



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»  
на тему: «Умножение матриц (сложность)»

Студент ИУ7-56Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Вольняга М. Ю.  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Волкова Л. Л.  
(И. О. Фамилия)

2023 г.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитический раздел</b>	<b>4</b>
<b>2 Конструкторский раздел</b>	<b>5</b>
2.1 Требования к программному обеспечению . . . . .	5
2.2 Описание используемых типов данных . . . . .	5
2.3 Разработка алгоритмов . . . . .	5
2.4 Модель вычислений . . . . .	10
2.5 Трудоемкость алгоритмов . . . . .	10
<b>3 Технологический раздел</b>	<b>15</b>
3.1 Средства реализации . . . . .	15
3.2 Сведения о модулях программы . . . . .	15
3.3 Реализация алгоритмов . . . . .	15
<b>4 Исследовательская часть</b>	<b>20</b>
4.1 Технические характеристики . . . . .	20
4.2 Демонстрация работы программы . . . . .	20
4.3 Временные характеристики . . . . .	21
4.4 Характеристики по памяти . . . . .	23
4.5 Вывод . . . . .	25
Вывод . . . . .	25
<b>ЗАКЛЮЧЕНИЕ</b>	<b>27</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>29</b>

# ВВЕДЕНИЕ

Матрицей — называется прямоугольная таблица, представляющая собой массив элементов, для которой определены математические операции. Элементами матрицы могут служить числа, алгебраические символы или математические функции [1]. Умножение матриц широко применяется в различных задачах, и поэтому изучение алгоритмов для его выполнения является важным вопросом на сегодняшний день [2].

Цель данной лабораторной работы — исследование следующих алгоритмов умножения матриц:

- классического алгоритма;
- алгоритма Штрассена;
- алгоритма Винограда;
- оптимизированная версия алгоритма Винограда в соответствии с заданным вариантом, а именно:
  - заменить умножение на 2 на побитовый сдвиг;
  - заменить операцию  $x = x + k$  на  $x+ = k$ ;
  - вычислять заранее некоторые слагаемые для алгоритма.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- разработать требуемые алгоритмы;
- оценить трудоемкость рассматриваемых алгоритмов;
- провести сравнительный анализ времени выполнения реализуемых алгоритмов и занимаемой памяти.

# 1 Аналитический раздел

*Классический алгоритм* умножения матриц является базовым методом, в котором каждый элемент результирующей матрицы вычисляется с использованием тройного вложенного цикла. Данный алгоритм — это реализация математического определения умножения матриц. Его асимптотическая сложность равна  $O(n^3)$ , такая ассимптотика делает его менее эффективным для больших матриц [1].

*Алгоритм Винограда* вводит предварительную обработку матриц для оптимизации вычислений. Он использует дополнительные массивы для хранения промежуточных результатов и требует меньше умножений по сравнению с классическим методом. Это делает реализацию этого алгоритма более эффективной по времени выполнению. Его асимптотическая сложность  $O(n^{2.3755})$  [3].

*Алгоритм Штрассена* представляет собой рекурсивный метод, который основан на разделении матриц на подматрицы и использовании 7 умножений вместо 8, для матриц размерностью  $2 \times 2$ . Его асимптотическая сложность  $O(n \log_2 7)$ , делает его привлекательным для работы с большими матрицами [4].

## 2 Конструкторский раздел

В этом разделе представлены: требования к программному обеспечению и схемы реализуемых алгоритмов.

### 2.1 Требования к программному обеспечению

На вход принимаются две матрицы с размерами. На выходе программе требуется получить результирующую матрицу после умножения входных матриц.

### 2.2 Описание используемых типов данных

При реализации алгоритмов будет использован следующий тип данных, матрица — двумерный массив значений целочисленного типа.

### 2.3 Разработка алгоритмов

Алгоритмы принимают на вход матрицы *mtrx* и *matry*. На выходе у алгоритмов результат перемножения двух матриц, записанный в матрицу *matr\_res*. На рисунках 2.1 — 2.4 приведены схемы для реализуемых алгоритмов.

Классический алгоритм умножения матриц

Вход:

