ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № V

«Алгоритмы сортировок Shaker Sort, Tree Sort и Merge Sort»

Выполнил работу

Никитин Данил Алексеевич

Академическая группа J3110

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

**Структура отчёта:**

* Введение

Цель: Познакомится и раобрать некоторые алгоритмы сортировок ,попробовать поработать в условтиях ьнеопределенного ТЗ.

Задачи:

1) Написать код алгоритмов сортировки.

2) Составить ход выполнения алгоритма.

3) Посчитать сложность по времени и памяти.

4) Составить ход выполнения алгоритма.

* Теоретическая подготовка

1) Использование контейнера std::vector для хранения данных.

2) Принципы работы с бинарным деревом поиска для реализации Tree Sort.

3) Понимание рекурсии для реализации алгоритмов Merge Sort и Tree Sort.

4) Знание сложности алгоритмов сортировки и их особенностей.

5) Умение анализировать стабильность алгоритмов сортировки.

6) Классы и структуры в С++ , конструктор классов.

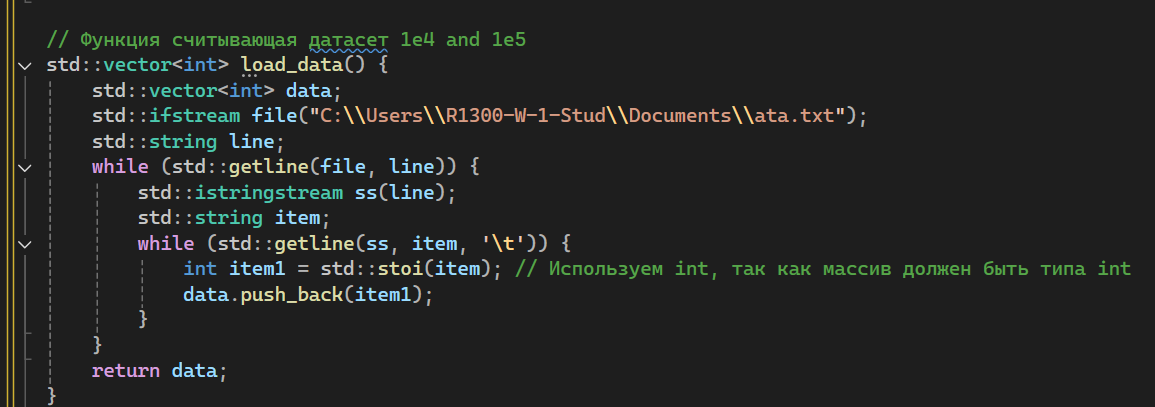
* **Получение таблиц для создания графиков.**

1) Создание файлов для графика 6 задания.

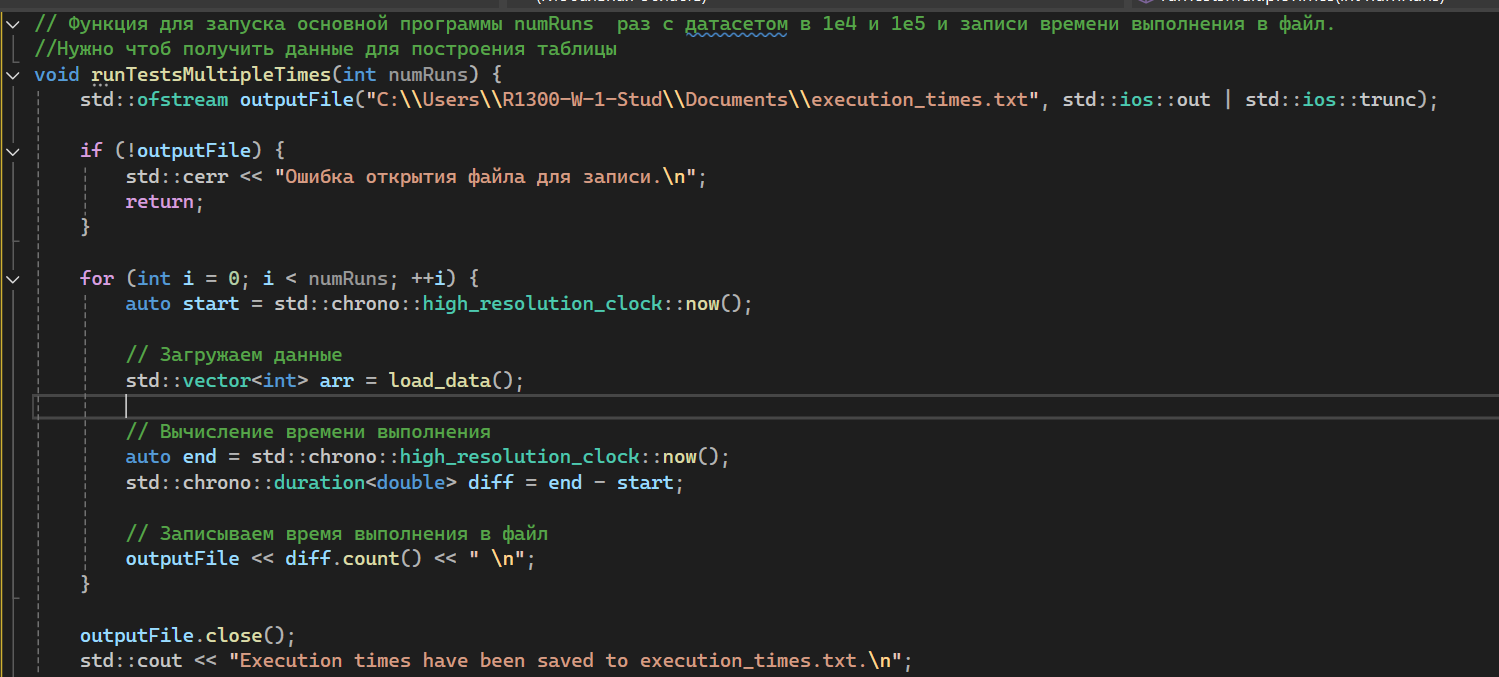
Я решил создавать датасеты размером 1е4 и 1е5 ,нужные для построения таблицы из задания номер 6 (box plot), с помощью данных нам в репозитории функций .

Считывать эти датасеты будем в два подхода так как у нас два файла.(1е4.txt или 1е5.txt)

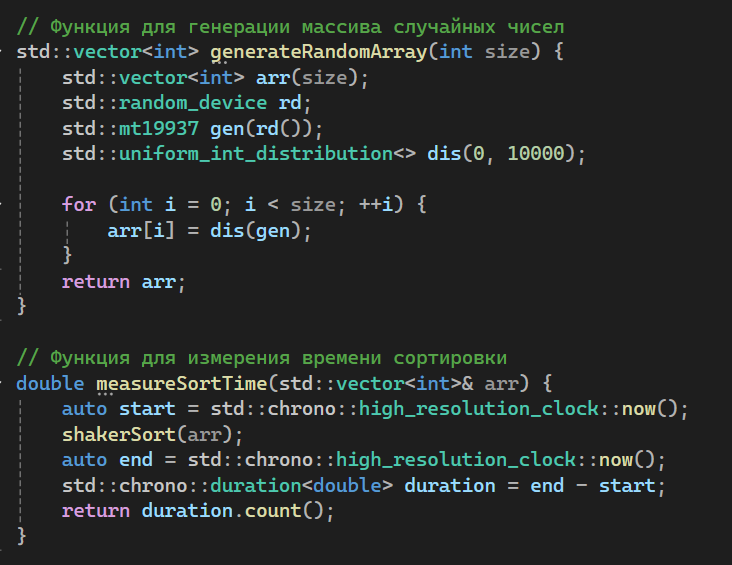
Считывать будем с помощью функции такого типа:

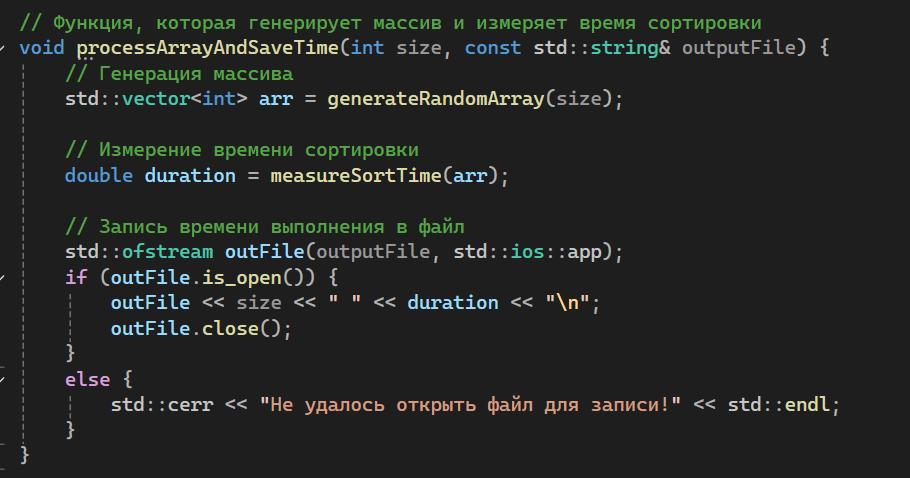


Также с помощью другой функции мы запускаем один и тот же датасет (1е4.txt или 1е5.txt) 100 раз:

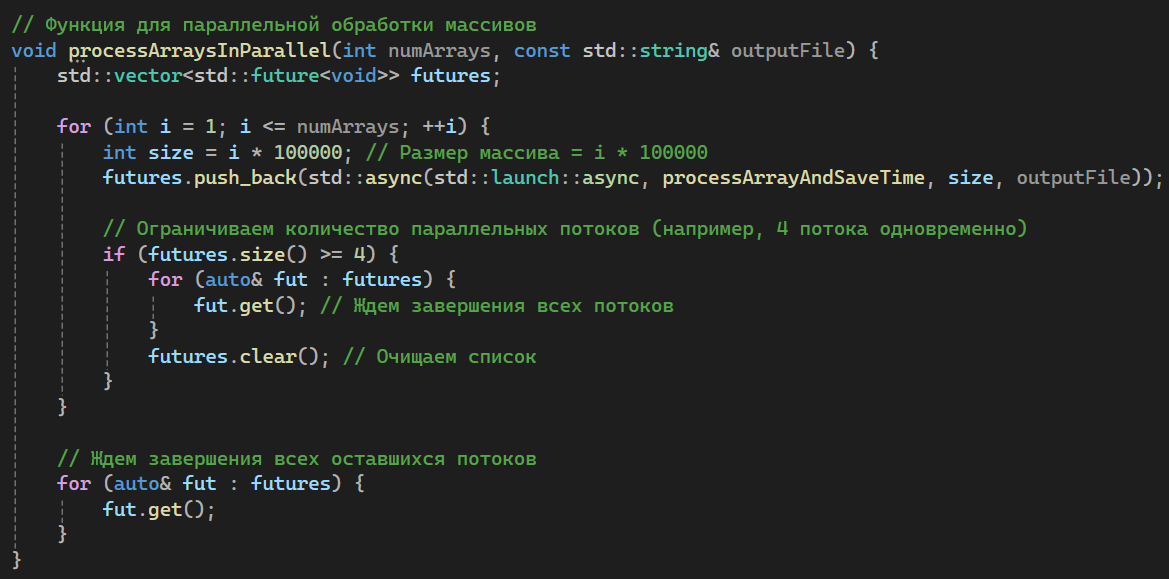


2) Создание массивов с данными для задания 5.



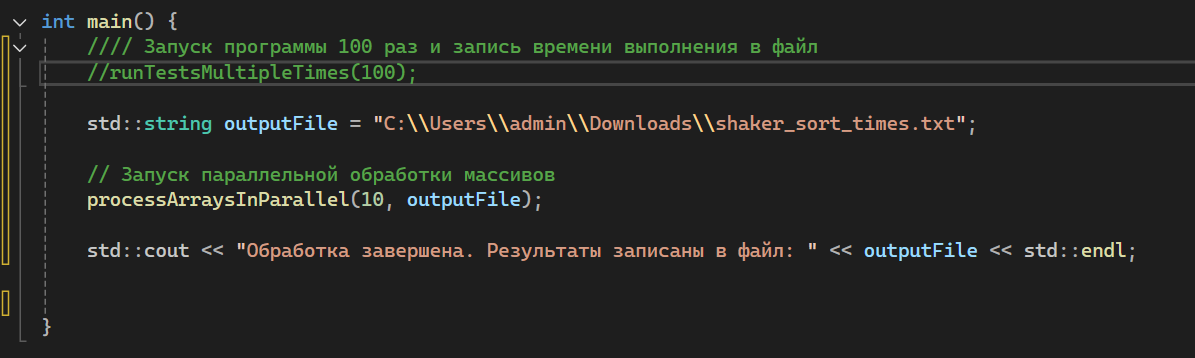


Из за сложности обработки данных пришлось пробовать параллельные вычисления:



Заметим что пункты 1) и 2) должны выполняться по отдельности потому что в обоих создается какой – то массив arr который сортируется в функции сортировке по своей ссылке.

То есть в главной программе создающей датасеты для графиков с помощью вызовов 1) или 2), опять же мы будем выводить 1) или 2) ,не не их вместе:



Функции формирующие датасеты для графиков идентичные в точности до вызова конкретной сортировки.

* **Реализация Shacker Sort**

1. Установить left = 0, right = n-1

2. Пока left < right:

a. Выполнить проход слева направо:

i. Для каждого i от left до right-1:

- Если arr[i] > arr[i+1], обменять их местами

ii. Уменьшить right на 1 (поскольку последний элемент на месте)

b. Выполнить проход справа налево:

i. Для каждого i от right до left+1:

- Если arr[i] < arr[i-1], обменять их местами

ii. Увеличить left на 1 (поскольку первый элемент на месте)

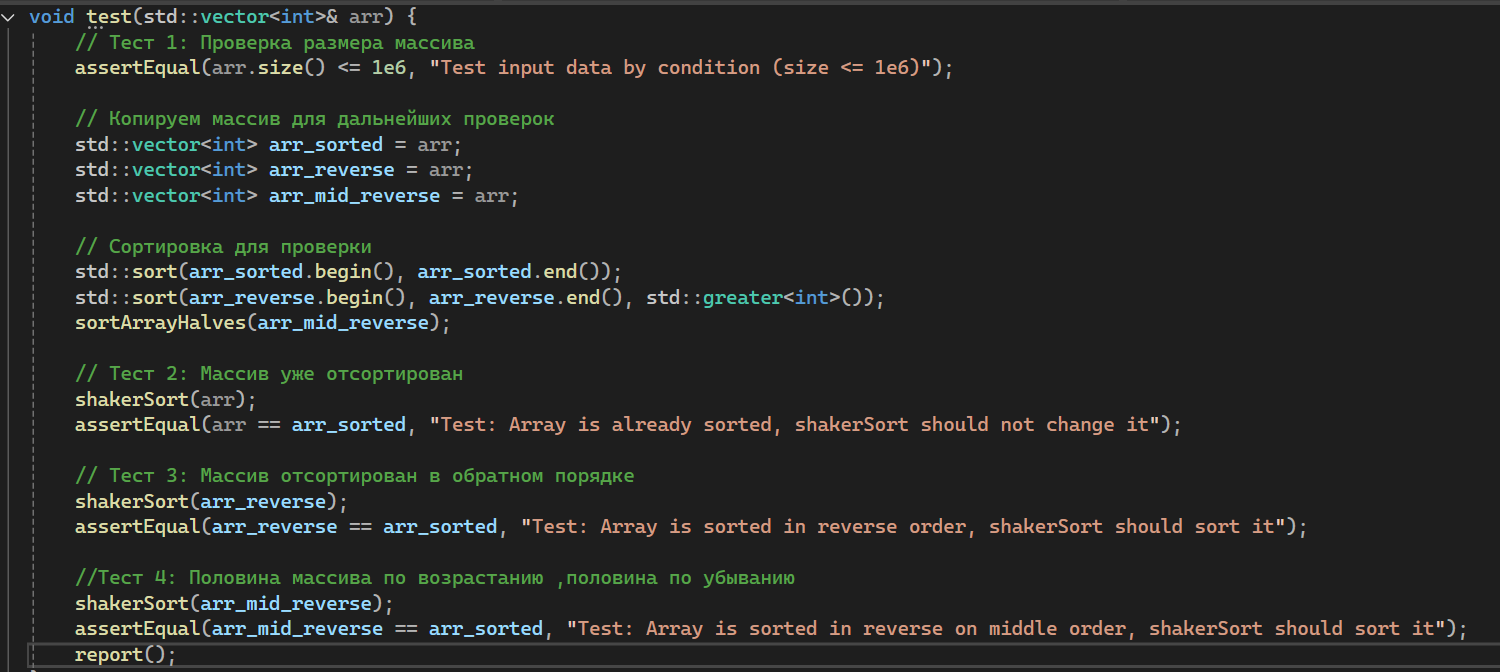
* Тесты для shacker sort

Первый тест – проверяет что входной массив меньше значения из условия

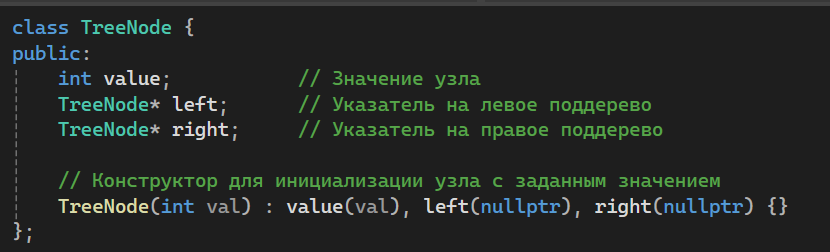
Второй тест – проверяет что алгоритм работает верно для уже отсортированного массива (это лучший случай)

