Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)
КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

Дисциплина: Анализ Алгоритмов

Лабораторная работа № 2
«Умножение матриц»

Выполнил: Тимонин А.С.
Группа: ИУ7-52Б

2019 г.

## Введение

Целью данной лабораторной работы является изучение различных алгоритмов перемножения матриц: классический алгоритм, алгоритм Винограда. А также предстоит провести сравнительный анализ.

Задачами данной лабораторной работы являются:

1. реализация и разработка алгоритмов;
2. проведение исследования на временные затраты алгоритмов;
3. сравнение и анализ полученных результатов.

# 1 Аналитический часть

В данном разделе приведены описания классического алгоритма и алгоритма Винограда.

## 1.1 Классический алгоритм

Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить.

Пусть матрица A имеет $n$ строк и $m$ столбцов, а матрица B имеет $m$ строк и $k$ столбцов.

Тогда матрица C будет иметь $m$ строк и $k$ столбцов.

$$A_{nm} * B_{mk} = Cink = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1(k-1)} & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2(k-1)} & c_{2k} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{n(k-1)} & c_{nk} \end{pmatrix}$$

## 1.2 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то можно заметить, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Также, такое умножение позволяет сделать предварительную обработку заранее.

Пусть два вектора **V** = ($v_1, v_2, v_3, v_4$) и **W** = ($w_1, w_2, w_3, w_4$).

Тогда Их скалярное произведение равно:

$$\mathbf{V} \bullet \mathbf{W} = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4.$$

Это равенство можно переписать в виде:

$$\mathbf{V} \bullet \mathbf{W} = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4.$$

Данный алгоритм позволяет разбить вычисления на несколько частей, одну из которых можно посчитать заранее. Также сокращается количество перемножений.

## 2 Конструкторская часть

В данном разделе приведены требования к программе, блок-схемы, сравнительный анализ по времени и по памяти.

### 2.1 Требования к программному продукту

Программный комплекс должен реализовывать 3 алгоритма перемножения матриц - классический, Винограда, оптимизированного Винограда. Пользователь должен иметь возможность создавать матрицы любого размера и перемножать матрицы одним из трех алгоритмов.

### 2.2 Средства реализации

Для выполнения поставленной задачи был использован язык программирования C++. Среда для разработки XCode. Для измерения процессорного времени была взята функция rdtsc из библиотеки ctime.

### 2.3 Схемы алгоритмов

Классическая реализация алгоритма умножения матриц представлена на рис. 1, на рис. 2 представлен алгоритм Винограда для перемножения матриц. На рис. 3 представлен алгоритм Винограда оптимизирован путем слияния 3 и 4 частей в алгоритме Винограда: перед тем как производить основные вычисления мы проверяем размер матрицы на четность и после этого вычисляем определенным образом. Также сокращено количество подсчетов длинны массива, операция умножения заменена на побитовый сдвиг вправо.

