Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Факультет вычислительной математики и кибернетики

Отчёт по лабораторной работе

Матрицы и векторы

Выполнил:

Назаров Н.С.

Проверил:

Нижний Новгород

2020 г.

Содержание

[Введение 3](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962758)

[Постановка задачи 4](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962759)

[Руководство пользователя 5](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962760)

[Руководство программиста 6](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962761)

[Описание структур данных 6](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962762)

[Описание алгоритмов 6](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962763)

[Описание структуры программы 6](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962764)

[Заключение 7](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962765)

[Литература 8](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962766)

[Приложения 9](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962767)

[Приложение 1 9](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962768)

[Приложение 2 9](file:///D:\загрузки\Шаблон%20отчёта%20по%20Лабораторной%20работе%20(2).doc#_Toc270962769)

# Введение

Матричная алгебра - раздел алгебры, посвященный правилам действий над матрицами - одним из самых важных, употребительных и содержательных понятий в математике.

Матрицы проникли почти во все отрасли человеческой деятельности. В математике они используются при исследовании систем m линейных уравнений с n неизвестными. В экономике - при отражении соотношений затрат, производственных и экономических структур. В технике - при расчете сооружений. В физике матрицы применены для повышения точности вычисления значений полей вблизи неоднородности, теории управления, статистики, других областей науки и знаний.

Практика - это критерий истинности знаний, и моя работа покажет, что сложные и непонятные с первого взгляда матрицы, определители и их свойства могут быть применимы в различных отраслях деятельности человека и в обычной жизни.

Объект исследования: применения матриц на практике, в экономике, математике и других науках.

Предмет исследования: матрица.

Цель исследования: выявить принципы применения матриц в различных областях науки.

Задачи исследования:

* - научиться выполнять действия над матрицами.
* - правильно составить математическую модель ситуации.
* - решить полученную матрицу.
* - выбрать правильный ответ.

Практическая ценность: я смогу применять свои знания в старших классах и ВУЗе при решении линейных систем уравнений. Помогать своим сверстникам, если у них возникнут затруднения в решении уравнений.

# Постановка задачи

|  |
| --- |
|  |
|  | Написать классы для работы с векторами и матрицами использовать шаблоны. |
|  | Вектора в математическом понимании: имеется набор значений из N мерного пространства, размерность задается как параметр. |
|  | Продемонстрировать их работу на примере (написать в main пример). |
|  | Должны быть: |
|  | конструкторы (по умолчанию, инициализатор, копирования), деструктор, доступ к защищенным полям; |
|  | перегруженные операции: +,-,\*,/,=,==, [] потоковый ввод и вывод; |
|  | перегруженные операции +,-,\*,/ должны быть реализованы для векторов (вектор +-\*/ вектор), матриц (матрица +-\* матрица), матрично-векторные (матрица \* вектор и наоборот); |
|  | в классе вектор должна быть возможность отсортировать его тремя способами (пузырек, вставка, быстрая сортировки, см. вторая лабораторная первого семестра). |

# Руководство пользователя

1. Создать переменную типа Matrix и Vector
2. Используя все доступные поля и методы, а также доступ к закрытым полям, написать программу для работы с классами
3. Получить результат

# Руководство программиста

## Описание структуры программы

Пять заголовочных файлов: Matrix.h, Matrix.cpp, Vector.h, Vector.cpp, main.cpp

Первый и третий для создания прототипов функций, второй и четвертый для описаний этих функций, пятый для работы со всеми данными нашей программы

## Описание структур данных

Шаблонные классы:

Class Matrix

Class Vector

Закрытые поля класса Matrix:

* Type\*\* m – шаблонный двойной указатель.
* int rows – целочисленный размер ширины матрицы
* int cols – целочисленный размер длины матрицы

Открытые поля и методы класса Matrix:

* Matrix() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все три поля 0, через списки инициализации.
* Matrix(int rows, int cols) – конструктор инциализатор, принимает на вход два параметра типа int, создает динамический массив заданных размеров, заполняет его 0, поля rows и cols, становятся значениями переданными в конструктор соответственно.
* Matrix(const Matrix& a) – конструктор копирования, принимает на вход объект типа Matrix, создает объект с теми же характеристиками что и переданный, и матрицей того же содержимого.
* ~Matrix() – деструктор, очищает выделенную методами и конструкторами память.
* Int GetRows() – метод возвращающий ширину матрицы
* Int GetCols() – метод возвращающий длину матрицы
* Type\*\* GetMAtrix() – метод возвращающий массив содержащий в себе все элементы матрицы.
* Void SetMatrix(int rows, int cols) – метод принимающий новые размеры матрицы, приводящий матрицу к этим размерам. В случае если новая матрица больше старой, новые элементы равняются нулю. В противном случае, все элементы на которые хватило места, останутся в матрице. В случае если матрица имеет любую размерность равную 0, метод ведет себя аналогично конструктору инциализатору
* Type\*& operator[ ](int index ) - перегрузка оператора индексации, позволяет возвращать элемент матрицы по индексу.
* Matrix<Type> operator+(const Matrix<Type>& a) – перегрузка оператора суммы, позволяет возвращать матрицу, являющуюся суммой двух матриц.
* Matrix<Type>operator–(const Matrix<Type>& a) – перегрузка оператора –, аналогична в своей сути перегрузке оператора +
* Matrix& operator =(const Matrix & a) - перегрузка оператора присвоить, позволяет присваивать один объект типа Matrix другому
* bool operator ==(const Matrix<Type>& a) – перегрузка оператора сравнения, позволяет сравнивать объекты типа Matrix
* Matrix<Type>operator\*(const Matrix<Type>& a) - перегрузка оператора умножить, позволяет перемножать обьекты типа Matrix
* friend ostream& operator <<(ostream& os, const Matrix<Type>& a) – перегрузка оператора вывода, позволяет выводить матрицы на экран
* friend istream& operator >>(istream& is, Matrix<Type>& a) - перегрузка оператора ввода, позволяет вводить матрицы уже заданного размера, либо задавать размер и вводить самостоятельно.

Шаблонный класс Vector, является public, наследником класса Matrix

Открытые поля и методы класса Vector

* Type21\* GetVector() - метод возвращающий массив содержащий в себе все элементы объекта Vector.
* Type21& operator [](const int index) - перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу.
* void SetCols2(int rows3, int cols3) - метод принимающий новые размеры вектора, приводящий вектор к этим размерам. В случае если новый вектор больше старого, новые элементы равняются нулю. В противном случае, все элементы на которые хватило места, останутся в векторе. В случае если вектор имеет размерность равную 0, метод ведет себя аналогично конструктору инциализатору
* Vector<Type21> operator/(const Vector<Type21>& b) – перегрузка оператора / возвращает вектор, координаты которого являются частным от деления двух соответствующих координат операндов. В случае, если делитель равен нулю, значение частного считается равным делимому.
* Type21 operator\*(const Vector<Type21>& b) – перегрузка оператора умножить, возвращает скалярное произведение операндов.
* friend istream& operator >>(istream& is, Vector<Type34>& b) - перегрузка оператора ввода, позволяет вводить матрицы уже заданного размера, либо задавать размер и вводить самостоятельно.
* clock\_t BubbleSort() - сортировка методом «Пузырек». Возвращает разницу между тактами процессора при входе в сортировку и при выходе из нее.
* clock\_t InsertionSort() - сортировка методом «Вставка». Возвращает разницу между тактами процессора при входе в сортировку и при выходе из нее.

## Описание алгоритмов

1. Сортировка BubbleSort

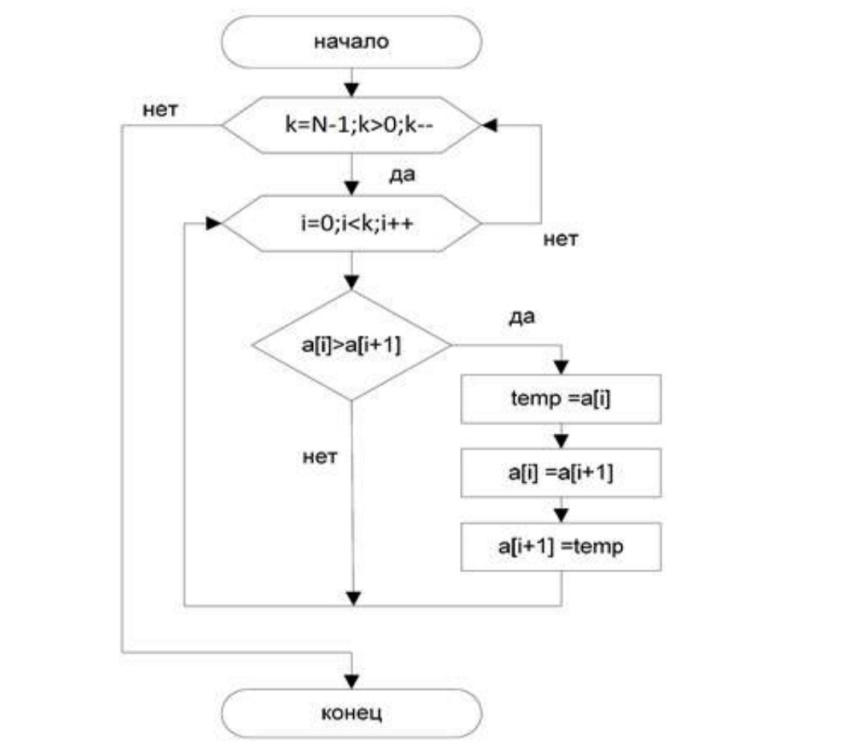


Рисунок 1. Блок-схема алгоритма сортировки пузырьком.

1. Сортировка InsertionSort

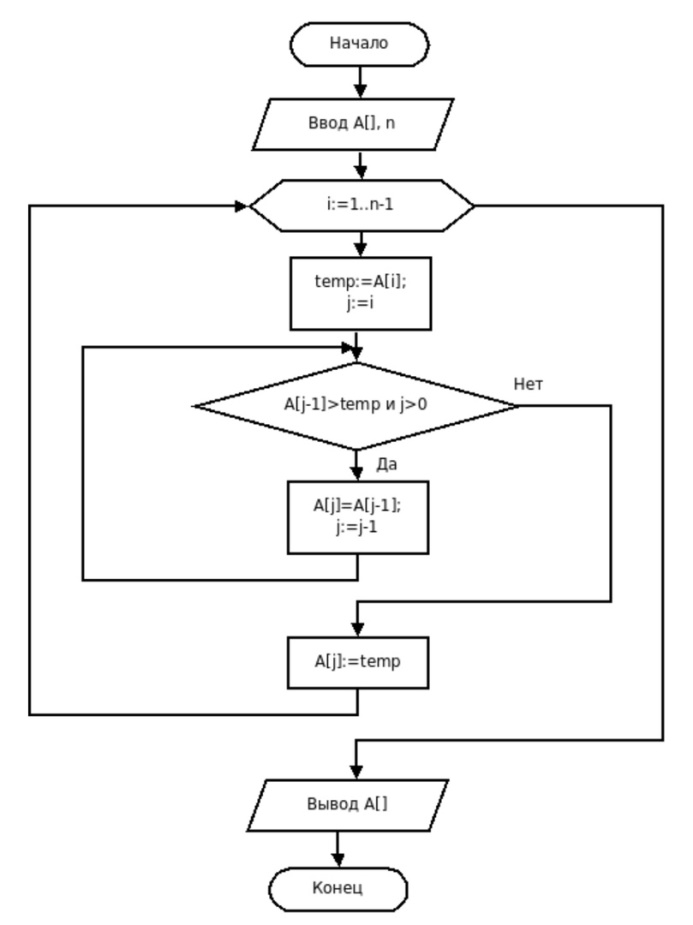


Рисунок 2. Блок-схема алгоритма сортировки вставками.

