Государственное бюджетное общеобразовательное учреждение гимназия №271 Красносельского района Санкт-Петербурга

имени П. И. Федулова

# Проектная работа по теме

# “Использование фракталов для структуризации данных на языке программирования Python”

Работу выполнил:

Блинов Федор, ученик 9-2 класса

Руководитель:

Анохина Александра Васильевна, учитель информатики

Санкт-Петербург

2022 год

# Оглавление

[Введение 4](#_Toc100179554)

[Актуальность темы 4](#_Toc100179555)

[Цель работы 5](#_Toc100179556)

[Задачи 5](#_Toc100179557)

[Этапы работы 6](#_Toc100179558)

[Глава 1: Теоретическая часть 7](#_Toc100179559)

[Что такое фракталы? 7](#_Toc100179560)

[Что подразумевается под “фрактальной философией” при разработке алгоритма? 9](#_Toc100179561)

[Глава 2: Практическая часть 10](#_Toc100179562)

[Создание концепта и базового алгоритма 10](#_Toc100179563)

[Коротко о реализации других сортировок 13](#_Toc100179564)

[Сравнение и подсчет эффективности 15](#_Toc100179565)

[Визуализация результатов 16](#_Toc100179566)

[Инструменты 19](#_Toc100179567)

[Гитхаб проекта 20](#_Toc100179568)

# Введение

## Актуальность темы

Структуризация данных – достаточно примитивная на первый взгляд задача. Многие люди считают, что все программы сортировки являются исключительно академической дисциплиной, благодаря которой ученик может развить алгоритмическое мышление и разработать навыки создания простых программ. Этот ход мысли не имеет ничего общего с жизнью. Алгоритмы сортировки используются везде: сортировка товаров по ценам в интернет-магазинах, сортировка баз данных с информацией о пользователях, сортировка статей по заголовкам – и все это лишь повседневные вещи, о которых человек даже не задумывается. Если углубится в тему, то благодаря грамотным и эффективным алгоритмам сортировки совершаются величайшие открытия в науке, ведь почти во всех расчетах, использующих большое количество переменных, необходимо их организовывать.

Новизна проекта заключается именно в принципах, лежащих в его основе. Фракталы – фундаментальная вещь, это принцип существования вселенной, и мне показалось удивительным столь скудное применение данной концепции в алгоритмическом программировании. Сортировок, работающих по принципу фракталов просто не существует.

С точки зрения практической ценности мой проект представляет интерес в качестве возможной альтернативы для оптимизации многих существующих систем. Мой алгоритм может послужить заменой своим менее эффективным собратьям в некоторых ситуациях.

## Цель работы

Реализовать алгоритм сортировки данных, в основе которого лежит теория о самоподобных объектах и фракталах и выяснить возможную эффективность и сферы применения данного алгоритма.

## 

## Задачи

* Создать алгоритм, в основе которого лежит принцип фракталов
* Оптимизировать данный алгоритм, используя возможности языка программирования Python
* Сравнить эффективность алгоритма с классическими программами сортировки: шейкерной сортировкой, примитивной пузырьковой сортировкой, сортировкой вставками.
* Сделать выводы

## Этапы работы

* Расшифровать понятие фракталов в контексте работы
* Изучить стандартные алгоритмы сортировки
* Реализовать собственный алгоритм, используя философию самоподобия в основе
* Переработать “сырой” алгоритм в более эффективный, используя возможности языка Python
* Создать эффективную программу тестирования для нескольких алгоритмов
* Протестировать эффективность алгоритмов на нескольких массивах данных
* Визуализировать результаты, используя библиотеку Matplotlib
* Подвести итоги работы

# Глава 1: Теоретическая часть

## Что такое фракталы?

Фрактал – некоторое множество, обладающее свойством самоподобия (объект, в точности или приближенно совпадающий с частью самого себя). Самое популярное и понятное представление фракталов – визуализация некоторых множеств и использованием компьютерных технологий: это дает наглядное представление о том, что такое самоподобие. Самыми известными примерами являются: кривая Коха, множество Мандельброта, треугольник Пифагора.