$mtrx[n][m]$  и  $mtry[m][k]$  - матрицы

$n$  - кол-во строк у  $mtrx$

$m$  - кол-во столбцов у  $mtrx$  и кол-во строк у  $mtry$

$k$  - кол-во столбцов у  $mtry$

Выход:  $mtr\_res$  размером  $n$  на  $k$

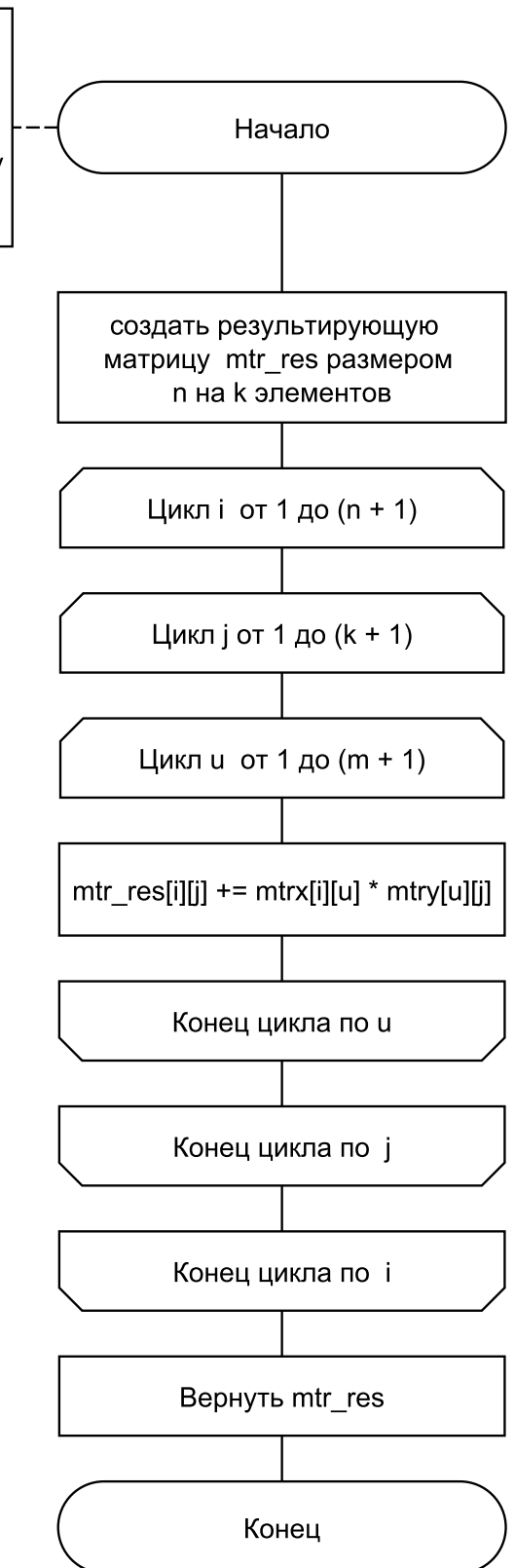


Рисунок 2.1 – Схема классического алгоритма умножения матриц

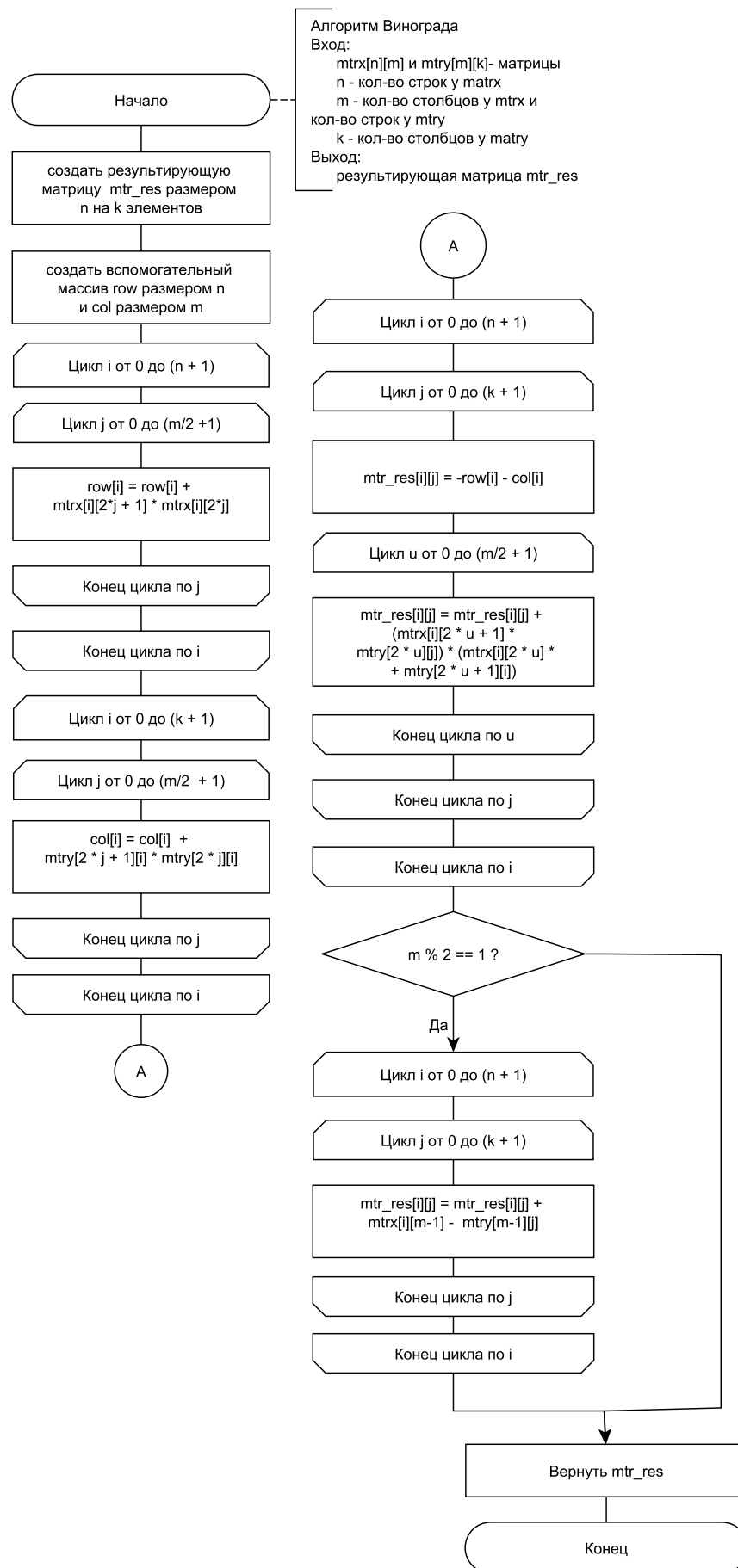


