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

Авторы отчета: Смирнов А.М. ПрИ-202,

Голубев Н.В. ПрИ-202,

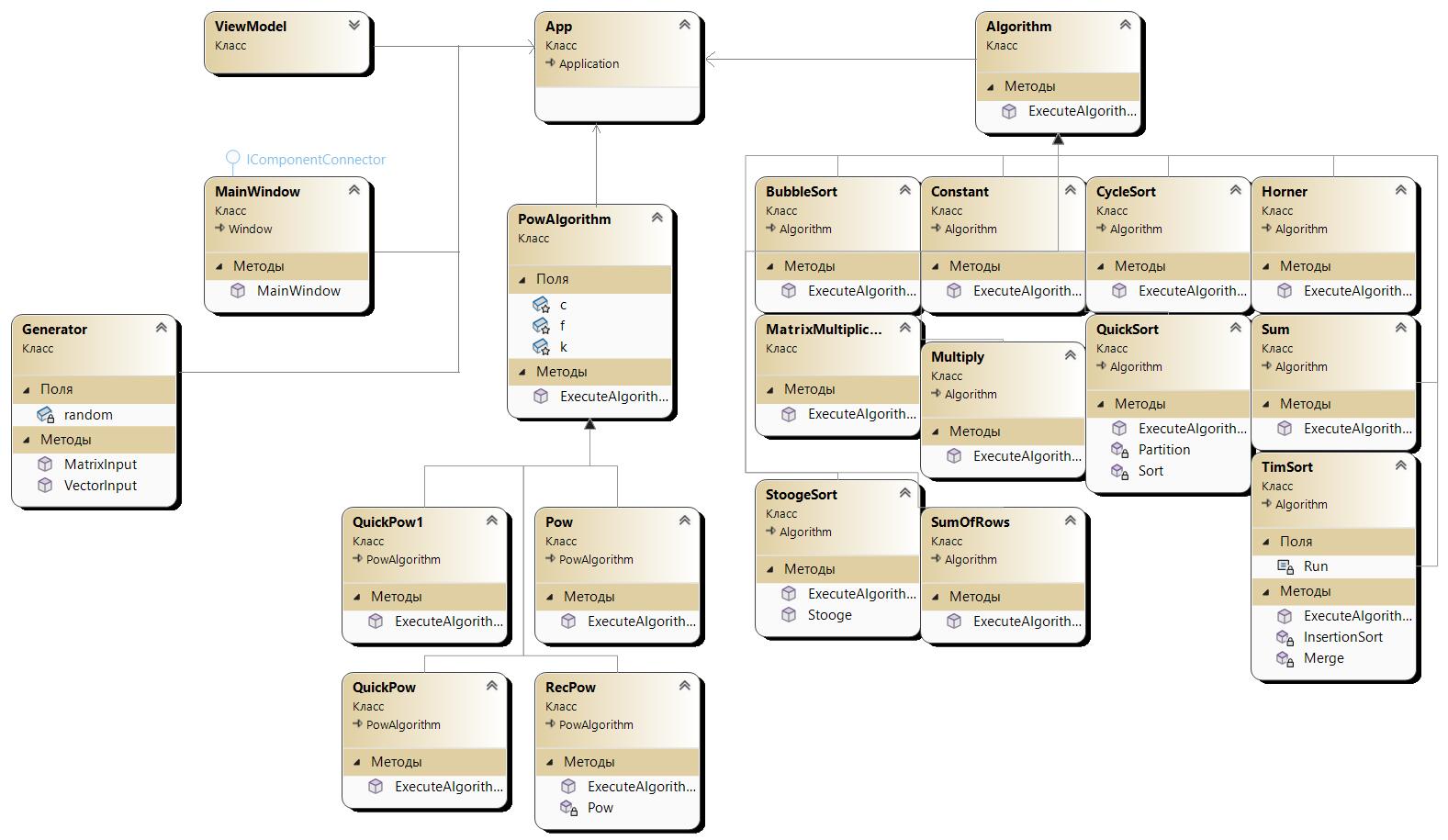
Сайбуллин Р.Д. ПрИ-202.

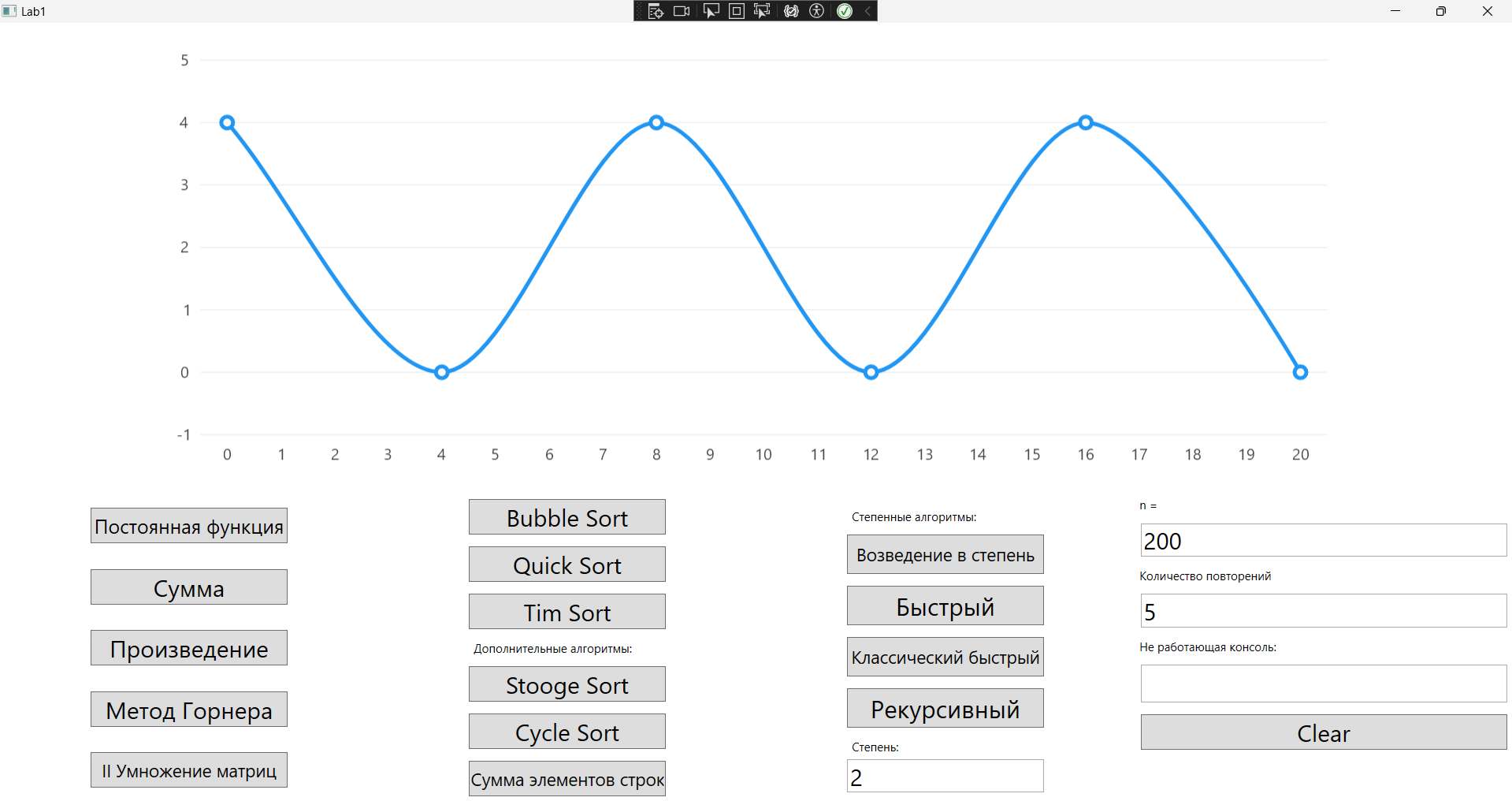
Проверил: Николаев И.Е.

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

дата оценка

Челябинск 2024 г.

**UML-диаграмма:**

****

После запуска появляется меню с графиком, опциями и полями ввода. После ввода n и выбора нужного алгоритма он начинает работу и потом вывод результаты на график.

1. **Постоянная функция**

Метод постоянной функции. В методе переменной х присваивается значение 1. Сложность алгоритма О(1). Экспериментальные данные представлены на Диаграмме 1. Реализация алгоритма представлена на Блок-схеме 1, Коде 1.

