



МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего
образования*

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания № 4

Тема:

«Алгоритмы внешних сортировок»

Дисциплина : «Структуры и алгоритмы обработки данных»

Выполнил студент: Борзов Дмитрий Олегович

Фамилия И.О

Группа: ИКБО-12-22

Номер группы

Москва - 2023

1 ЦЕЛЬ РАБОТЫ

1.1 Цель:

Освоить приёмы сортировки данных из файлов.

2 ХОД РАБОТЫ

2.1 Алгоритм сортировки прямого слияния для файлов

Фаза разделения:

1. Открыть файл А как входной.
2. Открыть файлы В и С как выходные (для записи).
3. Считываемые из А записи попеременно записываем в файлы В и С.
4. Закрываем файлы А, В, С.

Фаза слияния:

1. Открыть файл А как выходной (для записи).
2. Открываем файлы В и С как входные (для чтения).
3. Установить размер порции сливаемых данных: 1, 2, 4, 8 и т.д. для этого и следующих этапов.
4. Для каждой порции считываются по одной записи из файлов В и С.
5. Меньшая запись записывается в файл А, и считывается очередная запись из того файла, запись которого была переписана в файл А.
6. Пункты 4 и 5 повторяются до тех пор, пока записи очередной порции одного из файлов не будут исчерпаны.
7. Оставшиеся записи из порции другого файла переписываются в файл А.
8. Пункты с 4 по 7 повторяются до тех пор, пока не будет достигнут конец одного из файлов В и С. Тогда оставшиеся записи из другого файла переписываются в файл А.
9. Закрываются файлы А В С.

Сортировка завершается тогда, когда длина порции достигнет n .

Для удобства реализации алгоритм разбит на 3 части, основной цикл алгоритма проходит в главной функции, а разделение и слияние реализовано через отдельные функции. Блок схемы алгоритмов (см. рис 1-3).