Рисунок 2.2 – Схема алгоритма Винограда

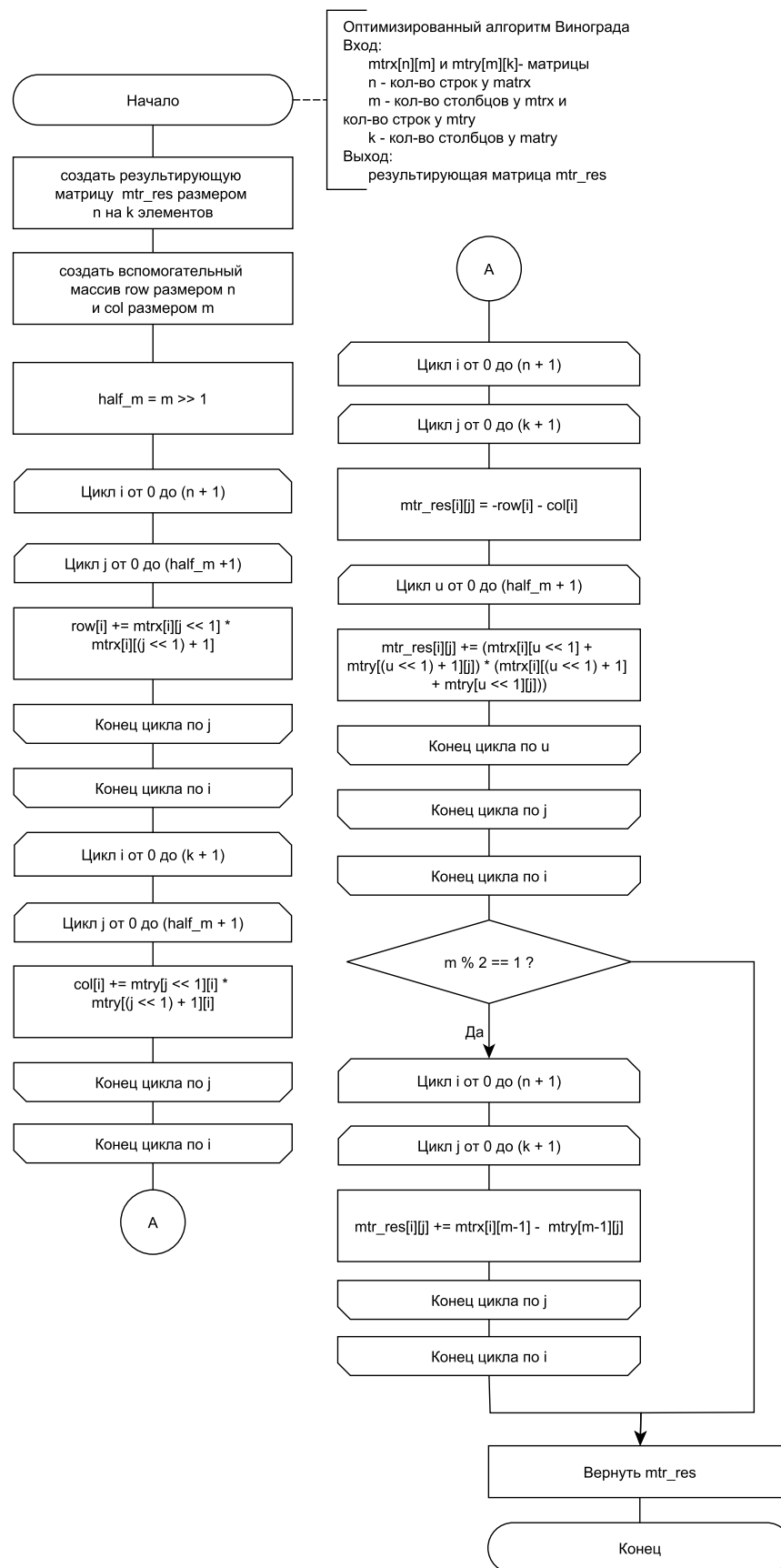


Рисунок 2.3 – Схема оптимизированного алгоритма Винограда



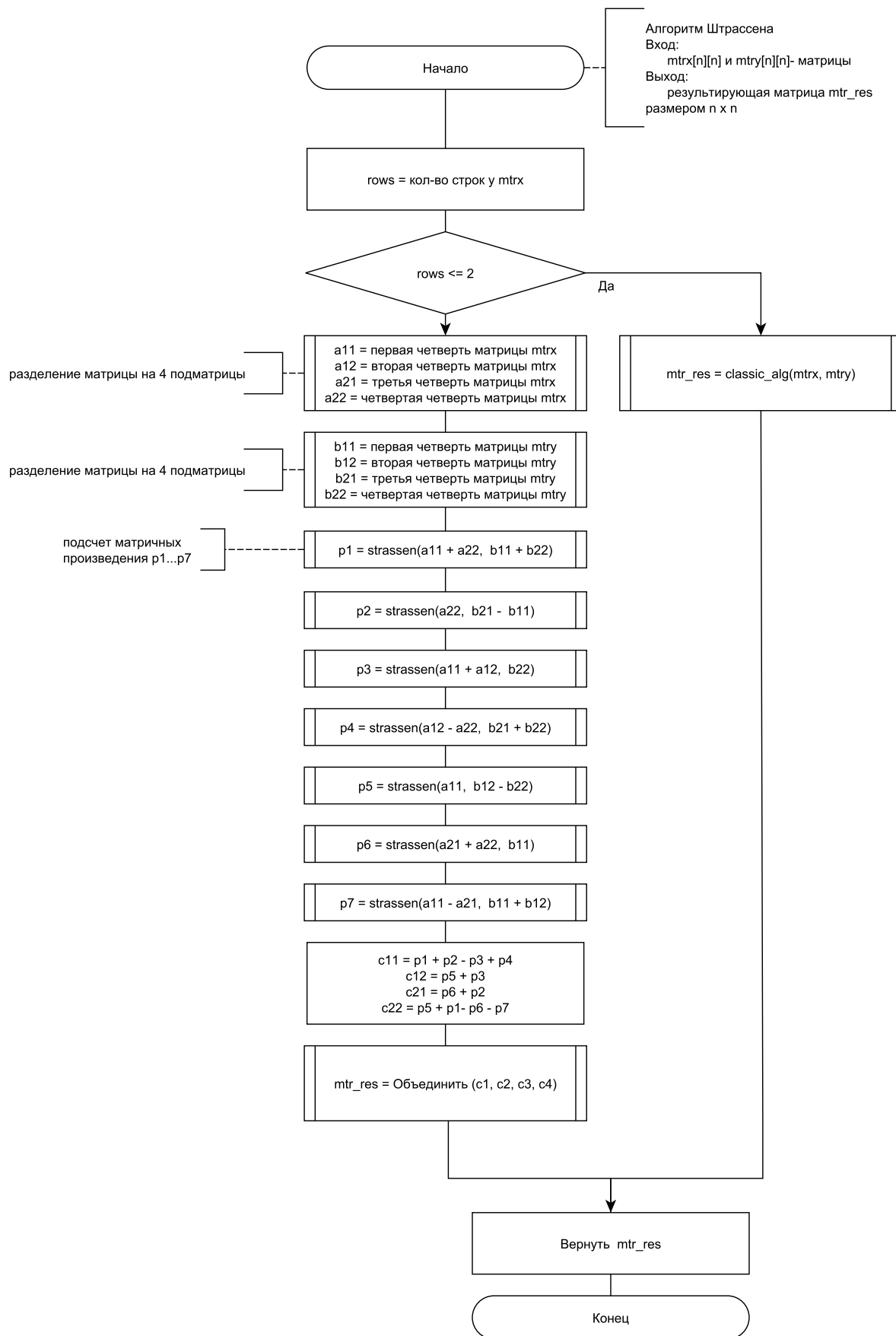


Рисунок 2.4 – Схема алгоритма Штрассена

## 2.4 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, *, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \quad (2.1)$$

