МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

Кафедра информационных технологий и экономической информатики

ОТЧЕТ

по лабораторной работе №1

Выполнил:

Студент группы ПрИ-202 Приходько Даниил Александрович

Студент группы ПрИ-202 Саламатин Алексей

Студент группы ПрИ-202 Скоробогатов Максим

Принял:

Преподаватель ИИТ Николаев Иван Евгеньевич

Отчет защищен \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата оценка

Челябинск 2024 г.

Содержание

Цель работы

Цель данной лабораторной работы — провести эмпирический анализ временной сложности алгоритмов. В рамках работы вводится понятие временной сложности алгоритма, рассматривается математический аппарат для оценки временной сложности и правила применения этого аппарата.

Задание

Для каждого n от 1 до 2000 произведите для пяти запусков замер среднего машинного времени исполнения программ, реализующих нижеуказанные алгоритмы и функции. Изобразите на графике полученные данные, отражающие зависимость среднего времени исполнения от n. Проведите теоретический анализ временной сложности рассматриваемых алгоритмов и сравните эмпирическую и теоретическую временные сложности.

Теоретическая часть

На деле был произведен уход от начально задания, так как объем входных данных был недостаточен для наглядной оценки временной сложности работы предоставленных алгоритмов. Длина входного вектора была изменена и теперь пользователь выбирает количество подаваемых на вход программе чисел в пределе от 1 до 1.000.000.000, для того чтобы увидеть прогрессию функции на большом размере входного вектора. Помимо этого также и количество запусков выбирается пользователем и варьируется от 1 до 10. Также пользователю была предоставлена возможность самостоятельно выбирать максимальный размер случайного числа, в размере до 1.000.000. Для алгоритмов возведения в степень, есть возможность выбирать степени в пределах от 2 до 200. В алгоритме перемножения матриц можно выбрать размер стороны квадратной матрицы от 1 до 1.000.000.000. А в алгоритме Дейкстры предоставляется возможность указать количество вершин графа также от 1 до 1.000.000.000.

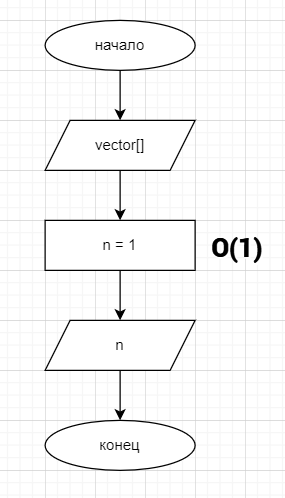
Блок схемы и оценка врменной сложности:

Блок 1

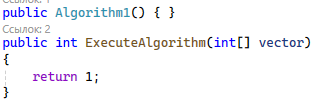
1) Постоянная функция – функция которая независимо от количества и размера входных данных возвращает 1.

Теоретическая сложность O(1) , т.к не зависит от n (рис.1.1)

Код алгоритма представлен на рисунке 1.2



**рис.1.1**

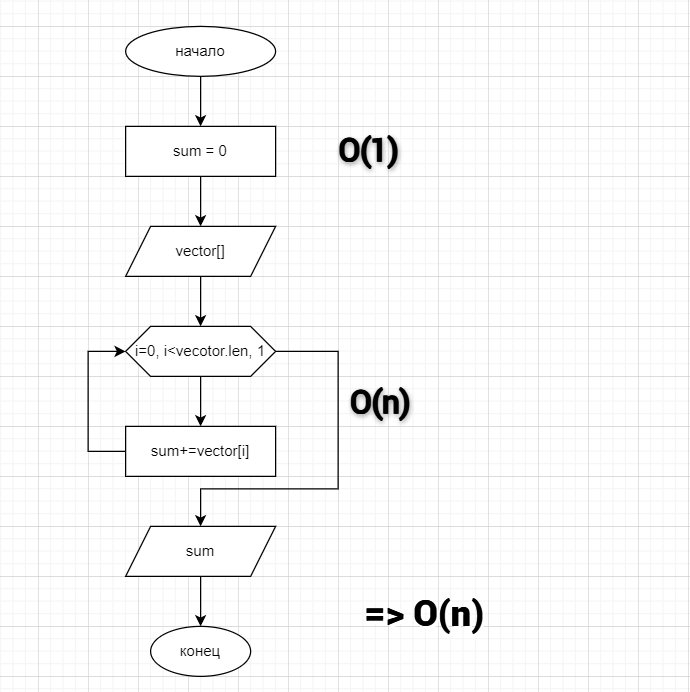


**рис.1.2**

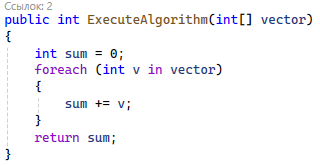
2) Сумма элементов – алгоритм, который перебором последовательно складывает все элементы вектора и возвращает их сумму

Теоретическая сложность O(n) (рис.2.1)

