ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Алгоритмы сортировки»

Выполнил работу

Бондаренко Анна

Академическая группа №J3112

Принято

Дунаев Максим

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель: Написание и анализ трех алгоритмов с разной асимптотикой

сортировки: гномья сортировка, быстрая сортировка и сортировка подсчётом.

Задачи:

1) Написать алгоритмы сортировки.

2) Построить графики времени по результатам запуска программ.

3) Произвести подсчеты по памяти и асимптотике.

4) Проанализировать графики и сравнить их с ожидаемой асимптотикой.

2. Теоретическая подготовка

В данное лабораторной работе были использованы типы данных: int(целые

числа), vector(динамический массив целых чисел), array(массив целых чисел);

библиотеки: iosream, vector, fstream, ctime, cassert и cstdlib . В ходе выполнения работы были использованы алгоритмы сортировки подсчётом, гномьей сортировки и быстрой сортировки.

3. Реализация

1. Написать алгоритмы, протестировав их на сгенерированных массивах.

2. Используя данные о работе алгоритмов на массивах разной длины,

построить графики с помощью библиотеки Python Matplotlib .

3. Оценить алгоритмы с точки зрения асимптотики и затрат памяти.

4. Сравнить ожидаемые результаты асимптотики и с получившимися

графиками и сделать выводы о том, насколько ожидания сходятся с реальными

данными.

4. Экспериментальная часть

1) Посчитаем память и асимптотику для гномьей сортировки:

Время работы

**1. Лучший случай**:

Алгоритм завершится, если элементы уже отсортированы. В этом случае он пройдет по массиву один раз, сравнив элементы, и все. Время работы — **O(n)**, где n — количество элементов в массиве.

**2. Худший случай**:

Когда элементы расположены в обратном порядке (или почти в нем), алгоритм будет работать по принципу "перетасовывания" элементов, часто возвращаясь назад. В худшем случае он может пройти через массив несколько раз, из-за чего время работы будет равно **O(n^2)**.

**3. Подсчет шагов**:

В худшем случае, каждый элемент будет сравниваться и перемещаться несколько раз. На каждом шаге мы либо двигаемся вперед (при сравнении с предыдущим элементом), либо назад (при перестановке элементов). В худшем случае алгоритм выполняет операцию "обратного прохода" несколько раз для каждого элемента, что приводит к квадратичной сложности **O(n^2)**.

Таким образом:

1. **Лучший случай**: O(n)
2. **Худший случай**: O(n^2)
3. **Средний случай**: O(n^2)

**Память**

Алгоритм использует **O(1)** дополнительной памяти, так как выполняет сортировку на месте, и не требует дополнительных структур данных.

2) Посчитаем память и асимптотику для быстрой сортировки:

Время работы

1. **Лучший случай**

1. В лучшем случае, каждый раз массив делится на две равные части. Это происходит, если опорный элемент делит массив идеально пополам.
2. В этом случае, на каждом уровне рекурсии количество элементов, которые нужно обработать, сокращается вдвое.
3. Количество уровней рекурсии будет равно log⁡n, и на каждом уровне нужно выполнить O(n) операций (для каждого элемента массива).
4. В итоге, общее время работы для лучшего случая: **O(n log n)**.

2. **Худший случай**

1. В худшем случае, если опорный элемент каждый раз будет либо самым маленьким, либо самым большим элементом, то массив делится на одну большую часть и одну маленькую. Это приводит к рекурсии на n−1 элементах, а на каждом уровне необходимо выполнить O(n) операций для разбиения.
2. В худшем случае количество уровней рекурсии будет равно n, что дает общее время работы **O(n^2)**.

3. **Средний случай**

* В среднем, опорный элемент делит массив не идеально, но и не совсем неудачно, поэтому число уровней рекурсии будет примерно log⁡n, а на каждом уровне потребуется обработка всех n элементов.
* Таким образом, средняя сложность остается **O(n log n)**.

Итог по времени:

1. **Лучший случай**: **O(n log n)**
2. **Худший случай**: **O(n^2)**
3. **Средний случай**: **O(n log n)**

**Память**

1. **Дополнительная память**

* Quicksort работает с рекурсией, и каждый рекурсивный вызов требует дополнительной памяти для хранения состояния вызова функции.
* В лучшем случае (когда массив делится на равные части), глубина рекурсии будет log⁡n, и следовательно, память, необходимая для хранения стеков вызовов, будет **O(log n)**.
* В худшем случае (когда массив делится на части, состоящие почти из одного элемента), глубина рекурсии будет n, что требует **O(n)** памяти.

2. **Память для разбиения**

* Сортировка не использует дополнительную память для хранения временных массивов (кроме стека рекурсии), поэтому память, используемая для обработки массива, — **O(1)**.

Итог по памяти:

1. **Лучший случай**: **O(log n)** (глубина рекурсии)
2. **Худший случай**: **O(n)** (глубина рекурсии)
3. **Средний случай**: **O(log n)** (глубина рекурсии)

3) Посчитаем память и асимптотику для сортировки с подсчётом:

Время работы:

**O(n + range)**, где:

* 1. n — количество элементов в массиве.
  2. range = maxVal - minVal + 1 — диапазон значений элементов (разница между максимальным и минимальным значением).

Время работы **сортировки подсчетом** зависит от диапазона значений элементов. Если диапазон значений range значительно больше, чем количество элементов n, то алгоритм может быть неэффективным (например, если значения сильно разбросаны по широкому диапазону).

**1) Лучший случай**

Все элементы в массиве могут быть одинаковыми или иметь небольшой диапазон значений, что делает алгоритм очень быстрым. Тогда сложность будет **O(n + range)**, где range маленький.

**2) Худший случай**

Когда диапазон значений очень велик (например, числа от 1 до 10^6 при маленьком числе элементов), сложность может стать **O(n + range)**, где range велико.

Память:

1. **Массив**count:

Массив count имеет размер, равный диапазону значений, то есть **O(range)**. Это может быть значительным потреблением памяти, если диапазон значений велик.

1. **Массив**output:

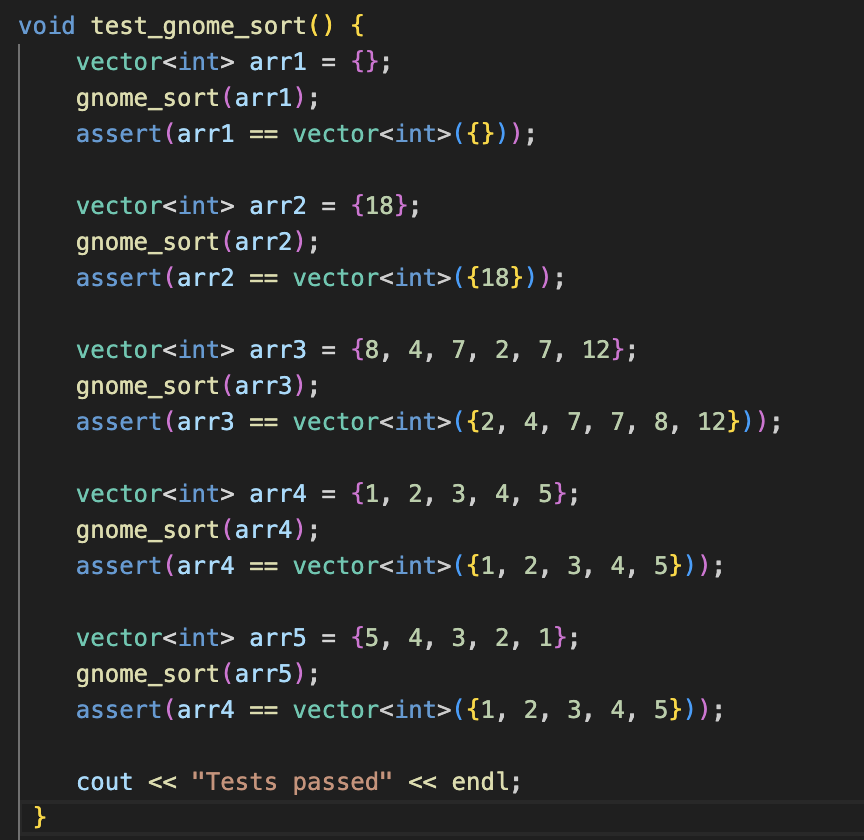
Массив output имеет тот же размер, что и исходный массив, и используется для хранения отсортированных элементов. Память для этого массива — **O(n)**.

1. **Дополнительная память**:

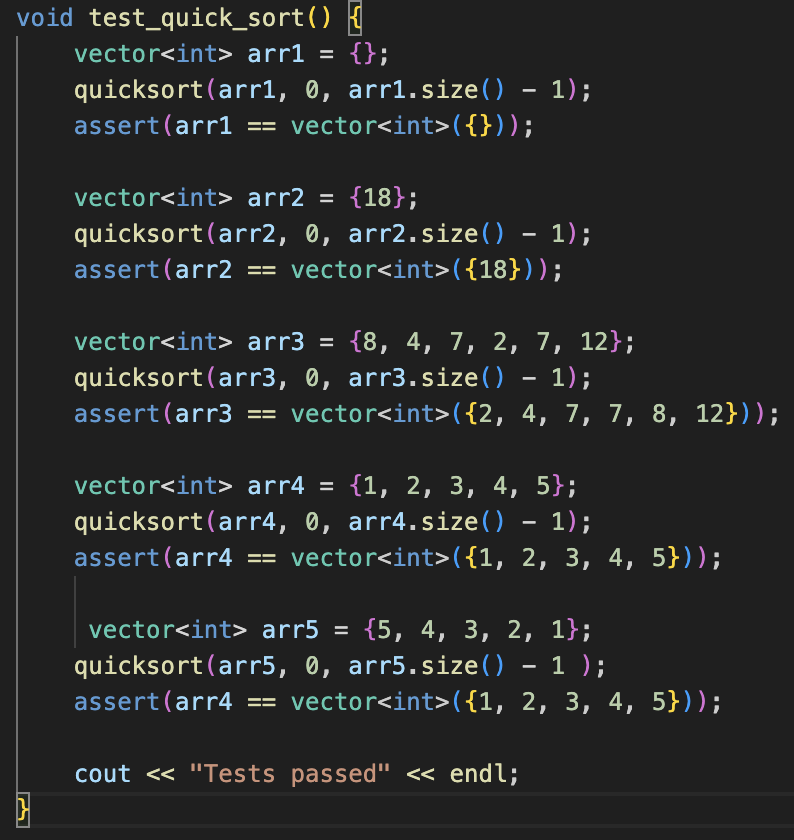
В данном алгоритме основное потребление памяти связано с массивами count и output, и оно равно **O(n + range)**.

Тесты:

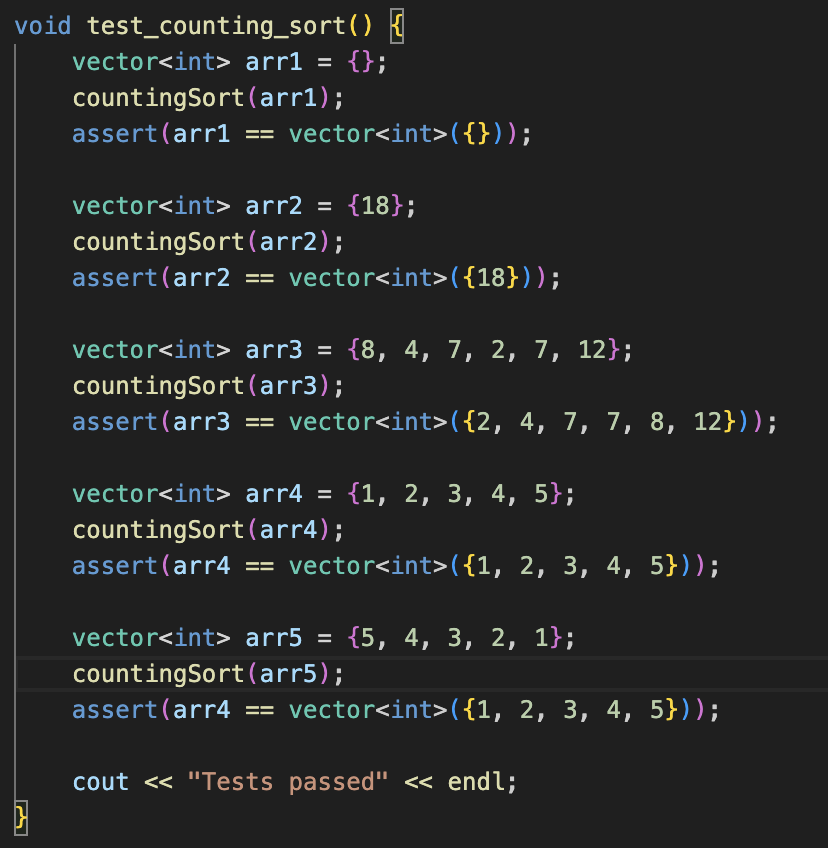
Тесты для гномьей сортировки представлены в листинге части кода на изображении №1, для быстрой сортировки на изображении №2 и для сортировки с подсчётом на изображении №3.



Изображение №1- тесты для гномьей сортировки



Изображение №2 — тесты для быстрой сортировки



Изображение №3 — тесты для сортировки с подсчётом

График зависимости времени от числа элементов.

Согласно требованиям моего варианта, на вход к моему алгоритму

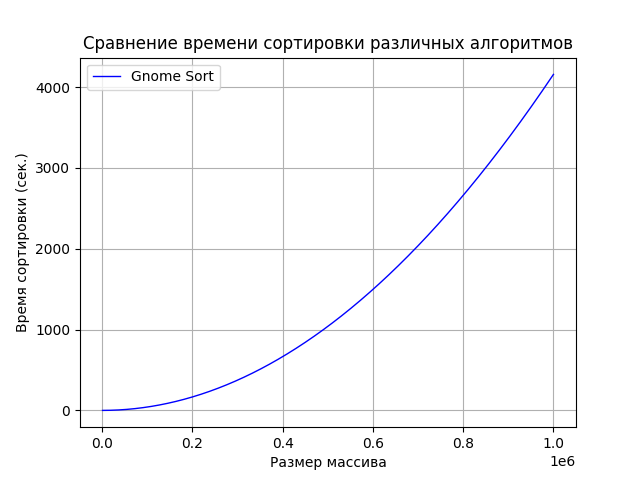
подается от 1000 до 1000000 элементов. Теоретически заданная сложность

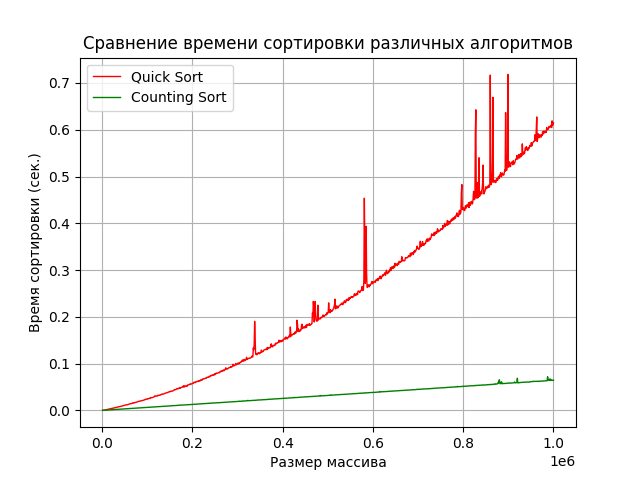
задач составляет O(N^2), O(N^logN) и O(N). Для тестирования алгоритма была

собрана статистика и построены следующие графики: график работы

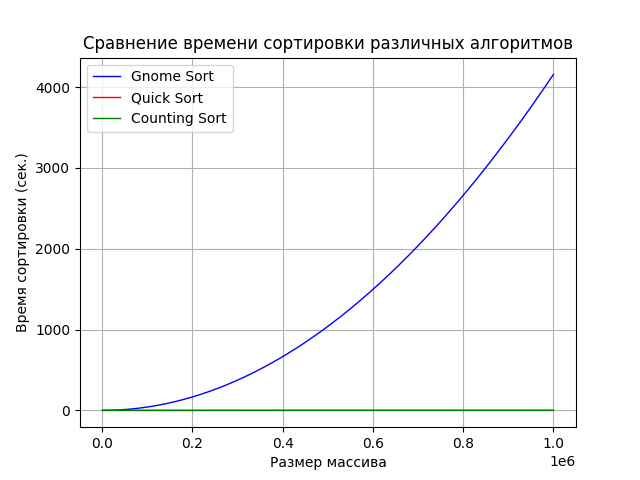
алгоритма для гномьей сортировки (изображение №4), график

работы алгоритма быстрой сортировки и сортировки с подсчётом (изображение №5). Так как время выполнения первого алгоритма и двух других сильно разнится, не представляется возможным представить их на одном графике(алгоритмы с лучшим временем «сливаются» в один) (изображение №6).

Изображение №4 — линейный график для гномьей сортировки



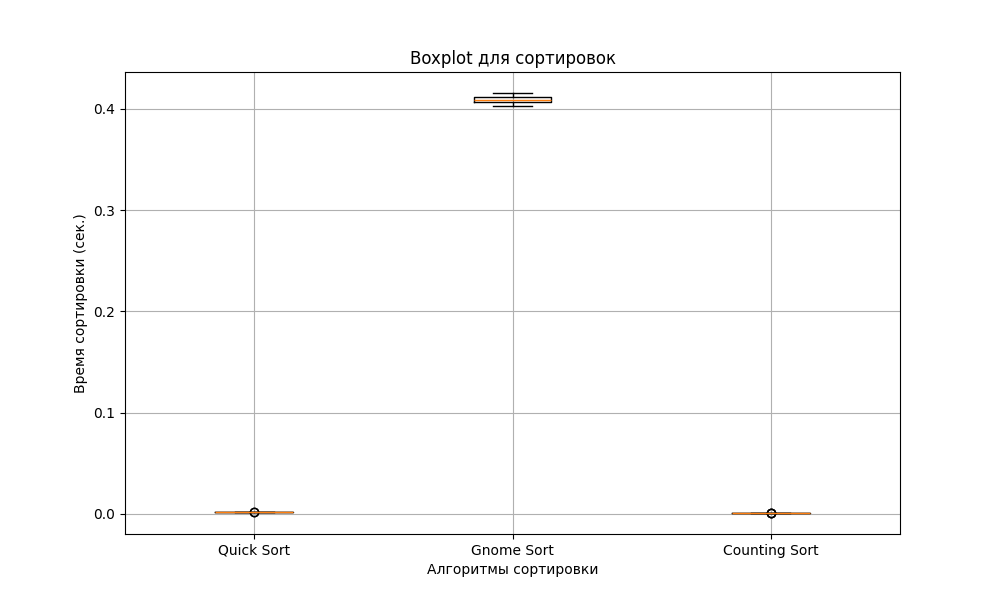
Изображение №5 — линейный график для быстрой и сортировки с подсчётом

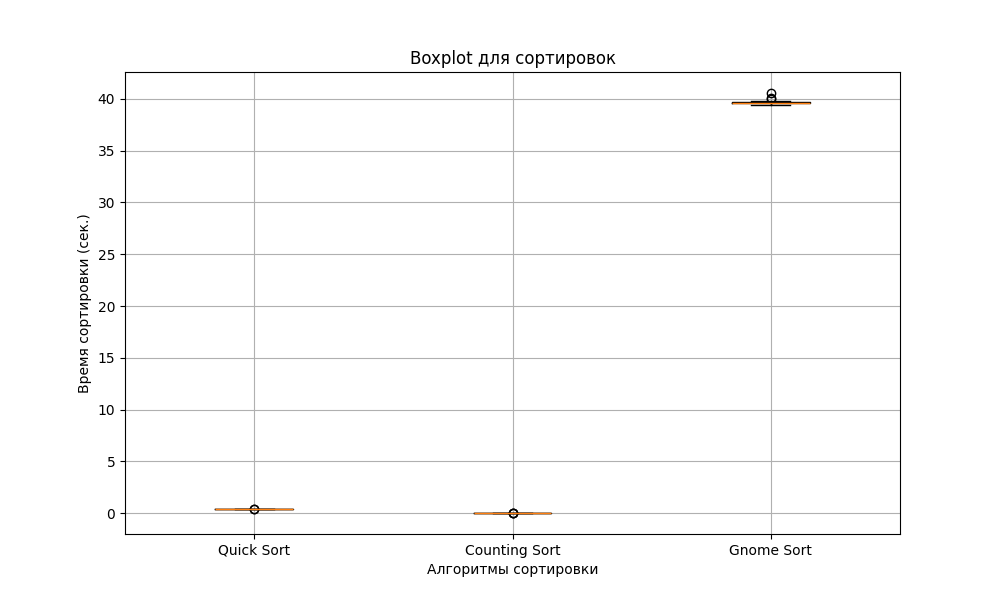
Изображение №6 — все три сортировки на линейном графике

Также были построены boxplot для анализа работы алгоритмов на

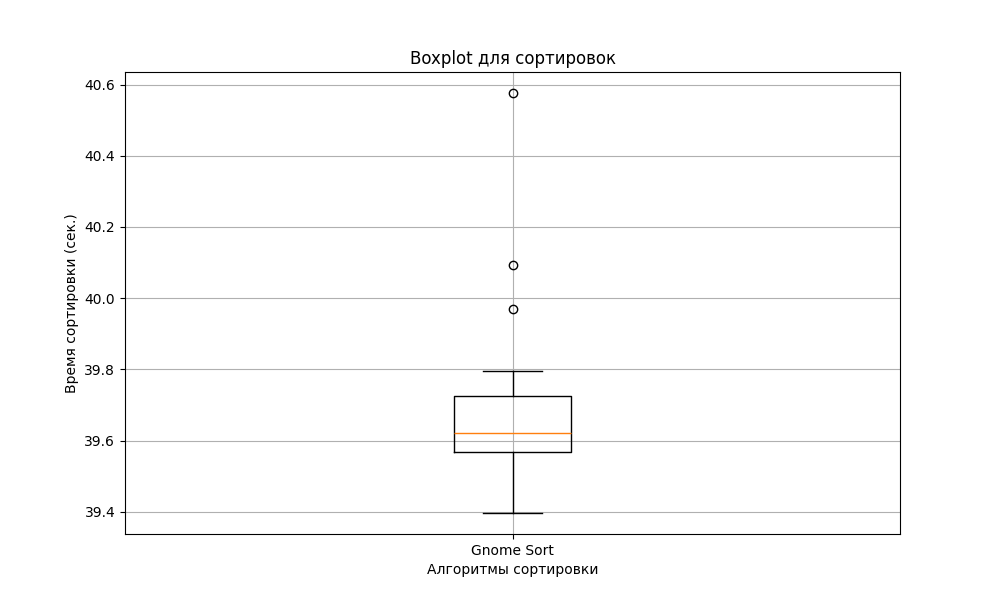
разных массивов одной длины, результаты приведены на изображении

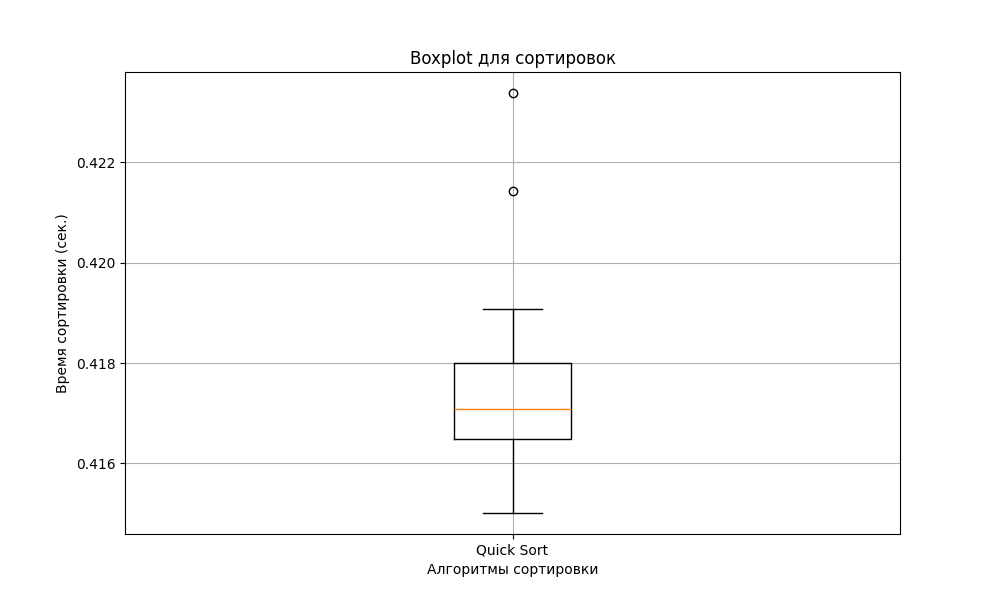
изображение №7(для 10e4 элементов) и изображение №8(для 10e5 элементов).

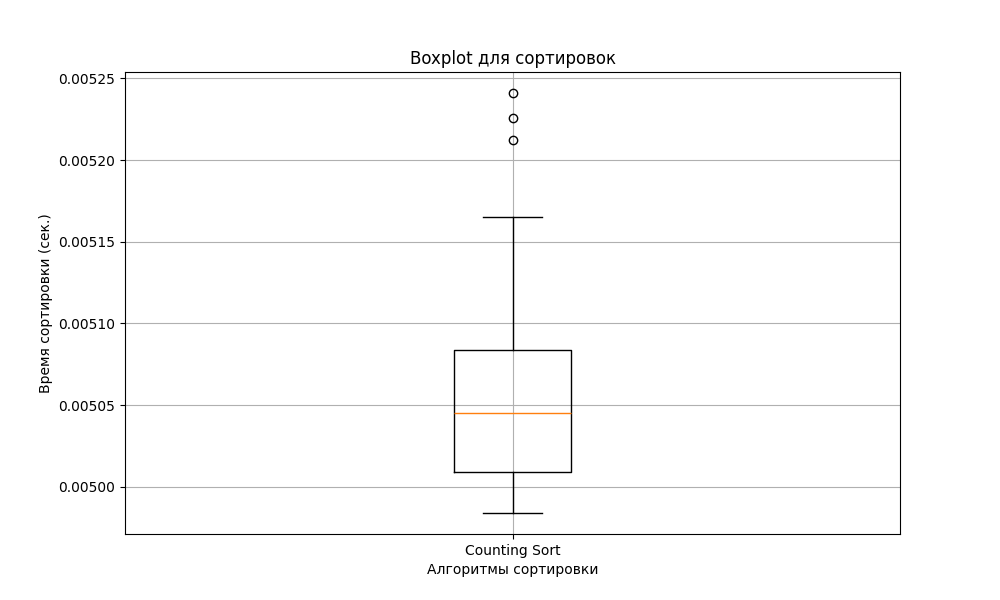
Изображение №7 — boxplot для сортировки 10e4 элементов

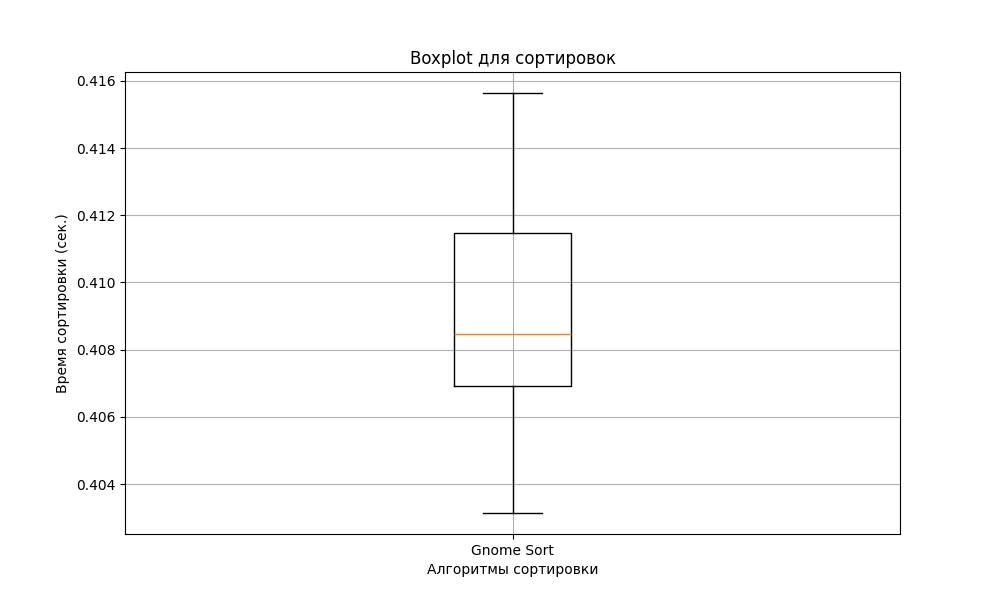
Изображение №8 — boxplot для сортировки 10е5 элементов

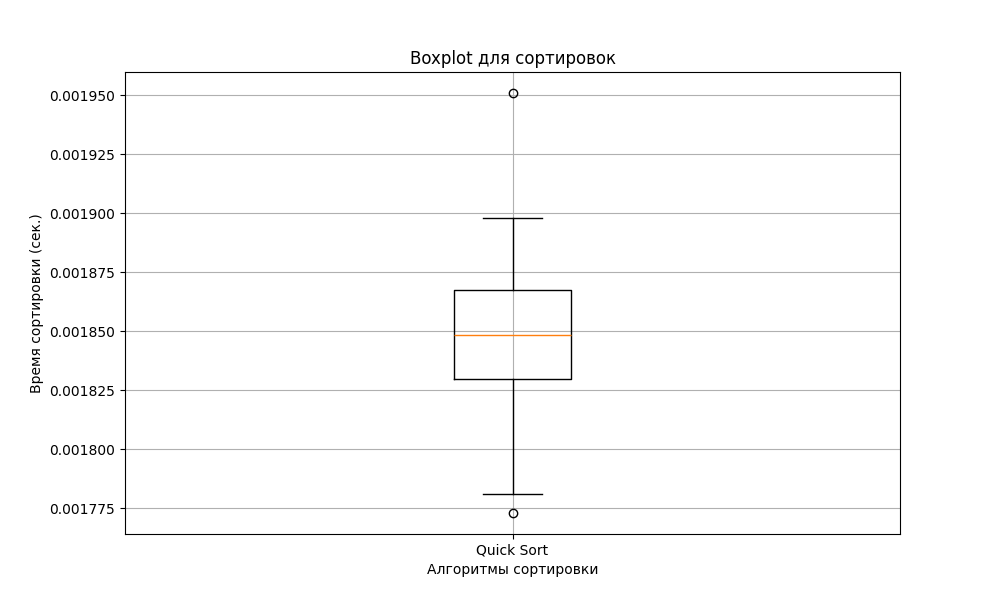
Также на основе данных составлены boxplot для каждого алгоритма сортировки отдельно для наглядности. На изображениях №9-11 представлены для 10е4 элементов, а на изображениях №12-14 для 10е5.

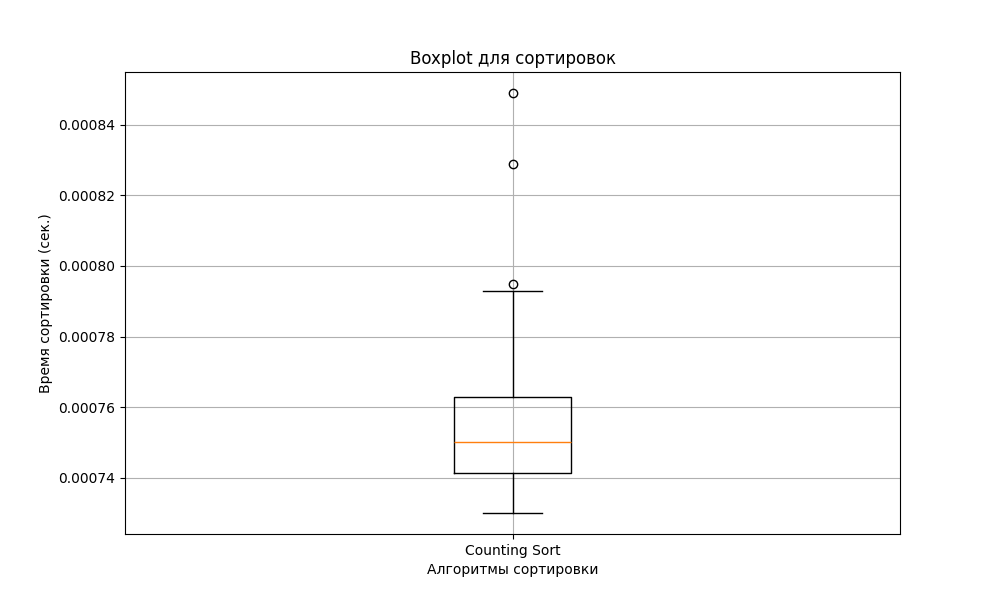
Изображение №9

Изображение №10

Изображение №11

Изображение №12

Изображение №13

Изображение №14

При анализе графиков с усами видно, что выбросы незначительны (они могут быть вызваны загруженностью процессора другими программами в тот или иной момент времени). Первый алгоритм работает гораздо медленне чем второй и третий, а третий быстрее второго.

Заключение

В ходе выполнения работы мною были реализованы разные алгоритмы

сортировки. Цель работы была достигнута путем анализа графиков и оценки асимптотики. Полученные результаты частично совпадают с теоретическими оценками сложности алгоритма. Асимптотика 1-го алгоритма выше 2-го, а 2-ая выше 3-его. Также можно сделать вывод, что первый алгоритм хорошо использовать для небольшого количества данных, его выполнение не требует большого количества памяти, второй алгоритм подходит для массивов с большим числом элементов, но имеет небольшие выбросы и требует затрат памяти, а третий алгоритм способен также обрабатывать большие массивы за короткий промежуток времени (лучше 2-го) и требует затрат памяти, но асимптотика очень зависит от диапазона значений массива.

Приложение А

Листинг трёх алгоритмов сортировки

