ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Сравнение алгоритмов сортировки»

Выполнил работу:

Апрелков Михаил

Академическая группа №J3113

Принято:

Дунаев Максим Владимирович

Санкт-Петербург

2024

1. ВВЕДЕНИЕ

Цель работы: изучение и сравнение трех алгоритмов сортировки: Сортировка подсчетом (Counting Sort), Быстрая сортировка (Quick Sort) и Коктейльная сортировка (Cocktail Sort).

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

В данной работе рассматриваются три алгоритма сортировки:

1. Метод: countingSort

**Описание**: Сортировка подсчетом — это алгоритм, который работает эффективно с массивами целых чисел небольшого диапазона значений. Она подсчитывает количество вхождений каждого числа и использует эту информацию для упорядочивания элементов.

1. Метод: quicksort

**Описание**: Быстрая сортировка — это рекурсивный алгоритм, использующий стратегию "разделяй и властвуй". Она выбирает опорный элемент (в данном коде — средний элемент) и разделяет массив на части: элементы меньше опорного и элементы больше опорного.

1. Метод: CoctailSort

**Описание**: Коктейльная сортировка — это модификация сортировки пузырьком. Она проходит по массиву в обоих направлениях, упорядочивая элементы в процессе "взад и вперед". Это позволяет быстрее перемещать большие элементы к своим позициям.

3. РЕАЛИЗАЦИЯ

1. Изучение алгоритмов сортировки

Были изучены принципы работы трех алгоритмов: сортировка подсчетом (Counting Sort), быстрая сортировка (Quick Sort) и коктейльная сортировка (Cocktail Sort). Каждый из алгоритмов был проанализирован с точки зрения временной сложности и подходов к реализации.

2. Реализация алгоритмов

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

Сортировка подсчётом (Counting Sort): Алгоритм подсчитывает количество вхождений каждого числа и использует эту информацию для упорядочивания элементов.

