**Содержание**

[Введение 2](#_Toc202391292)

[Постановка задачи 3](#_Toc202391293)

[Выбор решения 4](#_Toc202391294)

[Описание алгоритма 6](#_Toc202391295)

[Преимущества и недостатки алгоритма 7](#_Toc202391296)

[Типичные сценарии применения 8](#_Toc202391297)

[Описание программы 9](#_Toc202391298)

[Схемы программы 10](#_Toc202391299)

[Блок схема программы 10](#_Toc202391300)

[Блок-схема алгоритма сортировки 11](#_Toc202391301)

[Разработка и реализация программного модуля 12](#_Toc202391302)

[Отладка 14](#_Toc202391303)

[Совместная разработка 15](#_Toc202391304)

[Заключение 19](#_Toc202391305)

[Список используемой литературы 20](#_Toc202391306)

[Приложение А. Результаты работы программы 21](#_Toc202391307)

[Приложение B Листинг программы 22](#_Toc202391308)

Введение

В современном программировании обработка больших массивов данных требует эффективных алгоритмов, одним из которых является быстрая сортировка (QuickSort). Этот алгоритм, разработанный Тони Хоаром в 1959 году, стал ключевым инструментом для упорядочивания данных благодаря своей скорости и универсальности. QuickSort широко применяется в различных областях: от системного программирования до анализа данных и разработки приложений.

Быстрая сортировка относится к алгоритмам типа «разделяй и властвуй». Она структурирует процесс сортировки, выбирая опорный элемент (pivot) и разделяя массив на две части: элементы, меньшие или равные опорному, и элементы, большие него. Затем алгоритм рекурсивно применяется к полученным подмассивам, что позволяет эффективно упорядочивать данные. Средняя временная сложность QuickSort составляет O(n log n), что делает его одним из самых быстрых алгоритмов сортировки.

Особую роль быстрая сортировка играет в задачах, где важны производительность и оптимизация памяти. Алгоритм выполняется "на месте", минимизируя дополнительные затраты ресурсов, что делает его предпочтительным для работы с большими наборами данных. Благодаря простоте реализации и гибкости, QuickSort используется в стандартных библиотеках многих языков программирования, таких как C++ и Python.

В рамках данной практической работы была разработана программа, реализующая алгоритм быстрой сортировки. Программа позволяет вводить массив данных, выполнять его сортировку по заданным критериям и выводить упорядоченный результат. Реализация демонстрирует основные принципы работы QuickSort и его преимущества в обработке данных, обеспечивая наглядное представление алгоритма в действии.

Постановка задачи

Поставленная задача: необходимо заполнить массив из n-ого количества элементов случайными числами, записать данные элементы в дельный файл. После этого выполнить сортировку вставками над данными, находящимися в массиве, записать отсортированные данные в другой файл, посчитать время выполнения и количество перестановок значений массива при сортировке.

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

Оформить отчет по проведенной практике.

Выбор решения

Для написания данной программы будет использован язык программирования С++. Этот язык является распространённым языком программирования, который отличается высокой производительностью, гибкостью и широкими возможностями для реализации сложных программных решений.

В качестве среды разработки была выбрана среда Microsoft Visual Studio (Рисунок 1). Это программная среда по разработке приложений для OS Windows, как консольных, так и с графическим интерфейсом.

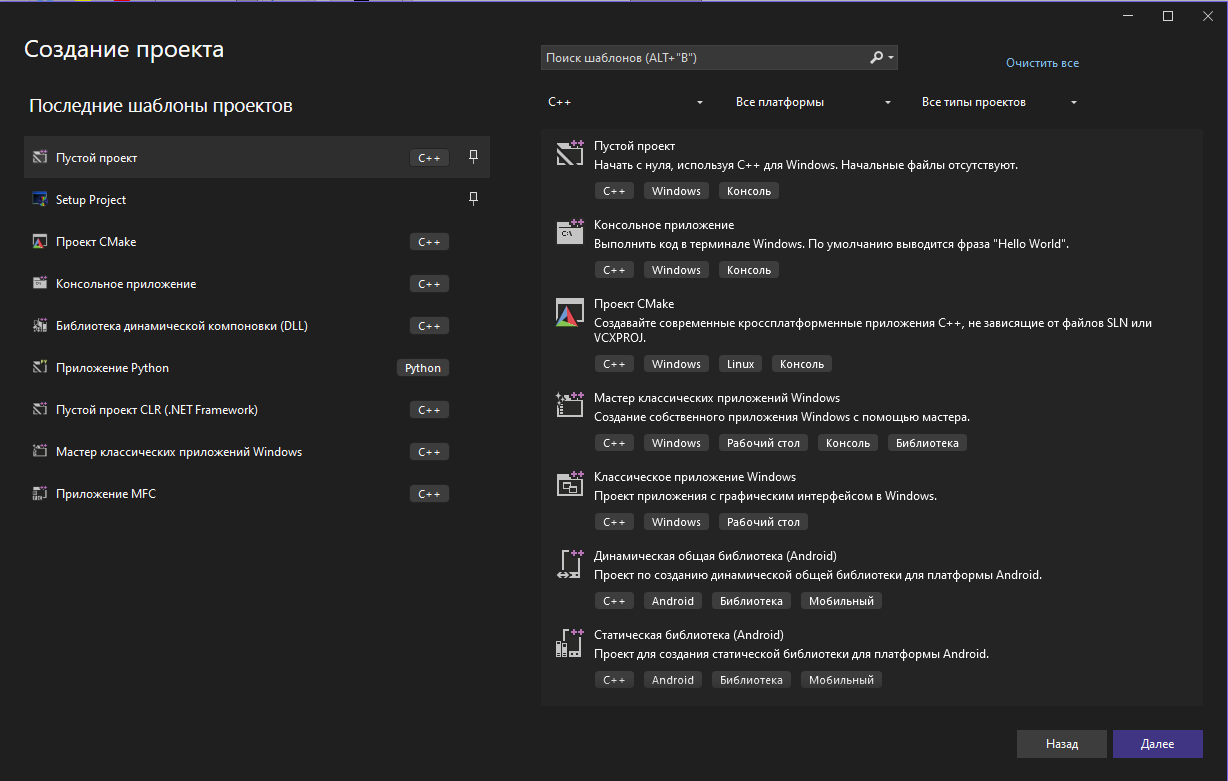


Рисунок 1 - Среда разработки MCVS.

Для совместной разработки был использован GitHub (Рисунок 2). GitHub - сервис для управления личными и командными проектами. В основе GitHub-Project лежит недельный планер и канбан-методология: доски, колонки и так далее.