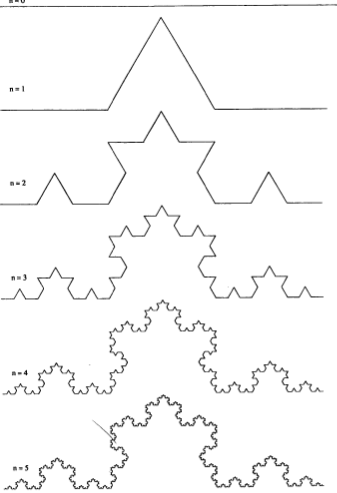


Рисунок 1 - кривая Коха

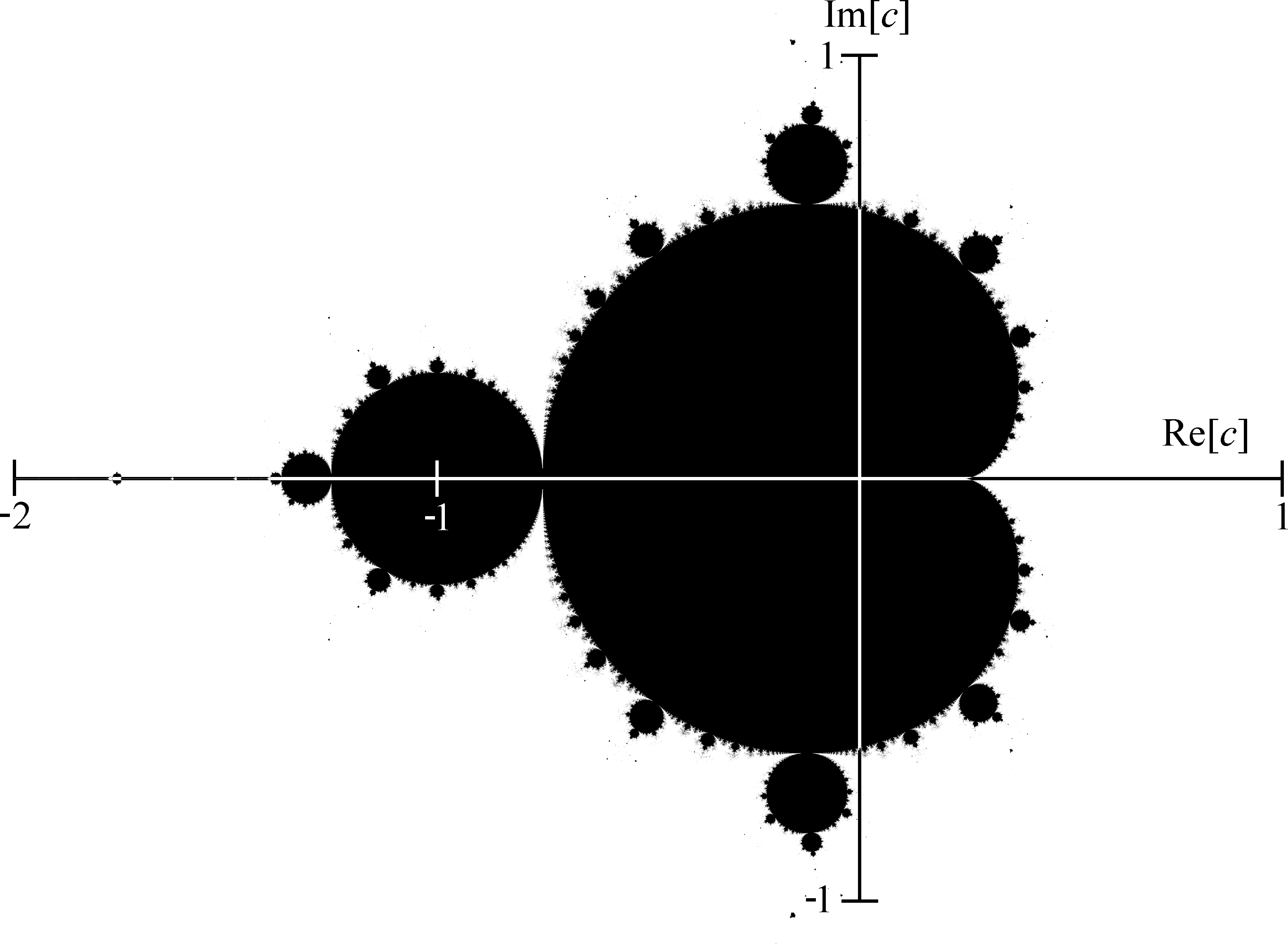


Рисунок 2 - множество Мандельброта

Фракталы в геометрическом плане представляют собой достаточно простую систему:

* Существует условное выражение (разнится для каждого множества) формата “берем x, делаем с x действие y, повторяем n раз”. К примеру, для кривой Коха условие будет выглядеть так: *“берём единичный отрезок, разделяем на три равные части и заменяем средний интервал равносторонним треугольником без этого сегмента”*
* Существует условная переменная n, обозначающая количество итераций алгоритма (т. е. кол-во раз, которое мы повторяем условие с исходным отрезком). Чем выше количество итераций, тем более комплексным и сложным будет результат.

В контексте данной исследовательской работы фрактал стоит воспринимать исключительно в качестве абстрактного понятия, под которым подразумевается некий объект, состоящий из одинаковых “кирпичиков”

## 

## Что подразумевается под “фрактальной философией” при разработке алгоритма?

Концепцию, которой я придерживался при разработке и составлении алгоритма, я назвал “фрактальной философией”. Под ней в контексте моей работы подразумевается общая идея наличия в каждом объекте сортировки определенных сходств.

Схема расширения и модификации фрактального множества выглядит следующим образом:

набор данных – обработка набора данных n-ое число раз – получение готового множества, имеющего четкую структуру.

Данная схема отлично описывает не только фракталы, но и практически любой алгоритм сортировки.

Именно данное сходство подтолкнуло меня к идее реализации этого проекта.

# Глава 2: Практическая часть

## Создание концепта и базового алгоритма

В первую очередь я начал изучать варианты реализации сортировки по принципам, в основе которых лежат фракталы, однако не смог найти никаких официальных исследований.

По мере изучения темы, у меня возникало много разнообразных идей реализации: создание хэш-карты массива с последующей ее сортировкой; поиск некоторых паттернов (шаблонов) в графике массива. Все методы оказывались либо неэффективными, либо бесполезными.

Спустя некоторое время раздумий родился концепт алгоритма. Принцип его работы звучит следующим образом:

Алгоритм проходится по некоторому массиву и распределяет данные по группам c ключом для сортировки.

Так, массив [10, 12, 10, 1, 5] изначально будет представлен в виде множества [(10, 10), (12), (1), (5)], которое позже будет отсортировано произвольным способом по первому ключу каждого кортежа [(1), (5), (10, 10), (12)]. После все кортежи могут быть раскрыты и исходный массив данных приобретает вид

[1, 5, 10, 10, 12]

Таким образом, теряется необходимость сортировки каждого отдельного элемента, сортируются лишь ключи (типы) элементов.

