

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ  
ИНСТИТУТ ФАКУЛЬТЕТ  
ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Утвержден на заседании кафедры

«Вычислительная

техника»

" \_\_\_\_ " \_\_\_\_\_ 20 \_\_\_\_ г. Заведующий

кафедрой

\_\_\_\_\_  
М.А. Митрохин

**ОТЧЕТ ПО УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКЕ**

(2023/2024 учебный год)

Балаев Глеб Сергеевич

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное  
обеспечение средств вычислительной техники и  
автоматизированных систем»

Форма обучения – очная Срок обучения в соответствии с ФГОС – 4 года

Год обучения \_\_\_\_\_ 1 \_\_\_\_\_ семестр \_\_\_\_\_ 2 \_\_\_\_\_

Период прохождения практики с 25.06.2024 по 8.07.2024

Кафедра «Вычислительная  
техника»

Заведующий кафедрой д.т.н., профессор, Митрохин М.А.

(должность, ученая степень, ученое звание, Ф.И.О.)

Руководитель практики к/н, доцент, Карамышева Н.С.

(должность, ученая степень, ученое звание)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ  
ИНСТИТУТ ФАКУЛЬТЕТ  
ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Утвержден на заседании кафедры

«Вычислительная

техника»

"\_\_" \_\_\_\_\_ 20\_\_ г. Заведующий

кафедрой

\_\_\_\_\_ М.А. Митрохин

**ИНДИВИДУАЛЬНЫЙ ПЛАН ПРОХОЖДЕНИЯ  
УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКИ**

(2023/2024 учебный год)

Балаев Глеб Сергеевич

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное  
обеспечение средств вычислительной техники и  
автоматизированных систем»

Форма обучения – очная Срок обучения в соответствии с

ФГОС – 4 года Год

обучения \_\_\_\_\_ 1 \_\_\_\_\_ семестр \_\_\_\_\_ 2 \_\_\_\_\_ Период

прохождения практики с 25.06.2024 по 8.07.2024

Кафедра «Вычислительная техника»

Заведующий кафедрой д.т.н., профессор, Митрохин М.А. \_\_\_\_\_

(должность, ученая степень, ученое звание, Ф.И.О.)

Руководитель практики к/н, доцент, Карамышева Н.С.

(должность, ученая степень, ученое звание)

№ п/п	Планируемая форма работы во время практики	Количество часов	Календарные сроки проведения работы	Подпись руководителя практики от вуза
1	Выбор темы и разработка индивидуального плана проведения работ	2	25.06.24 - 25.06.24	
2	Подбор и изучение материала по теме работы	15	26.06.24 – 01.07.24	
3	Разработка алгоритма	43	01.07.24 – 02.07.24	
4	Описание алгоритма и программы	18	02.07.24 – 03.07.24	
5	Тестирование	5	03.07.24 – 05.07.24	
6	Получение и анализ результатов	10	05.07.24 – 06.07.24	
7	Оформление отчёта	15	06.07.24 – 08.07.24	
	<b>Общий объём часов</b>	108		

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**ОТЧЁТ**  
**О ПРОХОЖДЕНИИ УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКИ**

(2023/2024 учебный год)

Балаев Глеб Сергеевич

---

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное  
обеспечение средств вычислительной техники и  
автоматизированных систем»

Форма обучения – очная Срок обучения в соответствии с ФГОС – 4 года

Год обучения 1 семестр 2

Период прохождения практики с 25.06.2023 по 8.07.2024

Кафедра «Вычислительная  
техника»

---

Балаев Г.С. выполнял практическое задание «Двоичная сортировка». На первоначальном этапе был изучен и проанализирован алгоритм двоичной сортировки, был выбран метод решения и язык программирования C++. Также, осуществил работу с реализацией алгоритма сортировки. Оформила отчёт.

Бакалавр Балаев Г.С. \_\_\_\_\_ " \_\_\_\_ " \_\_\_\_\_ 2024  
\_\_\_\_\_ г.

Руководитель Карамышева Н.С. \_\_\_\_\_ " \_\_\_\_ " \_\_\_\_\_ 2024 г.  
практики

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**ОТЗЫВ**

**О ПРОХОЖДЕНИИ УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКИ**

(2023/2024 учебный год)

Балаев Глеб Сергеевич

---

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное  
обеспечение средств вычислительной техники и  
автоматизированных систем»

Форма обучения – очная Срок обучения в соответствии с ФГОС – 4 года

Год обучения 1 семестр 2

Период прохождения практики с 25.06.2024 по 8.07.2024

Кафедра «Вычислительная  
техника»

---

В процессе выполнения практики Балаев Г.С. решал следующие задачи: реализация алгоритма сортировки.

