МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

ОТЧЕТ

по лабораторной работе №1

Авторы отчета: Постнов Алексей, ПрИ-202;

Ларюшкин Евгений, ПрИ-202;

Пеньков Фёдор, ПрИ-202;

Проверил: Николаев И.Е.

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

дата оценка

Челябинск 2024 г.

**UML-диаграмма:**

В решении находятся несколько проектов, ключевые из них: Core, Application, Infrastructure.

A screenshot of a chat

Description automatically generated

UML-диаграмма Core, множество реализаций опущены для краткости.

A group of white rectangular boxes with text

Description automatically generated

Краткая UML-диаграмма Application.

Процесс бенчмаркинга алгоритма начинается с того, что пользователю предоставляются возможные параметры для запуска этого процесса. С помощью IInputExecutorProvider находится подходящий IInputExecutor для текущего выбранного алгоритма. Пользователь вводит параметры. Параметры передаются в соответствующий IInputExecutor для конкретного алгоритма. IInputExecutor через DI получает нужные компоненты, необходимые для работы с выбранным алгоритмом.

Далее через метод SetInput параметры пользователя маппятся на нужные для бенчмаркинга объекты (с использованием компонента IMapper). После этого вызывается метод ValidateInput, который проверяет корректность введённых данных, например, чтобы количество итераций было положительным числом (используются валидаторы). Если данные валидны, запускается метод PrepareDataAsync, который готовит входные данные для тестирования алгоритма с заданными параметрами (с помощью IDataGenerator). Затем вызывается метод RunAsync, который асинхронно запускает бенчмаркинг алгоритма с использованием ICancellableAlgorithmRunner.

Во время запуска каждый вызов алгоритма замеряется по времени, а результаты сохраняются в список из AlgorithmRunResult. По завершении всех итераций формируется BenchmarkResult, который содержит общее время выполнения всех запусков и информацию о том, было ли выполнение отменено. После этого результаты бенчмарка можно использовать для визуализации, например, в виде графика, который показывает зависимость времени выполнения от параметров.

1. **Постоянная функция**

Метод постоянной функции. Из метода возвращается значение 1. Сложность алгоритма О(1). Экспериментальные данные представлены на Диаграмме 1. Реализация алгоритма представлена на Блок-схеме 1, Коде 1.

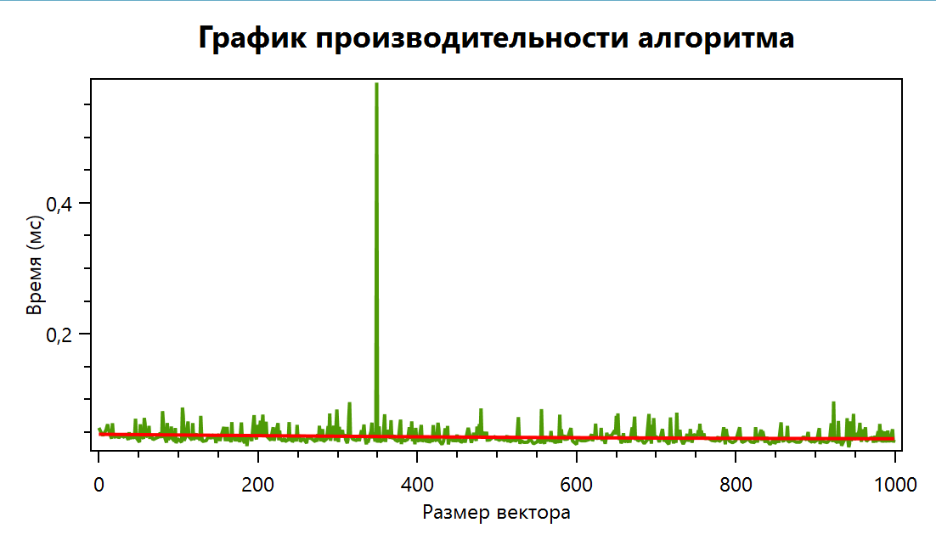
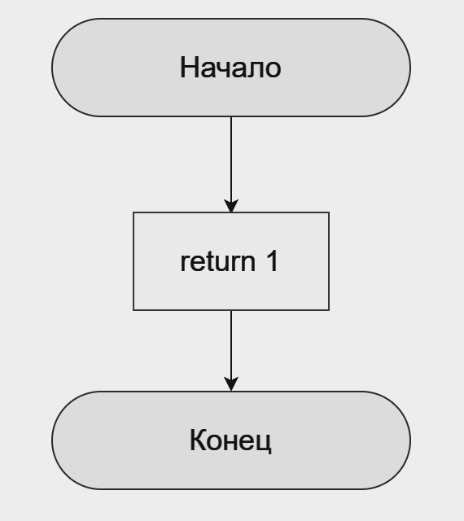
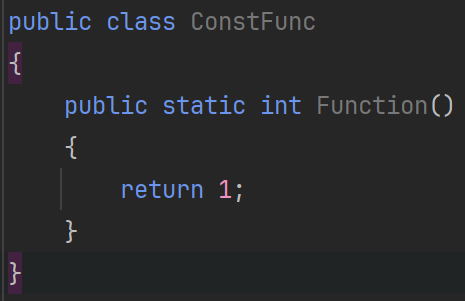


Диаграмма 1. Постоянная функция



Блок-схема 1. Постоянная функция



Код 1. Постоянная 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.

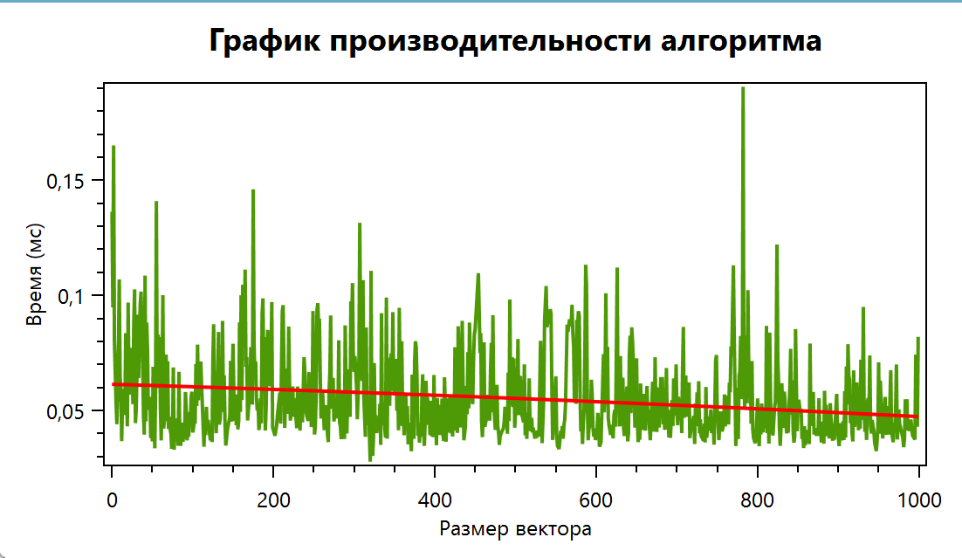
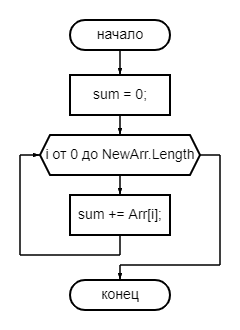
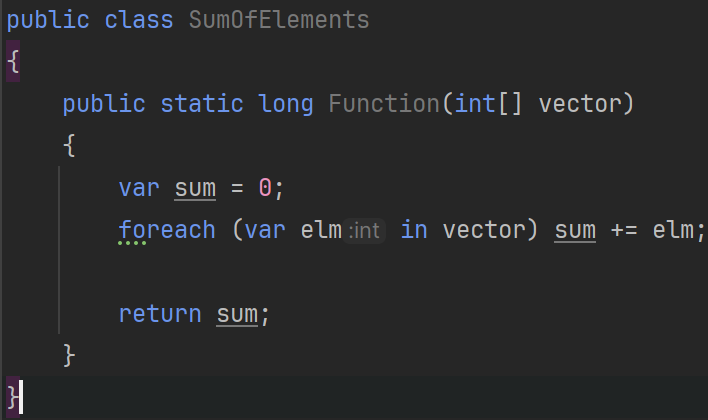


Диаграмма 2. Сумма элементов



Блок-схема 2. Сумма элементов



Код 2. Сумма элементов

1. **Произведение элементов**

Метод произведения элементов массива. Создаём переменную result = 1, с помощью цикла обходим массив, переменную result 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.

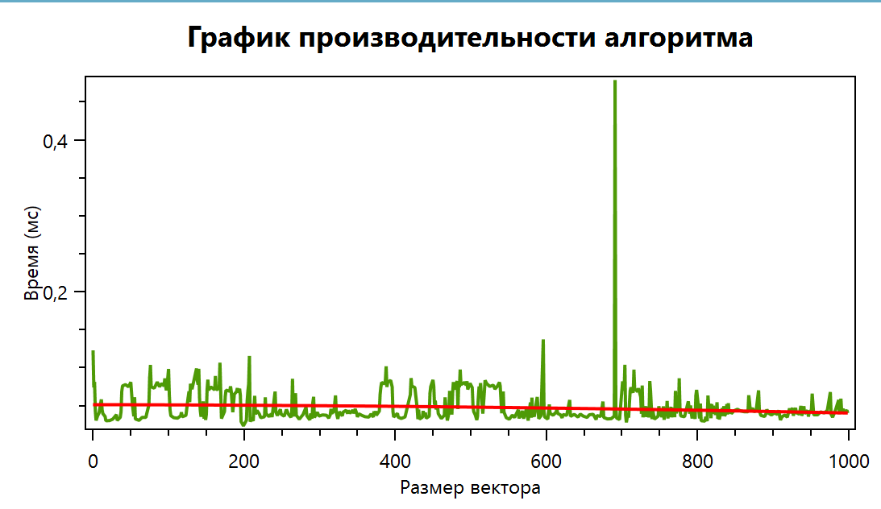
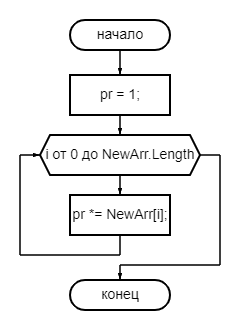
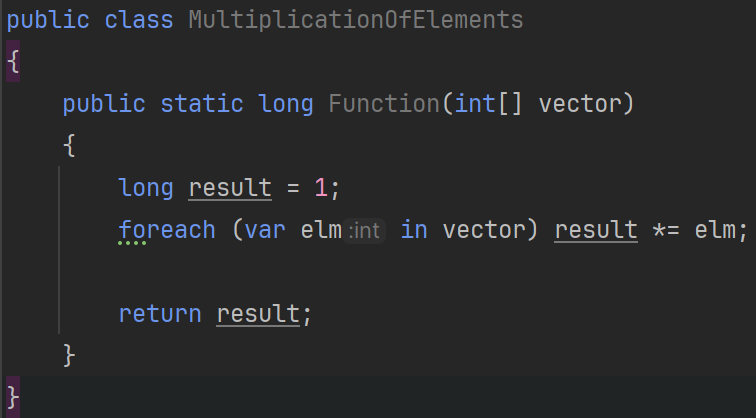


Диаграмма 3. Произведение элементов



Блок-схема 3. Произведение элементов



Код 3. Произведение элементов

1. **Метод Горнера**

Передаем в метод массив, параметр variable и номер шага iterationNumber. В начале метода проверяется: не вышла ли программа за пределы массива, если нет, то запускаем рекурсию, возвращая элемент массива под номером vector[iterationNumber] + variable \* Function(vector, variable, iterationNumber + 1), т.е. шаг увеличивается на 1. Сложность алгоритма О(n). Экспериментальные данные представлены на Диаграмме 4. Реализация алгоритма представлена на Блок-схеме 4, Коде 4.

