ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Сравнение алгоритмов сортировки»

Выполнил работу

Сурин Тимур

Академическая группа J3113

Принято

Дунаев Максим Владимирович

Санкт-Петербург

2024

Структура отчёта:

1. ВВЕДЕНИЕ

Цель работы: изучение и сравнение трех алгоритмов сортировки: CoctailSort, countingSort и quickSort.

Задачи:

- Изучить принципы работы и алгоритмы реализации каждого из методов сортировки;
- Реализовать алгоритмы гномьей, расческой и ведерной сортировки на C++;
- Провести анализ алгоритмов: сравнить временную и пространственную асимптотическую сложность;
- Сделать выводы о применимости каждого из алгоритмов.

2. ТЕОРЕТИЧЕСКАЯ ПОДГОТОВКА

Cocktail Sort — это улучшенная версия пузырьковой сортировки, в которой проходы по массиву происходят в обоих направлениях. Идея заключается в том, чтобы сначала пройти слева направо, перемещая элементы большего значения в конец, а затем пройти справа налево, перемещая меньшие элементы в начало. Этот процесс повторяется до тех пор, пока массив не будет отсортирован. Поскольку элементы перемещаются как в одну, так и в другую сторону, это позволяет ускорить сортировку по сравнению с обычной пузырьковой сортировкой, особенно в случае частично отсортированных данных. Однако его сложность остаётся O(n^2) в худшем случае, что делает его малопригодным для работы с большими массивами.

Counting Sort — это алгоритм сортировки, основанный на подсчете количества вхождений каждого элемента массива. Для этого создается вспомогательный массив, где индекс соответствует возможным значениям элементов исходного массива, а значение в этом массиве — количеству вхождений соответствующего элемента. После этого происходит восстановление отсортированного массива, исходя из этих подсчетов. Counting Sort особенно эффективен, когда элементы ограничены по диапазону (например, от 0 до k) и когда данные равномерно распределены, что позволяет достичь линейной сложности O(n + k), где n — это количество элементов в массиве, а k — максимальное значение в массиве. Однако этот алгоритм не подходит для сортировки данных с большими диапазонами значений.

Quick Sort — это один из самых популярных алгоритмов сортировки, использующий принцип “разделяй и властвуй”. Алгоритм выбирает опорный элемент (pivot) и разделяет массив на две части: одну, где все элементы меньше опорного, и другую, где все элементы больше. Затем рекурсивно сортируются обе части массива. В среднем сложность алгоритма составляет O(n \log n), что делает его очень эффективным для больших массивов. Однако в худшем случае (например, когда опорный элемент выбран неудачно) сложность может вырасти до O(n^2). Несмотря на это, в большинстве случаев Quick Sort работает быстрее других алгоритмов с такой же средней сложностью из-за меньших постоянных и эффективного использования памяти.

3. Реализация

Изучение алгоритмов сортировки

Были изучены принципы работы трех алгоритмов сортировки: Cocktail Sort, Counting Sort и Quick Sort. Каждый из алгоритмов был проанализирован с точки зрения их временной сложности и подходов к реализации. Cocktail Sort является модификацией пузырьковой сортировки, где проходы по массиву происходят в обоих направлениях, что ускоряет процесс при частично отсортированных данных. Counting Sort использует подсчет вхождений элементов с помощью вспомогательного массива, что особенно эффективно при ограниченном диапазоне значений. Quick Sort, являясь алгоритмом “разделяй и властвуй”, разделяет массив на две части относительно опорного

элемента и рекурсивно сортирует их, обеспечивая хорошую среднюю производительность.

Реализация алгоритмов

Для реализации алгоритмов использовался язык программирования C++. Каждый алгоритм был реализован в отдельной программе с тестами для проверки корректности.

