ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Сортировки shakerSort TimSort bucketSort»

Выполнил работу

Урядов Валерий

Академическая группа №J3111

Принято

Санкт-Петербург

2024

1. Введение

Цель данной работы заключается в изучении и реализации алгоритмов сортировки для анализа их производительности и характеристик.

Задачи, которые необходимо решить в рамках работы:

1. Реализовать алгоритмы сортировки по заданным критериям.
2. Разработать и провести тестирование реализованных алгоритмов на различных наборах данных для анализа корректности их работы.
3. Сравнить производительность и применимость алгоритмов в зависимости от типа и объема входных данных.
4. Подготовить выводы на основе полученных результатов, отражающие преимущества и недостатки каждого из рассмотренных алгоритмов.

Данная работа позволяет углубить понимание принципов работы алгоритмов сортировки, а также их эффективности в различных условиях.

1. Теоретическая подготовка

В данной работе используются три алгоритма сортировки: Bucket Sort, Tim Sort и Shaker Sort.

#### Bucket Sort

Идея: Этот алгоритм использует принцип распределения элементов массива по "корзинам" (buckets) в зависимости от их значений. После распределения каждая корзина сортируется отдельно, и элементы объединяются в упорядоченный массив.

Сложность:

* Лучшая: O(n+k), где k — число корзин.
* Худшая: O(k\*n) при неравномерном распределении элементов.
* Средняя: O(n\*log(n)).

#### Tim Sort

Идея: Tim Sort — это гибридный алгоритм, объединяющий идеи Merge Sort и Insertion Sort. Он оптимизирован для работы с реальными наборами данных, часто содержащими частично отсортированные последовательности.

* Сложность:
  + Лучшая: O(n), для почти отсортированных массивов.
  + Худшая: O(n\*log(⁡n)), для случайных данных.
  + Средняя: O(n\*log(⁡n))

#### Shaker Sort

Идея: Этот алгоритм является модификацией Bubble Sort, где проходы по массиву происходят в обоих направлениях (сперва слева направо, затем справа налево).

* Сложность:
  + Лучшая: O(n), для уже отсортированных данных.
  + Худшая: O(n^2), для обратного порядка элементов.
  + Средняя: O(n^2).

1. Реализация

Было реализовано 3 алгоритма сортировки, немного изучив какие есть разнообразные способы сортировки были выбраны Shakersort, Tim Sort, Bucket Sort, потому что они подходили под условие задачи. Идеи этих сортировок приведены в пункте два.

Этапы алгоритма Bucket Sort:

1.Разделение массива на диапазоны значений (корзины).

2.Распределение элементов массива по корзинам.

3.Сортировка каждой корзины (с использованием встроенного метода сортировки).

4.Объединение отсортированных корзин в итоговый массив.

Этапы алгоритма Tim Sort:

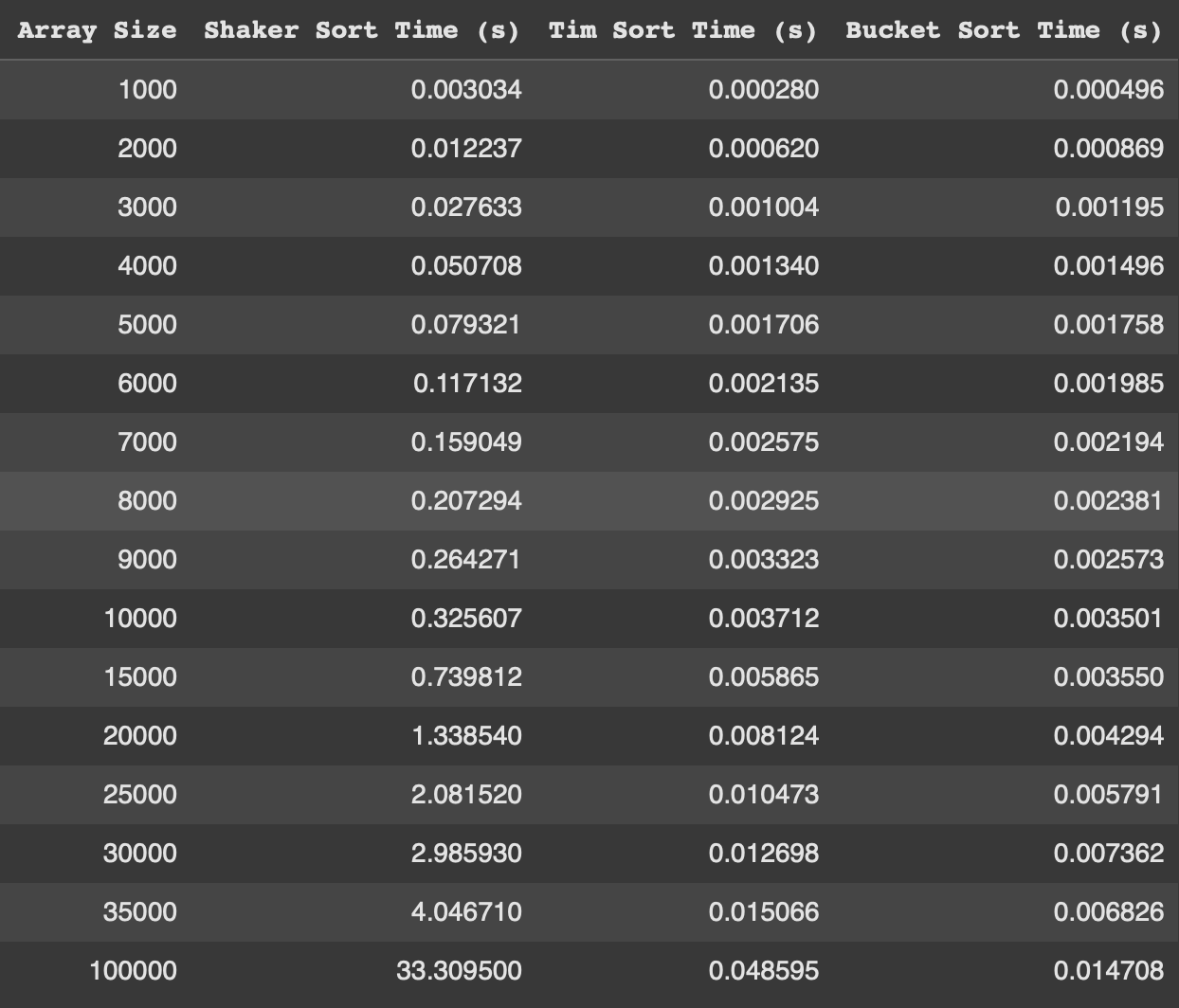
1. Разделение массива на небольшие подмассивы (runs).
2. Использование Insertion Sort для сортировки runs.
3. Слияние отсортированных runs с помощью Merge Sort.