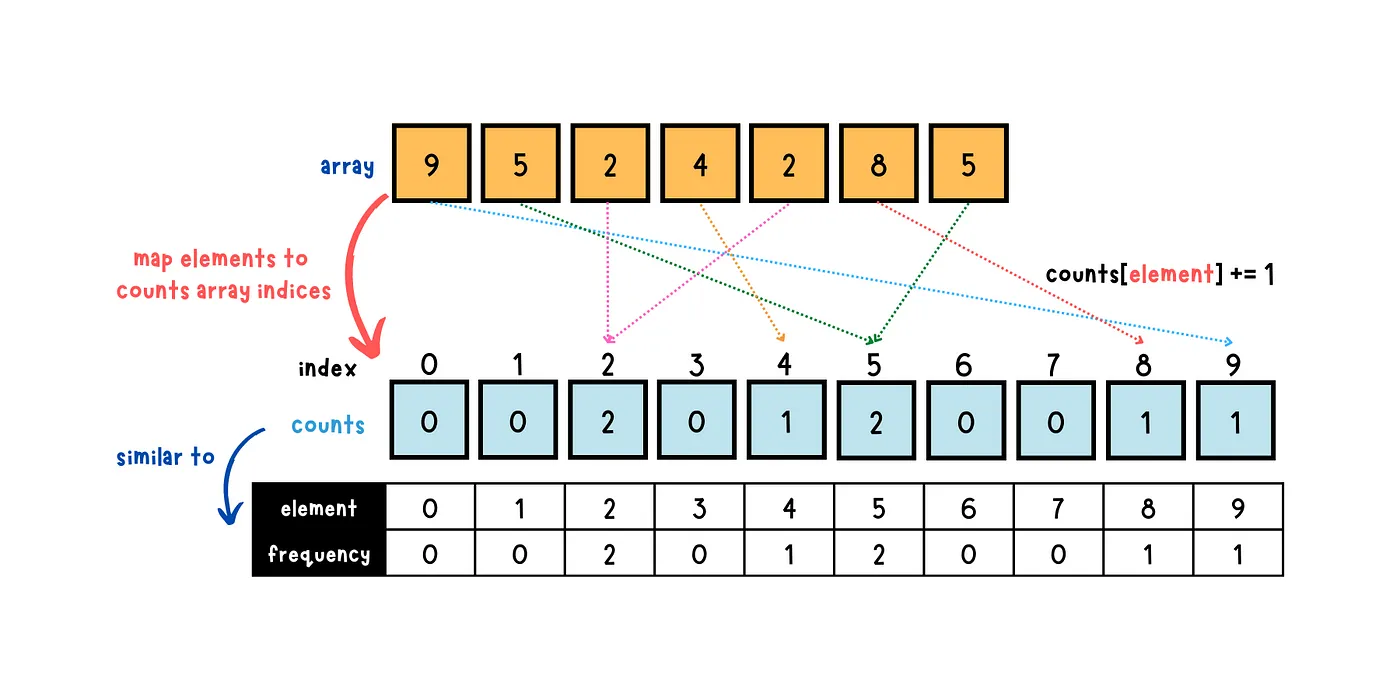


Рис. 1 Наглядный пример работы сортировки подсчётом.

Быстрая сортировка (QuickSort): Алгоритм выбирает опорный элемент (в данном коде — средний элемент) и разделяет массив на части: элементы меньше опорного и элементы больше опорного.

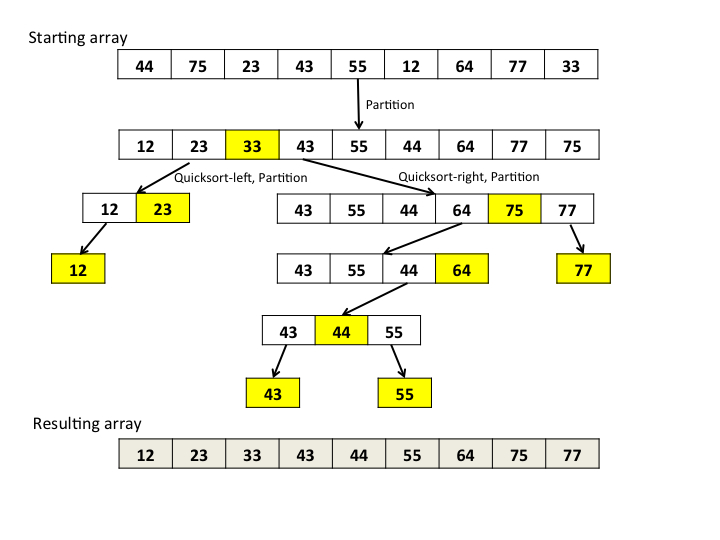


Рис. 2Наглядный пример работы быстрой сортировки.

Коктейльная сортировка: Алгоритм проходит по массиву в обоих направлениях, упорядочивая элементы в процессе "взад и вперед". Это позволяет быстрее перемещать большие элементы к своим позициям.

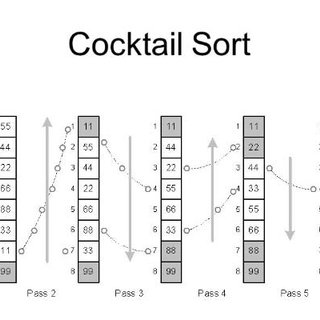
****

Рис. 3Наглядный пример работы вёдерной сортировки

3. Для каждого алгоритма сортировки были разработаны тесты для 4 различных случаев:

- Лучший случай: пустой массив;

- Средний случай: рандомное расположение элементов;

- Худший случай: обратно отсортированный массив.

Тесты были реализованы с помощью gtest при помощи googletest.

4. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

Рассмотрим результаты разработанных алгоритмов. Для оценки производительности алгоритмов сортировки были рассчитаны асимптотики каждого алгоритма, использование памяти и замерена скорость выполнения алгоритмов на разных наборах данных (размером от 1 тыс. до 1 млн. элементов с шагом 1 тыс. элементов) с помощью встроенной библиотеки chrono.

**Быстрая сортировка (Quick Sort)**

**Временная сложность**:

**Средняя сложность**: O(n log n) так как массив делится на части примерно одинакового размера.

**Худшая сложность**: O(n^2), если опорный элемент неудачно выбран (например, в случае отсортированного массива и опорного элемента — первого или последнего элемента).