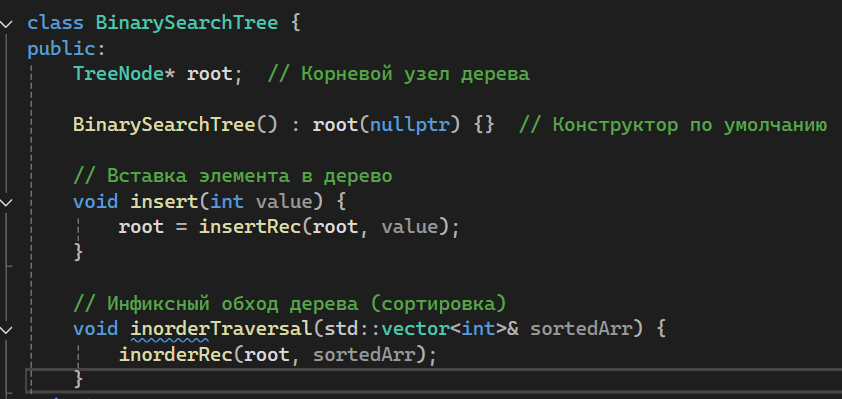
Третий тест – проверяет что алгоритм верно работает для отсортированного в порядке убывания ( это худший случай , его мы получаем просто отсортировав массив наоборот)

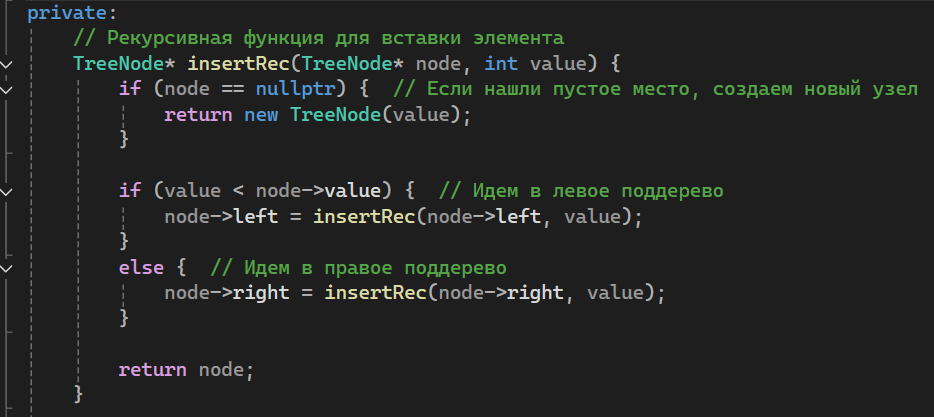
Четвертый тест – проверяет верность работы алгоритма для среднего случая, то есть для случая ,когда половина массива отсортирована , а вторая отсортирована наоборот. ( для реализации этого теста была написана специальная функция sortArrayHalves)

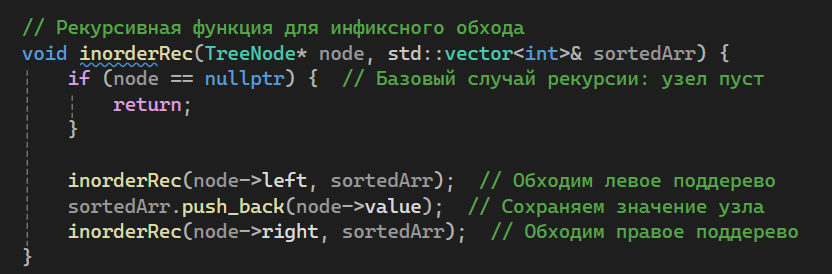


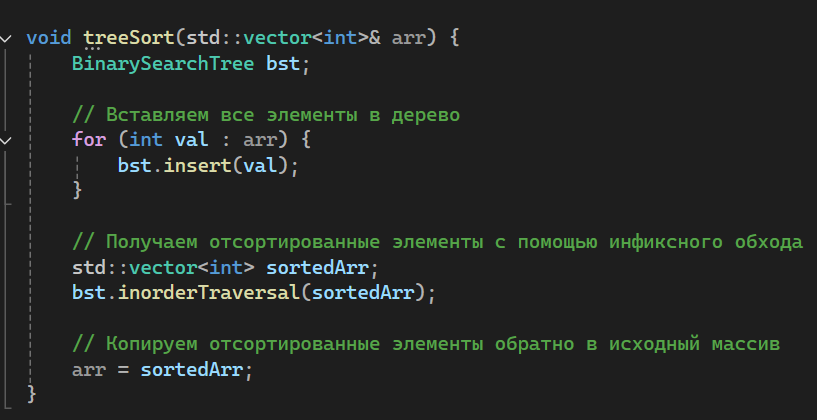
* **Реализация Tree sort**







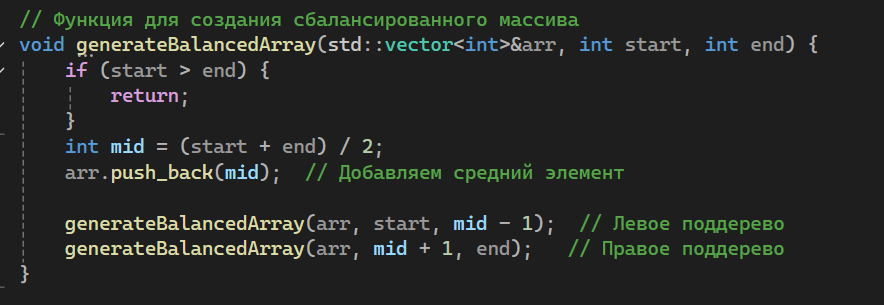




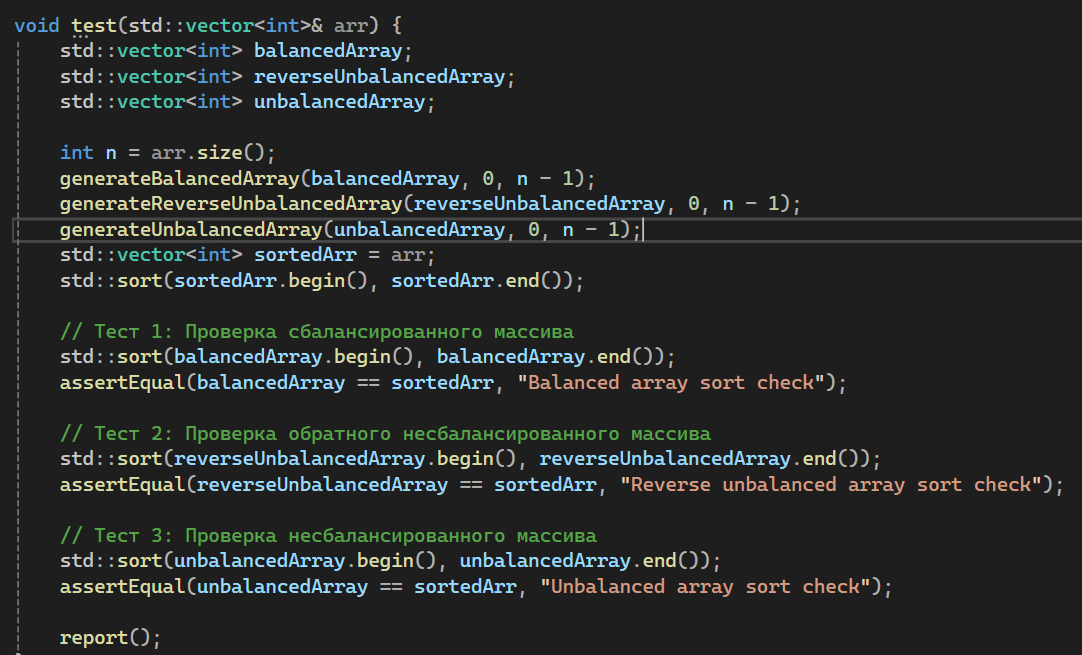
* Тесты для Tree Sort:

Тест 1 – Проверка условий ТЗ

Тест 2 – Лучший случай ,когда массив равномерно распределен (элемент то больше то меньше ), для того чтоб граф был симметричным. Функция для создания сбалансированного:

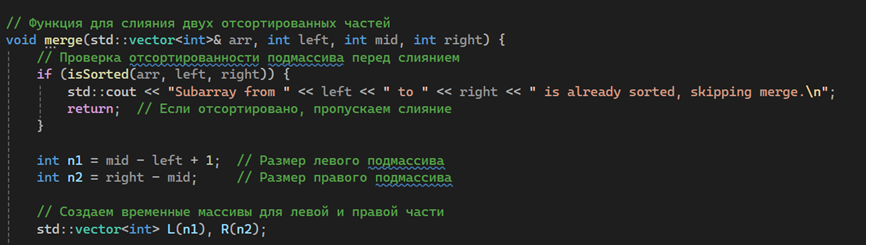


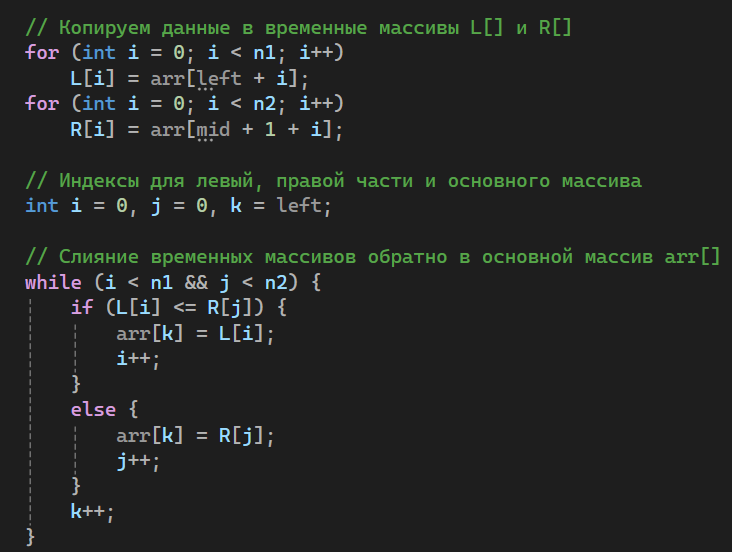
Тест 3 и 4 – Массив уже отсортирован (возр. или убыв.) поэтому дерево будет перекошено в одну строну ,что плохо по времени.

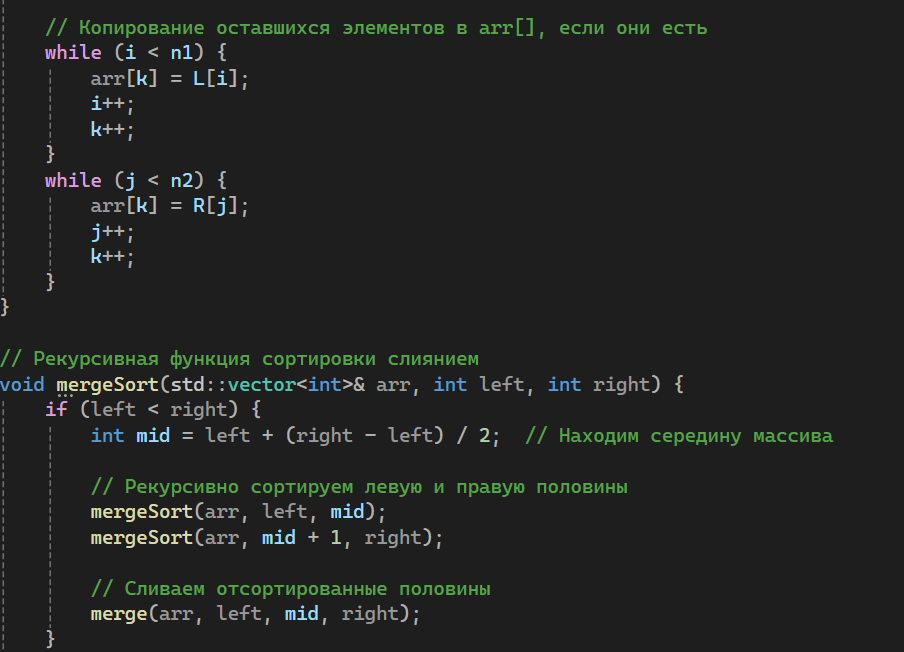


* **Реализация Merge Sort**

Функция isSort была убрана в дальнейшем так как ,если массив отсортирован Merge sort и так тратит О(n), поэтому добавление функции не оптимизировано.





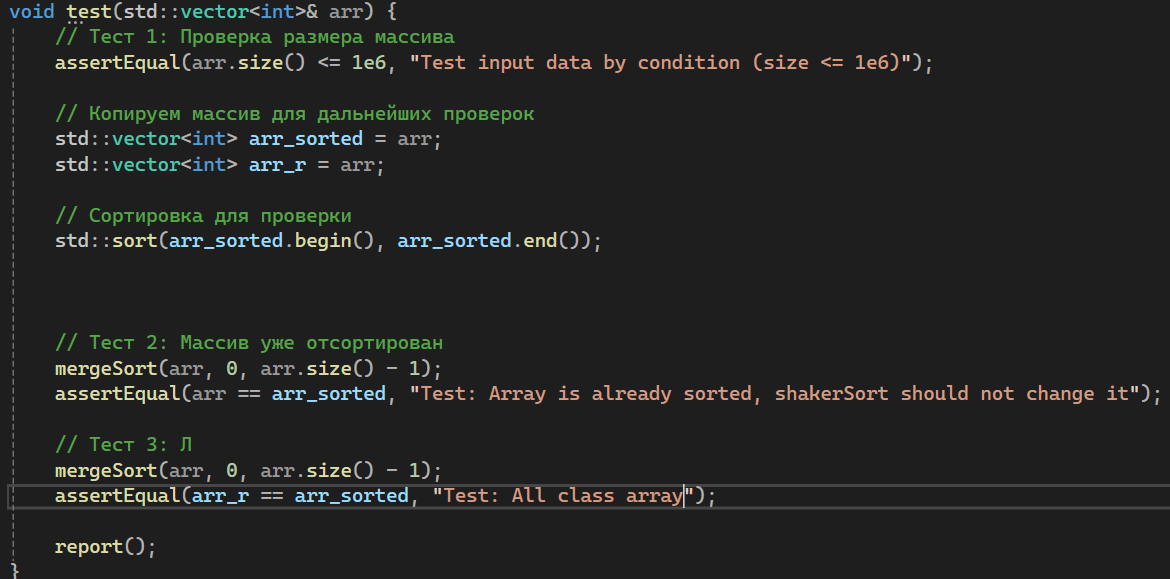


* Тесты для Merge Sort

1)Тест 1 – проверка условия ТЗ

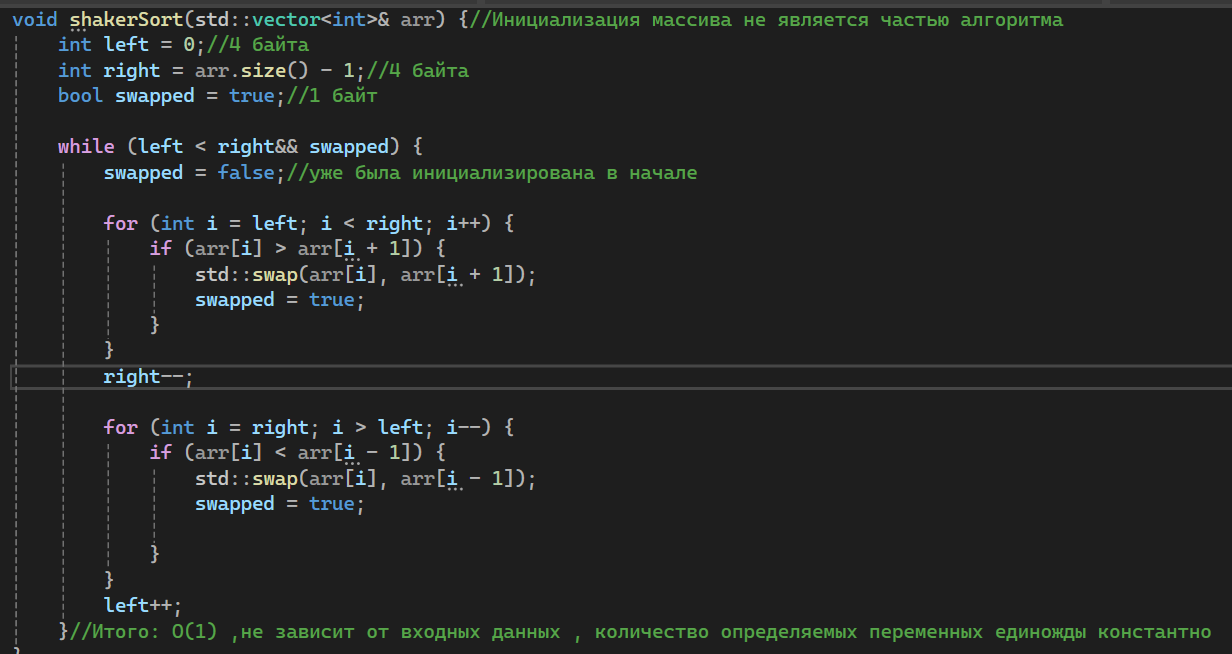
2)Тест 2 – лучший случай, когда массив уже отсортирован

