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 ИУ, Информатика и системы управления

КАФЕДРА ИУ7, Программное обеспечение ЭВМ и информационные технологии

ЛАБОРАТОРНАЯ РАБОТА №2

ПО ДИСЦИПЛИНЕ

“Анализ алгоритмов”

Студент ИУ7-54Б _____ **А.А. Андреев**
(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель _____ **Л.Л. Волкова**
(Подпись, дата) (И.О.Фамилия)

2021 г.

Оглавление

Введение.

Данная лабораторная работа посвящена исследованию и сравнению алгоритмов умножения матриц.

Матрица [1] — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задает размер матрицы.

Алгоритмы умножения матриц широко применяются в задачах, использующих линейную алгебру, компьютерной графики, физики и экономики.

Цель данной лабораторной работы: исследование и сравнение трех алгоритмов умножения матриц: стандартного алгоритма умножения матриц, алгоритма Винограда и модифицированный алгоритм Винограда.

Задачи данной лабораторной работы:

1. Изучение и реализация трех алгоритмов умножения матриц: стандартного алгоритма умножения матриц, алгоритма Винограда и модифицированный алгоритм Винограда;
2. Проведение сравнительного анализа трудоемкости алгоритмов;
3. Проведение сравнительного анализа алгоритмов на основе экспериментальных данных;

1 Аналитическая часть

В данном разделе будут два алгоритма умножения матриц: стандартного алгоритма умножения матриц, алгоритма Винограда.

Определим две матрицы: матрица A размером m x n и матрица B размером m x p, а их произведение, как C = A*B.

1.1 Стандартный алгоритм

Для определенных матриц A и B каждый элемент c_{ij} полученной матрицы произведения C будет определяться по Формуле 1.

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$$

Формула 1: Формула определения элемента по стандартному алгоритму

1.2 Алгоритм Винограда [2]

Рассмотрим два вектора:

$$V = (v1, v2, v3, v4) \text{ и } W = (w1, w2, w3, w4).$$

Их скалярное произведение равно:

$$V \bullet W = v1w1 + v2w2 + v3w3 + v4w4$$

Это равенство можно переписать в виде:

$$V \bullet W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4.$$

Менее очевидно, что выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй.

Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.3 Вывод

В аналитической части были описаны два алгоритма умножения матриц: стандартный алгоритм умножения матриц и алгоритм Винограда. Второй отличается от первого наличием предварительной обработки, меньшим количеством операций умножения.

2 Конструкторская часть.

В данном разделе будет приведены блок-схемы алгоритмов, описанных в аналитическом разделе п.1. и трудоемкость вычислений алгоритмов.

2.1 Схемы алгоритмов

Схемы алгоритмов пузырьком, выбором, вставками доступны на рисунках 1-3.

