Тема 04: Алгоритмы сортировки

Цель работы: Изучить и реализовать основные алгоритмы сортировки. Провести их теоретический и практический сравнительный анализ по временной и пространственной сложности. Исследовать влияние начальной упорядоченности данных на эффективность алгоритмов. Получить навыки эмпирического анализа производительности алгоритмов.

Теория (кратко):

- **Сортировка пузырьком (Bubble Sort):** Многократно проходит по массиву, сравнивая и меняя местами соседние элементы. Сложность: $O(n^2)$ во всех случаях.
- **Сортировка выбором (Selection Sort):** На каждом проходе находит минимальный элемент из неотсортированной части и ставит его на очередную позицию. Сложность: $O(n^2)$.
- **Сортировка вставками (Insertion Sort):** Построение окончательного массива путем пошагового вставления каждого элемента в правильную позицию в уже отсортированной части. Сложность: $O(n^2)$ в худшем и среднем, $O(n)$ в лучшем (уже отсортированный массив).
- **Сортировка слиянием (Merge Sort):** Рекурсивный алгоритм "разделяй и властвуй". Массив разбивается на две части, которые сортируются рекурсивно, а затем сливаются в один отсортированный массив. Сложность: $O(n \log n)$ во всех случаях. Требует $O(n)$ дополнительной памяти.
- **Быстрая сортировка (Quick Sort):** Рекурсивный алгоритм "разделяй и властвуй". Выбирается опорный элемент, массив разделяется на элементы меньше и больше опорного, которые сортируются рекурсивно. Сложность: $O(n \log n)$ в среднем, $O(n^2)$ в худшем случае (плохой выбор опорного элемента). Сортировка на месте, не требует дополнительной памяти.

Практика (подробно):

Задание:

1. Реализовать 5 алгоритмов сортировки.
2. Провести теоретический анализ сложности каждого алгоритма.
3. Экспериментально сравнить время выполнения алгоритмов на различных наборах данных.
4. Проанализировать влияние начальной упорядоченности данных на эффективность сортировок.

Шаги выполнения:

1. **Создание проекта:** Создать файл `sorts.py`.
2. **Реализация сортировок (в `sorts.py`):**
    - Реализовать все 5 алгоритмов сортировки.
    - **Для каждого метода в комментарии указать временную и пространственную сложность в худшем, среднем и лучшем случаях.**
3. **Подготовка тестовых данных (в `generate_data.py`):**
    - Сгенерировать массивы целых чисел разного размера (напр., 100, 1000, 5000, 10000 элементов).
    - Разные типы данных:
        - Случайные (`random`)
        - Уже отсортированные (`sorted`)
        - Отсортированные в обратном порядке (`reversed`)

        - Почти отсортированные (`almost_sorted`) (например, 95% упорядочено, 5% перемешано)
4. **Эмпирический анализ производительности (в `performance_test.py`):**
    - Замерить время выполнения каждой сортировки на всех типах данных и для всех размеров.
    - Использовать модуль `timeit` для точных замеров.
    - **ВАЖНО:**
        - Все замеры проводить на одной вычислительной машине.
        - Убедиться, что каждый алгоритм корректно сортирует данные (написать проверочные тесты).
        - Для каждого запуска использовать копию исходных данных, чтобы не сортировать уже отсортированный массив.
5. **Визуализация (в `plot_results.py`):**
    - Построить графики зависимости времени выполнения от размера массива для каждого алгоритма на одном типе данных (например, случайные данные).
    - Построить графики зависимости времени выполнения от типа данных для фиксированного размера массива (например, n=5000).
    - Создать сводную таблицу результатов.
6. **Анализ результатов:**
    - Сравнить теоретические оценки с практическими результатами.
    - Определить наиболее эффективный алгоритм для каждого типа данных.
    - Проанализировать поведение алгоритмов на краевых случаях (обратно отсортированный массив для Quick Sort).
7. **Оформление отчета:** Результаты оформить в файле `README.md`. Код должен соответствовать PEP8.
8. **Контроль версий:** Стратегия ветвления – GitHub Flow.

Критерии оценки:

- **Оценка «3» (удовлетворительно):**
    - Реализованы 3 алгоритма сортировки (например, пузырьком, выбором, вставками).
    - В коде присутствуют комментарии с оценкой сложности.
    - Проведены базовые замеры времени для 2-3 размеров массивов на случайных данных.
- **Оценка «4» (хорошо):**
    - Выполнены все критерии на «3».
    - Реализованы все 5 алгоритмов сортировки.
    - Код хорошо отформатирован и полностью прокомментирован.
    - Проведены замеры для всех типов тестовых данных.
    - Построены основные графики зависимости времени от размера массива для случайных данных.
- **Оценка «5» (отлично):**
    - Выполнены все критерии на «4».
    - Приведены характеристики ПК для тестирования.
    - Проведен полный анализ влияния типа данных на эффективность сортировок.
    - Построены сравнительные графики для всех алгоритмов на разных типах данных.

- В отчете присутствует детальный анализ результатов с выводами о применении каждого алгоритма (например, "Insertion Sort эффективен для маленьких или почти отсортированных массивов", "Quick Sort - лучший выбор в среднем случае", "Merge Sort стабилен и предсказуем по времени").
- Реализована и протестирована проверка корректности сортировки.

Рекомендованная литература

1. **Юрий Петров: "Программирование на Python"** — онлайн-курс и учебные материалы.
    - *Ссылка для изучения:* https://www.yuripetrov.ru/edu/python/index.html
2. **Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К.** Алгоритмы: построение и анализ, 3-е издание. — М.: Вильямс, 2022. — 1328 с.
    - *(Оригинальное название: Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. Introduction to Algorithms, 3rd Edition)*
3. **Скиена, С.** Алгоритмы. Руководство по разработке, 3-е издание. — СПб.: БХВ-Петербург, 2022. — 720 с.
    - *(Оригинальное название: Skiena, Steven S. The Algorithm Design Manual, 3rd ed.)*
4. Visualgo: Визуализация алгоритмов сортировки — https://visualgo.net/en/sorting