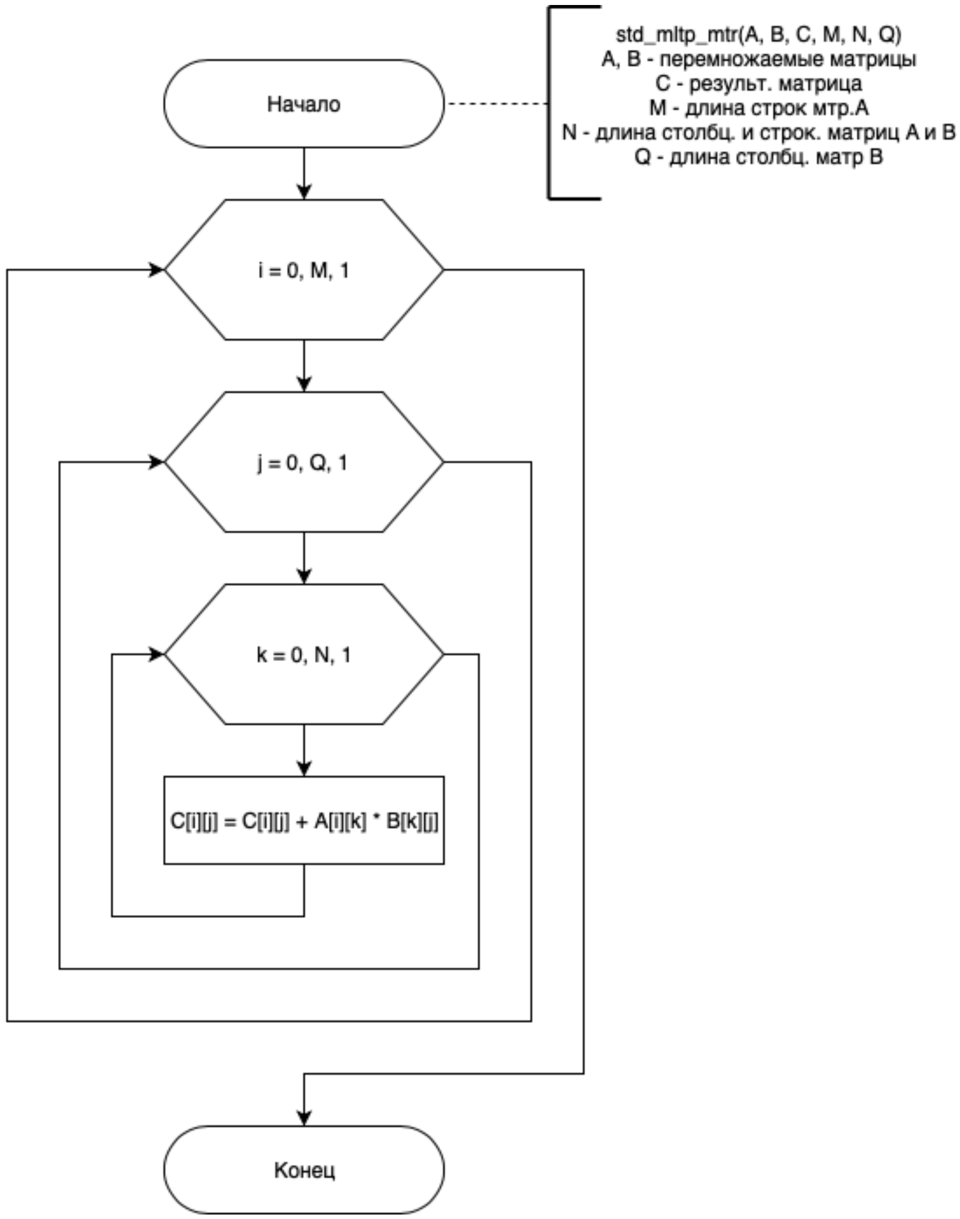


Рис 1. Классический алгоритм перемножений матриц

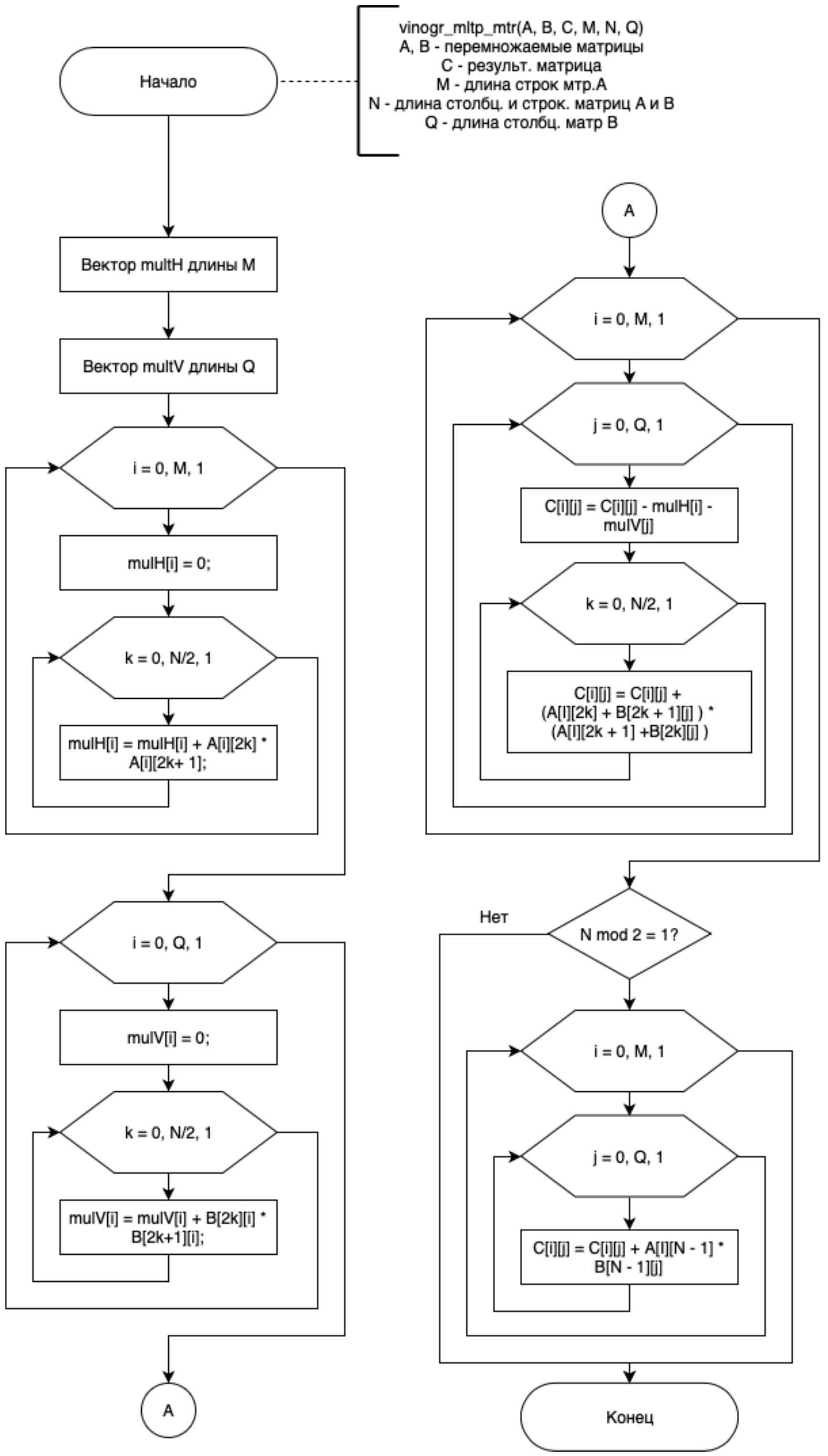


Рис 2. Алгоритм Винограда умножения матриц

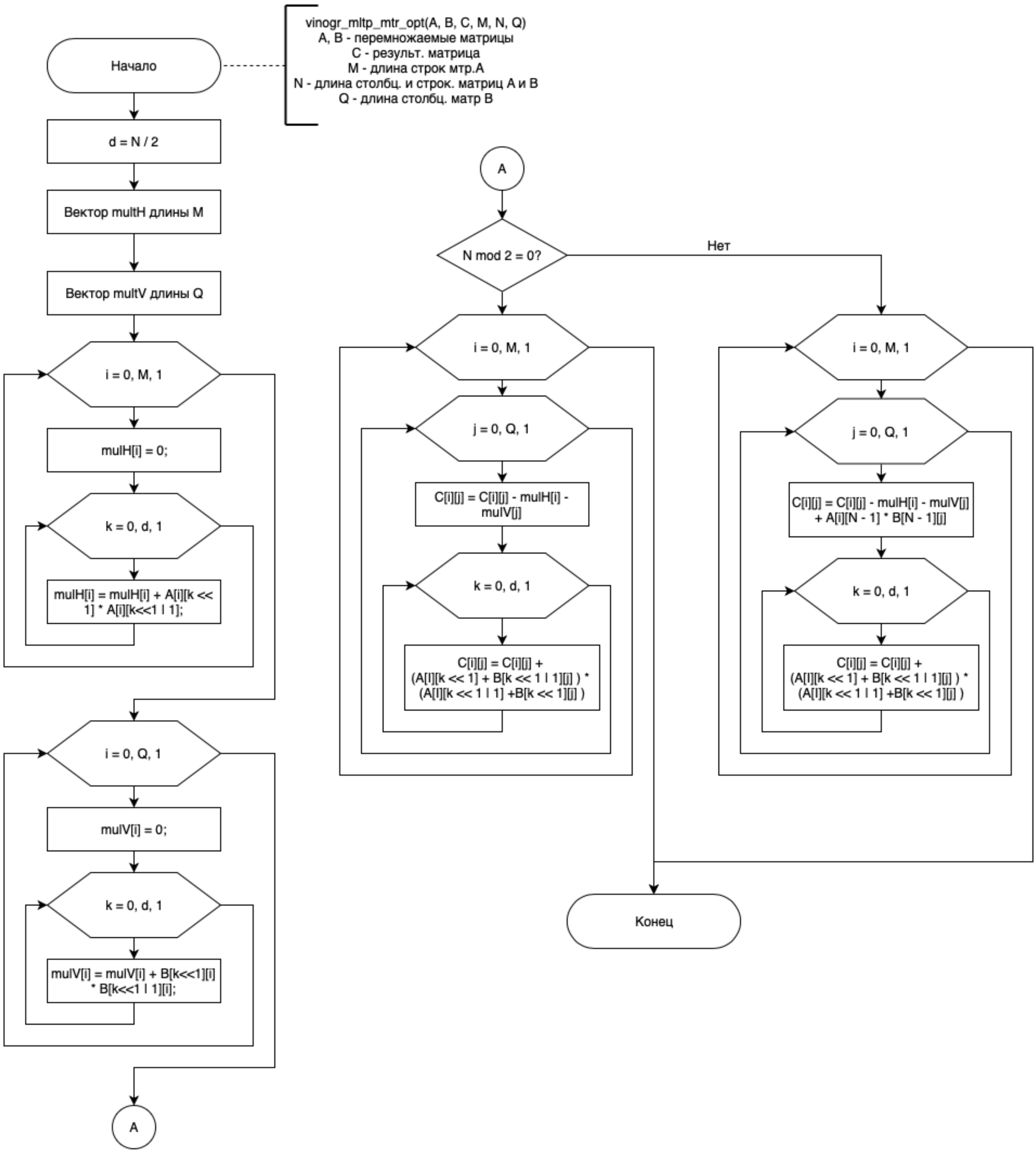


Рис 3. Оптимизированный алгоритм Винограда умножения матриц

## 2.4 Расчет сложности алгоритмов

**Модель вычислений**

Для корректного расчета сложности алгоритмов следует описать модель вычислений. Любая арифметическая операция имеет