2.1.1. Стандартный алгоритм

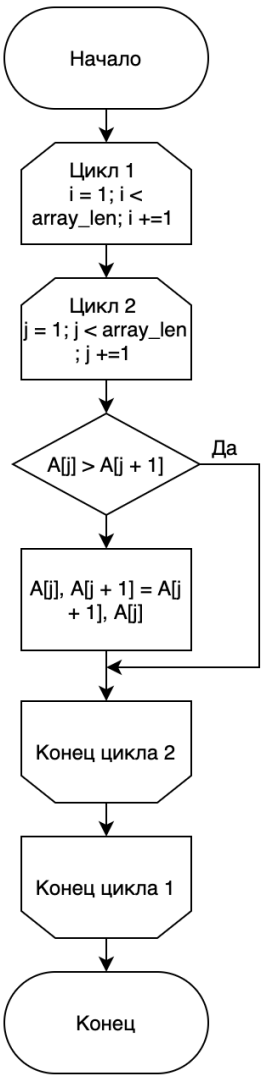

Рисунок 1: Схема стандартного алгоритма умножения матриц

2.1.2. Алгоритм Винограда

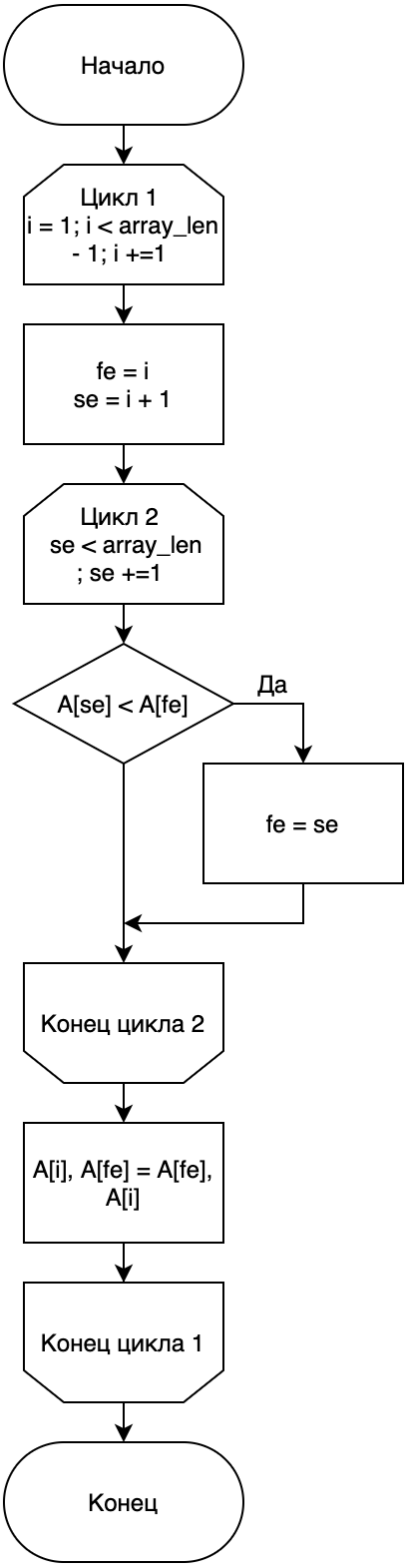

Рисунок 2: Схема алгоритма Винограда умножения матриц

2.1.3. Оптимизированный алгоритм Винограда

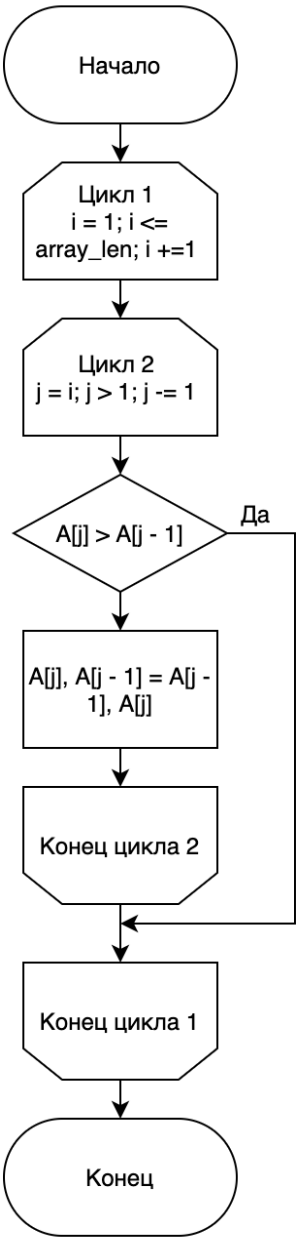

Рисунок 3: Схема оптимизированного алгоритма Винограда умножения матриц

2.2 Модель вычислений

Введенная модель вычислений трудоемкости доступна в Таблице 1.

Таблица 1: Введенная модель вычислений

№	Операции	Трудоемкость
1	$+, -, /, \%, ==, !=, <, >, <=, >=, [], ++, --$	1
2	Условие	$f_{if} = f_{\text{услов.}} + f_{\text{выполняемая операция}}$
3	Цикла	$f_{for} = f_{\text{иниц.}} + f_{\text{сравн.}} + N(f_{\text{тела}} + f_{\text{инк.}}$
4	Вызов функции	0

2.3 Трудоемкость алгоритмов

Обозначим размер массива N, а сам массив A.

2.3.1 Стандартный алгоритм

На основе вычислений трудоемкости составляющих стандартного алгоритма умножения матриц в Таблице 2 сделано суммирование общей трудоемкости в Таблице 3.