2. трудоемкость оператора выбора *if условие then A else B* рассчитывается как (2.2);

$$f_{if} = f_{условия} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2.2)$$

3. трудоемкость цикла рассчитывается как (2.3);

$$f_{for} = f_{инициализации} + f_{сравнения} + N(f_{тела} + f_{инкремента} + f_{сравнения}) \quad (2.3)$$

4. трудоемкость вызова функции равна 0.

## 2.5 Трудоемкость алгоритмов

Трудоемкость инициализации результирующей матрицы учитываться не будет, так как данное действие есть во всех алгоритмах и не является трудоемким.

Обозначения для расчета трудоемкости :

- N – кол-во строк первой матрицы;
- M – кол-во столбцов первой матрицы и кол-во строк второй матрицы;
- Q – кол-во столбцов второй матрицы.

**Трудоемкость классический алгоритм перемножения матриц**

Трудоемкость стандартного алгоритма умножения матриц состоит из:

- внешнего цикла по  $i \in [1..M]$ , трудоемкость которого:  $f = 2 + M \cdot (2 + f_{body})$ ;

- цикла по  $j \in [1..N]$ , трудоемкость которого:  $f = 2 + N \cdot (2 + f_{body})$ ;
- цикла по  $k \in [1..Q]$ , трудоемкость которого:  $f = 2 + 14 \cdot Q$ .

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то

$$f_{classic} = 2 + M \cdot (4 + N \cdot (4 + 14Q)) = 2 + 4M + 4MN + 14MQN \approx 14MQN \quad (2.4)$$

### Трудоемкость алгоритм Винограда

Трудоемкость алгоритма Винограда складывается из:

- трудоемкости создания и инициализации массивов  $row$  и  $col$ :

$$f_{arrs} = f_{row} + f_{col} \quad (2.5)$$

- трудоемкость заполнения массива  $row$ :

$$f_{row} = 2 + N \cdot (2 + 4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = 2 + 6M + \frac{19MN}{2}; \quad (2.6)$$

- трудоемкость заполнения массива  $col$ :

$$f_{col} = 2 + Q \cdot (2 + 4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = 2 + 6Q + \frac{19MQ}{2}; \quad (2.7)$$

- трудоемкость цикла умножения для четных размеров:

$$f_{mul} = 2 + N \cdot (4 + Q \cdot (13 + 32\frac{M}{2})) = 2 + 4N + 13NQ + \frac{32MNQ}{2} = 2 + 4N + 13NQ + 16MNQ; \quad (2.8)$$

- трудоемкость цикла, выполняемого в случае нечетных размеров матрицы:

$$f_{oddLoop} = 3 + \begin{cases} 0, & \text{четная,} \\ 2 + N \cdot (4 + Q \cdot (2 + 14)), & \text{иначе.} \end{cases} \quad (2.9)$$

Таким образом, для нечетного размера матрицы имеем:

$$f_{odd} = f_{arrs} + f_{row} + f_{col} + f_{mul} + f_{oddLoop} \approx 16MNQ; \quad (2.10)$$

для четного:

$$f_{even} = f_{arrs} + f_{row} + f_{col} + f_{mul} + f_{oddLoop} \approx 16MNQ. \quad (2.11)$$

### Оптимизированный алгоритм Винограда

Трудоемкость оптимизации алгоритма Винограда осуществляется следующим образом:

- операция  $x = x + k$  заменяется на операцию  $x+ = k$ ;
- операция  $x \cdot 2$  заменяется на  $x << 1$ ;
- некоторые значения для алгоритма вычисляются заранее.

Тогда трудоемкость алгоритма Винограда с примененными оптимизациями складывается из:

- трудоемкости предвычисления значения  $\frac{M}{2}$ , равной 3;
- трудоемкости  $f_{arrs}$  (2.5) создания и инициализации массивов  $row$  и  $col$ ;
- трудоемкость заполнения массива  $row$ :

$$f_{row} = 2 + N \cdot \left( 2 + 2 + \frac{M}{2} \cdot (2 + 2 + 2 + 7) \right) = 2 + 2N + \frac{13MN}{2}; \quad (2.12)$$

- трудоемкость заполнения массива  $col$ :

$$f_{col} = 2 + Q \cdot \left( 2 + 2 + \frac{M}{2} \cdot (2 + 2 + 2 + 7) \right) = 2 + 2Q + \frac{13MQ}{2}; \quad (2.13)$$

– трудоемкость цикла умножения для четных размеров:

$$\begin{aligned} f_{mul} &= 2 + N \cdot (4 + Q \cdot (4 + 2 + \frac{M}{2} \cdot (2 + 2 + 3 + 6 + 2 + 6))) = \\ &= 2 + 4N + 13NQ + \frac{32MNQ}{2} = 2 + 4N + 13NQ + 16MNQ; \end{aligned} \quad (2.14)$$

– трудоемкость цикла, выполняемого в случае нечетных размеров матрицы:

$$f_{oddLoop} = 3 + \begin{cases} 0, & \text{четная,} \\ 2 + N \cdot (4 + Q \cdot (2 + 11)), & \text{иначе.} \end{cases} \quad (2.15)$$

Таким образом, для нечетного размера матрицы имеем:

$$f_{odd} = f_{arrs} + f_{row} + f_{col} + f_{mul} + f_{oddLoop} \approx \frac{19MNQ}{2}; \quad (2.16)$$

для четного:

$$f_{even} = f_{arrs} + f_{row} + f_{col} + f_{mul} + f_{oddLoop} \approx \frac{19MNQ}{2}. \quad (2.17)$$