стоимость 1. Проверка условия имеет стоимость 1, а переход в блок иначе не имеет стоимости. В циклах происходит начальная инициализация и проверка, каждая операция имеет стоимость 1. В цикле происходит проверка условия и увеличение счетчика, каждая такая операция тоже имеет стоимость 1. Так, каждая итерация цикла имеет добавочную стоимость 2.

**Классический алгоритм:**

Вычисление матрицы - 2 + M(2 + 5 + Q(2 + 2 + N(2 + 11))) = 2 + 7M + 4MQ + 13MNQ

**Алгоритм Винограда:**

Вычисление multH - 2 + M(2 + 5 + N/2(3 + 12))

Вычисление multV - 2 + Q(2 + 5 + N/2(3 + 12))

Вычисление матрицы с четной разм. (лучший случ.) -

2 + M(2 +2+ Q(12 + N/2*(3+23)) = 2+M(2+Q(11+12.5N)) = 2+M(2+11Q+12.5NQ) = <u>4+4M + 12QM + 13MNQ</u>

Вычисление матрицы с нечетной разм. (худший случ.) -

6+4M + 12QM + 13MNQ + M(2+2+Q(2+13)) = <u>6+8M+27QM+13MNQ</u>

**Оптимизированный алгоритм Винограда:**

Вычисление multH - 2 + M(2 + 4 + N/2(12))

