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

Отчёт по лабораторной работе № 5

«Сравнение алгоритмов сортировки»

Выполнил работу

Баранов Владимир

Академическая группа №J3112

Принято

Дунаев Максим Владимирович

Санкт-Петербург

2024

1. ВВЕДЕНИЕ

Цель работы: изучение и сравнение трех алгоритмов сортировки: гномьей сортировки, сортировки расческой (combSort) и ведерной сортировки (bucketSort).

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. Гномья сортировка – простой алгоритм, основанный на сравнении и перестановке соседних элементов массива. Основная идея заключается в последовательной проверке пар элементов: если элементы находятся не в порядке, они меняются местами, а указатель возвращается назад, иначе переходит вперед.
2. Сортировка расческой (combSort) – усовершенствованный вариант пузырьковой сортировки, где для ускорения процесса на начальных этапах используется увеличенный шаг между сравниваемыми элементами, который постепенно уменьшается. Это помогает быстрее перемещать элементы ближе к их конечным позициям.
3. Вёдерная сортировка (BucketSort) – алгоритм, который делит элементы массива на несколько групп (ведер), основываясь на их значениях. Затем элементы внутри каждого ведра сортируются, и все ведра объединяются для формирования итогового массива. Этот метод особенно эффективен для равномерно распределенных данных.

3. РЕАЛИЗАЦИЯ

1. Изучение алгоритмов сортировки

Были изучены принципы работы трех алгоритмов: гномьей сортировки, сортировки расческой и вёдерной сортировки (bucketSort). 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.

Гномья сортировка: Алгоритм реализован через цикл, который перемещает указатель на элементы массива вперед или назад, в зависимости от сравнения текущего и предыдущего элементов.

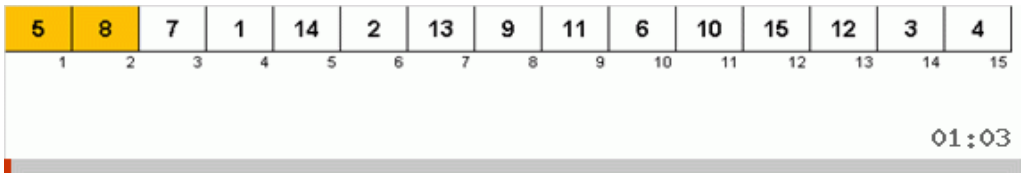

Рис. 1 Наглядный пример работы гномьей сортировки.

Сортировка расческой: Алгоритм основан на уменьшении шага между сравниваемыми элементами (gap), начиная с большого значения и постепенно уменьшая его, пока он не станет равен единице. В основе лежит идея оптимизации пузырьковой сортировки.

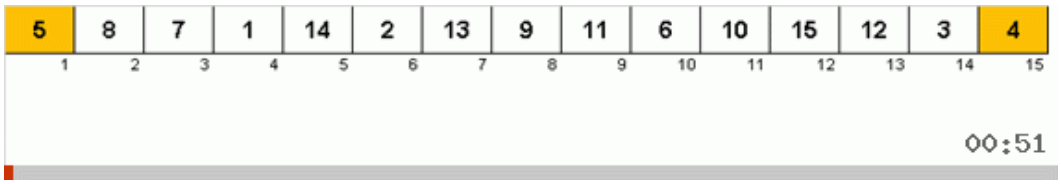

Рис. 2 Наглядный пример работы сортировки расческой.

Ведерная сортировка: Алгоритм, который распределяет элементы массива по нескольким группам (ведрам), основываясь на их значениях. Затем элементы внутри каждого ведра сортируются вставками, и все элементы из ведер объединяются в итоговый отсортированный массив.

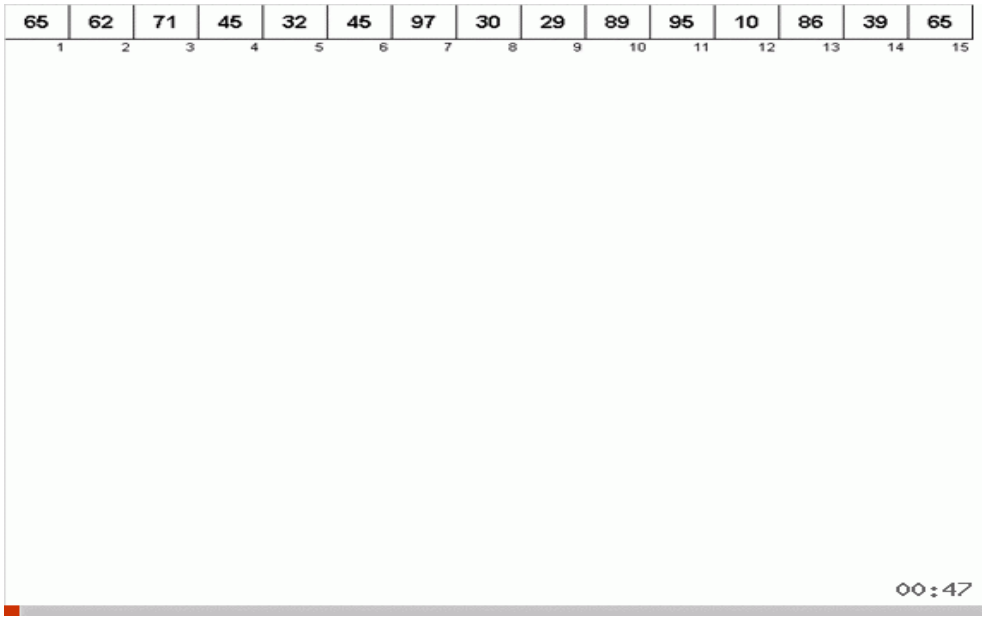

Рис. 3 Наглядный пример работы вёдерной сортировки

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

- Лучший случай: уже отсортированный массив;
- Средний случай: рандомное расположение элементов;
- Худший случай: обратно отсортированный массив.

Тесты были реализованы с помощью «assert» из библиотеки «cassert».

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.

Гномья сортировка последовательно сравнивает соседние элементы и перемещается по массиву, если элементы находятся в порядке, или возвращается назад, если порядок нарушен. В итоге, средняя асимптотика (она и худшая) получилась $O(N^2)$ для случайной расстановки элементов, поскольку перестановки происходят многократно на случайных позициях.

По памяти гномья сортировка выполняется на месте, так как все операции происходят внутри исходного массива без использования дополнительных структур данных. Память для массива – O(N).

Алгоритм сортировки расческой оптимизирует пузырьковую сортировку за счет начального использования большого шага (gap), который постепенно уменьшается. Если массив уже отсортирован, алгоритм выполняет один проход с уменьшением шага и не производит перестановок. В этом случае временная сложность составляет O(NlogN), так как шаг уменьшается примерно экспоненциально. В среднем сложность алгоритма составит $O(N^{1.247})$, где 1.247 является фактором уменьшения. Данное значение было выявлено экспериментальным путем.

Как и гномья сортировка, сортировка расческой выполняется на месте, используя входной массив без дополнительных структур данных. Использование памяти – O(N).

В ведерной сортировке, если элементы распределены равномерно по ведрам, и каждое ведро содержит мало элементов, сортировка вставками внутри ведер выполняется быстро. Общая сложность будет O(N+k), где k будет количеством ведер. Потребление памяти будет O(2N), так как используется память для основного массива и для ведер.

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:

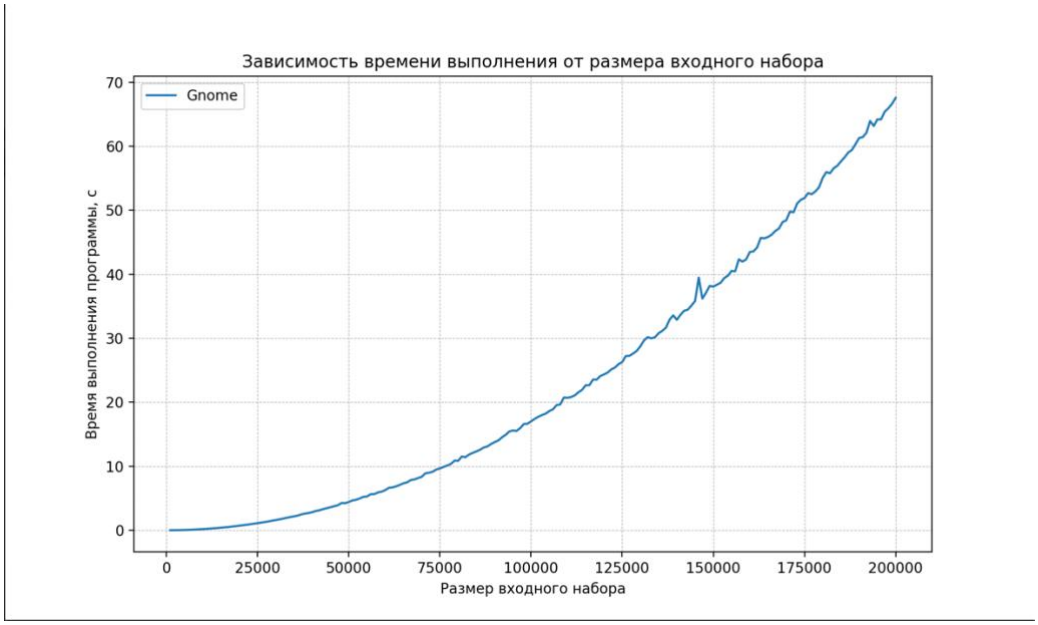

Рис. 4 График зависимости времени работы гномьей сортировки от размера входного набора

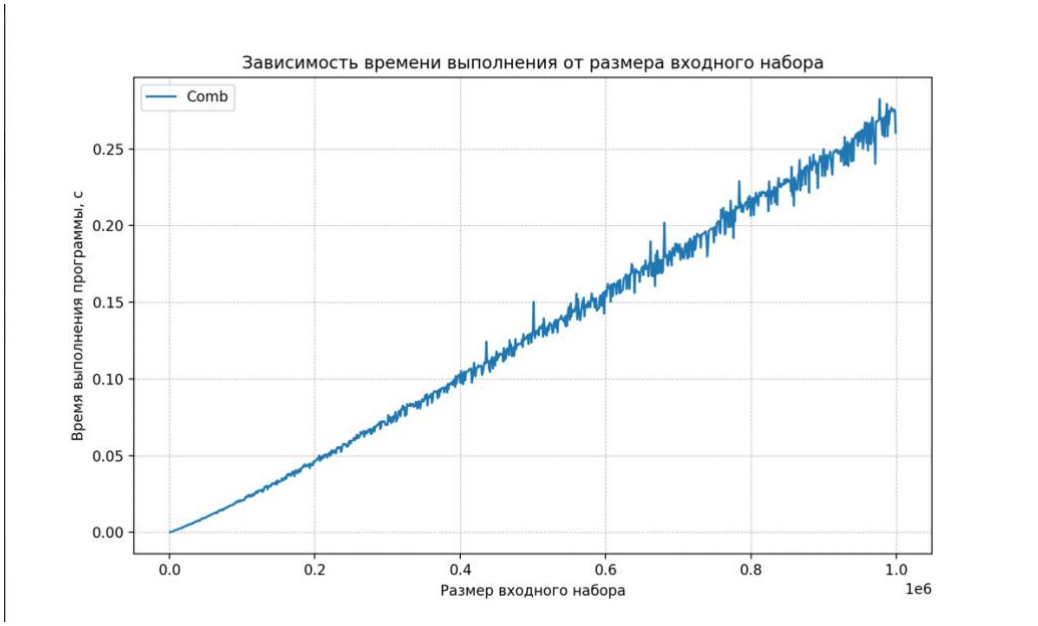

Рис. 5 График зависимости времени работы сортировки расческой от размера входного набора