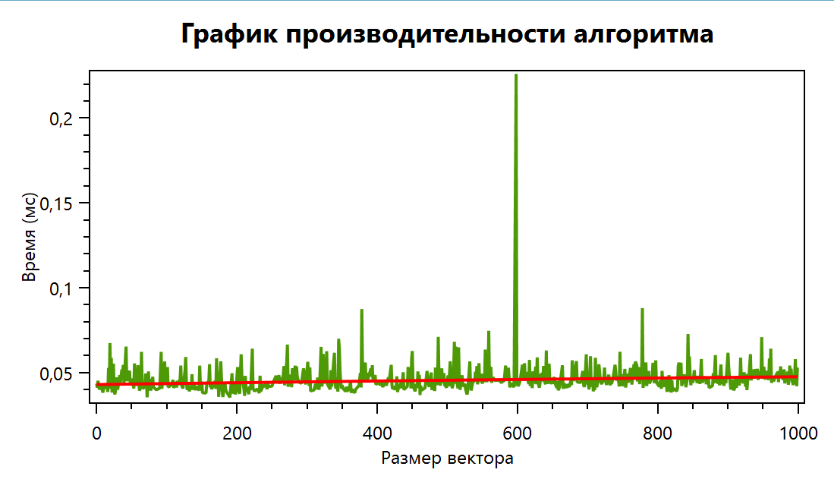
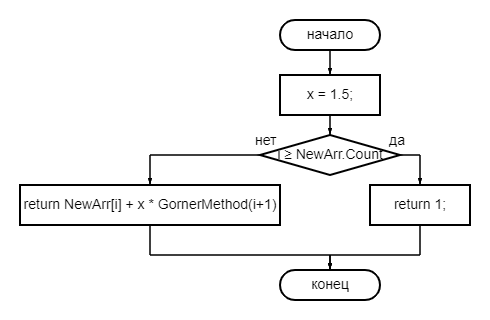
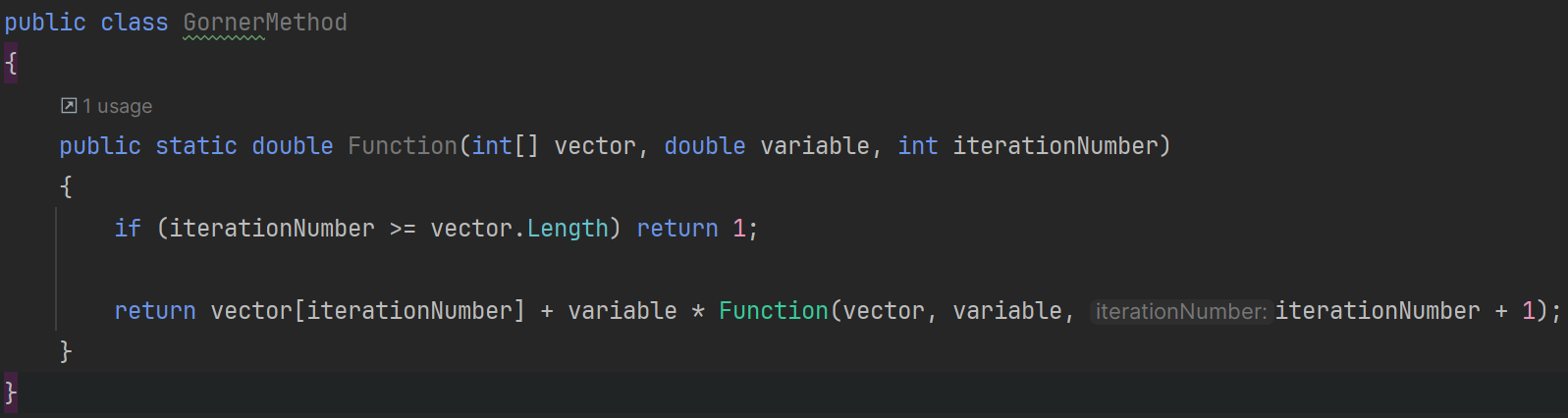


Диаграмма 4. Метод Горнера



Блок-схема 4. Метод Горнера



Код 4. Метод Горнера

1. **Прямое вычисление**

Метод прямого вычисления полинома Горнера. Создаём переменную result = 0, проходимся циклом по всем элементам массива. Внутри цикла к переменной result мы прибавляем элемент массива умноженный на variable в степени i - 1. Возвращаем result. Сложность алгоритма О(n). Экспериментальные данные представлены на Диаграмме 5. Реализация алгоритма представлена на Блок-схеме 5, Коде 5.

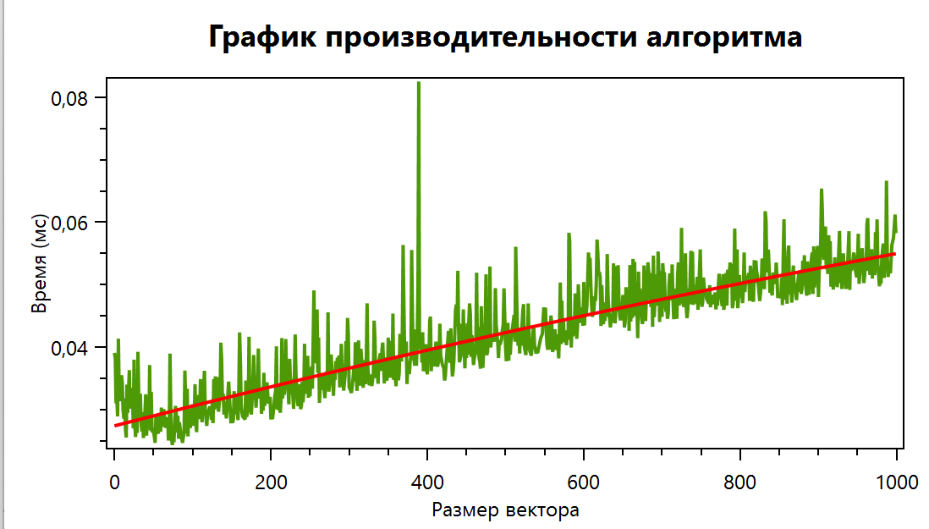
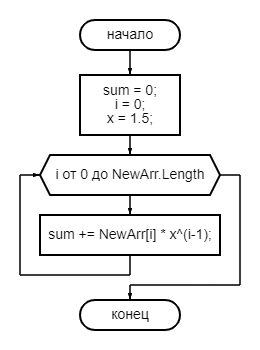
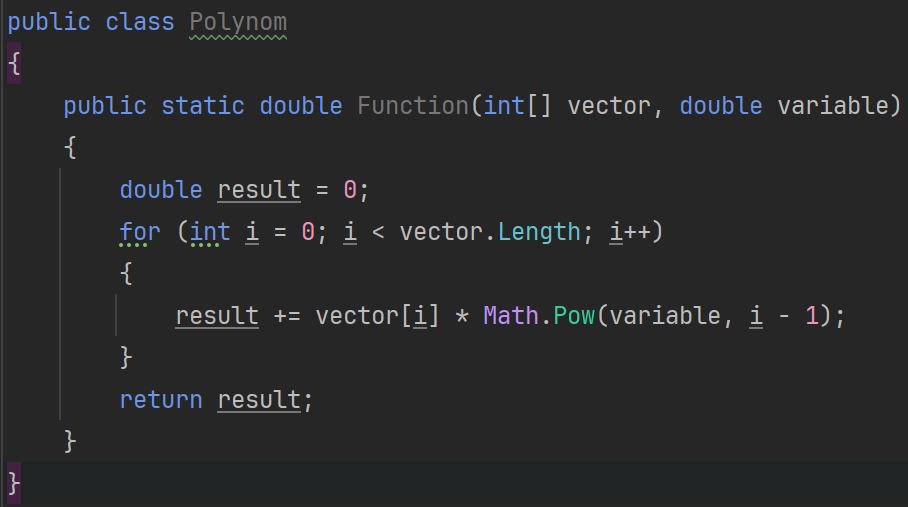


Диаграмма 5. Прямое вычисление



Блок-схема 5. Прямое вычисление



Код 5. Прямое вычисление

1. **Сортировка пузырьком**

Метод сортировки Пузырьком. Запускаем цикл от i = 0 до длины массива с шагом 1, затем запускаем цикл от j = 0 до длинны массива - 1 с шагом 1. Сравниваются переменные под номером i и j, и если элемент с индексом j больше элемента с индексом j + 1, то эти элементы меняются местами. Сложность алгоритма О(n^2). Экспериментальные данные представлены на Диаграмме 6. Реализация алгоритма представлена на Блок-схеме 6, Коде 6.

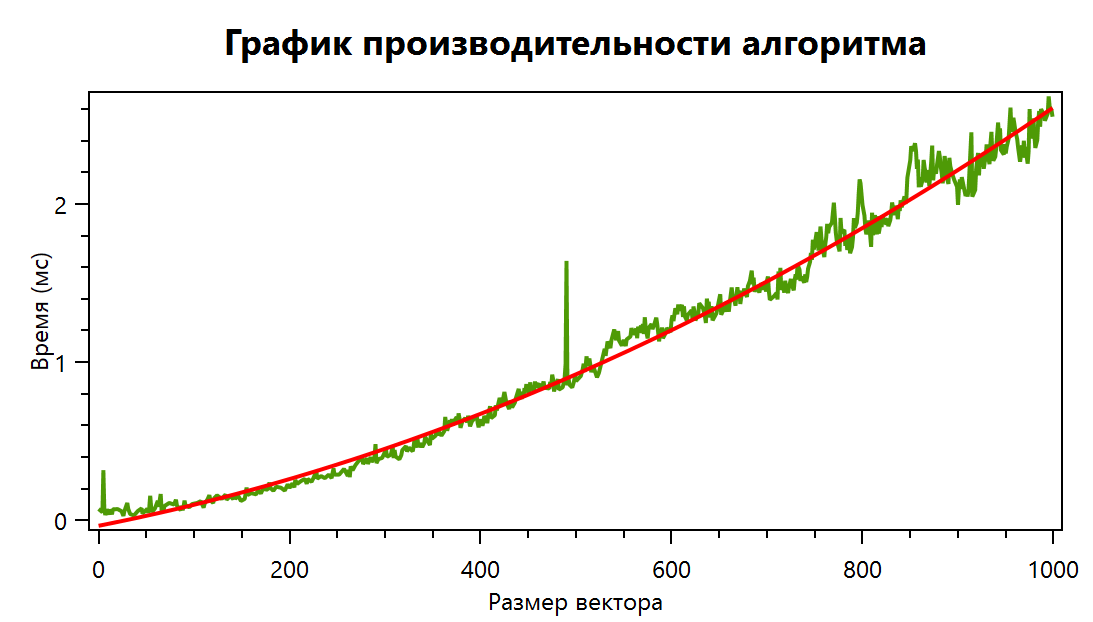
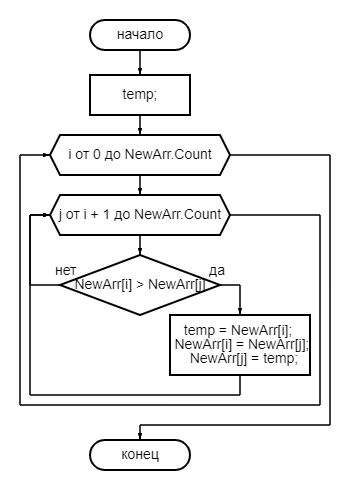
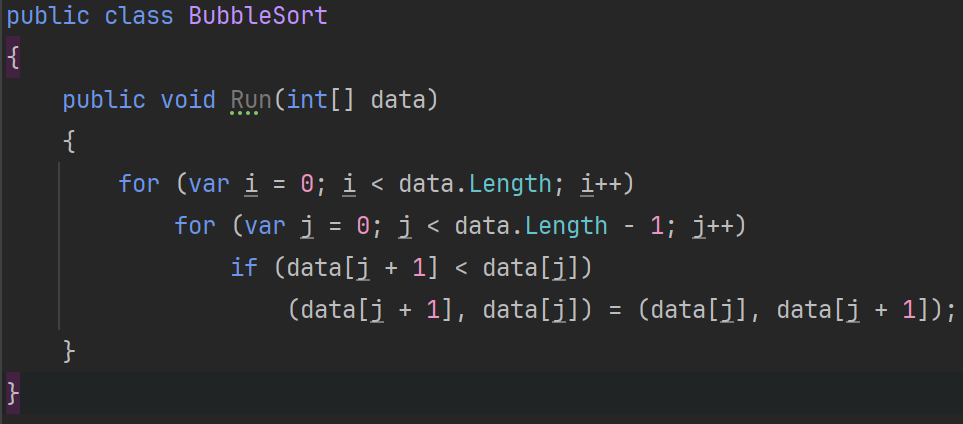


Диаграмма 6. Сортировка пузырьком



Блок-схема 6. Сортировка пузырьком



Код 6. Сортировка пузырьком

1. **Быстрая сортировка**

Выбрать из массива некоторый опорный элемент pivot. 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 7. Реализация алгоритма представлена на Блок-схеме 7, Коде 7.

