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 №3
по курсу «Анализы Алгоритмов»
на тему: «Трудоемкость сортировок»

Студент ИУ7-56Б — Мансуров В. М.
(Группа) (Фамилия И. О.)

Преподаватели — Волкова Л. Л., Строганов Ю. В.
(Фамилия И. О.)

Москва — 2022 г.

Содержание

Введение

В данной лабораторной работе будет рассмотрено алгоритмы умножения матриц. В программировании, как и в математике, часто приходится прибегать к использованию матриц. Существует огромное количество областей их применения в этих сферах. Матрицы активно используются при выводе различных формул в физике, таких, как:

- градиент;
- дивергенция;
- ротор.

Нельзя обойти стороной и различные операции над матрицами – сложение, возведение в степень, умножение. При различных задачах размеры матрицы могут достигать больших значений, поэтому оптимизация операций работы над матрицами является важной задачей в программировании. В данной лабораторной работе пойдёт речь об оптимизациях операции умножения матриц.

Целью данной лабораторной работы является изучение, реализация и исследование алгоритмов умножения матриц.

Для поставленной цели необходимо выполнить следующие задачи:

1) изучить алгоритмы умножения матриц;
2) создать ПО, реализующее следующие алгоритмы:
    - классический алгоритм умножения матриц;
    - алгоритм Винограда;
    - оптимизированный алгоритм Винограда.
3) Оценить трудоемкости сортировок;
4) Провести анализ затрат работы программы по времени, выяснить влияющие на них характеристики.
5) Провести сравнительный анализ между алгоритмами.

1 Аналитическая часть

В этом разделе будут рассмотрены классический алгоритм умножения матриц и алгоритм Винограда, а также его оптимизированная версия.

1.1 Матрица

Матрицой [1] называют таблицу чисел a_{ik} вида

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \tag{1.1}$$

состоящую из m строк и n столбцов. Числа a_{ik} называются её *элементами.*

Пусть A – матрица, тогда $A_{i,j}$ – элемент этой матрицы, который находится на *i-ой* строке и *j-ом* столбце.

Можно выделить следующие операции над матрицами:

1) сложение матриц одинакового размера;

2) вычитание матриц одинакового размера;

3) умножение матриц в случае, если количество столбцов первой матрицы равно количеству строк второй матрицы. В итоговой матрице количество строк будет, как у первой матрицы, а столбцов – как у второй.

Замечание: операция умножения матриц не коммутативна – если A и B – квадратные матрицы, а C – результат их перемножения, то произведение AB и BA дадут разный результат C.

1.2 Классический алгоритм

Пусть даны две матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.2}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.3}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{1.4}$$

будет называться произведением матриц A и B [1].

Классический алгоритм реализует данную формулу.

1.3 Алгоритм Винограда

Алгоритм Винограда [2] – алгоритм умножения квадратных матриц. В 1987 году Дон Копперсмит и Виноград опубликовали метод, содержащий асимптотическую сложность $O(n^{2,3755})$ и и улучшили его в 2011 до $O(n^{2,373})$, где n – размер стороны матрицы. После доработки он стал обладать лучшей асимптотикой среди всех алгоритмов умножения матриц.

Рассмотрим пример, пусть есть два вектора $U = (u_1, u_2, u_3, u_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $U \cdot W = u_1w_1 + u_2w_2 + u_3w_3 + u_4w_4$, что эквивалентно 1.5:

$$U \cdot W = (u_1 + w_2)(u_2 + w_1) + (u_3 + w_4)(u_4 + w_3) - u_1u_2 - u_3u_4 - w_1w_2 - w_3w_4 \tag{1.5}$$

Пусть матрицы A, BC, ранее определенных размеров. Упомянутое скалярное произведение, по замыслу Винограда, можно произвести иначе в фор-

муле 1.6:

$$C_{ij} = \sum_{k=1}^{q/2}(a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \sum_{k=1}^{q/2} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{q/2} b_{2k-1,j}b_{2k,j} \quad (1.6)$$

Казалось бы, это только увеличит количество арифметических операций по сравнению с классическим методом, однако Виноград предложил находить второе и третье слагаемые в Формуле 1.6 предварительном этапе вычислений, заранее для каждой строки матрицы A и столбца B соответственно. Так, вычислив единожды для строки i матрицы A значение выражения $\sum_{k=1}^{q/2} a_{i,2k-1}a_{i,2k}$ его можно далее использовать m раз при нахождении элементов i-ой строчки матрицы C. Аналогично, вычислив единожды для столбца j матрицы B значение выражения $\sum_{k=1}^{q/2} b_{2k-1,j}b_{2k,j}$ его можно далее использовать n раз при нахождении элементов j-ого столбца матрицы C. [2]

За счёт предварительной обработки данных можно получить прирост производительности: несмотря на то, что полученное выражение требует большего количества операций, чем стандартное умножение матриц, выражение в правой части равенства можно вычислить заранее и запомнить для каждой строки первой матрицы и каждого столбца второй матрицы. Это позволит выполнить лишь два умножения и пять сложений, при учёте, что потом будет сложено только с двумя предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Операция сложения выполняется быстрее, поэтому на практике алгоритм должен работать быстрее обычного алгоритма перемножения матриц.