### Трудоемкость алгоритма Штрассена

Трудоемкость алгоритма Штрассена осуществляется следующим образом:

Трудоемкость сложения или вычитания двух матриц размера  $N \times N$ , рассчитывается по формуле (2.18).

$$f_{sum}(N) = 2 + N \cdot (2 + 2 + N \cdot (2 + 8)) = 10N^2 + 4N + 2 \quad (2.18)$$

Тогда трудоемкость описывается рекуррентной формулой (2.19).

$$T(n) = \begin{cases} 7, & n = 1 \\ 7T(\frac{n}{2}) + 18f_{sum}(\frac{n}{2}) + 65, & n > 1 \end{cases} \quad (2.19)$$

Тогда по формуле (2.20) возможно рассчитать трудоемкость для матриц

порядка  $n$ .

$$T(n) = 7^{\log_2 n} T(1) + \sum_{i=0}^{(\log_2 n)-1} (7^i (18 f_{sum}(\frac{n}{2^{i+1}}) + 65)) \quad (2.20)$$

## 3 Технологический раздел

В данном разделе будут приведены средства реализации, листинги кода реализации алгоритмов и функциональные тесты.

### 3.1 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран *C++* [5]. Данный выбор обусловлен наличием библиотеки для замера процессорного времени *ctime* [6].

### 3.2 Сведения о модулях программы

Программа состоит из трех программных модулей:

1. `main.cpp` – файл, который включает в себя точку входа программы, из которой осуществляется вызов алгоритмов через разработанный интерфейс;
2. `algorithms.cpp`, `algorithms.h` – модуль с реализацией алгоритмов;
3. `measurement.cpp`, `measurement.h` – модуль замера времени работы алгоритмов.

### 3.3 Реализация алгоритмов

В листинге 3.1 приведена реализация классического алгоритма умножения двух матриц, в листинге 3.2 приведена реализация алгоритма Винограда, в листинге 3.3 приведена реализация оптимизированного алгоритма Винограда, в листинге 3.4 приведена реализация алгоритма Штрассена.

Листинг 3.1 – Функция классического алгоритма умножения матриц

```
1 vector<vector<int>> Matrix::classicAlg(const vector<vector<int>>
   mtrx, const vector<vector<int>> mtry, int n, int m, int k)
2 {
3     vector<vector<int>> mtr_res(n, vector<int>(k, 0));
4
5     for (int i = 0; i < n; ++i)
6         for (int j = 0; j < k; ++j)
7             for (int u = 0; u < m; ++u)
8                 mtr_res[i][j] += mtrx[i][u] * mtry[u][j];
9
10    return mtr_res;
11 }
```



### Листинг 3.2 – Функция алгоритма Винограда

```

1  vector<vector<int>> Matrix::winogradAlg(const
    vector<vector<int>> mtrx, const vector<vector<int>> mtry, int
    n, int m, int k)
2  {
3      vector<vector<int>> mtr_res(n, vector<int>(k, 0));
4
5      vector<int> row(n, 0);
6      vector<int> col(k, 0);
7
8      for (int i = 0; i < n; ++i)
9          for (int j = 0; j < m / 2; ++j)
10             row[i] = row[i] + mtrx[i][2 * j] * mtrx[i][2 * j +
                1];
11
12     for (int i = 0; i < k; ++i)
13         for (int j = 0; j < m / 2; ++j)
14             col[i] = col[i] + mtry[2 * j][i] * mtry[2 * j +
                1][i];
15
16     for (int i = 0; i < n; ++i)
17         for (int j = 0; j < k; ++j)
18         {
19             mtr_res[i][j] = -row[i] - col[j];
20             for (int u = 0; u < m / 2; ++u)
21                 mtr_res[i][j] = mtr_res[i][j] + (mtrx[i][2 * u]
                    + mtry[2 * u + 1][j]) * (mtrx[i][2 * u + 1] +
                    mtry[2 * u][j]);
22         }
23
24     if (m % 2 == 1)
25         for (int i = 0; i < n; ++i)
26             for (int j = 0; j < k; ++j)
27                 mtr_res[i][j] = mtr_res[i][j] + mtrx[i][m - 1] *
                    mtry[m - 1][j];
28     return mtr_res;

```

### Листинг 3.3 – Функция оптимизированного алгоритма Винограда

```

1  vector<vector<int>> Matrix::winogradAlg(const
    vector<vector<int>> mtrx, const vector<vector<int>> mtry, int
    n, int m, int k)
2  {
3      vector<vector<int>> mtr_res(n, vector<int>(k, 0));
4
5      vector<int> row(n, 0);
6      vector<int> col(k, 0);
7
8      int half_m = m >> 1;
9
10     for (int i = 0; i < n; ++i)
11         for (int j = 0; j < half_m; ++j)
12             row[i] += mtrx[i][j << 1] * mtrx[i][(j << 1) + 1];
13
14     for (int i = 0; i < k; ++i)
15         for (int j = 0; j < half_m; ++j)
16             col[i] += mtry[j << 1][i] * mtry[(j << 1) + 1][i];
17
18     for (int i = 0; i < n; ++i)
19         for (int j = 0; j < k; ++j)
20         {
21             mtr_res[i][j] = -row[i] - col[j];
22             for (int u = 0; u < half_m; ++u)
23                 mtr_res[i][j] += (mtrx[i][u << 1] + mtry[(u <<
                    1) + 1][j]) * (mtrx[i][(u << 1) + 1] + mtry[u
                    << 1][j]);
24         }
25
26     if (m % 2 == 1)
27         for (int i = 0; i < n; ++i)
28             for (int j = 0; j < k; ++j)
29                 mtr_res[i][j] += mtrx[i][m - 1] * mtry[m - 1][j];
30     return mtr_res;