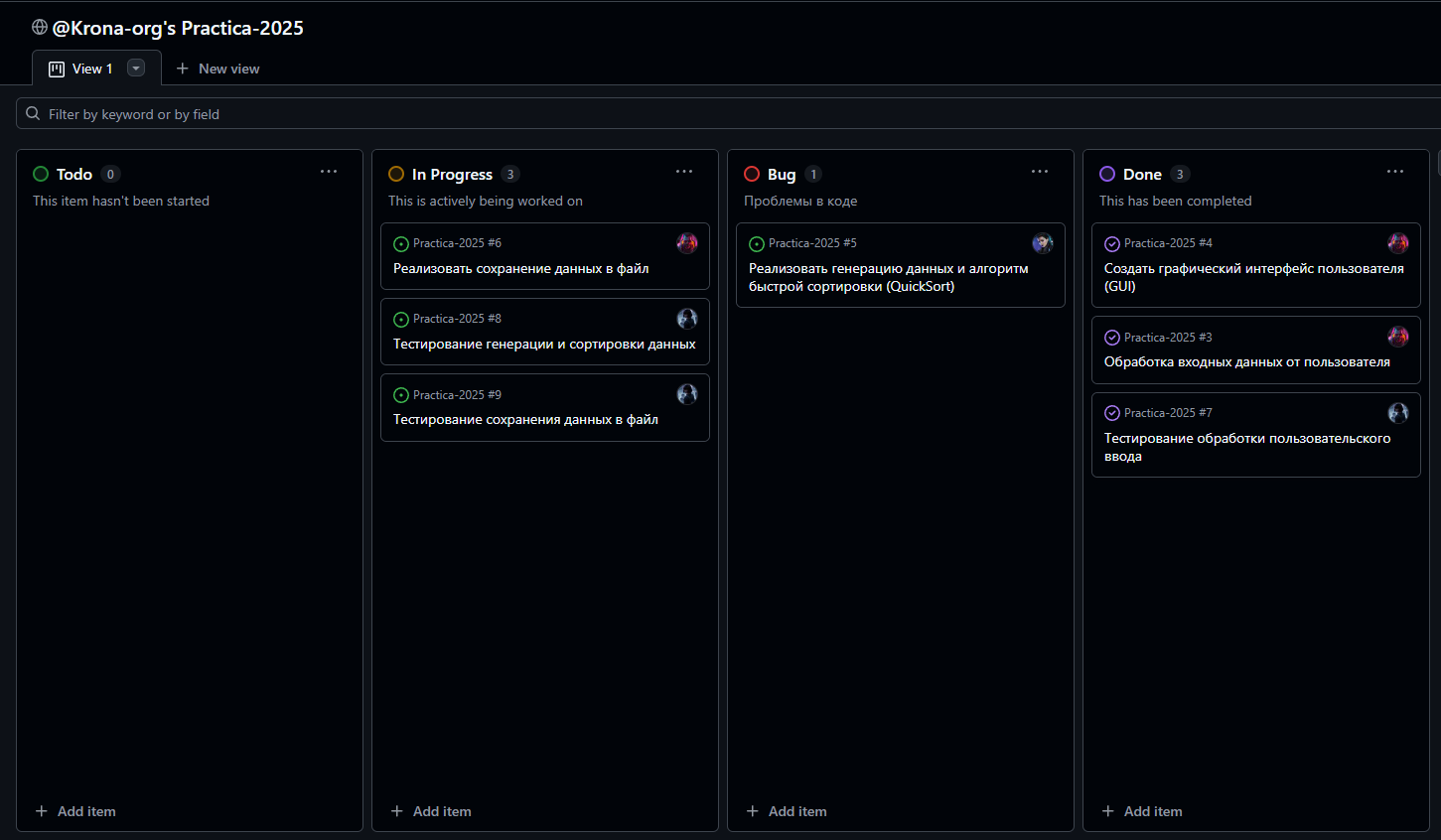


Рисунок 2 - Канбан-доска GitHub

Описание алгоритма

Быстрая сортировка (англ. QuickSort) – один из наиболее эффективных алгоритмов общего назначения для сортировки массивов. Метод был предложен британским ученым Чарльзом Хоаром в 1960 году, когда он работал в СССР и сталкивался с проблемами обработки больших объёмов данных. Благодаря своей рекурсивной природе и удачному принципу разбиения массива, алгоритм широко применяться в самых различных областях информатики.

Суть быстрой сортировки заключается в методе «разделяй и властвуй». Алгоритм выбирает один из элементов массива в качестве опорного, после чего переставляет остальные элементы так, чтобы слева от опорного оказались все значения меньше него, а справа – больше. Затем процесс рекурсивно повторяется для каждой из получившихся частей, пока массив полностью не отсортируется.

**Ключевые этапы алгоритма:**

* **Выбор опорного элемента**. В простых реализациях обычно выбирается первый, последний или средний элемент массива. Для повышения эффективности в худших случаях применяются методы случайного выбора или медианы трёх.
* **Разделение массива**. Элементы перегруппировываются так, чтобы значения меньше опорного оказались в одой части массива, а большие – в другой.
* **Рекурсивная сортировка**. Каждая из частей сортируется м же методом, пока не достигается базовый случай (подмассив из одного или нуля элементов).

На практике часто используют упрощённый подход, при котором массив делится не на три, а на две части – например, на элементы «меньшие опорного и «не меньшие». Такой вариант ускоряет процесс сортировки.

Преимущества и недостатки алгоритма

Как и любой другой алгоритм, быстрая сортировка имеет не только преимущества, но и определенные ограничения, которые необходимо учитывать при её применении на практике.

**К преимуществам быстрой сортировки можно отнести:**

* **Высокая средняя производительность**. В большинстве случаев алгоритм работает за время , что делает его одним из самых быстрых среди алгоритмов сортировки на сравнениях.
* **Эффективное использование памяти**. Базовая реализация быстрой сортировки не требует дополнительной памяти для хранения копий массива, в отличии от сортировки слиянием.
* **Простота реализации**. Несмотря на высокую эффективность, алгоритм легко реализуется, особенно в рекурсивной форме.
* **Адаптация к многопоточности**. Части массива после разбиения могут обрабатываться параллельно, что позволяет реализовать высокопроизводительную многопоточную сортировку.

**К недостаткам можно отнести:**

* **Худший случай – квадратичная сложность**. При неудачном выборе опорного элемента (например, если массив уже отсортирован) время работы алгоритма может возрасти до . Это делает его уязвимым в теоретическом плане.
* **Использование стека вызовов**. Рекурсивная реализация требует памяти для стека. При глубокой рекурсии возможно переполнение стека.
* **Сложность защиты от худших случаев**. Для надёжной работы в худших случаях приходится реализовывать улучшения: случайный выбор опорного элемента, использование медианы трех или переключение на другой алгоритм при малых подмассивах (например, на сортировку вставками).

Типичные сценарии применения

Алгоритм быстрой сортировки широко используется для обработки и упорядочивания больших объёмов данных благодаря своей высокой эффективности и хорошей средней производительности. Он часто применяется в следующих ситуациях:

* **Сортировка** **массивов** и списков в системном программировании особенно когда важна скорость выполнения.
* **Поисковые системы**: при ранжировании результатов поиска или индексации данных.
* **Графика и визуализация**: сортировка объектов по глубине (Z-сортировка) для корректного отображения на экране.
* **Научные вычисления**: когда необходимо упорядочить большие массивы чисел или структур.
* **Алгоритмы машинного обучения и анализа данных**, где требуется предварительная сортировка данных, например, при использовании алгоритмов, зависящих от порядка значений.

Благодаря простоте и высокой скорости работы на случайных данных, быстрая сортировка остается одним из самых популярных алгоритмов сортировки в программной инженерии.

Описание программы

При запуске программы открывается окно с набором полей и кнопок (Приложение А, Рисунок А.1):

* поле для ввода количества элементов массива
* поля для ввода нижней и верхней границы значений
* флажок «Уникальные элементы», при активации которого генерируются неповторяющиеся значения
* поля для ввода имен файлов: входного файла для чтения и файла для результата
* кнопки: «Генерировать и записать», «Прочитать и отсортировать», «Выход».

Пользователь может ввести необходимые значения и нажать кнопку «Генерировать и записать». В этом случае создается массив из случайных чисел в пределах указанных границ. Если установлен флажок уникальности, то значения будут неповторяющимися. Сгенерированный массив записывается в файл *input.txt* или иной, заданный пользователем.