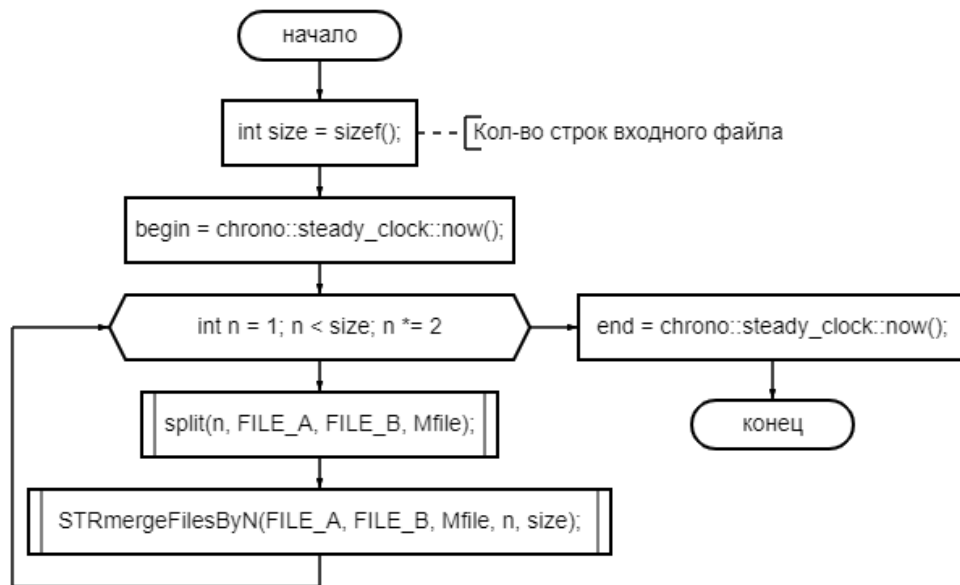


Рисунок 1 – Основная часть алгоритма

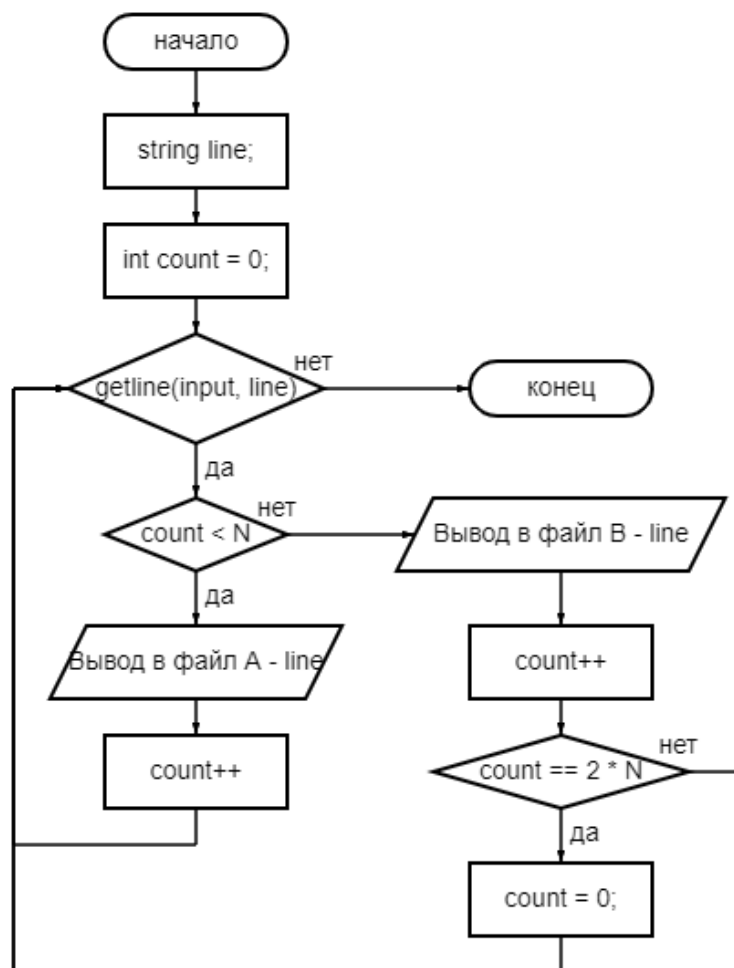


Рисунок 2 – Фаза разделения

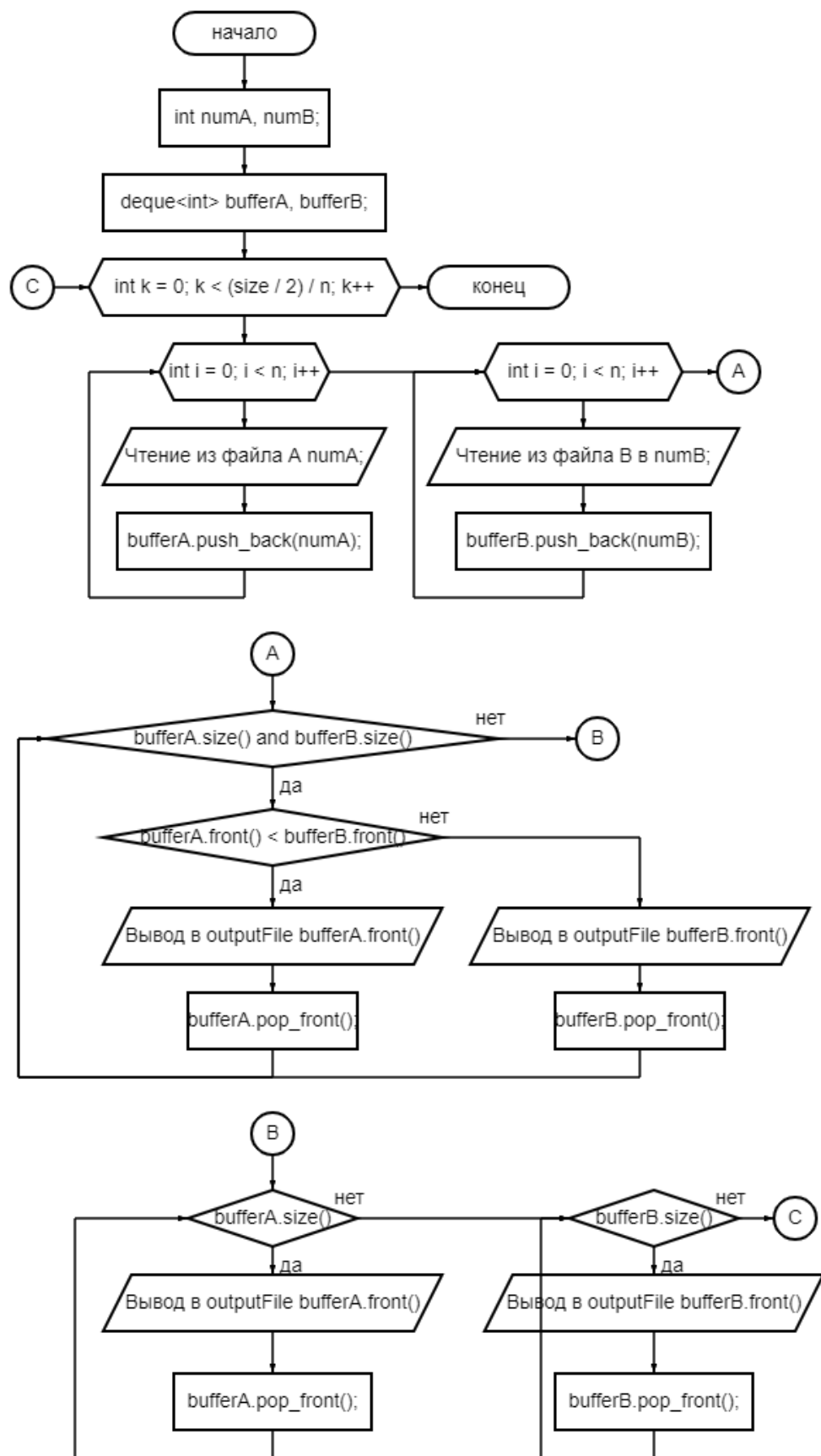


Рисунок 3 – Фаза слияния

2.1.1 Реализация алгоритма на языке с++

```
int split(int N, const char* file_a, const char* file_b, const char* file_in) //Фаза
разделения
{
    // Открываем файл generated_data.txt для чтения
    ifstream input(file_in);
    if (!input) {
        cerr << "Cannot open file " << file_in << " for reading\n";
        return 1;
    }

    // Открываем файл A.txt для записи
    ofstream output_a(file_a);
    if (!output_a) {
        cerr << "Cannot open file " << file_a << " for writing\n";
        return 1;
    }

    // Открываем файл B.txt для записи
    ofstream output_b(file_b);
    if (!output_b) {
        cerr << "Cannot open file " << file_b << " for writing\n";
        return 1;
    }

    // Читаем файл generated_data.txt построчно и записываем в файлы A.txt и B.txt
    string line;
    int count = 0;
    while (getline(input, line))
    {
        if (count < N)
        {
            output_a << line << '\n';
            count++;
        }
        else
        {
            output_b << line << '\n';
            count++;
            if (count == 2 * N)
            {
                count = 0;
            }
        }
    }

    // Закрываем все открытые файлы
    input.close();
    output_a.close();
    output_b.close();

    return 0;
}
```

```

void INTmergeFilesByN(const char* file_a, const char* file_b, const char* file_out,
int n, int size) //Фаза слияния
{
    ifstream fileA(file_a), fileB(file_b); // Открываем два файла для чтения
    ofstream outputFile(file_out); // Открываем выходной файл для записи

    int numA, numB; // Объявляем переменные для хранения считываемых чисел из файлов

    // Объявляем два буфера на n элементов каждый, для слияния
    deque<int> bufferA, bufferB;

    // Читаем по n элементов из каждого файла, помещая их в соответствующие буферы
    for (int k = 0; k < (size / 2) / n; k++)
    {
        for (int i = 0; i < n; i++)
        {
            fileA >> numA;
            bufferA.push_back(numA);
        }
        for (int i = 0; i < n; i++)
        {
            fileB >> numB;
            bufferB.push_back(numB);
        }

        // Сливаем два буфера, записывая отсортированный результат в выходной файл
        while (bufferA.size() and bufferB.size())
        {
            if (bufferA.front() < bufferB.front())
            {
                outputFile << bufferA.front() << endl;
                bufferA.pop_front();
            }
            else
            {
                outputFile << bufferB.front() << endl;
                bufferB.pop_front();
            }
        }

        // Если остались элементы только в одном из буферов, записываем их в
        выходной файл
        while (bufferA.size())
        {
            outputFile << bufferA.front() << endl;
            bufferA.pop_front();
        }

        while (bufferB.size())
        {
            outputFile << bufferB.front() << endl;
            bufferB.pop_front();
        }

        // Очищаем буферы для следующей итерации цикла
        bufferA.clear();
        bufferB.clear();
    }
}

```

```

int main() // Основная функция
{
    int size = sizeof();
    chrono::steady_clock::time_point begin = chrono::steady_clock::now();
    for (int n = 1; n < size; n *= 2)
    {
        split(n, FILE_A, FILE_B, Mfile);
        STRmergeFilesByN(FILE_A, FILE_B, Mfile, n, size);
    }
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Time difference = " << chrono::duration_cast<chrono::milliseconds>(end
- begin).count() << "[ms]" << endl;
    return 0;
}

```

2.1.2 Отладка программы на примере

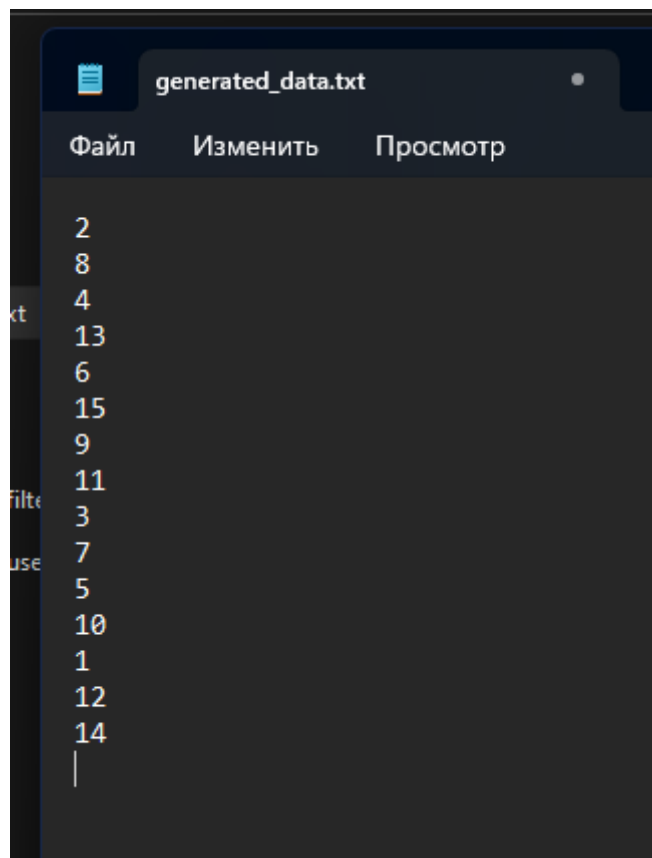


Рисунок 4 – Исходный файл

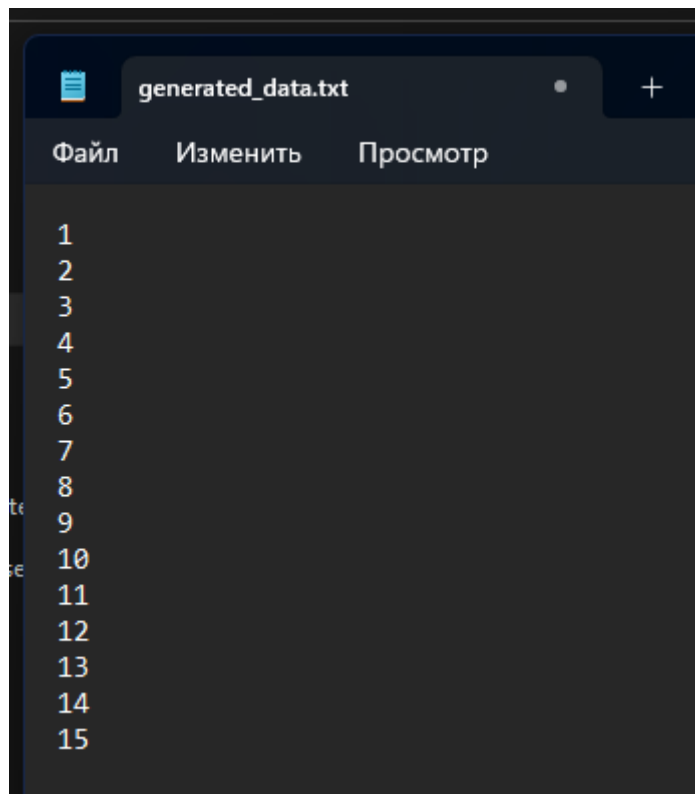


Рисунок 5 – Конечный файл

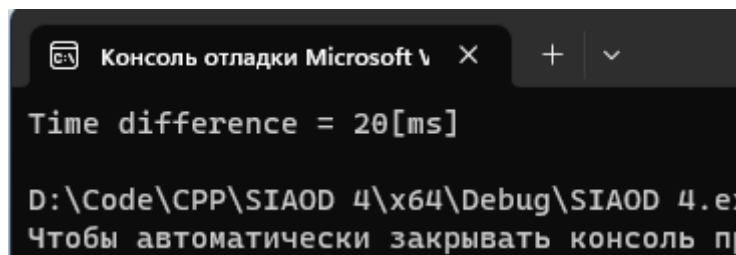


Рисунок 6 - Консоль

2.1.3 Персональный вариант

Пример файла размером 8 персонального варианта:

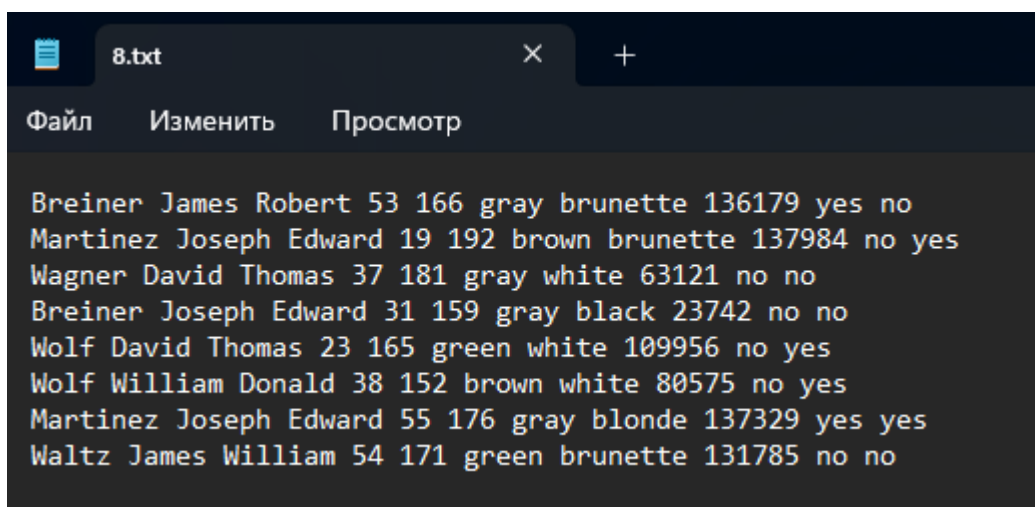


Рисунок 7 – Пример персонального варианта

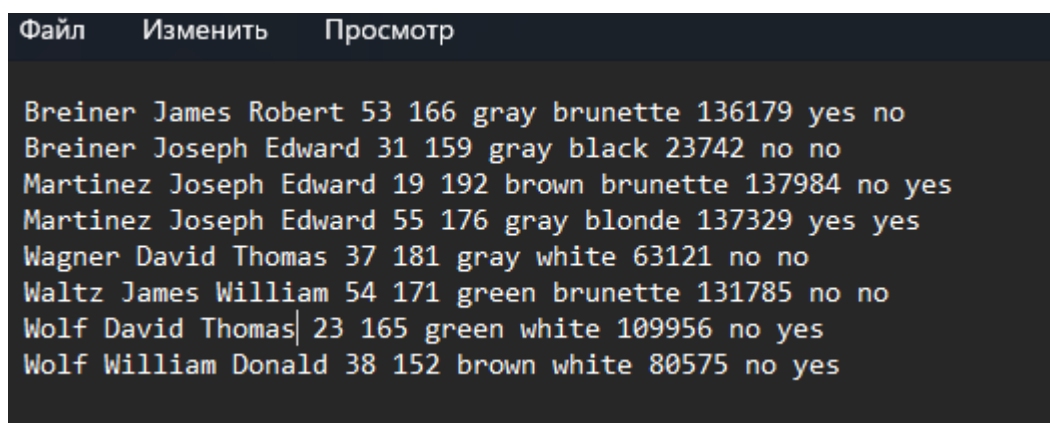


Рисунок 8 – Пример результата работы алгоритма

Таблица 1 – Результаты работы алгоритма на файлах разной размерности

n	T(n), мс
8	15
16	24
32	33
128	61
1024	239
1048576	310556

2.2 Алгоритм естественного слияния, оптимизированный для файлов

Для усовершенствования этой сортировки был предложен вариант предварительного разделения данных в файле на серии одной длины, загрузки каждой серии в оперативную память, сортировки этой серии, например, алгоритмом быстрой сортировки, и запись этих серий в исходный файл. Чем длиннее серию возможно выгрузить в память, отсортировать и вернуть в файл, тем эффективнее будет алгоритм самой сортировки.

Такое решение предлагается вам исследовать и разобраться в реализации.

Рассмотрим алгоритм и его фазы. Он так же является двух фазным.

1. Определить размер свободной оперативной памяти для выгрузки в нее серии из файла. В программе создаем массив для хранения серии *buf*.

2. Открыть исходный файл *A*, подлежащий сортировке.

3. Открыть два файла для записи *B* и *C*.

4. Считать последовательность данных в количестве достаточном для размещения в массиве *buf*. Отсортировать в массиве методом внутренней сортировки и записать в файл *B*.

5. Считать следующую последовательность данных в количестве достаточном для размещения в массиве *buf*. Отсортировать в массиве методом внутренней сортировки и записать в файл *C*.

6. Пункты 4 и 5 выполнять, пока все данные из файла *A* не будут переписаны отсортированными во вспомогательные файлы *B* и *C*.

7. Слить данные в файл *A* сначала из файла *B*, затем из файла *C*.

Теперь файл *A* содержит длинные упорядоченные серии, считаем, что данные в сериях упорядочены по возрастанию.

8. Фаза разделения включает поочередную запись серий из *A* в файлы *B* и *C*.

9. Фаза слияния имеет теперь следующий алгоритм:

- Считываем данные из одного и другого файлов, пока $a_i < a_{i+1}$, меньшее из сравниваемых записывать в файл *A*, пока одна из серий не будет исчерпана, тогда остаток другой переписываем в файл *A*, пока выполняется условие $a_i < a_{i+1}$.

- После этого считываем следующую серию и так пока один из файлов не станет пустым, тогда серии другого переписываются в файл *A*.

10. Пункты 8 и 9 повторяются пока в файл *A*, в результате слияния не будет переписана только одна серия. Блок схема алгоритма (см. рис 9-10).

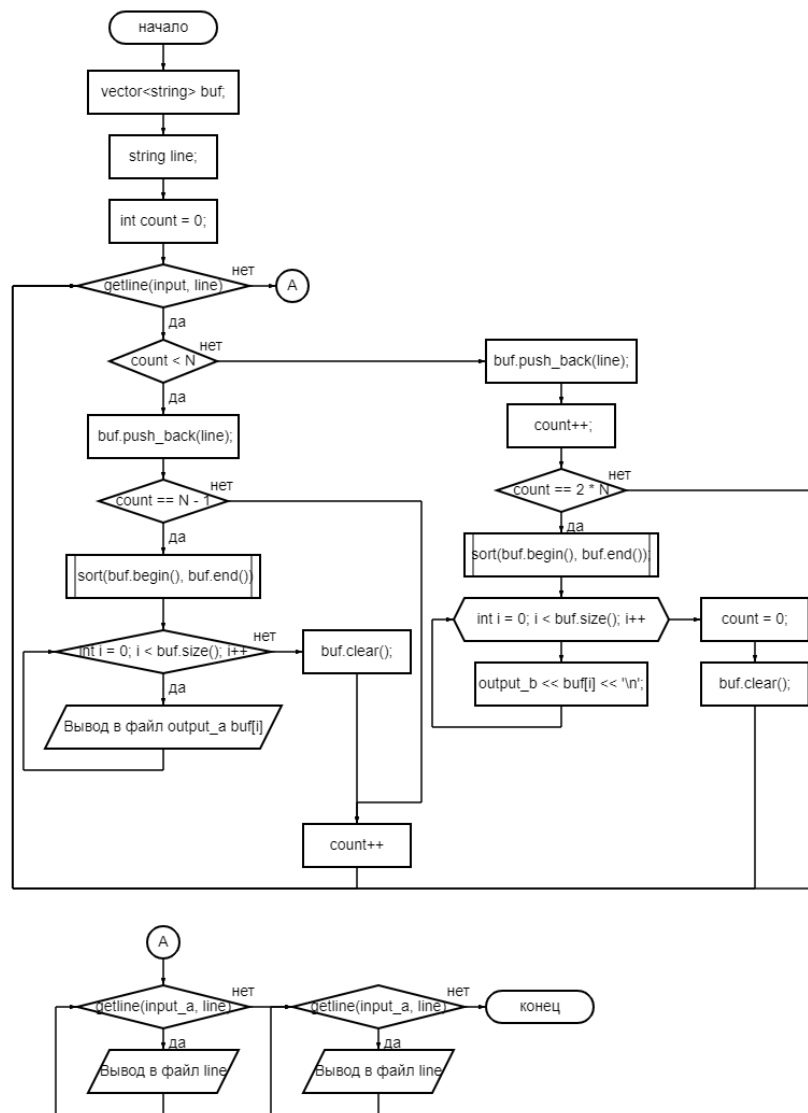


Рисунок 9 – Функция с методом внутренней сортировки

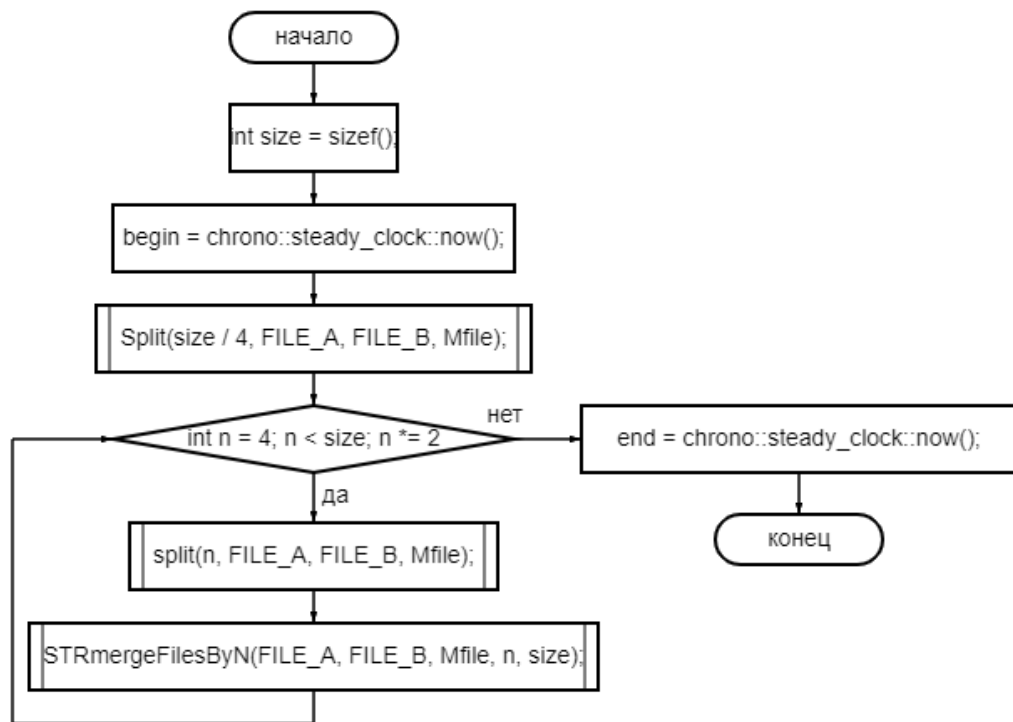


Рисунок 10 - Основная функция

2.2.1 Код алгоритма

```

int Split(int N, const char* file_a, const char* file_b, const char* file_in) // 1-7
пункты алгоритма
{
    // Открываем файл generated_data.txt для чтения
    ifstream input(file_in);
    if (!input) {
        cerr << "Cannot open file " << file_in << " for reading\n";
        return 1;
    }

    // Открываем файл A.txt для записи
    ofstream output_a(file_a);
    if (!output_a) {
        cerr << "Cannot open file " << file_a << " for writing\n";
        return 1;
    }

    // Открываем файл B.txt для записи
    ofstream output_b(file_b);
    if (!output_b) {
        cerr << "Cannot open file " << file_b << " for writing\n";
        return 1;
    }

    // Создаем вектор для хранения считанных значений
    vector<string> buf;

    // Читаем файл generated_data.txt построчно и записываем в файлы A.txt и B.txt
    string line;
    int count = 0;

```

```

while (getline(input, line)) // Читаем файл построчно и записываем в переменную line
{
    if (count < N) // Если количество считанных строк меньше N, то добавляем
    вектору buf
    {
        buf.push_back(line);
        if (count == N - 1) // Если buf заполнен, то выполняем сортировку и
запись в output_a
        {
            sort(buf.begin(), buf.end());

            for (int i = 0; i < buf.size(); i++)
            {
                output_a << buf[i] << '\n';
            }
            buf.clear(); // Очищаем буфер после записи
        }
        count++;
    }
    else // Если количество считанных строк больше или равно размеру серии N,
то записываем в buf и в output_b
    {
        buf.push_back(line);
        count++;
        if (count == 2 * N) // Если buf заполнен второй раз, то выполняем
сортировку и запись в output_b
        {
            sort(buf.begin(), buf.end());
            for (int i = 0; i < buf.size(); i++)
            {
                output_b << buf[i] << '\n';
            }
            // Если buf заполнен второй раз, то выполняем сортировку и запись в
output_b
            count = 0;
            buf.clear();
        }
    }
}

// Закрываем все открытые файлы
input.close();
output_a.close();
output_b.close();

// Открываем файл generated_data.txt для записи
ofstream output(file_in);
if (!output) {
    cerr << "Cannot open file " << file_in << " for writing\n";
    return 1;
}

// Открываем файл A.txt для чтения
ifstream input_a(file_a);
if (!input_a) {
    cerr << "Cannot open file " << file_a << " for reading\n";
    return 1;
}

// Открываем файл B.txt для чтения
ifstream input_b(file_b);
if (!input_b) {
    cerr << "Cannot open file " << file_b << " for reading\n";
    return 1;
}

```

```

while (getline(input_a, line))
{
    output << line << '\n';
}

while (getline(input_b, line))
{
    output << line << '\n';
}
return 0;
}

int main() // основная функция
{

    int size = sizeof();
    chrono::steady_clock::time_point begin = chrono::steady_clock::now();
    Split(size / 4, FILE_A, FILE_B, Mfile);

    for (int n = 4; n < size; n += 2)
    {
        split(n, FILE_A, FILE_B, Mfile);
        STRmergeFilesByN(FILE_A, FILE_B, Mfile, n, size);
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Time difference = " << chrono::duration_cast<chrono::milliseconds>(end
- begin).count() << "[ms]" << endl;
    return 0;
}

```

2.2.2 Отладка алгоритма на примере из п.4

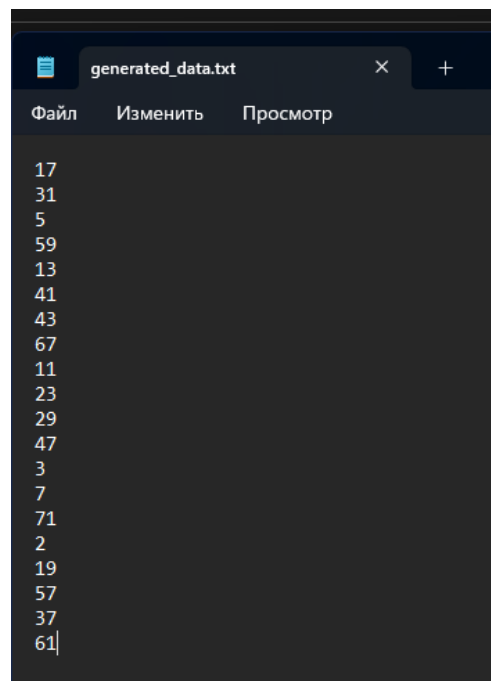


Рисунок 11 – Исходный файл

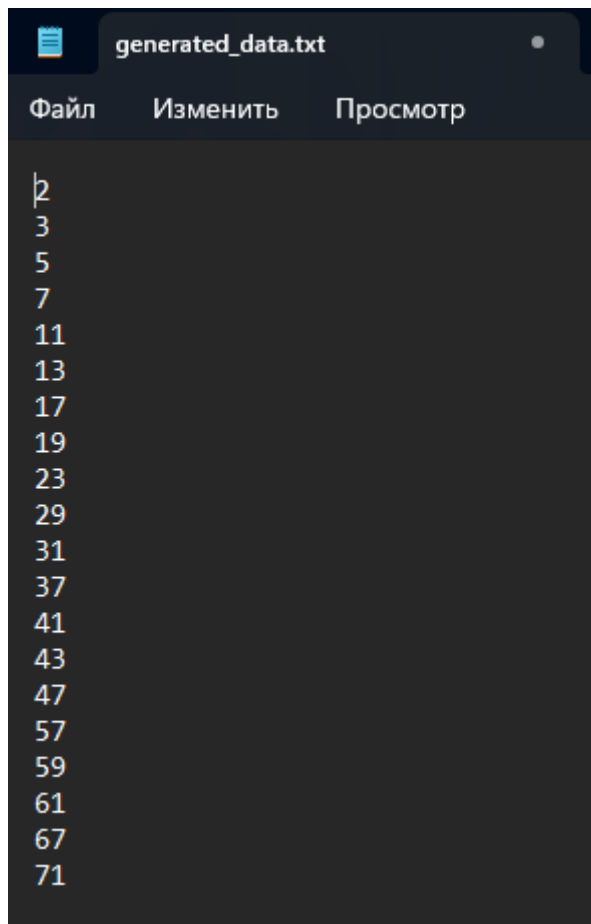


Рисунок 12 – Результат работы алгоритма

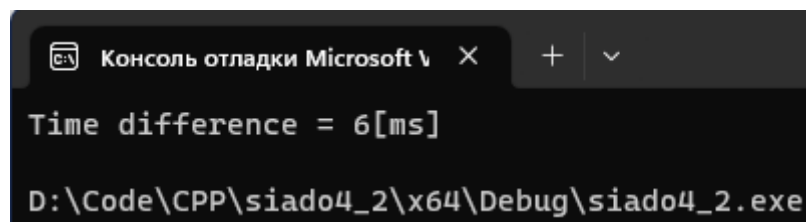
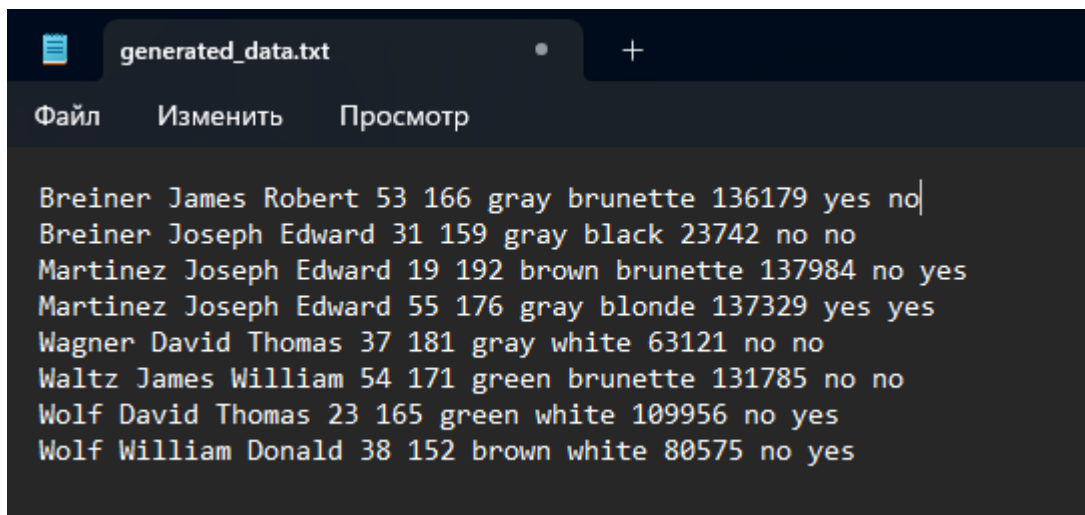


Рисунок 13 – Консоль

2.2.3 Персональный вариант



The screenshot shows a text editor window with a dark theme. The title bar at the top says 'generated_data.txt'. Below the title bar is a menu bar with three items: 'Файл' (File), 'Изменить' (Edit), and 'Просмотр' (View). The main text area contains eight lines of data, each representing a person with various attributes separated by spaces. The data is as follows:

Имя	Фамилия	Имя	Возраст	Рост	Цвет волос	Цвет глаз	Идентификатор	Атрибут 1	Атрибут 2
Breiner	James	Robert	53	166	gray	brunette	136179	yes	no
Breiner	Joseph	Edward	31	159	gray	black	23742	no	no
Martinez	Joseph	Edward	19	192	brown	brunette	137984	no	yes
Martinez	Joseph	Edward	55	176	gray	blonde	137329	yes	yes
Wagner	David	Thomas	37	181	gray	white	63121	no	no
Waltz	James	William	54	171	green	brunette	131785	no	no
Wolf	David	Thomas	23	165	green	white	109956	no	yes
Wolf	William	Donald	38	152	brown	white	80575	no	yes