```

### Листинг 3.4 – Функция алгоритма Штрассена

```
1  vector<vector<int>> Matrix::strassen(vector<vector<int>> mtrx,
    vector<vector<int>> mtry)
2  {
3      size_t rows = mtrx.rows();
4
5      if (rows <= 2)
6          return classicAlg(mtrx, mtry);
7
8      size_t half_row = mtrx.rows() / 2;
9
10     auto a11 = mtrx.slice(0, half_row, 0, half_row);
11     auto a12 = mtrx.slice(0, half_row, half_row, rows);
12     auto a21 = mtrx.slice(half_row, rows, 0, half_row);
13     auto a22 = mtrx.slice(half_row, rows, half_row, rows);
14
15     auto b11 = mtry.slice(0, half_row, 0, half_row);
16     auto b12 = mtry.slice(0, half_row, half_row, rows);
17     auto b21 = mtry.slice(half_row, rows, 0, half_row);
18     auto b22 = mtry.slice(half_row, rows, half_row, rows);
19
20     auto p1 = strassen(a11 + a22, b11 + b22);
21     auto p2 = strassen(a22, b21 - b11);
22     auto p3 = strassen(a11 + a12, b22);
23     auto p4 = strassen(a12 - a22, b21 + b22);
24     auto p5 = strassen(a11, b12 - b22);
25     auto p6 = strassen(a21 + a22, b11);
26     auto p7 = strassen(a11 - a21, b11 + b12);
27
28     auto c11 = p1 + p2 - p3 + p4;
29     auto c12 = p5 + p3;
30     auto c21 = p6 + p2;
31     auto c22 = p5 + p1 - p6 - p7;
32
33     return combine(c11, c12, c21, c22);
34 }
```

## 4 Исследовательская часть

В данном разделе будут приведены: технические характеристики устройства, демонстрация работы программы, сравнительный анализ времени выполнения реализуемых алгоритмов и занимаемой памяти.

### 4.1 Технические характеристики

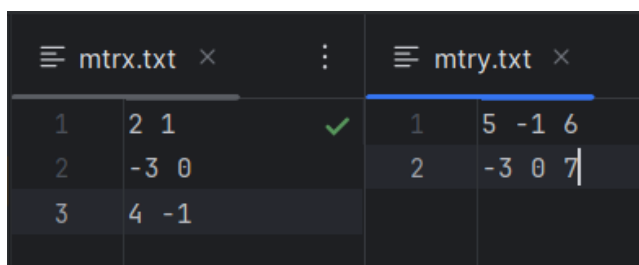
Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

- Процессор: Ryzen 5 4600H, тактовая частота ЦПУ 3.0 ГГц, максимальная частота процессора 4.0 ГГц [7].
- Оперативная память: 16 ГБайт.
- Операционная система: Windows 10 Pro 64-разрядная система [8].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

### 4.2 Демонстрация работы программы

На рисунке 4.2 представлено визуальное представление работы разработанного программного обеспечения. В частности, продемонстрирован результат вычисления классического алгоритма умножения матриц. Входными данными являются матрицы и располагаются в двух разных файлах, которые имеют следующий вид 4.1



≡ mtrx.txt ×		⋮	≡ mtry.txt ×	
1	2 1	✓	1	5 -1 6
2	-3 0		2	-3 0 7
3	4 -1			

Рисунок 4.1 – Пример входных данных

```
Menu
1. Standard matrix multiplication.
2. Winograd algorithm multiplication.
3. Optimized winograd algorithm multiplication.
4. Strassen algorithm multiplication.
5. Measure time for implemented algorithms.
6. Edit matrices
0. Exit.

Choose an option (0-6):1

| 7 -2 19 |
| -15 3 -18 |
| 23 -4 17 |
```

Рисунок 4.2 – Демонстрация работы программы

### 4.3 Временные характеристики

Замеры времени работы реализованных алгоритмов для определенного размера квадратных матриц проводились 100 раз, а затем бралось их среднее арифметическое значение. Значения для матриц генерировались случайно.

На рисунке 4.3 представлен результат замеров времени классического алгоритма умножения, алгоритма Винограда (с оптимизацией и без нее) для нечетного размера квадратных матриц.

На рисунке 4.4 представлен результат замеров времени классического алгоритма умножения, алгоритма Винограда (с оптимизацией и без нее) и алгоритма Штрассена для матриц, размер которых — степени 2.

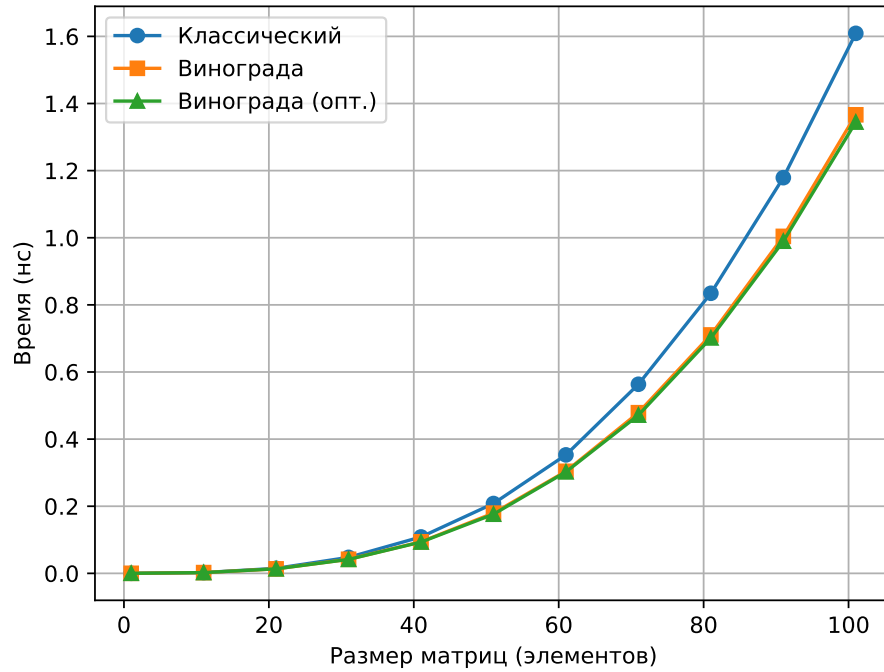


Рисунок 4.3 – Результат замеров времени реализуемых алгоритмов на матрицах нечетных размеров