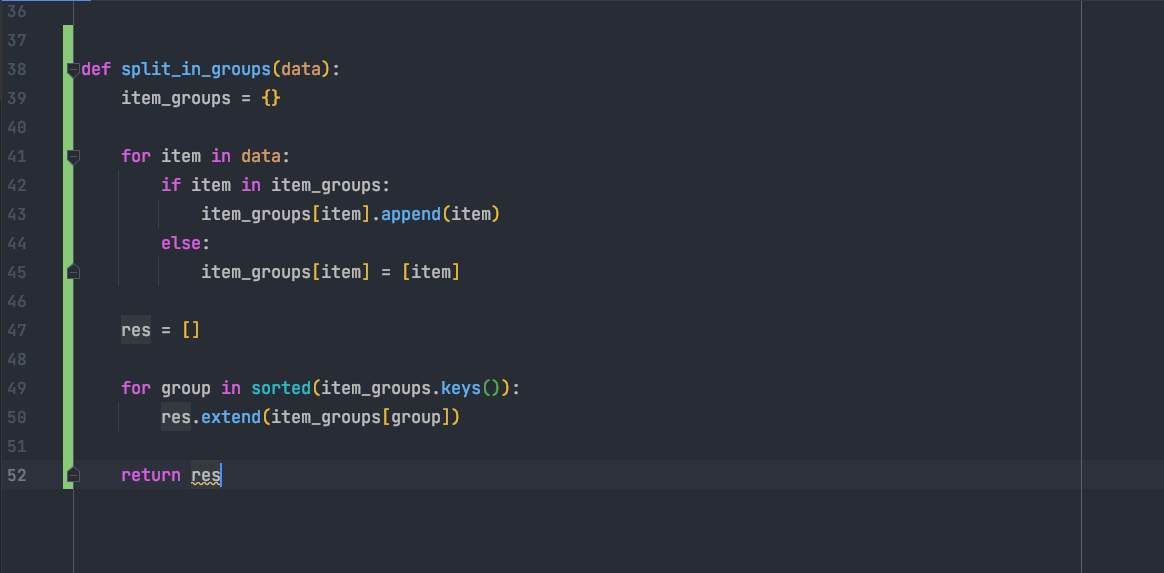


Рисунок 3 - первая вариация алгоритма

Первоначальный вариант алгоритма был реализован за несколько минут и представлял собой достаточно нагруженный и неоптимизированный, но рабочий блок кода. На его примере получилось понять основные плюсы и минусы самого принципа сортировки, так, например, выяснилось, что данный вариант сортировки представляет отличную альтернативу другим алгоритмам при условии, что работа ведётся с массивом с больших кол-вом повторяющихся ключей. Данное преимущество и является главной особенностью моего проекта.

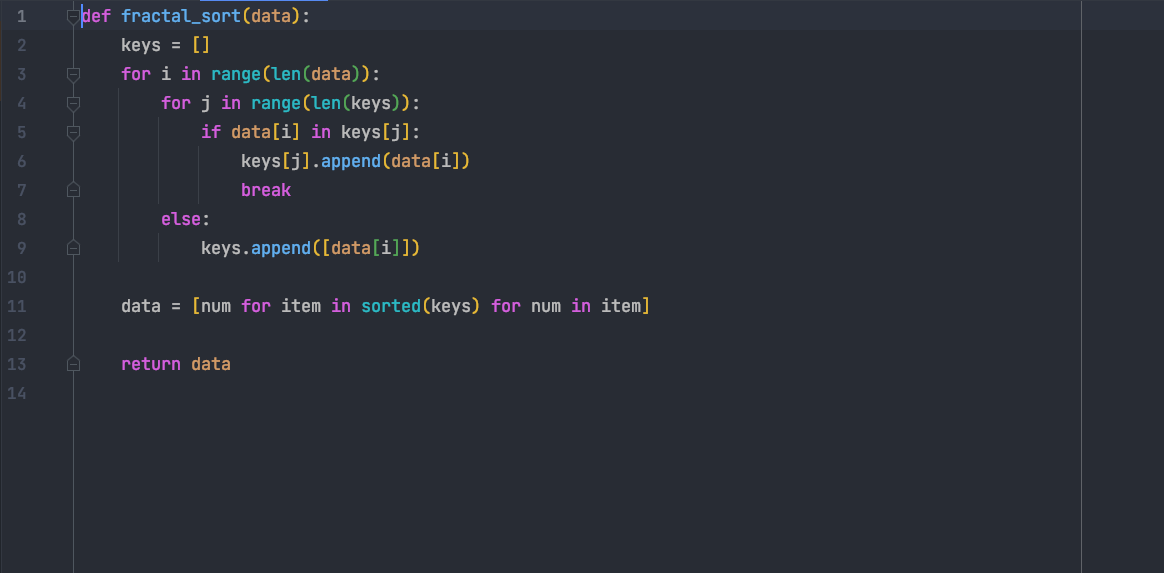


Рисунок 4 - базовый вариант алгоритма

Спустя некоторое время разработки были реализованы два финальных варианта программы: алгоритмичный и оптимизированных. Алгоритмичный вариант является мультиязычным (т.е. реализуемым на большинстве ЯП) и является наиболее простым и читаемым. Данный вариант реализован с использованием простых конструкций и типов данных, за счет чего может быть интерпретирован в других вариациях. Оптимизированный же вариант служит демонстрацией скоростных возможностей алгоритма и использует встроенные средства языка Python. В эффективности второй вариант выигрывает за счет использования *словарей (dict)* и *генераторов*, которыми обладает язык Python.