Код алгоритма представлен на рисунке 2.2



**Рис.2.1**

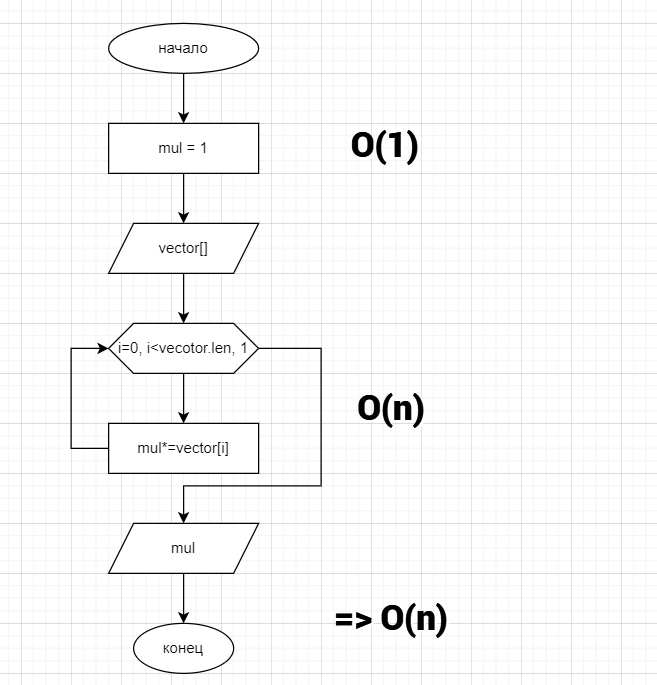


**Рис.2.2**

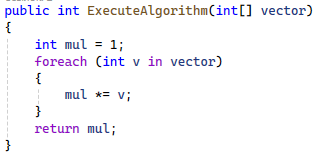
3) Произведение элементом - алгоритм, который перебором последовательно пееремножает все элементы вектора и возвращает их произведение

Теоретическая сложность O(n) (рис.3.1)

Код алгоритма представлен на рисунке 3.2



**рис.3.1**

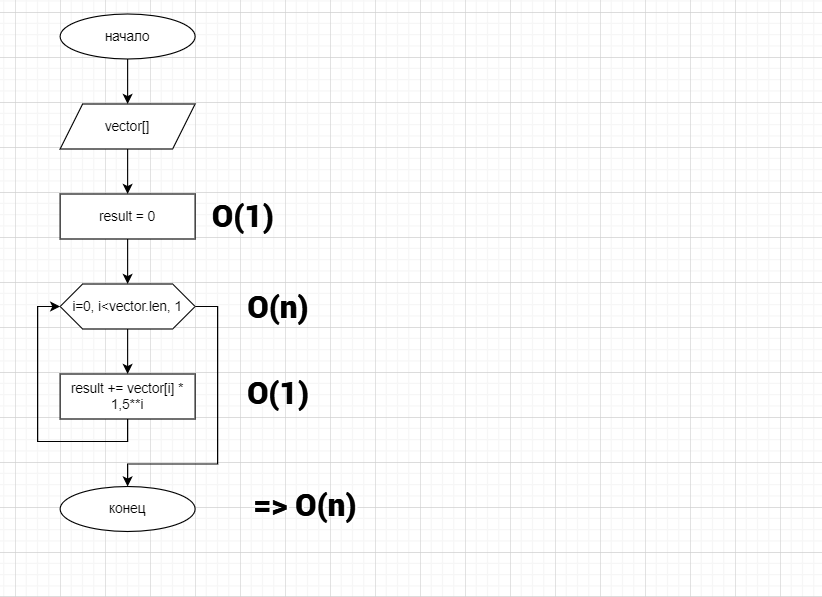


**рис.3.2**

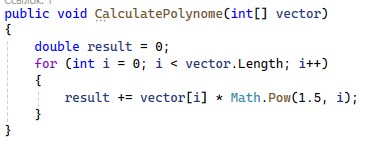
4) Алгоритм вычисления полинома - это последовательность шагов, которая позволяет вычислить значение полинома для заданных значений переменных. Обычно полином представляется в виде суммы слагаемых, каждое из которых содержит переменные, возведенные в некоторую степень. Для вычисления значения полинома нужно подставить заданные значения переменных вместо переменных в каждом слагаемом, произвести вычисления и сложить полученные результаты. Таким образом, алгоритм вычисления полинома помогает выяснить, какое значение полинома будет при конкретных значениях переменных.

Теоретическая сложность O(n) (рис.4.1)

Код алгоритма представлен на рисунке 4.2



**рис.4.1**

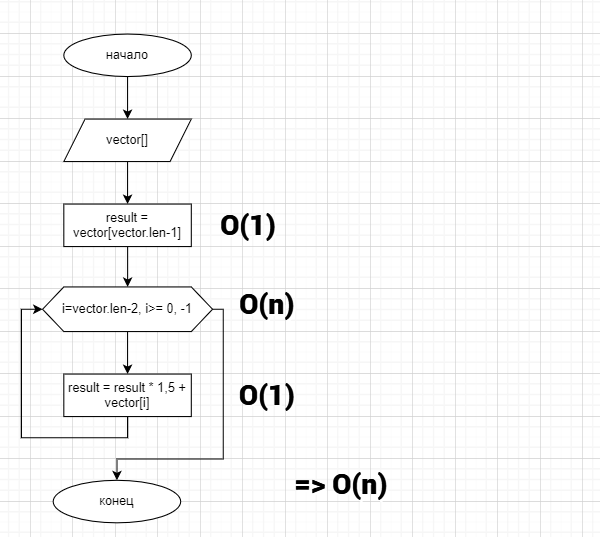


**рис.4.2**

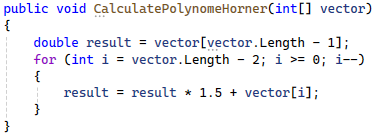
5) Метод Горнера - это метод вычисления значения полинома в точке, который позволяет уменьшить количество операций умножения при вычислении значений многочлена. Вместо того, чтобы вычислять каждое слагаемое по отдельности и складывать их все, используется более эффективный подход, который позволяет вычислить значение полинома за меньшее количество шагов.