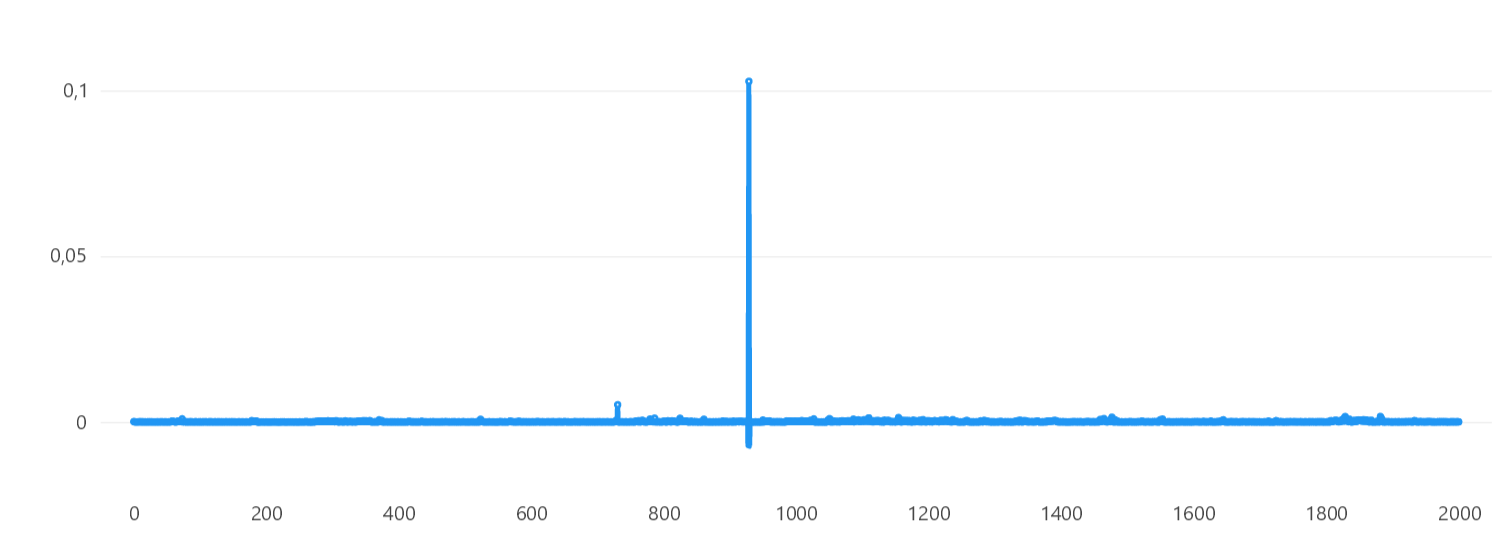
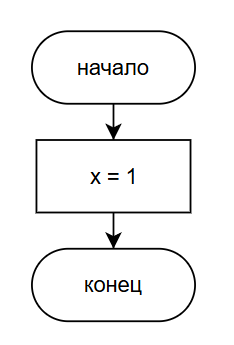
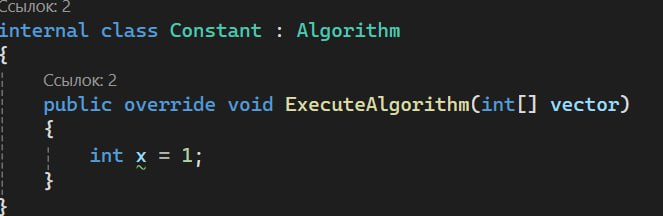


Диаграмма 1 Постоянная функция



Блок-схема 1 Постоянная функция



Код 1Постоянная функция

1. **Сумма элементов**

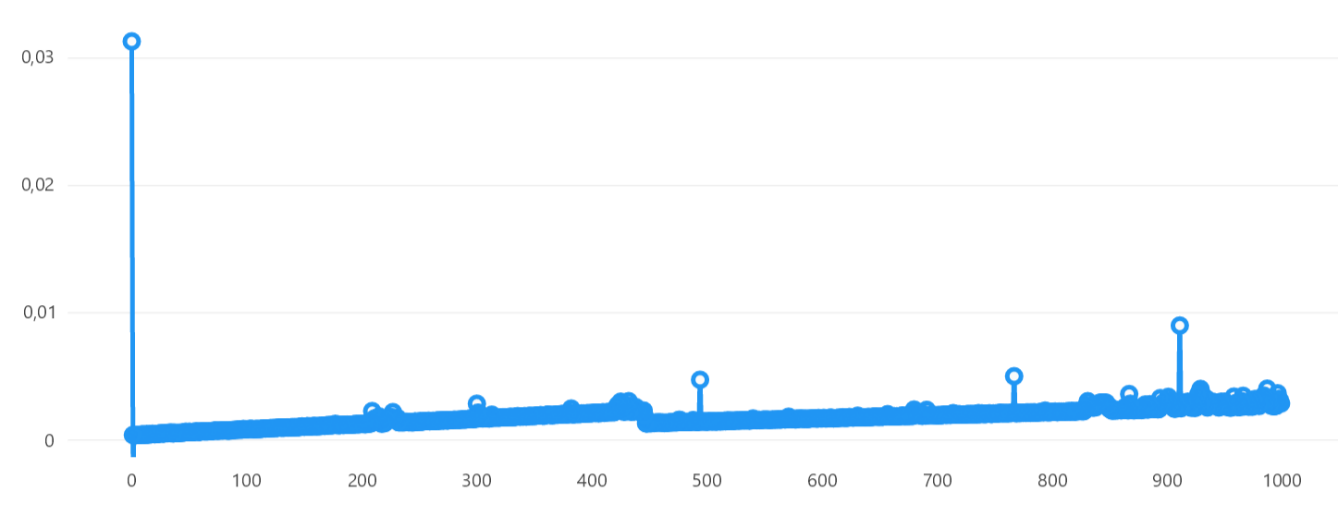
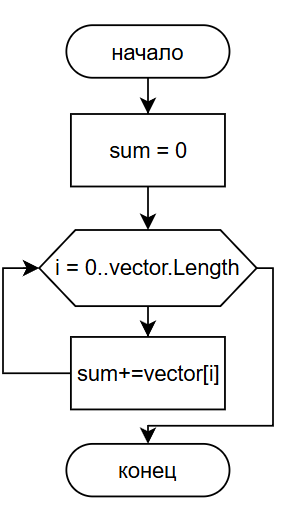
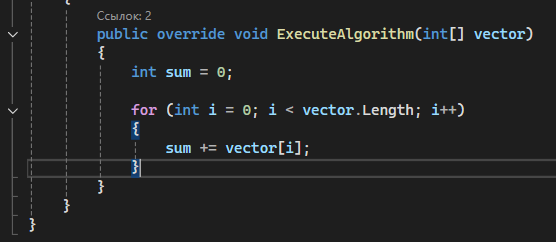
Метод суммы элементов массива. Создаём переменную sum = 0, в которой, с помощью цикла обходим массив, каждое значение прибавляем к переменной sum. Сложность алгоритма О(n). Экспериментальные данные представлены на Диаграмме 2. Реализация алгоритма представлена на Блок-схеме 2, Коде 2.

Диаграмма 2 Сумма значений



Блок-схема 2 Сумма значений



Код 2 Сумма значений

1. **Произведение элементов**

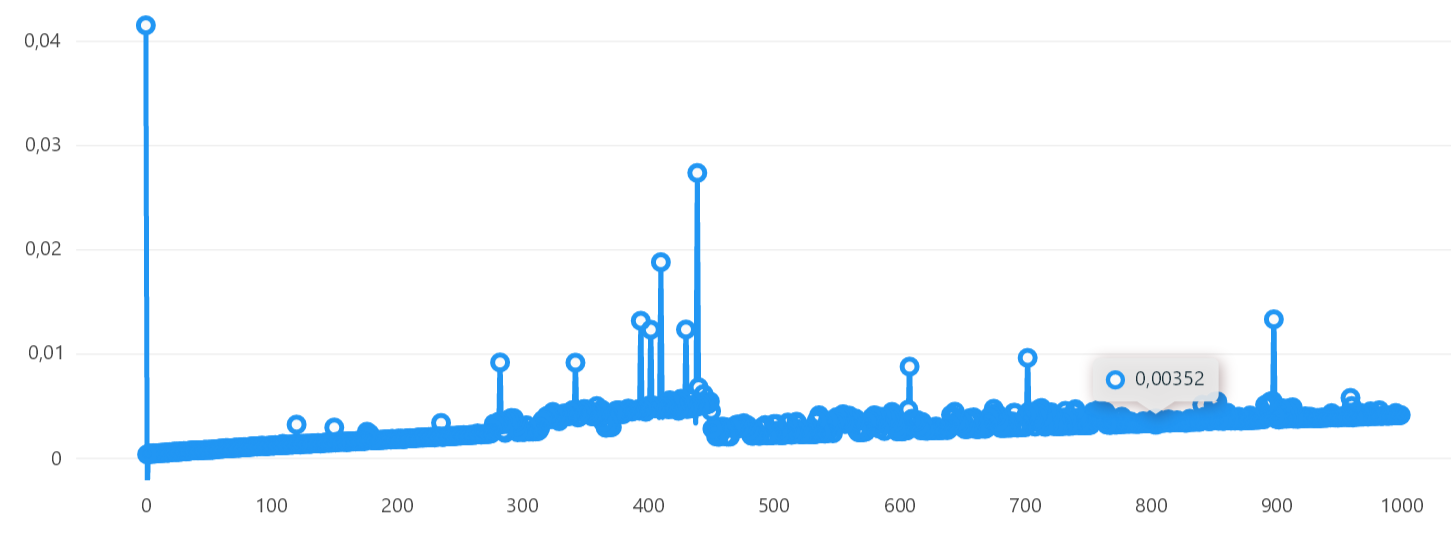
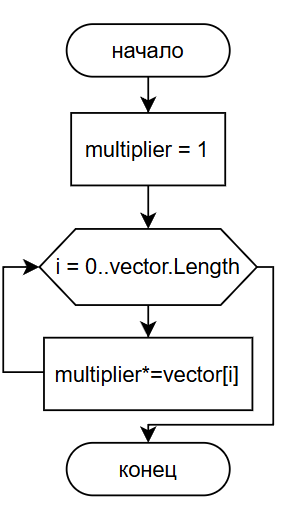
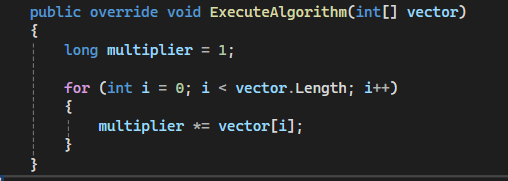
Метод произведения элементов массива. Создаём переменную multiplier = 1, с помощью цикла обходим массив, переменную multiplier умножаем на каждое значение в массиве. Сложность алгоритма О(n). Экспериментальные данные представлены на Диаграмме 3. Реализация алгоритма представлена на Блок-схеме 3, Коде 3.

Диаграмма 3 Произведение элементов



Блок-схема 3 Произведение элементов

Код 3 Произведение элементов

1. **Метод Горнера**

Метод прямого вычисления полинома Горнера. Передаем в метод массив. Используется цикл по всей длине массива. В цикле происходит вычисление component = component \* 1.5 + vector[i]. Сложность алгоритма О(n). Экспериментальные данные представлены на Диаграмме 4. Реализация алгоритма представлена на Блок-схеме 4, Коде 4.