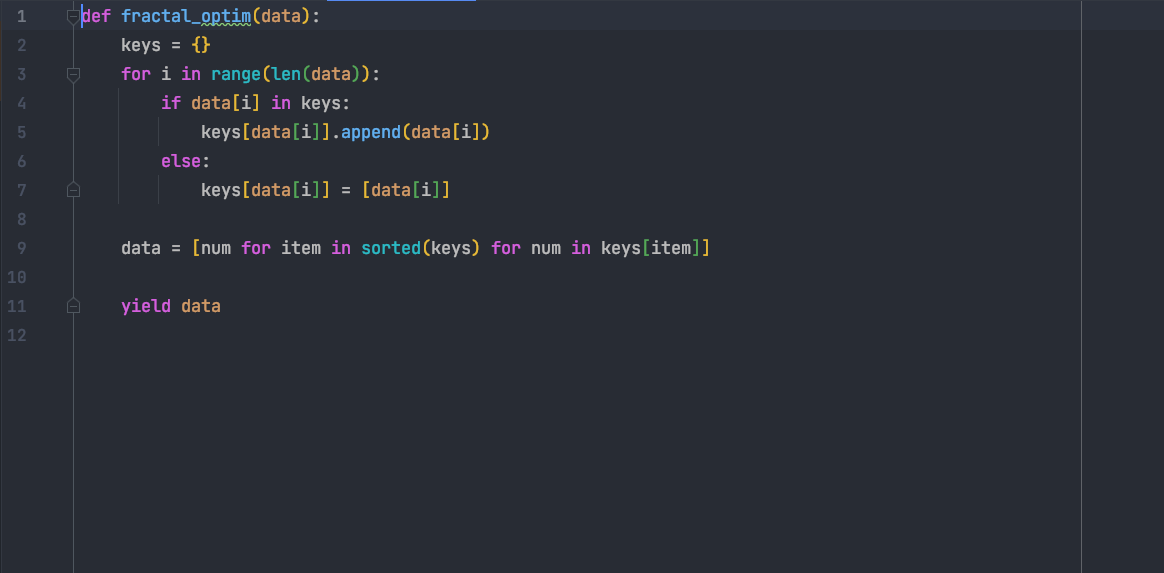


Рисунок 5 - оптимизированный вариант алгоритма

## Коротко о реализации других сортировок

Для оценки качества работы программы, в проекте присутствует два дополнительных алгоритма сортировки: пузырьковая сортировка и сортировка вставками. Оба этих варианта являются крайне популярными и прекрасно демонстрируют два разных уровня эффективности сортировки. Оба этих алгоритма активно применяются при разработке разнообразных программ и продуктов.

“Пузырьковая” сортировка – простейший и достаточно медленный алгоритм сортировки, чаще всего именно с ним программист сталкивается впервые. Принцип его работы следующий:

Алгоритм проходится по некоторому массиву данных и по мере прохождения сравнивает элементы попарно: если элемент, идущий раньше больше элемента, идущего позже (или наоборот), оба элемента меняются местами.



Рисунок 6 - сортировка пузырьком

Сортировка вставками – крайне эффективный алгоритм сортировки, часто использующийся в качестве вторичной сортировки в различных проектах. Основное преимущество – эффективен вне зависимости от типов данных. Основной недостаток – может сильно терять в скорости на крайне больших массивах. Принцип работы сортировки вставками:

Алгоритм проходится по некоторому массиву данных и “протаскивает” каждый элемент по всему множеству, пока тот не встанет на свое место.



Рисунок 7 - сортировка вставками

## Сравнение и подсчет эффективности

Для того, чтобы понять качество производительности алгоритмов, я помещаю их в одну среду, выдаю им одну задачу и измеряю время исполнения, т. е. все программы сортируют один и тот же массив данных, после чего можно увидеть их сравнительную эффективность. Для измерения эффективности используется библиотека timeit (подробнее в разделе Инструменты)

## Визуализация результатов

Самым простым способом показать результаты сравнения эффективности программ будет являться простая графическая визуализация. Для удобства я буду использовать библиотеку matplotlib (подробнее в разделе Инструменты).

Подробнее о тестировании: существует несколько массивов данных различной длины с различным диапазоном значений, а также специальный код, прогоняющих массивы через каждый из четырех алгоритмов. Данная операция повторяется произвольное кол-во раз (в моем случае - 1000). По результатам высчитывается среднее арифметическое время исполнения каждого алгоритма и выводится на график. Ниже представлены графики для трех случаев: массив с диапазоном d = 0;10 и кол-вом элементов n = 100, массив с диапазоном d = 0;100 и кол-вом элементов n = 100 и массив c d = 0;100 и n = 10000. *Каждый график показывает время выполнения сортировки каждым алгоритмом в секундах!*

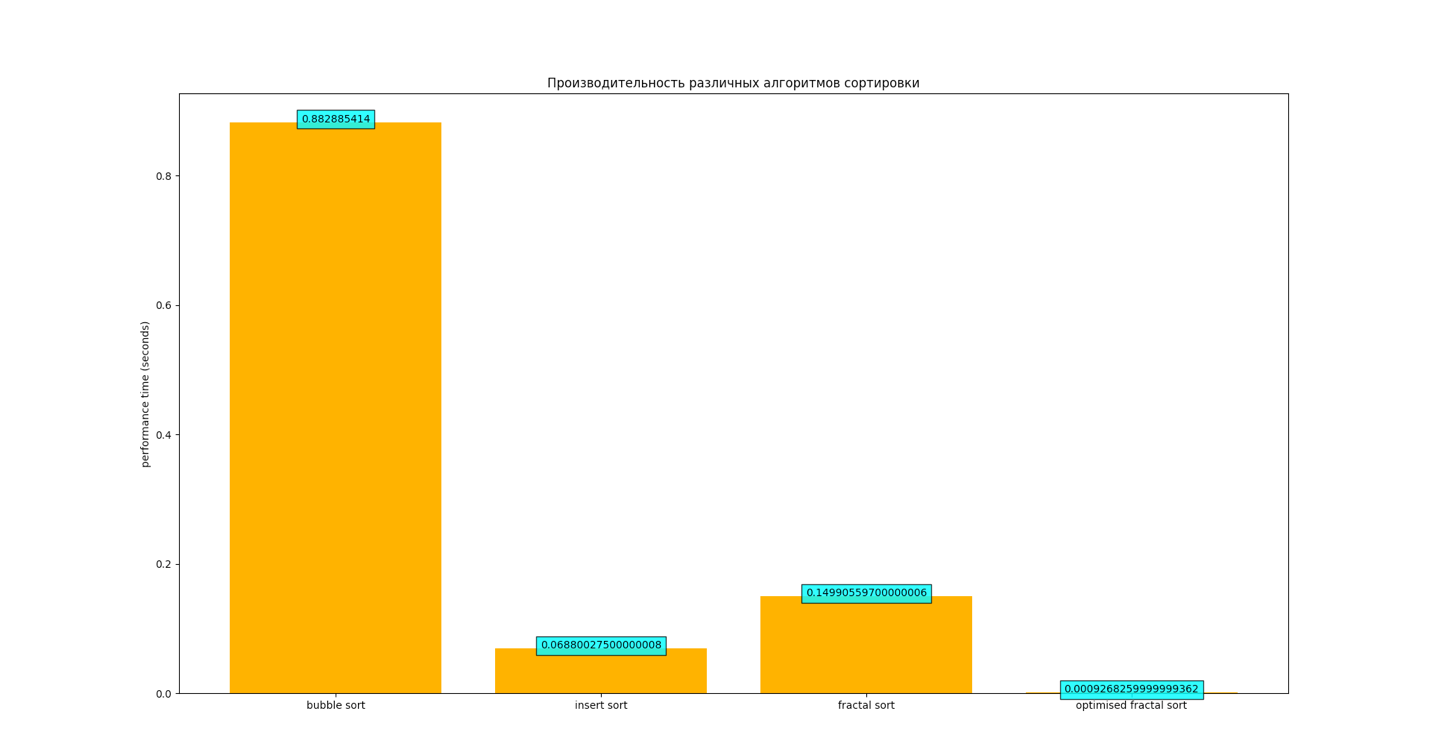


Рисунок 8 - d=0;10,n=100

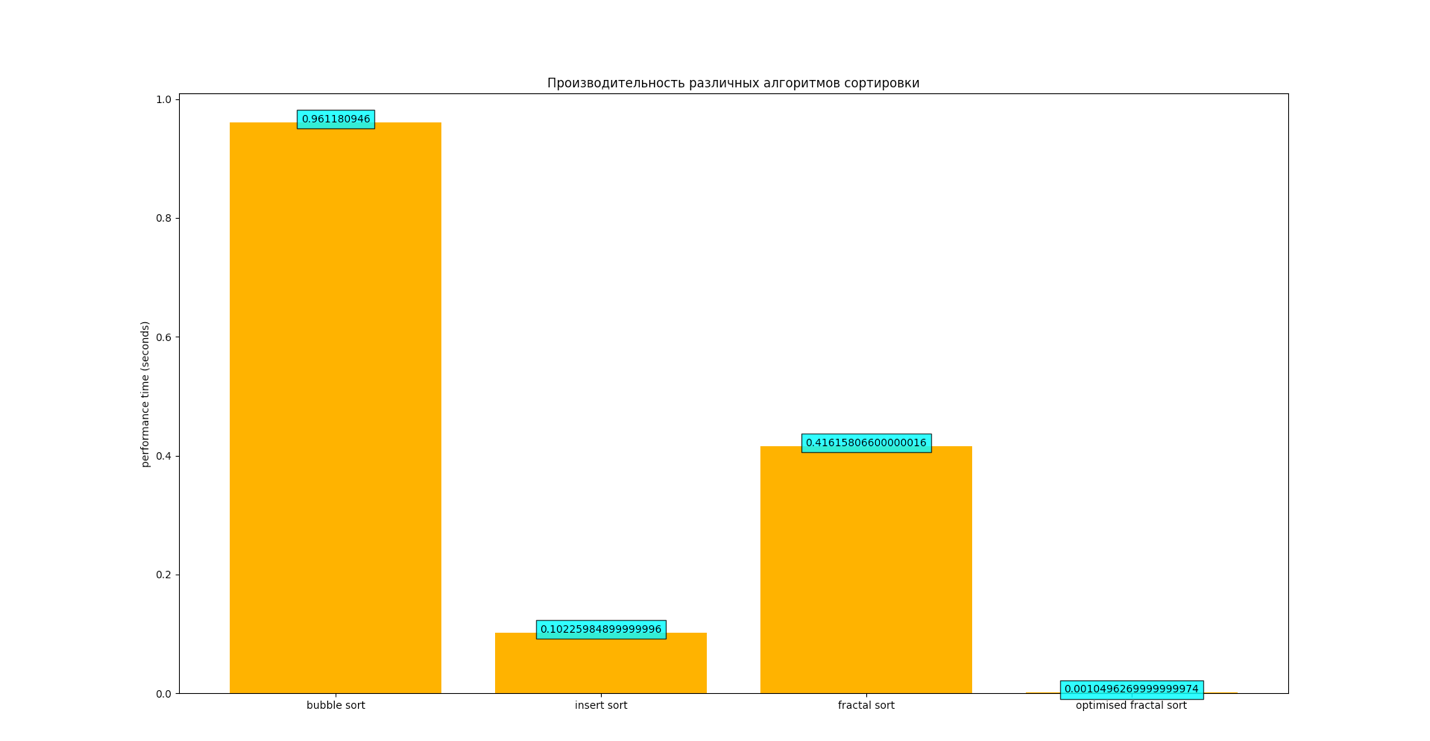


Рисунок 9 - d=0;100,n=100

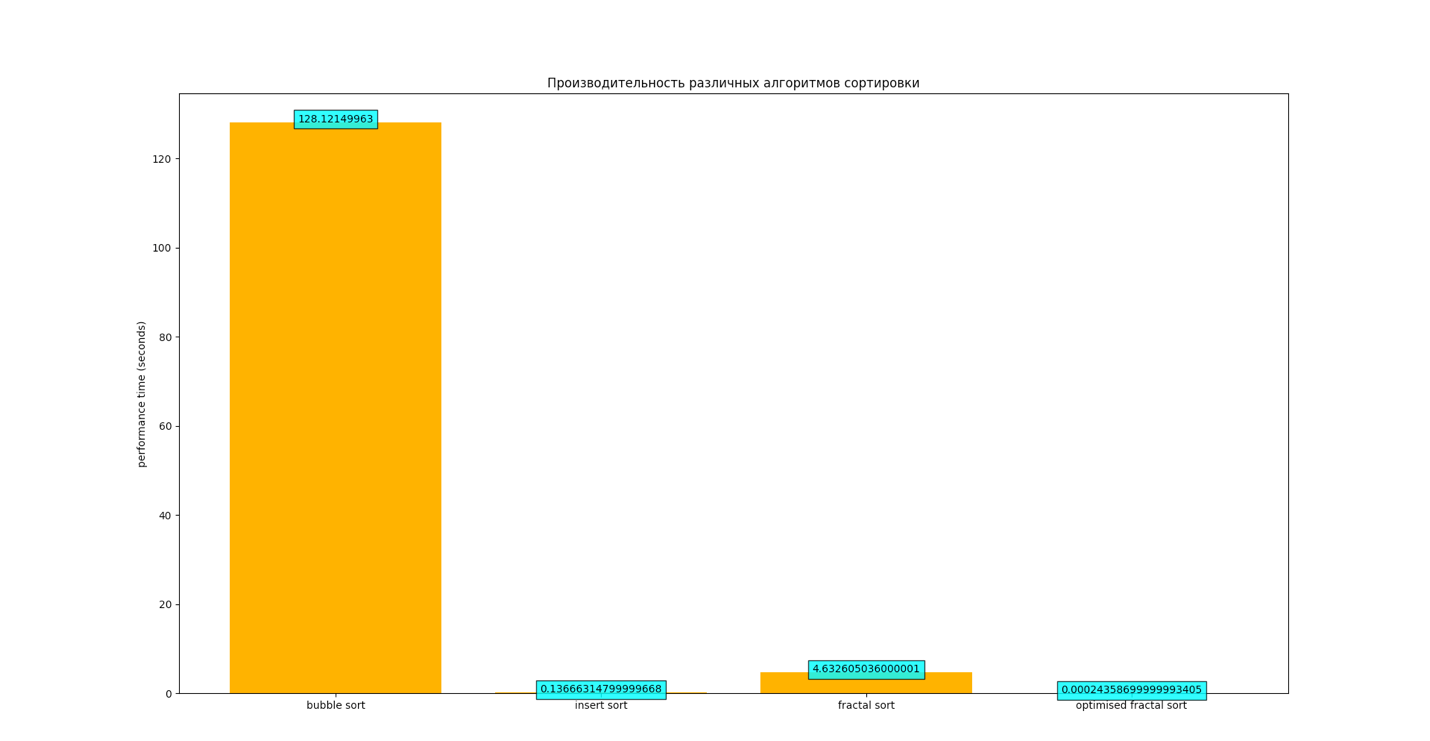


Рисунок 10 - d=0;100, n=10000