1. Сортировка QuickSort

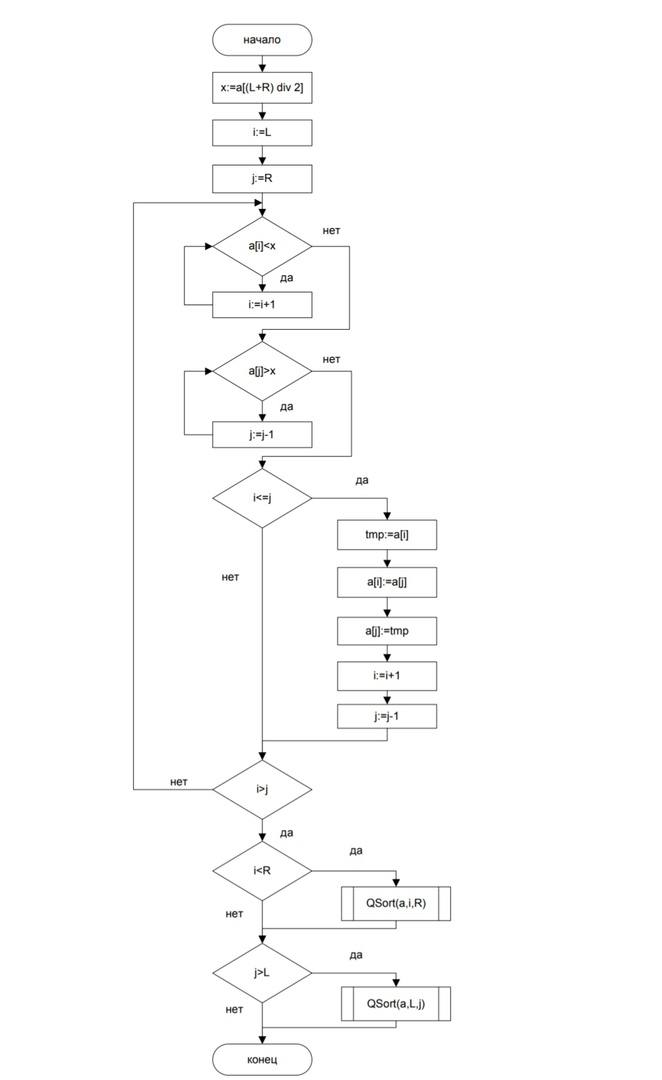


Рисунок 3. Блок-схема алгоритма быстрой сортировки.

# Эксперименты

1.Сумма матриц : template <typename Type>

Matrix<Type> Matrix<Type>:: operator+ (const Matrix<Type>& a) {

if (this->rows = a.rows && this->cols = a.cols) {

Matrix<Type> NewMatrix(int rows, int cols);

for (int i = 0; i < this->rows; i++) {

for (int j = 0; j < this->cols; j++) {

NewMatrix[i][j] = this->m[i][j] + a.m[i][j];

}

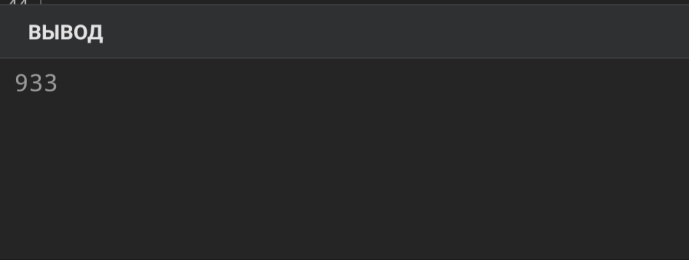
}

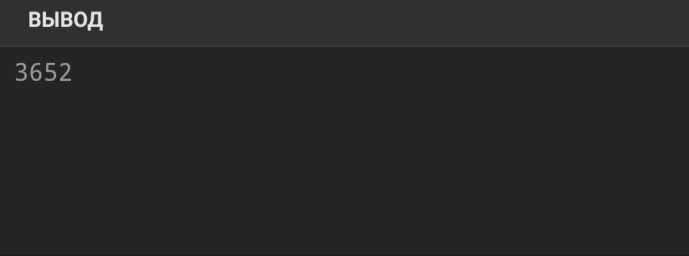
return NewMatrix;