За период выполнения практики были освоены основные понятия и технологии сортировки вставками, реализованы методы работы с файлами. Во время выполнения работы Балаев Г.С. показала себя ответственным, добросовестным учеником, знающим свой предмет, имеющим представление о современном состоянии науки, владеющим современными общенаучными знаниями по информатике и вычислительной технике, программированию и сортировке.

За выполнение работы Балаев Г.С. заслуживает оценки «      ».

Руководитель практики к/н, доцент, Карамышева Н.С. «      » 2024г.

## Содержание

Введение .....	2
1    Постановка задачи .....	3
1.1 Достоинства алгоритма бинарной сортировки .....	3
1.2 Недостатки алгоритма бинарной сортировки .....	3
1.3 Типичные сценарии применения данного алгоритма.....	3
2    Выбор решения.....	4
3    Описание программы .....	5
4.   Схемы программы .....	11
5    Тестирование программы.....	12
6    Отладка .....	13
7    Совместная разработка.....	14
Заключение .....	16
Список используемой литературы .....	17
Приложение А. Листинг программы .....	18

## Введение

В эпоху стремительного развития компьютерных технологий сортировка данных стала одним из ключевых процессов в обработке информации. Эта задача широко распространена в различных профессиональных областях.

Алгоритмы сортировки представляют собой особую категорию алгоритмов, которые находят применение практически во всех аспектах обработки информации. Их тесная взаимосвязь позволяет выделить их в отдельный класс. Основная цель применения алгоритмов сортировки - оптимизация последующего поиска. Например, использование словарей было бы затруднительно без алфавитного порядка слов.

Значимость сортировки заключается в том, что на ее примере можно продемонстрировать множество фундаментальных методов и приемов построения алгоритмов. Сортировка также иллюстрирует разнообразие алгоритмических подходов к решению одной задачи, причем некоторые из них имеют определенные преимущества перед другими. Усложнение алгоритма может значительно повысить его эффективность и быстродействие по сравнению с более простыми методами. В общем понимании, сортировка — это процесс упорядочивания элементов множества в определенной последовательности.

Сортировка с помощью двоичного дерева – это универсальный алгоритм сортировки, который заключается в построении двоичного дерева поиска по ключам массива и сборке результирующего массива путем обхода узлов построенного дерева в необходимом порядке следования ключей. Данная сортировка является оптимальной при получении данных путем непосредственного чтения из потока (файла, сокета, консоли). [1]

# **1 Постановка задачи**

Поставленная задача: необходимо заполнить массив из n-ого количества элементов случайными числами, записать данные элементы в отдельный файл. После этого выполнить двоичную сортировку над данными, находящимися в массиве, записать отсортированные данные в другой файл, посчитать время выполнения.

Использовать сервис GitHub для совместной работы. Создать и выложить коммиты, характеризующие действия, выполненные каждым участником бригады.

Оформить отчет по проведенной практике.

## **1.1 Достоинства алгоритма бинарной сортировки**

- алгоритм эффективен при поиске определенного элемента;
- экономия памяти;
- простая реализация алгоритма.

## **1.2 Недостатки алгоритма бинарной сортировки**

- ограничение 2 дочерними узлами;
- высокая алгоритмическая сложность  $O(n^2)$ ;
- сложный алгоритм балансировки.

## **1.3 Типичные сценарии применения данного алгоритма**

- товары в магазине (сортировка по цене, году выпуска, габаритам, весу, срокам поставки);
- студенты в вузе (сортировка по среднему балу, кол-ву прогулов, уровню IQ, числу хвостов, ФИО);
- города/страны (сортировка по населению, рождаемости, ВВП, ВВП на душу населения);



## 2 Выбор решения

Нашей бригадой было выбрано вести разработку в среде Microsoft Visual Studio на языке C++ с использованием библиотеки SFML. [2]

Для написания данной программы будет использован язык программирования C++. Язык программирования C++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день C++ является одним из самых популярных и распространенных языков. C++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко C++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. [3]

Microsoft Visual Studio — это программная среда по разработке приложений для ОС Windows, как консольных, так и с графическим интерфейсом.