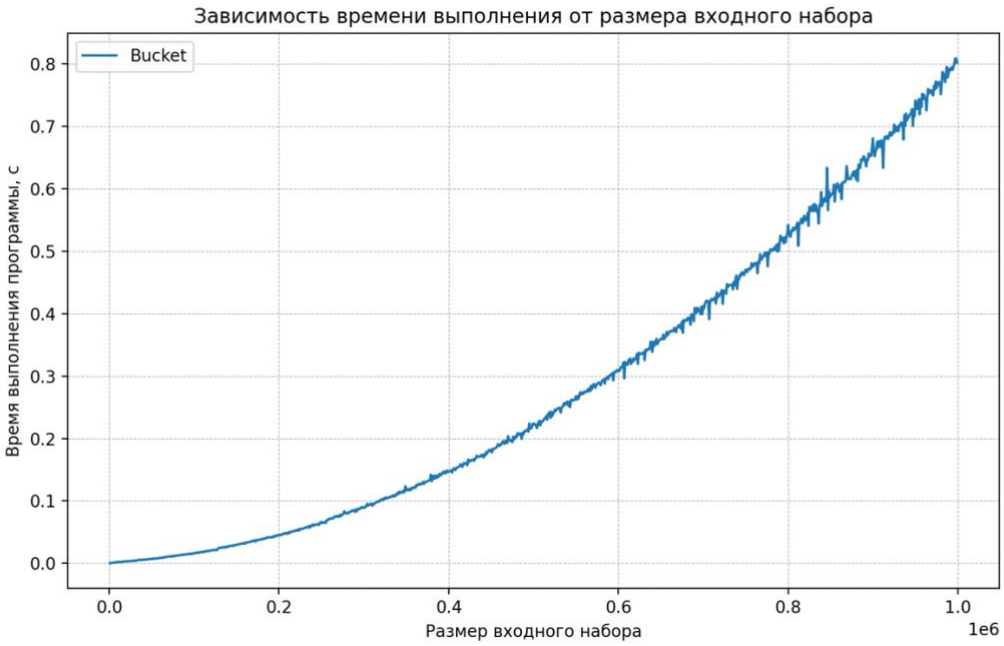

Рис. 6 График зависимости времени работы ведерной сортировки от размера входного набора

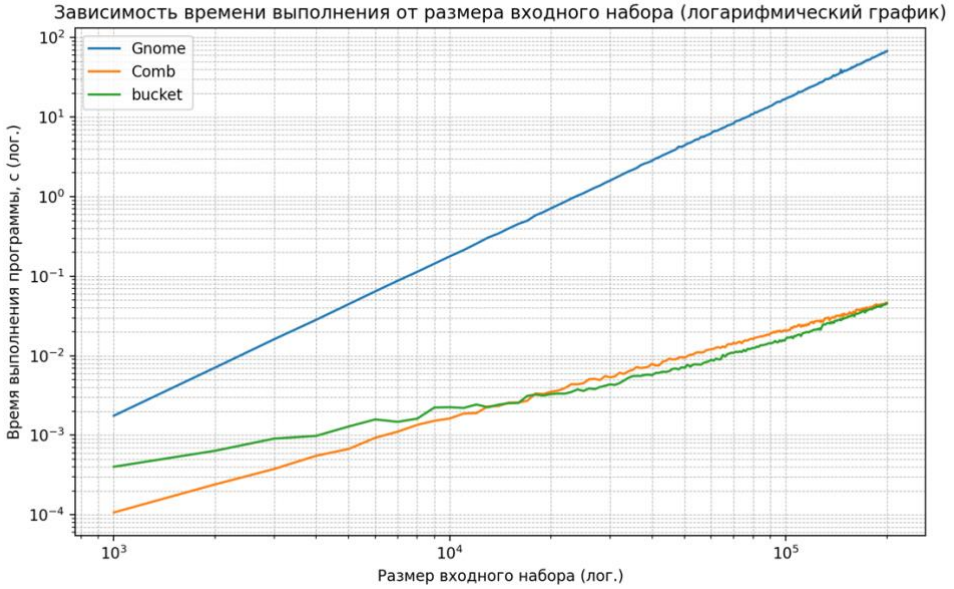

Рис. 7 Логарифмическая шкала сравнения времени работы всех 3-х сортировок от размера входного набора

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:

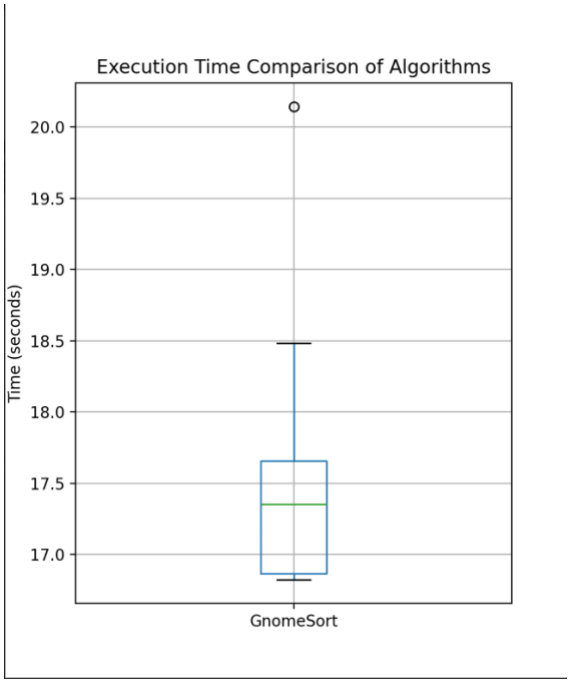

Рис. 8 График «ящик с усами» времени выполнения алгоритма гномьей сортировки на одном и том же наборе данных

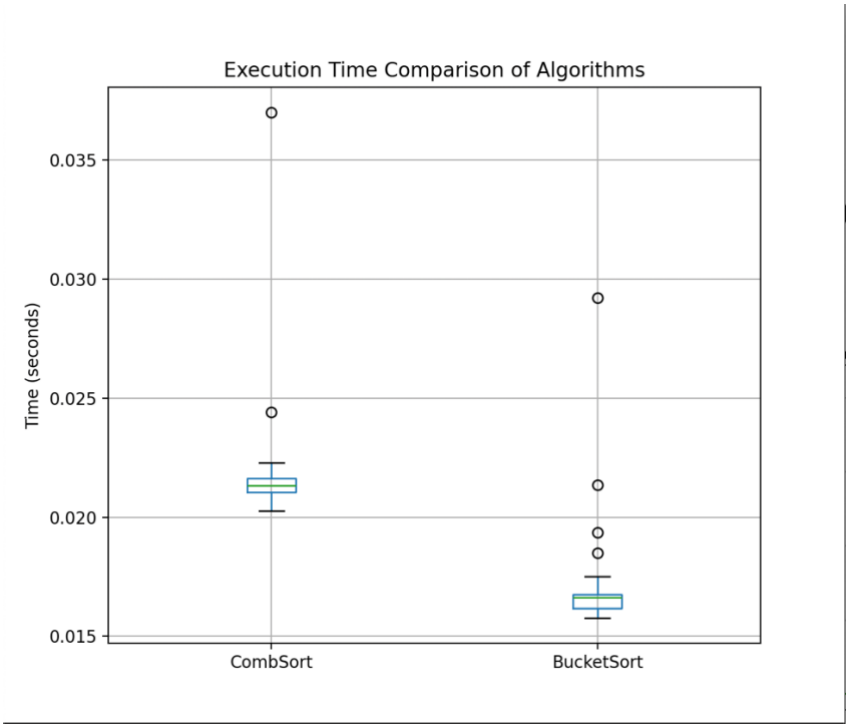

Рис. 9 График «ящик с усами» времени выполнения алгоритма сортировки расческой и ведерной на одном и том же наборе данных

5. ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были изучены и реализованы три алгоритма сортировки: гномья сортировка, сортировка расческой и ведерная сортировка. Реализация была выполнена на языке C++ с использованием тестирования на массивах различного размера (от 1000 до 1 000 000 элементов).

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

1. Гномья сортировка (GnomeSort) демонстрирует самую низкую производительность. Это видно на графиках, где время выполнения увеличивается квадратично с ростом объема данных. Алгоритм неэффективен для больших массивов, что подтверждается его временной сложностью $O(N^2)$.
2. Сортировка расческой (CombSort) показывает значительно более высокую производительность, чем гномья сортировка. Алгоритм имеет временную сложность около $O(n^{1.247})$, что подтверждается около линейным ростом времени на логарифмическом графике.
3. Ведерная сортировка (BucketSort) является самой быстрой среди рассмотренных алгоритмов на больших массивах, что объясняется линейной сложностью O(N) в лучшем случае. Однако для небольших наборов данных разница между CombSort и BucketSort минимальна.

Алгоритмы ведерной и сортировки расческой также показывают более стабильную работу по времени, нежели чем гномья сортировка.

Таким образом, гномья сортировка не рекомендуется для использования в реальных задачах из-за ее низкой эффективности, для небольших массивов или массивов без строгих требований к скорости сортировки можно использовать сортировку расческой из-за ее простоты реализации, а для больших массивов предпочтительнее использовать ведерную сортировку, так как она демонстрирует лучшую производительность при больших объемах данных.