Теоретическая сложность O(n) (рис.5.1)

Код алгоритма представлен на рисунке 5.2



**рис.5.1**

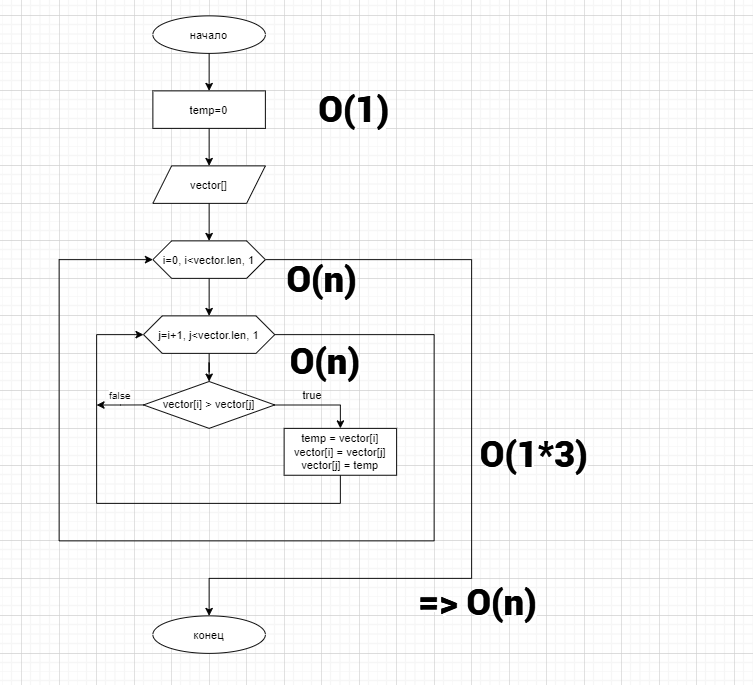


**рис.5.2**

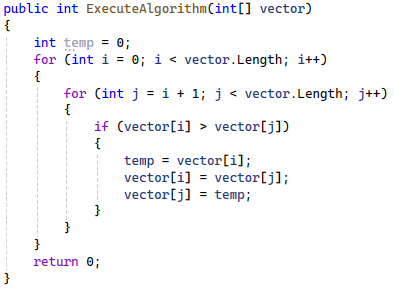
6) Алгоритм сортировки пузырьком - это простой алгоритм сортировки, который последовательно проходит по списку значений, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке. При каждом проходе по списку самый большой (или самый маленький, в зависимости от порядка сортировки) элемент "всплывает" на своё место, подобно пузырьку в воде, отсюда и название алгоритма. Процесс повторяется до тех пор, пока список не будет полностью отсортирован. Сортировка пузырьком неэффективна на больших наборах данных, но благодаря своей простоте она часто используется в образовательных целях или для сортировки небольших наборов данных.

Теоретическая сложность O(n^2) (рис.6.1)

Код алгоритма представлен на рисунке 6.2



**рис.6.1**



**рис.6.2**

7) Алгоритм быстрой сортировки (Quick Sort) - это эффективный алгоритм сортировки, который использует метод "разделяй и властвуй". Он работает следующим образом:

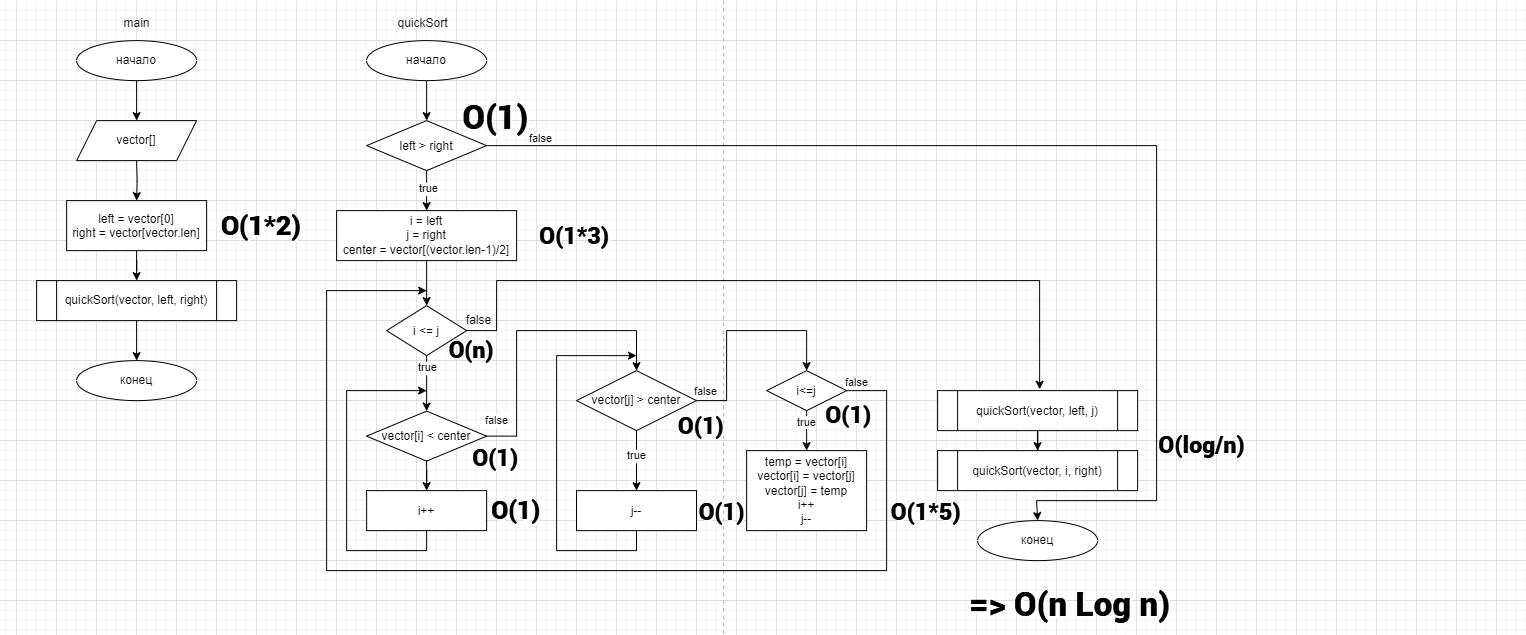
1. **Выбор опорного элемента**: Из массива выбирается один элемент, который называют опорным. Опорный элемент можно выбрать разными способами, например, взять первый, последний элемент или элемент, стоящий в середине массива.
2. **Разделение**: Массив делится на две части. Все элементы, которые меньше опорного, перемещаются влево от него, а все элементы, которые больше - вправо. Это называется операцией "разделение".
3. **Рекурсивная сортировка подмассивов**: Алгоритм рекурсивно применяется к двум получившимся частям массива - левому и правому подмассиву по отношению к опорному элементу.
4. **Слияние**: Обе части объединяются, хотя фактически в алгоритме Quick Sort само слияние не требуется, так как элементы уже находятся на своих окончательных местах.

