SФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Название лабораторной работы (Сортировка)»

Выполнил работу

Кек Герман Вадимович

Академическая группа №J3112

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

1. Введение
   1. Цель работы
      1. Цель лабораторной работы — изучение алгоритмов сортировки, их реализации и анализа, а также закрепление знаний, полученных на лекциях.
   2. Задачи работы
      1. Ознакомиться с теоретическими основами алгоритмов сортировки, изученных на лекции.
      2. Найти и реализовать алгоритмы сортировки.
      3. Запустить и протестировать написанные алгоритмы.
      4. Оформить отчет.
2. Подсчет по памяти
   1. Comb Sort:

int n – 4 байта

int gap – 4 байта

bool flag – 1 байт

int i – 4 байта

* 1. Binary Insertion Sort:

Функция binSearch:

int item – 4 байта.

int right – 4 байта.

int left – 4 байта.

int mid – 4 байта.

Функция:

int size – **4 байта**.

int i – **4 байта.**

int key – **4 байта**.

int swap\_pos – **4 байта.**

int j – **4 байта**.

* 1. Quick Sort:

Функция sortPart:

int left – **4 байта**.

int right – 4 байта.

int sort\_el – 4 байта.

Функция quickSort:

int int\_sort\_el – 4 байта.

1. Подсет асимптотики
   1. Comb Sort:

Лучший случай: O(n log(n))

Худший случай: O(n^2)

* 1. Binary Insertion Sort:

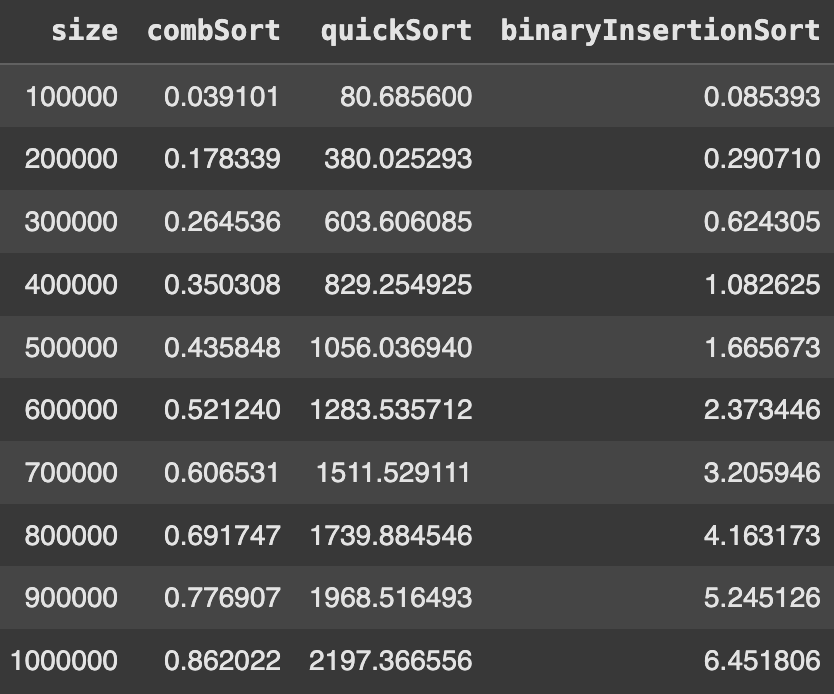
Лучший случай: O(n log(n))

Худший случай: O(n^2)

* 1. Quick Sort:

Лучший случай: O(n log(n))

Худший случай: O(n^2)

