Содержание

Введение

Соритровка данных – одна из базовых задач в программировании. Она играет ключевую роль при обработке информации и оптимизации вычислений. В данной работе рассматриваются два алгоритма: битонная сортировка и lsd-сортировка, отличающиеся высокой эффективностью и специфическими условиями

Основные приемущества данных алгоритмов:

- высокая скорость работы на больших массивах;
- параллелизуемость (для битонной сортировки);
- сохранение исходного порядка элементов, которые считаются равными по ключу сортировки.

Целью данной работы является изучение алгоритмов битонной и lsd-сортировки, сравнение их эффективности и выявление преимущества каждого из них в зависимости от типа данных и условий применения.

1 Аналитический обзор предметной области

В данном разделе будут проанализированы существующие алгоритмы сортировки, приведены их особенности, а также будут рассмотрены их достоинства и недостатки. Здесь же будут сформулированы конечные цели и задачи для выполняемой работы.

1.1 Обзор предметной области

Сортировка данных является одной из ключевых задач в области информатики и программирования, играя важную роль в оптимизации производительности программных систем. Среди множества существующих алгоритмов особое внимание привлекают специализированные методы, такие как битонная сортировка и LSD-сортировка (Least Significant Digit), благодаря своей эффективности в определённых контекстах применения.

Битонная сортировка представляет собой один из вариантов параллельной сортировки, основанный на использовании битонных последовательностей — специальных массивов, в которых элементы сначала возрастают, а затем убывают, или наоборот. Алгоритм особенно эффективен в средах с поддержкой параллельных вычислений, таких как GPU или многопроцессорные системы, поскольку позволяет выполнять сортировку с высокой степенью одновременности [1].

Вариант поразрядной сортировки LSD выполняет стабильную сортировку по счету в списке для каждой цифры, начиная с наименее значивой (самой правой) цифры. Он работает за время O(wn), где n — размер ввода, а w — размер слова (количество цифр в самом большом числе для данного разряда) [2]

1.2 Обзор метода битонной сортировки

Битонная сортировка относится к классу детерминированных алгоритмов, работающих по принципу последовательных сравнений и обменов элементов по заранее заданной схеме. Её архитектура основана на сортировочных сетях, где порядок операций не зависит от входных данных, что обеспечивает стабильную производительность вне зависимости от начального распределения значений [1].

Отличительной чертой алгоритма является его модульная структура, благодаря которой его можно легко масштабировать и применять в аппаратной реализации. Это особенно актуально при проектировании систем на кристалле (SoC) и FPGA, где важна предсказуемость и постоянная глубина логики [1].

Благодаря своей упорядоченности и повторяемости, битонная сортировка часто используется как базовый строительный блок в разработке параллельных алгоритмов и демонстрационных примеров параллельной обработки данных [1].

1.3 Обзор метода LSD-сортировки

LSD-сортировка относится к категории разрядных сортировок, в основе которых лежит обработка элементов поразрядно — от младших к старшим позициям. В отличии от сравниетльных алгоритмов, таких как быстрая сортировка, LSD-сортировка не использует прямые равенения между значениями, что позволяет достигать линейной временной сложности при соблюдении определенных условий [1].

Одним из ключевых требований к применению алгоритма является одинаковая длина обрабатываемых записей (например, чисел с фиксированным количеством разрядов или строк одинаковой длины). Благодаря этому LSD-сортировка широко применяется в задачах обработки строк, чисел и других структурированных данных, где важна высокая скорость и устойчивость результата [1].

Алгоритм особенно эффективен в ситуациях, когда необходимо созранить порядок одинаковых элементов — он является стабильным, что делает его полезным в составе более сложных каскадных схем сортировки. LSD-сортировка может использовать, например, для предварительной обработки данных пере применением более ресурсоемких методов [1].

1.4 Постановка задачи

Целью работы является выявление приемуществ и недостатков этих сортировок, сравнение их эффективности при различных условиях, для упрощения выбора конкретной сортировки под конкретную задачу.

Разрабатываемое приложение должно содержать следующий основной функционал:

– удобный текстовый интерфес;

– возможность выбора сортировки;

– отображение списка файлов входа, содержащих входные параметры и выбор файла входа;

– отображение файлов выхода, в котоырые будет записан результат сортировки, а так же возможность выбирать файл для выхода;

– вывод времени выполнения алгоритма;

Для достижения поставленной цели необходимо решить следующие задачи:

– реализовать оба алгоритма;

– реализовать проверку на наличие в файле входа дефектов и его отсутствия;

– добавить функционал измерения времени выполнения алгоритма;

– разработать удобный и интуитивно понятный для пользователя интерфейс;

– протестировать приложение.

1.5 Выводы по главе

В данном разделе были рассмотрены особенности и принципы работы битонной и LSD-сортировки, проведен анализ их применимости и эффективнсти в различных вычислительных условиях.

Таким образом, на основе преведенного обзора были сформулированы задачи разработки приложения с реализацией указанных алгоритмов и соответствующим функционалом для анализа их производительности.

2 Алгоритмическое конструирование

В данном разделе рассматриваются основные алгоритмы работы разрабатываемого приложения: общий алгоритм работы приложения, алгоритм битонной сортировки, алгоритм lsd-сортировки, алгоритм проверки файлов.

2.1 Общий алгоритм работы

Общий алгоритм работы разрабатываемого приложения состоит из следующей последовательности действий:

– запуск консольного приложения;
– открытие вывод на экран приветсвия;
– выбор сортировки;
– выбор файла входа, откуда будет взят массив;
– выбор файла выхода, куда будет записан отсортированный массив;
– запуск сортировки;
– просмотр времени, которое заняла сортировка.

Общий алгоритм работы разрабатываемого приложения представлен на рисунке 2.1.