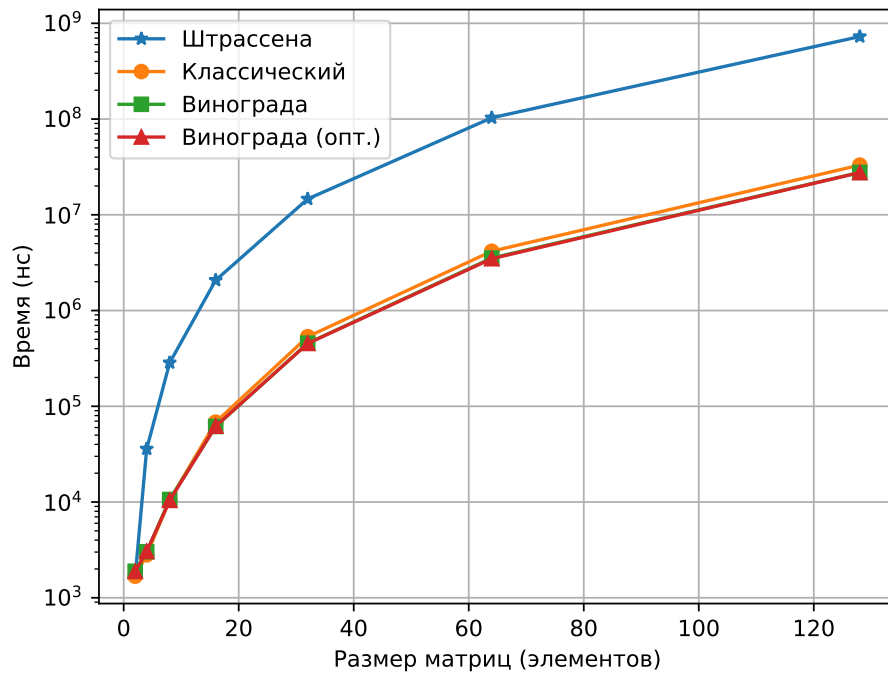


Рисунок 4.4 – Результат замеров времени реализуемых алгоритмов на матрицах размеры которых — степени 2

## 4.4 Характеристики по памяти

Введем следующие обозначения:

- $\text{size}(v)$  — функция, вычисляющая размер входного параметра  $v$  в байтах;
- $\text{int}$  — целочисленный тип данных.

### Классический алгоритм

Теоретическая оценка объема используемой памяти для реализации классического алгоритма умножения целочисленных матриц, размером  $n \times n$ :

$$\begin{aligned} M_{Classic} &= 3 \cdot \text{size}(\text{int}) + 3 \cdot \text{size}(\text{int}) + n \cdot n \cdot \text{size}(\text{int}) = \\ &= \text{size}(\text{int}) \cdot (6 + n^2), \end{aligned} \quad (4.1)$$

где  $3 \cdot \text{size}(\text{int})$  — размер переменных для хранения размеров матриц,  
 $3 \cdot \text{size}(\text{int})$  — размер переменных цикла,  
 $n \cdot n \cdot \text{size}(\text{int})$  — размер результирующей матрицы.

### Алгоритм Винограда

Теоретическая оценка объема используемой памяти для реализации алгоритма Винограда (матрицы целочисленные и размером  $n \times n$ ):

$$M_{Winograd} = M_{row} + M_{col} + M_{res} + M_{mul} + M_{odd}, \quad (4.2)$$

где  $M_{row}, M_{col}$  — размер вспомогательных массивов,  
 $M_{res}$  — размер результирующей матрицы,  
 $M_{mul}$  — размер памяти, используемой при умножении матриц,  
 $M_{odd}$  — размер памяти, используемой при обработке условия о нечетности размеров матриц.

Размер результирующей матрицы рассчитывается по формуле (4.3)

$$M_{res} = n \cdot n \cdot \text{size}(\text{int}). \quad (4.3)$$

Количество памяти, затрачиваемой на хранение массивов  $row$  и  $col$  равно

$$M_{row} = M_{col} = n \cdot \text{size}(\text{int}). \quad (4.4)$$

При умножении размер используемой памяти равен

$$M_{mul} = 3 \cdot \text{size}(int), \quad (4.5)$$

где  $3 \cdot \text{size}(int)$  — переменные цикла.

Количество памяти, затрачиваемой на обработку случая, когда матрица имеет нечетный размер, равно

$$M_{odd} = \begin{cases} 0, & \text{четная} \\ 2 \cdot \text{size}(int), & \text{нечетная} \end{cases} \quad (4.6)$$

### Оптимизированный алгоритм Винограда

Теоретическая оценка объема используемой памяти для реализации оптимизированной версией алгоритма Винограда (матрицы целочисленные и размером  $n \times n$ ):

$$M_{WinogradOpt} = M_{row} + M_{col} + M_{res} + M_{mul} + M_{odd} + \text{size}(int), \quad (4.7)$$

где  $M_{row}, M_{col}$  — размер вспомогательных массивов,

$M_{res}$  — размер результирующей матрицы,

$M_{mul}$  — размер памяти, используемой при умножении матриц,

$M_{odd}$  — размер памяти, используемой при обработке условия о нечетности размеров матриц,

$\text{size}(int)$  — размер переменной, используемой для кеширования.

Размер памяти, необходимой для обработки массивов  $row$  и  $col$ , равен

$$M_{row} = M_{col} = n \cdot \text{size}(int) + \text{size}(int), \quad (4.8)$$

где  $\text{size}(int)$  — размер переменной, используемой для кеширования.

Количество памяти, затрачиваемой на умножение, равно

$$M_{mul} = 3 \cdot \text{size}(int) + \text{size}(int), \quad (4.9)$$

где  $\text{size}(int)$  — размер переменной, используемой для кеширования.

$M_{res}$  и  $M_{odd}$  рассчитываются согласно соотношениям (4.3) и (4.6) соот-



вественно.

### Алгоритм Штрассена

Теоретическая оценка объема используемой памяти, затрачиваемой при каждом рекурсивном вызове для реализации алгоритма Штрассена (матрицы целочисленные и размером  $n \times n$ ):

$$M_{StrassenCall} = (4 \cdot \frac{n}{2} \cdot \frac{n}{2} + 4 \cdot \frac{n}{2} \cdot \frac{n}{2} + 11 \cdot \frac{n}{2} \cdot \frac{n}{2} + n \cdot n) \cdot \text{size}(int) + \\ + 2 \cdot \text{size}(int), \quad (4.10)$$

где  $2 \cdot \text{size}(int)$  — размер переменных, используемых для кеширования,  
 $4 \cdot \frac{n}{2} \cdot \frac{n}{2} \cdot \text{size}(int)$  — размер матриц, полученных в результате разбиения исходных на 4 части,  
 $11 \cdot \frac{n}{2} \cdot \frac{n}{2} \cdot \text{size}(int)$  — размер промежуточных матриц,  
 $n \cdot n \cdot \text{size}(int)$  — размер результирующей матрицы.

## 4.5 Вывод

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы:

1. реализация оптимизированного алгоритма Винограда оказалась более эффективной по времени, независимо от размерности входных матриц (см. рисунки 4.3 – 4.4).
2. реализация алгоритма Штрассена по итогам исследования оказалась наименее эффективной по времени (см. рисунок 4.4). Полученный результат объясняется малыми размерами матриц. Однако из-за того, что алгоритм рекурсивный, достижение необходимых размеров матриц, при которых реализация алгоритма Штрассена показала бы преимущество, оказывается невозможным.
3. на матрицах нечетного размера оптимизированная и неоптимизированная реализация алгоритмов Винограда работают медленнее, чем на матрицах четного размера (см. рисунки 4.3 – 4.4). Это происходит из-за дополнительных вычислений для крайних строк и столбцов в результирующей матрице.

В результате теоретической оценки объема используемой памяти для реализаций алгоритмов можно сделать следующие выводы:

1. реализация классического алгоритма умножения матриц требует наименьших расходов по памяти.
2. реализация алгоритма Штрассена, напротив, является самой требовательной по памяти за счет использования вспомогательных подматриц для выполнения расчетов и рекурсивных вызовов.
3. оптимизированная реализация алгоритма Винограда более ресурсозатратна по сравнению с неоптимизированной, поскольку включает в себя использование дополнительных переменных для хранения промежуточных расчетов.

## ЗАКЛЮЧЕНИЕ

Цель работы достигнута, проведен сравнительный анализ следующих алгоритмов умножения матриц:

- классического алгоритма;
- алгоритма Штрассена;
- алгоритма Винограда;
- оптимизированная версия алгоритма Винограда;

В ходе выполнения лабораторной работы были решены все задачи:

- разработаны требуемые алгоритмы;
- оценена трудоемкость рассматриваемых алгоритмов;
- проведен сравнительный анализ времени выполнения реализуемых алгоритмов и занимаемой памяти.

В результате исследования реализуемых алгоритмов по времени выполнения были сделаны следующие выводы:

1. реализация оптимизированного алгоритма Винограда оказалась наиболее эффективной по времени, независимо от размерности входных матриц (см. рисунки 4.3 – 4.4).
2. реализация алгоритма Штрассена по итогам исследования оказалась наименее эффективной по времени (см. рисунок 4.4). Полученный результат объясняется малыми размерами матриц. Однако из-за того, что алгоритм рекурсивный, достижение необходимых размеров матриц, при которых реализация алгоритма Штрассена показала бы преимущество, оказывается невозможным.
3. на матрицах нечетного размера оптимизированная и неоптимизированная реализация алгоритмов Винограда работают медленнее, чем на матрицах четного размера (см. рисунки 4.3 – 4.4). Это происходит из-за дополнительных вычислений для крайних строк и столбцов в результирующей матрице.

В результате теоретической оценки объема используемой памяти для реализаций алгоритмов были сделаны следующие выводы:

1. реализация классического алгоритма умножения матриц требует наименьших расходов по памяти.
2. реализация алгоритма Штрассена, напротив, является самой требовательной по памяти за счет использования вспомогательных подматриц для выполнения расчетов и рекурсивных вызовов.
3. оптимизированная реализация алгоритма Винограда более ресурсозатратна по сравнению с неоптимизированной, поскольку включает в себя использование дополнительных переменных для хранения промежуточных расчетов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Конев В. В.* Электронный учебник по линейной алгебре. — Режим доступа: [https://portal.tpu.ru/SHARED/k/KONVAL/Sites/Russian\\_sites/index1.htm](https://portal.tpu.ru/SHARED/k/KONVAL/Sites/Russian_sites/index1.htm) (дата обращения: 15.10.2023).
2. *Желудков А. В., Макаров Д. В.* Исследование алгоритмов перемножения матриц. — 2017.
3. *Головашкин Д. Л.* векторные алгоритмы вычислительной линейной алгебры. — Режим доступа: <https://inlnk.ru/LAYY70> (дата обращения: 15.10.2023).
4. *Охотин А.* Лекция 5: Реализация очереди с приоритетами в алгоритме Дейкстры. Пути между всеми парами вершин в графе, алгоритм Варшалла, алгоритм Варшалла-Флойда. Использование умножения матриц. Алгоритм Штрассена быстрого умножения матриц. Умножение булевых матриц через числовые. — Режим доступа: [https://users.math-cs.spbu.ru/~okhotin/teaching/algorithms\\_2019/okhotin\\_algorithms\\_2019\\_15.pdf](https://users.math-cs.spbu.ru/~okhotin/teaching/algorithms_2019/okhotin_algorithms_2019_15.pdf) (дата обращения: 15.10.2023).
5. Документация по C++20. — Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170/> (дата обращения: 20.11.2023).
6. Standard library header <ctime>. — Режим доступа: <https://en.cppreference.com/w/cpp/header/ctime> (дата обращения: 20.11.2023).
7. Ryzen 4600H. — Режим доступа: <https://www.amd.com/en/products/apu/amd-ryzen-5-4600h> (дата обращения: 20.09.2023).
8. Windows 10 Pro 2h21 64-bit. — Режим доступа: <https://www.microsoft.com/ru-ru/software-download/windows10> (дата обращения: 25.09.2022).