Алгоритм завершает работу, когда все массивы, которые нужно сортировать, становятся единичными или пустыми, что, безусловно, означает, что они отсортированы.

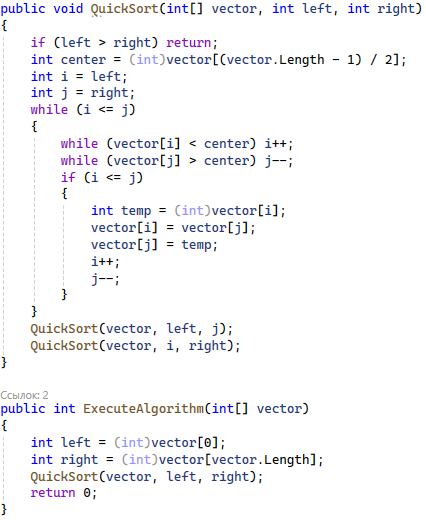
Quick Sort популярен из-за своей эффективности и обычно работает быстрее, чем другие алгоритмы сортировки, такие как сортировка пузырьком или сортировка выбором.

Теоретическая сложность O(n log n) (рис.7.1)

Код алгоритма представлен на рисунке 7.2



**рис.7.1**



**рис.7.2**

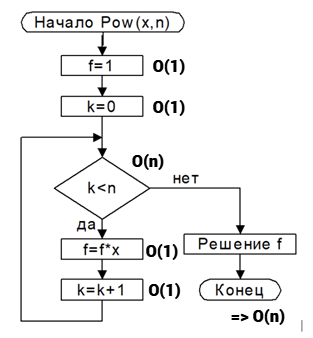
8) Возведение в степень

8.1. Классическое возведение в степень.

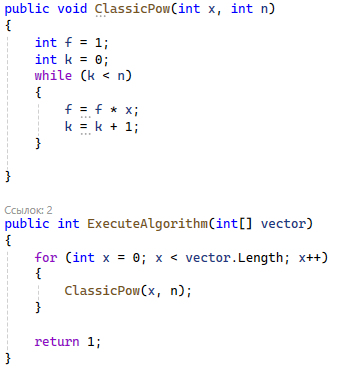
Алгоритм получает на вход основание и степень. С помощью цикла он вычисляет значение и выводит его.

Теоретическая сложность O(n) (рис.8.1.1)

Код алгоритма представлен на рисунке 8.1.2



**рис.8.1.1**



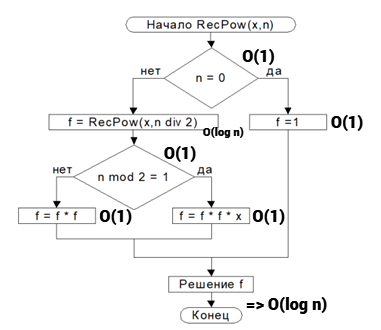
***рис.8.1.2***

8.2. Рекурсивное возведение в степень

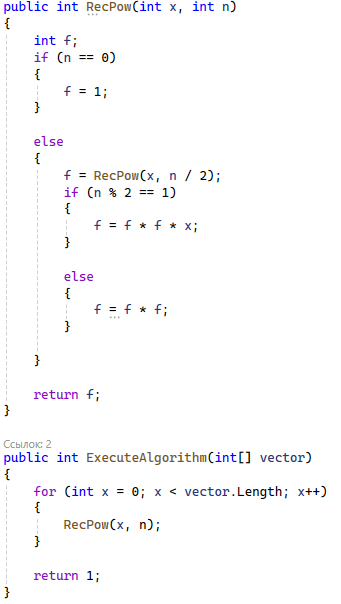
Алгоритм "ныряет" всё глубже, уменьшая степень вдвое на каждом шаге, пока не достигнет базового случая (n = 0). Затем, возвращаясь "назад", он комбинирует результаты подзадач, чтобы получить окончательный ответ.