3)Тест 3 – проверка что рассортированный массив правильно сортируется (средний и сложный случаи)

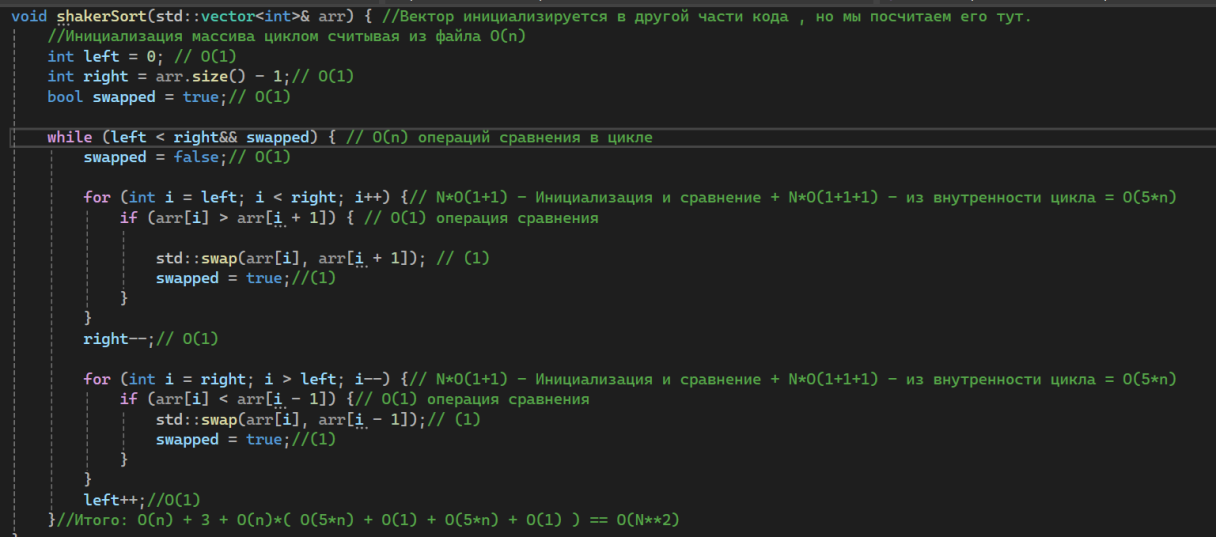


**Экспериментальная часть**

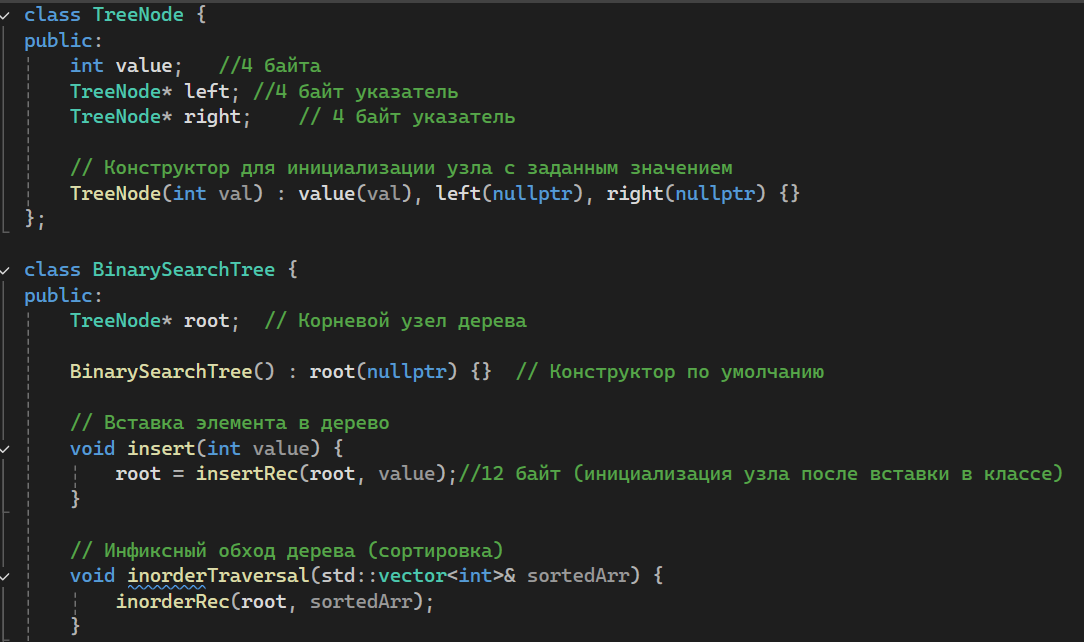
Подсчет памяти для Shaker Sort

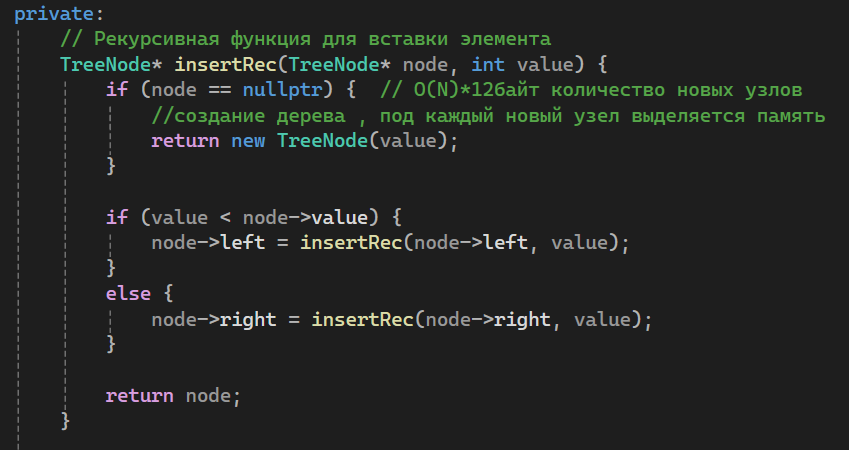


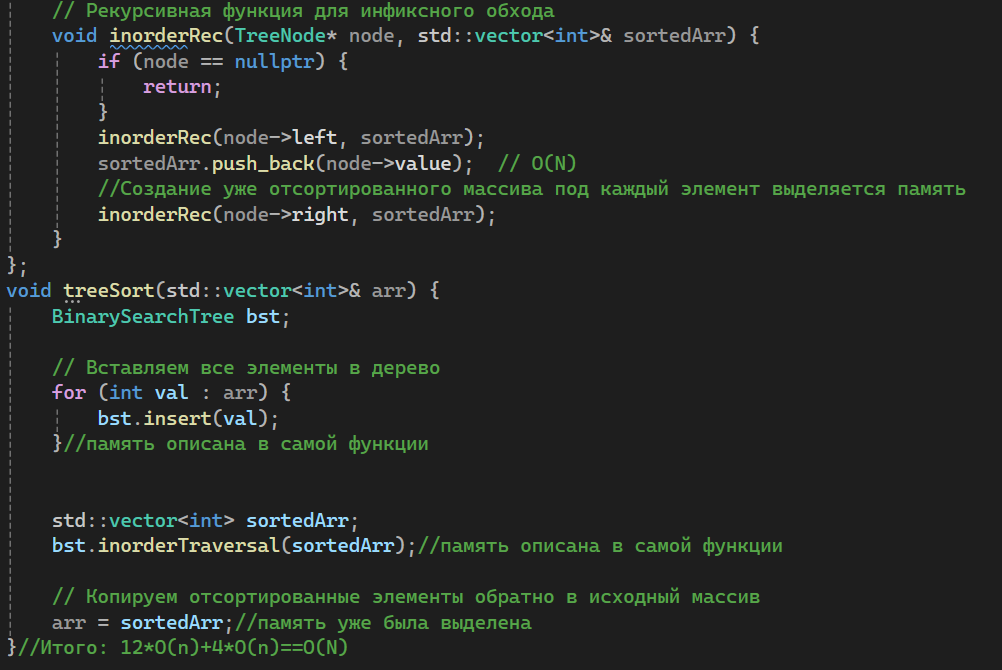
Подсчет асимптотики для Shaker Sort:



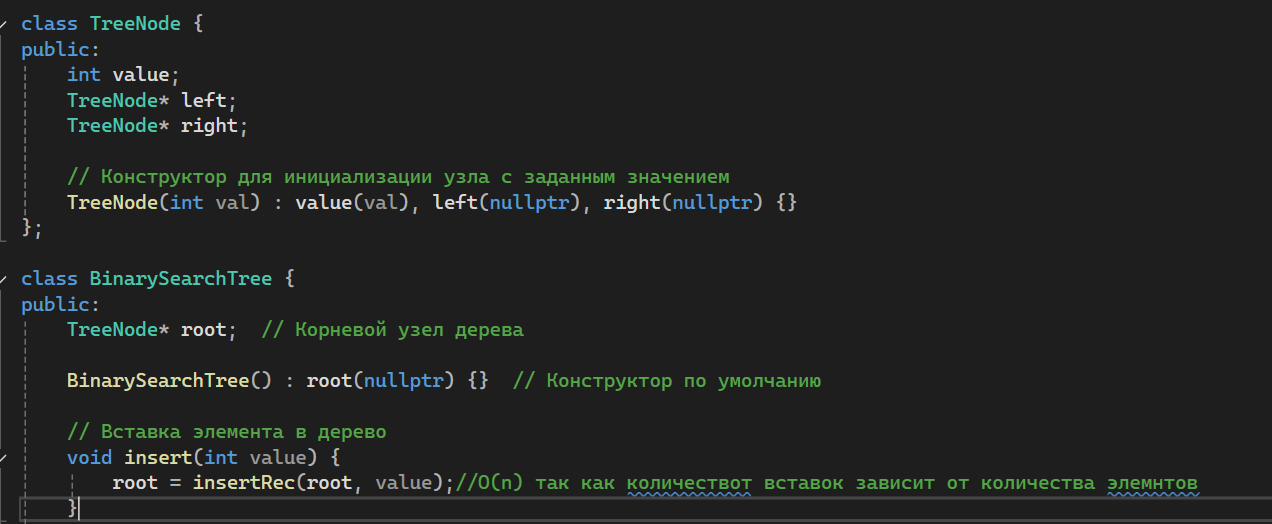
Подсчет памяти для Tree Sort:

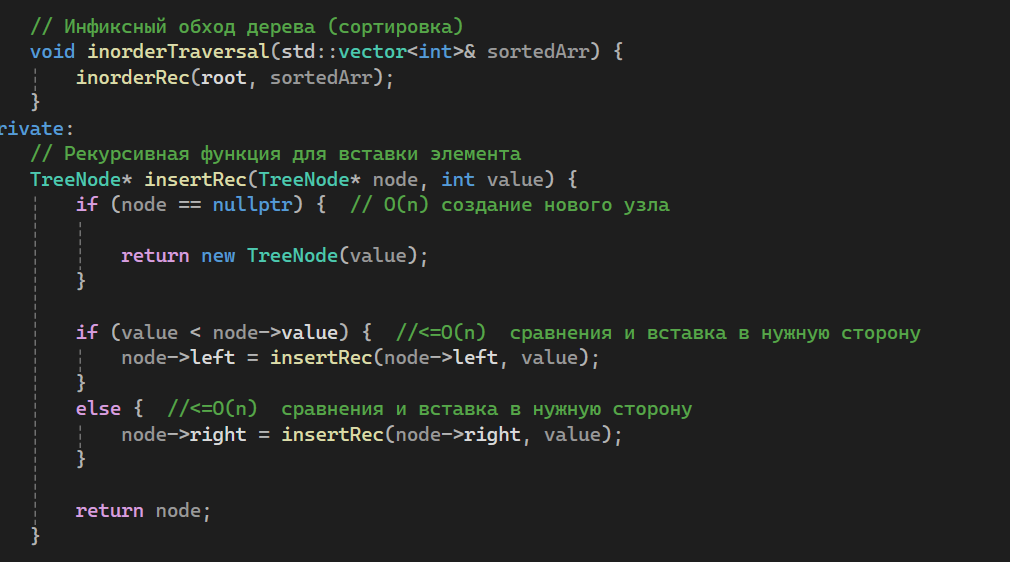


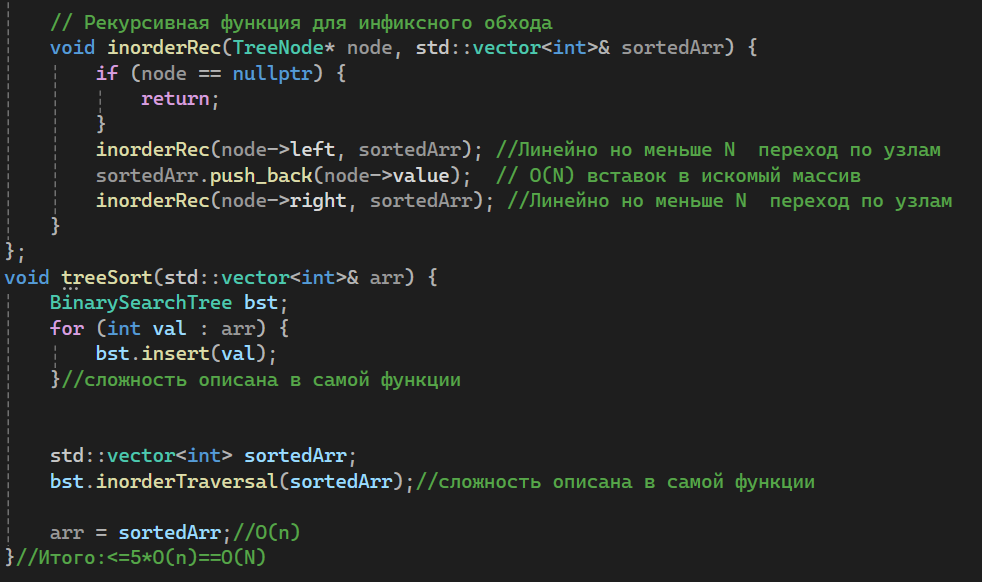




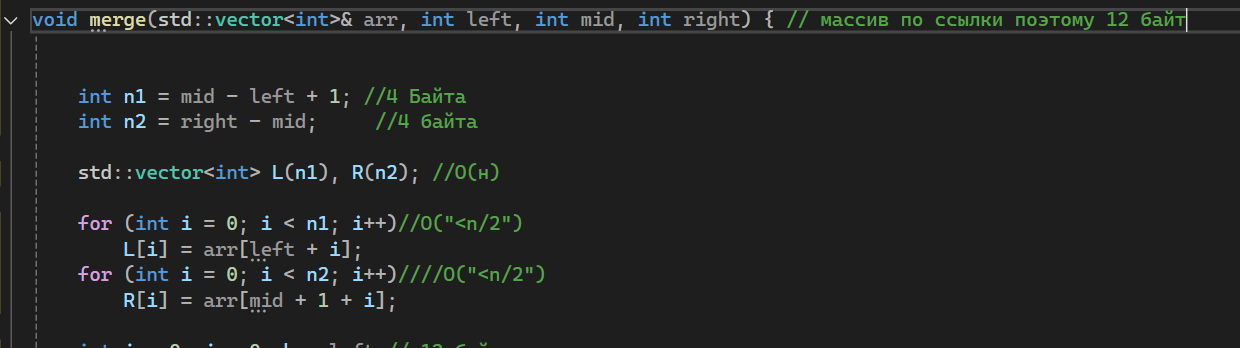
Подсчет сложности для Tree Sort:

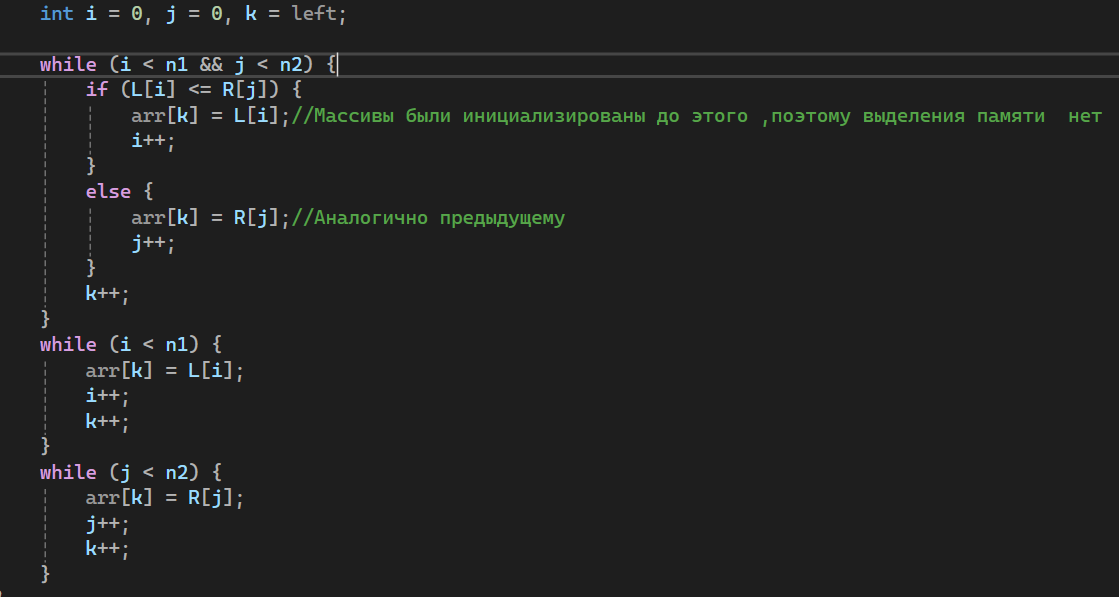


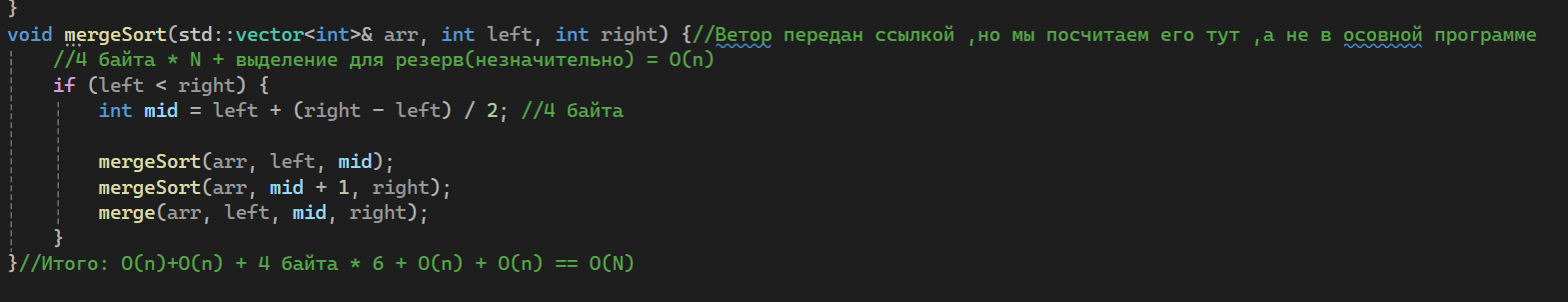




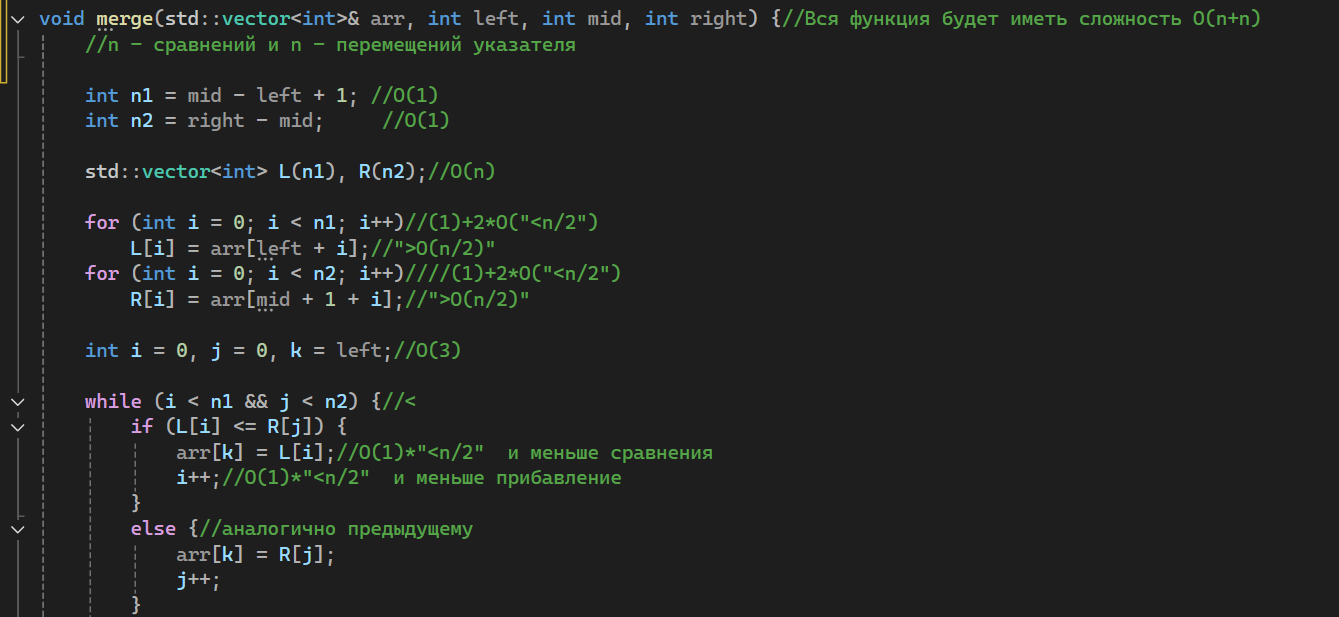
Подсчет памяти для Merge Sort:

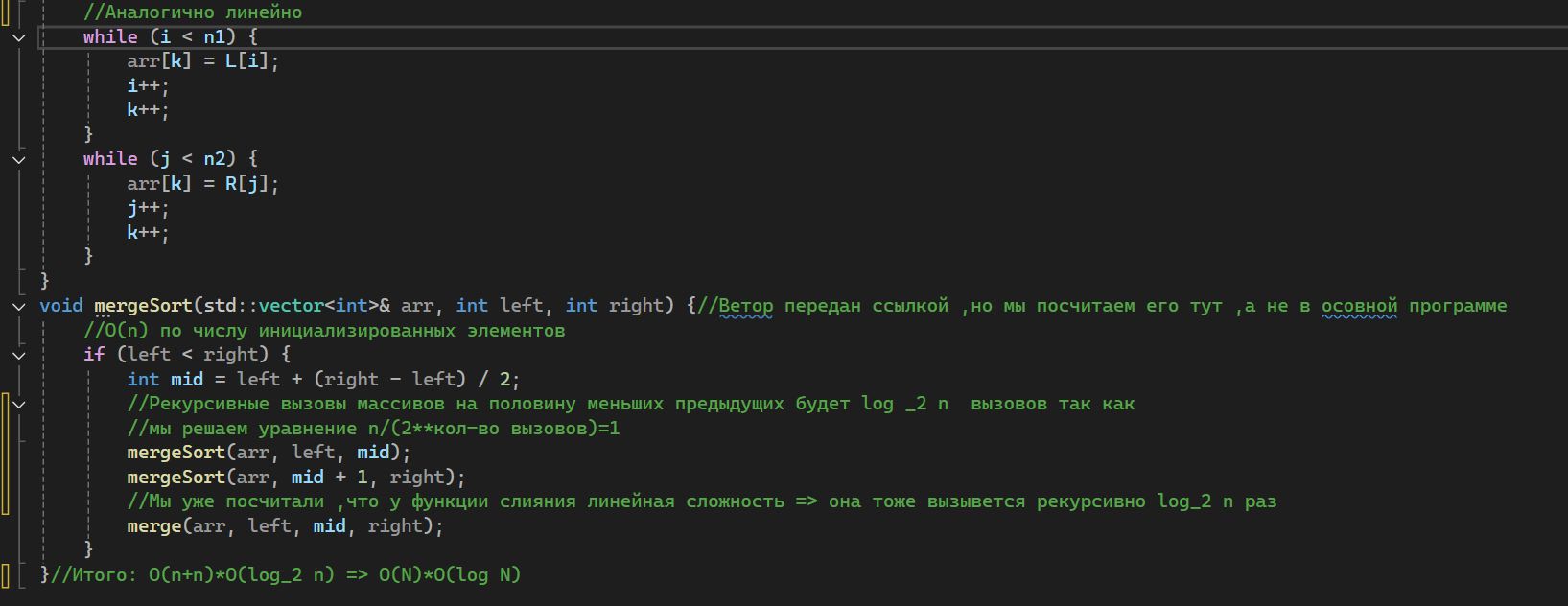






Подсчет асимптотики Merge Sort:





**Исследование практических результатов**:

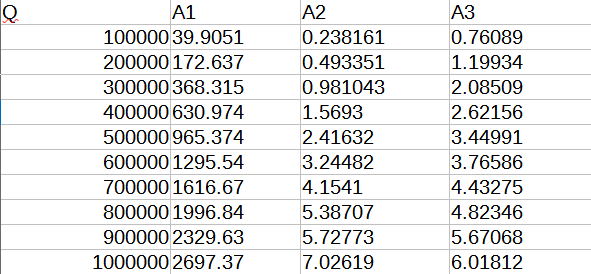
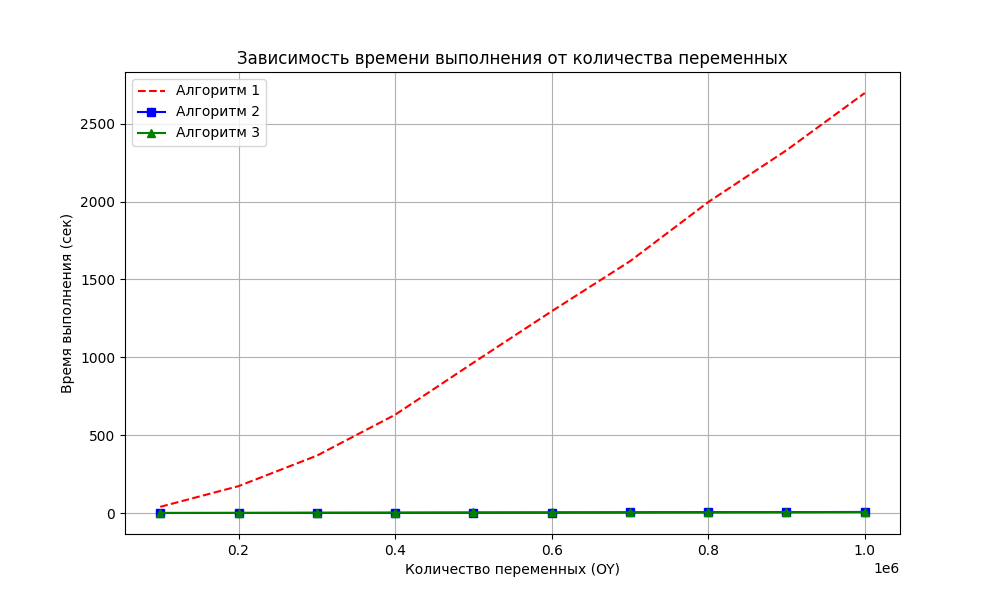


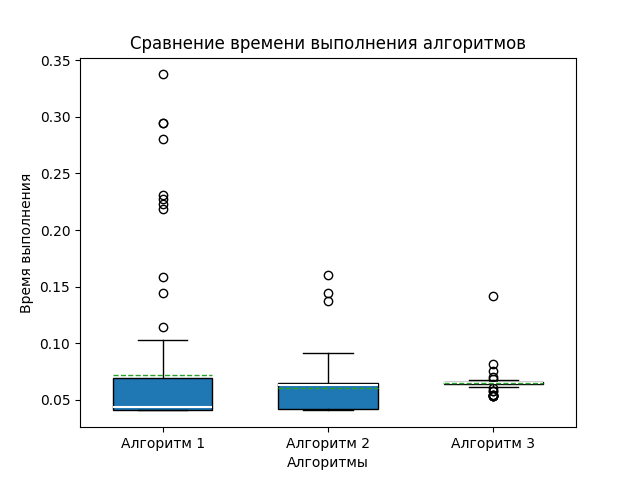
Таблица 1 – Таблица ,Q- Кол-во переменных, время в миллисекундах



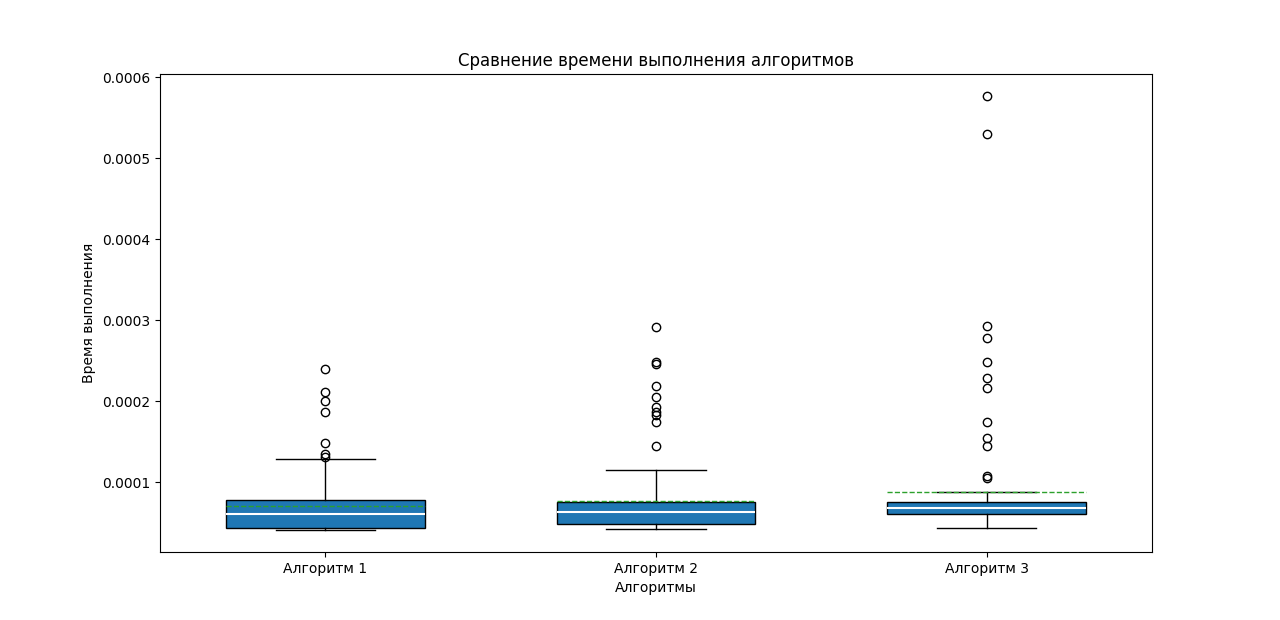
Изображении №1 – График зависимости времени работы от количества данных. Построен по Таблице 1.

Анализ графика 1: Был выбран довольно крупный масштаб ,но все равно при этом видно ,что шейкер сорт образует кривую. При этом Мерж сорт и три сорт почти совпадают по результатам потому что при рандомном образовании массивов датасета, функции рандома имеет довольно хорошее распределение и дерево получается симметричным , ветки равномерно распределены – а как мы знаем если дерево сбалансировано , то алгоритм может выполняться и за О(log n \* n) как и постоянная сложность сортировки слиянием. С такими большими шагами такая асимптотика почти неотличима от константной ,что мы и видим. (логарифм на довольно больших расстояниях может почти совпадать с константой)

Данные для диаграмм с размахом слишком большие ,могут быть представлены виде файлов по требованию.



Изображение №2 – Диаграмма размаха для 1е4



Изображение №3 – Диаграмма размаха для 1е5

Анализ изображения 2,3: shaker sort – контейнер в обоих диаграммах самый объемный , также самые длинные усы и большое количество выбросов находящихся очень далеко от среднего значение (зеленая черта). Это говорит о том что шейкер сорт совершенно нестабилен ведь массив был один и тот же. Это подтверждается еще тем ,что его стабильность (на самом деле случайно) увеличилась при прогонке по большему массиву(для 1е5 уменьшилось количество выбросов и объем контейнера).

Tree sort – Усы и контейнер короче чем в шейкер сорт ,значит он более стабильной ,медиана на первой диаграмме выше середины контейнера , а на второй примнерно в середине , опять же это случайно и доказывает нестабильность , количество выбросов увеличилось при увеличения количества данных (доказывает нестабильность)

Merge sort – Самая стабильная на первой диаграмме , так как имеет меньший контейнер, самые короткие усы и меньшее всего выбросов, медиана расположена по середине контейнера => (данные одинаковые) следовательно время было распределено симметрично. Но в связи с «усталостью железа» на обработке последнего файла , количество выбросов на второй диаграмме сильно увеличилось.

Вывод из исследования:

* Заключение

В ходе выполнения работы мною были реализованы алгоритмы сортировки разных характеристик по пространственной и временной сложности. Были построены два вида диаграмм доказывающие основные теоретические характеристики алгоритмов. Были отмечены аспекты диаграмм возможно связанные с характеристиками «железа». Для построения диаграмм мы написали специальные функции создающие и обрабатывающие большое в локальном смысле количество данных и создающие файлы с временем работы алгоритмов.

Заметим ,что шейкер сорт не применяется в рабочей практике из-за своей нестабильности и большой сложности по времени,

## Сортировка бинарным деревом тоже редко применяется в жизни , зато часто применяется ее улучшение - Splay tree (Тарьян Роберт и Слитор Даниэль - создатели).

## Сортировка слиянием (Джон фон Нейман – изобретатель) является довольно эталонным алгоритмом , а самое главное предсказуемым и стабильным , применяющимся во многих сферах, поэтому применяется в практике.

* Приложение:

Код для создания графиков и по созданным файлам.

Линейный график:



Контейнер с усами:

