгоритм работает быстрее обычного алгоритма Винограда.

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 101 до 1001 с шагом 100. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.2 и на рисунке 4.5.

Таблица 4.2 – Время работы реализаций алгоритмов умножения матриц при нечетных размерах (тики)

Размер	Classic	Vinograd	Optimized Vinograd
101	4,7	4,7	3,5
201	35,9	35,3	26,1
301	120,1	116,8	87,2
401	283,6	277,27	206,6
501	555,6	543	406,4
601	987,8	931,2	693,1
701	1559,6	1470,9	1096,2
801	2373,9	2194,1	1631,4
901	3496,1	3173,9	2354,7
1001	4966,4	4670,1	3452,7

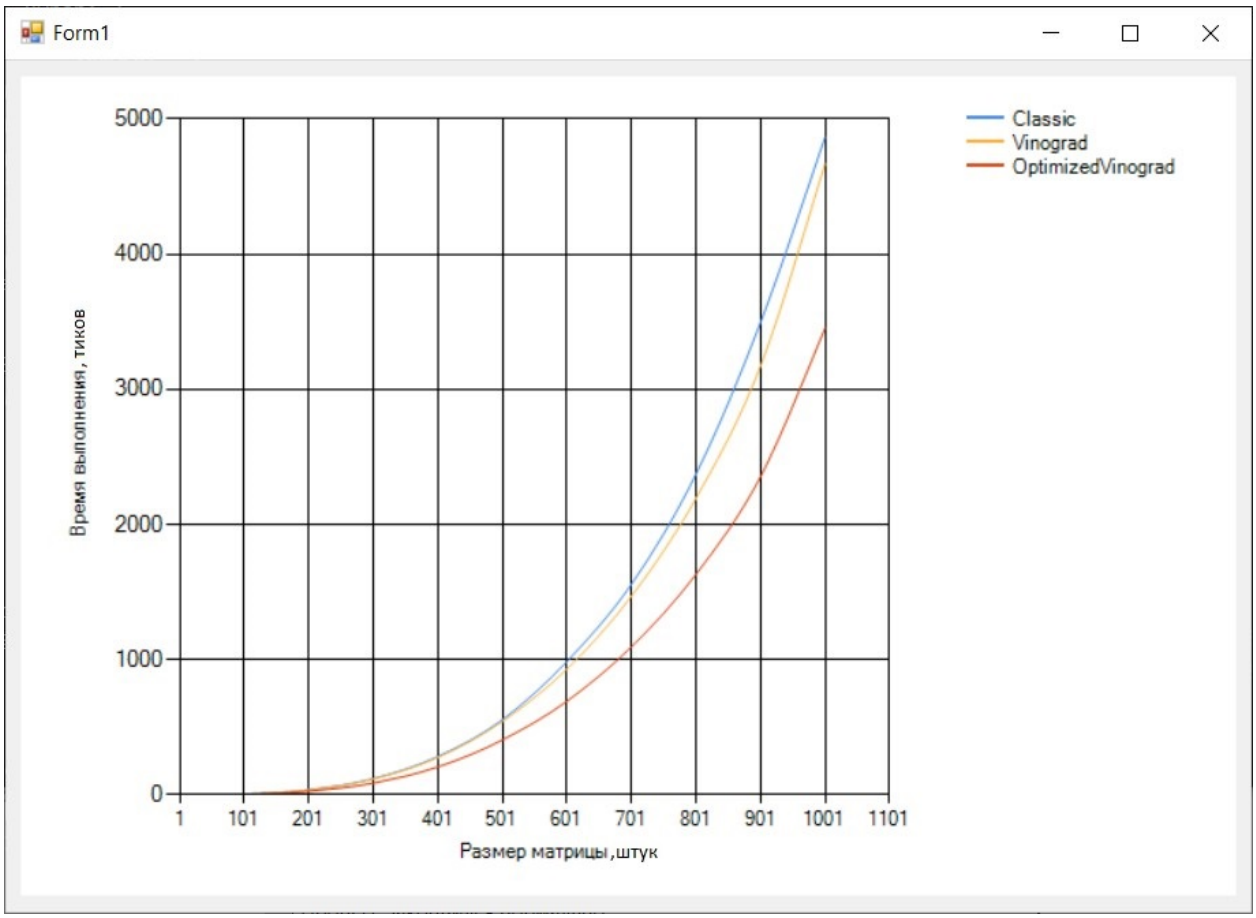

Рисунок 4.5 – Зависимость времени работы алгоритма от размера (нечетного) матрицы (время в тиках)

Для случая с нечётными размерами матриц можно сделать те же выводы, что и для случая с чётными. При этом можно заметить, что классический алгоритм в среднем работает за то же время, что и при чётных размерах, в то время как алгоритм Винограда и его оптимизация работают дольше за счёт дополнительных операций при нечётном случае.

Вывод

Алгоритм Винограда эффективнее по времени чем классический, но требует дополнительную память.

Заключение

В ходе выполнения работы достигнута поставленная цель: проведен сравнительный анализ алгоритмов умножения матриц и получены навыки оптимизации трудоёмкости алгоритмов. Решены все поставленные задачи.

- изучены и реализованы классический алгоритм умножения матриц и алгоритм Винограда;
- оптимизирована работа алгоритма Винограда;
- выполнен сравнительный анализ трудоёмкостей алгоритмов;
- сравнена эффективность алгоритмов по времени и памяти.

В ходе экспериментов по замеру времени работы было установлено, что оптимизированный алгоритм Винограда быстрее неоптимизированого. Оптимизированный алгоритм Винограда также всегда быстрее стандартного алгоритма Винограда.

Список использованных источников

1. Умножение матриц [Электронный ресурс]. Режим доступа: `http://algolib.narod.ru/Math/Matrix.html` (дата обращения: 09.10.2022).

2. Краткий обзор языка C# [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/` (дата обращения: 09.10.2022).

3. AMD Ryzen™ 5 4600H [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-5-4600h` (дата обращения: 09.10.2022).

4. ISO/IEC 9899:1999 [Электронный ресурс]. Режим доступа: `https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf` Глава 7.23.2.1 The clock function (дата обращения: 09.10.2022).