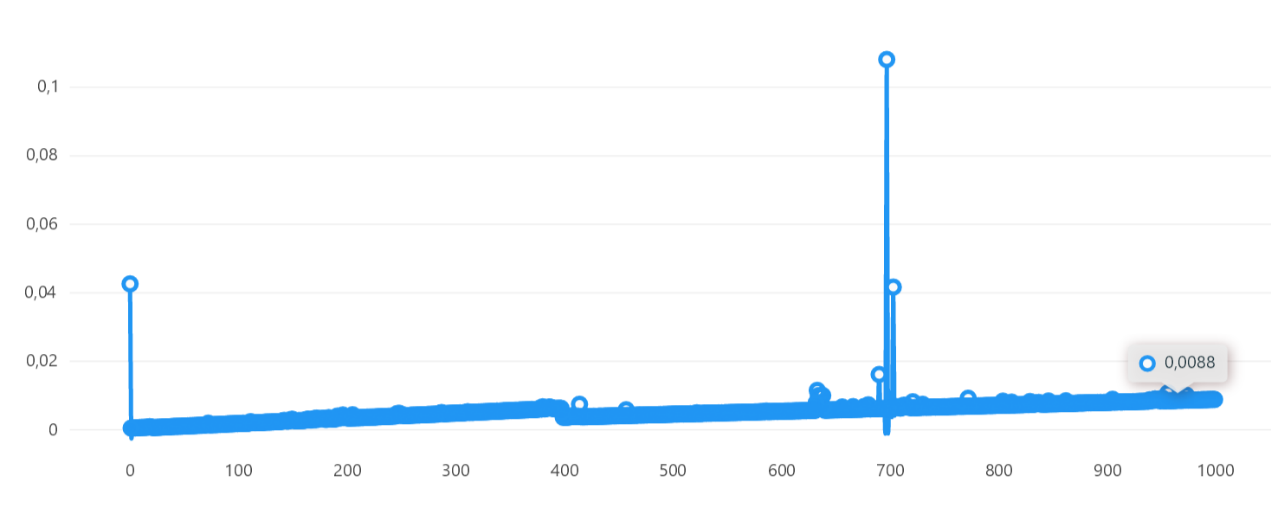
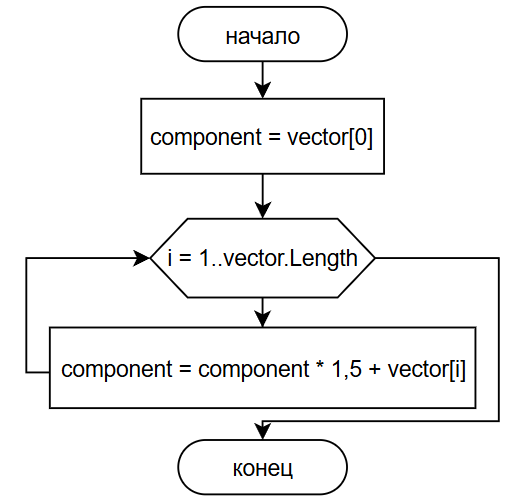
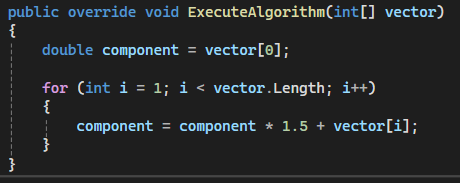


Диаграмма 4 Метод Горнера



Блок-схема 4 Метод Горнера



Код 4 Метод Горнера

1. **Сортировка пузырьком**

Метод сортировки Пузырьком. Запускаем цикл от i = 0 до длины массива с шагом 1, затем запускаем цикл от j = 1 до длинны массива с шагом 1. Сравниваются переменные под номером i и j, и если элемент с индексом i больше элемента с индексом j, то эти элементы меняются местами. Сложность алгоритма О(n^2). Экспериментальные данные представлены на Диаграмме 5. Реализация алгоритма представлена на Блок-схеме 5, Коде 5.

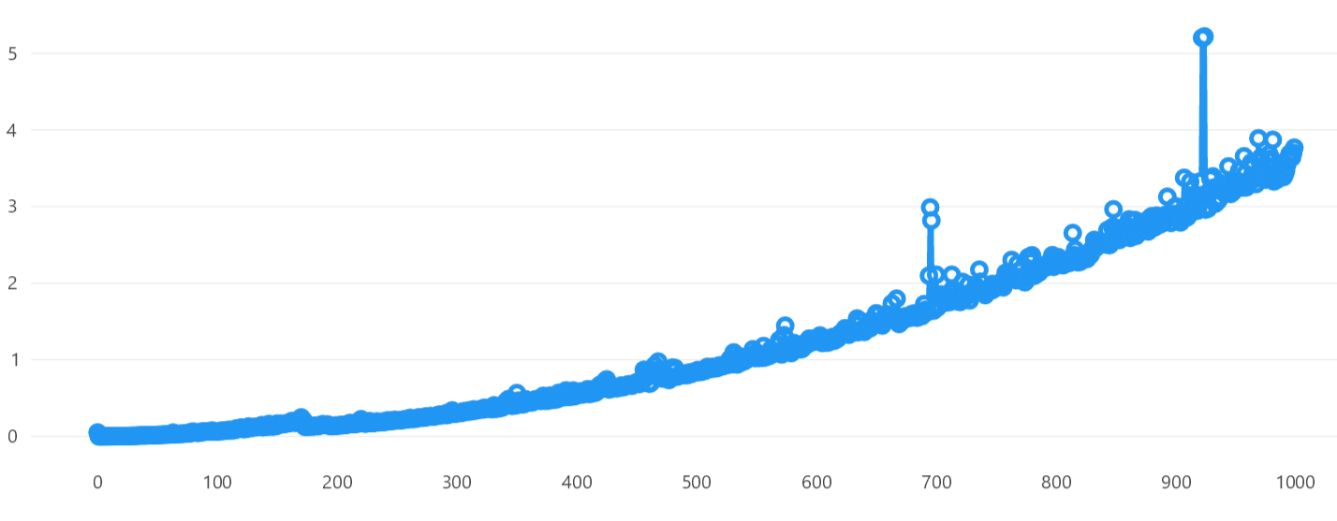
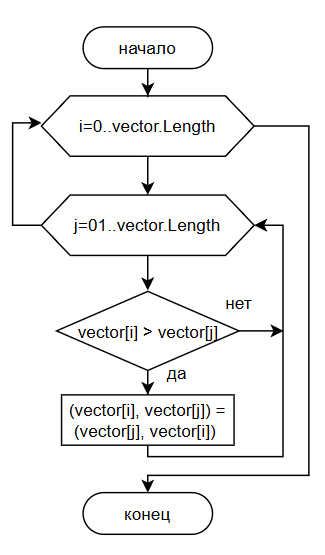
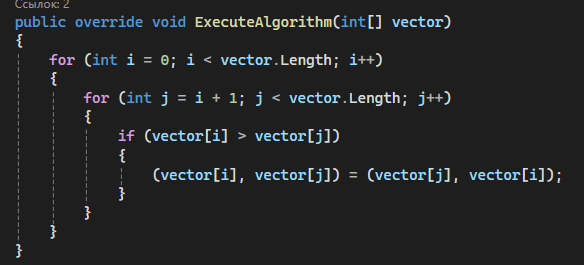


Диаграмма 5 Сортировка пузырьком



Блок-схема 5 Сортировка пузырьком



Код 5 Сортировка пузырьком

1. **Быстрая сортировка**

Быстрая сортировка. Идея алгоритма состоит в следующем:

Выбрать из массива некоторый опорный элемент. В нашем случае это крайний слева элемент массива. От выбора опорного элемента может зависеть его эффективность.

Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы слева от опорного элемента были все элементы, меньшие опорного, а справа – те что больше опорного. Таким образом опорный элемент ставится на своё место в отсортированном массиве, а левая и правая часть начинают сортироваться рекурсивно таким же способом, если длина этих частей не равна 1. Сложность алгоритма О(n^2) в худшем случае. Экспериментальные данные представлены на Диаграмме 6. Реализация алгоритма представлена на Блок-схеме 6, Коде 6.