**Лучшая сложность**: O(n log n), если массив равномерно делится на две части на каждом шаге.

**Сортировка подсчетом (Counting Sort)**

**Временная сложность**:

**Средняя и худшая сложность**: O(n + k), где n — количество элементов массива, а k — диапазон значений в массиве. Если k велико (например, массив от 1 до миллиона), сложность увеличивается из-за большого размера вспомогательного массива.

**Лучшая сложность** тоже O(n + k), поскольку массив частот все равно нужно заполнить.

**Коктейльная сортировка (Cocktail Sort)**

**Временная сложность**:

**Средняя и худшая сложность**: O(n^2), поскольку при случайной расстановке элементов требуется множество сравнений и перестановок.

**Лучшая сложность**: O(n), если массив уже отсортирован, так как потребуется всего один проход по массиву.

Для тестирования алгоритмов было подготовлено 1000 наборов данных, размером от 1000 элементов до 1 000 000. Все тесты проводились в равных условиях, в результате получились такие результаты:

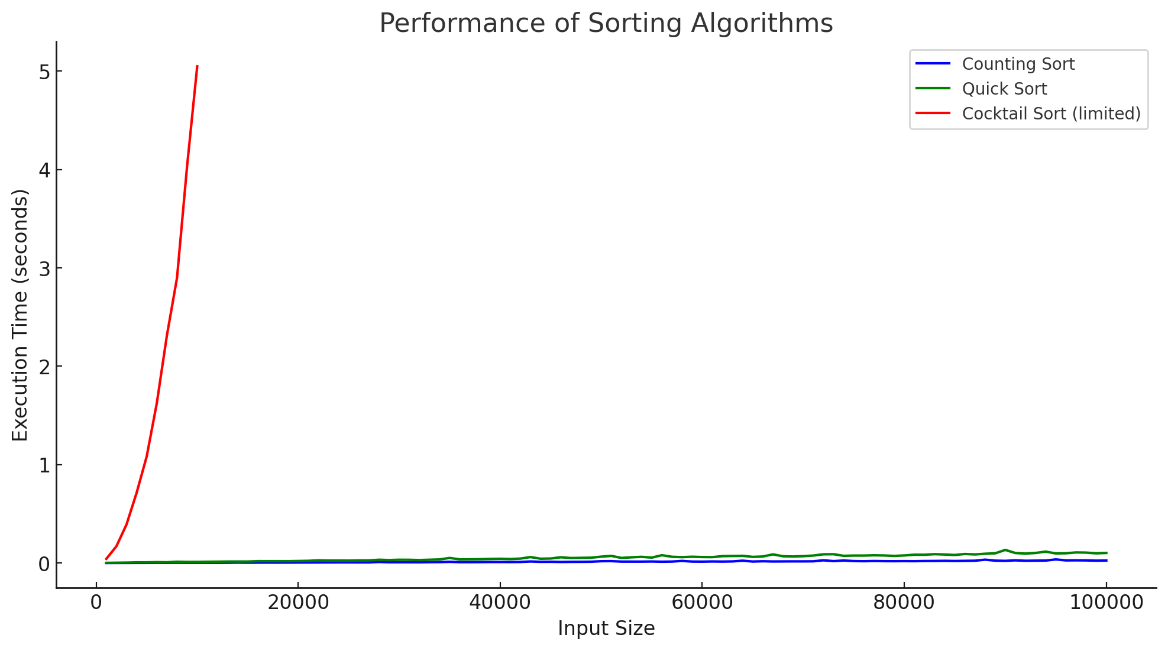
****

Рис. 4 График зависимости времени работы всех трёх сортировок от размера входного массива

Также для проверки стабильности работы алгоритмов сортировки, было проведено 100 запусков каждого алгоритма на одном и том же наборе данных размеров в 10000 и 100000 элементов:

Алгоритм Коктейль сортировки не был использован в этом анализе так как он является самым худшим на большом наборе данных.