При нажатии кнопки «Прочитать и отсортировать» происходит считывание чисел из указанного файла. далее выполняется алгоритм быстрой сортировки с подсчетом количества перестановок. После завершения сортировки, результат сохраняется в выходной файл sorted.txt (ли указанный вручную), а также пользователю отображается сообщение с количеством перестановок и временем выполнения сортировки в секундах.

Кнопка «Выход» завершает работу программы.

Схемы программы

Блок схема программы

Ниже, на рисунке 3 представлена основная логика работы программы.

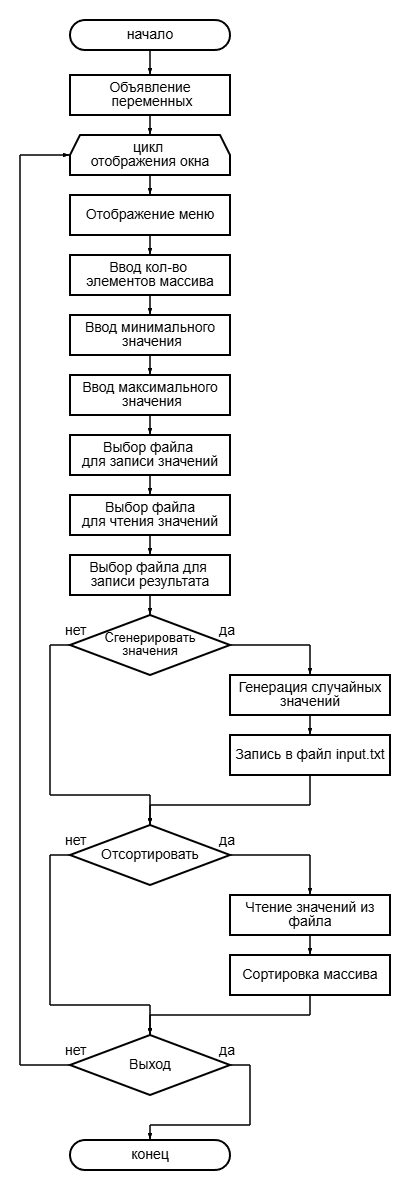


Рисунок 3 – Блок схема программы.

Блок-схема алгоритма сортировки

На рисунке 4 отображена логика работы алгоритма.

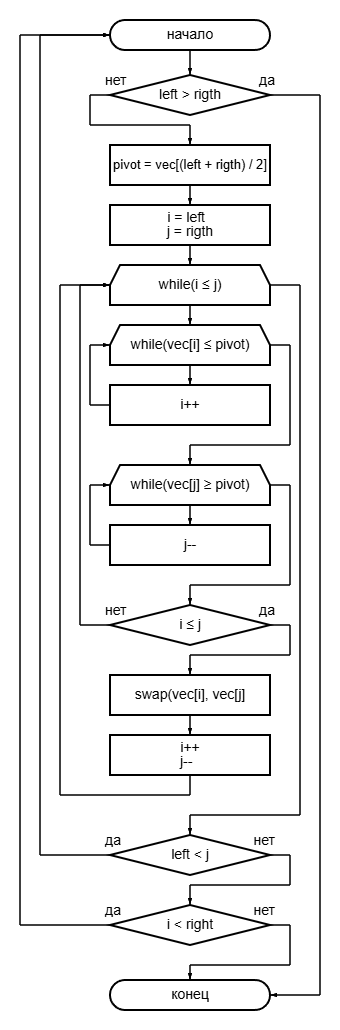


Рисунок 4 – Блок схема алгоритма быстрой сортировки.

Разработка и реализация программного модуля

В рамках практики я разработал графический интерфейс программы на основе WinAPI, а также реализовал обработку пользовательского ввода и сохранение результатов работы в файл. Программа позволяет пользователю задать параметры генерации массива, выбрать режим работы и указать файлы ввода/вывода через удобные элементы управления.

Графический интерфейс построен с использованием стандартной функции CreateWindowW, включающей поля ввода (EDIT), статические подписи (STATIC) и кнопки (BUTTON) . Например, создание поля ввода для количества элементов массива реализовано следующим образом(Рисунок 5):

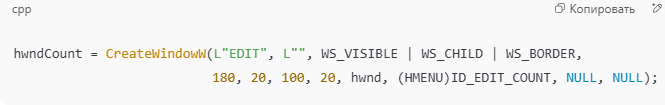


Рисунок 5 – Создание поля ввода.

Аналогичным образом добавлены и другие элементы: чекбокс для уникальности, поля для ввода диапазона, а также кнопки управления (Рисунок 6):

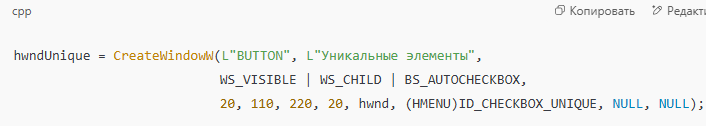


Рисунок 6 – Чекбокс для уникальных элементов.

Для считывания значений из полей используется вспомогательная функция GetEditText, которая возвращает текст из элемента управления (Рисунок 7):

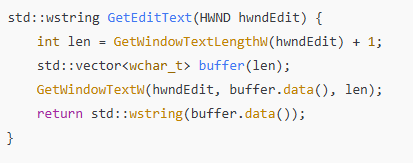


Рисунок 7 – Функция считывания значений из полей ввода.

По завершении операций (например, генерации массива или сортировки) данные сохраняются в файл, имя которого задаётся пользователем. Для записи используется std::wofstream с установленной кодировкой Windows-1251 (Рисунок 8):

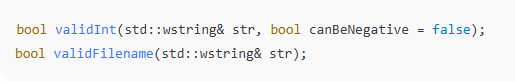


Рисунок 8 – Функции проверки

По завершении операций (например, генерации массива или сортировки) данные сохраняются в файл, имя которого задаётся пользователем. Для записи используется std::wofstream с установленной кодировкой Windows-1251 (Рисунок 9, 10):



Рисунок 9 – открытие файла для записи



Рисунок 10 – Запись в файл

Отладка

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

Для отладки программы использовались точки остановки и пошаговое выполнение кода программы, анализ содержимого локальных перемен.

Точки остановок - позволяют прервать выполнение программы для вызова отладчика. Отладчик является инструментом для исследования программы, так же он необходим для поиска и устранения ошибок в коде.

Иногда выявить ошибку сразу невозможно, поэтому использовался приём поэтапного анализа, в котором программа разбивалась на участки и каждый из них проверялся отдельно. Такой подход помогает быстрее локализовать проблему, особенно если её симптомы проявляются далеко от истинной причины.

Отладка проводилась не только после написания кода, но и по ходу разработки. Такой подход позволяет оперативно находить и устранять ошибки, не откладывая их на поздний этап. После завершения программы было проведено финальное тестирование и устранены все обнаруженные недочёты.

Совместная разработка

Для удобства совместной разработки был использован сервис GitHub.

С помощью встроенной доски Project Board были определены основные задачи проекта, распределены роли участников и назначены приоритеты задачам (Рисунок 6).