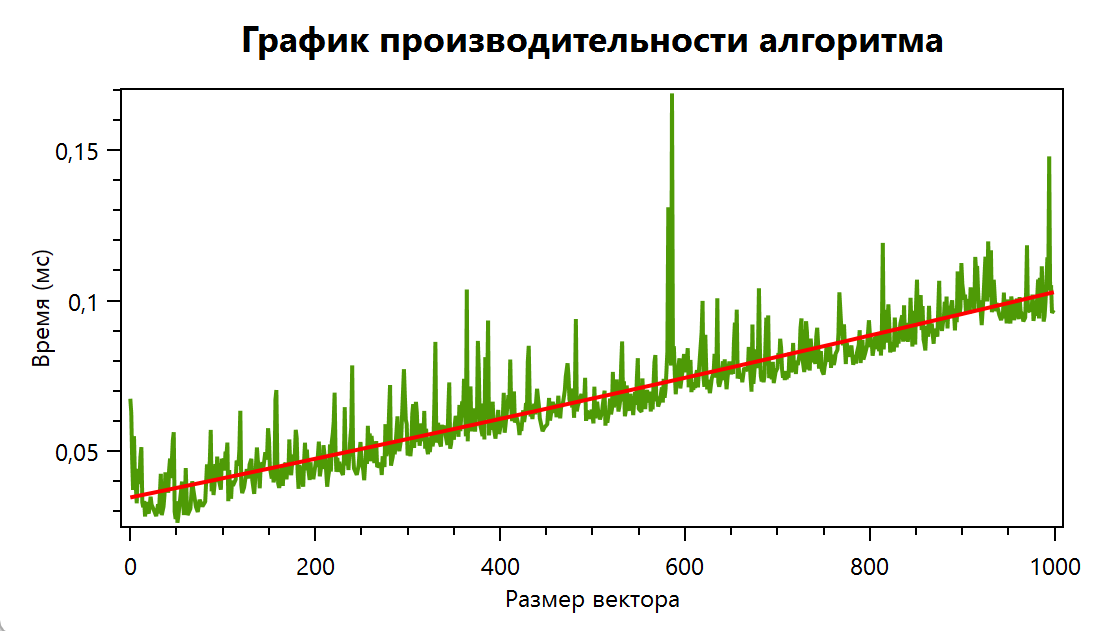
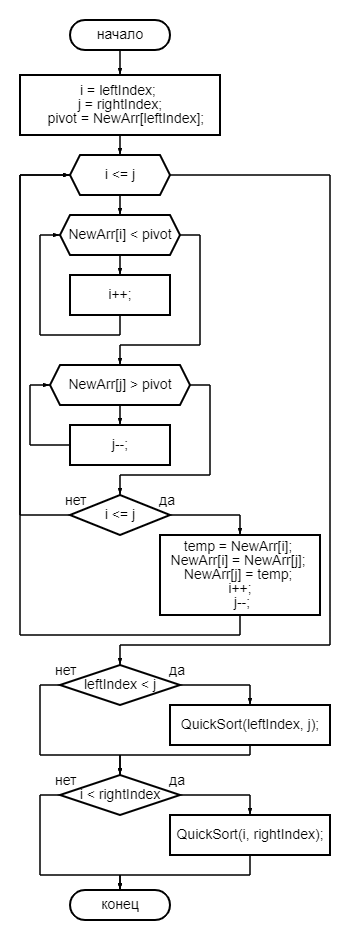
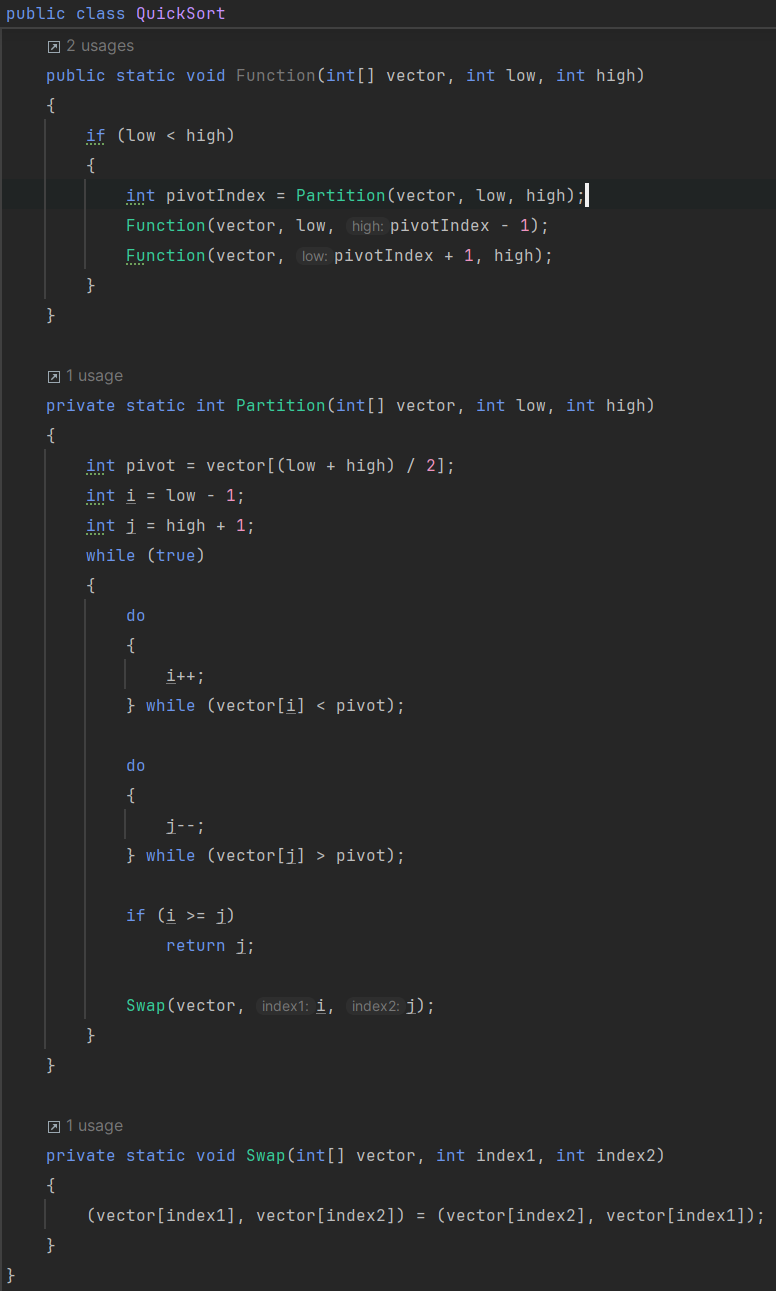


Диаграмма 7. Быстрая сортировка



Блок-схема 7. Быстрая сортировка



Код 7. Быстрая сортировка

1. **Timsort**

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 8. Реализация алгоритма представлена на Блок-схеме 8, 9, 10, Коде 8, 9.

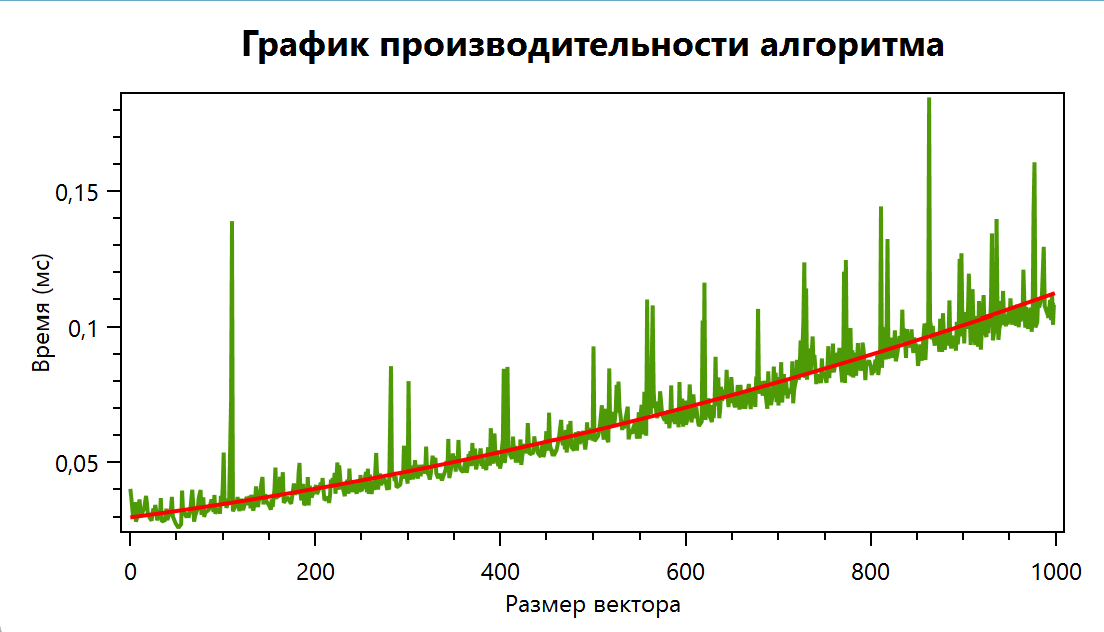
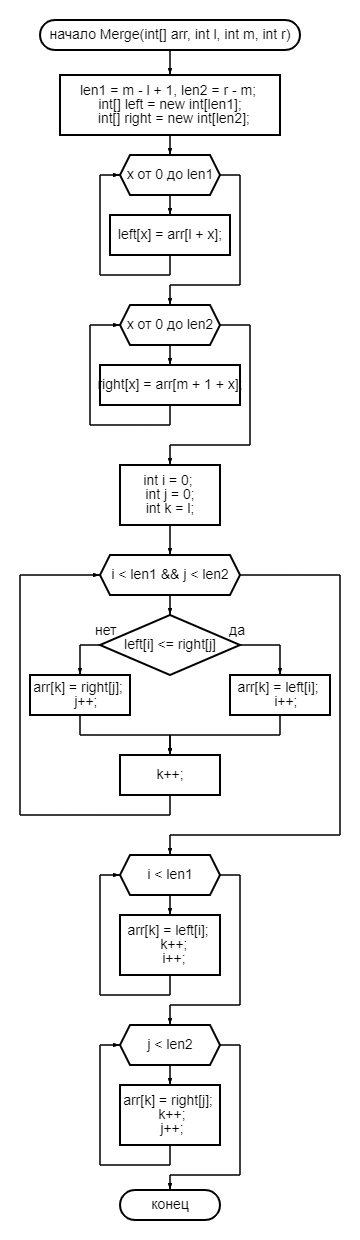
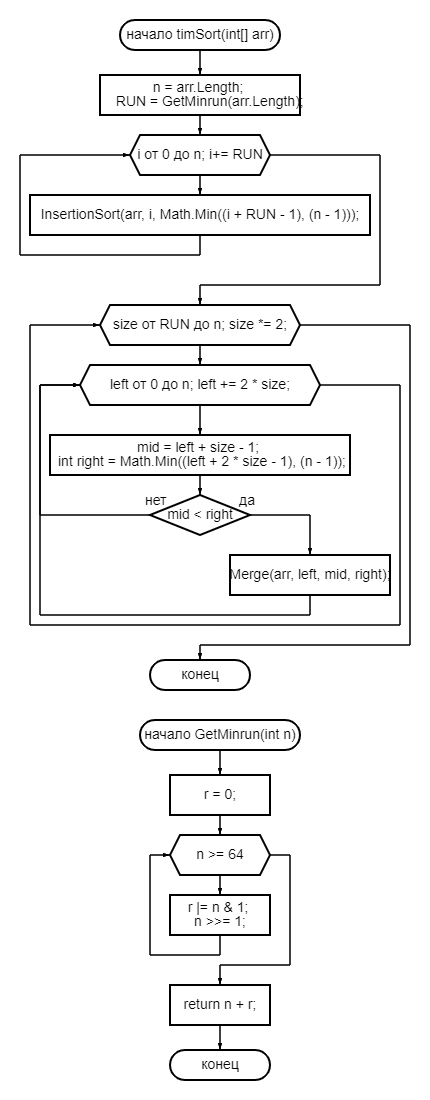
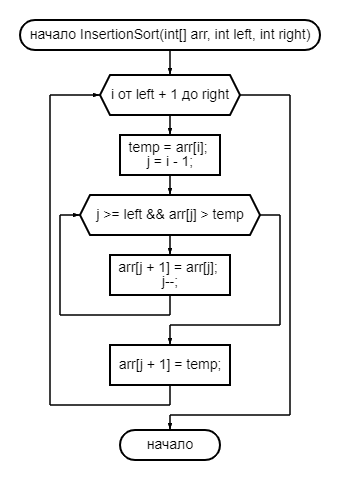