}

else cout << "Error operator +" << endl;

}





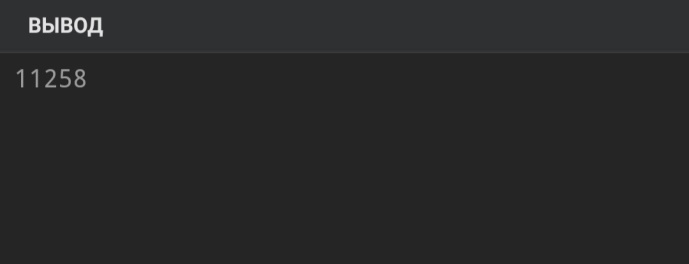


Рис.1 Результаты сложения матриц

Асимптотическая сложность этого кода:



|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (в мс) |
| 500х500 | 933 |
| 1000х1000 | 3652 |
| 1500х1500 | 11258 |

Таблица 1: Время суммирования матриц.

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

2.Умножение матриц: template <typename Type5>

Matrix<Type5> Matrix<Type5>:: operator\* (const Matrix<Type5>& a) {

if (this->rows = a.rows && this->cols = a.cols) {

Matrix<Type5> NewMatrix(rows, cols);

Type5 Sum;

for (int i = 0; i < this->rows; i++) {

for (int j = 0; j < this->cols; j++) {

Sum = 0;

for (int l = 0; l < this->rows; l++) {

Sum = Sum + (this->m[i][l]) \* (a.m[l][j]);

}

NewMatrix[i][j] = Sum;

}

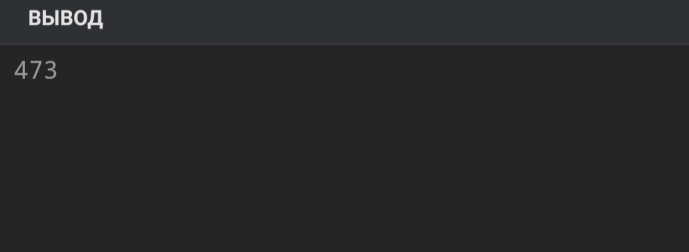
}

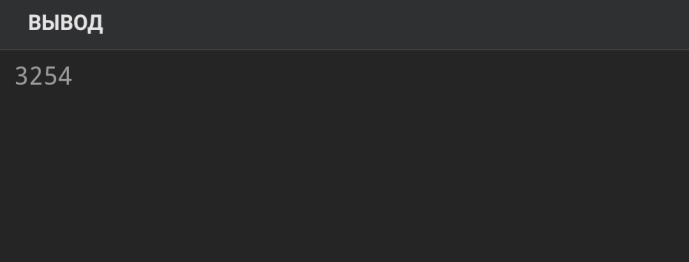
return NewMatrix;

}

else cout << "Error operator \*" << endl;

}





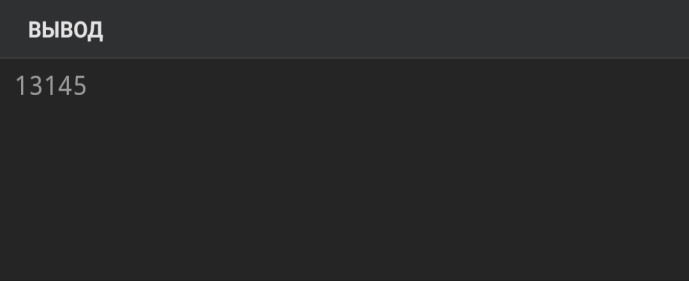


Рис.2 Результаты умножения матриц

Асимптотическая сложность умножения:



|  |  |
| --- | --- |
| Количество элементов в матрице | Время выполнения (в мс) |
| 50x50 | 473 |
| 100x100 | 3254 |
| 150x150 | 13145 |

Таблица 2: Время умножения матриц.