Вычисление multV - 2 + Q(2 + 4 + N/2(12))

Вычисление матрицы с четной разм. (лучший случ.) - 4+M(4+Q(4+7+N/2(3+23))) = 4+M(4+Q(11 + 13N)) = 4+M(4+11Q + 13NQ) = <u>4+4M+11MQ+13MNQ</u>

Вычисление матрицы с нечетной совпадающей размерностью исходных матриц (худший случай) - $+M(4+Q(4+14+N/2(2+23))) = 4+M(4+Q(18+12.5N)) = 4+M(4+18Q+12.5NQ) = \underline{4+4M+18MQ+12.5MNQ}$

# 3 Технологическая часть

В данном разделе приведено требования к программному обеспечению, а также листинг кода

## 3.1 Требования к ПО

Программа была написана на языке C++ в среде Xcode. Данный язык обусловлен наличием библиотеки, позволяющей считать тики процессора, что более точно определяет эффективность программы

## 3.2 Листинг кода

Реализация алгоритмов перемножения матриц представлена в листингах 1, 2, 3 соответственно.

Листинг 1. Классический алгоритм умножения матриц

```
void std_mult_matrix(int ** A, int ** B, int ** C,
                     unsigned M, unsigned N, unsigned Q) {

    for (unsigned i = 0; i < M; i++)
        for (unsigned j = 0; j < Q; j++) {

            C[i][j] = 0;

            for (unsigned k = 0; k < N; k++) {
                C[i][j] += + A[i][k] * B[k][j];
            }
        }
}
```

Листинг 2. Алгоритм Винограда для умножения матриц

```
void vingr_mult_matrix(int ** A, int ** B, int ** C,
                       unsigned M, unsigned N, unsigned Q) {
    int *mulH = new int [M];
    int *mulV = new int [Q];

    for (unsigned i = 0; i < M; i++) {
        mulH[i] = 0;
        for (unsigned k = 0; k < N / 2; k++)
```

```
            mulH[i] += A[i][2 * k] * A[i][2 * k + 1];
    }

    for (unsigned i = 0; i < Q; i++) {
        mulV[i] = 0;
        for (unsigned k = 0; k < N / 2; k++)
            mulV[i] += B[2 * k][i] * B[2 * k + 1][i];
    }

    for (unsigned i = 0; i < M; i++)
        for (unsigned j = 0; j < Q; j++) {
            C[i][j] = -mulH[i] - mulV[j];

            for (unsigned k = 0; k < N / 2; k++) {
                C[i][j] = C[i][j] +
                           (A[i][2 * k] + B[2 * k + 1][j]) *
                           (A[i][2 * k + 1] + B[2 * k][j]);
            }
        }

    if (N % 2 == 1) {
        for (unsigned i = 0; i < M; i++)
            for (unsigned j = 0; j < Q; j++)
                C[i][j] = C[i][j] + A[i][N - 1] * B[N - 1][j];
    }

    delete [] mulH;
    delete [] mulV;
}
```