На всех графиках заметна стабильная тенденция: пузырьковая сортировка стабильно во много раз уступает фрактальной сортировке и сортировке вставками в скорости; фрактальная сортировка уступает сортировке вставками и своей оптимизированной версии; сортировка вставками всегда занимает второе место по скорости, однако начинает сдавать на больших объемах данных.

Смотря на графики, можно увидеть крайне интересную статистику: чем больше диапазон, тем менее эффективен мой алгоритм. Логика ясна – больше диапазон = больше групп, которые необходимо сортировать.

# Инструменты

При создании проекта были использованы следующие инструменты:

* ***PyCharm*** – среда разработки, одна из наиболее популярных программ для создания софта на языке программирования Python. Позволяет быстро и удобно настраивать интерпретатор, оперировать git, оперировать структурой проекта и писать код.
* ***Python*** – интерпретируемый язык программирования, активно набирающий популярность в последние годы. Основная причина использования именно этого языка при разработке проекта – простота синтаксиса: данный язык часто используется в научном программировании из-за удобства написания и чтения кода.
* ***Matplotlib*** – библиотека языка Python, использующаяся в научном программирования для эффективной визуализации данных. Позволяет создавать и строить графики и диаграммы.
* ***timeit*** – библиотека языка Python, использующаяся для измерения времени исполнения определенных функций. Встроенный функционал библиотеки позволяет передать библиотеке алгоритм и кол-во повторений исполнения алгоритма, тем самым библиотека может эффективно высчитать среднее арифметическое время исполнения программы с высокой точностью.
* ***random*** – встроенный модуль языка Python. Используется для генерации случайных чисел.

# Гитхаб проекта

Ссылка на гитхаб проекта - https://github.com/Qarkarich/fractal\_sort