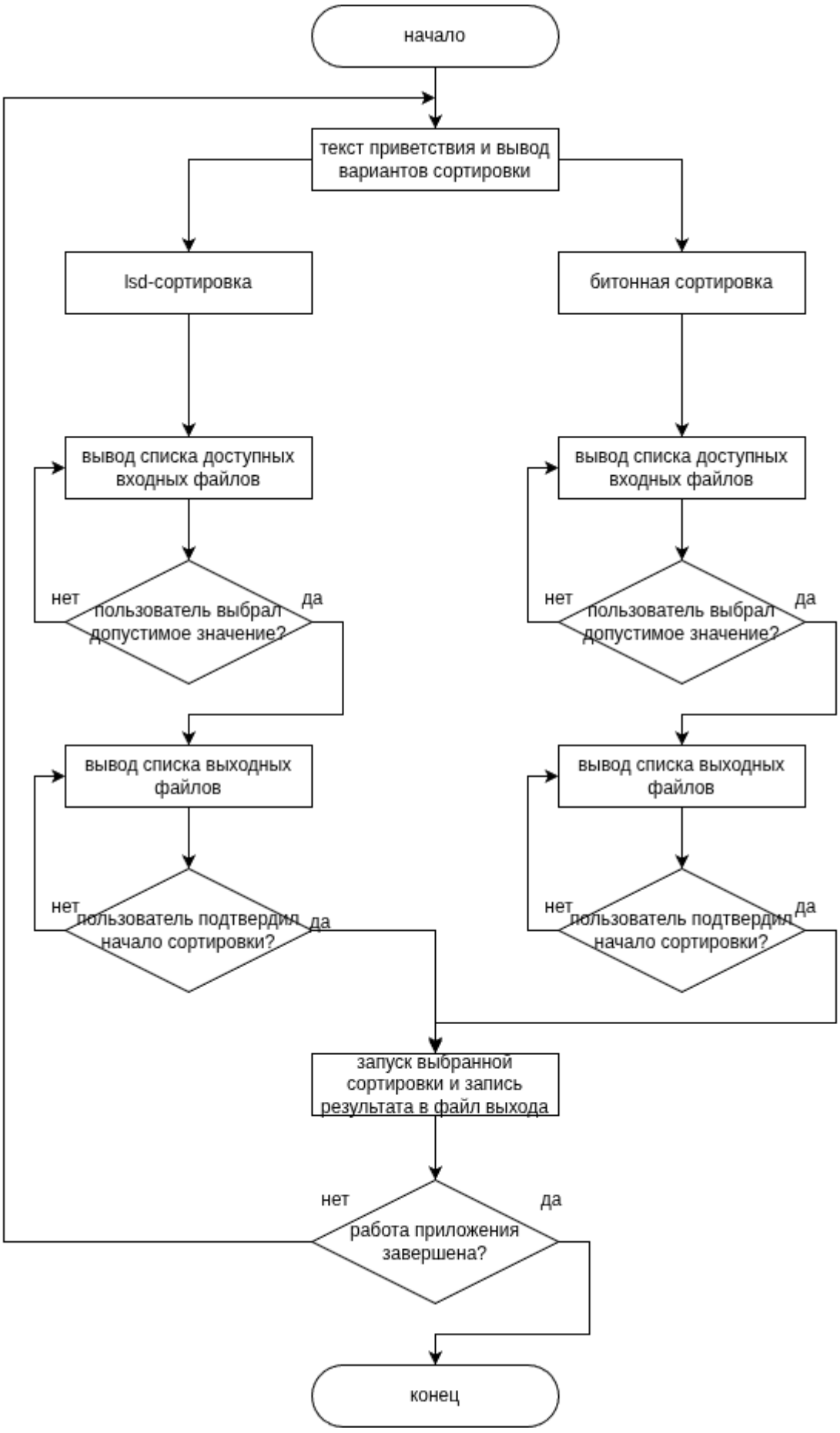

Рисунок 2.1 – Общий алгоритм работы веб-приложения

2.2 Алгоритм битонной сортировки

Алгоритм битонной сортировки выполняется при выборе этого алгоритма и после окончательного подтверждения выполнения. Алгоритм битонной сортировки состоит из создания битонных последовательностей который показан на рисунке 2.2. А так же на слиянии битонных последовательностей как это показано на рисунке 2.3.

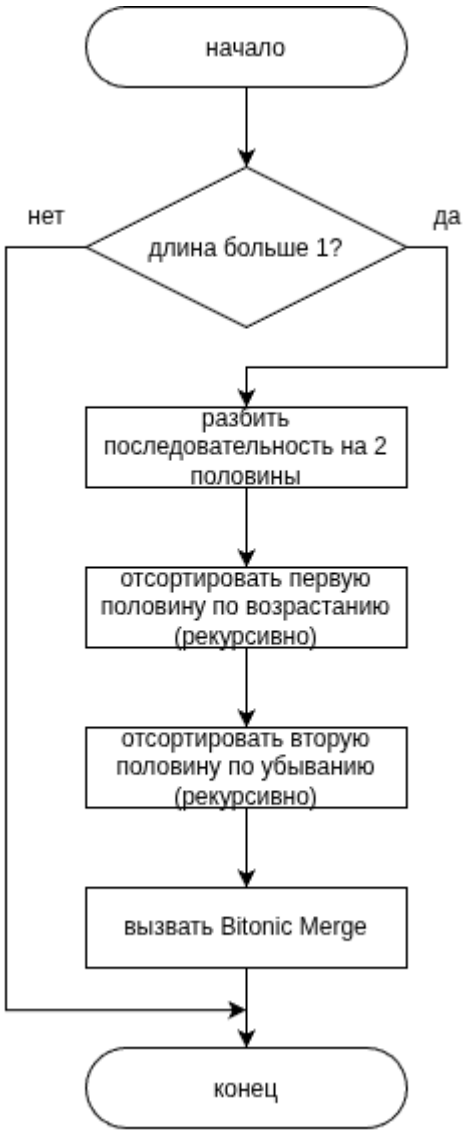

Рисунок 2.2 – Алгоритм создания битонной последовательности

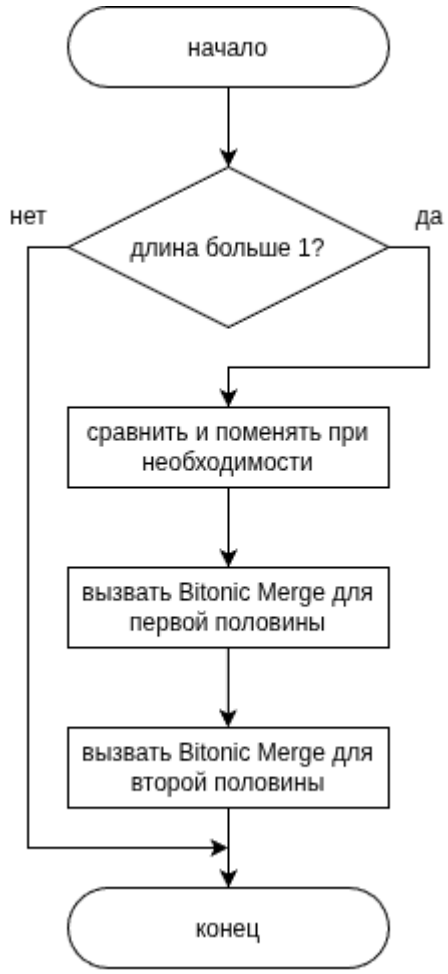

Рисунок 2.3 — Алгоритм слияния битонных последовательностей

2.3 Алгоритм lsd-сортировки

Алгоритм lsd-сортировки выполняется при выборе данного алгоритма и после окончательного подтверждения выполнения. Алгоритм lsd-сортировки показан на рисунке 2.4.

Рисунок 2.4 — Алгоритм lsd-сортировки

2.4 Выводы по главе

В данной главе было произведено алгоритмическое конструирование, то есть разработаны методы и процессы, основанные на алгоритмах и структурах данных, с целью достижения определенных результатов или решения задач. Этот подход позволяет систематизировать знания и опыт, а также обеспечивает эффективное решение сложных задач.

3 Программное конструирование

В данном разделе будут обоснованы выбор языка программирования и технологий, используемых для реализации программного средства и основания выбора среды программирования. Будут определены и описаны основные классы разрабатываемого приложения, структура проекта и структура базы данных.

3.1 Обоснование выбора средств разработки

Для реализации программного средства был выбран компилятор g++ и текстовый редактор Visual Studio Code.

Visual Studio Code - этот текстовый редактор, созданный Microsoft для операционных систем Windows, Linux и macOS, предназначен для удобной разработки любых приложений, под которые его настроит разработчик. Он обладает множеством плагинов, установив которые открывается доступ к необходимому для разработчика инструментарию. Редактор обеспечивает широкие возможности настройки, включая пользовательские темы, настройки сочетания клавиш и файлы конфигурации. Более того, он доступен бесплатно и разрабатывается как программное обеспечение с открытым исходным кодом, но сборки предоставляются под проприетарной лицензией.[3]

3.2 Описание программной реализации

Программа реализована на основе функциональной парадигмы программирования

Функция «main» является основной функцией приложения, в ней осуществляется выбор метода сортировки, выбор входного файл, выбор выходного файла и вызов всех необходимых функций.

В таблице 3.1 представлены реализованные функции.

Таблица 3.1 – Функции

Функция	Входные параметры	Описание	Возвращаемое значение
1	2	3	4
readNumbersFromFile	Имя входного файа, массив чисел	Записывает числа из файла в переданный вектор	Истина, если удалось прочитать файл. Ложь, если не удалось
writeNumbersToFile	Имя выходного файла, отсортированный массив чисел	Запись отсортированного массива чисел в файл	Истина, если получилось. Ложь, если не получилось
canReadFile	Имя файла	Проверяет, возможно ли прочитать файл	Истина, если возможно. Ложь, если невозможно
canWriteFile	Имя файла	Проверяет, возможно ли записать данные в файл	Истина, если возможно. Ложь, если невозможно
showWelcome	-	Выводит текст приветствия	-
chooseSortingMethod	-	Позволяет пользователю выбрать, какую сортировку использовать	Номер варианта выбора
chooseInputFile	-	Позволяет выбрать, из какого файла брать входные данные	Имя файла
chooseOutputFile	-	Позволяет выбрать, в какой файл записывать выходные данные	Имя файла
confirmAction	Название действия	Подтверждает действие	Истина, если пользователь подтвердил действие, иначе ложь
showResults	Количество обработынных чисел, время сортировки, файл выхода	Выводит информацию по окончании сортировки на экран	-
getYesNoExit	-	Принимает от пользователя выбор в виде числа от 1 до 3	Номер выбора
performSorting	Массив чисел, метод сортировки	Измеряет время сортировки а так же вызывает функцию выбранной сортировки	Время сортировки

Окончание таблицы 3.1

1	2	3	4
bitonicSort	Массив чисел	Вызывает вспомогательные функции для битонной сортировки и начинает ее	-
prepareForBitonicSort	Массив чисел	Подготавливает массив к сортировке	-
bitonicSortRecursive	Массив чисел, начальный индекс подмассива, количество элементов, направление сортировки	Формирует битонную последовательность	-
bitonicMerge	Массив чисел, начальный индекс подмассива, количество элементов, направление сортировки	Сливает два подмассива в один в нужном направлении	-
compareAndSwap	Массив чисел, индекс 1, индекс 2, направление сортировки	Сравнивает и при необходимости меняет местами элементы массива	-
lsdRadixSort	Массив чисел	Основная функция lsd-сортировки	-
countingSort	Массив чисел, разряд текущей итерации сортировки	Вспомогательная функция lsd-сортровки, сортирует числа по разрядам	-
getMax	Массив чисел	Нахождение максимального элемента массива чисел	Максимальный элемент массива чисел
main	-	Основная функция программы	0

3.3 Выводы по главе

В данной главе был обоснован выбор языка и среды программирования для программного средства, были выделены преимущества их выбора для данного приложения.

Были описаны основные функции программного средства, в описание которых входило: описание замкнутых функций и переменных, а также входные параметры, типы параметров и результата функции.

4 Демонстрация работы программного средства

В данном разделе продемонстрирована работа программного средства на основе скриншотов.

4.1 Описание контрольного примера

Для демонстрации работы программного средства рассматривается решение задачи сортировки целых чисел.

4.2 Описание процесса работы с программой

При запуске приложения на экране текст приветствия, предлагающий выбрать метод сортировки или выйти из приложения, показанный на рисунке 4.1. После выбора варианта сортировки будет предложено указать путь к файлу, из которого будут взяты входные данные, после необходимо будет подтвердить ввод, показано на рисунке 4.2. Далее нужно ввести путь для файла, в который будет записан результат работы программы, показано на рисунке 4.3. Далее необходимо подтвердить начало сортировки как это показано на рисунке 4.4.

После последнего подтверждения будет выведена информация о результатах сортировки: количество обработанных чисел, время и файл где находиться результат работы программы, показано на рисунке 4.5.

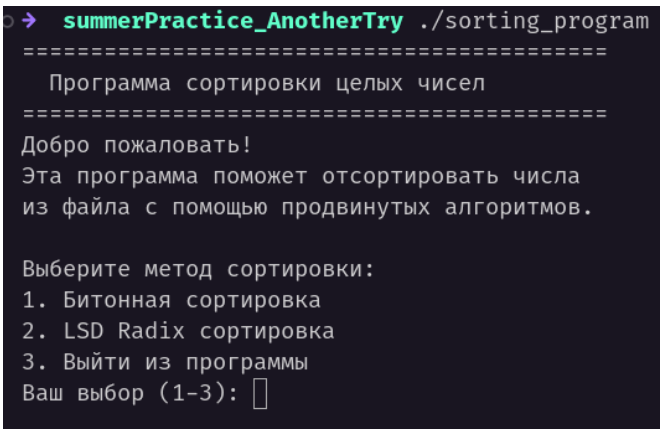

Рисунок 4.1 – текст приветствия

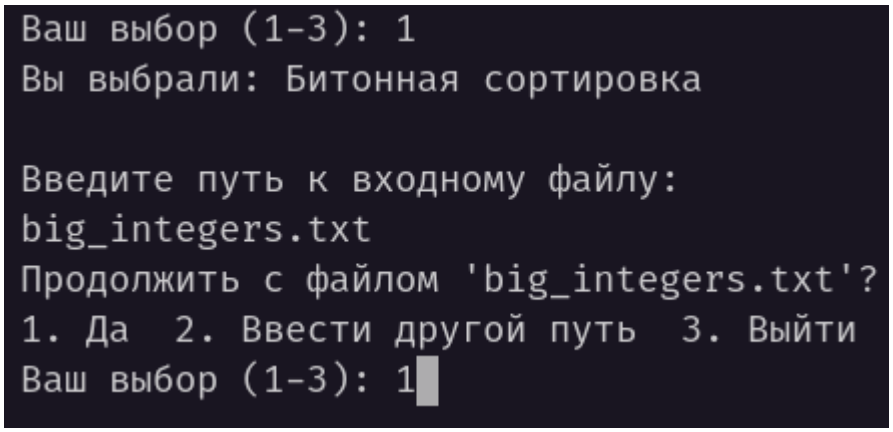

Рисунок 4.2 – Ввод пути к файлу с входными данными

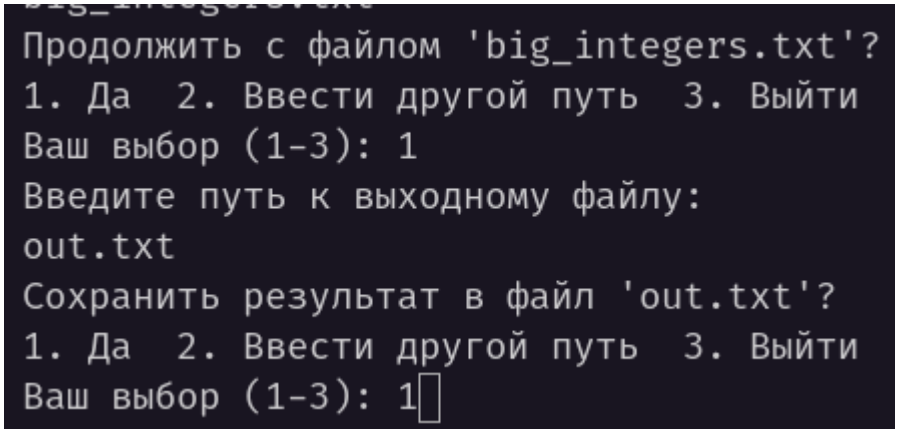

Рисунок 4.3 — Ввод файла для выходных данных

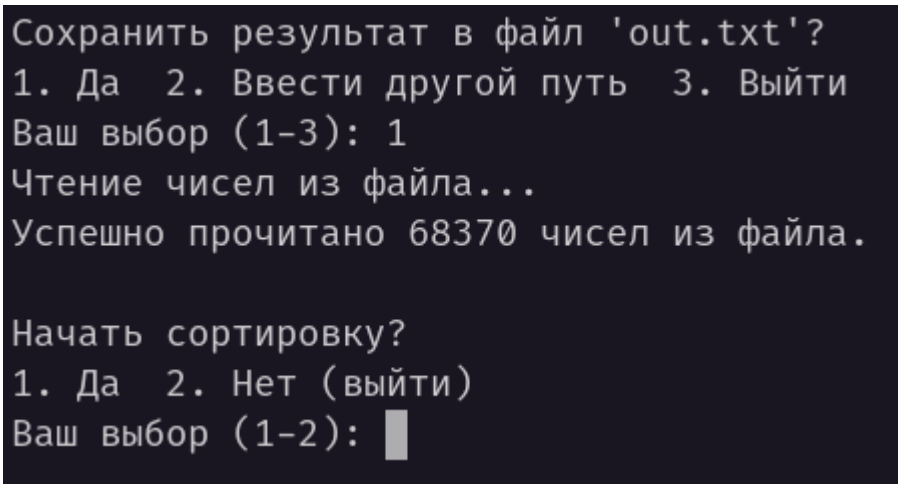

Рисунок 4.4 — Подтверждение начала сортировки

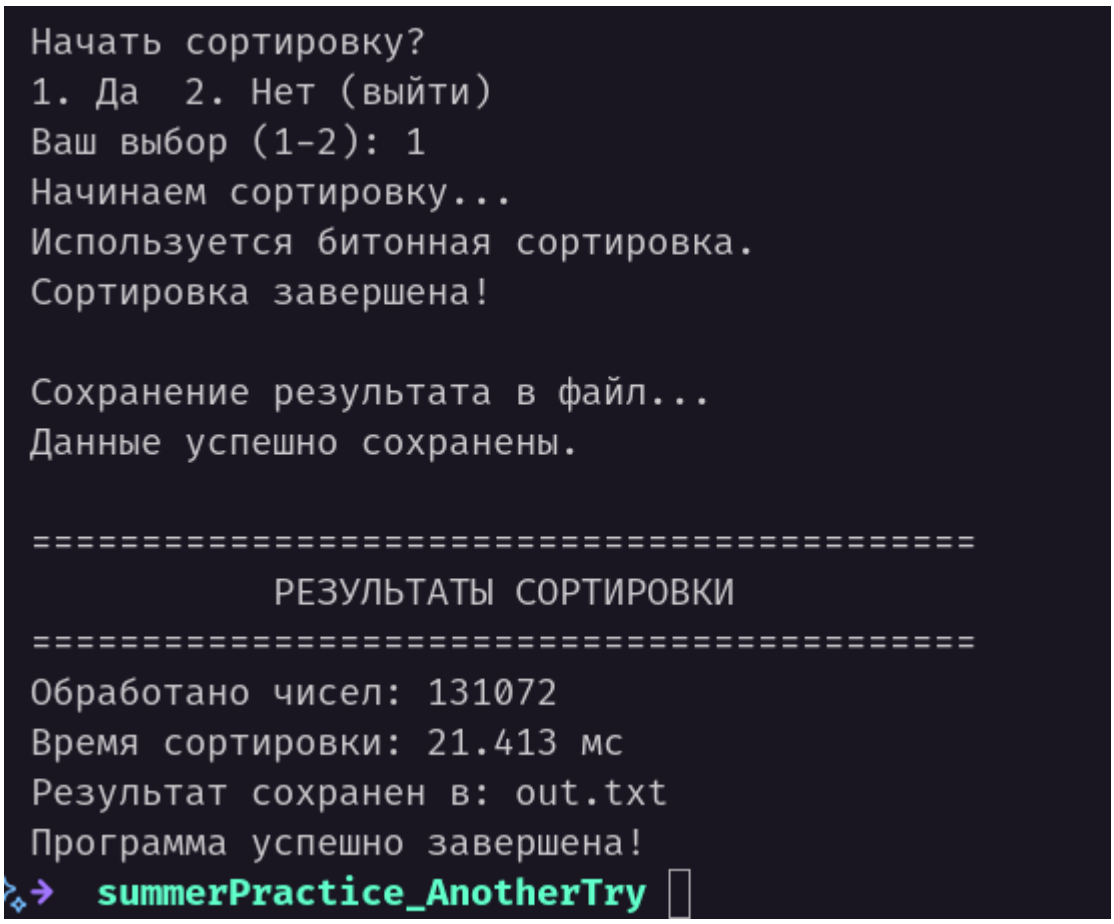

Рисунок 4.5 — Результат работы программы

4.3 Сравнение эффективности алгоритмов сортировки

Для сравнения эффективности битонной соритровки и lsd-сортировки будет использовано 3 файла: с 100000 числами, длина которых не превышает 9 цифр; с 250000 числами, длина которых не превышает 6 цифр; с 500000 числами, длина которых не превышает 5 цифр.

Результаты работы программы для этих файлов показанына рисунках 4.6, 4.7, 4.8, 4.9, 4.10, 4.11 и предстовляют битонную сортировку для первого файла, lsd-сортировку для первого файла, битонную сортировку для второго файла, lsd-сортировку для второго файла, битонную сортировку для третьего файла, lsd-сортировку для третьего файла соответственно

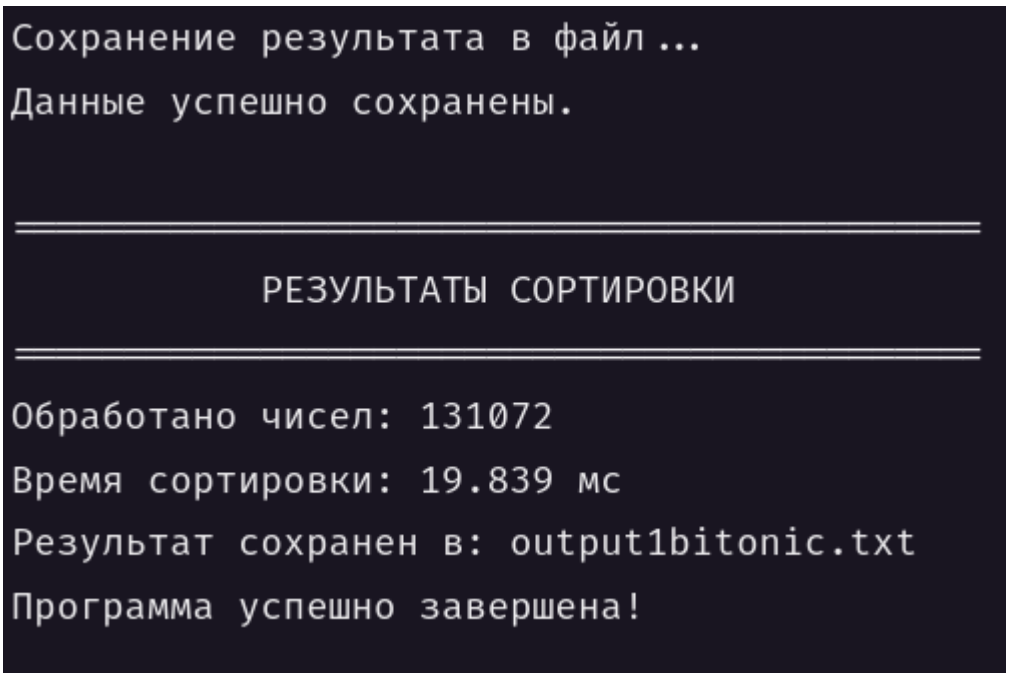

Рисунок 4.6 — битонная сортировка для первого файла

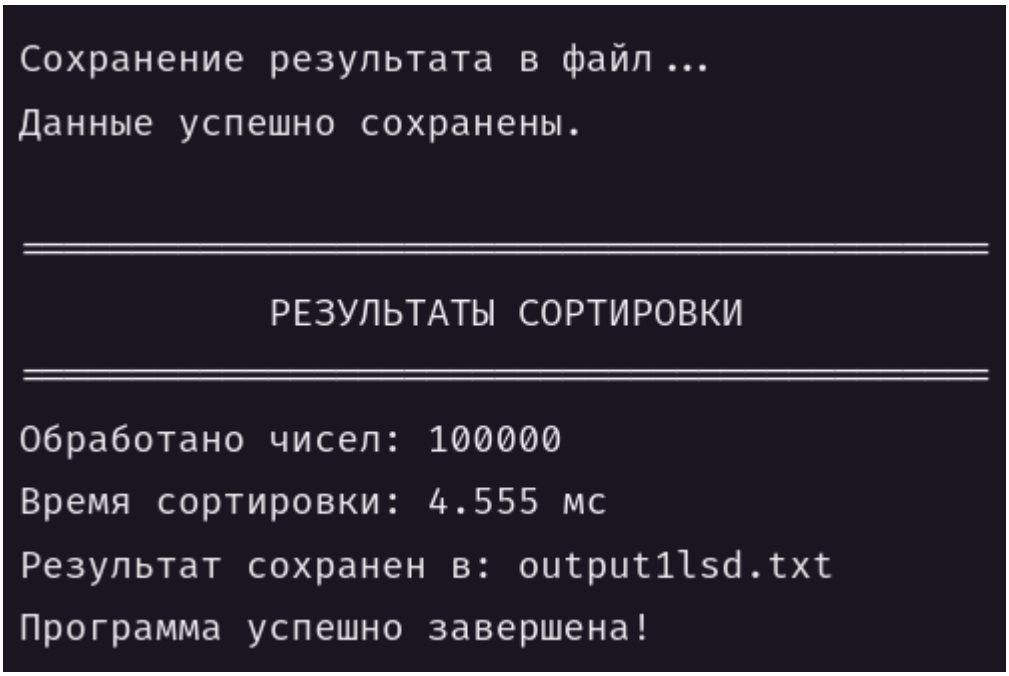

Рисунок 4.7 — lsd-сортировка для первого файла

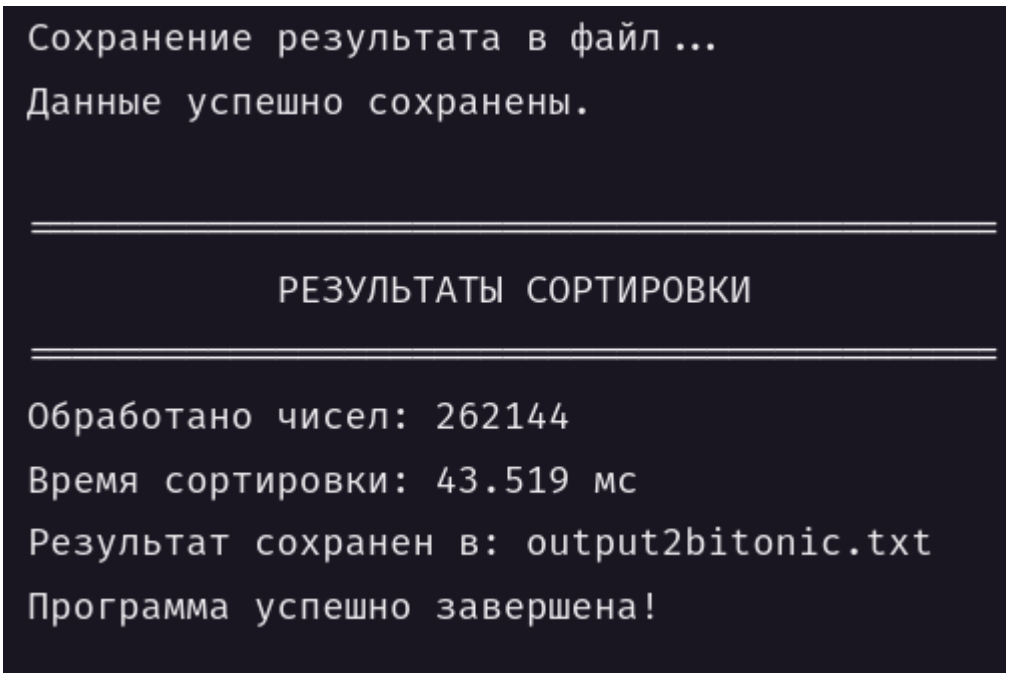

Рисунок 4.8 — битонная сортировка для второго файла

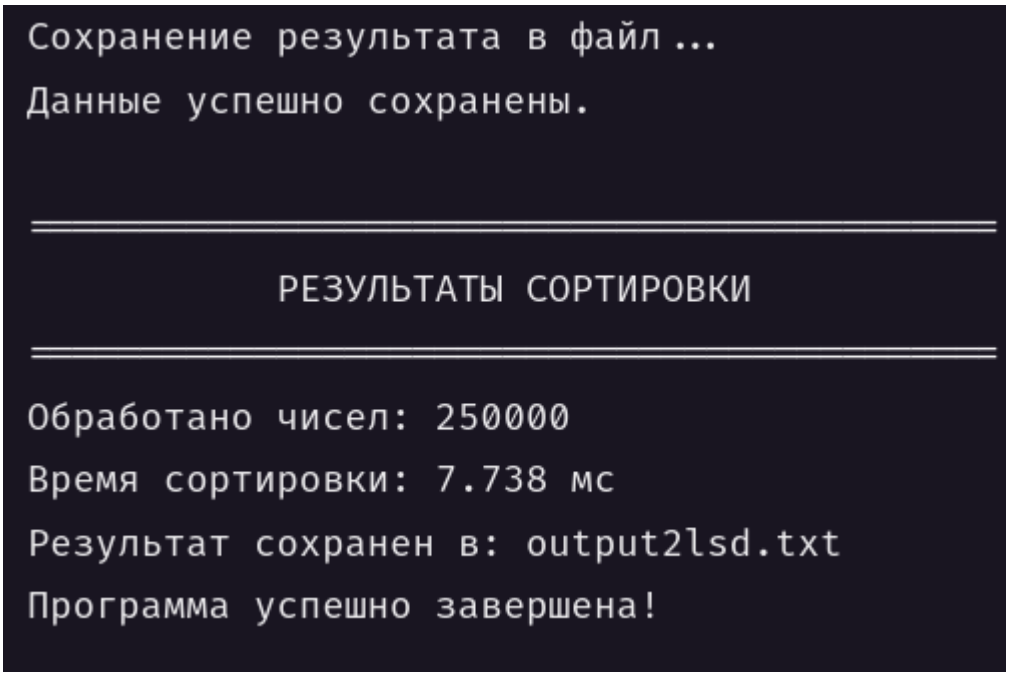

Рисунок 4.9 — lsd-сортировка для второго файла

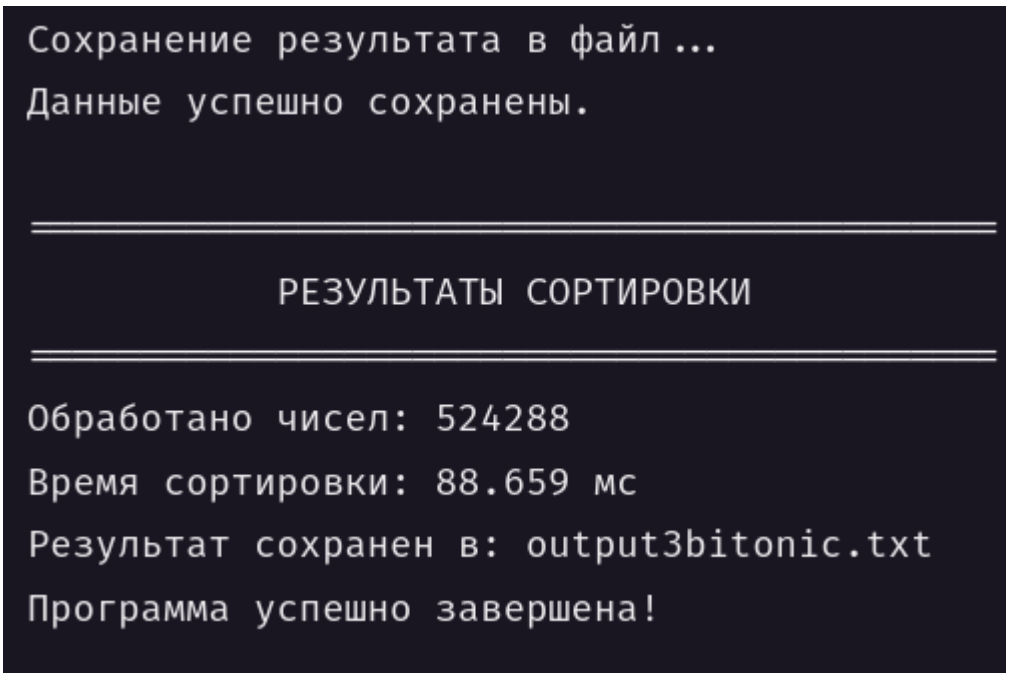

Рисунок 4.10 — битонная сортировка для третьего файла

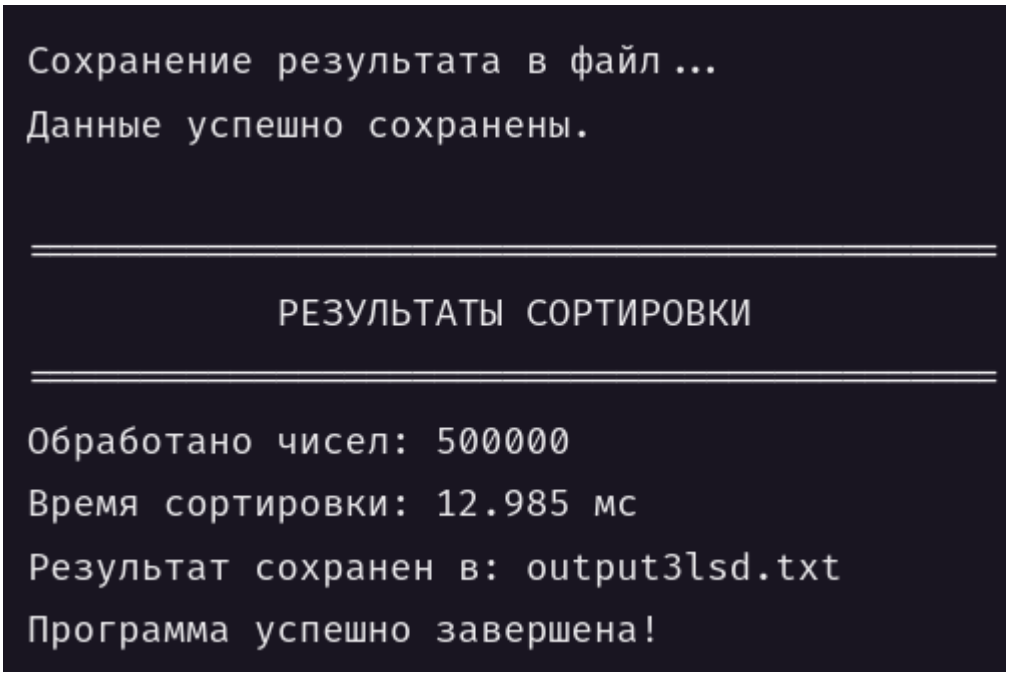

Рисунок 4.11 — lsd-сортировка для третьего файла

4.4 Выводы по главе

В данной главе была продемонстрирована работа разработанного программного обеспечения, а также проведено сравнительное тестирование двух алгоритмов сортировки на трёх различных входных файлах. Результаты экспериментов показали, что LSD-сортировка существенно превосходит

битонную сортировку по скорости выполнения. Однако следует отметить, что преимущество битонной сортировки заключается в её высокой эффективности при параллельной обработке данных, что не было реализовано в рамках данного проекта из-за ограниченных знаний и опыта в области параллельного программирования. Именно отсутствие распараллеливания, вероятнее всего, стало причиной её значительно более низкой производительности по сравнению с альтернативным методом.

В ходе выполнения данной работы было разработано консольное приложение на языке C++ в процедурной парадигме программирования, предназначенное для сортировки целых чисел, считываемых из файла, с последующей записью отсортированных данных в другой файл. Программа реализует два алгоритма сортировки: битонную и LSD-сортировку, а также обеспечивает базовый пользовательский интерфейс с возможностью выбора параметров и измерения времени выполнения операций.

Программное средство соответствует поставленным в разделе 1.4 задачам, в частности:

- реализованы оба алгоритма сортировки — битонная и LSD;
- реализована проверка доступности входных и выходных файлов;
- обеспечен механизм измерения времени выполнения сортировки;
- реализован интуитивно понятный текстовый интерфейс, позволяющий пользователю пошагово взаимодействовать с программой.

Таким образом, достигнута основная цель работы — исследование алгоритмов битонной и LSD-сортировки, реализация их программной поддержки и проведение анализа их эффективности при различных условиях.

Получены и закреплены навыки разработки и создания программного средства, связанные с использованием языка C++.

Перечень использованных информационных ресурсов

1. Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн, Алгоритмы. Постороение и анализ. Второе издание. / Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн.

2. Growing with the Web [Электронный ресурс], URL: https://www.growingwiththeweb.com/sorting/radix-sort-lsd/ (дата обращения: 11.06.2025 г.)

3. Skillfactory media [Электронный ресурс], URL: https://blog.skillfactory.ru/glossary/visual-studio-code/ (дата обращения: 11.06.2025 г.).

Приложение А Листинг программы

Листинг А.1 – Функция main

```
int main() {
    showWelcome();
    int sortMethod = chooseSortingMethod();
    if (sortMethod == -1) {
        return 0;
    }
    std::string inputFile = chooseInputFile();
    if (inputFile.empty()) {
        return 0;
    }
    std::string outputFile = chooseOutputFile();
    if (outputFile.empty()) {
        return 0;
    }
    std::vector<int> numbers;
    if (!readNumbersFromFile(inputFile, numbers)) {
            std::cout << "Ошибка при чтении файла. Программа
завершена.\n";
        return 1;
    }
    if (numbers.empty()) {
        std::cout << "Файл пуст или не содержит чисел.\n";
        return 1;
    }
    if (!confirmAction("Начать сортировку")) {
        return 0;
    }
    double sortTime = performSorting(numbers, sortMethod);
    if (!writeNumbersToFile(outputFile, numbers)) {
            std::cout << "Ошибка при записи в файл. Программа
завершена.\n";
        return 1;
    }
    showResults(numbers.size(), sortTime, outputFile);

    return 0;
}
```

Листинг А.2 – Функция showWelcome:

```
void showWelcome() {
    std::cout << "========================================\n";
    std::cout << "  Программа сортировки целых чисел\n";
    std::cout << "========================================\n";
    std::cout << "Добро пожаловать!\n";
    std::cout << "Эта программа поможет отсортировать числа\n";
    std::cout << "из файла с помощью продвинутых алгоритмов.\n\n";
```

```
}
```

Листинг А.3 — Функция readNumbersFromFile

```
     bool    readNumbersFromFile(const    std::string&    filename,
std::vector<int>& numbers) {
         if (!canReadFile(filename)) {
             return false;
         }
         std::ifstream file(filename);
         if (!file.is_open()) {
              std::cout << "Не удалось открыть файл для чтения: "
<< filename << "\n";
             return false;
         }
         numbers.clear();
         std::string line;
         int totalNumbers = 0;
         std::cout << "Чтение чисел из файла...\n";
         while (std::getline(file, line)) {
             std::istringstream iss(line);
             int number;
             while (iss >> number) {
                 numbers.push_back(number);
                 totalNumbers++;
             }
         }

         file.close();
          std::cout << "Успешно прочитано " << totalNumbers << "
чисел из файла.\n\n";
         return true;
     }
```

Листинг А.4 — Функция writeNumbersToFile

```
     bool  writeNumbersToFile(const  std::string&  filename,  const
std::vector<int>& numbers) {
         if (!canWriteFile(filename)) {
             return false;
         }
         std::ofstream file(filename);
         if (!file.is_open()) {
              std::cout << "Не удалось открыть файл для записи: "
<< filename << "\n";
             return false;
         }
         std::cout << "Сохранение результата в файл...\n";
         for (size_t i = 0; i < numbers.size(); ++i) {
             file << numbers[i];
             if (i < numbers.size() - 1) {
```

```
                file << "\n";
            }
        }
        file << "\n";
        file.close();
        std::cout << "Данные успешно сохранены.\n";
        return true;
    }
```

Листинг А.5 — Функция canReadFile

```
    bool canReadFile(const std::string& filename) {
        std::ifstream file(filename);
        if (!file.is_open()) {
              std::cout << "Ошибка: Не удается открыть файл для
чтения: " << filename << "\n";
             std::cout << "Проверьте, что файл существует и у вас
есть права на чтение.\n";
            return false;
        }
        file.close();
        return true;
    }
```

Листинг А.6 — Функция canWriteFile

```
    bool canWriteFile(const std::string& filename) {
        std::ofstream file(filename, std::ios::app);
        if (!file.is_open()) {
              std::cout << "Ошибка: Не удается открыть файл для
записи: " << filename << "\n";
             std::cout << "Проверьте права доступа к папке и имя
файла.\n";
            return false;
        }
        file.close();
        return true;
    }
```

Листинг А.7 — Функция chooseSortingMethod

```
    int chooseSortingMethod() {
        std::cout << "Выберите метод сортировки:\n";
        std::cout << "1. Битонная сортировка\n";
        std::cout << "2. LSD Radix сортировка\n";
        std::cout << "3. Выйти из программы\n";
        int choice;
        while (true) {
            std::cout << "Ваш выбор (1-3): ";
            std::cin >> choice;
            if (choice == 1 || choice == 2) {
                  std::cout << "Вы выбрали: " << (choice == 1 ?
"Битонная сортировка" : "LSD Radix сортировка") << "\n\n";
```

```
                return choice;
            } else if (choice == 3) {
                std::cout << "До свидания!\n";
                return -1;
            } else {
                 std::cout << "Неверный выбор. Попробуйте снова.
\n";
                std::cin.clear();
                 std::cin.ignore(10000, '\n'); // Очистка буфера
при ошибке
            }
        }
    }
```