Листинг 3. Оптимизированный алгоритм Винограда для умножения матриц

```
void vingr_mult_mtr_opt(int ** A, int ** B, int ** C,
                        unsigned M, unsigned N, unsigned Q) {

    unsigned d = N / 2;
    int *mulH = new int [M];
    int *mulV = new int [Q];

    for (unsigned i = 0; i < M; i++) {
        mulH[i] = 0;
        for (unsigned k = 0; k < d; k++)
            mulH[i] += A[i][k << 1] * A[i][k << 1 | 1];
    }

    for (unsigned i = 0; i < Q; i++) {
        mulV[i] = 0;
        for (unsigned k = 0; k < d; k++)
            mulV[i] += B[k << 1][i] * B[k << 1 | 1][i];
```

```
    }

    if (N % 2 == 0) {

        for (unsigned i = 0; i < M; i++)
            for (unsigned j = 0; j < Q; j++) {

                C[i][j] = -mulH[i] - mulV[j];

                for (unsigned k = 0; k < N / 2; k++) {
                    C[i][j] = C[i][j] +
                                    (A[i][k << 1] + B[k << 1 | 1][j]) *
                                    (A[i][k << 1 | 1] + B[k << 1][j]);
                }
            }

    } else {

        for (unsigned i = 0; i < M; i++)
            for (unsigned j = 0; j < Q; j++) {
                C[i][j] = -mulH[i] - mulV[j] +
                               A[i][N - 1] * B[N - 1][j];

                for (unsigned k = 0; k < N / 2; k++) {
                    C[i][j] = C[i][j] +
                           (A[i][k << 1] + B[k << 1 | 1][j]) *
                           (A[i][k << 1 | 1] + B[k << 1][j]);
            }
        }

    }

    delete [] mulH;
    delete [] mulV;
}
```

# 4 Экспериментальная часть

В данном разделе проведено исследование временных затрат написанных алгоритмов с подробным сравнительным анализом.

## 4.1 Сравнительное исследование

Замеры времени выполнялись на квадратных матрицах размеров от 100x100 до 1000x1000 с интервалом в 100 элементов. Также замеры времени проводились над квадратными матрицами нечетной размерностью от 101x101 до 1001x1001 с шагом 100. В табл. 4.1.1-4.1.2 и рис. 4.1.1-4.1.2 представлены результаты замеров времени в тактах.

Таблица 4.1.1.

Сравнение времени работы алгоритмов в тактах процессора для четной размерности таблиц

| | Classic | Vinograd | Vinograd opt. |
|---|---|---|---|
| 100 | 5309906 | 3492994 | 3438812 |
| 200 | 41791620 | 31873814 | 29161740 |
| 300 | 178998050 | 125717644 | 133166774 |
| 400 | 547620612 | 393795716 | 380205336 |
| 500 | 1095450900 | 835462300 | 761248202 |
| 600 | 1698521190 | 1328236332 | 1282629266 |
| 700 | 2638849132 | 2247398672 | 2298834432 |
| 800 | 3422139986 | 3171317544 | 3261247570 |
| 900 | 3545636741 | 2998248558 | 3473267372 |
| 1000 | 7356693575 | 6934621043 | 6744072550 |

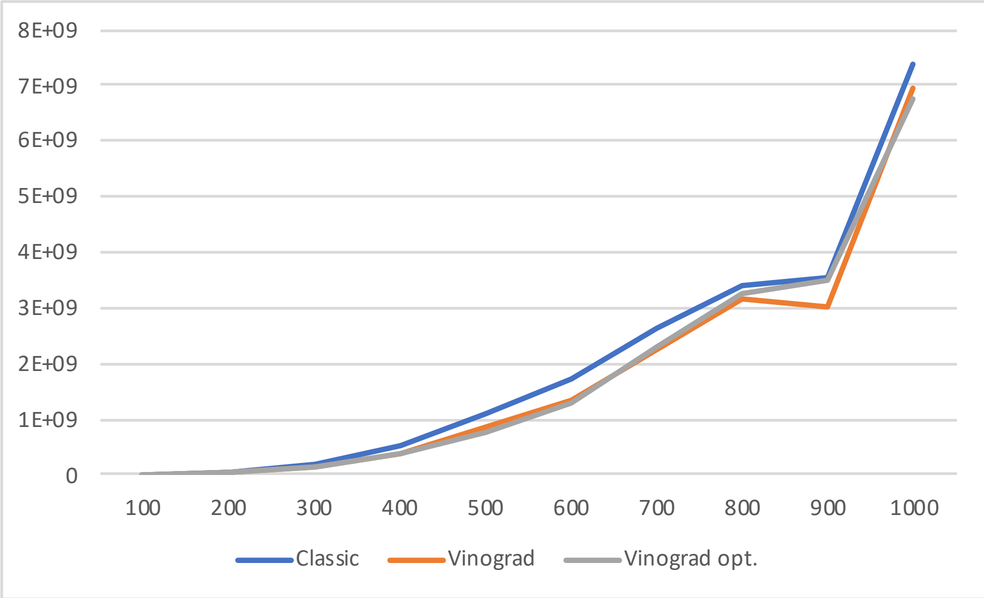


Рисунок 4.1.1. График зависимости времени работы алгоритмов для четной размерности матриц

| | Classic | Vinograd | Vinograd opt. |
|---|---|---|---|
| 101 | 4842611 | 3566992 | 3501080 |
| 201 | 40046584 | 31284620 | 28390012 |
| 301 | 170642172 | 132633753 | 123052866 |
| 401 | 510572952 | 386237826 | 380056734 |
| 501 | 1031029677 | 788027506 | 761248202 |
| 601 | 1408245229 | 1111112926 | 1154359310 |
| 701 | 2399545870 | 2122864749 | 2043793498 |
| 801 | 2795353636 | 2712904320 | 2658934893 |
| 901 | 3321456235 | 3268833774 | 3203435897 |
| 1001 | 5936626724 | 6004934586 | 6134903434 |

Таблица 4.1.2. Сравнение времени работы алгоритмов в тактах процессора для нечетной размерности таблиц

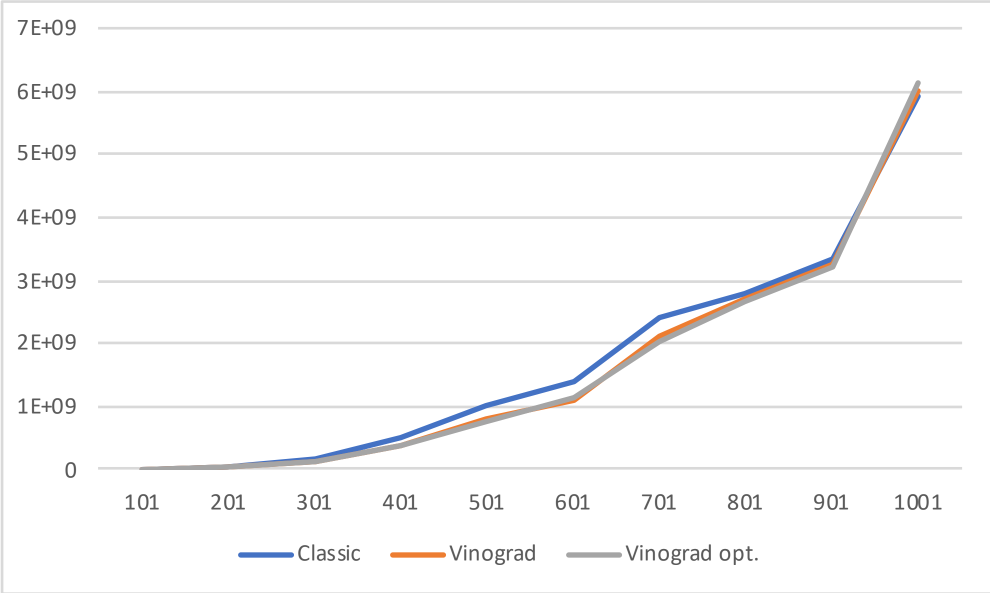


Рисунок 4.1.2. График зависимости времени работы алгоритмов для нечетной размерности матриц

## 4.2 Вывод

По данным эксперимента можно заметить, что алгоритм Винограда и оптимизированный алгоритм Винограда выдают практически одинаковые результаты, и несколько выигрывают у стандартного алгоритма.

## Заключение

В ходе выполнения данной лабораторной работы были изучены и реализованы различные алгоритмы перемножения матриц. В аналитической части было приведено описание алгоритмов. В конструкторской части были представлены блок-схемы алгоритмов. Также был выполнен расчет сложности алгоритмов. В экспериментальной части проведен сравнительный анализ временных затрат, после которого было выявлена незначительная разность обоих алгоритмов.