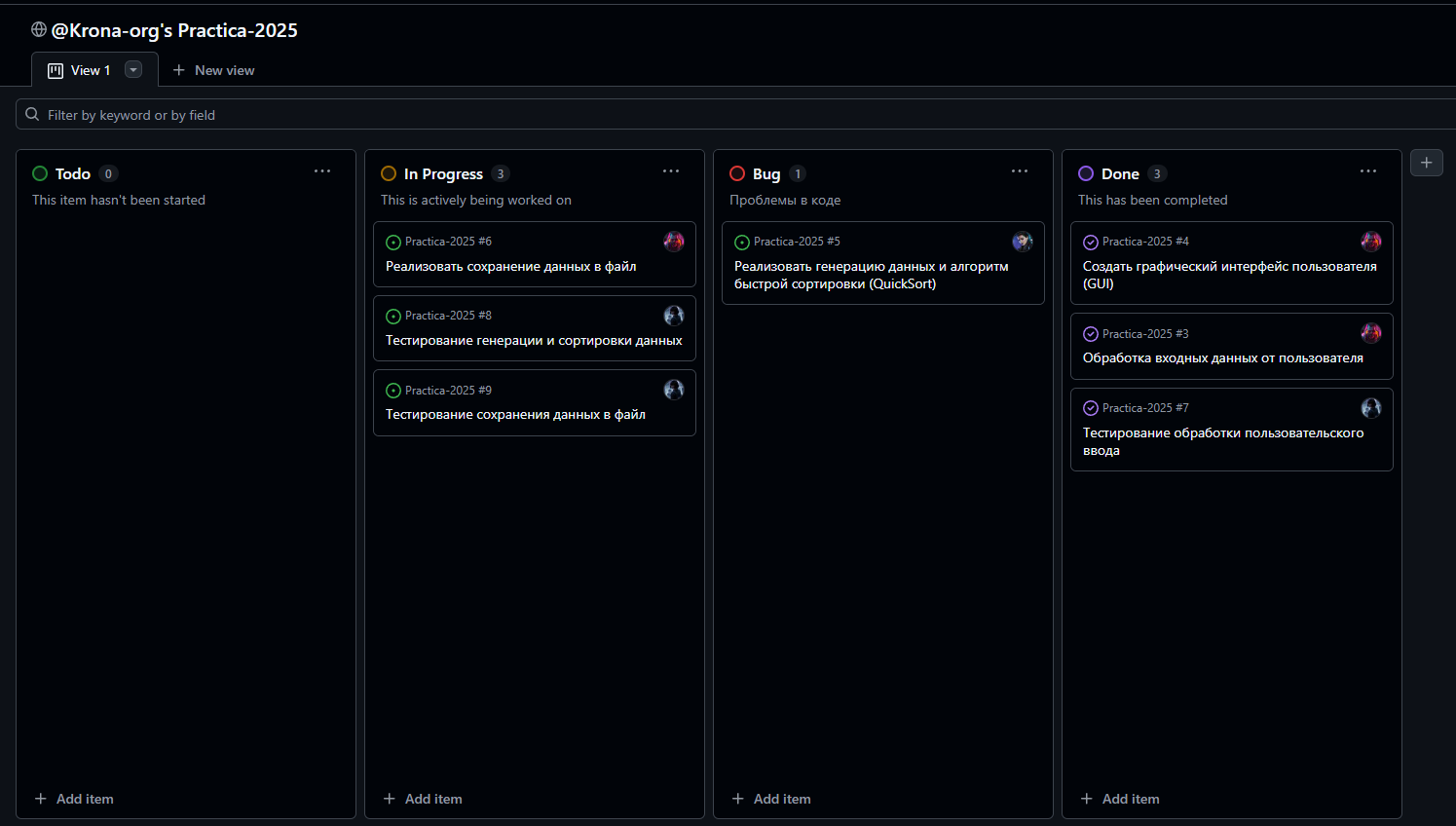


Рисунок 8 – Определение задач проекта

Разделили роли, назначили исполнителей (Рисунок 7).

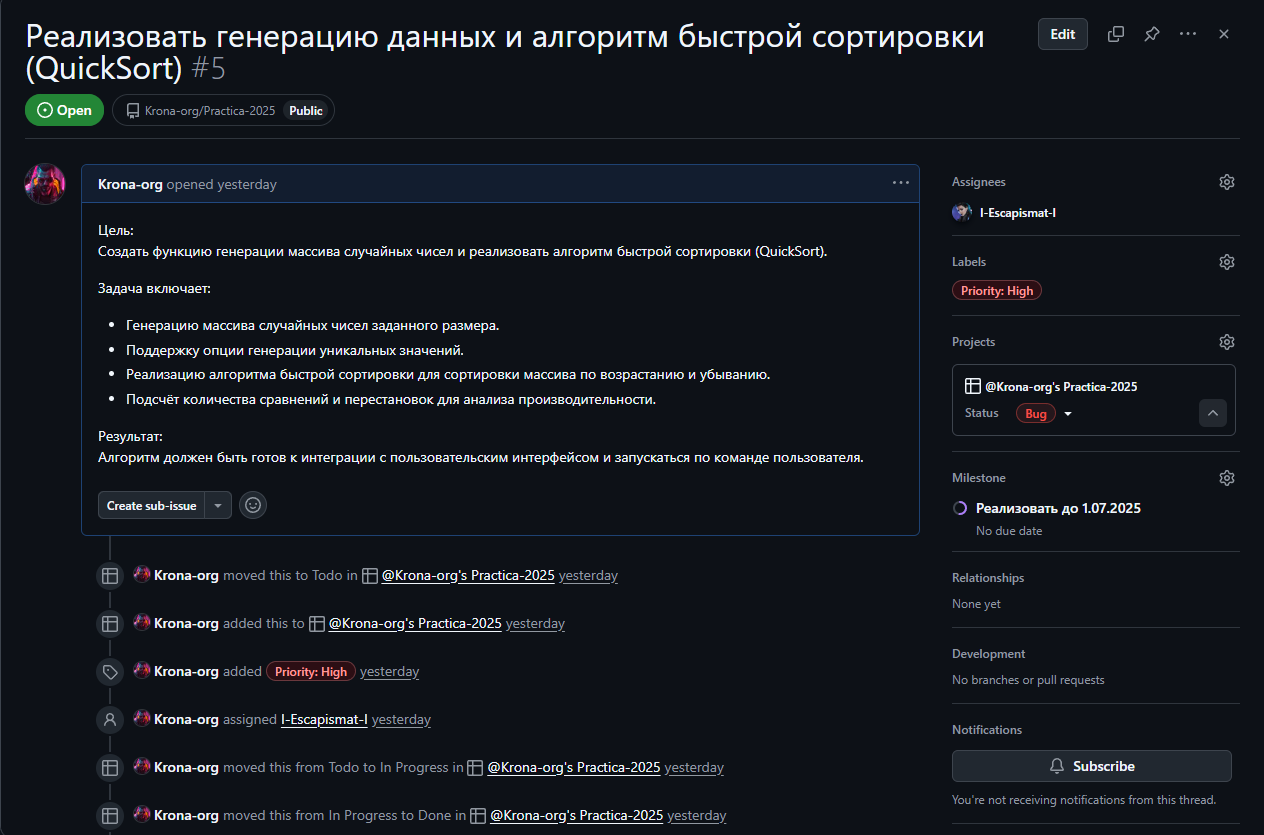


Рисунок 9 – Распределение задач проекта.

Обсуждали выполнение задачи на канбан-доске (Рисунок 8).

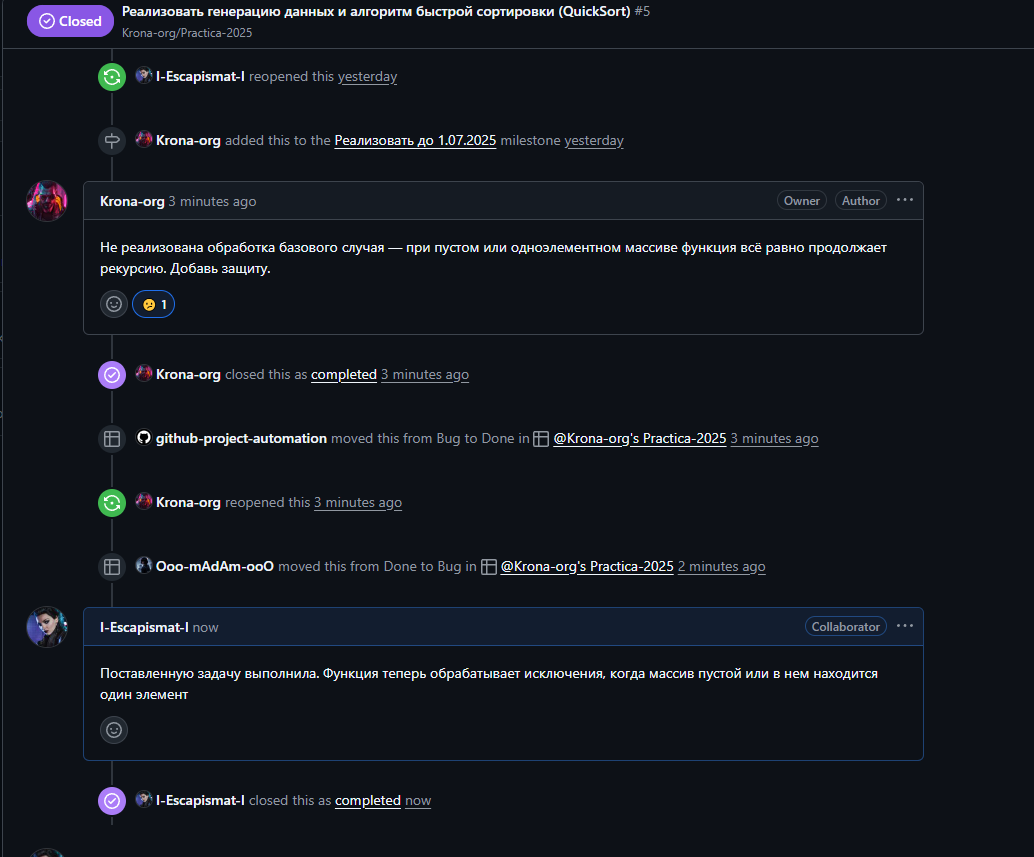


Рисунок 10 – Обсуждение задач проекта.

Корректировали статус задач, по мере их выполнения (Рисунок 9).

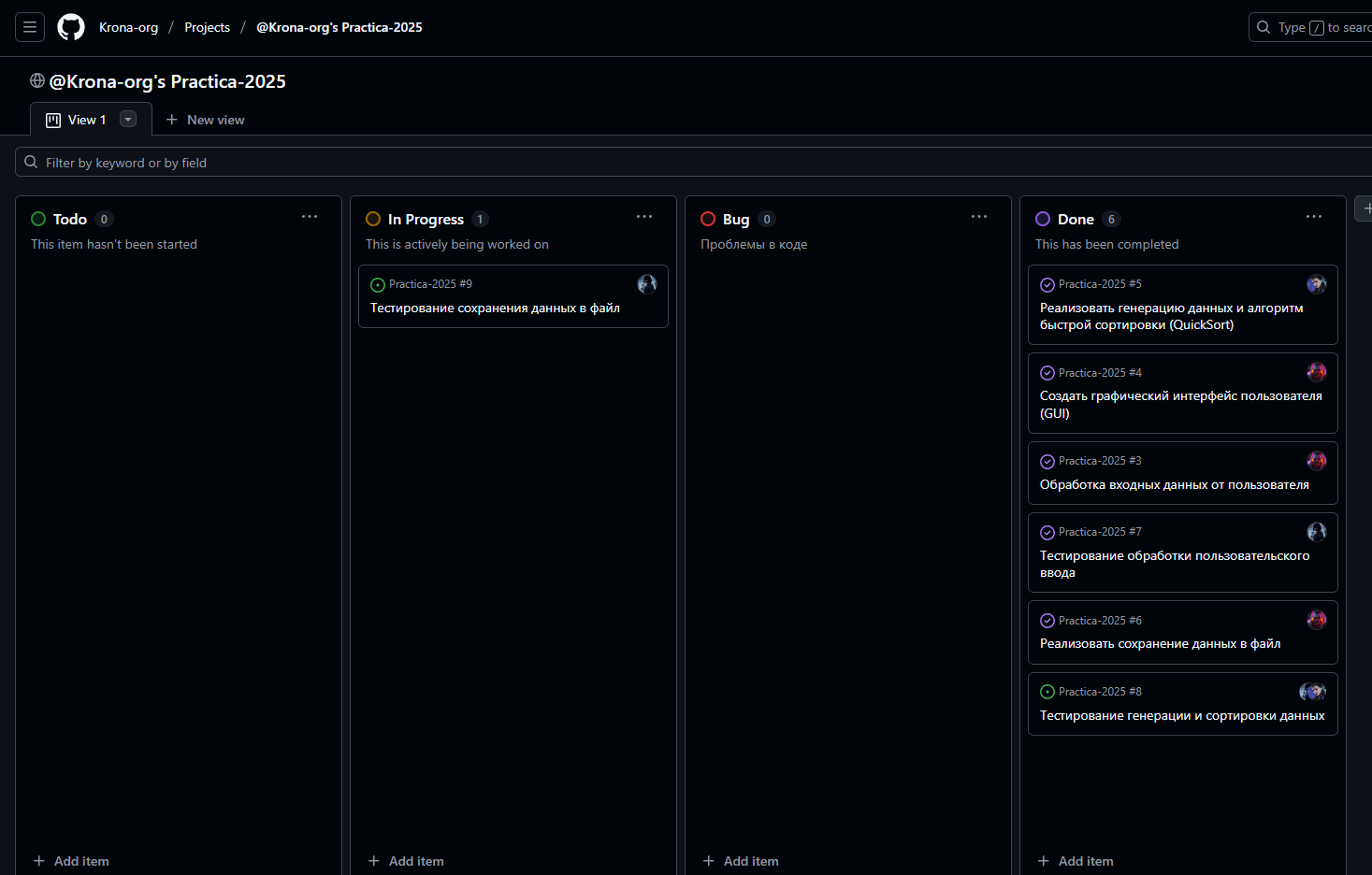


Рисунок 11 – Изменение статуса задачи.

Данная программа была загружена на компьютер, используя Git Bash и

прописав команду git clone <ссылка>.

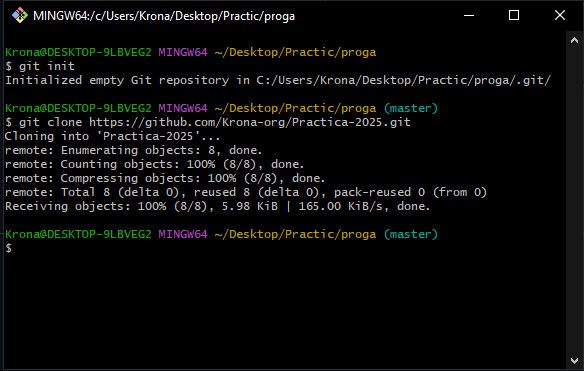


Рисунок 12 – Загрузка проекта с удаленного репозитория.

Мной был добавлен алгоритм быстрой сортировки, функция заполнения алгоритма случайными числами, а также подсчет количества перестановок.

Я загрузила обновлённый код программы на удаленный репозиторий GitHub.