Листинг А.8 — Функция chooseInputFile

```
    std::string chooseInputFile() {
        while (true) {
            std::cout << "Введите путь к входному файлу:\n";
            std::string filename;
            std::cin.ignore(); // Очистка буфера перед getline
            std::getline(std::cin, filename);
             std::cout << "Продолжить с файлом '" << filename <<
"'?\n";
               std::cout << "1. Да   2. Ввести другой путь   3.
Выйти\n";
            int choice = getYesNoExit();
            if (choice == 1) {
                return filename;
            } else if (choice == 0) {
                continue; // Повторить ввод
            } else {
                std::cout << "До свидания!\n";
                return "";
            }
        }
    }
```

Листинг А.9 — Функция chooseOutputFile

```
    std::string chooseOutputFile() {
        while (true) {
            std::cout << "Введите путь к выходному файлу:\n";
            std::string filename;
            std::getline(std::cin, filename);
                std::cout << "Сохранить результат в файл '" <<
filename << "'?\n";
               std::cout << "1. Да   2. Ввести другой путь   3.
Выйти\n";
            int choice = getYesNoExit();
            if (choice == 1) {
                return filename;
            } else if (choice == 0) {
```

```
                continue;
            } else {
                std::cout << "До свидания!\n";
                return "";
            }
        }
    }
```

Листинг А.10 — Функция confirmAction

```
    bool confirmAction(const std::string& action) {
        std::cout << action << "?\n";
        std::cout << "1. Да  2. Нет (выйти)\n";
        int choice;
        while (true) {
            std::cout << "Ваш выбор (1-2): ";
            std::cin >> choice;
            if (choice == 1) {
                return true;
            } else if (choice == 2) {
                std::cout << "До свидания!\n";
                return false;
            } else {
                 std::cout << "Неверный выбор. Попробуйте снова.
\n";
                std::cin.clear();
                std::cin.ignore(10000, '\n');
            }
        }
    }
```

Листинг А.11 — Функция showResults

```
    void showResults(int numbersCount, double sortTime, const
std::string& outputFile) {
                                            std::cout          <<
"\n=========================================\n";
        std::cout << "          РЕЗУЛЬТАТЫ СОРТИРОВКИ\n";
                                            std::cout          <<
"=========================================\n";
          std::cout << "Обработано чисел: " << numbersCount <<
"\n";
        std::cout << "Время сортировки: " << sortTime << " мс\n";
         std::cout << "Результат сохранен в: " << outputFile <<
"\n";
        std::cout << "Программа успешно завершена!\n";
    }
```

Листинг А.12 — Функция getYesNoExit

```
    int getYesNoExit() {
        int choice;
        while (true) {
            std::cout << "Ваш выбор (1-3): ";
            std::cin >> choice;
```

```
            if (choice == 1) {
                std::cin.ignore();
                return 1; // Да
            } else if (choice == 2) {
                std::cin.ignore();
                return 0; // Нет (повторить)
            } else if (choice == 3) {
                std::cin.ignore();
                return -1; // Выйти
            } else {
                  std::cout << "Неверный выбор. Попробуйте снова.
\n";
                std::cin.clear();
                std::cin.ignore(10000, '\n');
            }
        }
    }
```

Листинг А.13 — Функция performSorting

```
    double    performSorting(std::vector<int>&    numbers,    int
sortMethod) {
        std::cout << "Начинаем сортировку...\n";
        auto start = std::chrono::high_resolution_clock::now();
        if (sortMethod == 1) {
            std::cout << "Используется битонная сортировка.\n";
            bitonicSort(numbers);
        } else if (sortMethod == 2) {
            std::cout << "Используется LSD Radix сортировка.\n";
            lsdRadixSort(numbers);
        }
        auto end = std::chrono::high_resolution_clock::now();
                                    auto        duration        =
std::chrono::duration_cast<std::chrono::microseconds>(end        -
start);
        double timeMs = duration.count() / 1000.0;
        std::cout << "Сортировка завершена!\n\n";
        return timeMs;
    }
```

Листинг А.14 — Функция bitonicSort

```
    void bitonicSort(std::vector<int>& arr) {
        if (arr.empty()) return;
        prepareForBitonicSort(arr);
        bitonicSortRecursive(arr, 0, arr.size(), true);
    }
```

Листинг А.15 — Функция prepareForBitonicSort

```
    void prepareForBitonicSort(std::vector<int>& arr) {
        int n = arr.size();
        int powerOfTwo = 1;
        while (powerOfTwo < n) {
```

```
            powerOfTwo *= 2;
        }
        if (powerOfTwo > n) {
                int  maxVal  =  *std::max_element(arr.begin(),
arr.end());
            arr.resize(powerOfTwo, maxVal + 1);
        }
    }
```

Листинг А.16 — Функция bitonicSortRecursive

```
    void bitonicSortRecursive(std::vector<int>& arr, int low, int
cnt, bool dir) {
        if (cnt > 1) {
            int k = cnt / 2;
            bitonicSortRecursive(arr, low, k, true);
            bitonicSortRecursive(arr, low + k, k, false);
            bitonicMerge(arr, low, cnt, dir);
        }
    }
```

Листинг А.17 — Функция bitonicMerge

```
    void bitonicMerge(std::vector<int>& arr, int low, int cnt,
bool dir) {
        if (cnt > 1) {
            int k = cnt / 2;
            for (int i = low; i < low + k; i++) {
                compareAndSwap(arr, i, i + k, dir);
            }
            bitonicMerge(arr, low, k, dir);
            bitonicMerge(arr, low + k, k, dir);
        }
    }
```

Листинг А.18 — Функция compareAndSwap

```
    void compareAndSwap(std::vector<int>& arr, int i, int j, bool
dir) {
        if ((arr[i] > arr[j]) == dir) {
            std::swap(arr[i], arr[j]);
        }
    }
```

Листинг А.19 — Функция lsdRadixSort

```
    void lsdRadixSort(std::vector<int>& arr) {
        if (arr.empty()) return;
        int maxNum = getMax(arr);
        for (int exp = 1; maxNum / exp > 0; exp *= 10) {
            countingSort(arr, exp);
        }
    }
```

Листинг А.20 — Функция countingSort

```
    void countingSort(std::vector<int>& arr, int exp) {
```

```
    int n = arr.size();
    std::vector<int> output(n);
    std::vector<int> count(10, 0);
    for (int i = 0; i < n; i++) {
        count[(arr[i] / exp) % 10]++;
    }
    for (int i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
```

Листинг А.21 — Функция getMax

```
int getMax(const std::vector<int>& arr) {
    int maxVal = arr[0];
    for (size_t i = 1; i < arr.size(); i++) {
        if (arr[i] > maxVal) {
            maxVal = arr[i];
        }
    }
    return maxVal;
}
```