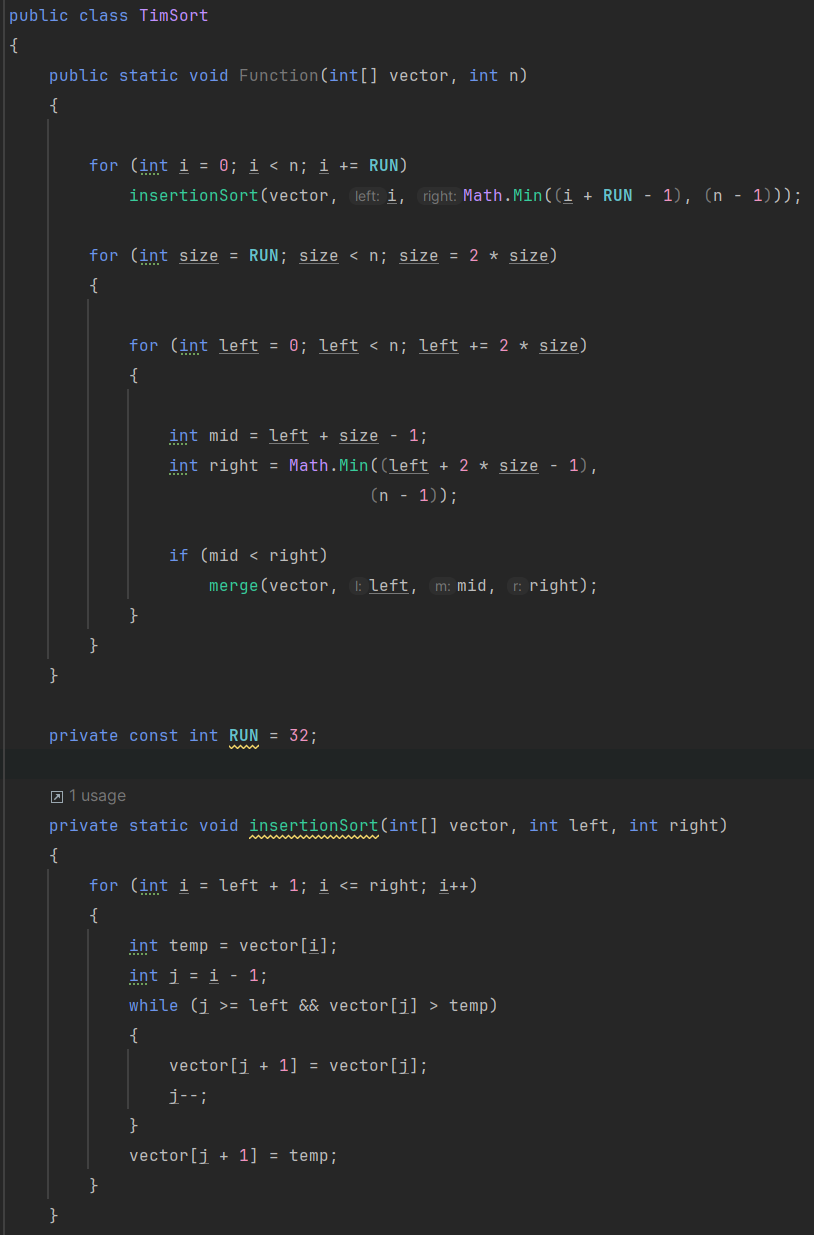
Диаграмма 8. Timsort



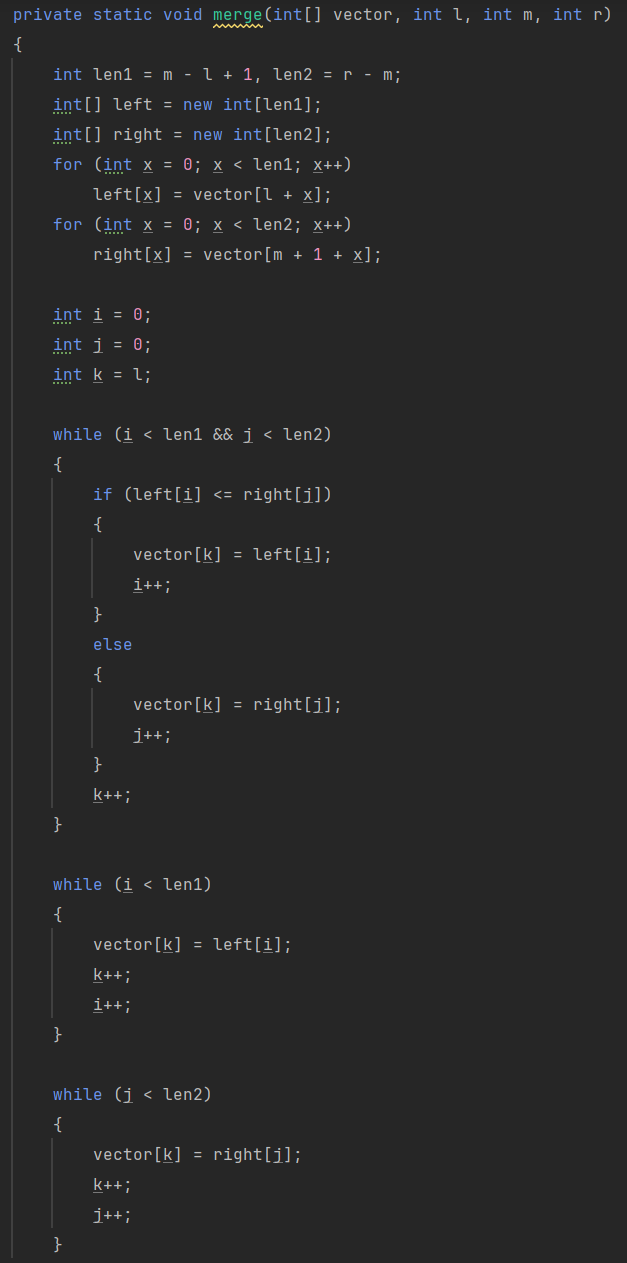
Блок-схема 8. Timsort Блок-схема 9. Timsort



Блок-схема 10. Timsort



Код 8. Timsort



Код 9. Timsort

1. **Простое возведение в степень**

Метод простого возведения в степень. На вход подается коллекция степеней vector и число, возводимое в степень vector[i]. Сложность алгоритма О(n).

Экспериментальные данные представлены на Диаграмме 9. Реализация алгоритма представлена на Блок-схеме 11, Коде 10.

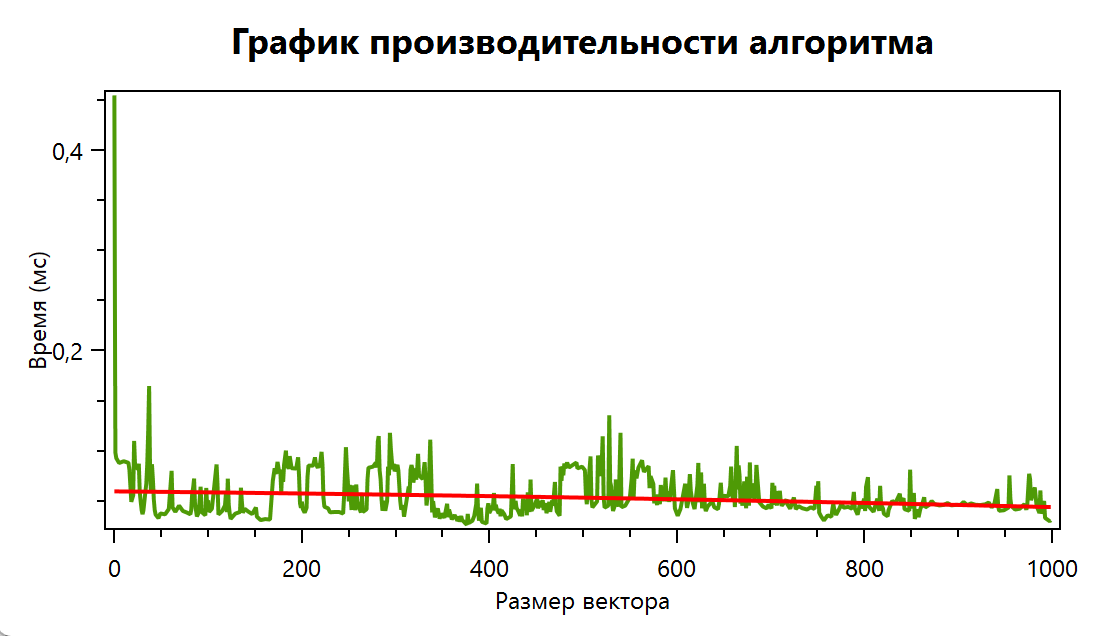
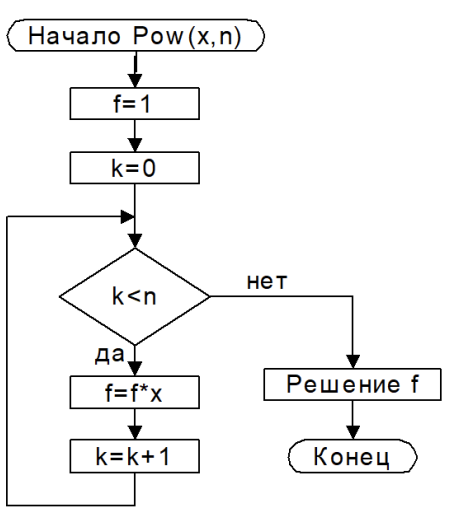
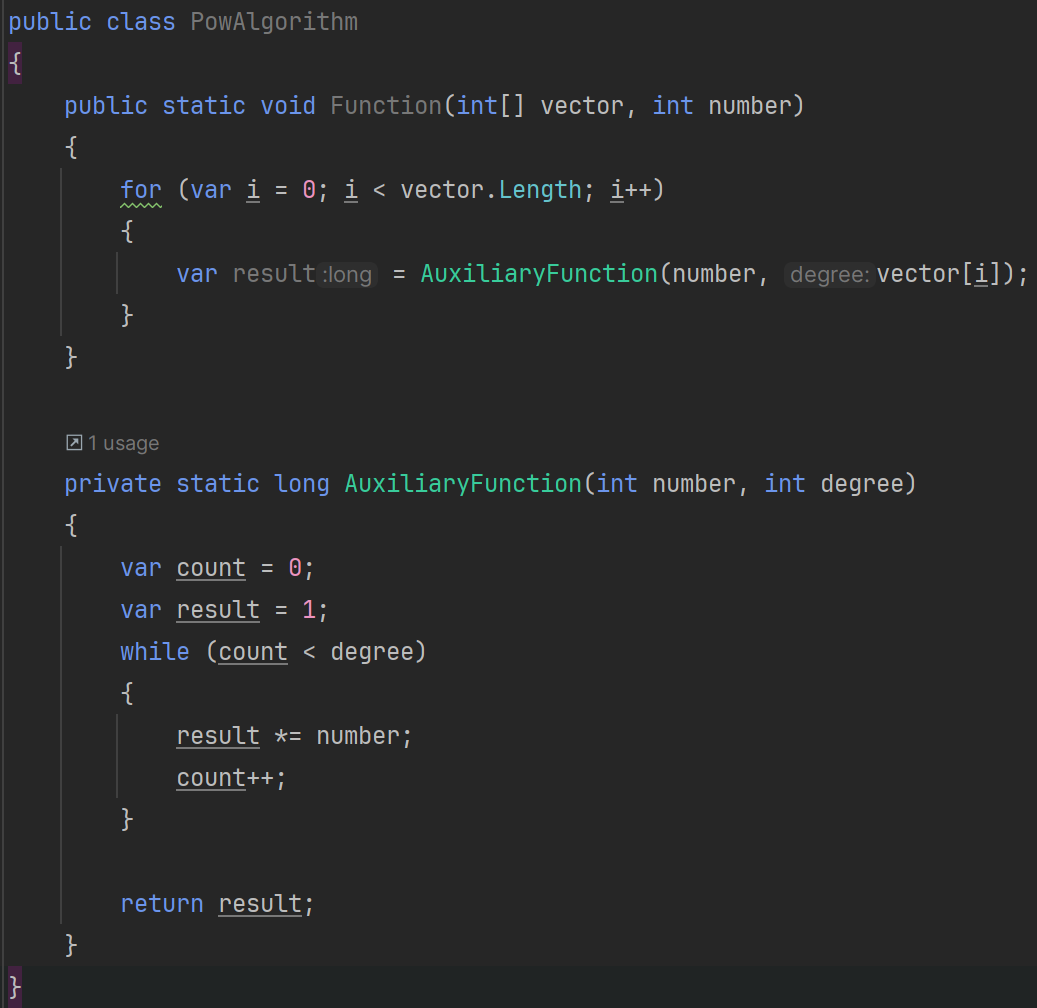


Диаграмма 9. Простое возведение в степень



Блок-схема 11. Простое возведение в степень



Код 10. Простое возведение в степень

1. **Рекурсивное возведение в степень**

Метод рекурсивного возведения в степень. На вход подается коллекция степеней. В случае, если степень degree равна 0, возвращаем 1, иначе возвращаем произведение основания на эту же функцию AuxiliaryFunction, в которой degree уменьшена на единицу. Таким образом мы накапливаем произведение числа n самого на себя ровно degree раз, что эквивалентно возведению основания в степень degree. Сложность алгоритма О(n). Экспериментальные данные представлены на Диаграмме 10. Реализация алгоритма представлена на Блок-схеме 12, Коде 11.