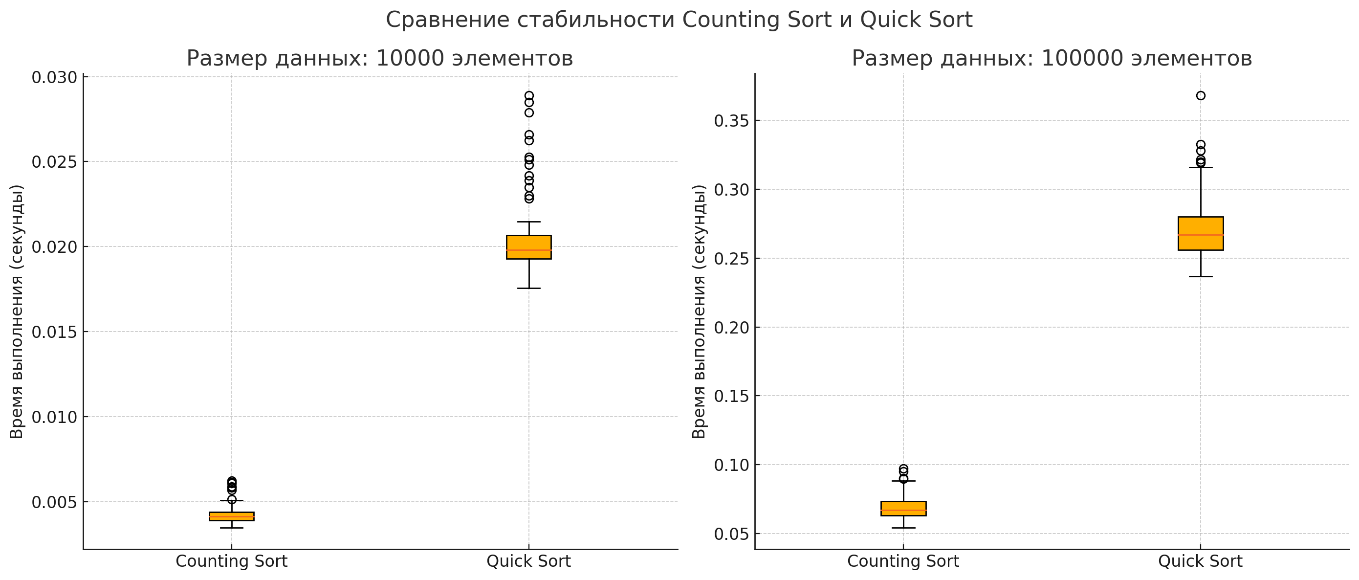
****

Рис. 5 График «ящик с усами» времени выполнения двух алгоритмов сортировки подсчётом и быстрой сортировки на двух разных наборах данных.

5. ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были изучены и реализованы три алгоритма сортировки: коктейльная сортировка, сортировка подсчётом и быстрая сортировка. Реализация была выполнена на языке Python с использованием тестирования на массивах различного размера (от 10,000 до 100,000 элементов).

Анализируя представленные графики, можно сделать несколько выводов по алгоритмам сортировки:

1. **Коктейльная сортировка (Cocktail Sort)** демонстрирует самую низкую производительность среди всех рассмотренных алгоритмов. Это объясняется её временной сложностью O(N^2), что приводит к значительному увеличению времени выполнения на больших массивах. Алгоритм оказался настолько неэффективным, что тестирование на массивах размером 100,000 элементов было прервано из-за слишком долгого выполнения.
2. **Сортировка подсчётом (Counting Sort)** показывает высокую производительность, особенно на больших массивах. Её линейная сложность O(N + K) позволяет эффективно обрабатывать большие объёмы данных при условии, что диапазон значений элементов массива невелик. Однако сортировка требует значительных затрат памяти для хранения промежуточных результатов, что может быть ограничением для её использования.
3. **Быстрая сортировка (Quick Sort)** является наиболее универсальной среди представленных алгоритмов. Она демонстрирует хорошую производительность как на небольших, так и на больших массивах, благодаря своей средней временной сложности O(N log N). Тем не менее, в редких случаях возможно ухудшение до O(N^2), что влияет на стабильность времени выполнения.

Таким образом, коктейльная сортировка не рекомендуется для использования в реальных задачах из-за её низкой эффективности. Сортировка подсчётом предпочтительна для массивов с узким диапазоном значений и больших объёмов данных, если доступны достаточные ресурсы памяти. Быстрая сортировка является наиболее подходящей для универсального использования, особенно на массивах произвольного размера и структуры.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание