ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Сортировки»

Выполнил работу

Гуйван Артём Русланович

Академическая группа №J3111

Принято

Санкт-Петербург

2024

1. Введение

Цель данной работы заключается в изучении и реализации различных алгоритмов сортировки с целью анализа их эффективности и характеристик.

Задачи, которые необходимо выполнить в рамках данной работы:

1. Реализовать алгоритмы сортировки в соответствии с заданными требованиями.

2. Разработать и провести тестирование реализованных алгоритмов на разных наборах данных для проверки их корректности.

3. Выполнить сравнение производительности и применимости алгоритмов в зависимости от объема и типа входных данных.

4. Подготовить выводы на основе результатов анализа, выделив сильные и слабые стороны каждого из исследованных алгоритмов.

Данная работа способствует углублению понимания принципов функционирования алгоритмов сортировки и их эффективности в различных ситуациях.

2. Теоретическая подготовка

В данной работе используются три алгоритма сортировки: Pigeonhole Sort, Gnome Sort и Merge Sort.

Pigeonhole Sort

Идея: Алгоритм основан на распределении элементов массива в "отсеки" (pigeonholes) на основе их значений. После этого элементы собираются обратно в упорядоченном порядке. Данный алгоритм подходит для случаев, когда диапазон значений элементов массива относительно небольшой.

Сложность:

- Лучшая: O(n + k), где k — диапазон значений элементов.

- Худшая: O(n + k), при больших диапазонах значений (k >> n)

- Средняя: O(n + k)

Gnome Sort

Идея: Алгоритм похож на Bubble Sort и основан на принципе "садового гнома", который проверяет соседние элементы и, если они не упорядочены, меняет их местами и возвращается назад. Если элементы упорядочены, он продолжает двигаться вперед.

Сложность:

- Лучшая: O(n), для уже отсортированных данных.

- Худшая: O(n²), для обратного порядка элементов.

- Средняя: O(n²).

Merge Sort

Идея: Это рекурсивный алгоритм "разделяй и властвуй", который делит массив на две равные части, рекурсивно сортирует каждую из них, а затем объединяет в один отсортированный массив. Алгоритм стабилен и эффективно работает с большими объемами данных.

\*\*Сложность\*\*:

- Лучшая: O(n log(n)), для всех входных данных.

- Худшая: O(n log(n)), вне зависимости от порядка элементов.

- Средняя: O(n log(n))

Эти алгоритмы демонстрируют различные подходы к сортировке данных, что позволяет оценить их производительность и применимость в различных условиях.

3. Реализация

Было реализовано 3 алгоритма сортировки. Их можно увидеть в файле sorts.cpp. Идеи этих сортировок приведены в пункте два.

Были использованы стандартные библиотеки:

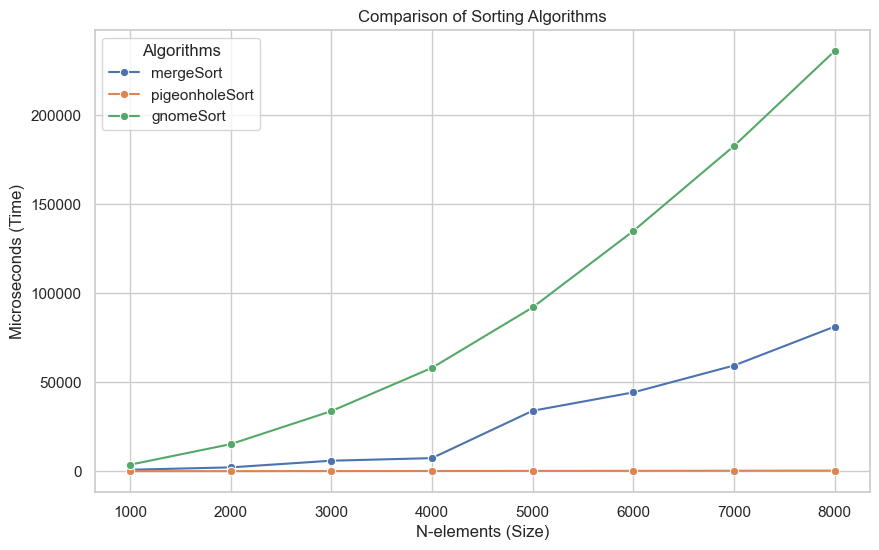
<fstream> <vector>

<string> algorithm> <chrono>

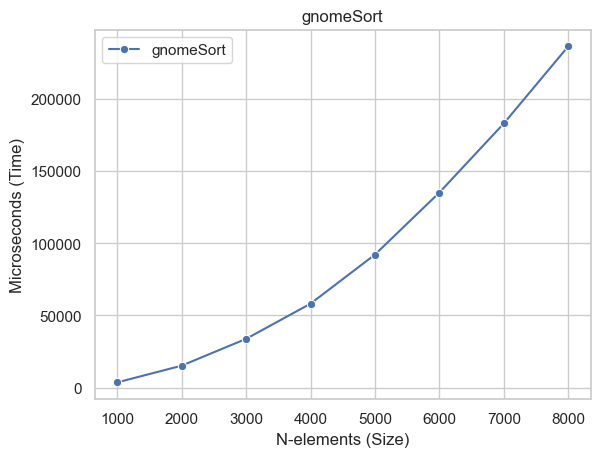
<sstream> <cassert>

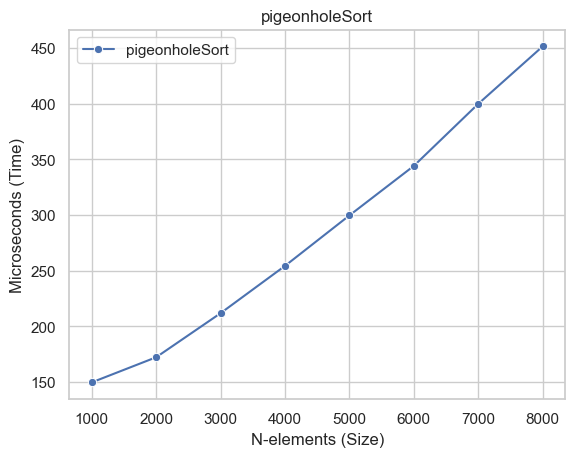
4. Экспериментальная часть

Изображение №1 - График работы алгоритма

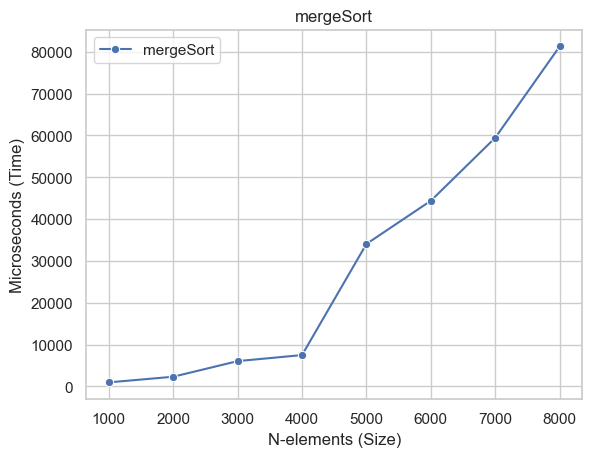


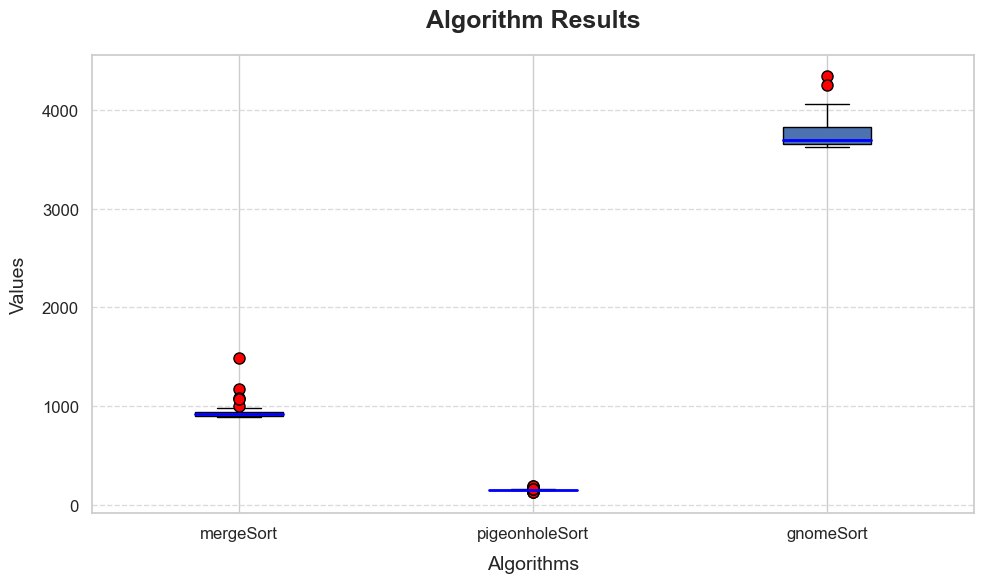
Изображение №2 - График работы алгоритма GnomeSort

Изображение №3 - График работы алгоритма PigeonholeSort



Изображение №3 - График работы алгоритма MergeSort

Изображение №4 - График работы алгоритмов Boxplot



### Анализ графиков

****1. Pigeonhole Sort****

* **Время выполнения**:  
  Алгоритм демонстрирует **линейное время выполнения** при условии небольшого диапазона значений элементов массива (O(n + k)).
* **Особенности**:
  + Алгоритм эффективен для массивов с небольшим диапазоном значений (k) и практически равномерным распределением данных.
  + При увеличении диапазона значений (k >> n) или при неравномерном распределении производительность существенно ухудшается из-за увеличения затрат на создание и обработку "отсеков".
* **Рекомендации по использованию**:  
  Подходит для задач с заранее известным небольшим диапазоном значений, например, сортировка оценок или дат. Однако использование для массивов с большим диапазоном значений неэффективно.

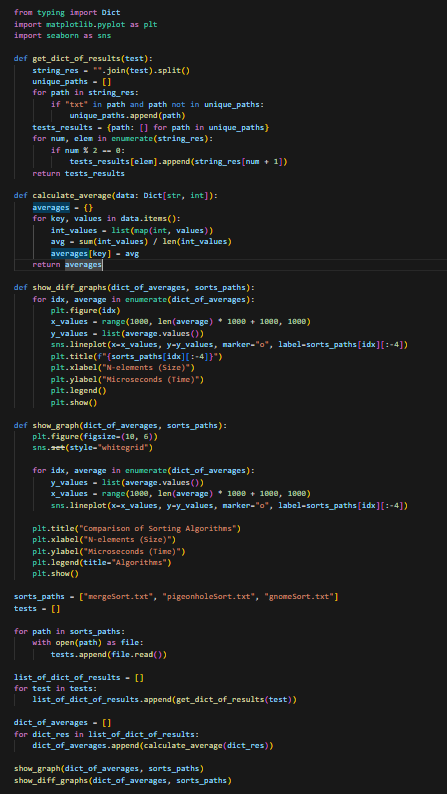
**2. Gnome Sort**

* **Время выполнения**:  
  Производительность алгоритма сильно зависит от исходного порядка элементов:
  + **O(n)** для почти отсортированных массивов.
  + **O(n²)** для случайного или обратного порядка данных.
* **Особенности**:
  + Простая реализация.
  + Неэффективен для больших массивов из-за квадратичной сложности.
* **Рекомендации по использованию**:  
  Подходит для небольших массивов (до нескольких тысяч элементов) или частично отсортированных данных. Для больших объемов данных алгоритм лучше не применять.

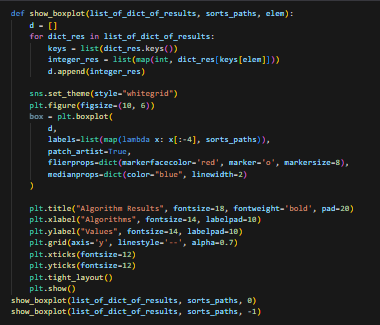
****3. Merge Sort****

* **Время выполнения**:  
  Алгоритм демонстрирует **стабильное время выполнения O(n log(n))** независимо от порядка данных.
* **Особенности**:
  + Стабильный алгоритм.
  + Требует дополнительной памяти (O(n)), что может быть ограничением для работы с большими объемами данных.
  + Эффективен как для случайных, так и для частично отсортированных массивов.
* **Рекомендации по использованию**:  
  Идеален для сортировки больших массивов, особенно если требуется стабильность. Однако при ограниченной памяти или необходимости работы "на месте" стоит рассмотреть другие сортировки

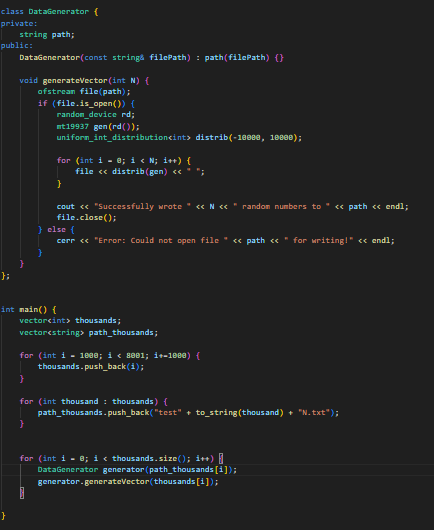
4. Приложения



Изображение №1 — Листинг кода отрисовки line plot гарфиков



Изображение №2 — Листинг кода отрисовки boxplot графиков



Изображение №3 — Листинг кода генерации данных