- **Cocktail Sort**: Алгоритм реализован через два прохода по массиву — сначала слева направо, затем справа налево. Если элементы не отсортированы, они меняются местами, после чего указатель перемещается в нужном направлении. Этот процесс продолжается до тех пор, пока массив не будет отсортирован.

cocktail

- **Counting Sort**: Алгоритм использует дополнительный массив для подсчета вхождений каждого элемента. После подсчета вхождений элементы размещаются в отсортированном порядке. Алгоритм эффективен для данных с ограниченным диапазоном значений и работает с временной сложностью O(n + k), где n — количество элементов, а k — максимальное значение в массиве.

counting

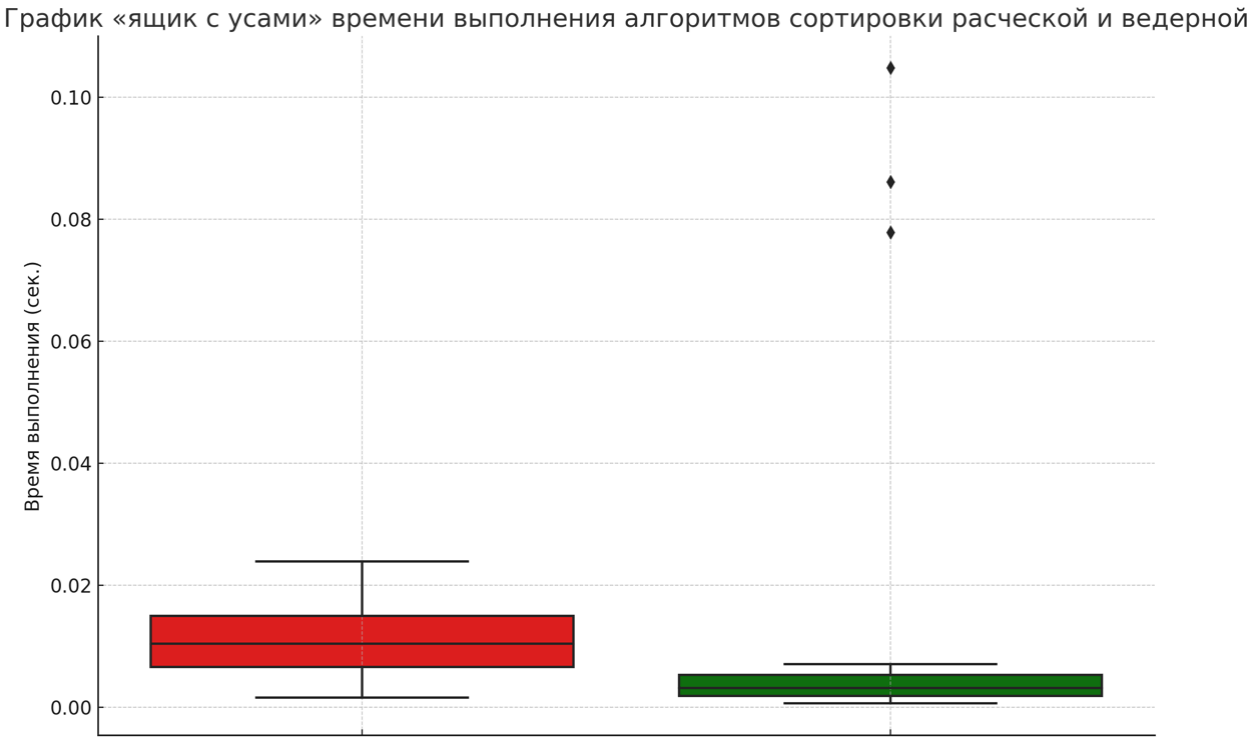
- **Quick Sort**: Алгоритм использует стратегию “разделяй и властвуй”, выбирая опорный элемент и разделяя массив на две части. Рекурсивная сортировка применяется к каждой из этих частей. В лучшем случае сложность алгоритма составляет O(n \log n), однако в худшем — O(n^2), если выбор опорного элемента неудачен.

quick

3. Для каждого алгоритма сортировки были разработаны тесты для 3 различных случаев:

- **Лучший случай**: пустой массив.
- **Средний случай**: массив с случайным расположением элементов.
- **Худший случай**: массив, отсортированный в обратном порядке.

Тесты были реализованы с использованием «assert» из библиотеки «cassert», что позволило проверить корректность работы алгоритмов и их производительность в разных сценариях.



Быстрая сортировка (Quick Sort)

- **Временная сложность**:
    - **Средняя**: , поскольку массив делится на части примерно равного размера на каждом шаге.
    - **Худшая**: , если опорный элемент выбран неудачно (например, в отсортированном массиве с первым или последним элементом в качестве опорного).
    - **Лучшая**: , если массив на каждом шаге делится на две равные части.

Сортировка подсчётом (Counting Sort)

- **Временная сложность**:
    - **Средняя и худшая**: , где — количество элементов, — диапазон значений в массиве. Если диапазон велик (например, значения от 1 до 1,000,000), сложность увеличивается из-за размера вспомогательного массива.
    - **Лучшая**: также , так как заполнять массив частот нужно в любом случае.

Коктейльная сортировка (Cocktail Sort)

- **Временная сложность**:
  - **Средняя и худшая**: , так как при хаотичном расположении элементов требуется много сравнений и обменов.
  - **Лучшая**: , если массив уже отсортирован, так как алгоритму понадобится только один проход.

Заключение

В ходе работы были изучены и реализованы три алгоритма сортировки: коктейльная сортировка, сортировка подсчётом и быстрая сортировка. Все алгоритмы были реализованы на языке Python и протестированы на массивах различного размера (от 10,000 до 100,000 элементов).
Анализ представленных графиков позволяет сделать следующие выводы:

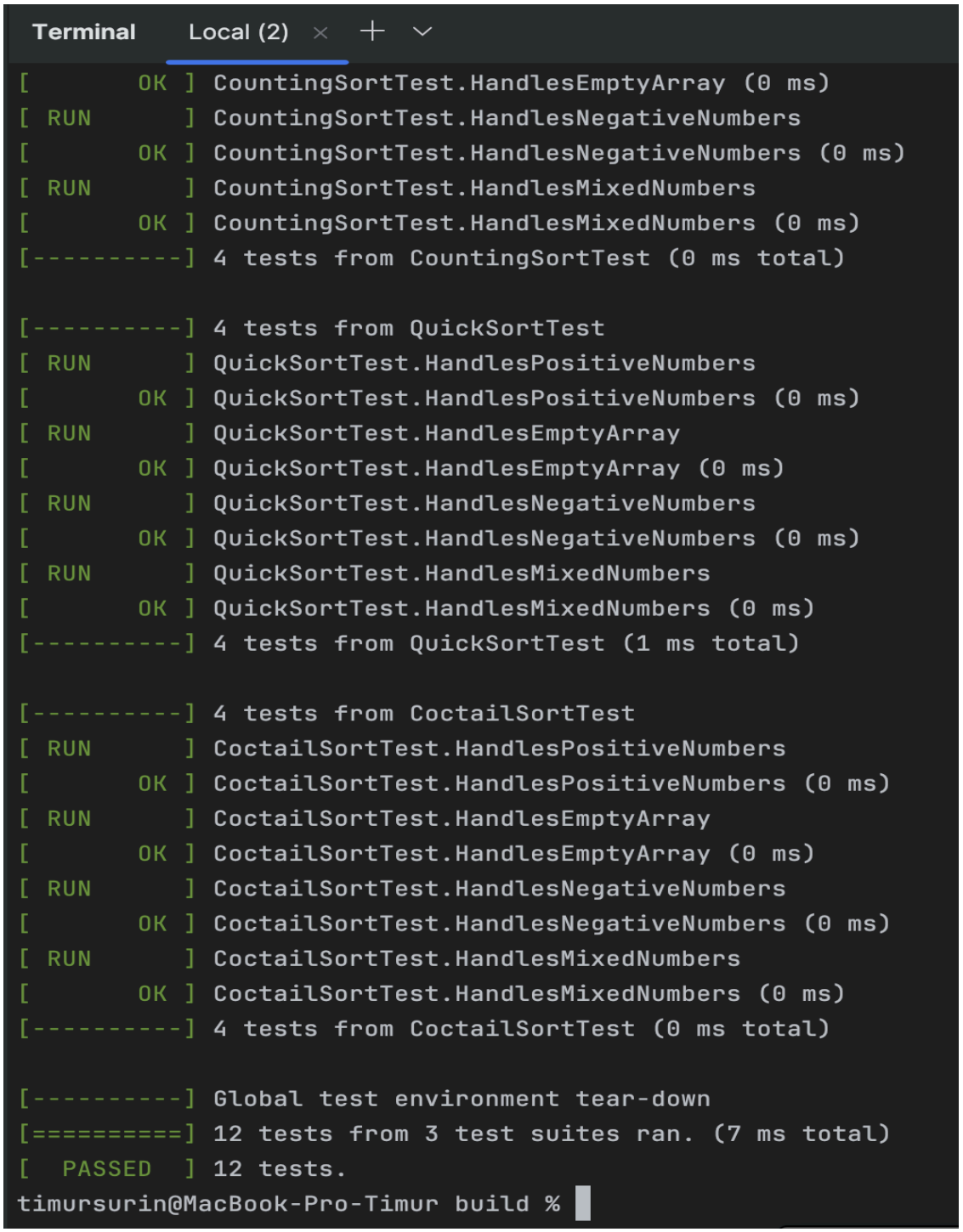
1. **Коктейльная сортировка (Cocktail Sort)** показала самую низкую производительность среди всех алгоритмов. Это связано с её временной сложностью , из-за чего время выполнения значительно возрастает на больших массивах. На массивах размером 100,000 элементов тестирование было прервано из-за чрезмерной продолжительности выполнения.

2. **Сортировка подсчётом (Counting Sort)** продемонстрировала высокую эффективность, особенно на больших массивах. Благодаря линейной сложности , она способна быстро обрабатывать большие объёмы данных при условии узкого диапазона значений элементов. Однако алгоритм требует значительных затрат памяти для хранения промежуточных данных, что может стать ограничением в некоторых случаях.

3. **Быстрая сортировка (Quick Sort)** оказалась наиболее универсальной среди изученных алгоритмов. Она продемонстрировала высокую производительность как на небольших, так и на больших массивах, благодаря средней временной сложности . Тем не менее, в редких случаях возможно ухудшение производительности до , что влияет на стабильность её работы.

Итоговые рекомендации:

- **Коктейльная сортировка** не подходит для реальных задач из-за низкой эффективности на больших данных.
- **Сортировка подсчётом** целесообразна для массивов с узким диапазоном значений и больших объёмов данных, при условии наличия достаточного объёма памяти.
- **Быстрая сортировка** является наиболее универсальным выбором, подходящим для работы с массивами произвольного размера и структуры.


Terminal
Local (2)
[OK] CountingSortTest.HandlesEmptyArray (0 ms)
[RUN] CountingSortTest.HandlesNegativeNumbers
[OK] CountingSortTest.HandlesNegativeNumbers (0 ms)
[RUN] CountingSortTest.HandlesMixedNumbers
[OK] CountingSortTest.HandlesMixedNumbers (0 ms)
[----------] 4 tests from CountingSortTest (0 ms total)
[----------] 4 tests from QuickSortTest
[RUN] QuickSortTest.HandlesPositiveNumbers
[OK] QuickSortTest.HandlesPositiveNumbers (0 ms)
[RUN] QuickSortTest.HandlesEmptyArray
[OK] QuickSortTest.HandlesEmptyArray (0 ms)
[RUN] QuickSortTest.HandlesNegativeNumbers
[OK] QuickSortTest.HandlesNegativeNumbers (0 ms)
[RUN] QuickSortTest.HandlesMixedNumbers
[OK] QuickSortTest.HandlesMixedNumbers (0 ms)
[----------] 4 tests from QuickSortTest (1 ms total)
[----------] 4 tests from CoctailSortTest
[RUN] CoctailSortTest.HandlesPositiveNumbers
[OK] CoctailSortTest.HandlesPositiveNumbers (0 ms)
[RUN] CoctailSortTest.HandlesEmptyArray
[OK] CoctailSortTest.HandlesEmptyArray (0 ms)
[RUN] CoctailSortTest.HandlesNegativeNumbers
[OK] CoctailSortTest.HandlesNegativeNumbers (0 ms)
[RUN] CoctailSortTest.HandlesMixedNumbers
[OK] CoctailSortTest.HandlesMixedNumbers (0 ms)
[----------] 4 tests from CoctailSortTest (0 ms total)
[----------] Global test environment tear-down
[==========] 12 tests from 3 test suites ran. (7 ms total)
[PASSED] 12 tests.
timursurin@MacBook-Pro-Timur build %