Для удобства совместной разработки был использован сервис GitHub Desktop. GitHub Desktop — это приложение, которое помогает работать с файлами, размещенными на GitHub или других службах размещения Git. GitHub Desktop можно использовать вместе с любыми инструментами, которые необходимо внести в проект.

### 3 Описание программы

#### Структура данных «Node»:

Для реализации алгоритма бинарной сортировки были выбраны структуры. Структура «Node» используется для представления узлов дерева. Она включает четыре поля:

- «int value»: значение, хранящееся в узле.
- «Node\* left»: указатель на левое поддерево.
- «Node\* right»: указатель на правое поддерево.
- «Node\* parent»: указатель на родительский узел.

#### Код структуры «Node»:

```
Node* createNode(int val, Node* parent = nullptr) {  
    Node* newNode = new Node;  
    newNode->value = val;  
    newNode->parent = parent;  
    return newNode;  
}
```

#### Функция «createNode»:

##### Аргументы функции:

- «val»: Значение, хранящееся в узле.
- «parent»: Указатель на родительский узел (по умолчанию «nullptr»).

##### Возвращаемое значение:

- Указатель на новый узел «Node».

##### Логика работы:

1. Создаем новый динамический объект типа Node, используя оператор new.
2. Присваиваем полю value значение val.
3. Инициализируем указатель на родителя parent переданным значением.
4. Указатели на левое и правое поддерева (left и right) устанавливаем в nullptr (по умолчанию).
5. Возвращаем указатель на созданный узел.

Причины выбора реализации:

- Простота функции: логика создания узла изолирована для повторного использования.
- Использование указателя на родителя упрощает навигацию по дереву.

Код функции «createNode»:

```
Node* createNode(int val, Node* parent = nullptr) {  
    Node* newNode = new Node;  
    newNode->value = val;  
    newNode->parent = parent;  
    return newNode;  
}
```

**Функция «InsertNode»:**

Аргументы:

- «Node\*& root»: Ссылка на корневой узел дерева.
- «int val»: Значение, которое добавляется в дерево.

Логика работы:

1. Если корневого узла нет ('root == nullptr'), создается новый узел и назначается корнем.
2. В противном случае начинается поиск подходящего узла для добавления нового элемента:

- Узел «current» инициализируется как корень.
- Узел «parent» хранит родительский узел текущего узла.
- Пока не найдено подходящее место («current != nullptr»):
  - Если значение меньше текущего узла, переход к левому поддереву.
  - Иначе переход к правому поддереву.
- После нахождения нужного узла:
- Если значение меньше узла «parent», новый узел добавляется в левое поддерево.
- Иначе в правое.

### Причины выбора реализации:

- Использование итеративного подхода вместо рекурсивного для избежания переполнения стека вызовов при большом количестве элементов.
- Сохранение указателя на родителя упрощает навигацию и модификацию дерева.

### Код функции «InsertNode»:

```
void insertNode(Node*& root, int val) {
    if (root == nullptr) {
        root = createNode(val);
    }
    else {
        Node* current = root;
        Node* parent = nullptr;
        while (current != nullptr) {
            parent = current;
            if (val < current->value) {
                current = current->left;
            }
            else {
                current = current->right;
            }
        }
        if (val < parent->value) {
            parent->left = createNode(val, parent);
        }
        else {
            parent->right = createNode(val, parent);
        }
    }
}
```

### Функция «buildBinarySortTree»:

#### Аргументы:

- ``const std::vector<int>& arr``: Входной массив чисел.

#### Возвращаемое значение:

- ``Node*``: Указатель на корневой узел дерева.

Логика работы:

1. Инициализируем переменную `root` указателем на корневой узел дерева как `nullptr`, так как дерево изначально пустое.
2. Для каждого элемента `val` в массиве `arr` вызываем функцию `insertNode`, которая добавляет элемент в дерево. Если корневой узел пуст (т.е `root == nullptr`), создаем новый корневой узел с помощью `createNode`. Если корневой узел уже существует, выполняем итеративный поиск подходящего места для нового узла и вставляем его.
3. После обработки всех элементов массива возвращаем указатель на корневой узел дерева.

Причины выбора реализации:

- Простота функции: логика создания узла изолирована для повторного использования.
- Использование указателя на родителя упрощает навигацию по дереву.