Этапы алгоритма Shakersort:

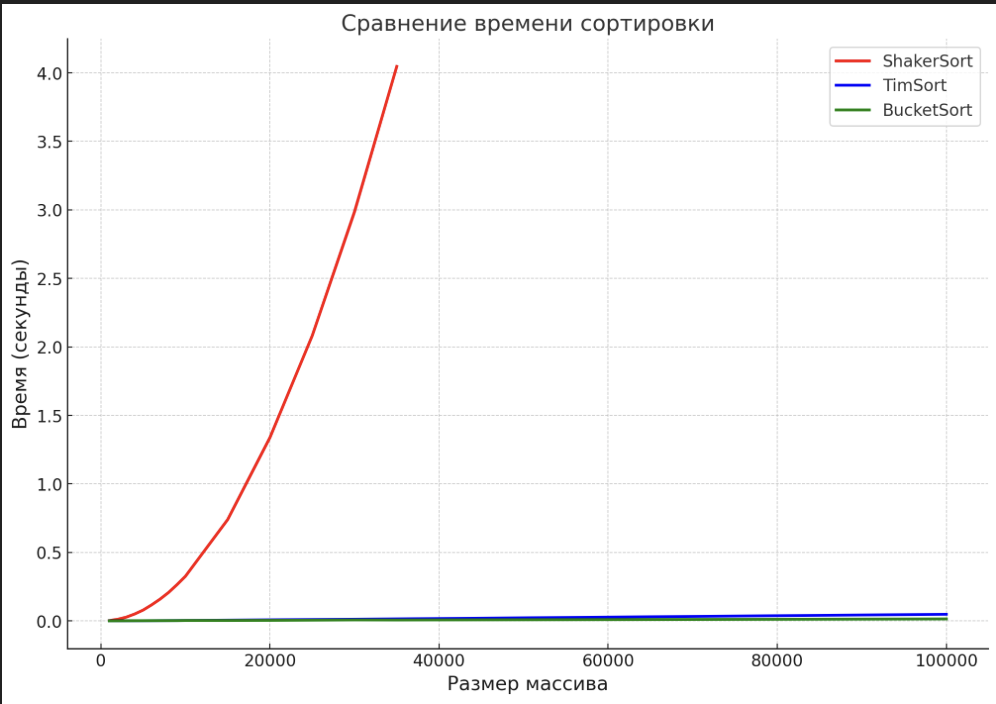
1. Проход слева направо, перемещение наибольшего элемента к концу массива.
2. Проход справа налево, перемещение наименьшего элемента к началу массива.
3. Повторение шагов до завершения сортировки.

Были использованы стандартные библиотеки <vector>, <algorithm>, <cassert>, <fstream>.