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 8 раза.

3.Умножение вектора и матрицы: template <typename Type36>

Type36 Vector<Type36> :: operator\* (const Vector<Type36>& b) {

if (this->GetCols2() == b.GetCols2()) {

Type36 Vec = 0;

for (int i = 0; i < b.GetCols2(); i++)

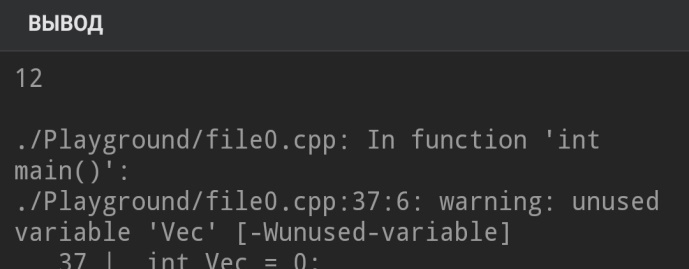
Vec = Vec + this->GetVector()[i] \* b.GetVector[i];

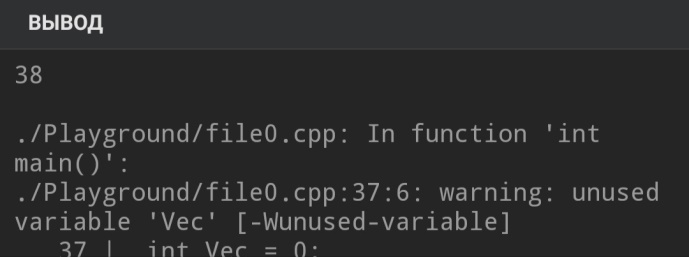
return Vec;

}

else cout << "Error operator \* vector" << endl;

}





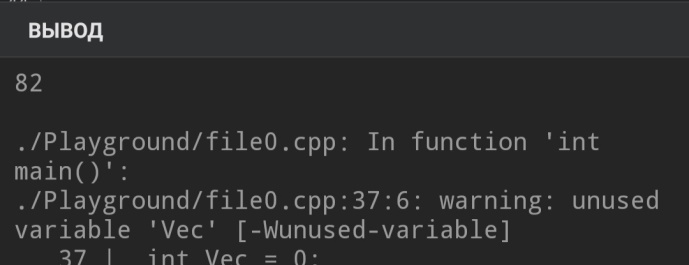


Рис.3 Результаты умножения матрицы на вектор

Асимптотическая сложность этого кода :

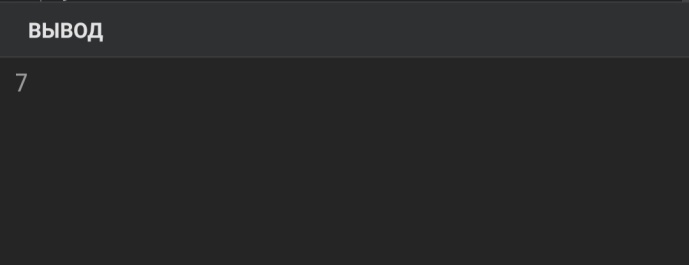


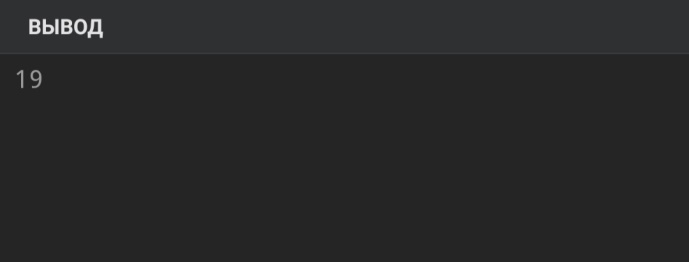
|  |  |
| --- | --- |
| Количество элементов в векторе | Время выполнения (в мс) |
| 50 | 12 |
| 100 | 38 |
| 150 | 82 |

Таблица 3: Время умножения матриц и векторов

Как мы можем видеть действительно, при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.