1. Тесты  
   

Изображение 1 – время выполнение тестов

1. Линейные графики работы алгоритма

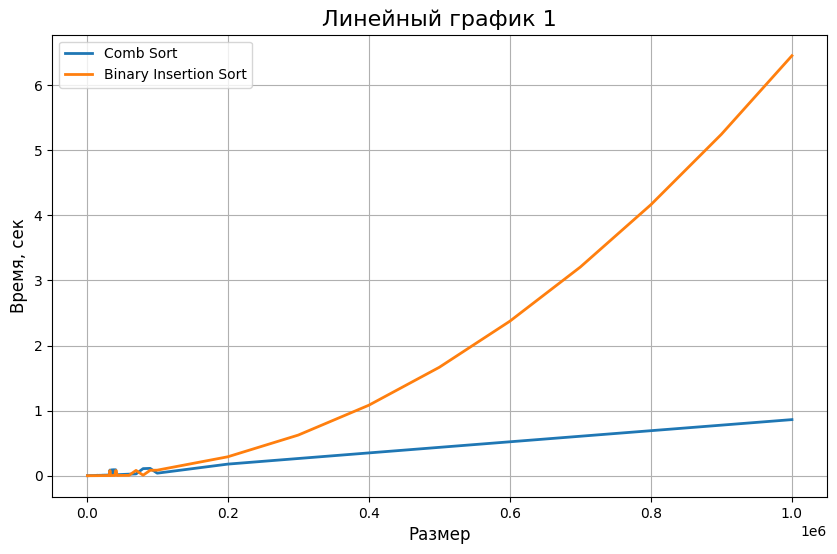


График 1 – работа алгоритмов Comb Sort и Binary Insertion Sort

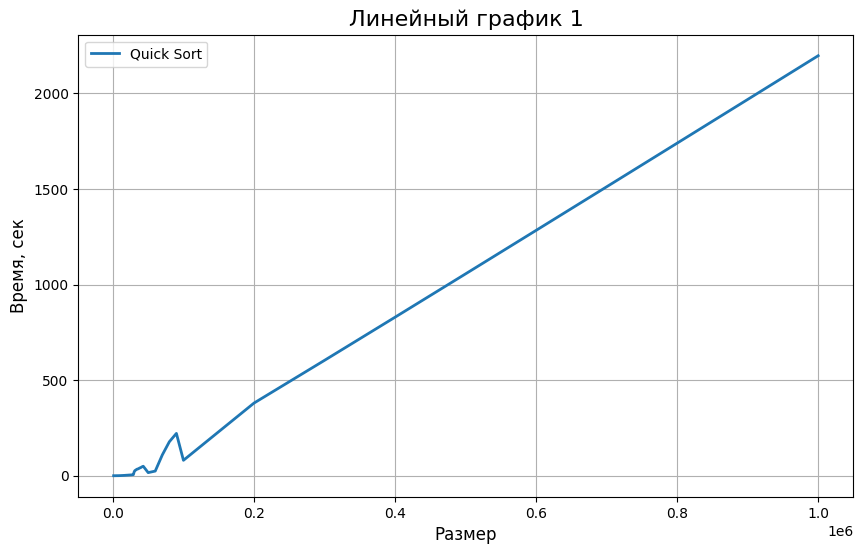