Стоит упомянуть, что при нечётном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

1.4 Оптимизированный алгоритма Винограда

При программной реализации рассмотренного выше алгоритма Винограда можно сделать следующие оптимизации:

1. значение $\frac{N}{2}$, используемое как ограничения цикла подсчёта предварительных данных, можно кэшировать;

2. операцию умножения на 2 программно эффективнее реализовывать как побитовый сдвиг влево на 1;

3. операции сложения и вычитания с присваиванием следует реализовывать при помощи соответствующего оператора $+=$ или $-=$ (при наличии данных операторов в выбранном языке программирования).

1.5 Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц – классического и Винограда, который имеет большую эффективность за счёт предварительных вычислений. Также были рассмотрены оптимизации, которые можно учесть при программной реализации алгоритма Винограда.

Алгоритмы будут получать на вход две матрицы, причём количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы. При вводе пустой матрицы будет выведено сообщение об ошибке. Требуется реализовать программное обеспечение, которое даёт возможность выбрать один из алгоритмов или все сразу, ввести две матрицы и вывести результат их перемножения. Также необходимо провести замеры времени работы реализаций алгоритмов для чётных и нечётных размеров матриц и сравнить результаты, используя графическое представление.

2 Конструкторская часть

В этом разделе будут представлены описания используемых типов данных, а также схемы алгоритмов перемножения матриц – стандартного, Винограда и оптимизации алгоритма Винограда

2.1 Функциональные требования

К программе предъявлены ряд функциональных требований:

- входные данные – размеры матрицы, целые числа и матрица, двумерный массив целых чисел;
- выходные данные – матрица;

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма для стандартного умножения матриц. На рисунках 2.2,2.3 схема алгоритма Винограда умножения матриц, а на 2.4,2.5 — схема оптимизированного алгоритма Винограда.

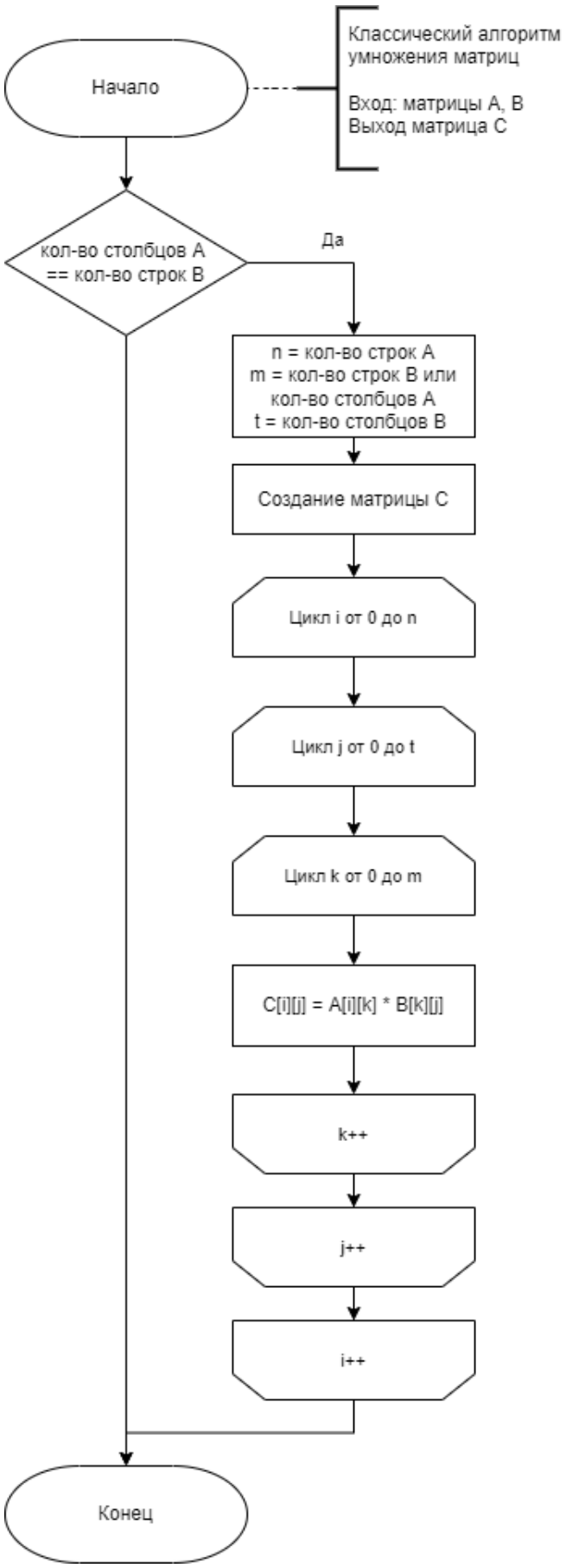

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

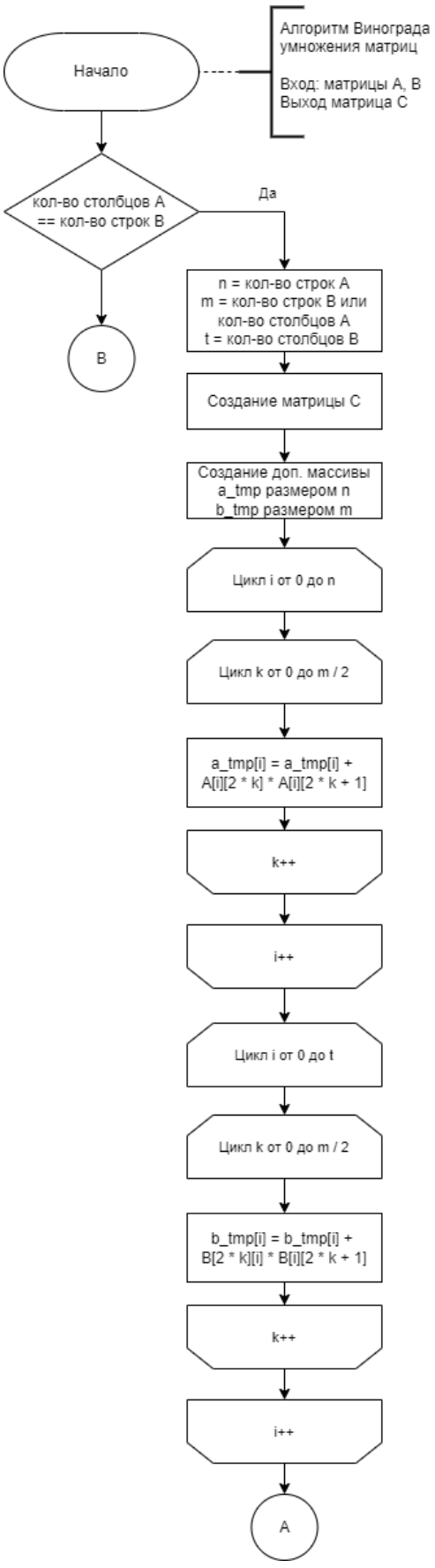

Рисунок 2.2 – Схема умножения матриц по алгоритму Винограда (Часть 1)

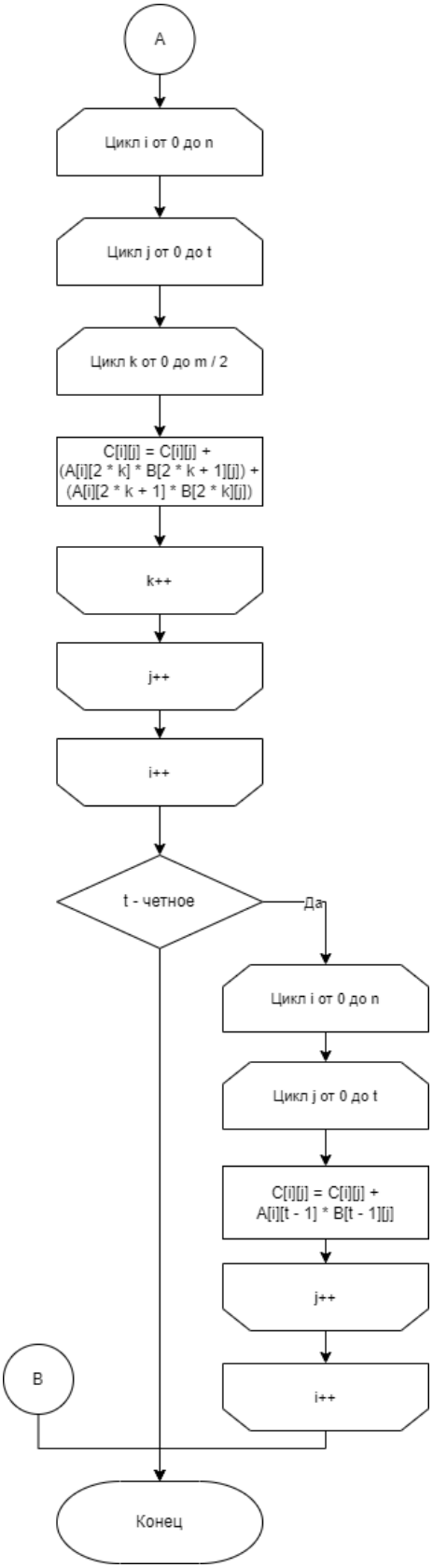

Рисунок 2.3 – Схема умножения матриц по алгоритму Винограда (Часть 2)

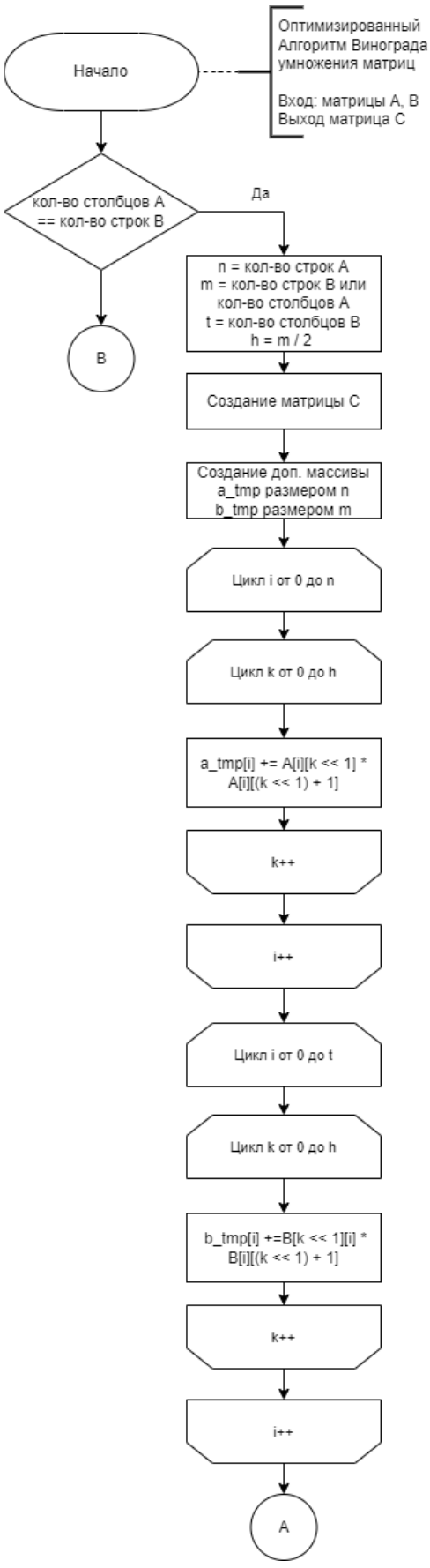

Рисунок 2.4 – Схема умножения матриц по оптимизированному алгоритму Винограда (Часть 1)