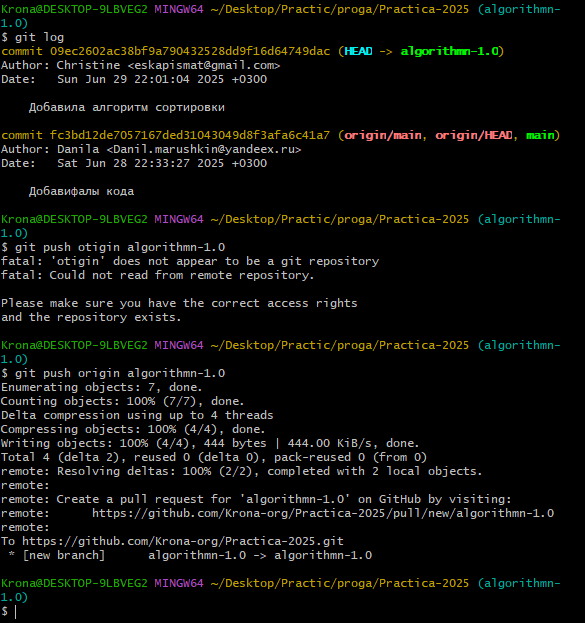


Рисунок 13 – Загрузка изменённой программы на GitHub

Ссылка на удаленный репозиторий:

<https://github.com/Krona-org/Practica-2025.git>

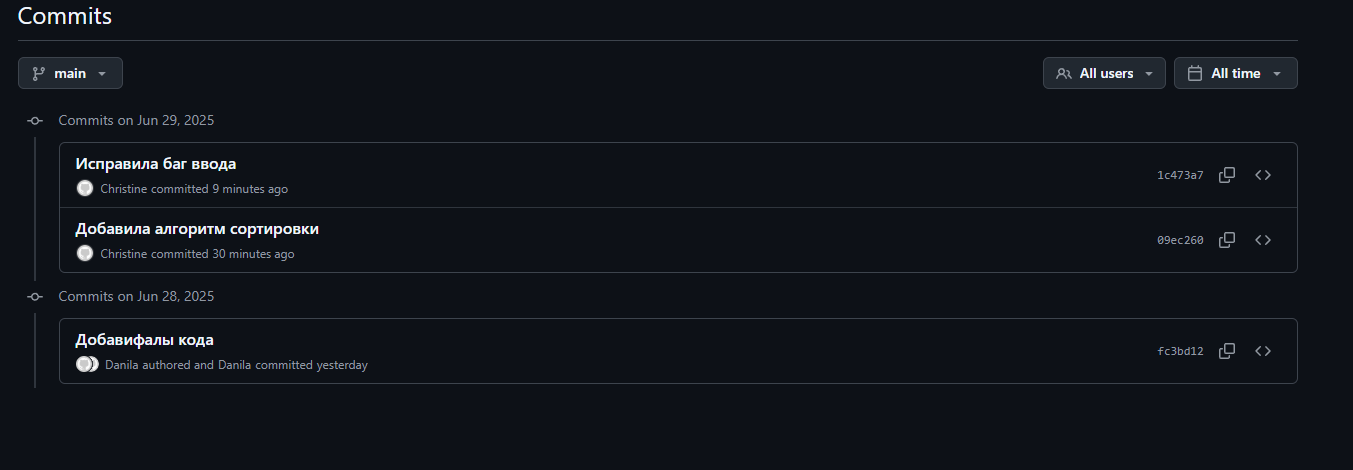


Рисунок 14 – История коммитов ветки main.

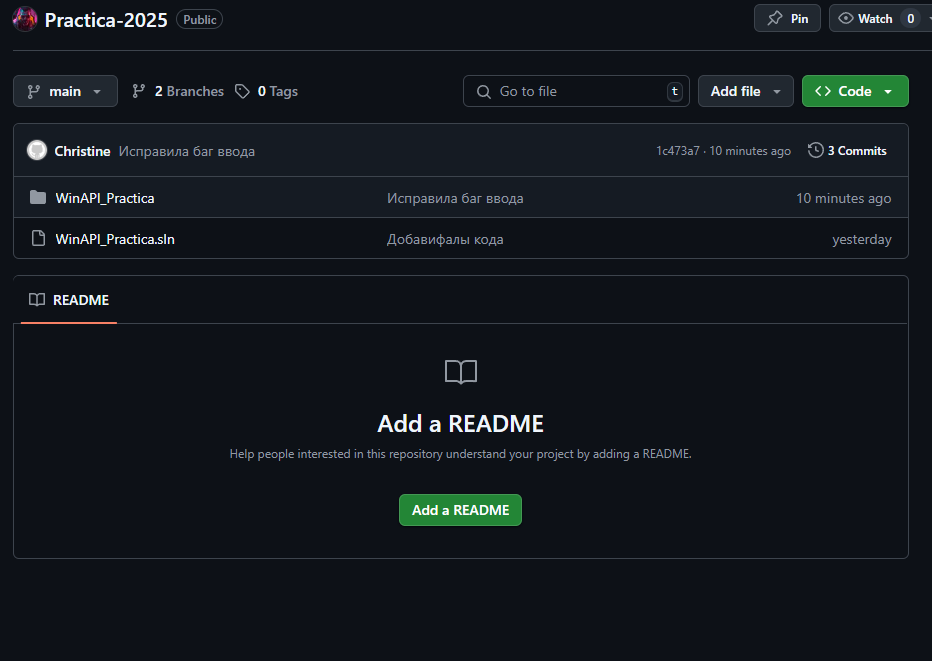


Рисунок 15 - Ветка main

Заключение

В ходе выполнения данной практической работы были получены навыки совместной работы над проектом с использованием сервиса GitHub, а также навыки работы с Git Bash.

Был изучен и реализован алгоритм сортировки вставками. Мною был написан код, позволяющий подсчитывать количество перестановок элементов массива во время сортировки, а также измерять время работы программы в секундах.

Программа прошла тестирование на различных наборах данных, была проведена её отладка и анализ результатов.

В рамках работы был также реализован графический интерфейс, обеспечивающий удобное взаимодействие с пользователем и наглядное отображение результатов сортировки.

В процессе выполнения задания были улучшены базовые навыки программирования на языке C++, а также навыки отладки, тестирования программ и работы с графическим выводом данных.

В дальнейшем программу можно доработать, добавив поддержку других алгоритмов сортировки, расширив возможности визуализации и оптимизировав производительность для работы с большими массивами данных.

Список используемой литературы

1. YouTube канал «SimpleCode», «Основы С++. Программирование для начинающих»

<https://www.youtube.com/@SimpleCode>

2. Роберт Мартин. «Чистый код: создание, анализ и рефакторинг». 2019 г.