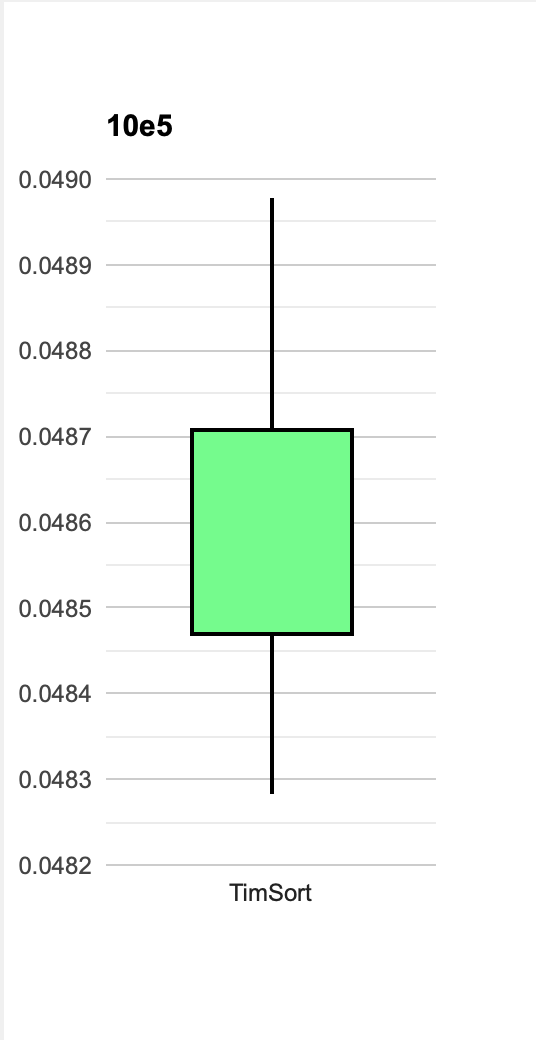
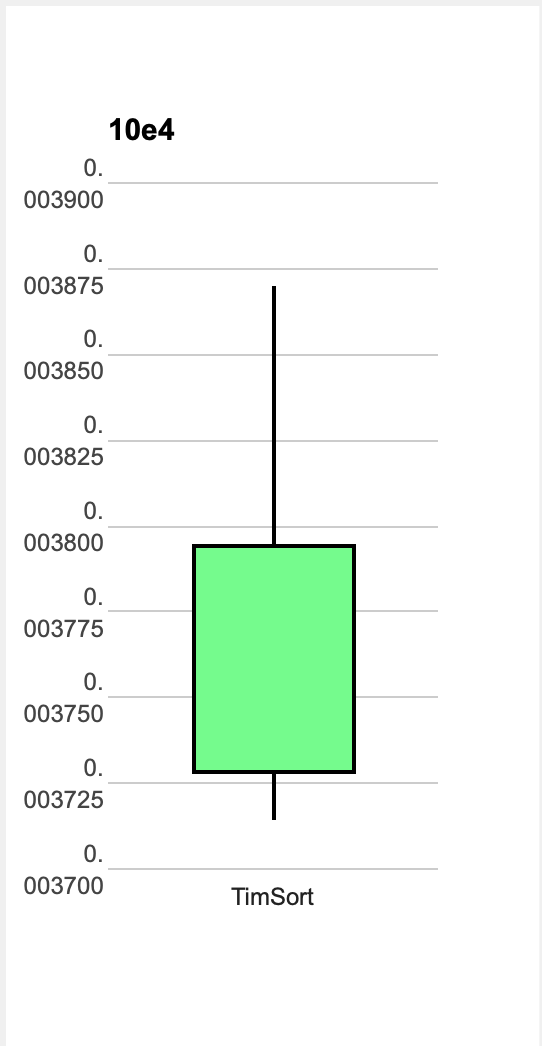
1. Экспериментальная часть

Таблица №1 -Измерение скорости работы трех алгоритмовГрафик представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №1.

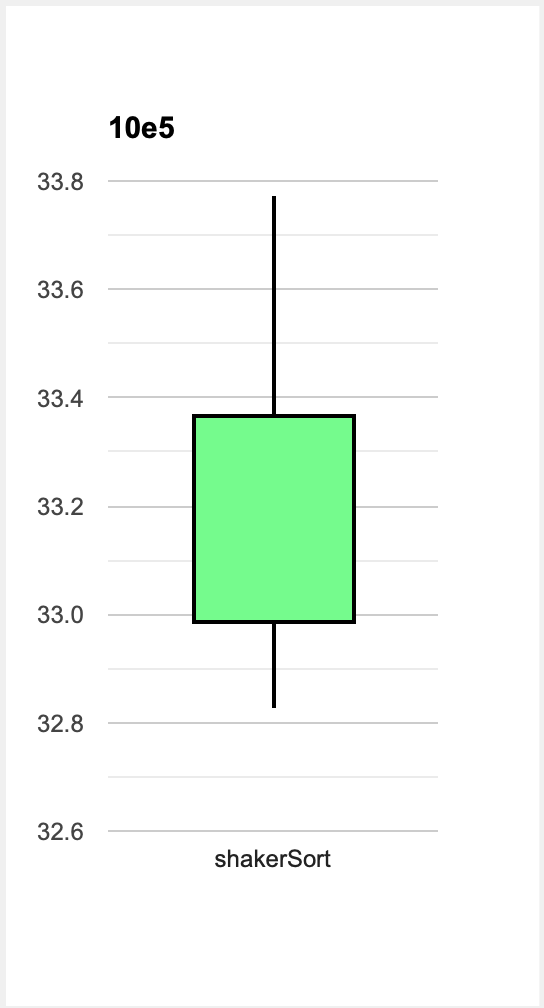
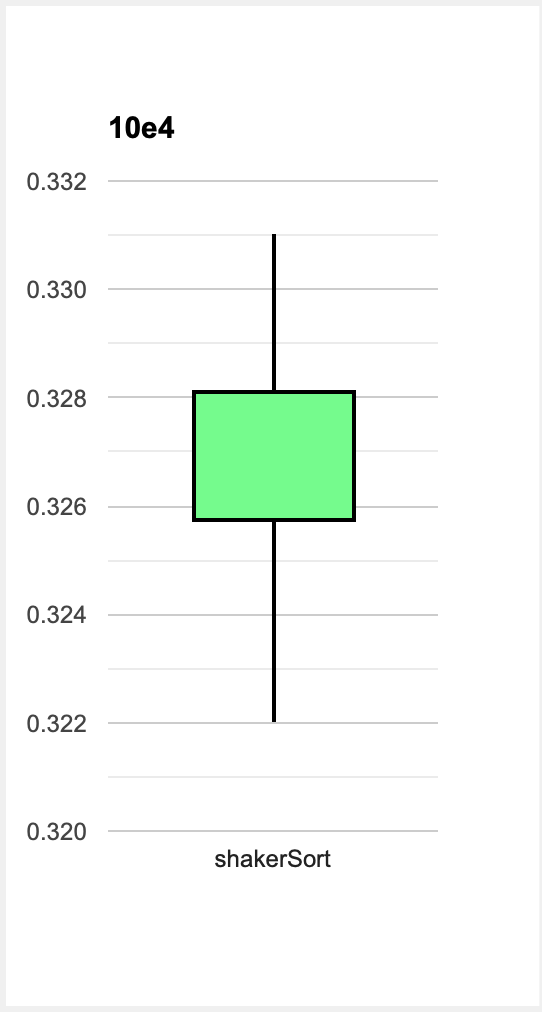
Изображение №1 - График работы алгоритма



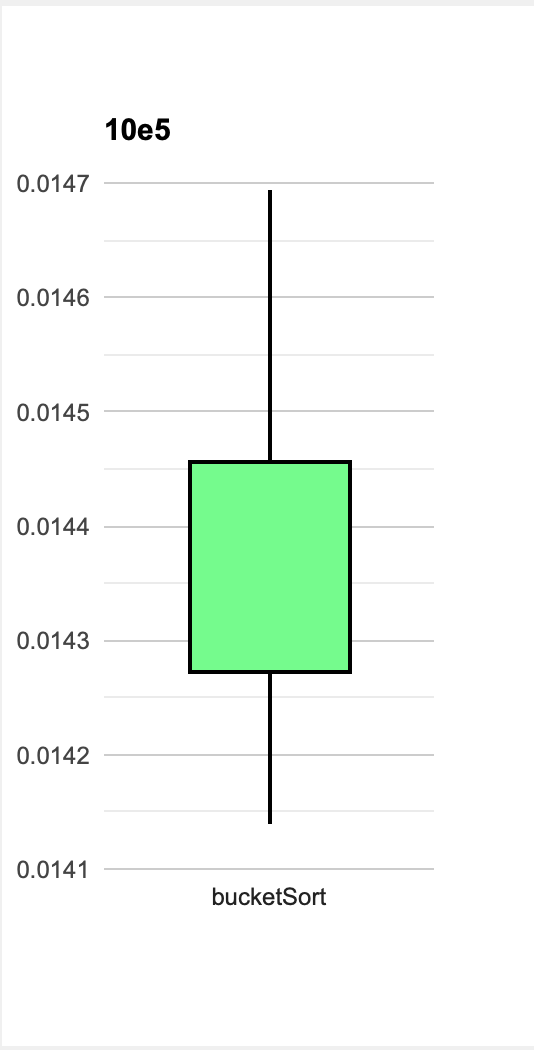
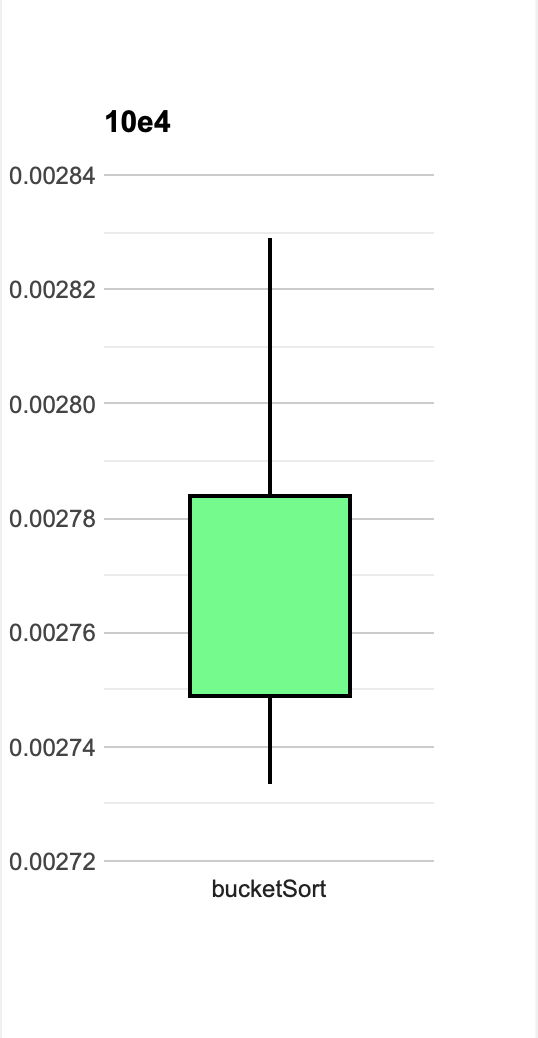
Изображение №2 - График работы алгоритма TimSort