Таблица 2: Составляющие трудоемкости стандартного алгоритма умножения матриц

№	Операции	Трудоемкость
1	Сравнение, увеличение внешнего цикла *от 1 до N*	$2 + 2 \cdot (N - 1)$
2	Итерации внутренних циклов *от 1 до N - 1*	$3 \cdot (N - 1) + 0.5 \cdot N \cdot (N - 1) \cdot (3 + f_{y}$
3	Внутреннее условие	В лучшем: $4 + 0$ В худшем: $4 + 9$

Таблица 3: Трудоемкость стандартного алгоритма умножения матриц

Случай	Трудоемкость
Лучший и худший случай	$10 \cdot M \cdot N \cdot Q + 4 \cdot M \cdot Q + 4 \cdot M + 2$

2.3.2 Алгоритм Винограда

На основе вычислений трудоемкости составляющих алгоритма умножения матриц Винограда в Таблице 4 сделано суммирование общей трудоемкости в Таблице 5.

Таблица 4: Составляющие трудоемкости алгоритма умножения матриц Винограда

№	Операции	Трудоемкость
1	Первые два цикла заполнения	$2 \cdot (15/2 \cdot M \cdot N + 5 \cdot M + 2)$
2	Третий цикл	$13 \cdot M \cdot N \cdot Q + 12 \cdot M \cdot Q + 4 \cdot M + 2$
3	Внутреннее условие	В лучшем (невыполнения): 2 В худшем: $15 \cdot Q \cdot M + 4 \cdot M + 2$

Таблица 5: Трудоемкость алгоритма умножения матриц Винограда

Случай	Трудоемкость
Лучший	$15 \cdot M \cdot N + 14 \cdot M + 8 + 13 \cdot M \cdot N \cdot Q + 12 \cdot M \cdot Q$
Худший	$15 \cdot M \cdot N + 18 \cdot M + 8 + 13 \cdot M \cdot N \cdot Q + 12 \cdot M \cdot Q$

2.3.3 Оптимизированный алгоритм Винограда

На основе вычислений трудоемкости составляющих алгоритма сортировки вставками в Таблице 6 сделано суммирование общей трудоемкости в Таблице 7.

Таблица 6: Составляющие трудоемкости оптимизированного алгоритма умножения матриц Винограда

№	Операции	Трудоемкость
1	Первые два цикла заполнения	$2 \cdot (11/2 \cdot M \cdot N + 4 \cdot M + 2)$
2	Третий цикл	$17/2 \cdot M \cdot N \cdot Q + 9 \cdot M \cdot Q + 4 \cdot M + 2$
3	Внутреннее условие	В лучшем (невыполнения):1 В худшем: $10 \cdot Q \cdot M + 4 \cdot M + 2$

Таблица 7: Трудоемкость оптимизированного алгоритма умножения матриц Винограда

Случай	Трудоемкость
Лучший	$11 \cdot M \cdot N + 12 \cdot M + 7 + 17/2 \cdot M \cdot N \cdot Q + 9 \cdot M \cdot Q$
Худший	$11 \cdot M \cdot N + 16 \cdot M + 8 + 17/2 \cdot M \cdot N \cdot Q + 19 \cdot M \cdot Q$

2.4 Вывод

Блок-схемы и анализ трудоемкости алгоритмов умножения матриц в данном разделе позволяют перейти к технологической части - непосредственно к программной реализации решения.

3 Технологическая часть.

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать поставленную на лабораторную работу задачу. Интерфейс для взаимодействия с программой - командная строка. Программа должна сравнивать работы трех алгоритмов умножения матриц: стандартного, Винограда и оптимизированного Винограда, показывать потраченное на это время.

3.2 Выбор и обоснование языка и среды программирования.

Для разработки данной программы применён язык Python 3 с библиотекой time.clock() [4] для вычисления времени работы процессора, чтобы расширить знания в области данного языка программирования.

3.3 Реализация алгоритмов

В листингах 1-9 приведена реализация алгоритмов умножения матриц.

Программа была реализована в парадигме ООП [2], где в базовый класс был вынесен объект MatrixMultiplication (Листинг 1-2), внутри него с доступом protected, используемые алгоритмами функции инициализации результирующей матрицы *_generate_result_matrix(self)*
, установка объекта результата умножения матриц *_set_result_matrix(self, result_matrix)*
и функции работы со временем *_set_start_processor_time(self)*, *_set_end_processor_time(self), get_result_processor_time(self)*.