4.BubbleSort





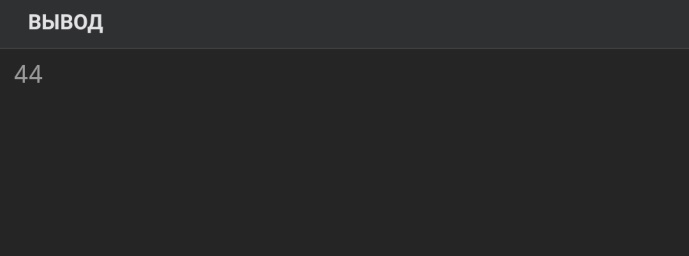


Рис.4 Результаты сортировки BubbleSort

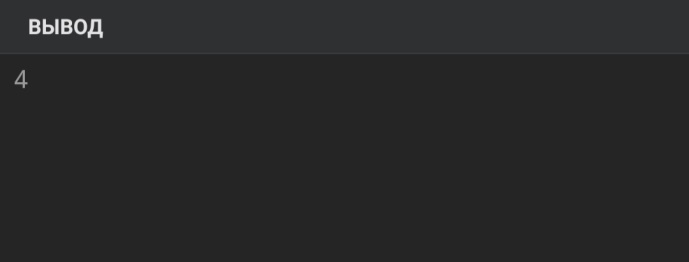
Асимптотическая сложность этого кода :

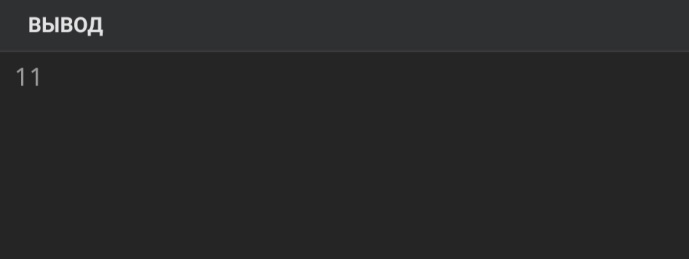


|  |  |
| --- | --- |
| Количество элементов в векторе | Время выполнения (в мс) |
| 50 | 7 |
| 100 | 19 |
| 150 | 44 |

Таблица 4: Время работы сортировки BubbleSort

5.InsertionSort





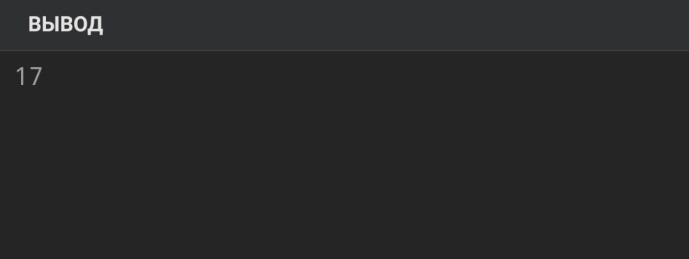


Рис.5 Результаты сортировки InsertionSort

Асимптотическая сложность этого кода :



|  |  |
| --- | --- |
| Количество элементов в векторе | Время выполнения (в мс) |
| 50 | 4 |
| 100 | 11 |
| 150 | 17 |

Таблица 5: Время работы сортировки InsertionSort

Как видно из таблицы, действительно, затрачиваемое на реализацию операции время растет эквивалентно асимптотической функции.

# Заключение

В результате моей работы, мне удалось самостоятельно разобраться в основах темы: «Матрицы и операции над ними», курса Высшей Математики. Прочитав множество источников, просмотрев множество сайтов, мне удалось систематизировать основные данные по этой теме. Очень надеюсь, что моя работа может помочь разобраться другим ученикам.