Код функции «`buildBinarySortTree`»:

```
Node* buildBinarySortTree(const std::vector<int>& arr) {  
    Node* root = nullptr;  
    for (int val : arr) {  
        insertNode(root, val);  
    }  
    return root; // 3. Возвращаем корень дерева  
}
```

**Функция «`collectSortedValues`»:**

Аргументы:

- `Node* root`: Указатель на корневой узел бинарного дерева сортировки.
- `std::vector<int>& sortedArray`: Ссылка на вектор, куда будут собраны отсортированные значения.

Логика работы:

1. Если указатель на корневой узел не равен nullptr:
  - 1.1 Сначала рекурсивно вызываем collectSortedValues для левого поддерева.
  - 1.2 Добавляем значение текущего узла (root->value) в вектор sortedArray.
  - 1.3 Затем рекурсивно вызываем collectSortedValues для правого поддерева.
2. Если указатель на корневой узел равен nullptr (базовый случай рекурсии), функция завершает работу без изменений.

Причины выбора реализации:

- Использование рекурсии: Позволяет легко обойти дерево в порядке "левый - корень - правый", что гарантирует отсортированный порядок значений.
- Эффективность: Функция добавляет значения в отсортированном порядке благодаря свойствам двоичного дерева поиска.

Код функции «collectSortedValues»:

```
void collectSortedValues(Node* root, std::vector<int>& sortedArray) {  
    if (root != nullptr) {  
        collectSortedValues(root->left, sortedArray);  
        sortedArray.push_back(root->value);  
        collectSortedValues(root->right, sortedArray);  
    }  
}
```

**Функция «binaryTreeSort»:**

Аргументы:

- const std::vector<int>& arr: Входной массив чисел, которые необходимо отсортировать.

Возвращаемое значение:

- std::vector<int>: Отсортированный вектор чисел.

Логика работы:

1. Вызываем функцию `buildBinarySortTree`, которая строит бинарное дерево сортировки на основе входного массива `arr`. Это дерево будет содержать все элементы из входного массива в виде узлов.
2. Инициализируем пустой вектор `sortedArray` для хранения отсортированных значений.
3. Вызываем функцию `collectSortedValues`, которая обходит дерево в симметричном порядке (`in-order traversal`) и добавляет значения узлов в вектор `sortedArray`. Симметричный обход гарантирует, что значения будут добавлены в отсортированном порядке.
4. Возвращаем вектор `sortedArray`, содержащий отсортированные элементы.

Причины выбора реализации:

- Эффективность: Бинарное дерево поиска позволяет выполнять сортировку с эффективностью порядка  $O(n \log n)$ .
- Сохранение порядка: Симметричный обход дерева обеспечивает отсортированный порядок элементов.
- Простота кода: Использование функций `buildBinarySortTree` и `collectSortedValues` изолирует сложную логику в отдельных функциях, что делает код более понятным и модульным.

—

Код функции «`collectSortedValues`»:

```
std::vector<int> binaryTreeSort(const std::vector<int>& arr) {  
    Node* root = buildBinarySortTree(arr);  
    std::vector<int> sortedArray;  
    collectSortedValues(root, sortedArray);  
    return sortedArray;  
}
```

Листинг программы приведен в приложении А.

## 4. Схемы программы

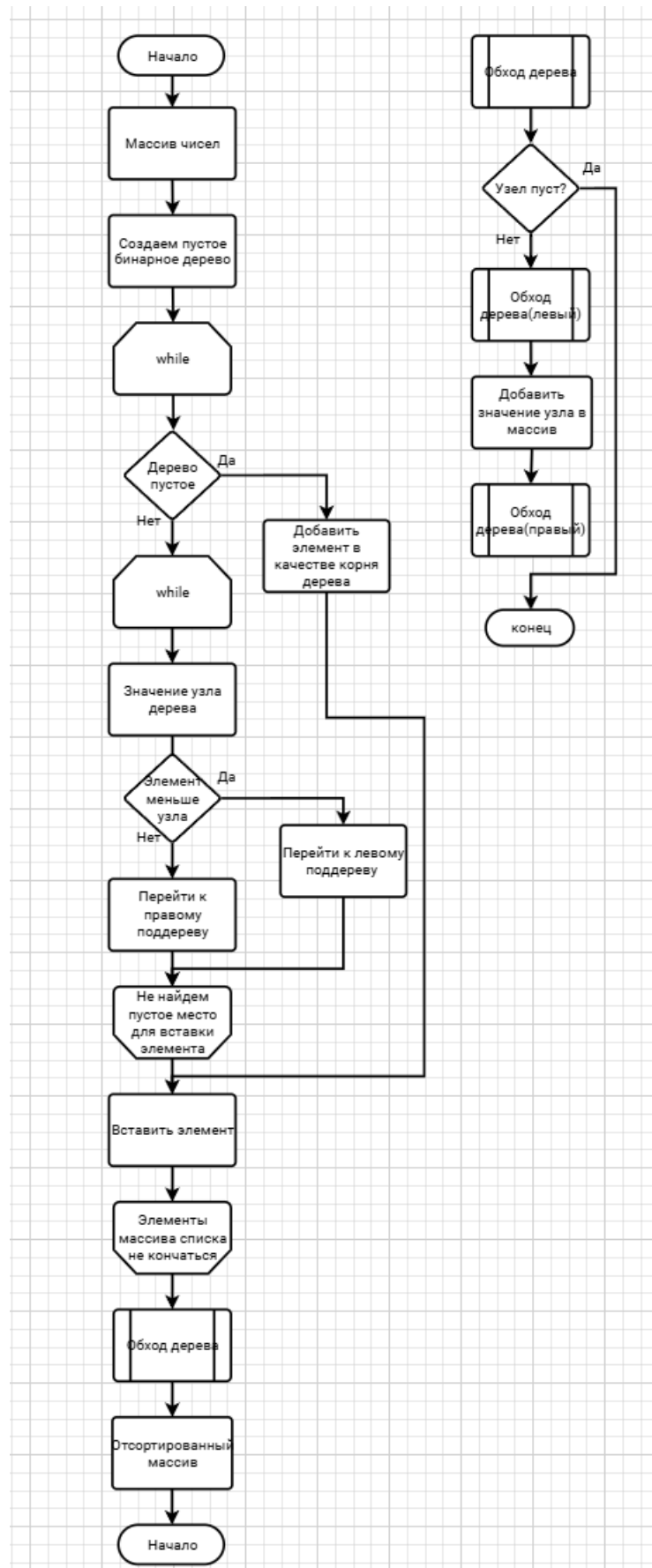


Рисунок 1 - Блок-схема алгоритма



## 5 Тестирование программы

Тестирование показало, что с увеличением количества элементов пропорционально увеличивается время работы программы, ниже представлен график результатов тестирования.



Рисунок 8 – Результаты тестирования пользовательских значений

## 6 Отладка

В качестве среды разработки была выбрана программа Microsoft Visual Studio, которая содержит в себе все необходимые средства для разработки и отладки модулей и программ.

Для отладки программы использовались точки останова и пошаговое выполнение кода программы, анализ содержимого локальных переменных.

Точки останова – это прерывание выполнения программы, при котором выполняется вызов отладчика. Отладчик является инструментом для поиска и устранения ошибок в программе, с помощью которого можно исследовать состояние программы.

Был использован метод бинарного поиска, он включает в себя разделение частей кода для упрощения процесса отладки. Это может быть особенно полезно, если причина ошибки находится в начале языка программирования, а фактическая ошибка ближе к концу.

Команда шаг с заходом (step into) выполняет следующую инструкцию в обычном пути выполнения программы, а затем приостанавливает выполнение программы, чтобы мы могли проверить состояние программы с помощью отладчика. Если выполняемый оператор содержит вызов функции, шаг с заходом заставляет программу перескакивать в начало вызываемой функции, где она приостанавливается

## 7 Совместная разработка

Для удобства совместной разработки был использован сервис GitHub Desktop.

Разделили роли, назначили исполнителей задачам.




<input type="checkbox"/>	<input checked="" type="checkbox"/>	реализация работы с файлами	#4 opened now by qqishy	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	реализация UI и наборов данных для тестирования	#3 opened 2 minutes ago by qqishy	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	реализация алгоритма	#2 opened 4 minutes ago by qqishy	

Рисунок 9 – распределение задач

Во время работы над данной практикой наша бригада осуществляла совместную работу в GitHub.

Мною было написан алгоритм сортировки, это было зафиксировано и загружено на удаленный репозиторий Github, на ветку algorithm, после было сделано слияние веток с main.