Изображение №3 - График работы алгоритма ShakerSort



Изображение №4 - График работы алгоритма Bucket Sort



### Анализ графиков

1. Shaker Sort:
   * Время выполнения значительно увеличивается с ростом размера массива.
   * Этот алгоритм демонстрирует квадратичную зависимость времени выполнения от размера массива (O(n2)O(n2)).
   * Использование Shaker Sort для больших массивов (например, 105105) является неэффективным.
2. TimSort:
   * Алгоритм показывает практически линейное увеличение времени выполнения с ростом размера массива (O(nlog⁡n)O(nlogn)).
   * Tim Sort стабильно демонстрирует минимальное время выполнения по сравнению с другими алгоритмами.
   * Это объясняется гибридным подходом, сочетающим преимущества Merge Sort и Insertion Sort.
3. Bucket Sort:
   * Время выполнения практически линейное и сопоставимо с TimSort для определённых размеров массива.
   * Алгоритм чувствителен к распределению данных. В данном случае, данные, скорее всего, равномерно распределены, что позволило Bucket Sort эффективно работать.
   * Однако в реальных сценариях, с неравномерным распределением данных, его производительность может ухудшиться.
4. Заключение

В ходе выполнения работы были реализованы и протестированы три алгоритма сортировки: TimSort, Bucket Sort и Shaker Sort. Цель работы, заключающаяся в изучении принципов работы и сравнении производительности этих алгоритмов, была успешно достигнута. Проведенные тесты подтвердили теоретические оценки сложности алгоритмов и продемонстрировали их поведение на массивах различного размера.

Полученные результаты показали, что:

* TimSort является наиболее универсальным и эффективным алгоритмом, подходящим для работы с массивами любых размеров.
* Bucket Sort демонстрирует высокую производительность при равномерном распределении данных и известном диапазоне значений.
* Shaker Sort, обладая высокой временной сложностью, подходит только для небольших массивов или частично отсортированных данных.

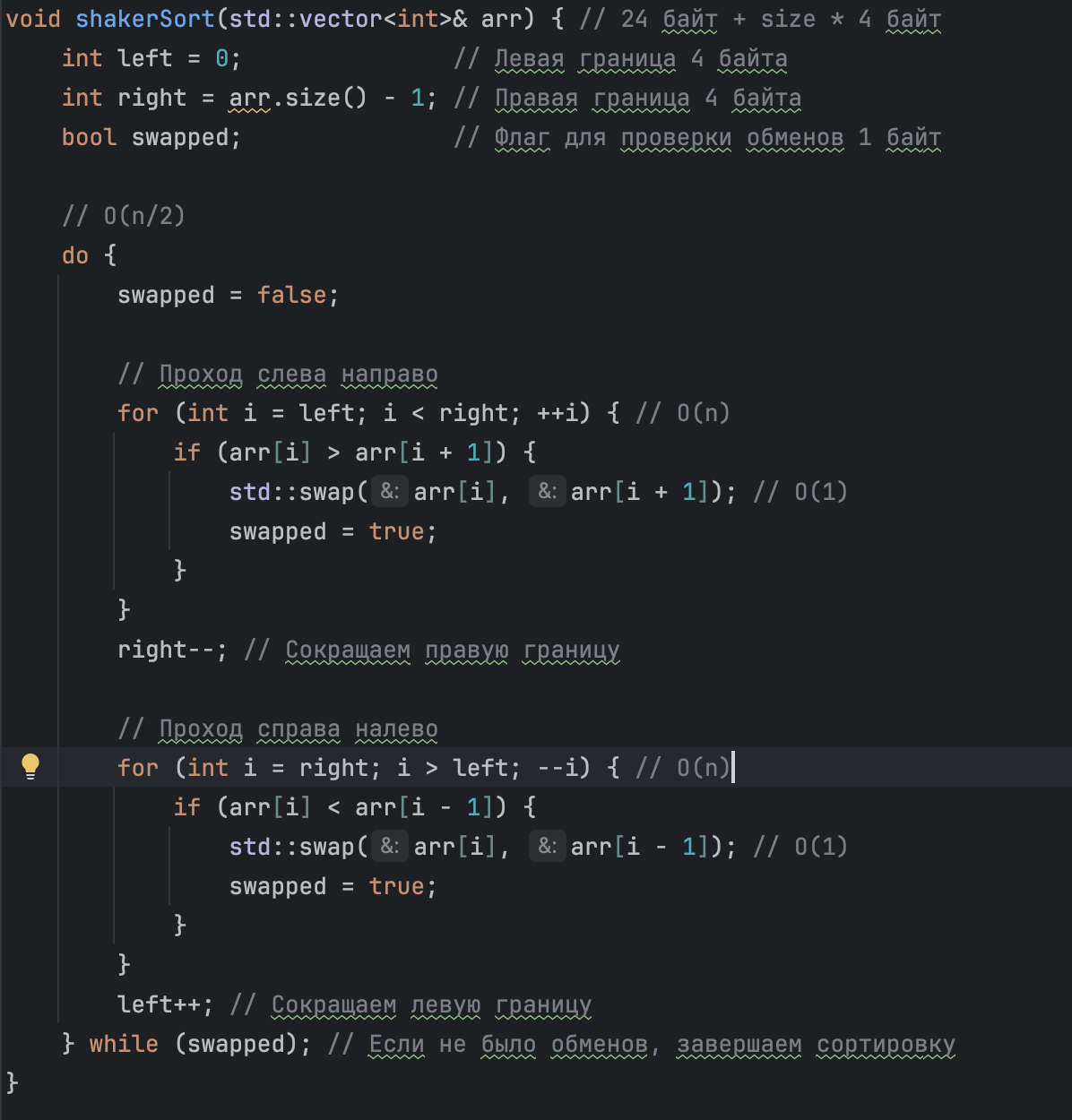
Возможные направления для дальнейшего исследования включают:

1. Оптимизацию реализованных алгоритмов для работы с данными различных типов (например, строками или объектами).
2. Изучение специализированных алгоритмов сортировки, таких как Radix Sort или Quick Sort, и их сравнение с рассмотренными методами.
3. Анализ производительности алгоритмов в условиях ограничений на использование оперативной памяти.

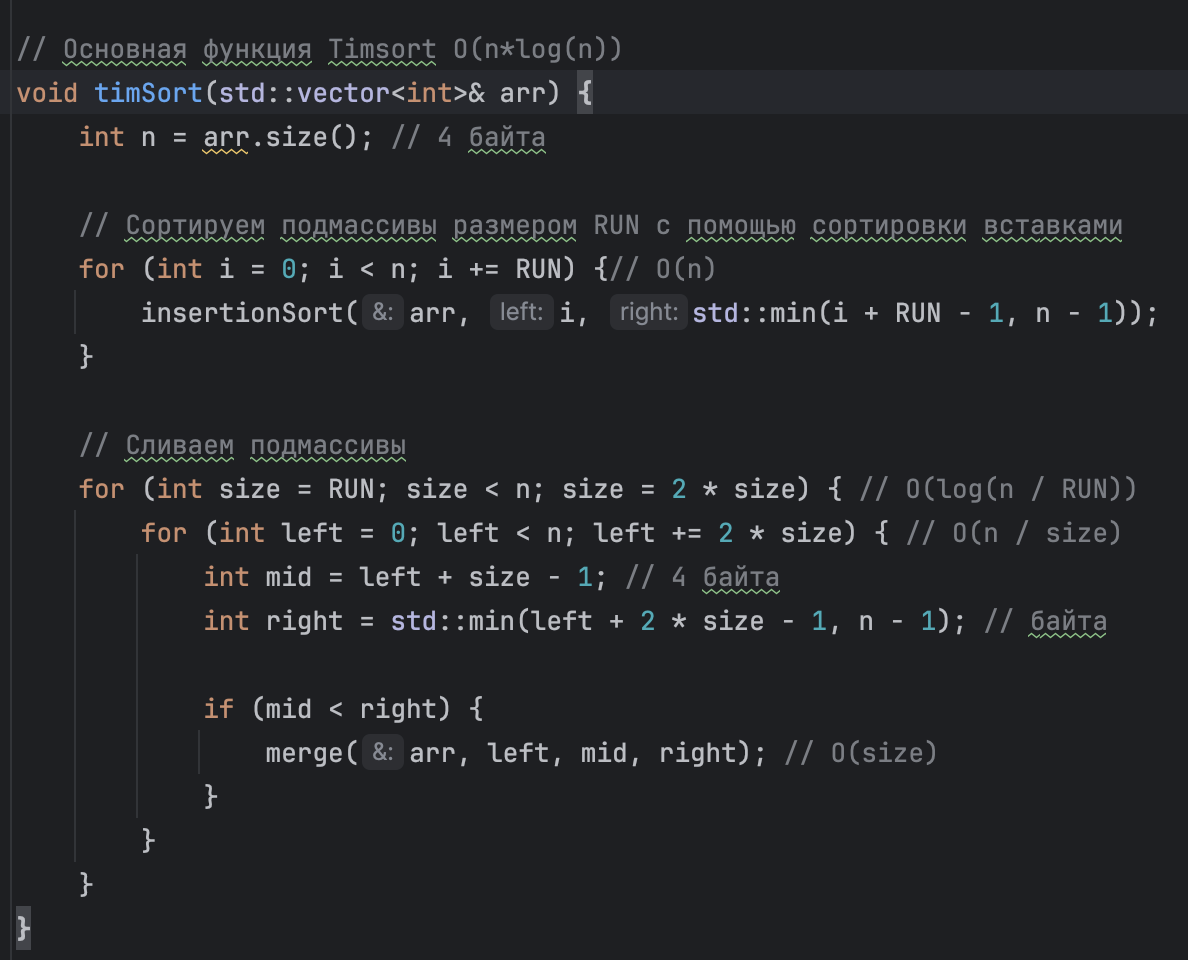
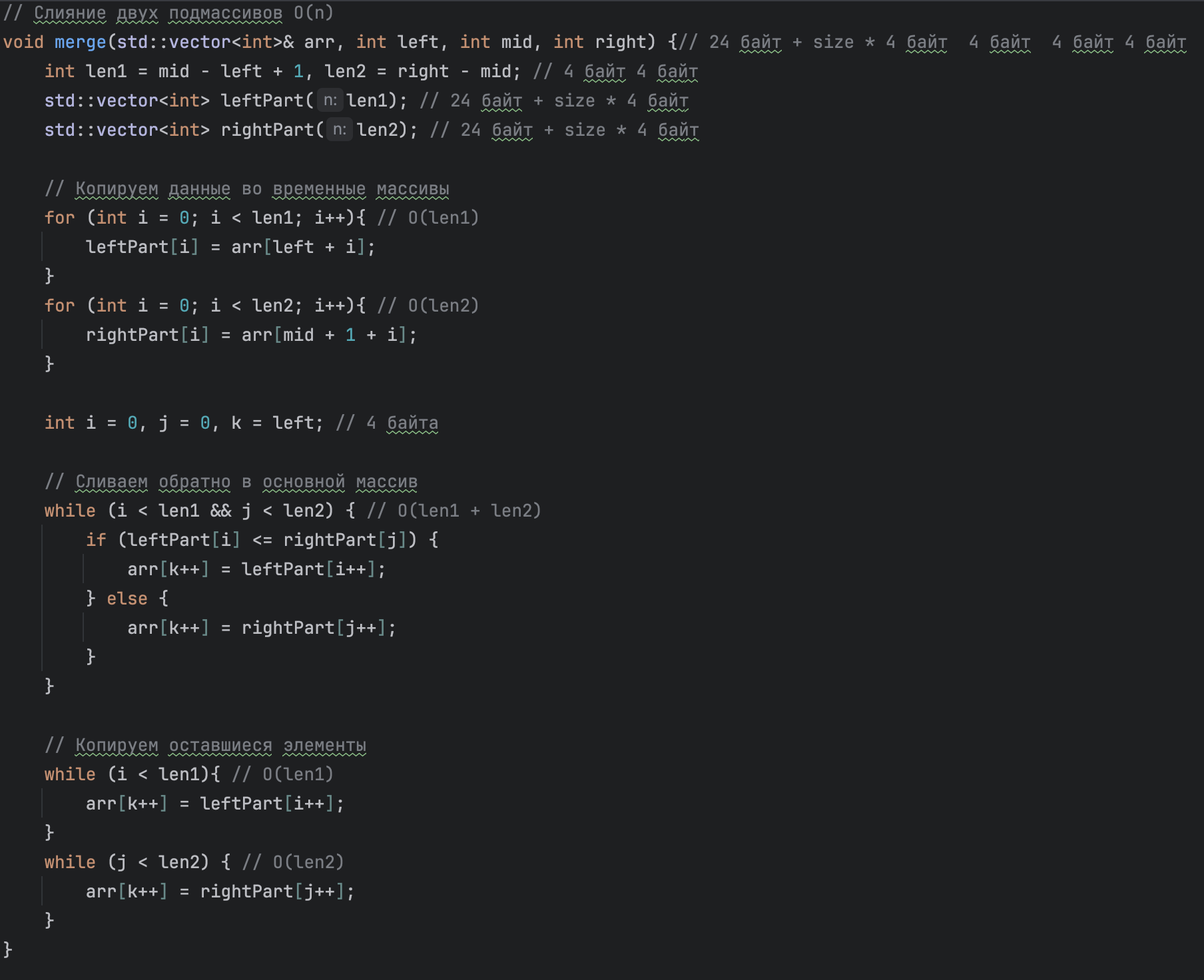
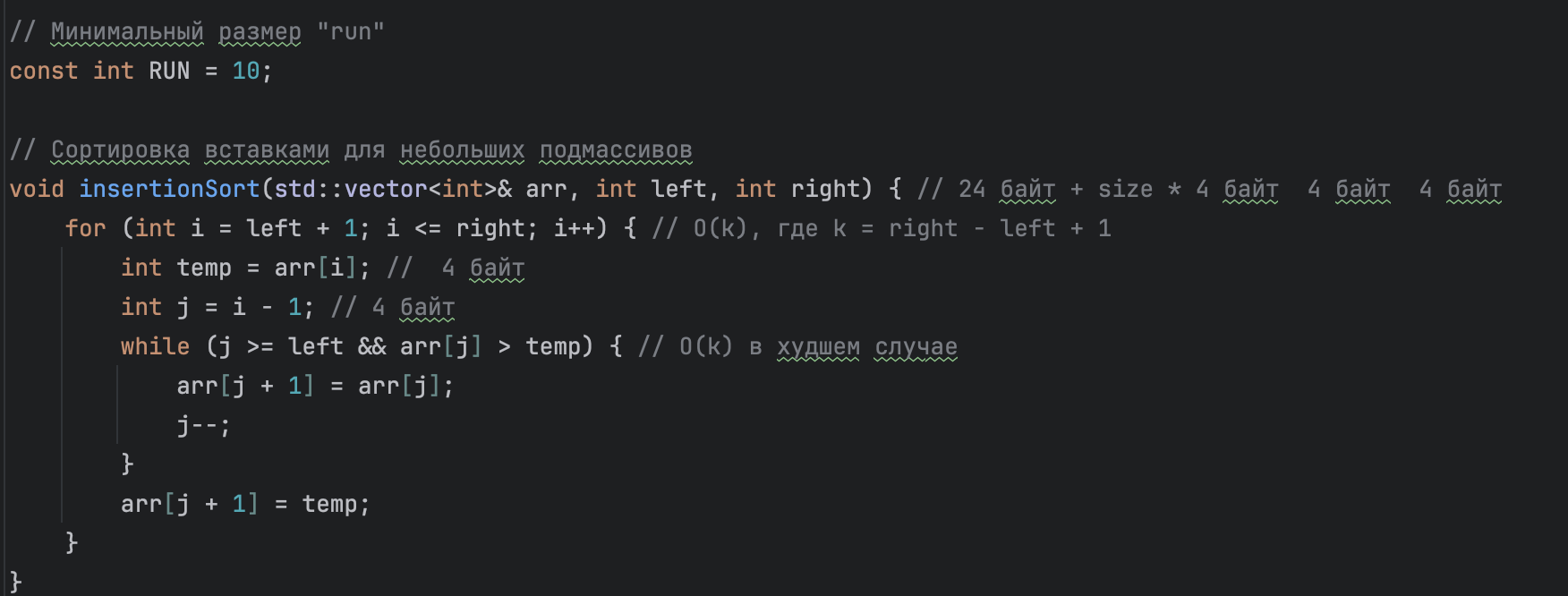
Выполненная работа предоставила ценную практическую основу для понимания различных подходов к сортировке данных и их применимости в реальных задачах.

