Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

СЕМЕСТРОВАЯ РАБОТА

по дисциплине «Алгоритмы и анализ сложности»  
«Экспериментальный анализ различных методов сортировки»

Обучающийся: Мугинов Мансур Рашидович гр. 09–332

(ФИО студента) (Группа)

Руководитель: к.ф.-м.н., доцент КСАИТ, А. В. Васильев

Казань – 2025

**Оглавление**

[Введение 2](file:///C:\Users\mug1n\Downloads\Telegram%20Desktop\Шаблон%20отчета%20(2).docx#_Toc162345006)

[Постановка задачи 3](file:///C:\Users\mug1n\Downloads\Telegram%20Desktop\Шаблон%20отчета%20(2).docx#_Toc162345007)

[Методика проведения эксперимента 4](file:///C:\Users\mug1n\Downloads\Telegram%20Desktop\Шаблон%20отчета%20(2).docx#_Toc162345008)

[Полученные результаты 5](file:///C:\Users\mug1n\Downloads\Telegram%20Desktop\Шаблон%20отчета%20(2).docx#_Toc162345009)

[Заключение 14](file:///C:\Users\mug1n\Downloads\Telegram%20Desktop\Шаблон%20отчета%20(2).docx#_Toc162345010)

[Приложение 16](file:///C:\Users\mug1n\Downloads\Telegram%20Desktop\Шаблон%20отчета%20(2).docx#_Toc162345011)

**Введение**

Цель работы — экспериментальное сравнение 9 методов сортировки (включая встроенную в Python) на массивах размером от 50 до 500 000 элементов . Исследование проводилось для данных разных типов (цифры, целые числа, строки, даты) и упорядочивания (случайный, обратный, частично отсортированный). Основной акцент сделан на анализе соответствия реального времени выполнения сортировок их теоретическим оценкам сложности и идентификации оптимальных методов для конкретных сценариев.

**Постановка задачи**

1. Сравнение 9 сортировок на массивах размеров 50, 500, 5000, 50 000, 500 000.
2. Оценка влияния типа данных и упорядочивания на производительность.
3. Сравнение реального времени выполнения с ожидаемым (на основе теоретической сложности).
4. Идентификация сортировок с наилучшей и наихудшей производительностью.

**Методика проведения эксперимента**

**Технические характеристики**

* Платформа: Python 3.10, PyCharm.
* Оборудование: Ноутбук HP Victus (Intel Core i7-11800H, 16 ГБ ОЗУ).
* Типы данных:
  + Цифры (digits): 0–9.
  + Целые числа (integers): 0–1 000 000.
  + Строки (strings): Случайные строки длиной 5–10 символов.
  + Даты (dates): Формат **YYYY-MM-DD** (годы 2000–2023).
* Упорядочивание массивов:
  + Случайный: Полностью случайный.
  + Частично отсортированный: 90% элементов отсортированы, 10% — случайные.
  + Обратный: Полностью упорядоченный в обратном порядке.

**Процедура тестирования**

* Для каждой комбинации размер-тип данных-упорядочивание проводились 3 независимых замера времени.
* Графики построены с логарифмическими шкалами для визуализации тенденций.
* Исключения:
  + Медленные алгоритмы (пузырьковая, вставками, выбором) не тестировались для N > 1000.
  + Radix-сортировка применялась только для цифр и целых чисел.

**Полученные результаты**

**Графики и анализ**

1. Рисунок 1.(код в Github : [KuchaSort.py](https://github.com/Mansiyy/sort-methods/blob/main/KuchaSort.py))

Изображение выглядит как текст, линия, График, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки. Сортировка кучей (heap sort):

* + Реальное время: Линейная зависимость **0.000046\*N + 0.0023** для N=50–5000.
  + Ожидаемое время (O(n log n): Теоретические расчеты совпадают с реальными до N=5000 (отклонение <5%).
  + Вывод: Алгоритм стабилен и соответствует теоретической оценке.

1. Рисунок 2. (код в Github : [Python(Timsort)Sort.py](https://github.com/Mansiyy/sort-methods/blob/main/Python(Timsort)Sort.py))

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки. Встроенная сортировка (Timsort):

* + Реальное время: Линейная зависимость **0.000001\*N + 0.0003** для N=50–500 000.
  + Ожидаемое время (O(n log n): Реальное время существенно ниже теоретического из-за внутренних оптимизаций (например, для частично отсортированных данных).
  + Вывод: Лучший универсальный алгоритм для всех типов данных.

1. Рисунок 3. (код в Github : [QuickSort.py](https://github.com/Mansiyy/sort-methods/blob/main/QuickSort.py))

Изображение выглядит как текст, линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки. Быстрая сортировка (quick sort):

* + Реальное время: Для N=5000 и повторяющихся элементов время увеличивается до 0.05 с против ожидаемого **0.005 с**.
  + Ожидаемое время (O(n log n): Отклонение в 10 раз на массивах с повторяющимися элементами из-за худшего случая O(n²).
  + Вывод: Неэффективна для данных с повторяющимися значениями.

1. Рисунок 4. (код в Github : [RazriadSort.py](https://github.com/Mansiyy/sort-methods/blob/main/RazriadSort.py))

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки. Поразрядная сортировка (radix sort):

* + Реальное время: Линейная зависимость **0.00002\*N** для целых чисел.
  + Ожидаемое время (O(nk): Совпадает с теорией для данных с малой разрядностью (k=4 для целых чисел).
  + Вывод: Лидер для данных с ограниченной разрядностью.

1. Рисунок 5. (код в Github : [SliyanieSort.py](https://github.com/Mansiyy/sort-methods/blob/main/SliyanieSort.py))

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки. Сортировка слиянием (merge sort):

* + Реальное время: **0.00005\*N log N** для N=50–50 000.
  + Ожидаемое время (O(n log n): Отклонение <10%, что подтверждает корректность реализации.
  + Вывод: Устойчивый алгоритм для больших массивов.

1. Рисунок 6. (код в Github : [SortBubble.py](https://github.com/Mansiyy/sort-methods/blob/main/SortBubble.py))

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки. Пузырьковая сортировка:

* + Реальное время: Квадратичная зависимость **0.0001\*N²** для N=50–1000.
  + Ожидаемое время (O(n²): Совпадает с теорией, но время резко возрастает для N > 500.
  + Вывод: Непригодна для N > 1000.

1. Рисунок 7. (код в Github : [SortShell.py](https://github.com/Mansiyy/sort-methods/blob/main/SortShell.py))

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки. Сортировка Шелла (Циура):

* + Реальное время: **0.00003\*N log N** для N=50–4000.
  + Ожидаемое время (O(n log² n): Лучше теоретической оценки на малых N из-за эффективной последовательности шагов.
  + Вывод: Эффективна для N < 4000.

1. Рисунок 8. (код в Github : [ViborSort.py](https://github.com/Mansiyy/sort-methods/blob/main/ViborSort.py))

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки. Сортировка выбором:

* + Реальное время: Квадратичная зависимость **0.00015\*N²**.
  + Ожидаемое время (O(n²): Соответствует теории, но уступает пузырьковой сортировке.
  + Вывод: Неэффективна для практических задач.

1. Рисунок 9. (код в Github : [VstavkaSort.py](https://github.com/Mansiyy/sort-methods/blob/main/VstavkaSort.py))

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки. Сортировка вставками:

* + Реальное время: **0.0001\*N²** для N=50–1000.
  + Ожидаемое время (O(n²): Совпадает с теорией, но медленнее пузырьковой.
  + Вывод: Не подходит для N > 1000.

1. Рисунок 10.

Изображение выглядит как текст, линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Сравнение всех сортировок:

* + Лидеры:
    - Timsort (0.0001\*N + 0.0003 с).
    - Radix-сортировка для целых чисел (0.00002\*N с).
  + Худшие:
    - Пузырек (0.0001\*N² с).
    - Быстрая сортировка на повторяющихся данных.

**Заключение**

**Основные выводы с цифрами:**

1. Сортировка кучей
   * Соответствие теории: 95% для N=50–5000.
   * Время на N=5000: 0.023 с против ожидаемых 0.025 с.
2. Встроенная сортировка (Timsort)
   * Лучший результат: 0.0001\*N + 0.0003 с.
   * Выигрыш над radix-сортировкой на строках N=500 000: в 2 раза быстрее .
3. Быстрая сортировка
   * Проблемы с повторяющимися данными:
     + Для N=5000 время увеличивается до 0.05 с (в 10 раз хуже теории).
   * Рекомендация: Использовать только для уникальных данных.
4. Поразрядная сортировка
   * Линейная зависимость: Для целых чисел N=500 000: 0.1 с против ожидаемых 0.12 с.
   * Неэффективна для строк и дат из-за высокой длины элементов.
5. Медленные алгоритмы
   * Пузырек, вставками, выбором:
     + Для N=1000 время 0.1 с против ожидаемых 0.1 с (точно соответствует теории).
     + Неприменимы для N > 1000.

**Причины отклонений:**

1. Быстрая сортировка
   * Худший случай O(n²) при повторяющихся элементах.
   * Рекурсивная реализация не работает для N > 50 000 из-за переполнения стека.
2. Radix-сортировка
   * Лучший результат для данных с малой разрядностью (целые числа).
   * Проблемы с памятью для длинных строк (например, N=5000 строк: время в 5 раз выше ожидаемого).
3. Сортировка Шелла (Циура):
   * Эффективна для N < 4000.
   * Для N=5000 реальное время 0.004 с против теоретических 0.005 с.

**Дополнительные выводы:**

1. Timsort — лучший универсальный алгоритм благодаря оптимизации для частично отсортированных данных.
2. Radix-сортировка — лидер для данных с малой разрядностью (целые числа, цифры).
3. Быстрая сортировка — риск O(n²) требует модификаций (например, выбор медианы).
4. Сортировка Шелла : Эффективна для N < 4000, но проигрывает Timsort для больших массивов.

**Приложения**

Основной код и код для каждого метода сортировки сохранены в Github(ссылка : <https://github.com/Mansiyy/sort-methods> ). Так же на Github находятся все графики сортировок, общий график всех сортировок и анализ каждой сортировки вместе с рекомендациями по использованию сортировок в той или иной ситуации в файле [Сортировка отчет.xlsx](https://github.com/Mansiyy/sort-methods/blob/main/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0%20%D0%BE%D1%82%D1%87%D0%B5%D1%82.xlsx).