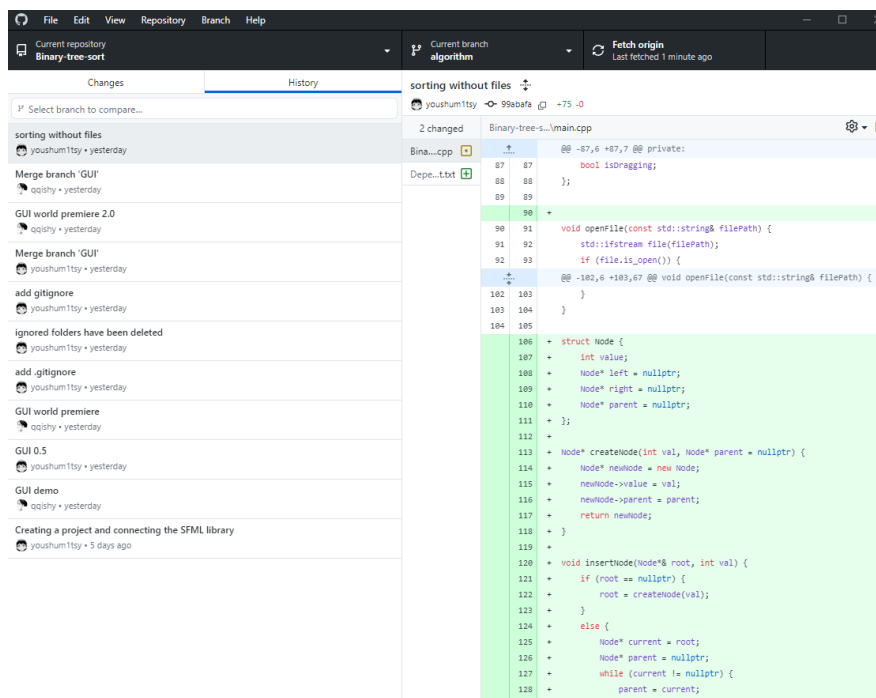


Рисунок 2 – изменения на ветке

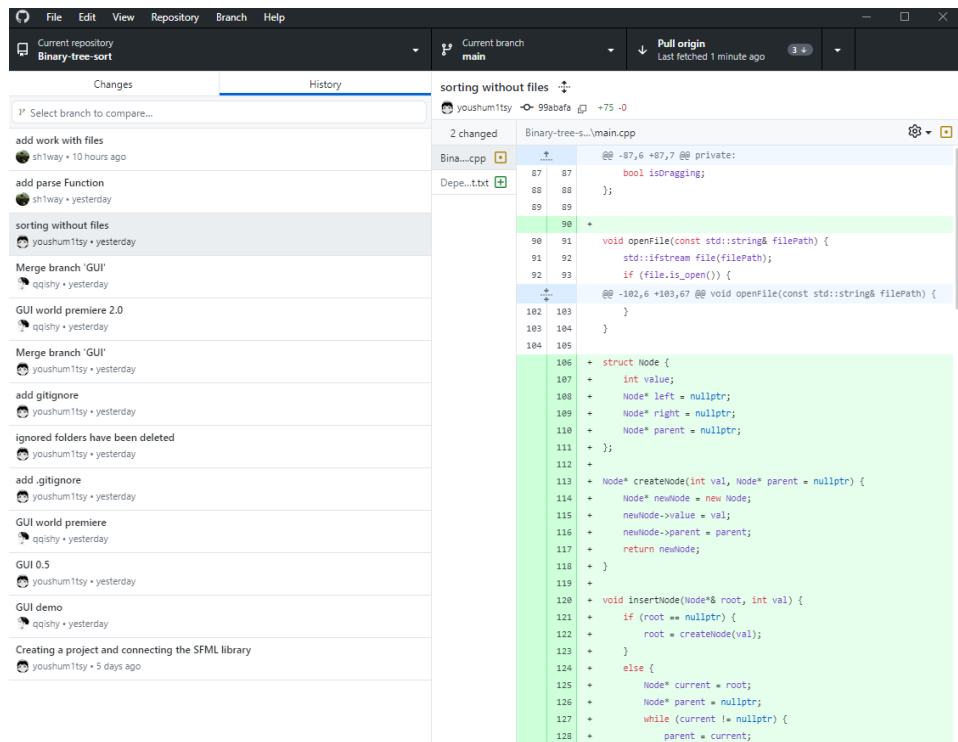


Рисунок 3 – слияние веток

Репозиторий находится на платформе GitHub в общественном доступе[3]

## **Заключение**

При выполнении данной работы были получены навыки совместной работы с помощью сервисов GitHub. Был изучен алгоритм двоичной сортировки.