3. Б. Страуструп. «Программирование. Принципы и практика с использованием C++»

4. Скотт Мейерс. «Эффективный и современный C++»

5. **Документация языка C++ от Microsoft (MSVC)** <https://learn.microsoft.com/cpp>

Приложение А. Результаты работы программы

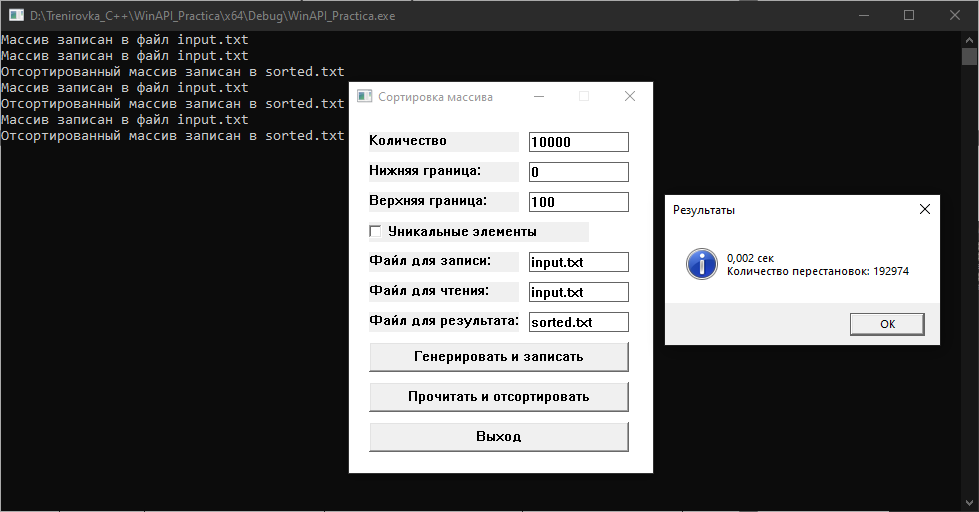


Рисунок А.1 – Отображение окна программы

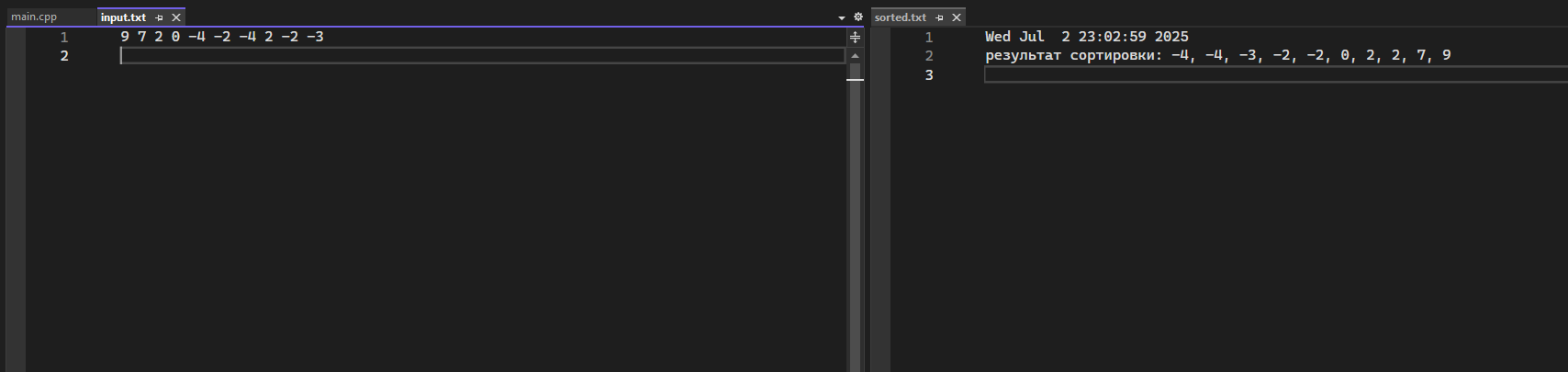


Рисунок А.2 – Результат работы программы

Приложение B Листинг программы

#include <windows.h>

#include <iostream>

#include <vector>

#include <random>

#include <chrono>

#include <fstream>

#include <sstream>

#include <ctime>

#include <iomanip>

#include <locale>

static HWND hwndCount, hwndMin, hwndMax, hwndUnique, hwndInputFile, hwndReadFile, hwndOutputFile;

static std::vector<int> vec, before;

#define ID\_EDIT\_COUNT 2001

#define ID\_EDIT\_MIN 2002

#define ID\_EDIT\_MAX 2003

#define ID\_CHECKBOX\_UNIQUE 3001

#define ID\_EDIT\_INPUT\_FILE 2004

#define ID\_EDIT\_READ\_FILE 2005

#define ID\_EDIT\_OUTPUT\_FILE 2006

#define ID\_BUTTON\_GENERATE\_WRITE 1004

#define ID\_BUTTON\_READ\_SORT 1005

#define ID\_BUTTON\_EXIT 1006

std::wstring GetEditText(HWND hwndEdit) {

int len = GetWindowTextLengthW(hwndEdit) + 1;

std::vector<wchar\_t> buffer(len);

GetWindowTextW(hwndEdit, buffer.data(), len);

return std::wstring(buffer.data());

}

bool validInt(std::wstring& str, bool canBeNegative = false) {

str.erase(0, str.find\_first\_not\_of(L" "));

str.erase(str.find\_last\_not\_of(L" ") + 1);

if (str.empty()) return false;

if (!canBeNegative && str[0] == L'-') return false;

for (wchar\_t c : str) {

if (c != L'-' && (c < L'0' || c > L'9')) return false;

}

return true;

}

bool validFilename(std::wstring& str) {

str.erase(0, str.find\_first\_not\_of(L" "));

str.erase(str.find\_last\_not\_of(L" ") + 1);

if (str.empty()) return false;

std::wstring forbidden = L"<>:\"/\\|?\*";

for (wchar\_t c : str) {

if (forbidden.find(c) != std::wstring::npos) return false;

}

return true;

}

void ShowVec(const std::vector<int>& v) {

for (int num : v) {

std::wcout << num << L" ";

}

std::wcout << std::endl;

}

bool GenerateUniqueVector(std::vector<int>& vec, int n) {

if (n <= 0) {

MessageBoxW(NULL, L"Количество элементов должно быть больше 0!", L"Ошибка", MB\_OK | MB\_ICONERROR);

return false;

}

vec.clear();

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

vec.push\_back(i);

}

for (int i = n - 1; i > 0; i--) {

int j = rand() % (i + 1);

std::swap(vec[i], vec[j]);

}

return true;

}

bool WriteVectorToFile(const std::vector<int>& vec, const std::wstring& filename, bool isOutputFile = false) {

std::wofstream out(filename);

out.imbue(std::locale(".1251"));

if (!out.is\_open()) return false;

if (isOutputFile) {

time\_t now = time(0);

char timeStr[26];

ctime\_s(timeStr, sizeof(timeStr), &now);

std::wstring wTimeStr;

for (int i = 0; timeStr[i]; i++) {

wTimeStr += static\_cast<wchar\_t>(timeStr[i]);

}

out << wTimeStr.substr(0, wTimeStr.length() - 1) << L"\n";

out << L"результат сортировки: ";

for (size\_t i = 0; i < vec.size(); i++) {

out << vec[i];

if (i < vec.size() - 1) out << L", ";

}

out << std::endl;

}

else {

for (size\_t i = 0; i < vec.size(); i++) {

out << vec[i];

if (i < vec.size() - 1) out << L" ";

}

out << std::endl;

}

out.close();

return true;

}

bool ReadVectorFromFile(std::vector<int>& vec, const std::wstring& filename) {

vec.clear();

std::wifstream in(filename);

in.imbue(std::locale(".1251"));

if (!in.is\_open()) return false;

int num;

while (in >> num) {

vec.push\_back(num);

}

in.close();

return !vec.empty();

}

void QuickSort(std::vector<int>& vec, int left, int right, int& swaps) {

if (left >= right) return;

int pivot = vec[(left + right) / 2];

int i = left;

int j = right;

while (i <= j) {

while (vec[i] < pivot) i++;

while (vec[j] > pivot) j--;

if (i <= j) {

std::swap(vec[i], vec[j]);

swaps++;

i++;

j--;

}

}

if (left < j) QuickSort(vec, left, j, swaps);

if (i < right) QuickSort(vec, i, right, swaps);