График 2 – работа алгоритма Quick Sort

1. Графики для времени работы алгоритмов с числом элементов 1e4 и 1e5

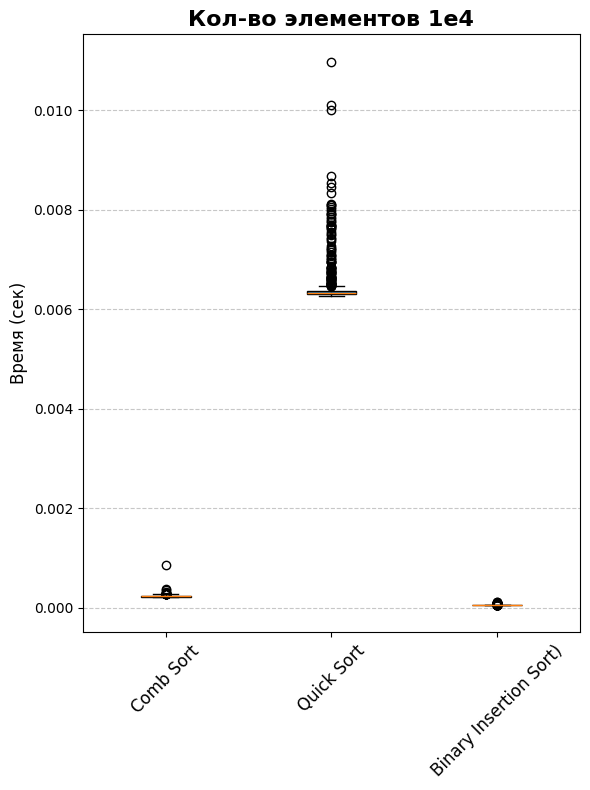


График 3 – кол-во элементов равно 1e4

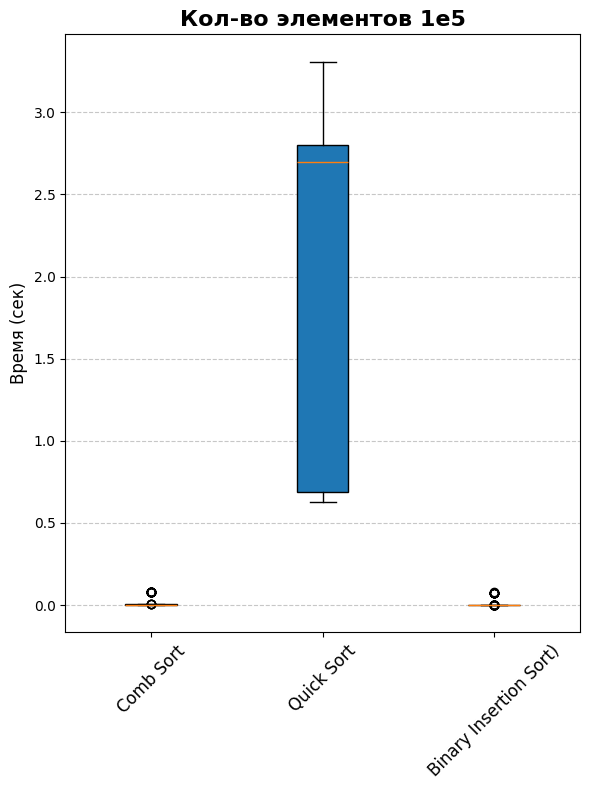
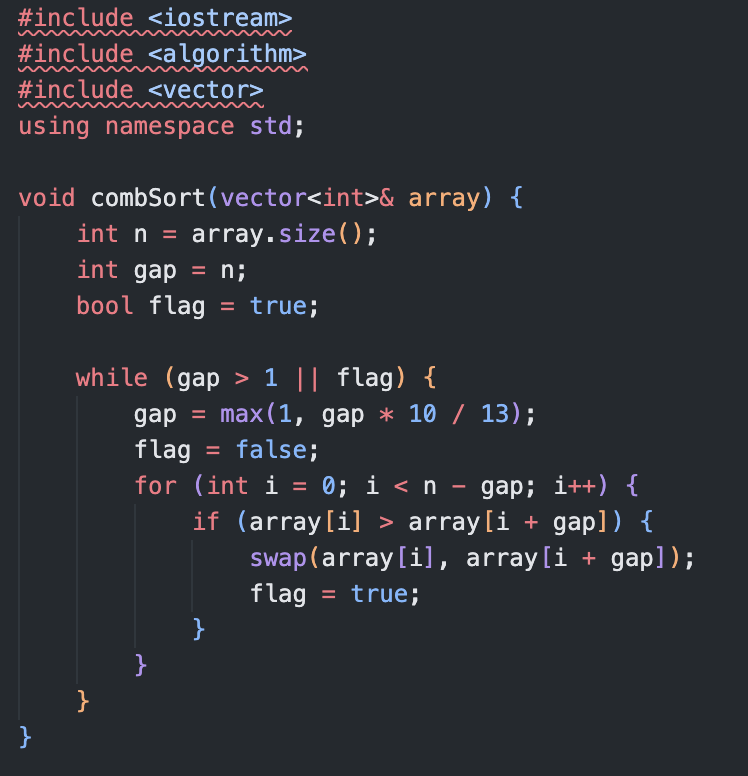