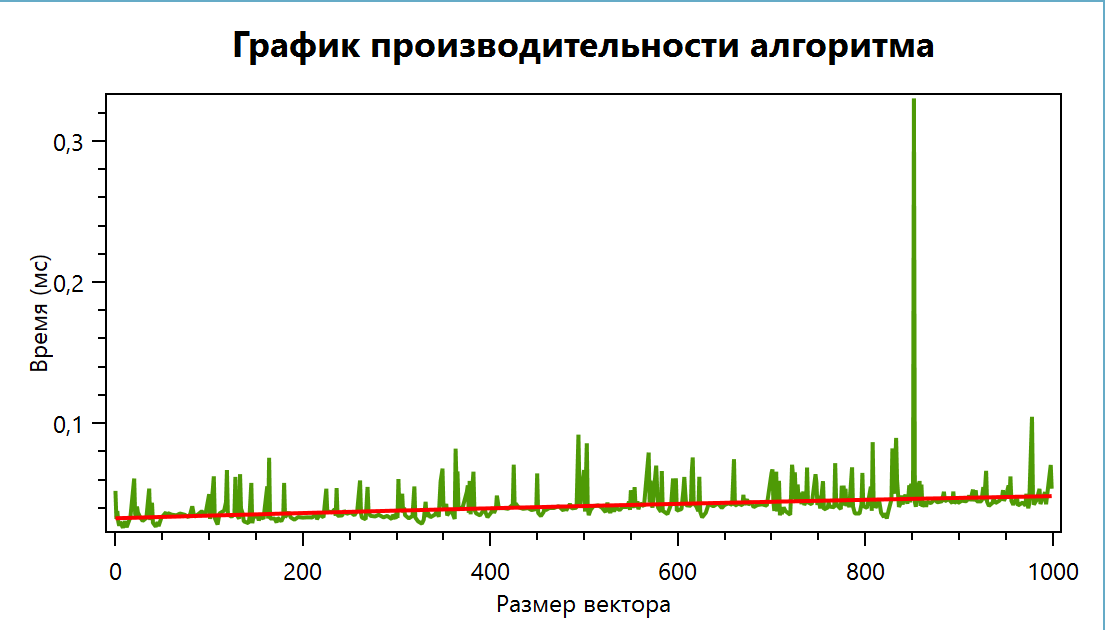
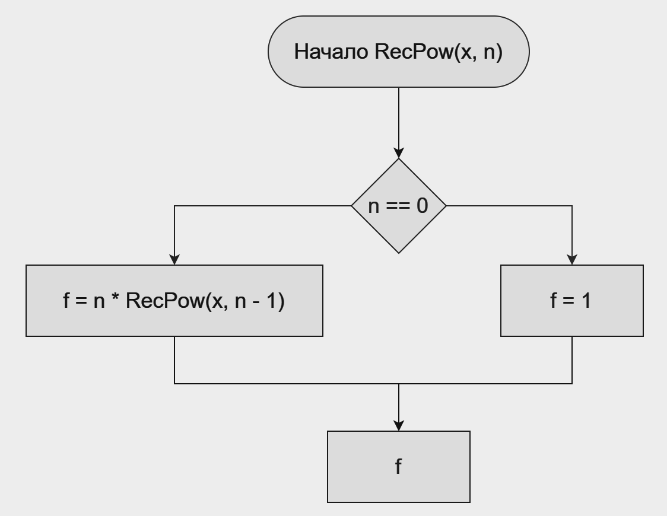
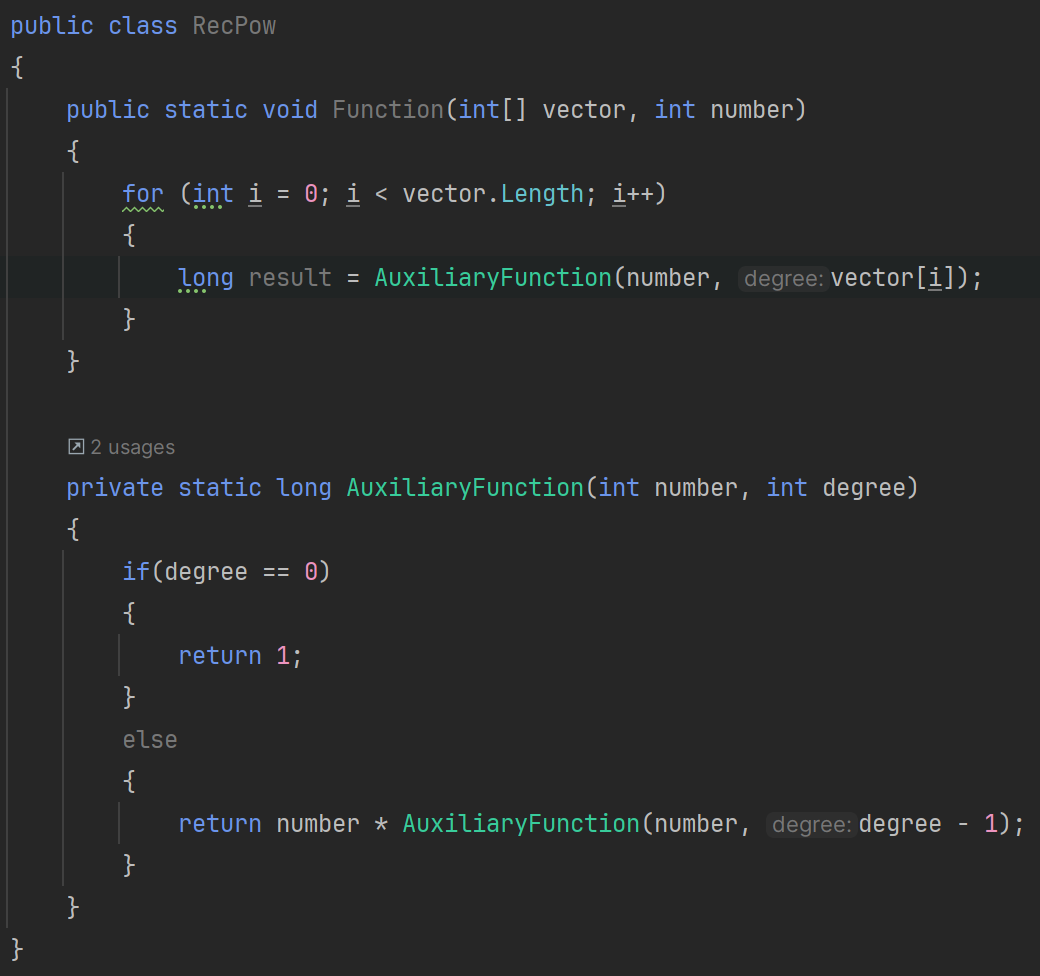


Диаграмма 10. Рекурсивное возведение в степень

****

Блок-схема 12

****

Код 11. Рекурсивное возведение в степень

1. **Рекурсивное возведение (+ быстрое возведение) в степень**

Метод рекурсивного возведения в степень, совмещённый с быстрым возведением. На вход подается коллекция степеней. В случае, если степень degree равна 0, возвращаем 1, иначе возвращаем произведение основания на эту же функцию AuxiliaryFunction, в которой degree уменьшаем в половину. Сложность алгоритма O(log n). Экспериментальные данные представлены на Диаграмме 10. Реализация алгоритма представлена на Блок-схеме 13, Коде 12.

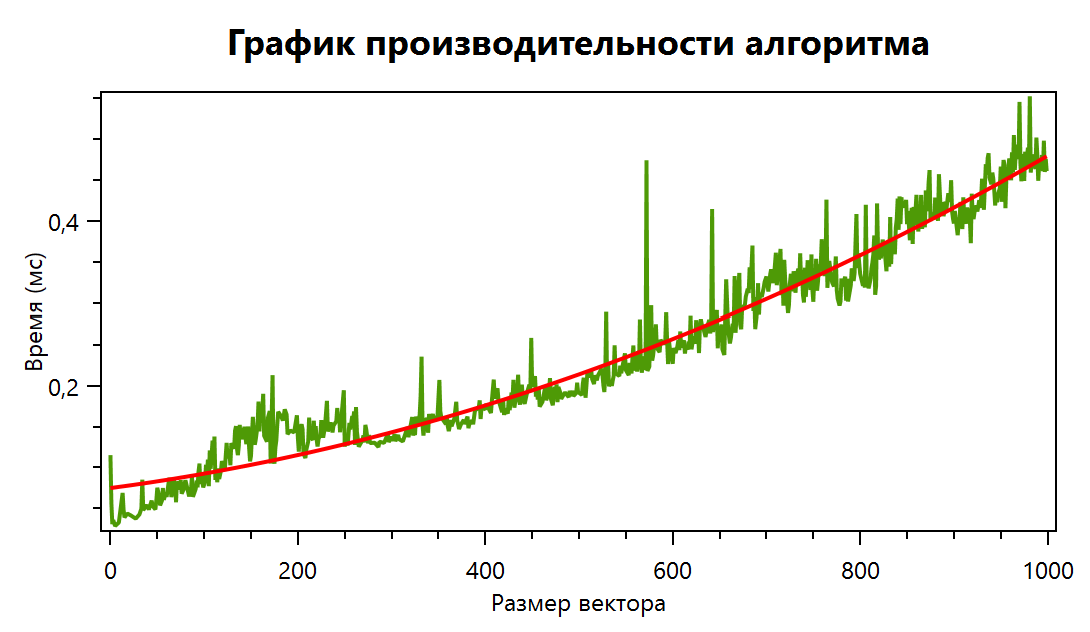
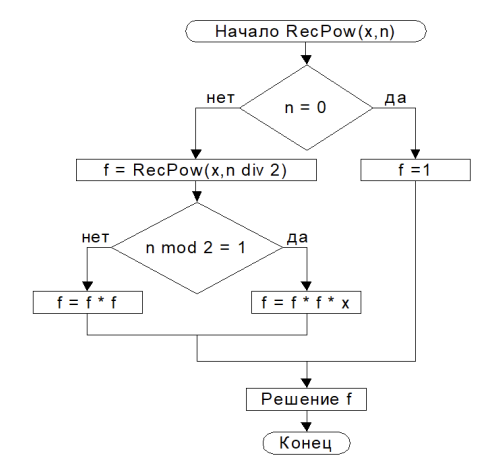
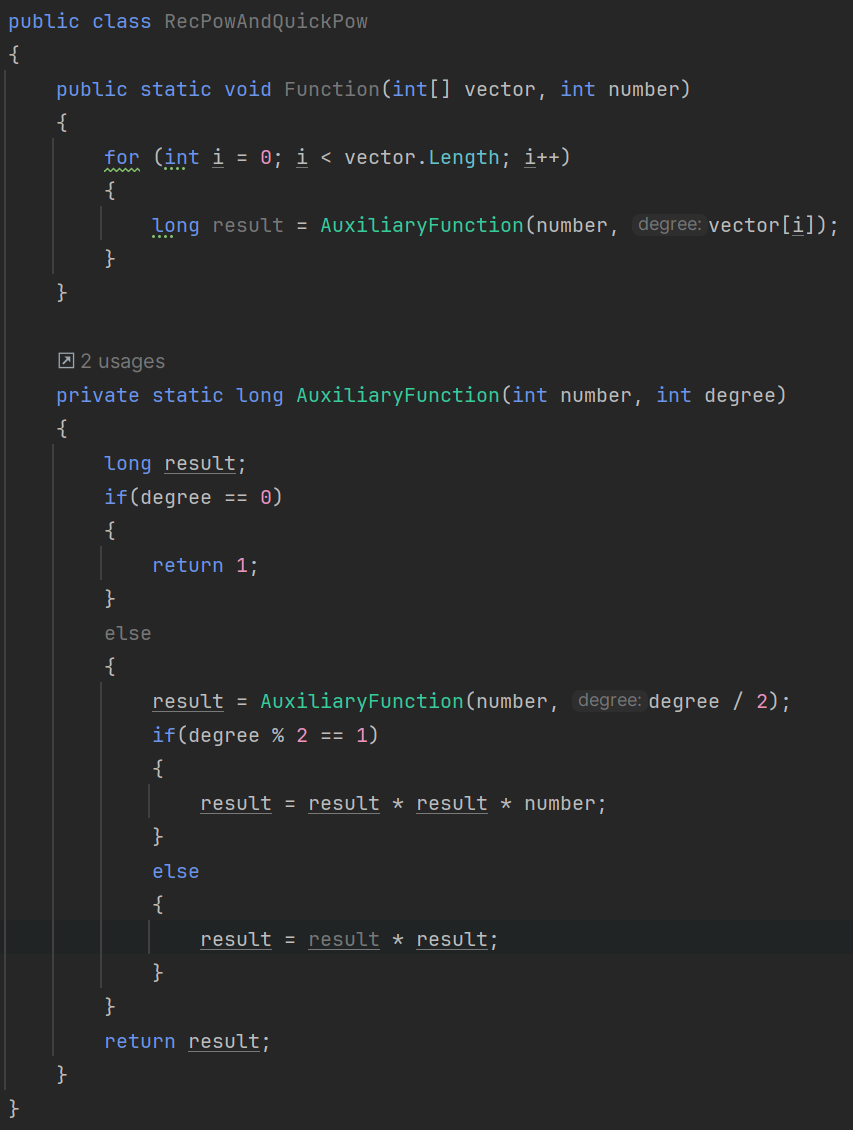


Диаграмма 11. Рекурсивное возведение в степень с разным основанием



Блок-схема 13. Рекурсивное + быстрое возведениев степень



Код 12. Рекурсивное + быстрое возведение в степень

1. **Быстрое возведение в степень**

На вход подается коллекция степеней. Создаём переменную temp, равную основанию, и переменную count, равную степени degree. Далее создаем переменную result = 1. Входим в цикл if. Если count равен 0, возвращаем число как искомое. В ином случае проверяем четность числа count. Если четно, умножаем переменную саму на себя и целочисленно делим count на 2, в ином случае умножаем result на temp и уменьшаем count на 1. В обоих случаях возвращаемся в начало цикла. Сложность алгоритма О(log(n)). Экспериментальные данные представлены на Диаграмме 12. Реализация алгоритма представлена на Блок-схеме 14, Коде 14.

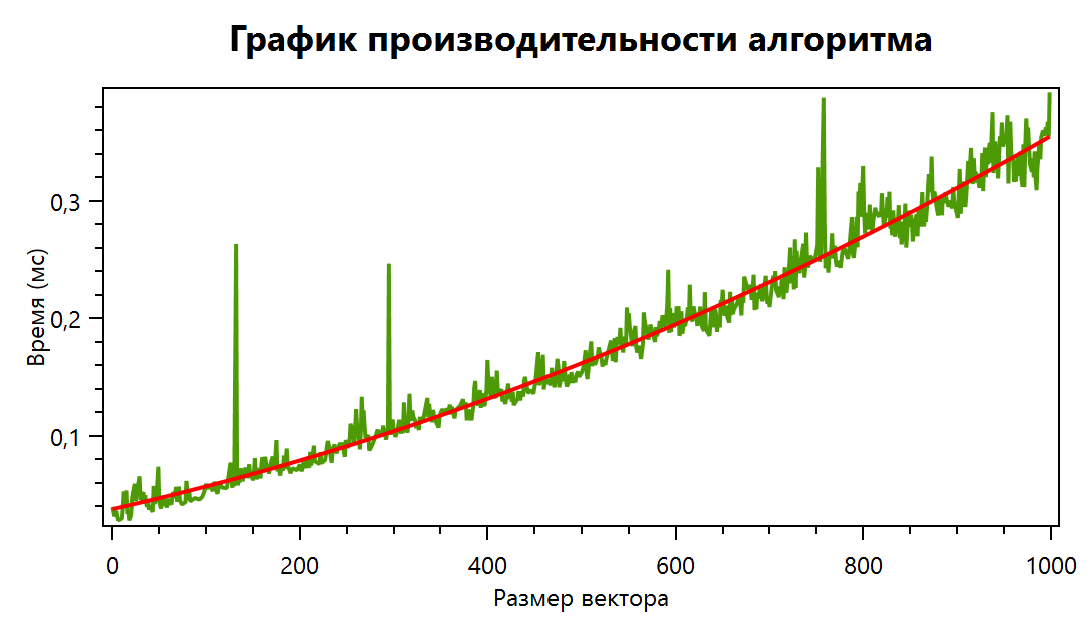
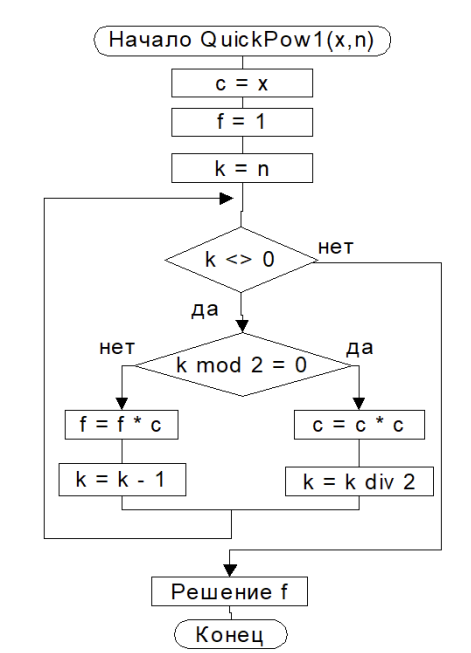
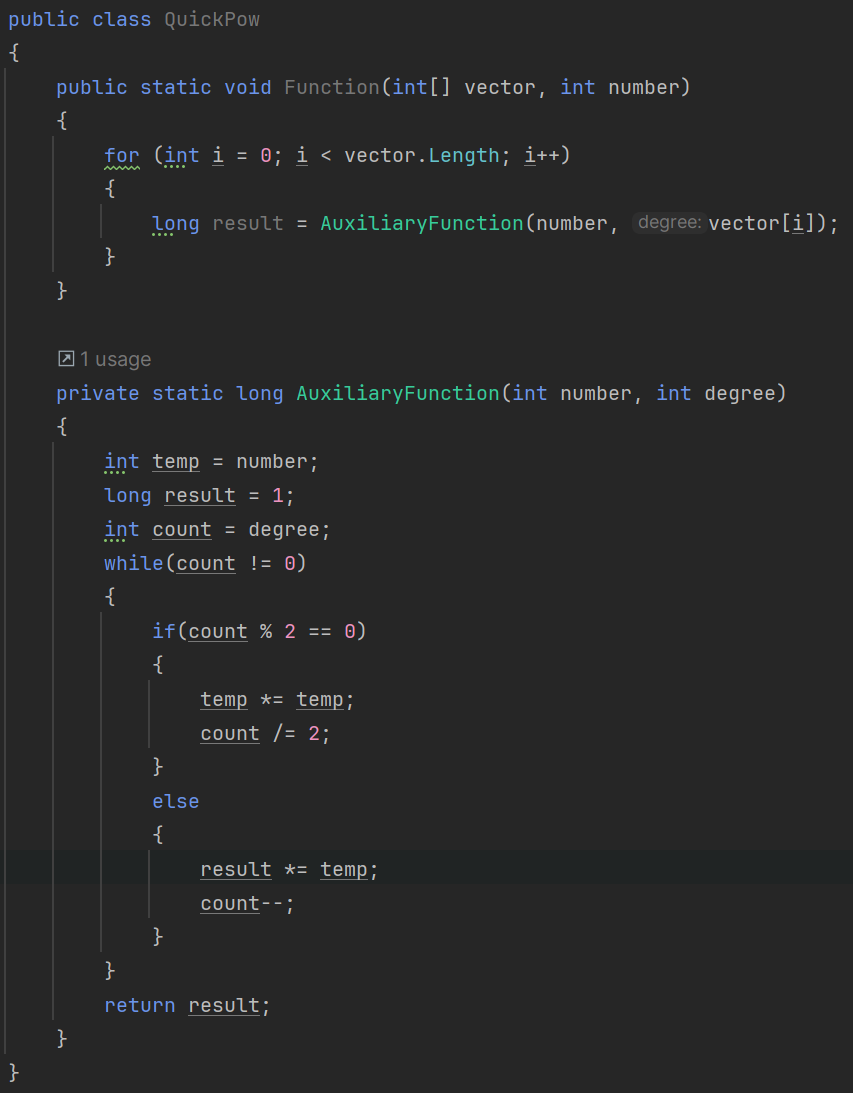


Диаграмма 12. Быстрое возведение в степень



Блок-схема 14. Быстрое возведение в степень



Код 13. Быстрое возведение в степень

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 13. Реализация алгоритма представлена на Блок-схеме 15, Коде 14, 15.