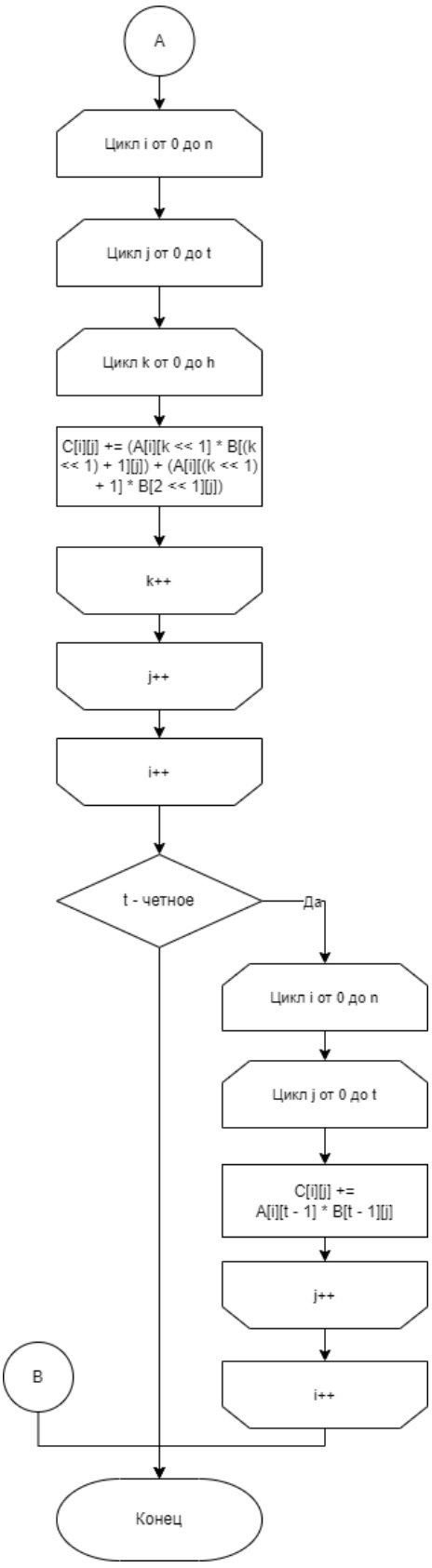

Рисунок 2.5 – Схема умножения матриц по оптимизированному алгоритму Винограда (Часть 2)

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

- количество строк – целое число;
- количество столбцов – целое число;
- матрица – двумерный список целых чисел

2.4 Модель вычислений для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма сортировки:

1. трудоемкость базовых операций имеет:
   - равную 1, указанные в 2.1:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

   - равную 2, указанные в 2.2:

$$*, /, \%, * =, / =, \% = \tag{2.2}$$

2. трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай} \\ max(f_1, f_2), & \text{худший случай} \end{cases} \tag{2.3}$$

3. трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

4. Трудоемкость передачи параметра в функции и возврат из функции равны 0.

2.5 Трудоемкость алгоритмов

Рассчитаем трудоемкость алгоритмов умножения матриц.

2.5.1 Классический алгоритм

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться из:

- внешнего цикла по $i \in [1 \dots N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;
- цикла по $j \in [1 \dots T]$, трудоёмкость которого: $f = 2 + 2 + T \cdot (2 + f_{body})$;
- цикла по $k \in [1 \dots M]$, трудоёмкость которого: $f = 2 + 2 + 14M$;

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то:

$$\begin{aligned} f_{standart} = 2 + N \cdot (2 + 2 + T \cdot (2 + 2 + M \cdot (2 + 8 + 1 + 1 + 2))) = \\ = 2 + 4N + 4NT + 14NMT \approx 14NMT = O(N^3) \end{aligned} \tag{2.5}$$

2.5.2 Алгоритм Винограда

Чтобы вычислить трудоемкость алгоритма Винограда, нужно учесть следующее:

- создания и инициализации массивов a_{tmp} и b_{tmp}, трудоёмкость которых 2.6:

$$f_{init} = N + M \tag{2.6}$$

- заполнения массива a_{tmp}, трудоёмкость которого 2.7:

$$\begin{aligned} f_{atmp} = 2 + N \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 4N + \frac{19NM}{2} = 2 + 4N + 9,5NM \end{aligned} \tag{2.7}$$

- заполнения массива b_{tmp}, трудоёмкость которого 2.8:

$$\begin{aligned} f_{btmp} = 2 + T \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 4T + \frac{19TM}{2} = 2 + 4T + 9,5TM \end{aligned} \tag{2.8}$$

- цикла заполнения для чётных размеров, трудоёмкость которого 2.9:

$$\begin{aligned} f_{cycle} = 2 + N \cdot (4 + T \cdot (2 + 7 + 4 + \frac{M}{2} \cdot (4 + 28))) = \\ = 2 + 4N + 13NT + \frac{32NTM}{2} = 2 + 4N + 13NT + 16NTM \end{aligned} \tag{2.9}$$

- цикла, который дополнительно нужен для подсчёта значений при нечётном размере матрицы, трудоемкость которого 2.10:

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + N \cdot (4 + T \cdot (2 + 14)) = 2 + 4N + 16NT, & \text{иначе} \end{cases} \tag{2.10}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем 2.11:

$$f_{worst} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3) \tag{2.11}$$

Для лучшего случая (чётный общий размер матриц) имеем 2.12:

$$f_{best} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3) \tag{2.12}$$

2.5.3 Оптимизированный алгоритм Винограда

Оптимизация заключается в:

- кэшировании значения $\frac{M}{2}$ в циклах;
- использовании побитового сдвига вместо умножения на 2;
- замены операции сложения и вычитания на операции $+=$ и $-=$ соответственно.