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {

static int count = 0;

switch (msg) {

case WM\_CREATE:

CreateWindowW(L"STATIC", L"Количество элементов:", WS\_VISIBLE | WS\_CHILD, 20, 20, 150, 20, hwnd, NULL, NULL, NULL);

hwndCount = CreateWindowW(L"EDIT", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 180, 20, 100, 20, hwnd, (HMENU)ID\_EDIT\_COUNT, NULL, NULL);

CreateWindowW(L"STATIC", L"Нижняя граница:", WS\_VISIBLE | WS\_CHILD, 20, 50, 150, 20, hwnd, NULL, NULL, NULL);

hwndMin = CreateWindowW(L"EDIT", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 180, 50, 100, 20, hwnd, (HMENU)ID\_EDIT\_MIN, NULL, NULL);

CreateWindowW(L"STATIC", L"Верхняя граница:", WS\_VISIBLE | WS\_CHILD, 20, 80, 150, 20, hwnd, NULL, NULL, NULL);

hwndMax = CreateWindowW(L"EDIT", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 180, 80, 100, 20, hwnd, (HMENU)ID\_EDIT\_MAX, NULL, NULL);

hwndUnique = CreateWindowW(L"BUTTON", L"Уникальные элементы", WS\_VISIBLE | WS\_CHILD | BS\_AUTOCHECKBOX, 20, 110, 220, 20, hwnd, (HMENU)ID\_CHECKBOX\_UNIQUE, NULL, NULL);

CreateWindowW(L"STATIC", L"Файл для записи:", WS\_VISIBLE | WS\_CHILD, 20, 140, 150, 20, hwnd, NULL, NULL, NULL);

hwndInputFile = CreateWindowW(L"EDIT", L"input.txt", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 180, 140, 100, 20, hwnd, (HMENU)ID\_EDIT\_INPUT\_FILE, NULL, NULL);

CreateWindowW(L"STATIC", L"Файл для чтения:", WS\_VISIBLE | WS\_CHILD, 20, 170, 150, 20, hwnd, NULL, NULL, NULL);

hwndReadFile = CreateWindowW(L"EDIT", L"input.txt", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 180, 170, 100, 20, hwnd, (HMENU)ID\_EDIT\_READ\_FILE, NULL, NULL);

CreateWindowW(L"STATIC", L"Файл для результата:", WS\_VISIBLE | WS\_CHILD, 20, 200, 150, 20, hwnd, NULL, NULL, NULL);

hwndOutputFile = CreateWindowW(L"EDIT", L"sorted.txt", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 180, 200, 100, 20, hwnd, (HMENU)ID\_EDIT\_OUTPUT\_FILE, NULL, NULL);

CreateWindowW(L"BUTTON", L"Генерировать и записать", WS\_VISIBLE | WS\_CHILD, 20, 230, 260, 30, hwnd, (HMENU)ID\_BUTTON\_GENERATE\_WRITE, NULL, NULL);

CreateWindowW(L"BUTTON", L"Прочитать и отсортировать", WS\_VISIBLE | WS\_CHILD, 20, 270, 260, 30, hwnd, (HMENU)ID\_BUTTON\_READ\_SORT, NULL, NULL);

CreateWindowW(L"BUTTON", L"Выход", WS\_VISIBLE | WS\_CHILD, 20, 310, 260, 30, hwnd, (HMENU)ID\_BUTTON\_EXIT, NULL, NULL);

break;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case ID\_BUTTON\_GENERATE\_WRITE: {

std::wstring nStr = GetEditText(hwndCount);

std::wstring minStr = GetEditText(hwndMin);

std::wstring maxStr = GetEditText(hwndMax);

if (!validInt(nStr) || !validInt(minStr, true) || !validInt(maxStr, true)) {

MessageBoxW(hwnd, L"Некорректный ввод чисел!", L"Ошибка", MB\_OK | MB\_ICONERROR);

break;

}

int n = std::stoi(nStr);

int MIN = std::stoi(minStr);

int MAX = std::stoi(maxStr);

if (n <= 0 || MAX <= MIN) {

MessageBoxW(hwnd, L"Количество > 0, MAX > MIN", L"Ошибка", MB\_OK | MB\_ICONERROR);

break;

}

std::wstring inputFile = GetEditText(hwndInputFile);

if (!validFilename(inputFile)) {

MessageBoxW(hwnd, L"Неверное имя файла!", L"Ошибка", MB\_OK | MB\_ICONERROR);

break;

}

vec.clear();

bool isUnique = (SendMessage(hwndUnique, BM\_GETCHECK, 0, 0) == BST\_CHECKED);

if (isUnique) {

// ЗАДАНИЕ: генерация уникального массива (алгоритмист)

if (!GenerateUniqueVector(vec, n)) break;

}

else {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dist(MIN, MAX);

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

vec.push\_back(dist(gen));

}

}

before = vec;

if (WriteVectorToFile(vec, inputFile)) {

std::wcout << L"Массив записан в файл " << inputFile << std::endl;

}

else {

MessageBoxW(hwnd, L"Ошибка записи в файл!", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

break;

}

case ID\_BUTTON\_READ\_SORT: {

std::wstring readFile = GetEditText(hwndReadFile);

std::wstring outputFile = GetEditText(hwndOutputFile);

if (!validFilename(readFile) || !validFilename(outputFile)) {

MessageBoxW(hwnd, L"Проверьте имена файлов!", L"Ошибка", MB\_OK | MB\_ICONERROR);

break;

}

vec.clear();

if (!ReadVectorFromFile(vec, readFile)) {

MessageBoxW(hwnd, L"Не удалось прочитать файл!", L"Ошибка", MB\_OK | MB\_ICONERROR);

break;

}

auto start = std::chrono::high\_resolution\_clock::now();

// ЗАДАНИЕ: сортировка массива быстрой сортировкой и подсчет перестановок (алгоритмист)

QuickSort(vec, 0, vec.size() - 1, count);

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::milli> duration = end - start;

if (WriteVectorToFile(vec, outputFile, true)) {

std::wcout << L"Отсортированный массив записан в " << outputFile << std::endl;

}

else {

MessageBoxW(hwnd, L"Ошибка записи в файл результата!", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

std::wstringstream ss;

ss.imbue(std::locale(std::locale::classic(), new comma\_decimal));

ss << std::fixed << std::setprecision(3) << (duration.count() / 1000.0); // в секундах

ss << L" сек\nКоличество перестановок: " << count;

MessageBoxW(hwnd, ss.str().c\_str(), L"Результаты", MB\_OK | MB\_ICONINFORMATION);

break;

}

case ID\_BUTTON\_EXIT:

PostQuitMessage(0);

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hwnd, msg, wParam, lParam);

}

return 0;

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow) {

srand(static\_cast<unsigned>(time(0)));

AllocConsole();

FILE\* f;

freopen\_s(&f, "CONOUT$", "w", stdout);

SetConsoleOutputCP(CP\_UTF8);

SetConsoleCP(CP\_UTF8);

std::wcout.imbue(std::locale("ru\_RU.UTF-8"));

WNDCLASSW wc = { 0 };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.lpszClassName = L"SortApp";

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

RegisterClassW(&wc);

HWND hwnd = CreateWindowW(L"SortApp", L"Сортировка массива",

WS\_OVERLAPPEDWINDOW & ~WS\_MAXIMIZEBOX & ~WS\_SIZEBOX,

CW\_USEDEFAULT, CW\_USEDEFAULT, 320, 400,

NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

MSG msg = { 0 };

while (GetMessageW(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessageW(&msg);

}

return 0;

}