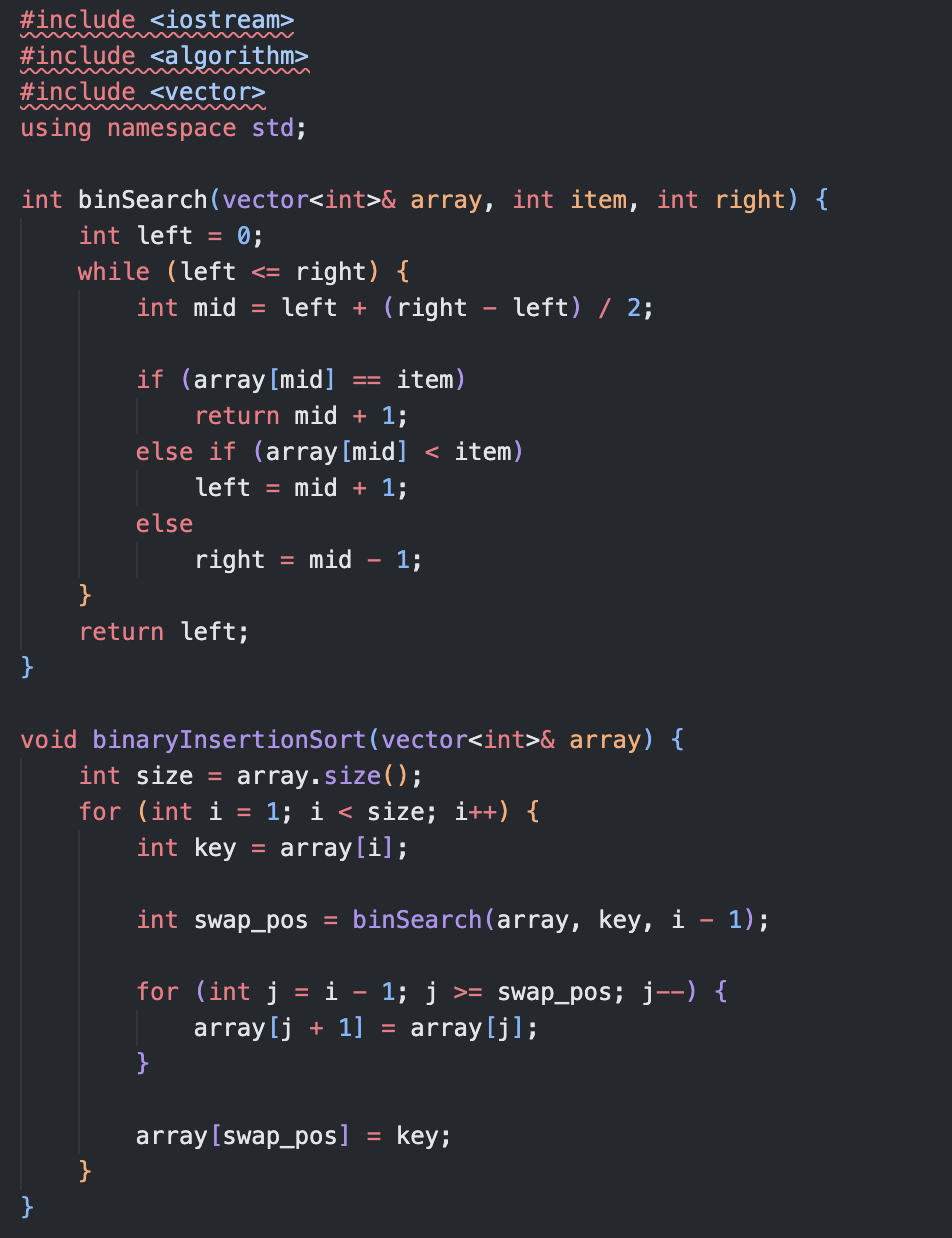
График 4 – кол-во элементов равно 1e5

1. **Вывод:**
2. **Практическая производительность**:
   1. **Quick Sort** демонстрирует худшие результаты по времени выполнения, особенно на больших массивах. Это подтверждает его уязвимость к худшему случаю O(n^2).
   2. **Comb Sort** оказался стабильнее на разных размерах массива, что делает его более предсказуемым.
   3. **Binary Insertion Sort** показывает минимальное время для малых массивов, однако на больших размерах его рост времени выполнения становится заметным.
3. **Причины выбросов в результатах:**
   1. **Quick Sort**:
      1. Основная причина выбросов — худший случай выполнения. Если массив уже отсортирован или почти отсортирован, а опорный элемент выбран неудачно (например, всегда первый или последний элемент), это приводит к разбалансированным разделениям и увеличению глубины рекурсии, что резко увеличивает время работы.
      2. Также возможные выбросы могут быть связаны с большим использованием памяти из-за рекурсивных вызовов, особенно при отсутствии хвостовой рекурсии.
   2. **Comb Sort**:
      1. Выбросы могут быть вызваны неэффективным выбором промежутков (gap), особенно в случаях, когда промежутки становятся слишком малыми, что замедляет алгоритм.
   3. **Binary Insertion Sort**:
      1. Этот алгоритм демонстрирует выбросы на больших массивах из-за увеличения количества операций вставки. Если массив отсортирован в обратном порядке, каждая вставка требует сдвига всех предыдущих элементов, что приводит к наибольшему времени выполнения.
4. **Системные факторы**:
   1. Непредсказуемость выполнения задач на процессоре, вызванная фоновыми процессами, может вызывать выбросы.
5. **Рекомендации по применению**:
   1. **Quick Sort** следует использовать только с улучшениями для обработки худших случаев (например, случайный выбор опорного элемента).
   2. **Comb Sort** предпочтителен для массивов среднего размера.
   3. **Binary Insertion Sort** отлично подходит для небольших массивов, особенно если они частично отсортированы.
6. Приложения   
   

Код 1 – Comb Sort



Код 2 – Quick Sort



Код 3 – Binary Insertion Sort