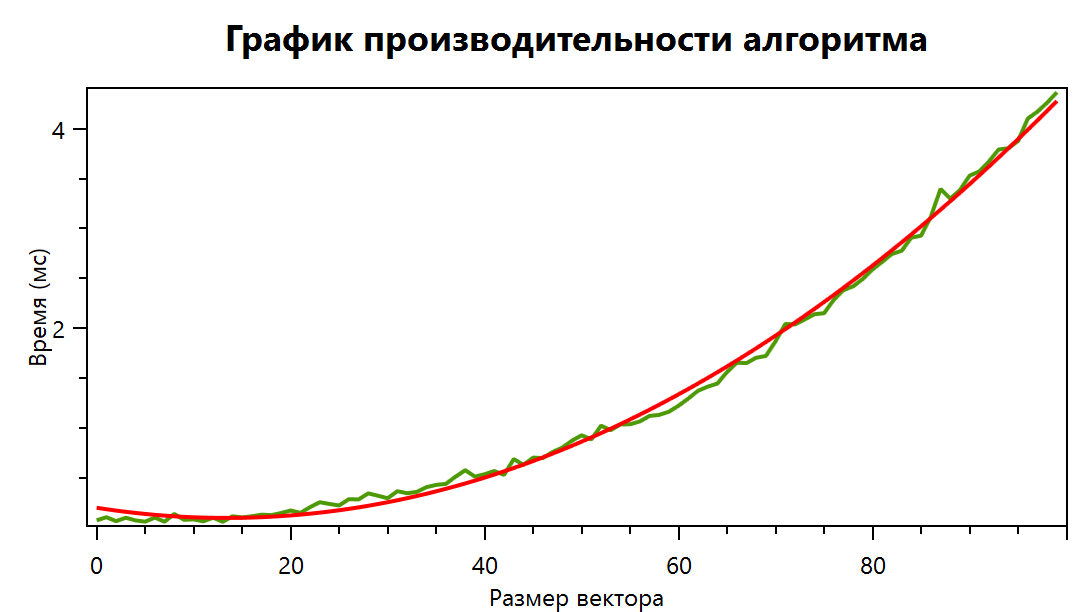
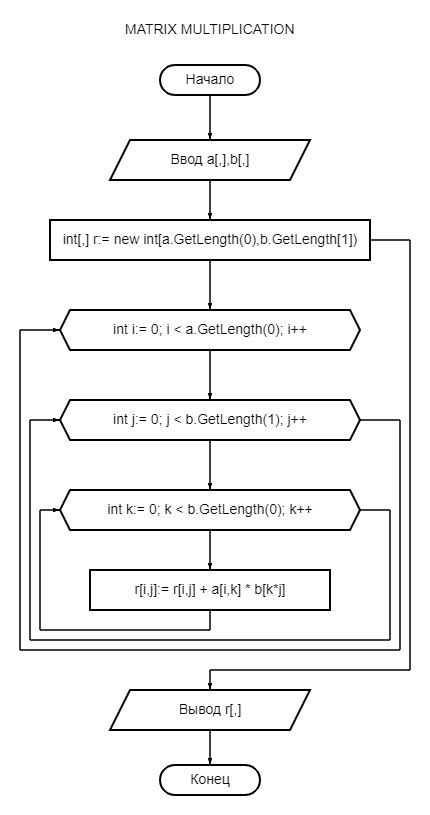
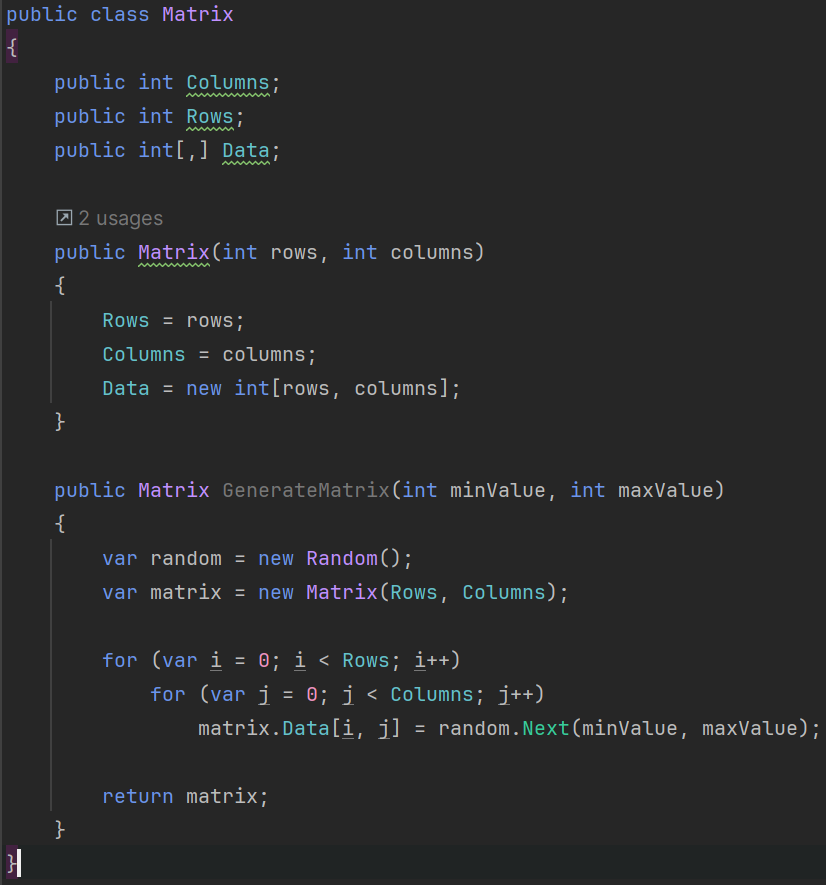


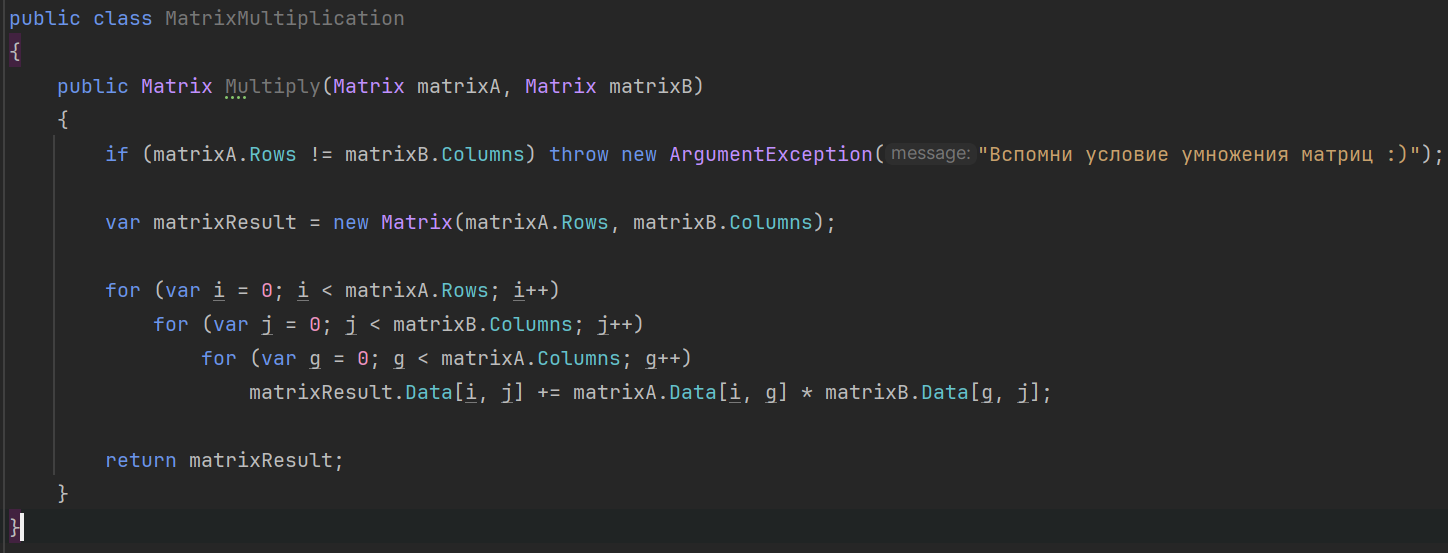
Диаграмма 13. Перемножение матриц



Блок-схема 15. Перемножение матриц



Код 14. Класс матриц



Код 15. Перемножение матриц

1. **Bitonic sort**

Алгоритм предназначен для сортировки массива, размер которого должен быть степенью двойки. Создается копия исходного массива для сортировки. Алгоритм рекурсивно делит массив на две части, сортируя одну в порядке возрастания, а другую в порядке убывания. Затем вызывается процедура слияния BitonicMerge, которая рекурсивно сравнивает и меняет элементы местами для формирования битонической последовательности. Процесс продолжается до тех пор, пока не будет отсортирован весь массив. Сложность алгоритма O(log^2(n)).

Схема алгоритма:  
Вот текстовое описание блок-схемы алгоритма Bitonic sort:

1. Ввод массива (Копирование входного массива для дальнейшей сортировки
2. Проверка размера массива (Условие: если размер массива не является степенью двойки, то выводится ошибка (завершение работы алгоритма). В противном случае переходим к следующему шагу)
3. Рекурсивное деление массив (Массив делится на две части: Одна часть сортируется по возрастанию, Другая часть - по убыванию)
4. Рекурсивная сортировка первой половины (Выполняется рекурсивная сортировка первой половины массива в порядке возрастания)
5. Рекурсивная сортировка второй половины (Выполняется рекурсивная сортировка второй половины массива в порядке убывания)
6. Слияние частей массива (Выполняется слияние двух частей массива с помощью операций сравнения и обмена для формирования битонической последовательности)
7. Сортировка слиянием (Bitonic Merge) (Каждая часть сравнивается и объединяется рекурсивно, пока весь массив не будет отсортирован)
8. Вывод отсортированного массива

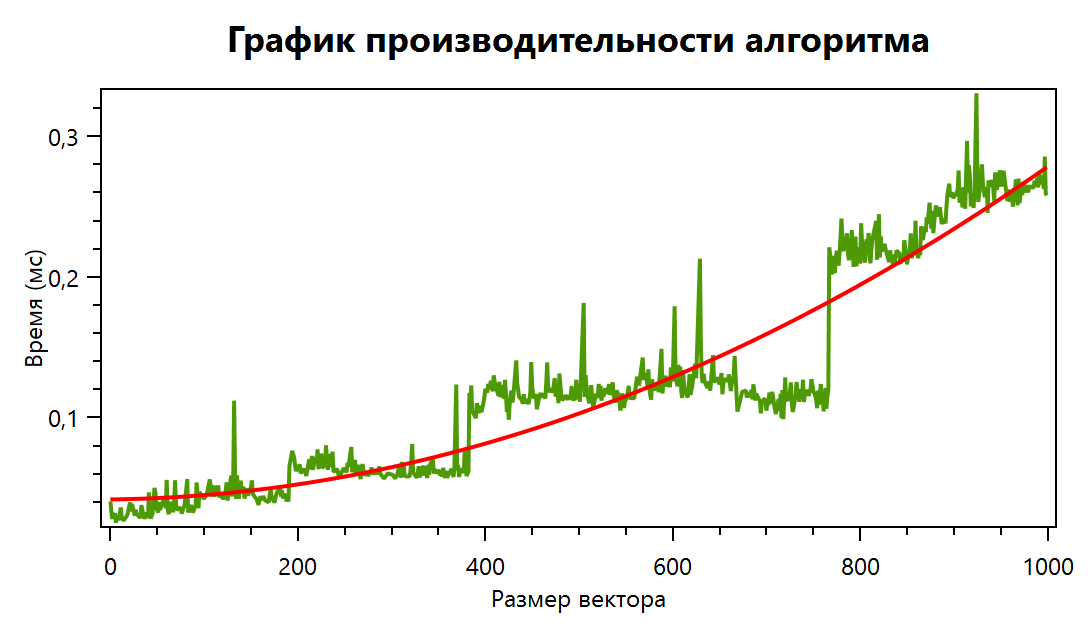
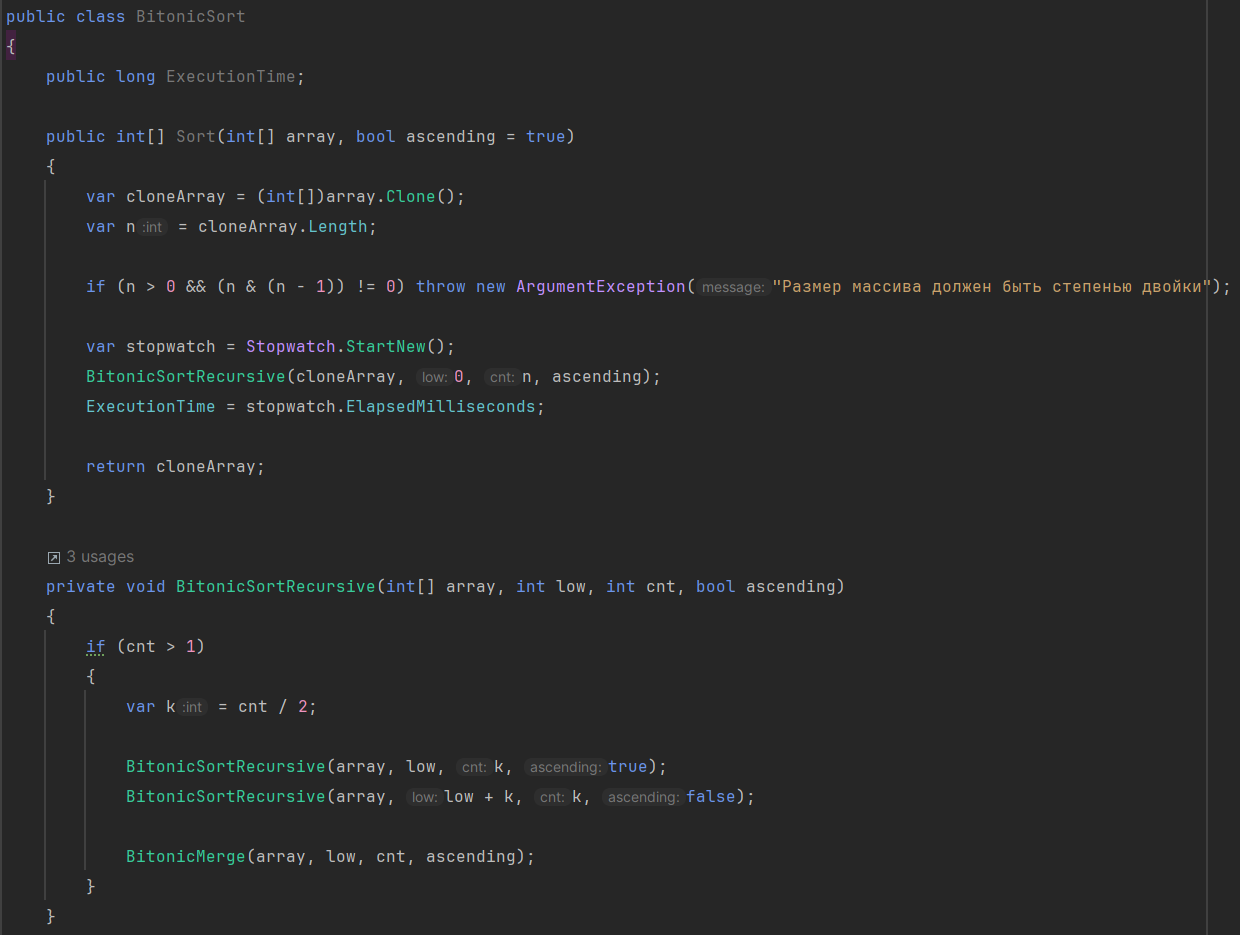
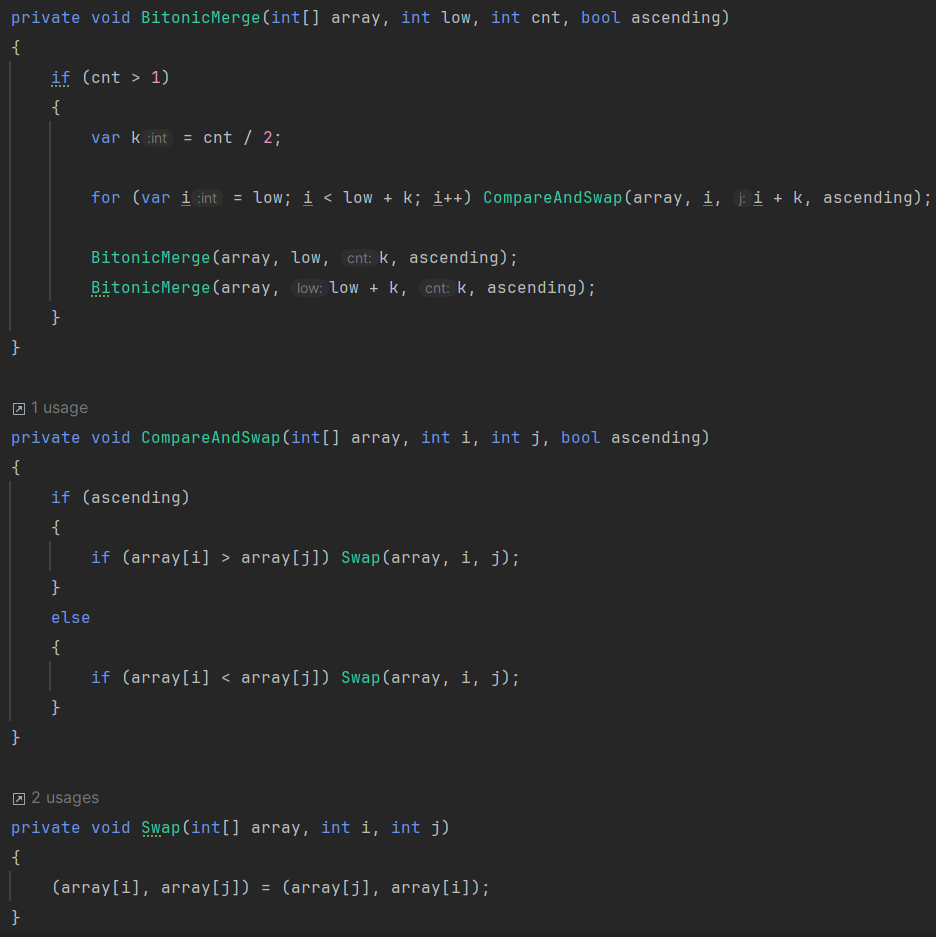


Диаграмма 14. Bitonic sort



Код 16. Bitonic Sort



Код 17. Bitonic Sort

1. **Алгоритм Кадане**

Алгоритм Кадане - это эффективный метод для нахождения наибольшей суммы подмассива в одномерном массиве чисел. Он работает за линейное время, что делает его особенно полезным для больших массивов.

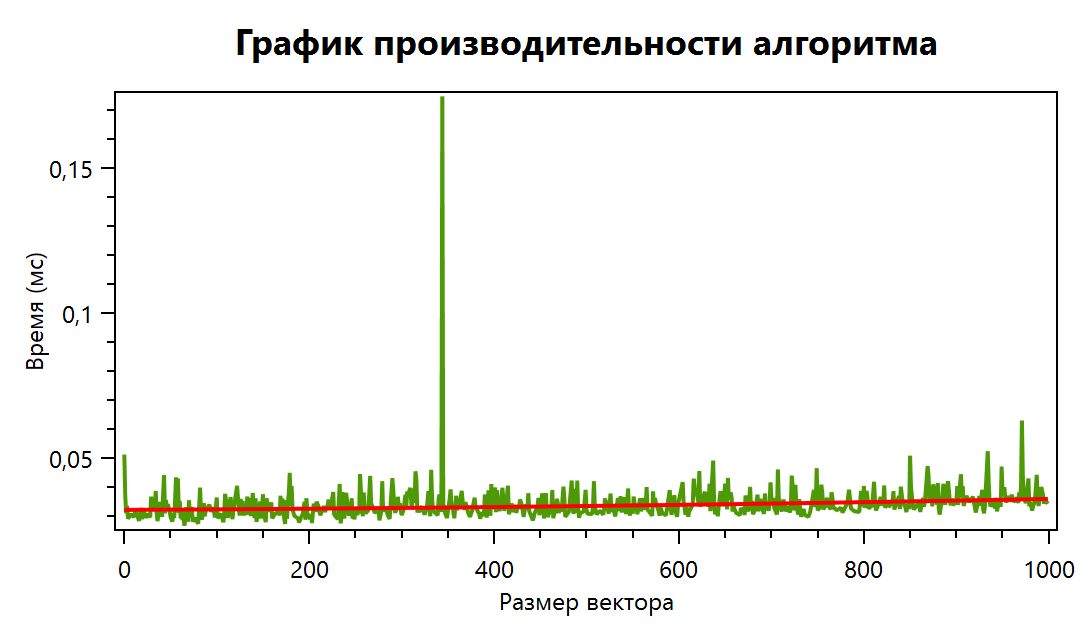
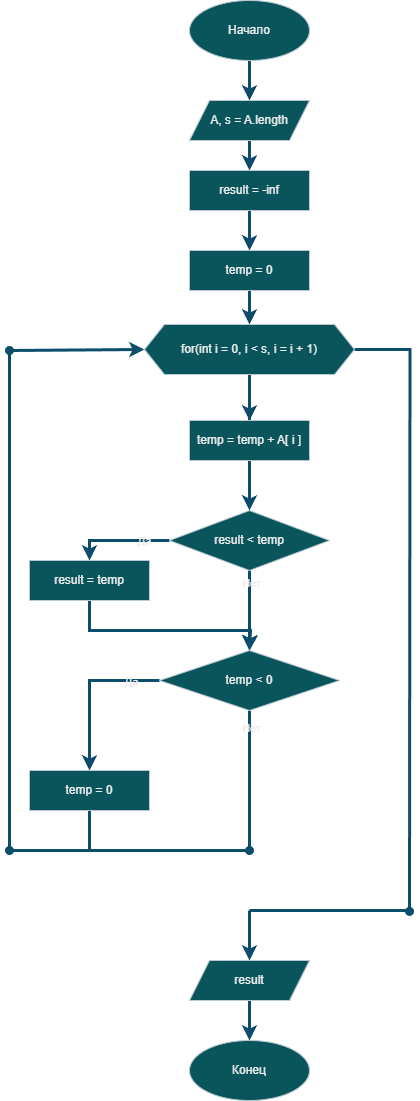
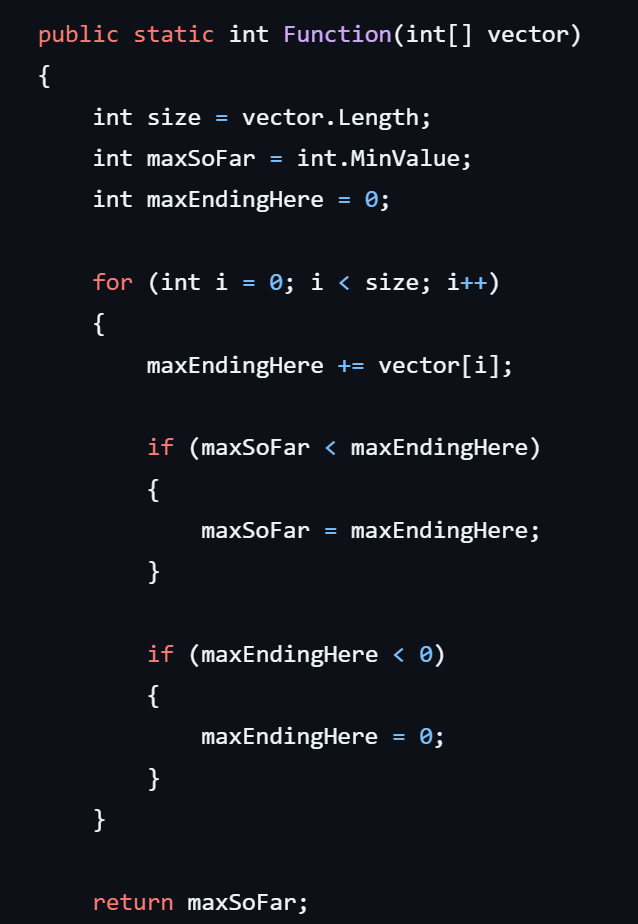


Диаграмма 15. Алгоритм Кадане

Схема:



Код 18. Kadane’s Algorithm



Код 19. Kadane’s Algorithm

1. **Алгоритм наибольшей увеличивающейся подпоследовательности**

Задача поиска *наибольшей увеличивающейся подпоследовательности* состоит в нахождении наиболее длинной возрастающей подпоследовательности в данной последовательности элементов. Временная сложность – n2.

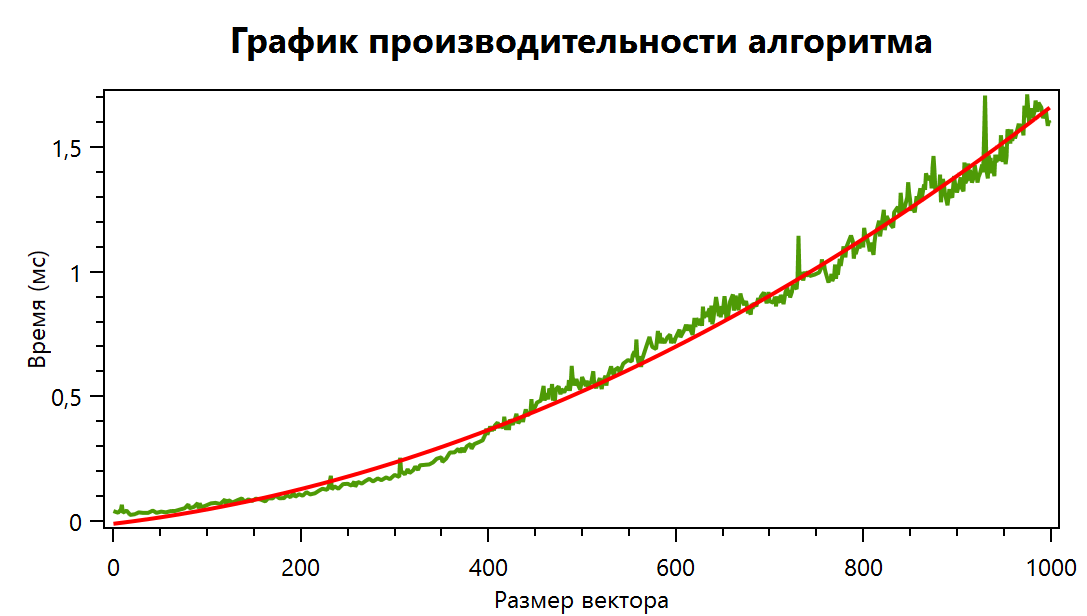


Диаграмма 16. Алгоритм НУП

A screenshot of a computer program

Description automatically generated

Код 20. Longest Increasing Subsequence

**Шаги алгоритма:**

1. Инициализируем массив LIS длиной, равной длине исходного массива.
2. Сравниваем каждый элемент массива с предыдущими.
3. Обновляем значения в массиве LIS, если элемент можно добавить в возрастающую подпоследовательность.
4. Выбираем максимальное значение из массива LIS.
5. Выводим длину наибольшей возрастающей подпоследовательности.