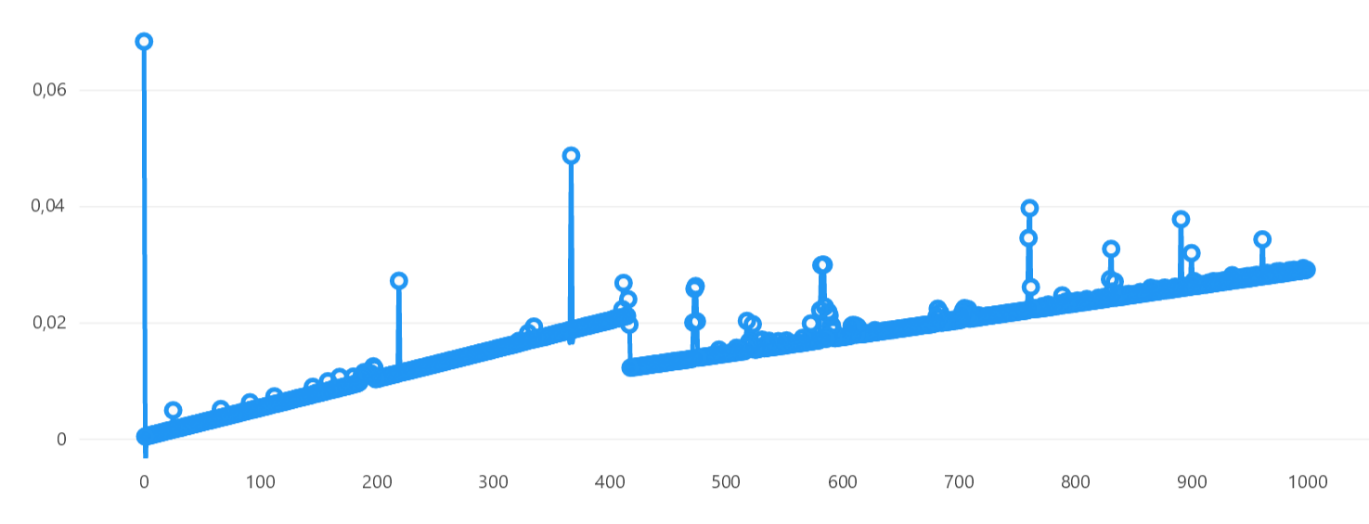
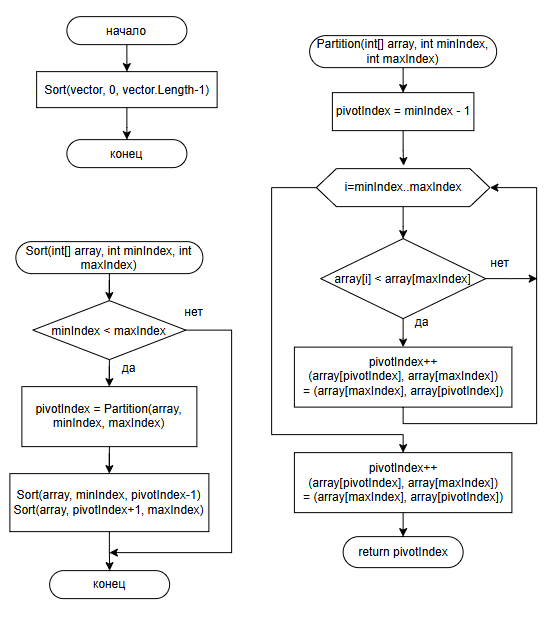
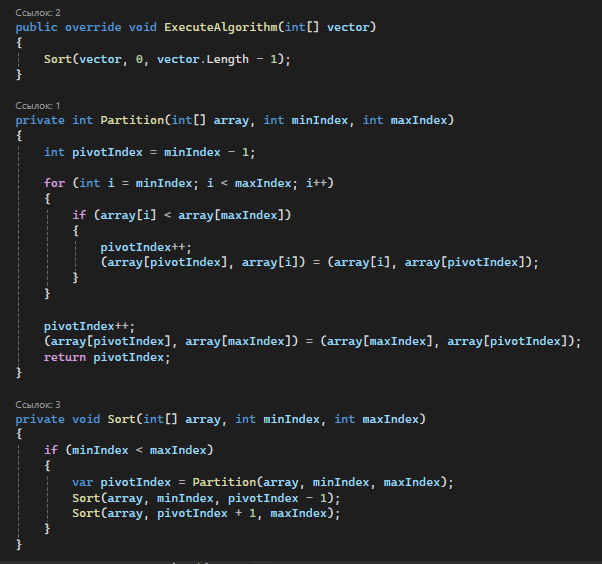


Диаграмма 6 Быстрая сортировка



Блок-схема 6 Быстрая сортировка



Код 6 Быстрая сортировка

1. **Timsort**

Идея алгоритма:

По специальному алгоритму входной массив разделяется на подмассивы некоторой длины. Каждый подмассив отдельно сортируется сортировкой вставками. Отсортированные подмассивы сливаются в единый массив с помощью модифицированной сортировки слиянием. Сложность алгоритма О(nlog(n)).

Экспериментальные данные представлены на Диаграмме 7. Реализация алгоритма представлена на Блок-схеме 7, Блок-схеме 8, Коде 7, Коде 8, Коде 9.

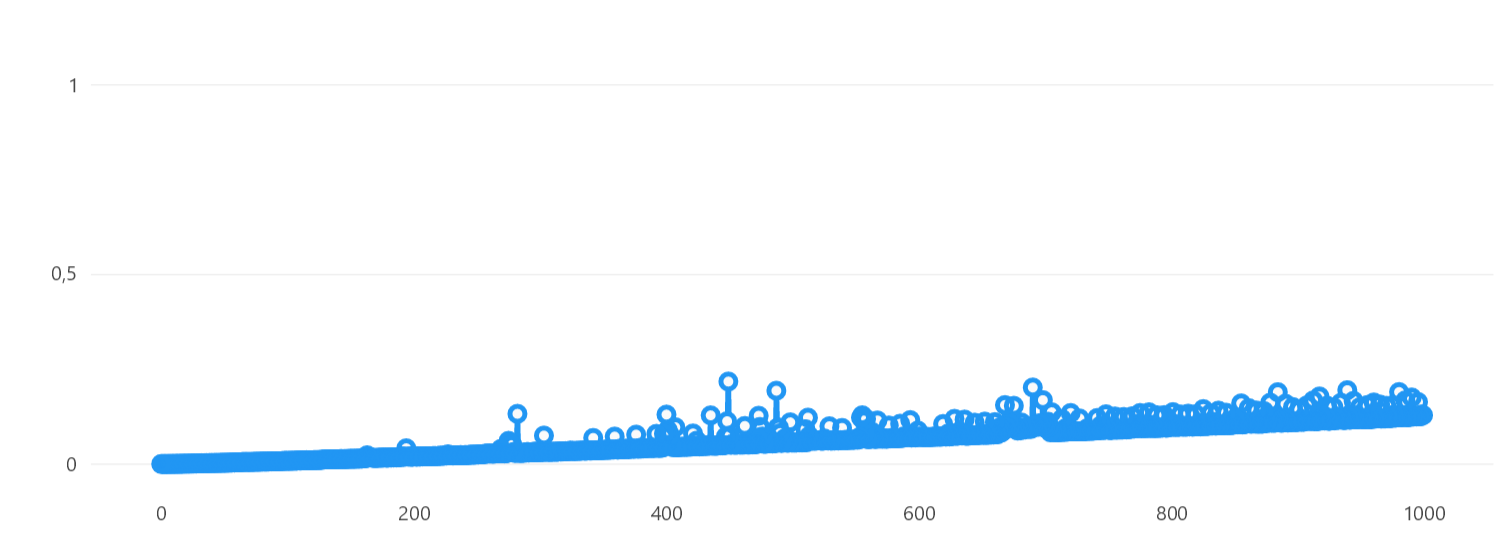
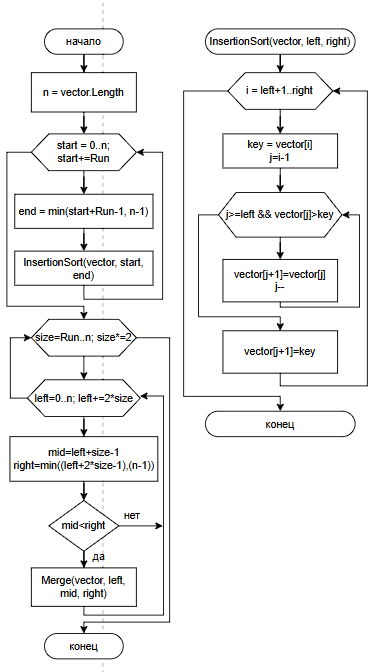
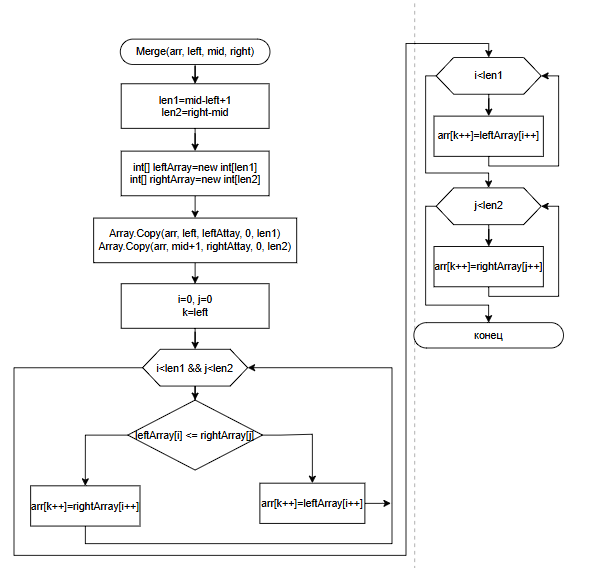


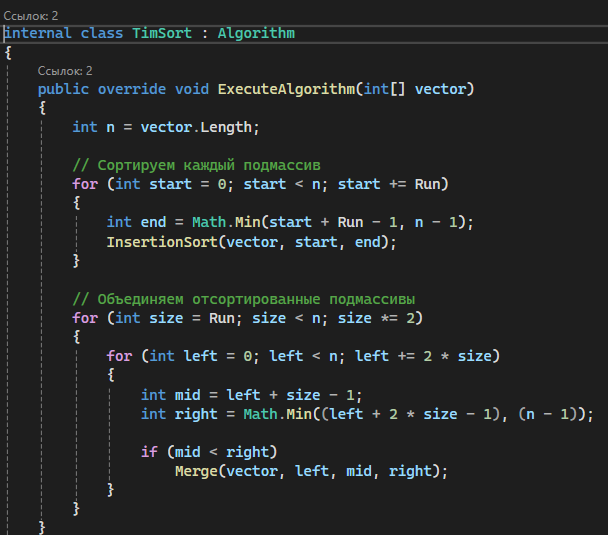
Диаграмма 7 TimSort



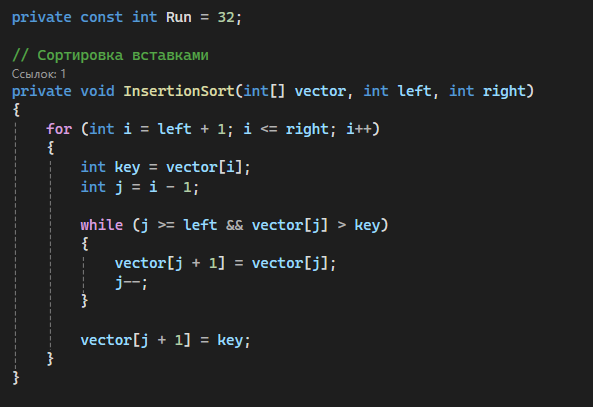
Блок-схема 7 TimSort



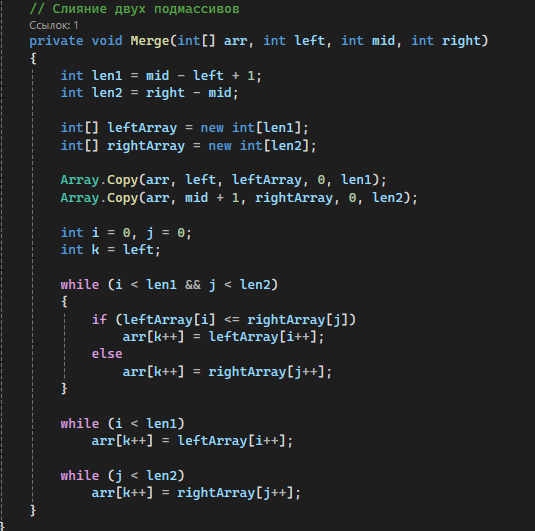
Блок-схема 8 TimSort



Код 7 TimSort



Код 8 TimSort



Код 9 TimSort

1. **Простое возведение в степень**

Метод простого возведения в степень. На вход подается степень power. Случайно выбирается значение переменной, которая будет возводиться в степень power, создаётся переменная f = 1. После этого в цикле умножаем переменную f на заранее выбранное основание требуемое количество раз. В конце прохода присваиваем vector[i] значение из f. Сложность алгоритма О(n). Экспериментальные данные представлены на Диаграмме 8. Реализация алгоритма представлена на Блок-схеме 9, Коде 10.

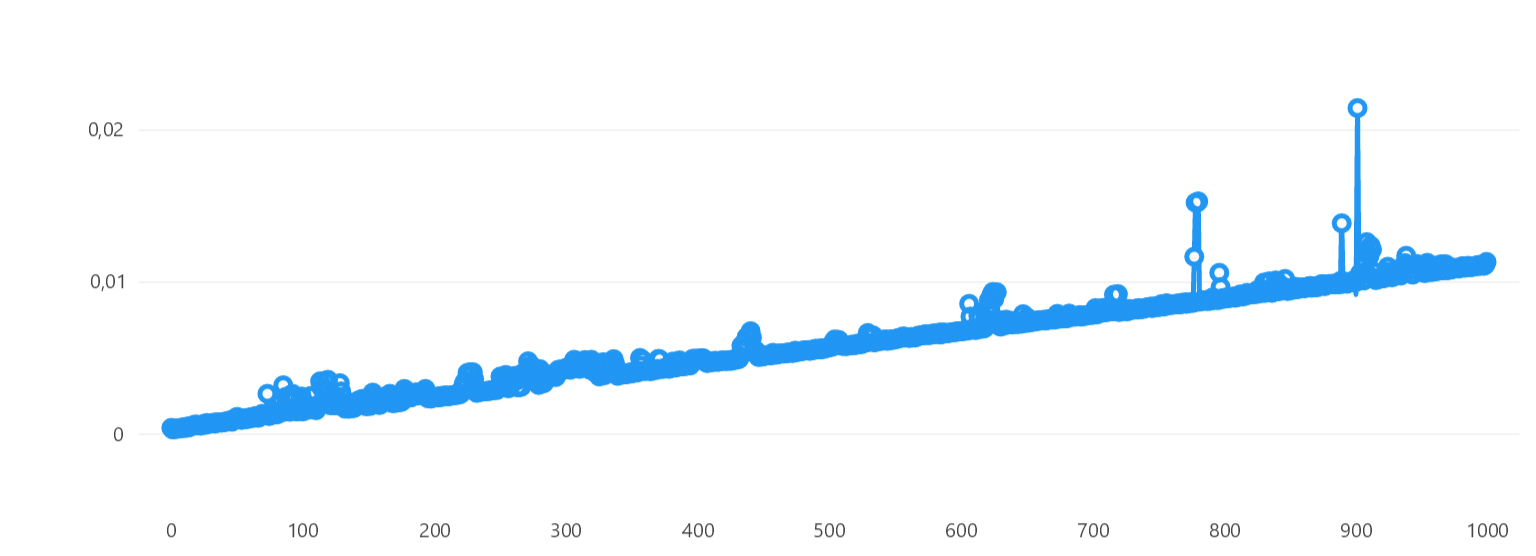
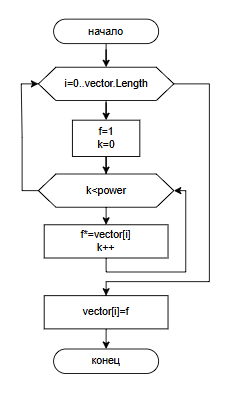
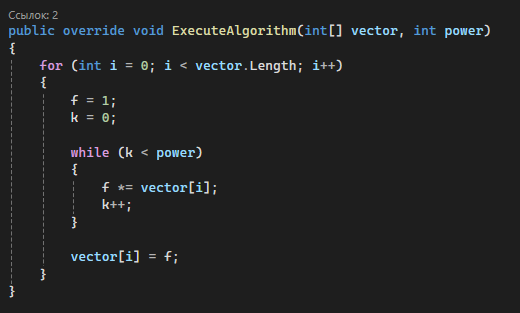


Диаграмма 8 Простое возведение в степень



Блок-схема 9 Простое возведение в степень



Код 10 Простое возведение в степень

**9. Рекурсивное возведение в степень**

Метод рекурсивного возведения в степень. На вход подается степень n. Случайно выбирается основание, которое будет возводиться в степень n, создаётся переменная f. В случае, если степень n равна 0, возвращаем 1, иначе возвращаем произведение основания на эту же функцию Pow, в которой n уменьшена в да раза. Сложность алгоритма О(logn). Экспериментальные данные представлены на Диаграмме 9. Реализация алгоритма представлена на Блок-схеме 10, Коде 11.

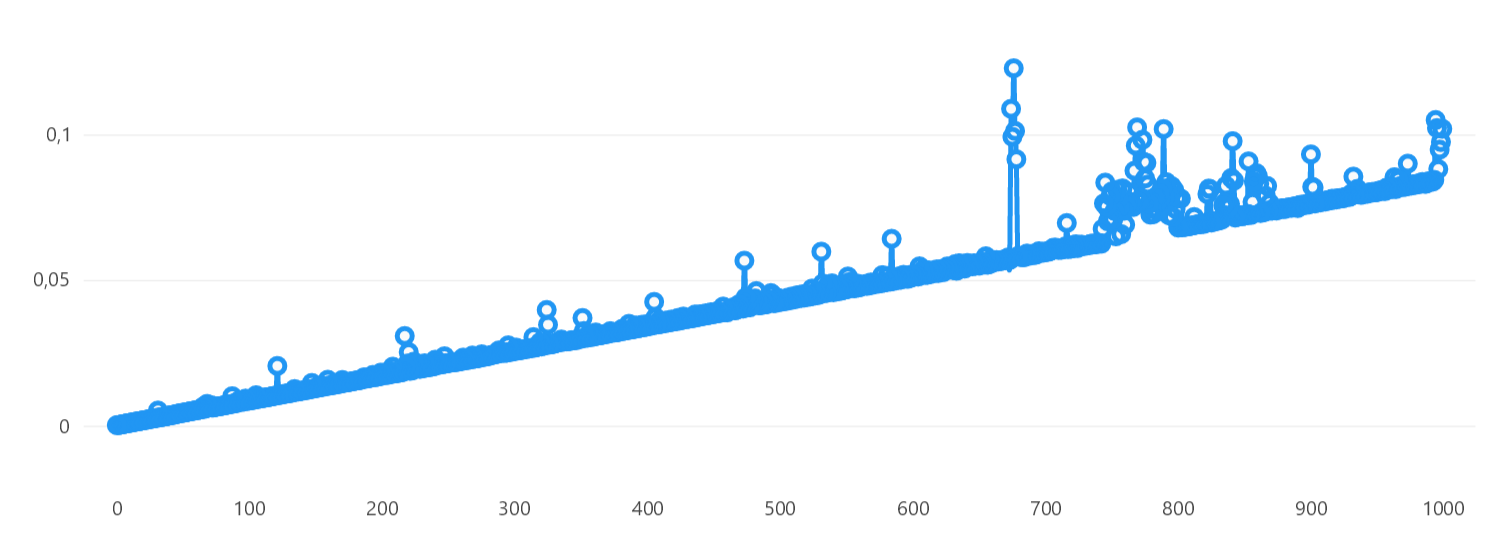
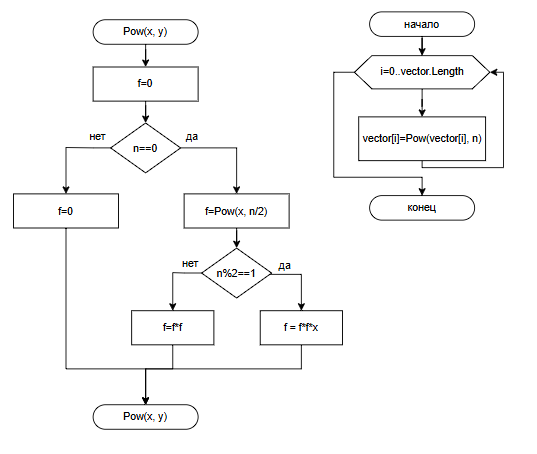
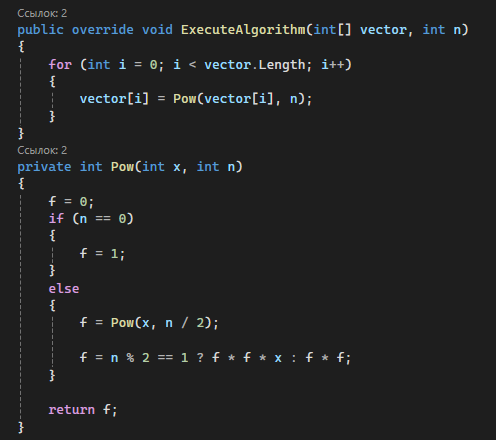


Диаграмма 9 Рекурсивное возведение в степень



Блок-схема 10 Рекурсивное возведение в степень



Код 11 Рекурсивное возведение в степень

1. **Быстрое возведение в степень**

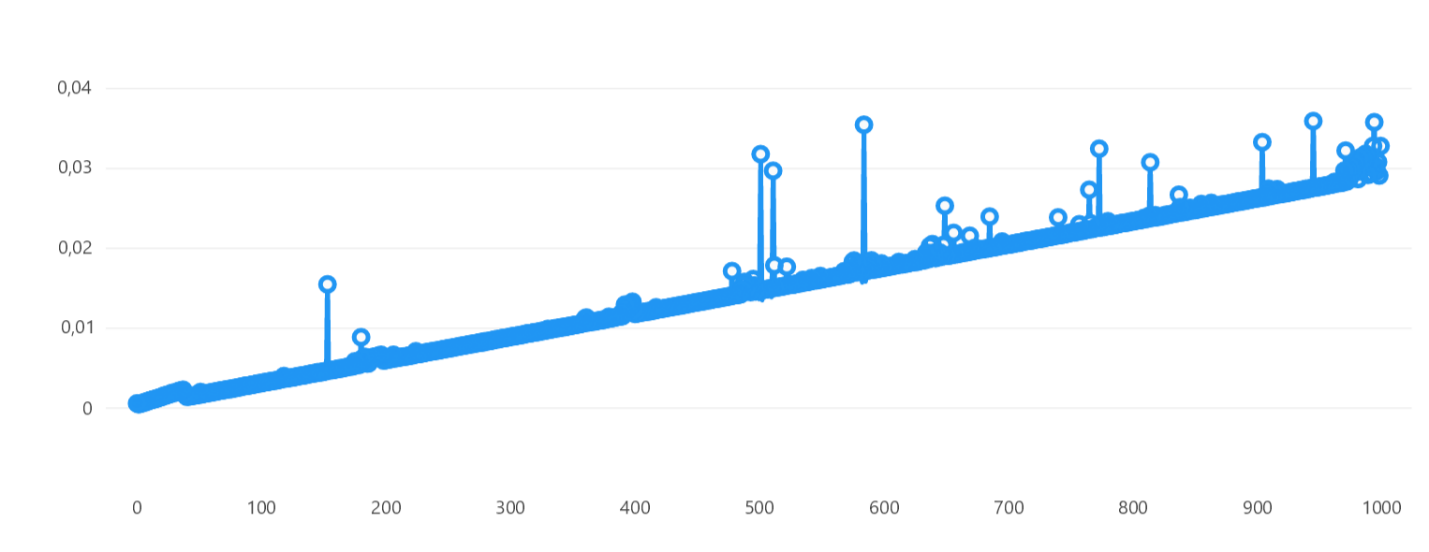
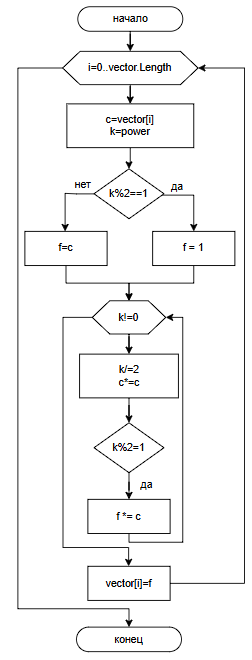
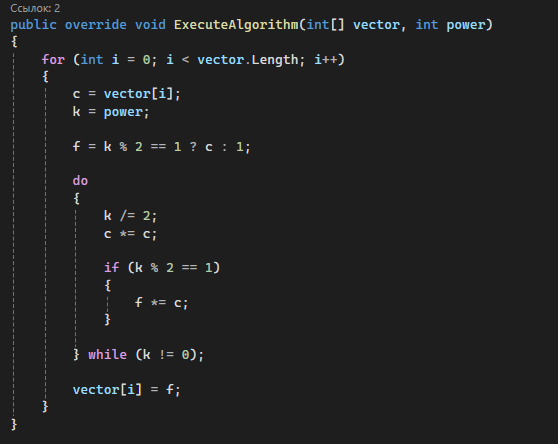
Метод быстрого возведения в степень. На вход подается степень power. Случайно выбирается основание, которое будет возводиться в степень power. Создаем переменную c, равную основанию, и переменную k, равную степени power. Далее создаем переменную f для результата. Если переменная k нечетна, f = c, в ином случае f = 1. Далее, в теле цикла do while, число k целочисленно делим на 2, переменную с умножаем саму на себя. Если k нечетно, умножаем result на c, если k не равно 0 итеративно повторяем цикл do while, в ином случае выходим из цикла, и получаем искомое число f. Сложность алгоритма О(log(n)). Экспериментальные данные представлены на Диаграмме 10. Реализация алгоритма представлена на Блок-схеме 11, Коде 11.

Диаграмма 10 Быстрое возведение в степень



Блок-схема 11 Быстрое возведение в степень



Код 12 Быстрое возведение в степень

1. **Классическое быстрое возведение в степень**

Метод классического быстрого возведения в степень. На вход подается степень power. Случайно выбирается основание, которое будет возводиться в степень power. Создаём переменную c, равную основанию, и переменную k, равную степени power. Далее создаем переменную f = 1. Входим в цикл. Если k равен 0, возвращаем число f как искомое. В ином случае проверяем четность числа k. Если четен, умножаем переменную c саму на себя и целочисленно делим k на 2, в ином случае умножаем f на c и вычитаем 1 из k. В обоих случаях возвращаемся в начало цикла. Сложность алгоритма О(log(n)). Экспериментальные данные представлены на Диаграмме 11. Реализация алгоритма представлена на Блок-схеме 12, Коде 13.

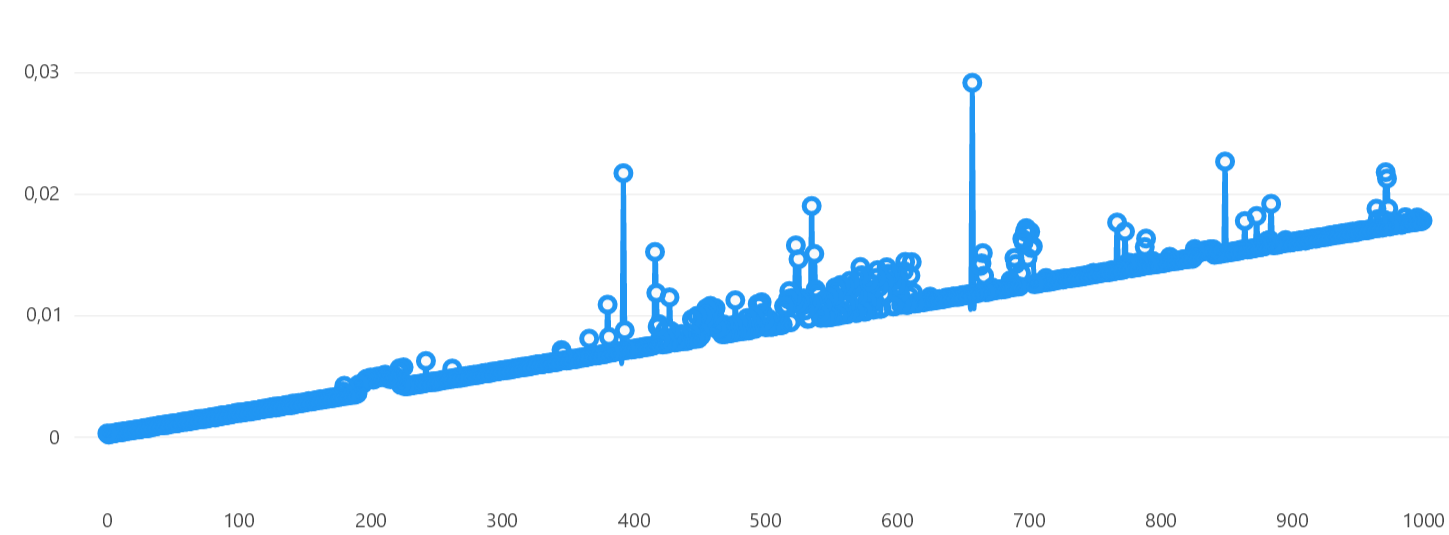
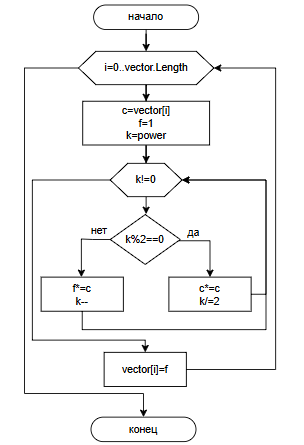
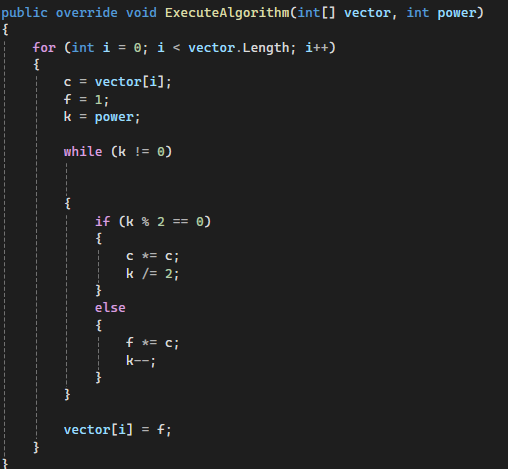


Диаграмма 11 Классический быстрый



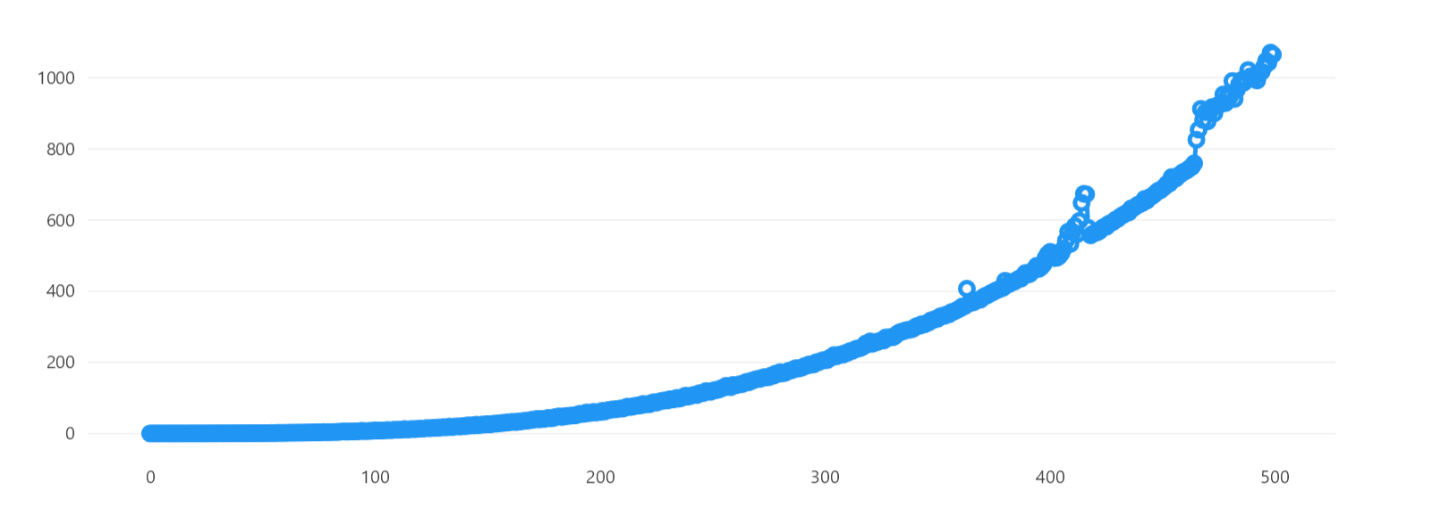
Блок-схема 12 Классический быстрый

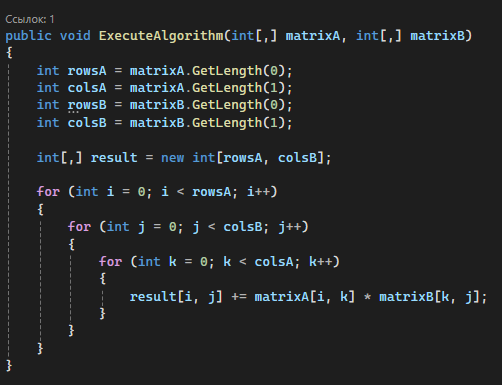


Код 13 Классический быстрый

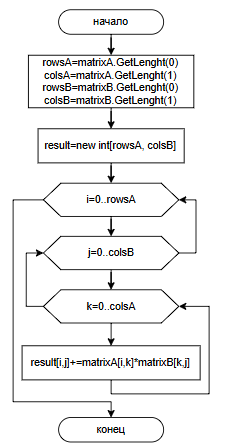
1. **Перемножение матриц**

Метод перемножения матриц. В метод передаются две случайно сгенерированные матрицы и их размерность n. Далее проходимся циклом по матрицам и перемножаем строки первой матрицы со столбцами второй, записываем это произведение в элемент новой матрицы. Сложность алгоритма О(n^2). Экспериментальные данные представлены на Диаграмме 12. Реализация алгоритма представлена на Блок-схеме 13, Коде 14.



Диаграмма 12 Перемножение матриц

Код 14 Перемножение матриц



Блок-схема 13 Перемножение матриц

1. **Stooge Sort**

Сравнение элементов:

Если элемент в конце списка меньше элемента в начале, они меняются местами. Рекурсивная сортировка:

Если в текущем подмножестве массива три или более элементов:

Рекурсивно вызывается сортировка для первых 2/3 массива.

Затем вызывается сортировка для последних 2/3 массива.

И снова вызывается сортировка для первых 2/3 массива.

Сложность алгоритма **O(nlog 3 / log 1.5)**. Экспериментальные данные представлены на Диаграмме 13. Реализация алгоритма представлена на Блок-схеме 14, Коде 15.

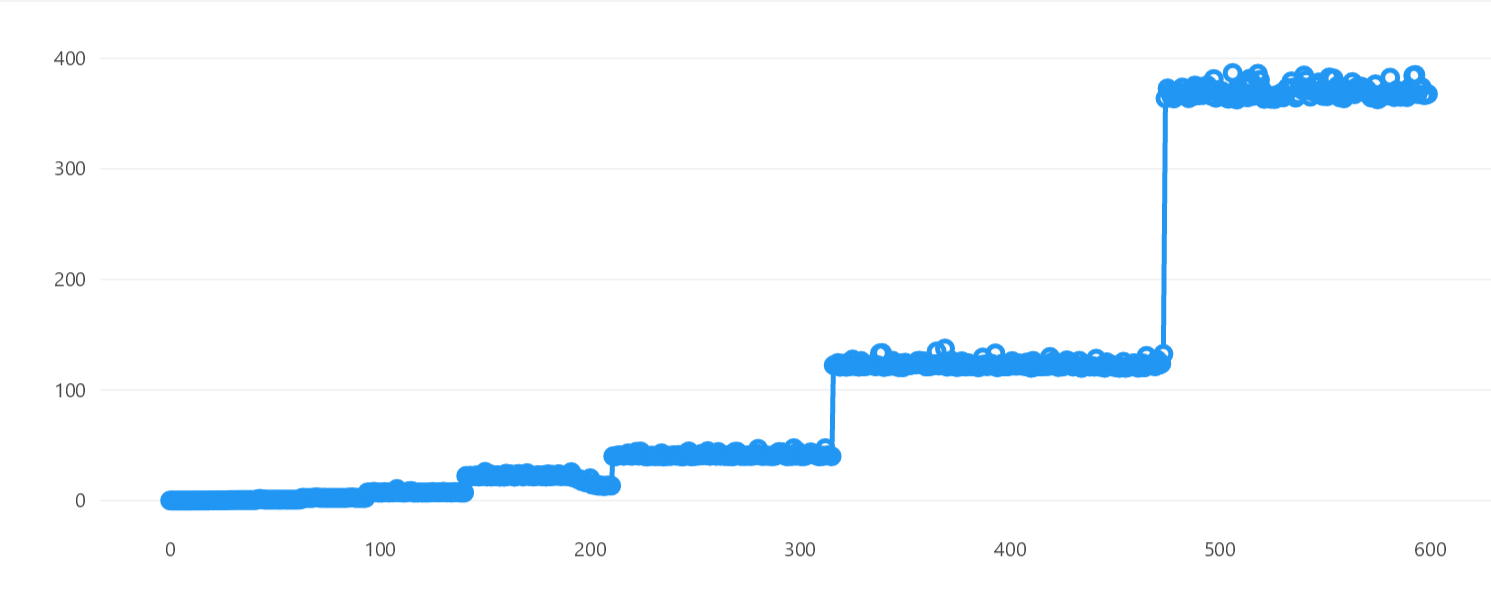
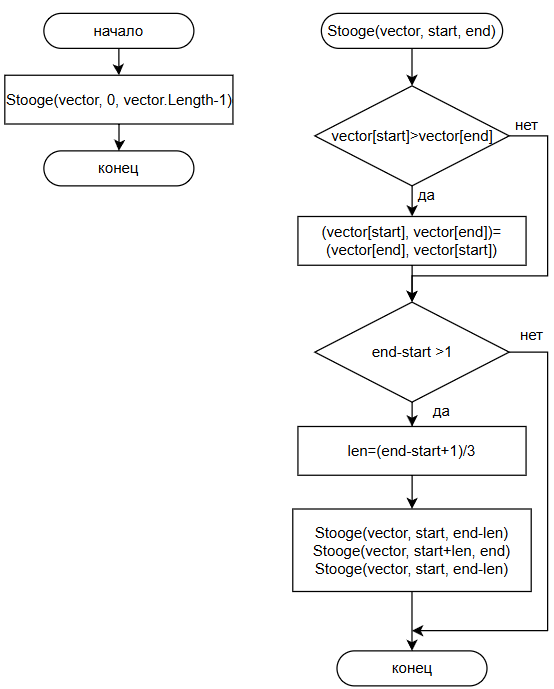
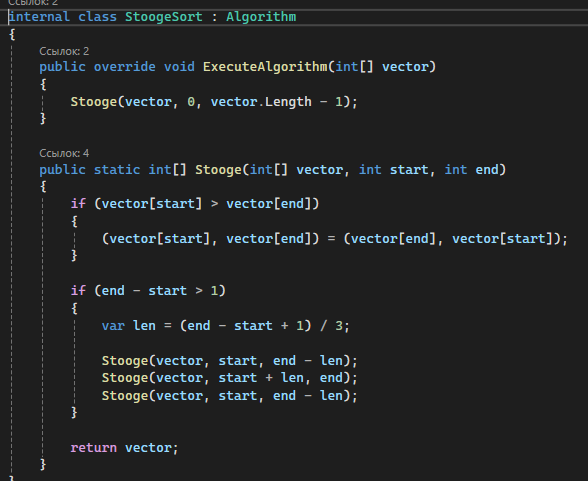


Диаграмма 13 Stooge Sort



Блок-схема 14 Stooge Sort



Код 15 Stooge Sort

1. **Нахождение суммы всех элементов строк в двухмерном массиве**

Простой алгоритм, который получает двухмерный массив и проходит по каждой строке. В каждой строке он складывает все её элементы и записывает результат в новый массив. Итогом работы алгоритма является одномерный массив, содержащий значения суммы элементов каждой строки. Сложность алгоритма **O(n \* m)**, где n — количество строк, а m — количество столбцов. Экспериментальные данные представлены на Диаграмме 14. Реализация алгоритма представлена на Блок-схеме 15, Коде 16.

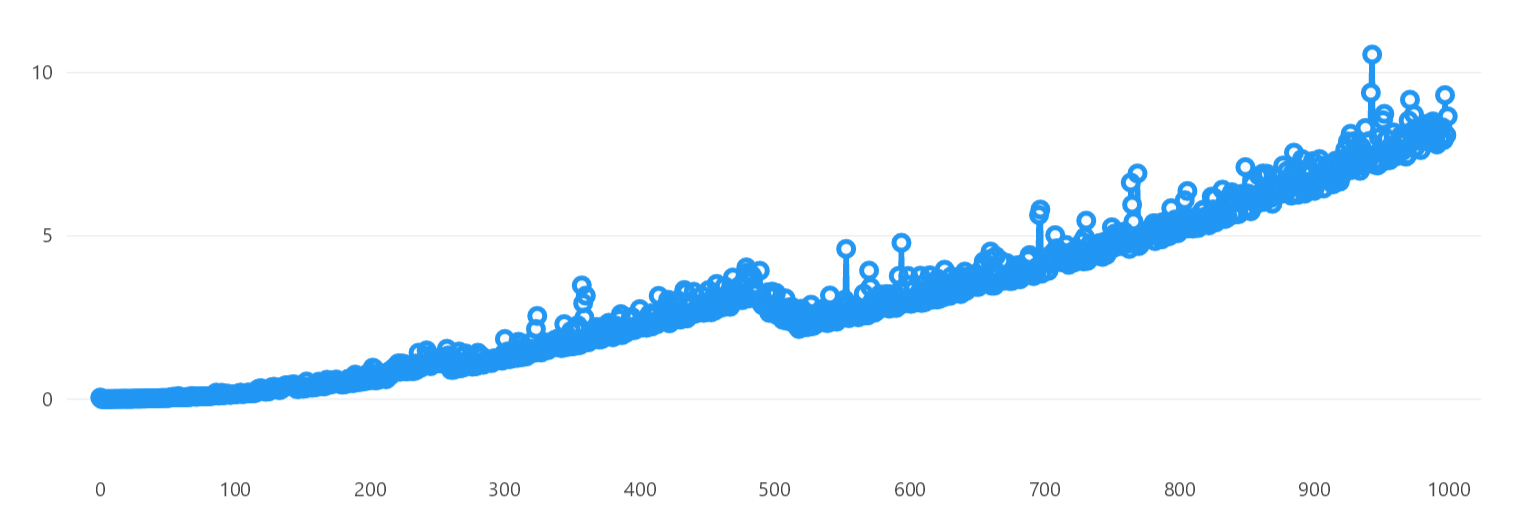
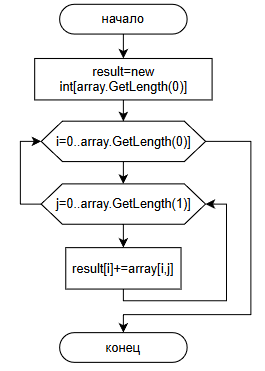
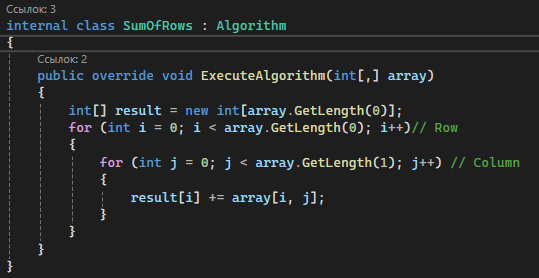


Диаграмма 14 Нахождение суммы всех элементов строк в двухмерном массиве



Блок-схема 15 Нахождение суммы всех элементов строк в двухмерном массиве



Код 16 Нахождение суммы всех элементов строк в двумерном массиве

1. **Cycle Sort**

Это неустойчивый сортировочный алгоритм, работающий за время O(n²). Он эффективен по использованию памяти, так как использует только постоянное количество дополнительной памяти — O(1). Основная идея заключается в минимизации количества записей (перестановок) элементов в массиве, что делает его полезным в сценариях, где количество записей должно быть минимальным. В начале инициализируем элемент как начальную точку и находим куда он подходит и помещаем его туда. Если элемент не находится на своей позиции, алгоритм делает одну перестановку: перемещает текущий элемент на его правильную позицию:

После этого цикл продолжается, и алгоритм находит новую позицию для перемещённого элемента (теперь это item), который был вытеснен.

Экспериментальные данные представлены на Диаграмме 15. Реализация алгоритма представлена на Блок-схеме 16, Коде 17.

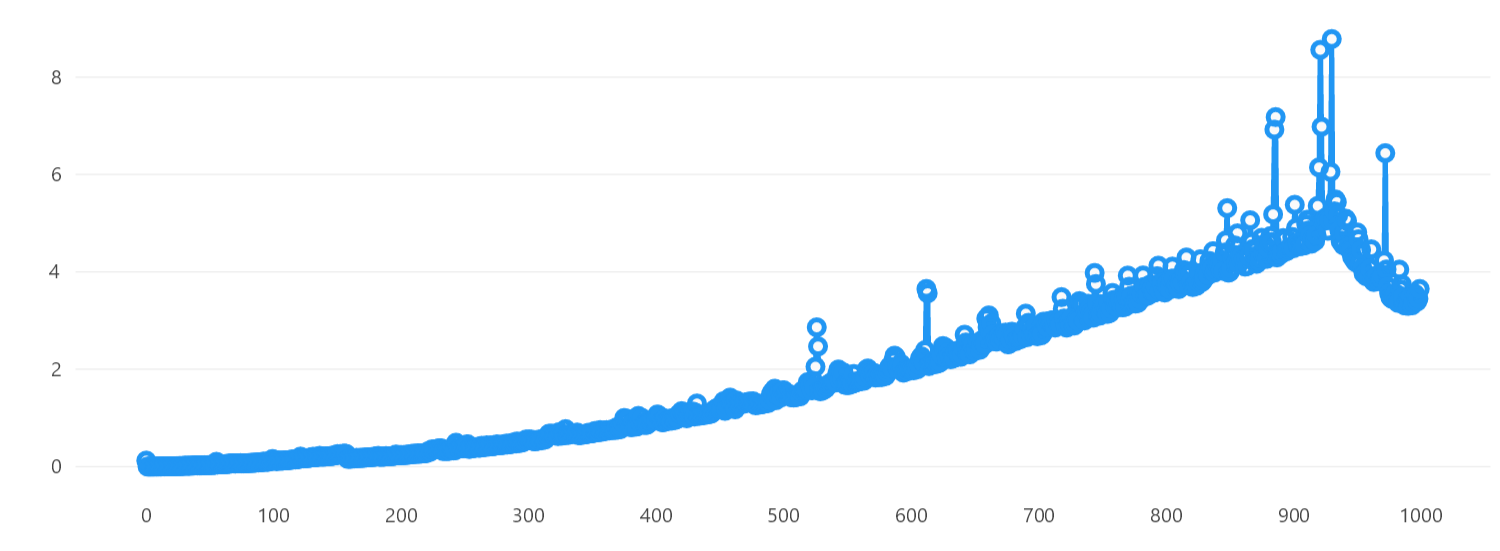
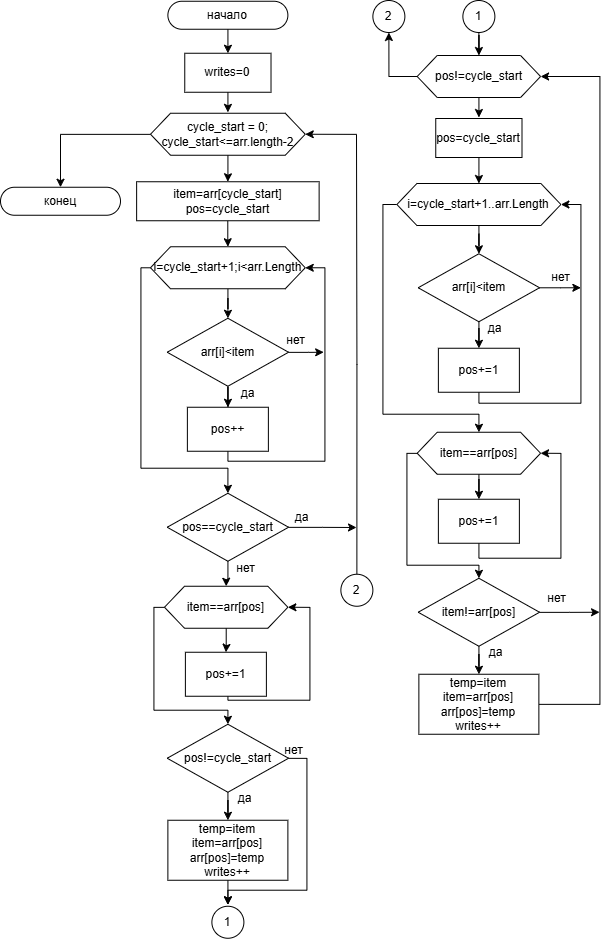
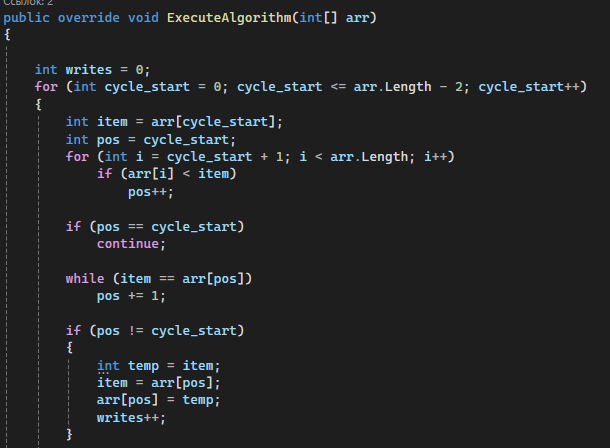


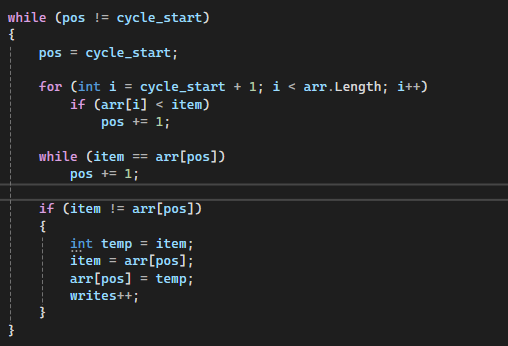
Диаграмма 15 Cycle Sort



Блок-схема 16 Cycle Sort



Код 17 Cycle Sort



Код 18 Cycle Sort