Наследуемые объекты умножения матрицы стандартным методом (Листинг 3), Винограда (Листинг 4-6), оптимизированным Виноградом (Листинг 7-9) имеют публичную функцию получения произведения матриц *multiply()*;

Листинг 1: Базовый класс MatrixMultiplication, Часть 1

```
1. # Базовый объект Умножения матрицы
2. class MatrixMultiplication:
3.     # Приватные рабочие поля
4.     _first_matrix = None
5.     _second_matrix = None
6.     _result_matrix = None
7.
8.     # Приватные поля сбора аналитики
9.     _processor_time = None
10.
```

Листинг 2: Базовый класс MatrixMultiplication, Часть 2

```
11.           # Создание объекта матрицы
12.       def __init__(self, first_matrix, second_matrix):
13.           # Установка введенных матриц
14.           self._first_matrix = first_matrix
15.           self._second_matrix = second_matrix
16.
17.           # Установка полей сбора аналитики
18.           self._processor_time = analitics.ProcessorTime()
19.
20.           # Конфигурационная настройка результирующей матрицы
21.           self._result_matrix = self._generate_result_matrix()
22.
23.       # Получение умноженной матрицы
24.       def get_multiplied_matrix(self):
25.           return self._result_matrix
26.
27.       # Общая функция инициализации результирующей матрицы
28.       def _generate_result_matrix(self):
29.           return matrix.Matrix(
30.               self._first_matrix.get_size_of_matrix(),
31.               [[0 for _ in
   range(self._first_matrix.get_size_of_matrix())]
32.                for _ in
   range(self._first_matrix.get_size_of_matrix())]
33.           )
34.
35.       # Установка объекта результата умножения матрицы
36.       def _set_result_matrix(self, result_matrix):
37.           self._result_matrix = result_matrix
38.
39.       # Установка объекта начала отсчета процессорного времени
40.       def _set_start_processor_time(self):
41.           return self._processor_time.set_start_processor_time()
42.
43.       # Установка объекта конца отсчета процессорного времени
44.       def _set_end_processor_time(self):
45.           return self._processor_time.set_end_processor_time()
46.
47.       # Получение результата вычисления процессорного времени
48.       def get_result_processor_time(self):
49.           return self._processor_time.get_result()
```

Листинг 3: Наследуемый объект стандартного алгоритма умножения матриц ClassicalMultiplication

```
1. # Объект классического умножения матрицы
2. class ClassicalMultiplication(MatrixMultiplication):
3.
4.     def multiply(self):
5.
6.         # Установка отсечки времени
7.         self._set_start_processor_time()
8.
9.         # Виртуальная результирующая матрица
10.            result_matrix = self._generate_result_matrix()
11.
12.            # Непосредственная операция умножения матрицы
13.            for check_row in
    range(result_matrix.get_size_of_matrix()):
14.                for check_column in
    range(result_matrix.get_size_of_matrix()):
15.                    for check_par in
    range(result_matrix.get_size_of_matrix()):
16.
    result_matrix.update_matrix_value(check_row, check_column,
17.
    result_matrix.get_matrix_value(check_row, check_column) +
18.
    self._first_matrix.get_matrix_value(check_row, check_par) *
19.
    self._second_matrix.get_matrix_value(check_par, check_column))
20.
21.            # Установка отсечки времени
22.            self._set_end_processor_time()
23.
24.            self._set_result_matrix(result_matrix)
```

Листинг 4: Наследуемый объект алгоритма умножения Винограда, Часть 1

```
1. # Объект умножения матриц Копперсмита-Винограда
2. class CoppersmittWinogradMultiplication(MatrixMultiplication):
3.
4.     def multiply(self):
5.
6.         # Установка отсечки времени
7.         self._set_start_processor_time()
8.
9.         # Необходимые для вычислений данные
10.            _first_matrix_size, _second_matrix_size =
    self._first_matrix.get_size_of_matrix(), \
11.
    self._second_matrix.get_size_of_matrix()
12.
13.            # Виртуальная результирующая матрица
14.            result_matrix = self._generate_result_matrix()
```

Листинг 5: Наследуемый объект алгоритма умножения Винограда, Часть 2

```
15.
16.             _mul_h, _mul_v = [0 for _ in
    range(self._first_matrix.get_size_of_matrix())], \
17.                              [0 for _ in
    range(self._second_matrix.get_size_of_matrix())]
18.
19.             for check_row in
    range(self._first_matrix.get_size_of_matrix()):
20.                 for check_column in
    range(self._first_matrix.get_size_of_matrix() // 2):
21.                     _mul_h[check_row] +=
    self._first_matrix.get_matrix()[check_row][2 * check_column] *
    \
22.
    self._first_matrix.get_matrix()[check_row][2 * check_column +
    1]
23.
24.             for check_row in
    range(self._second_matrix.get_size_of_matrix()):
25.                 for check_column in
    range(self._second_matrix.get_size_of_matrix() // 2):
26.                     _mul_v[check_row] +=
    self._second_matrix.get_matrix()[check_row][2 * check_column] *
    \
27.
    self._second_matrix.get_matrix()[check_row][2 * check_column +
    1]
28.
29.             for check_row in
    range(self._first_matrix.get_size_of_matrix()):
30.                 for check_column in
    range(self._first_matrix.get_size_of_matrix()):
31.                     result_matrix.update_matrix_value(check_row,
    check_column,
32.                                                       -
    _mul_h[check_row] - _mul_v[check_column]
33.                                                       )
34.
35.                     for k in
    range(self._first_matrix.get_size_of_matrix() // 2):
36.
    result_matrix.update_matrix_value(check_row, check_column,
37.
    result_matrix.get_matrix_value(check_row, check_column) +
38.
    self._first_matrix.get_matrix()[check_row][2 * k] +
39.
    self._second_matrix.get_matrix()[2 * k + 1][
40.
    check_column] * \
```

Листинг 6: Наследуемый объект алгоритма умножения Винограда, Часть 3

```
41.   (
42.
   self._first_matrix.get_matrix()[check_row][
43.
   2 * k + 1] +
44.
   self._second_matrix.get_matrix()[2 * k][
45.
   check_column]
46.                                                                 ))
47.
48.           if self._second_matrix.get_size_of_matrix() % 2:
49.               for check_row in
   range(self._first_matrix.get_size_of_matrix()):
50.                   for check_column in
   range(self._first_matrix.get_size_of_matrix()):
51.
   result_matrix.update_matrix_value(check_row, check_column,
52.
   result_matrix.get_matrix_value(check_row, check_column) +
53.
   self._first_matrix.get_matrix()[check_row][
54.
   self._first_matrix.get_size_of_matrix() - 1] *
55.
   self._second_matrix.get_matrix()[
56.
   self._second_matrix.get_size_of_matrix() - 1][check_column]
57.                                                                 )
58.
59.           # Установка отсечки времени
60.           self._set_end_processor_time()
61.
62.           self._set_result_matrix(result_matrix)
```

Листинг 7: Наследуемый объект оптимизированного алгоритма умножения Винограда, Часть 1

```
1. # Объект умножения матриц оптимизированный
   Копперсмитта-Винограда
2. class
   CoppersmittWinogradOptimizedMultiplication(MatrixMultiplication
   ):
3.
4.     def multiply(self):
5.
6.         # Установка отсечки времени
7.         self._set_start_processor_time()
```

Листинг 8: Наследуемый объект оптимизированного алгоритма умножения Винограда, Часть 2

```
8.  # Необходимые для вычислений данные
9.          _first_matrix_size, _second_matrix_size =
    self._first_matrix.get_size_of_matrix(), \
10.
    self._second_matrix.get_size_of_matrix()
11.
12.         # Виртуальная результирующая матрица
13.         result_matrix = self._generate_result_matrix()
14.
15.         _mul_h, _mul_v = [0 for _ in
    range(self._first_matrix.get_size_of_matrix())], \
16.                          [0 for _ in
    range(self._second_matrix.get_size_of_matrix())]
17.
18.         for check_row in
    range(self._first_matrix.get_size_of_matrix()):
19.             _mul_h[check_row] = sum(
20.                 self._first_matrix.get_matrix()[check_row][2 *
    check_column] *
21.                 self._first_matrix.get_matrix()[check_row][2 *
    check_column + 1]
22.                 for check_column in
    range(self._first_matrix.get_size_of_matrix() // 2)
23.             )
24.
25.             _mul_v[check_row] = sum(
26.                 self._second_matrix.get_matrix()[check_row][2
    * check_column] *
27.                 self._second_matrix.get_matrix()[check_row][2
    * check_column + 1]
28.                 for check_column in
    range(self._second_matrix.get_size_of_matrix() // 2)
29.             )
30.
31.         for check_row in
    range(self._first_matrix.get_size_of_matrix()):
32.             for check_column in
    range(self._first_matrix.get_size_of_matrix()):
33.                 result_matrix.update_matrix_value(check_row,
    check_column,
34.                                                   sum(
35.
    (self._first_matrix.get_matrix()[check_row][2 * k] +
36.
    self._second_matrix.get_matrix()[2 * k + 1][
37.
    check_column]) * (
38.
    self._first_matrix.get_matrix()[check_row][
39.
    2 * k + 1] +
```

Листинг 9: Наследуемый объект оптимизированного алгоритма умножения Винограда, Часть 3

```
40.   self._second_matrix.get_matrix()[2 * k][
41.
   check_column])
42.                                                    for k in
   range(self._first_matrix.get_size_of_matrix() // 2)) \
43.                                                    -
   _mul_h[check_row] - _mul_v[check_column]
44.                                                    )
45.
46.           # Установка отсечки времени
47.           self._set_end_processor_time()
48.
49.           self._set_result_matrix(result_matrix)
50.
```

3.4 Тестовые данные

Тестовые данные, на которых было протестировано разработанное программное обеспечение, представлено в Таблице 1.

Таблица 8: Тестовые данные

№	Входной массив	Результат	Ожидаемый результат
1	1, 2, 3, 4, 5, 6, 7, 8	1, 2, 3, 4, 5, 6, 7, 8	1, 2, 3, 4, 5, 6, 7, 8
2	8, 7, 6, 5, 4, 3, 2, 1	8, 7, 6, 5, 4, 3, 2, 1	8, 7, 6, 5, 4, 3, 2, 1
3	0, 0, 0, 0, 0	0, 0, 0, 0, 0	0, 0, 0, 0, 0
4	Пустой массив	Пустой массив	Пустой массив
5	1	1	1
6	1, 990, 12, 345, -95	-95, 1, 12, 345, 990	-95, 1, 12, 345, 990

3.5 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, приведены результаты работы программы на тестовых данных.

4 Исследовательская часть.

4.1. Демонстрация работы программы

Пример работы программы представлен на рисунке 6.

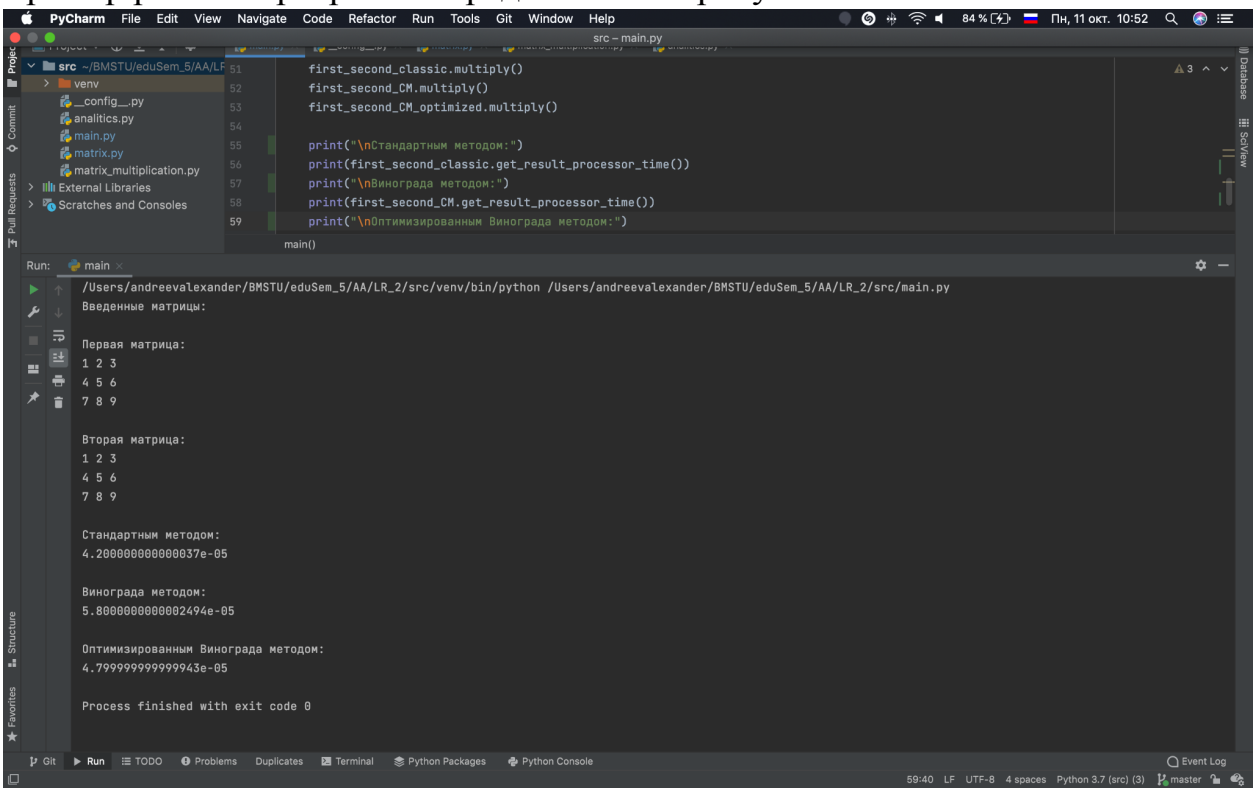

Рисунок 4: Демонстрация работы программы

4.2. Технические характеристики

В Таблице 3. приведены технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения.

Таблица 9: Технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения

ОС	Mac OS Mojave 64-bit
ОЗУ	8 Gb 2133 MHz LPDDR3
Процессор	2,3 GHz Intel Core i5

4.3. Время выполнения алгоритмов

В Таблице 10. приведена информация о времени выполнения алгоритмов на отсортированных данных в микросекундах, в Таблице 11 на отсортированных данных в обратном порядке, в Таблице 12 на случайных числах.

Таблица 10: Таблица времени выполнения алгоритмов на отсортированных данных (в микросекундах)

№	Длина строк	Время		
		Пузырек	Выбором	Вставками
1	10	223	21	11
2	20	605	22	12
3	1000	19 300	30	23
4	1600	45 400	20	14
5	2000	65 910	13	91

Таблица 11: Таблица времени выполнения алгоритмов на отсортированных данных (в микросекундах)

№	Длина строк	Время		
		Пузырек	Выбором	Вставками
1	10	238	210	199
2	20	829	801	793
3	1000	24 642	23 998	23 638
4	1600	77 411	75 630	69 779
5	2000	89 193	87 904	83 786

Таблица 12: Таблица времени выполнения алгоритмов на отсортированных данных (в микросекундах)

№	Длина строк	Время		
		Пузырек	Выбором	Вставками
1	10	193	125	112
2	20	675	239	217
3	1000	19 389	13 001	12 390
4	1600	47 611	18 992	18 429
5	2000	67 560	22 305	21 129

4.5. Вывод

Лучше всего себя показывает сортировка вставками, она делает это стабильно на всех трех видах последовательности, затрачивая примерно одинаковое время на одну длину последовательности. Когда последовательность упорядочена сортировка вставками работает в 190 раз быстрее сортировки пузырьком, сортировка выбором и вставками выдает примерно равные показатели.

Заключение.

В рамках данной лабораторной работы были изучены и реализованы трех нерекурсивные алгоритмы сортировки: пузырек, выбором, вставками, проведен сравнительный анализ трудоемкости алгоритмов, сделан сравнительный анализ алгоритмов на основе экспериментальных данных.

Экспериментальным путем было выявлено, что лучше всего себя показывает сортировка вставками, она делает это стабильно на всех трех видах последовательности, затрачивая примерно одинаковое время на одну длину последовательности. Когда последовательность упорядочена сортировка вставками работает в 190 раз быстрее сортировки пузырьком, сортировка выбором и вставками выдает примерно равные показатели.

Список использованной литературы

[1] Кормен Т.Х., Лейзерсон Ч.И., Алгоритмы: Построение и анализ, год выпуска 2019, тираж 1328, 700 страниц.

[2] Наследование в Python [Электронный ресурс] Режим доступа: https://younglinux.info/oopython/inheritance. Дата обращения: 13.09.2021

[3] Гасфилд, Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, год выпуска 2003, типаж 900, 653 страницы.

[4] Вычисление процессорного времени выполенения программы [Электронный ресурс] Режим доступа: https://www.tutorialspoint.com/python/time_clock.htm. Дата обращения: 13.09.2021