Теоретическая сложность O(log n)

Код алгоритма представлен на рисунке 8.2.2



**рис.8.2.1**



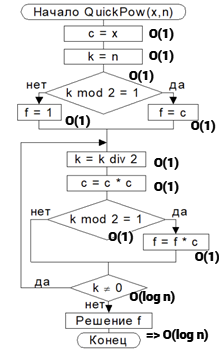
**рис.8.2.2**

8.3. Быстрый алгоритм

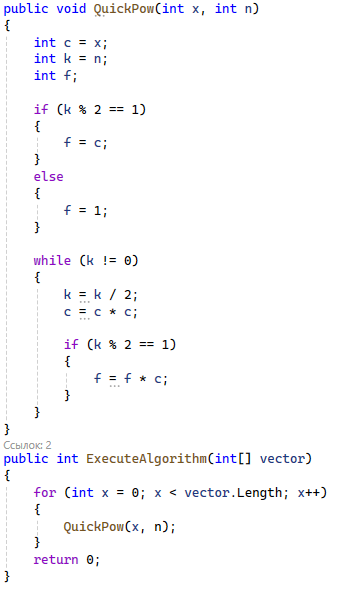
Алгоритм быстро возводит число в степень, сокращая количество умножений. Он делает это, деля степень пополам на каждом шаге и одновременно возводя основание в квадрат. Если степень нечётная, результат умножается на текущее основание. Этот процесс повторяется, пока степень не станет равна нулю, а результат будет содержать искомое значение

Теоретическая сложность O(log n)

Код алгоритма представлен на рисунке 8.3.2



**рис.8.3.1**



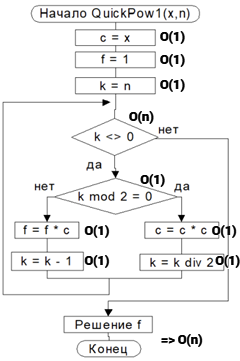
***рис.8.3.2***

8.4. Классический быстрый алгоитм

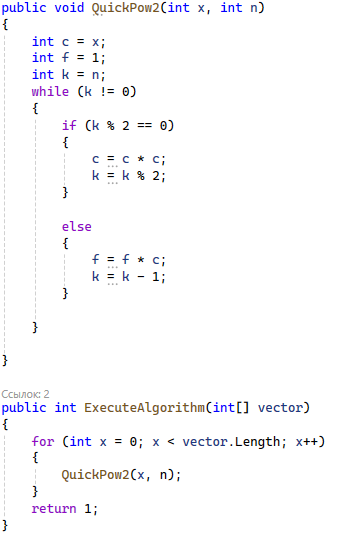
Алгоритм QuickPow1 быстро вычисляет степень числа, даже если она отрицательная. Он использует цикл, в котором на каждом шаге проверяет чётность степени. Если степень чётная, то основание возводится в квадрат, а степень делится на два. Если степень нечётная, то результат умножается на текущее основание, а степень уменьшается на единицу. Этот процесс повторяется до тех пор, пока степень не станет равной нулю.

Теоретическая сложность O(n)

Код алгоритма представлен на рисунке 8.4.2



**рис.8.4.1**



**рис.8.4.2**

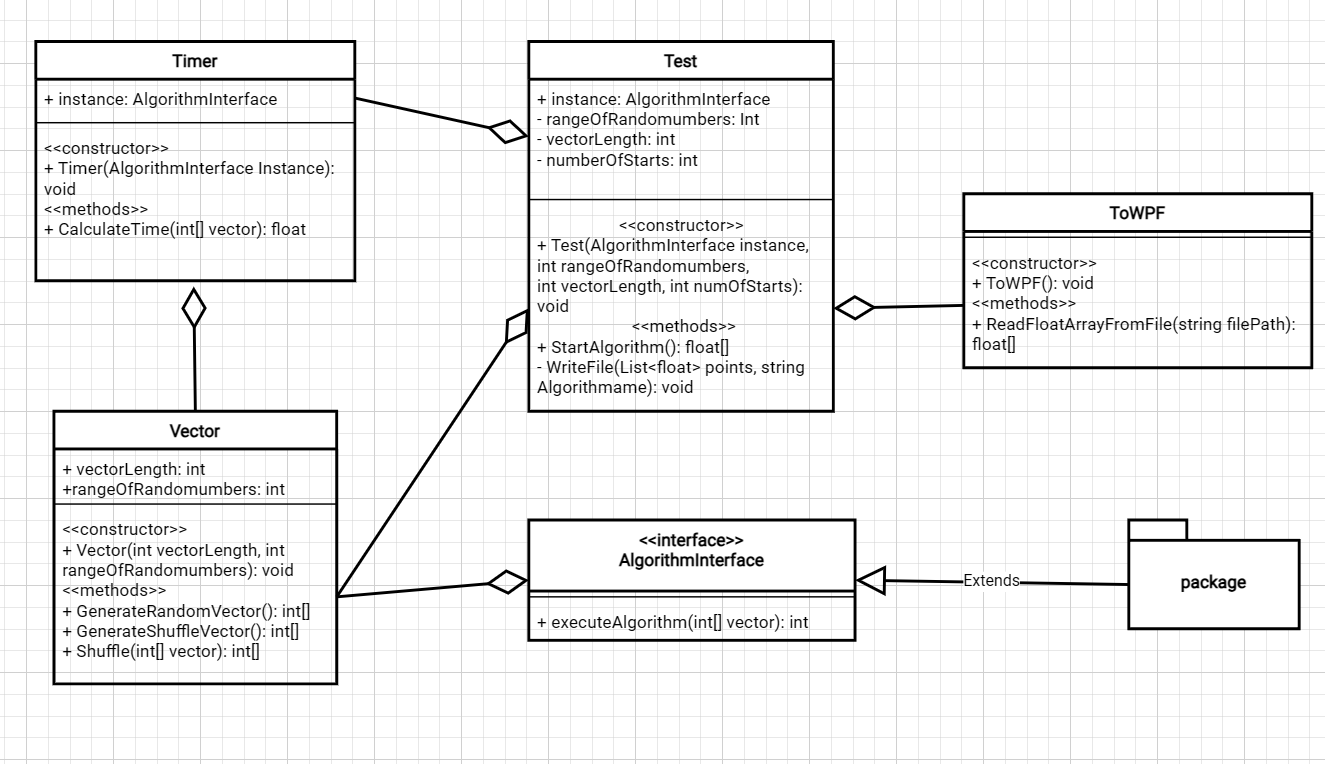
Блок 2

Модель проекта

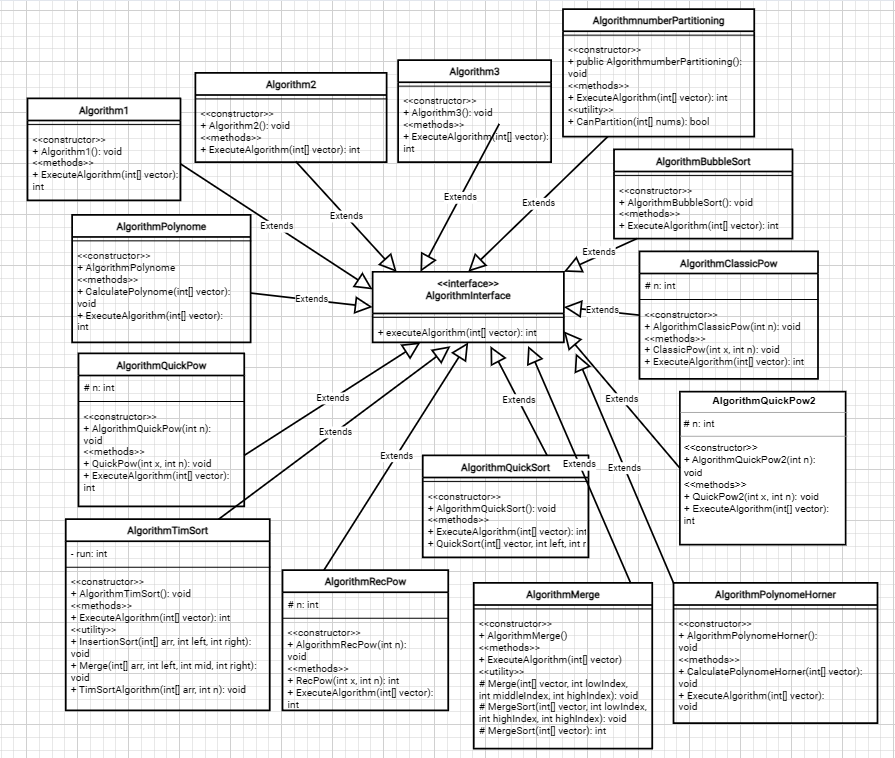
Все приложение работает на языке C#, для содания внешней оболочки использован WPF, для отрисовки графиков – ScottPlot.

Приложение поделено на 2 части – фронтенд и бэкенд. В основе бэкенда лежит три созданных библиотеки:

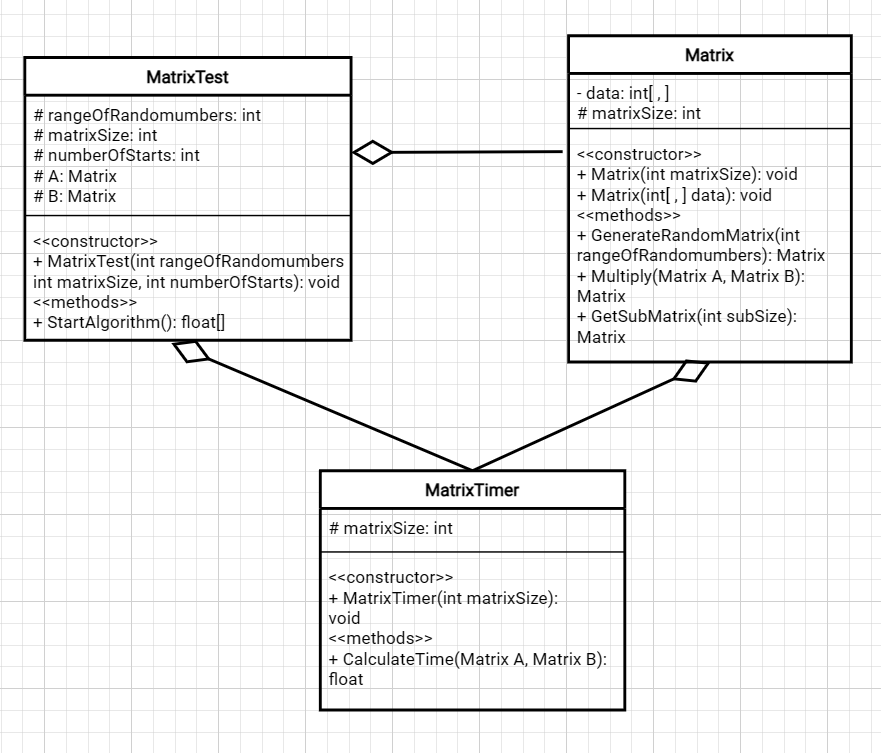
1) AlgLogic – необходима для работы с алгоритмами которым на вход подается вектор



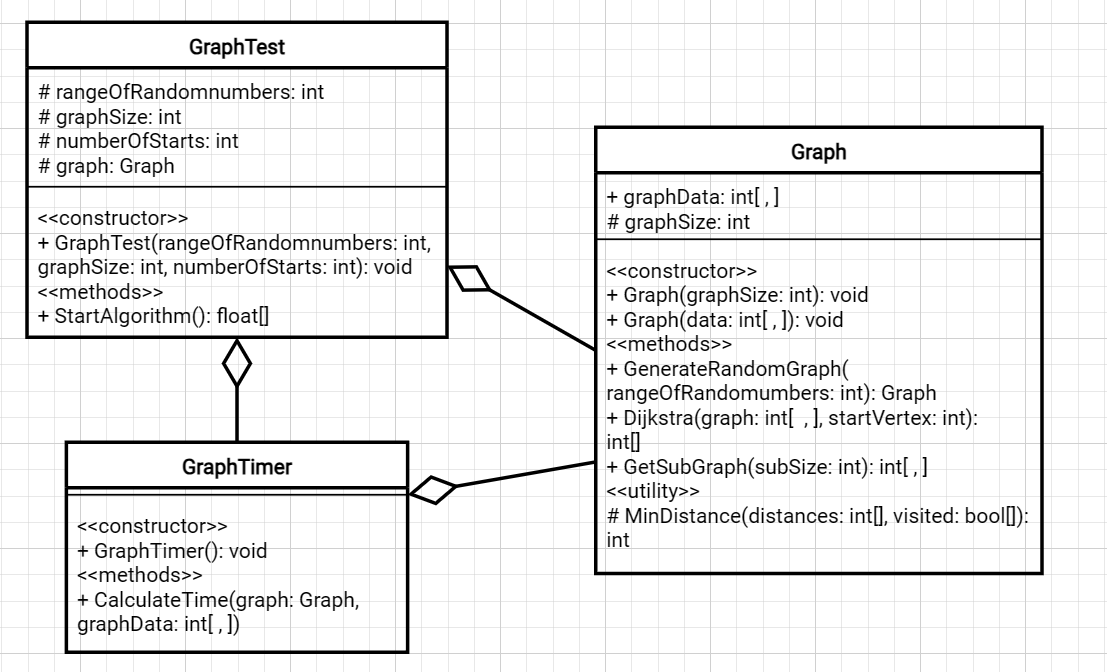
Здесь под пакетом имеются все классы-наследники интерфейса



2) MatrixEntities – необходима для выполнения операций над матрицами



3) DijkstraAlgorithm – необходима для работы с графами и выполнения алгоритма Дейкстры.



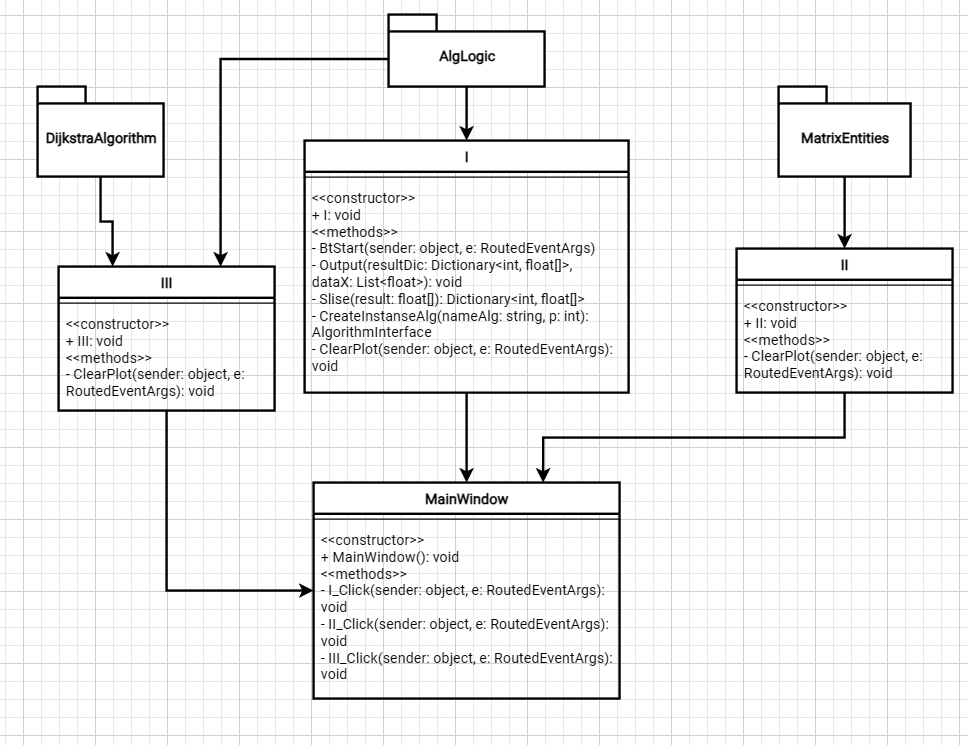
Фронтенд состоит из одной библиотеки в которой содержатся четыре файла:

1) MainWindow.xaml + MainWindow.cs – эти классы отвечают за начальный экран приложения, на котором расположены три кнопки отвечающие за переходы в следующие части программы

2) I.xaml + I.cs – отвечает за вывод результатов первого блока лабораторной работы

3) II.xaml + II.cs – отвечает за вывод результатов второго блока лабораторной работы (матрицы)

4) III.xaml + III.cs – отвечает за вывод результатов третьего блока лабораторной работы (собственные алгоритмы)



Блок 2

В данном блоке требуется перемножить две квадратные матрицы размера n.

Чтобы перемножить квадратные матрицы, нужно **умножить элементы в строках первой матрицы на элементы в столбцах второй матрицы и сложить полученные значения**.

Умножать можно только **согласованные матрицы**, где количество столбцов первой матрицы равно количеству строк второй.

**Алгоритм умножения:**

**1)** Перемножаем числовые значения первой строки на значения из первого столбца:

1.1) умножаем первый элемент первой строки на соответствующий элемент из первого столбца;

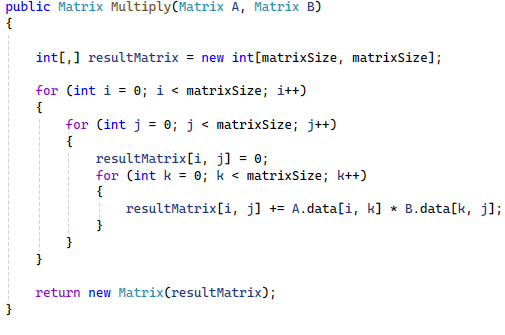
1.2) находим произведение второго элемента первой строки и второго элемента, который берём из столбца №1.

2) Проделываем такие же действия со всеми элементами, пока не дойдём до конца первой строки матрицы.

3) Вычисленные произведения необходимо сложить между собой. Вычисленный результат будет равняться элементу для первой строки.

4) Аналогичные действия нужно проводить с каждой строкой вычисляемой матрицы. Вычисления проводятся до тех пор, пока все строчки новой матрицы не будут заполнены значениями.

Теоретическая сложность: O(n^3)



Блок 3

1)Алгоритм о разбиении множества на два подмножества с одинаковой суммой

Временная сложность

Теоретическая сложность: O(n \* targetSum),

где n — количество элементов в массиве nums, а targetSum — половина суммы всех элементов массива.

Объяснение временной сложности

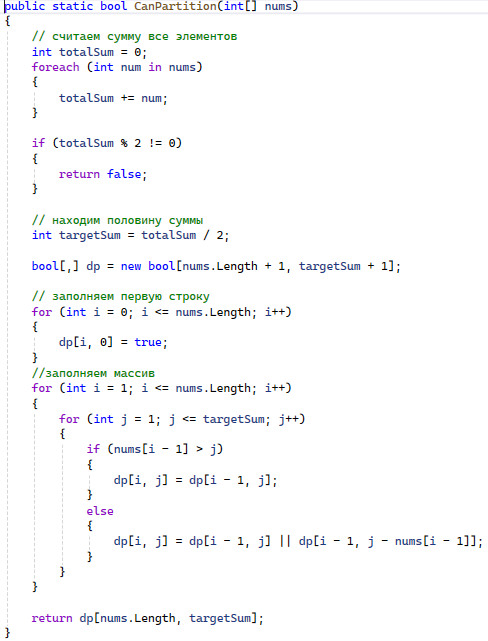
Вычисление общей суммы: O(n)

Заполнение массива dp: O(n \* targetSum)

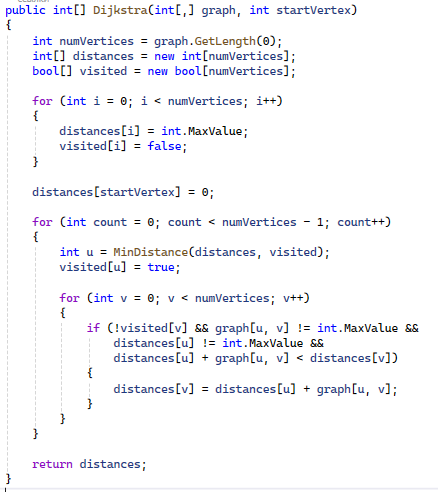
Итоговая сложность: O(n \* targetSum)

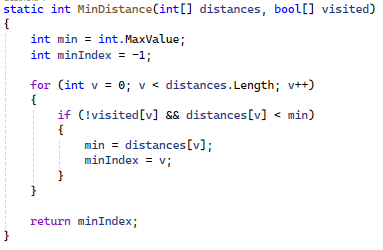
Заключение:

Алгоритм CanPartition решает задачу разбиения множества на два подмножества с одинаковой суммой с использованием динамического программирования. Временная сложность алгоритма составляет O(n \* targetSum), что делает его эффективным для небольших и средних размеров массива. Для больших массивов время выполнения может стать значительным из-за экспоненциального роста ‘targetSum’

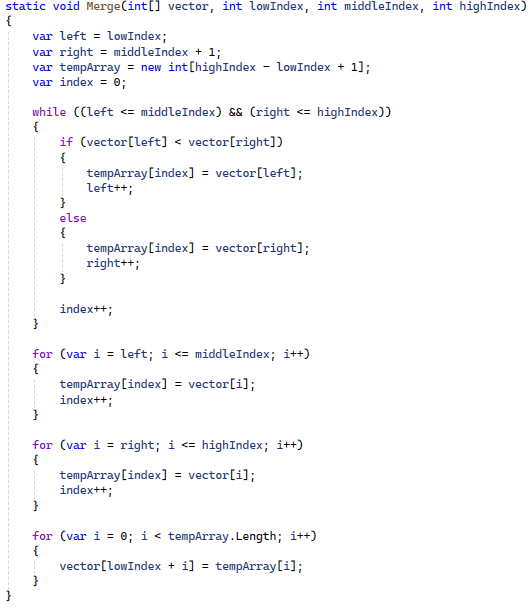


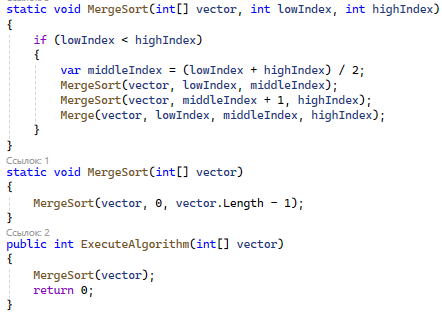
Дейкстра





Merge





Timsort