Мною был создан алгоритм бинарной сортировки.

При выполнении практической работы были улучшены базовые навыки программирования на языке C++. Улучшены навыки отладки, тестирования программ и работы со сложными типами данных.

В дальнейшем программу можно улучшить путем подключения упрощающих реализацию данной сортировки библиотек и улучшения графического интерфейса.

## Список используемой литературы

1. К 78 Алгоритмы. Просто как дважды два / И. В. Красиков, И. Е. Красикова. — М. : Эксмо, 2007. — 256 с. — (Просто как дважды два).
2. Артур Морейра, Ян Халлер, Хенрик Фогелиус Хансон. Разработка игр на языке SFML. — Издательство Packt, 2013 – 296р.
3. Youshum1tsu. Binary tree sort: репозиторий исходного кода [Электронный ресурс]. URL: <https://github.com/youshum1tsy/Binary-tree-sort>

## Приложение А. Листинг программы

```
#include
<SFML/Graphics.hpp>

#include <string>

#include <cmath>

#include <iomanip>

#include <sstream>

#include <iostream>

#include <fstream>

#include <windows.h>

namespace MenuConstants {

    const int MainMenu = 0;

    const int RandomMenu =
1;

    const int File1 = 2;

    const int File2 = 3;

}

class Slider {

public:

    Slider(float x, float
y, float width, float
height, sf::Font& font) :
minValue(0),
maxValue(100000),
currentValue(0),
isDragging(false) {
```

```
track.setSize(sf::Vector2f(
width, height / 4));
```

```
track.setPosition(x, y +
height / 2 -
track.getSize().y / 2);
```

```
track.setFillColor(sf::Color(
200, 200, 200));
```

```
knob.setSize(sf::Vector2f(h
eight, height));
```

```
knob.setOrigin(knob.getSize
().x / 2, knob.getSize().y
/ 2);
```

```
knob.setPosition(x,
y + height / 2);
```

```
knob.setFillColor(sf::Color
(100, 100, 100));
```

```
valueText.setFont(font);
```

```
valueText.setCharacterSize(
24);
```

```
valueText.setFillColor(sf::
Color::Black);
```

```
valueText.setPosition(x, y
- 30);
```

```
updateValueText();
```



```

    }

    void handleEvent(const
sf::Event& event) {

        if (event.type ==
sf::Event::MouseButtonPress
ed) {

            sf::Vector2f
mousePos(event.mouseButton.
x, event.mouseButton.y);

            if
(knob.getGlobalBounds().con
tains(mousePos)) {

                isDragging
= true;

            }

        }

        else if (event.type
==
sf::Event::MouseButtonRelea
sed) {

            isDragging =
false;

        }

        else if (event.type
== sf::Event::MouseMove) {

            if (isDragging)
{

                float newX
=
static_cast<float>(event.mo
useMove.x);

                newX =

```

```

std::max(track.getPosition(
).x, std::min(newX,
track.getPosition().x +
track.getSize().x));

knob.setPosition(newX,
knob.getPosition().y);

currentValue =
static_cast<int>(minValue +
(maxValue - minValue) *
((newX -
track.getPosition().x) /
track.getSize().x));

updateValueText();

    }

}

}

void
draw(sf::RenderWindow&
window) const {

    window.draw(track);

    window.draw(knob);

window.draw(valueText);

}

int getValue() const {

    return
currentValue;

}

void setValue(int

```

```

value) {

    currentValue =
std::max(minValue,
std::min(maxValue, value));

    float newX =
track.getPosition().x +
track.getSize().x *
((currentValue - minValue)
/
static_cast<float>(maxValue
- minValue));

knob.setPosition(newX,
knob.getPosition().y);

    updateValueText();

}

private:

    void updateValueText()
{

std::vector<int> data;

clock.restart();

parseNumbersFromFile("../De
pendencies/FILES/UnsortedSe
t2.txt", data);

        else if
(currentMenu ==
MenuConstants::File1) {

            for (int i = 0;
i < 4; ++i) {

window.draw(text);

```

```

window.draw(OtherMenu1[i]);

        }

    }

    else if
(currentMenu ==
MenuConstants::File2) {

        for (int i = 0;
i < 4; ++i) {

window.draw(text);

window.draw(OtherMenu2[i]);

        }

    }

    window.display();

}

return 0;

```