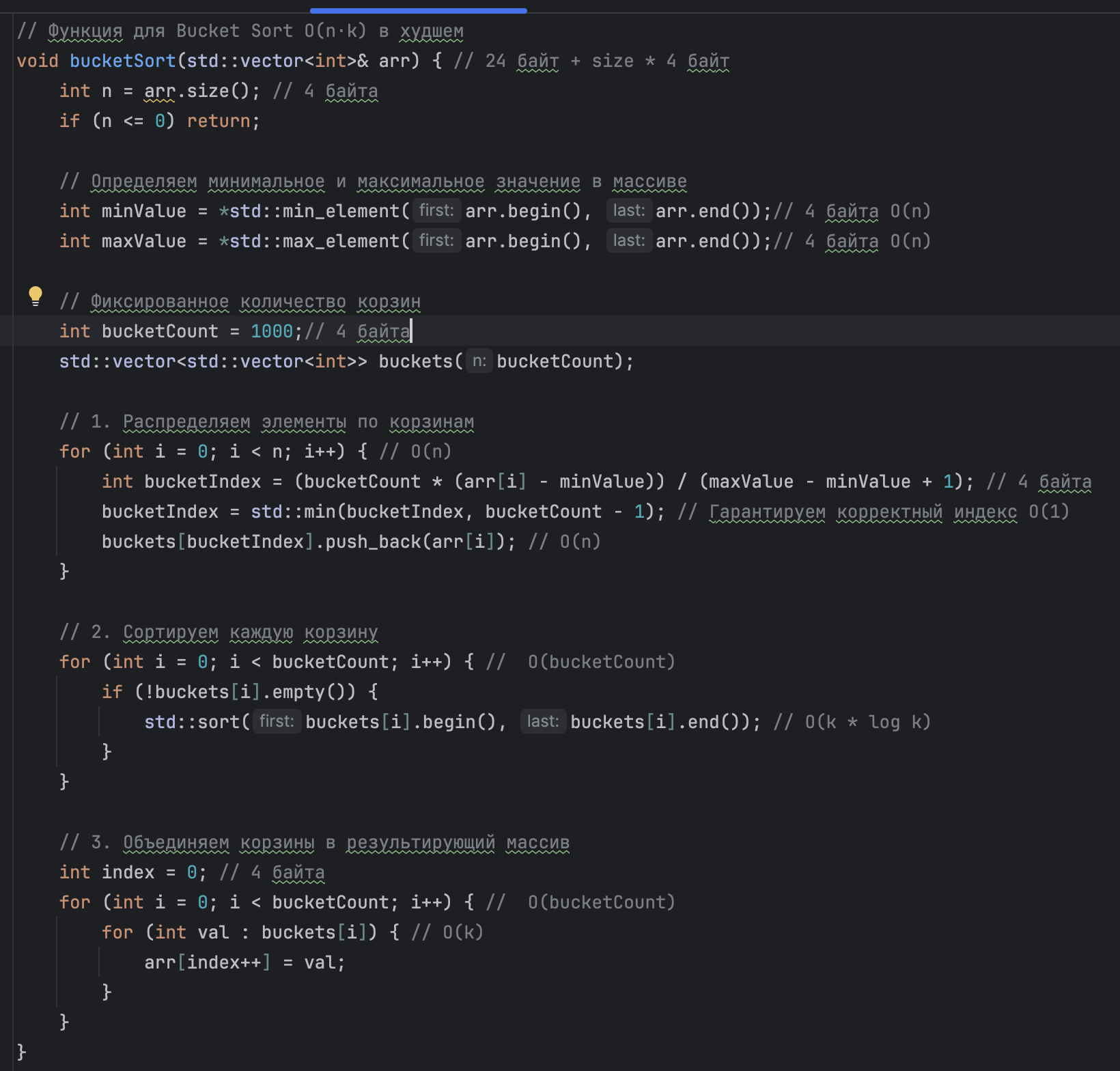
1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла shakerSort.cpp

Листинг кода файла timSort.cpp



Листинг кода файла bucketSort.cpp

Листинг кода чтения файла и проверки (на примере shakerSort)