Тогда трудоемкость оптимизированного алгоритма Винограда состоит из:

- кэширования значения $\frac{M}{2}$ в циклах, которое равно 3;
- создания и инициализации массивов a_{tmp} и b_{tmp} 2.6;
- заполнения массива a_{tmp}, трудоёмкость которого 2.7;
- заполнения массива b_{tmp}, трудоёмкость которого 2.8;
- цикла заполнения для чётных размеров, трудоёмкость которого 2.13:

$$\begin{gathered} f_{cycle} = 2 + N \cdot (4 + T \cdot (4 + 7 + \frac{M}{2} \cdot (2 + 10 + 5 + 2 + 4))) = \\ = 2 + 4N + 11NT + \frac{23NTM}{2} = 2 + 4N + 11NT + 11,5 \cdot NTM \end{gathered} \tag{2.13}$$

- условие, которое нужно для дополнительных вычислений при нечётном размере матрицы, трудоемкость которого 2.14:

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + N \cdot (4 + T \cdot (2 + 10)) = 2 + 4N + 12NT, & \text{иначе} \end{cases} \tag{2.14}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем 2.15:

$$f_{worst} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMT = O(N^3) \tag{2.15}$$

Для лучшего случая (чётный общий размер матриц) имеем 2.16:

$$f_{best} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMT = O(N^3) \tag{2.16}$$

2.6 Вывод

В данном разделе были построены схемы алгоритмов умножения матриц, рассматриваемых в лабораторной работе, были описаны классы эквивалентности для функционального тестирования и модули программы. Проведённая теоретическая оценка трудоемкости алгоритмов показала, что в трудоёмкость выполнения алгоритма Винограда в случае оптимизации в 1.2 раза меньше, чем у простого алгоритма Винограда.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинг кода и функциональные тесты.

3.1 Требования к программному обеспечению

К программе предъявлены ряд требований:

- должна иметь интерфейс для выбора действий;
- должна иметь стандартные реализации вектора с динамическим выделением памяти;
- должна замерять процессорное время алгоритмов умножения матриц.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык $C++$[3]. Данный выбор обусловлен тем, что я имею некоторый опыт разработки на нем, а так же наличием у языка встроены библиотеки измерения процессорного времени и соответствие с выдвинутыми техническими требованиям.

В качестве измерения время использовалось функция *clock()* из библиотеки *ctime*[4].

3.3 Сведения о модулях программы

Данная программа разбита на следующие модули:

- `main.cpp` – Файл, содержащий точку входа в программу. В нем происходит общение с пользователем и вызов алгоритмов;
- `multy_mtr.cpp` — Файл содержит алгоритмы умножения матриц.

- `print_mtr.cpp` – Файл содержит функцию вывода матрицы.
- `read_mtr.cpp` – Файл содержит функции чтение матрицы.
- `cpu_time.cpp` — Файл содержит функции, замеряющее процессорное время алгоритмов умножения матриц.

Листинг 3.1 – Функция классического алгоритма умножения матриц

```
1  matrix_v standart_multy(const matrix_v &mtr1, const matrix_v &mtr2)
2  {
3      if (mtr1[0].size() != mtr2.size())
4          return empty_matrix();
5
6      size_t n = mtr1.size();
7      size_t m = mtr2.size();
8      size_t t = mtr2[0].size();
9
10     matrix_v res;
11
12     array_v row;
13     row.resize(t, 0);
14     res.resize(n, row);
15
16     for (size_t i = 0; i < n; i++)
17         for (size_t j = 0; j < t; j++)
18             for (size_t k = 0; k < m; k++)
19                 res[i][j] += mtr1[i][k] * mtr2[k][j];
20
21     return res;
22 }
```

Листинг 3.2 – Функция умножения матриц по алгоритму Винограда

```
matrix_v vinograd_alg(const matrix_v &mtr1, const matrix_v &mtr2)
{
    if (mtr1[0].size() != mtr2.size())
        return empty_matrix();

    size_t n = mtr1.size();
    size_t m = mtr2.size();
    size_t t = mtr2[0].size();
    matrix_v res;
    array_v rowSum, colSum;
    rowSum.resize(n, 0);
    colSum.resize(m, 0);
    res.resize(n, rowSum);

    for (size_t i = 0; i < n; i++)
        for (size_t j = 0; j < m / 2; j++)
            rowSum[i] = rowSum[i] + mtr1[i][2 * j] * mtr1[i][2 * j
               + 1];

    for (size_t i = 0; i < t; i++)
        for (size_t j = 0; j < m / 2; j++)
            colSum[i] = colSum[i] + mtr2[2 * j][i] * mtr2[2 * j +
               1][i];

    for (size_t i = 0; i < n; i++)
        for (size_t j = 0; j < t; j++) {
            res[i][j] = -rowSum[i] - colSum[j];
            for (size_t k = 0; k < m / 2; k++)
                res[i][j] = res[i][j] + (mtr1[i][2 * k + 1] +
                   mtr2[2 * k][j]) * (mtr1[i][2 * k] + mtr2[2 * k +
                   1][j]);
        }

    if (m % 2 == 1)
        for(size_t i = 0; i < n; i++)
            for(size_t j = 0; j < t; j++)
                res[i][j] = res[i][j] + mtr1[i][m - 1] * mtr2[m -
                   1][j];
    return res;
}
```

Листинг 3.3 – Функция умножения матриц по оптимизированному алгоритму Винограда

```
matrix_v opt_vinograd_alg(const matrix_v &mtr1, const matrix_v
    &mtr2)
{
    if (mtr1[0].size() != mtr2.size())
        return empty_matrix();

    size_t n = mtr1.size();
    size_t m = mtr2.size();
    size_t t = mtr2[0].size();
    size_t halfm = m / 2;
    matrix_v res;
    array_v rowSum, colSum;
    rowSum.resize(n, 0);
    colSum.resize(m, 0);
    res.resize(n, rowSum);

    for (size_t i = 0; i < n; i++)
        for (size_t j = 0; j < halfm; j++)
            rowSum[i] += mtr1[i][j << 1] * mtr1[i][(j << 1) + 1];

    for (size_t i = 0; i < t; i++)
        for (size_t j = 0; j < halfm; j++)
            colSum[i] += mtr2[(j << 1)][i] * mtr2[(j << 1) + 1][i];

    for (size_t i = 0; i < n; i++)
        for (size_t j = 0; j < t; j++) {
            res[i][j] = -rowSum[i] - colSum[j];
            for (size_t k = 0; k < halfm; k++)
                res[i][j] += (mtr1[i][(k << 1) + 1] + mtr2[k <<
                    1][j]) * (mtr1[i][k << 1] + mtr2[(k << 1) +
                    1][j]);
        }

    if (m % 2 == 1)
        for(size_t i = 0; i < n; i++)
            for(size_t j = 0; j < t; j++)
                res[i][j] += mtr1[i][m - 1] * mtr2[m - 1][j];
    return res;
}
```

Листинг 3.4 – Функции ввода матрицы

```
1  int read_size(int &n)
2  {
3      if (!(std::cin >> n) || n <= 0)
4          return 1;
5      return 0;
6  }
7
8  int read_mtr(matrix_v &mtr, std::string mode)
9  {
10     int n, m;
11     std::cout << "Ввод размер матрицы: ";
12     if (read_size(n)) {
13         std::cout << "Ошибка! Размер N введен неверно!" <<
               std::endl;
14         return 1;
15     }
16     if (mode == "square")
17         m = n;
18     else
19         if (read_size(m)) {
20             std::cout << "Ошибка! Размер M введен неверно!" <<
                   std::endl;
21             return 1;
22         }
23
24     array_v row;
25     row.resize(m, 0);
26     mtr.resize(n, row);
27
28     int rc = 0;
29     std::cout << "Ввод элементов матрицы:\n";
30     for (size_t i = 0; !rc && i < n; i++)
31         for (size_t j = 0; !rc && j < m; j++)
32             if (!(std::cin >> mtr[i][j])) {
33                 std::cout << "Ошибка! Элемент матрицы введен неверно
                       !" << std::endl;
34                 rc = 1;
35             }
36     return rc;
37 }
```

Листинг 3.5 – Функция вывода матрицы

```
1  void print_mtr(matrix_v mtr)
2  {
3      size_t n = mtr.size();
4      if (n == 0)
5          std::cout << "[]" << std::endl;
6      else {
7          size_t m = mtr[0].size();
8          if (m == 0) {
9              std::cout << "[";
10             for (size_t i = 0; i < n; i++)
11                 if (i == n - 1)
12                     std::cout << "[]";
13                 else
14                     std::cout << "[],";
15             std::cout << "]" << std::endl;
16         }
17         for (size_t i = 0; i < n; i++) {
18             for (size_t j = 0; j < m; j++) {
19                 std::cout << mtr[i][j] << "␣";
20             }
21             std::cout << std::endl;
22         }
23     }
24 }
```

3.4 Функциональные тесты

В таблице 3.1, 3.2 приведены функциональные тесты для разработанных алгоритмов сортировки. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты для классического алгоритма умножения матриц

Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix}1 & 5 & 7\\2 & 6 & 8\\3 & 7 & 9\end{pmatrix}$	( )	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix}1 & 5 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$
$\begin{pmatrix}3 & 5\\2 & 1\\9 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}$	$\begin{pmatrix}23 & 31 & 39\\6 & 9 & 12\end{pmatrix}$	$\begin{pmatrix}23 & 31 & 39\\6 & 9 & 12\end{pmatrix}$
$\begin{pmatrix}10\end{pmatrix}$	$\begin{pmatrix}35\end{pmatrix}$	$\begin{pmatrix}350\end{pmatrix}$	$\begin{pmatrix}350\end{pmatrix}$

Таблица 3.2 – Функциональные тесты для умножения матриц по алгоритму Винограда

Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix}1 & 5 & 7\\2 & 6 & 8\\3 & 7 & 9\end{pmatrix}$	( )	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix}1 & 5 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$
$\begin{pmatrix}3 & 5\\2 & 1\\9 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix}10\end{pmatrix}$	$\begin{pmatrix}35\end{pmatrix}$	$\begin{pmatrix}350\end{pmatrix}$	$\begin{pmatrix}350\end{pmatrix}$

3.5 Вывод

Были представлены листинги реализаций всех алгоритмов умножения матриц – стандартного, Винограда и оптимизированного алгоритма Винограда. Также в данном разделе была приведена информации о выбранных средствах для разработки алгоритмов

4 Исследовательская часть

4.1 Технические характеристики

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

- Процессор: Intel(R) Core(TM) i5-10300H CPU @ 2.50 ГГц 2.50 ГГц. [5]
- Оперативная память: 16 ГБ.
- Операционная система: Windows 11 Pro 64-разрядная система версией 22H2. [6]

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

4.2 Демонстрация работы программы

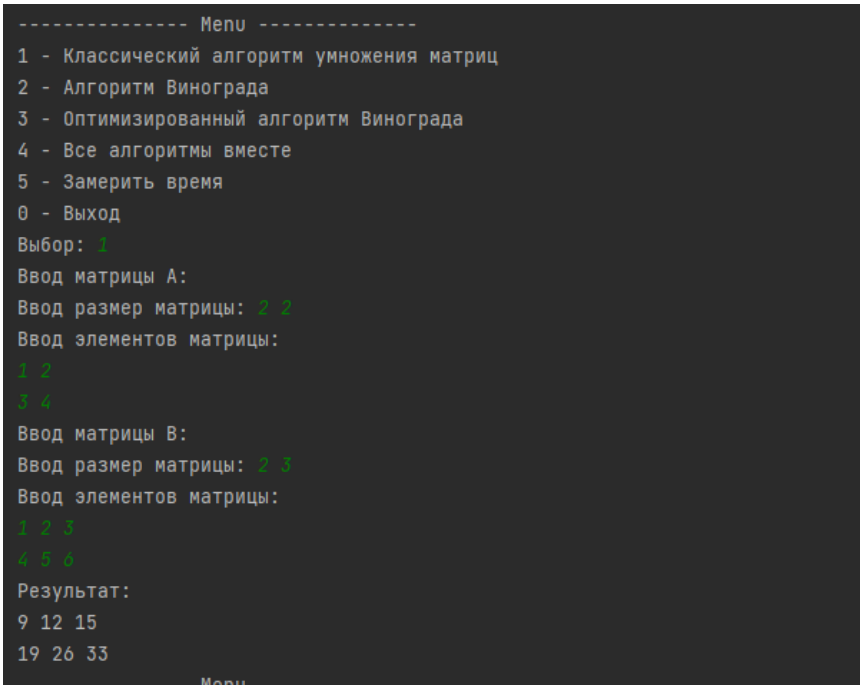

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в Таблицах 4.1, 4.2.

В Таблице 4.1 приведены результаты замеров по времени алгоритмов умножения матриц при четных размеров квадратных матриц, размеров от 10 до 100 по 1000 раз на различных входных матрицах.

Таблица 4.1 – Результаты замеров времени (чётные размеры матриц)

Размер матрицы	**Время, мс**		
	Классический	**Винограда**	**(опт.) Винограда**
10	0.02004	0	0.0501
20	0.02104	0.1503	0.07014
30	0.30261	0.2515	0.24248
40	0.61924	0.69539	0.53908
50	1.28958	1.20441	1.1232
60	2.10621	2.02906	1.7244
70	3.29259	3.17034	2.8377
80	4.93487	4.4499	3.9028
90	6.9018	6.49098	5.7926
100	9.38477	9.03607	7.7635

По таблице 4.1 был построен графики 4.2. Исходя из этих данных можно понять, что лучшего всего работает оптимизированный алгоритм Винограда, а классический алгоритм работаем в 1,5 раз хуже, чем алгоритм Винограда.

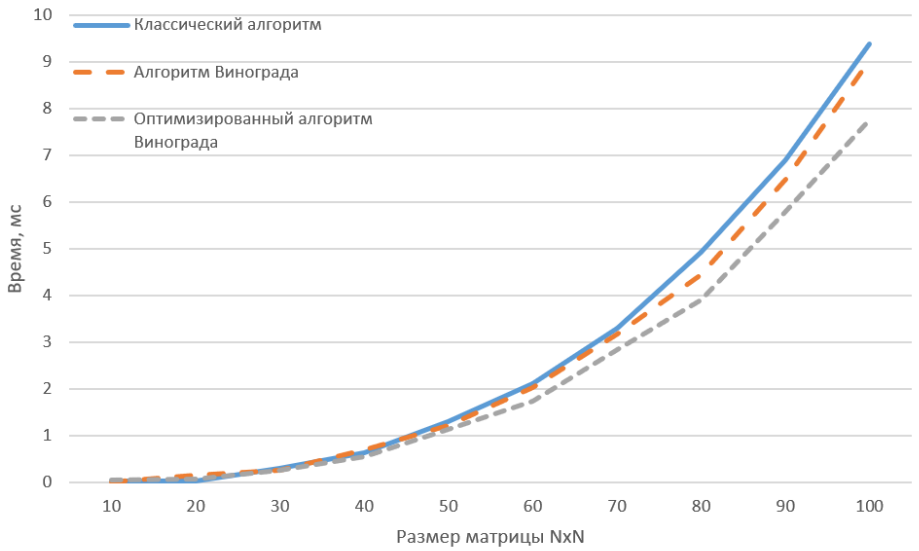

Рисунок 4.2 – Сравнение по времени алгоритмов умножения матриц на чётных размерах матриц

В Таблице 4.2 приведены результаты замеров по времени алгоритмов умножения матриц при нечетных размеров квадратных матриц, размеров от 11 до 101 по 1000 раз на различных данных.

Таблица 4.2 – Результаты замеров времени (нечётные размеры матриц)

Размер матрицы	**Время, мс**		
	Классический	**Винограда**	**(опт.) Винограда**
11	0.06012	0.07014	0.11022
21	0.18938	0.19238	0.22044
31	0.39379	0.43788	0.36072
41	0.73146	0.7515	0.61122
51	1.24048	1.31463	1.0421
61	2.80762	2.62425	2.4329
71	3.97094	3.98898	3.4449
81	5.37876	5.2495	4.7435
91	7.30261	7.10321	6.1172
101	9.58918	9.4499	8.1513

По таблице 4.2 был построен графики 4.3. Исходя из этих данных можно понять, что лучшего всего работает оптимизированный алгоритм Винограда, а классический алгоритм работаем в 1,1 раз хуже, чем алгоритм Винограда.

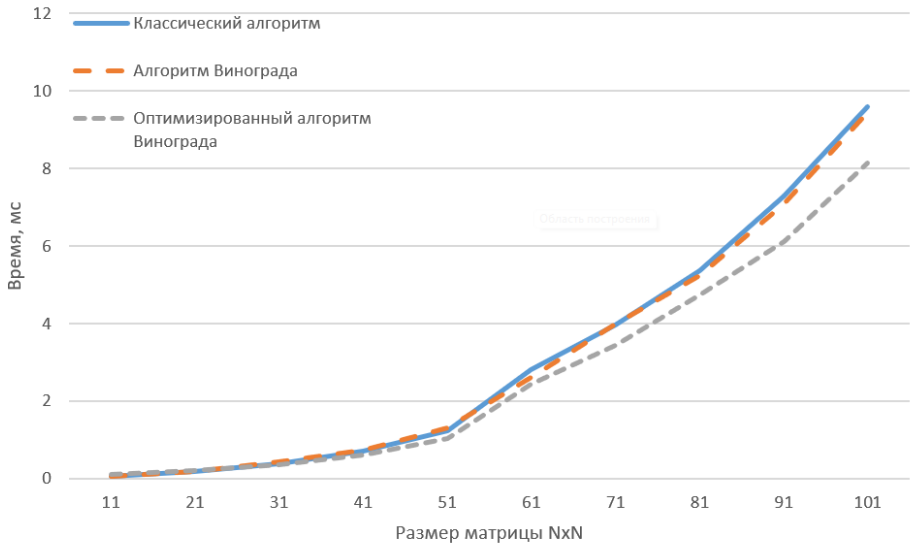

Рисунок 4.3 – Сравнение по времени алгоритмов умножения матриц на нечётных размерах матриц

4.4 Вывод

В результате эксперимента было получено, что при больших размерах матриц (свыше 10), алгоритм Винограда быстрее стандартного алгоритма более, чем 1.1 раза, а оптимизированный алгоритм Винограда быстрее стандартного алгоритма в 1.5 раза. В итоге, можно сказать, что при таких данных следует использовать оптимизированный алгоритм Винограда.

Также при проведении эксперимента было выявлено, что на чётных размерах реализация алгоритма Винограда в 1.1 раза быстрее, чем на нечётных размерах матриц, что обусловлено необходимостью проводить дополнительные вычисления для крайних строк и столбцов. Следовательно, стоит использовать алгоритм Винограда для матриц, которые имеют чётные размеры.

Заключение

В данной лабораторной работе были рассмотрены алгоритмы умножения матриц. В программировании, как и в математике, часто приходится прибегать к использованию матриц. Существует огромное количество областей их применения в этих сферах. Матрицы активно используются при выводе различных формул в физике, таких, как:

- градиент;
- дивергенция;
- ротор.

Нельзя обойти стороной и различные операции над матрицами – сложение, возведение в степень, умножение. При различных задачах размеры матрицы могут достигать больших значений, поэтому оптимизация операций работы над матрицами является важной задачей в программировании. В данной лабораторной работе пойдёт речь об оптимизациях операции умножения матриц.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи:

1) изучены алгоритмы умножения матриц;
2) создано ПО, реализующее следующие алгоритмы:
    - классический алгоритм умножения матриц;
    - алгоритм Винограда;
    - оптимизированный алгоритм Винограда.
3) Оценено трудоемкости сортировок;
4) Проведено анализ затрат работы программы по времени, выяснить влияющие на них характеристики.
5) Проведено сравнительный анализ между алгоритмами.

Список использованных источников

[1] Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. — М.: Гуманит. изд. центр ВЛАДОС, 2003. с. 45–49.

[2] Головашкин Д. Л. Векторные алгоритмы вычислительной линейной алгебры: учеб. пособие. – Самара: Изд-во Самарского университета, 2019. С. 28–35.

[3] Документация по Microsoft C++ [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017` (дата обращения: 25.09.2022).

[4] C library function clock() [Электронный ресурс]. Режим доступа: `https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm` (дата обращения: 20.10.2022).

[5] Intel [Электронный ресурс]. Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/201839/intel-core-i510300h-processor-8m-cache-up-to-4-50-ghz.html` (дата обращения: 20.10.2022).

[6] Windows 11 Pro 2H21 64-bit [Электронный ресурс]. Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows11` (дата обращения: 20.10.2022).