Таблица 2 - Результаты работы алгоритма на файлах разной размерности

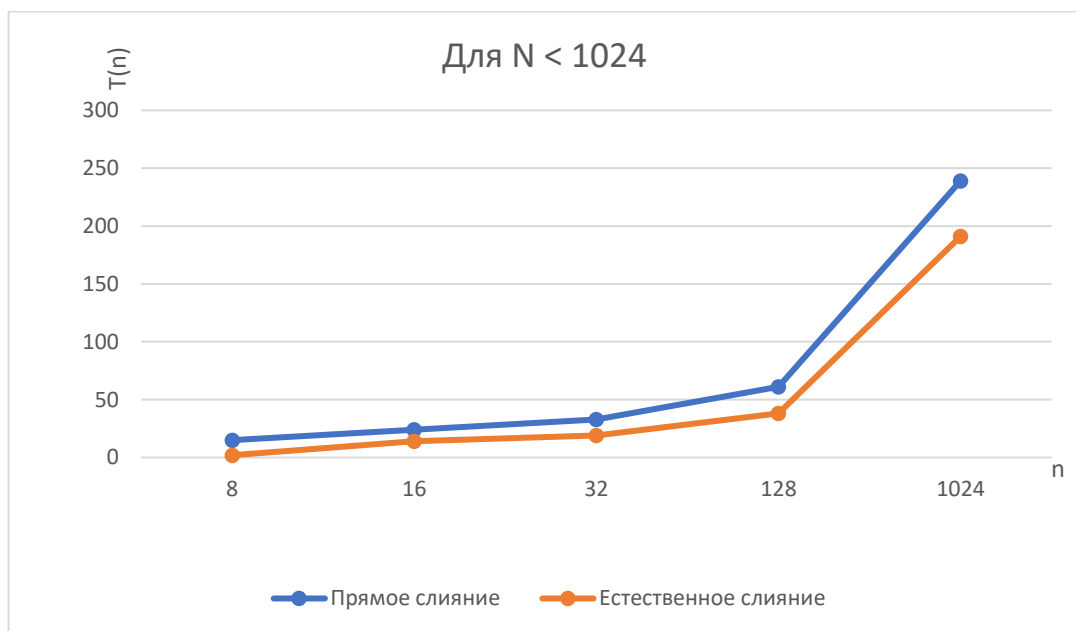
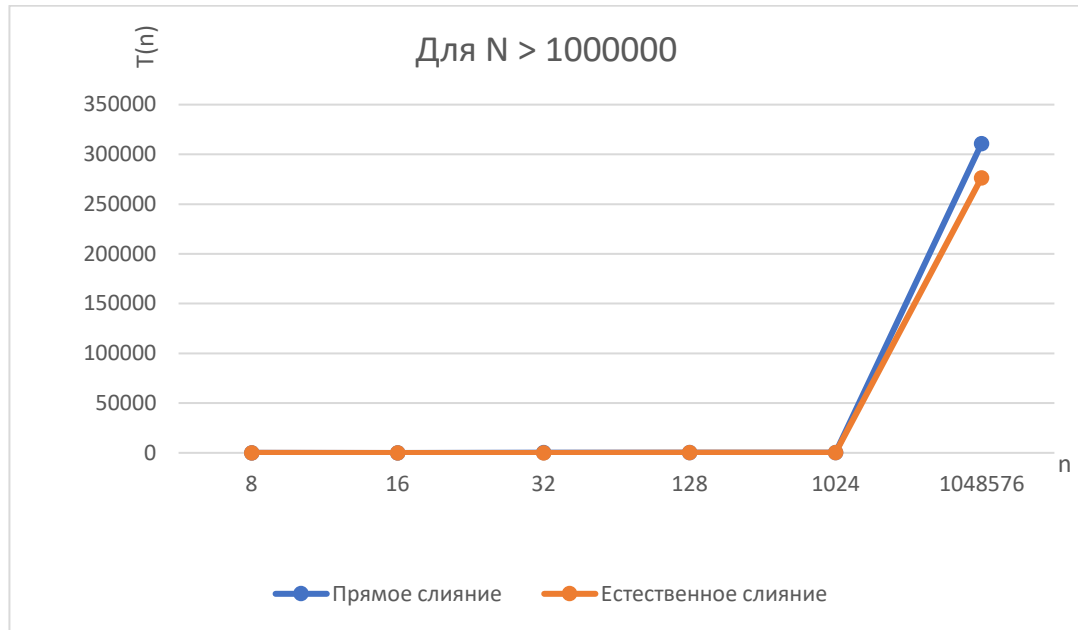
n	T(n), мс
8	2
16	14
32	19
128	38
1024	191
1048576	276273

2.3 Сравнение двух алгоритмов

Таблица 3 – Общая таблица результатов

n	T1(n), мс	T2(n), мс
8	15	2
16	24	14
32	33	19
128	61	38
1024	239	191
1048576	310556	276273

Для наглядности построим график:



В сортировке прямым слиянием мы разделяем исходный массив на четыре части, каждую из которых сортируем отдельно, а затем сливаем эти части вместе в отсортированный массив. Этот алгоритм имеет сложность $O(n \cdot \log n)$ и показывает хорошие результаты на больших массивах. Из результатов замеров видно, что на небольших массивах этот алгоритм работает быстрее, но при увеличении размера массива разница во времени выполнения между сортировками уменьшается.

С другой стороны, естественная сортировка предполагает, что исходный массив уже частично отсортирован. Если это так, то сортировка занимает меньше времени, чем сортировка прямым слиянием.

Исходя из результатов замеров времени, можно сделать вывод, что на всех массивах естественная сортировка работает быстрее. Также следует учитывать, что результаты замеров времени могут зависеть от многих факторов, включая аппаратное и программное обеспечение, поэтому необходимо проводить более широкие и точные эксперименты, чтобы сделать окончательные выводы.

3 ВЫВОД

Цель работы заключалась в освоении приемов сортировки данных из файлов. В процессе работы были рассмотрены два алгоритма сортировки: сортировка прямым слиянием и естественная сортировка. Для каждого алгоритма был написан соответствующий код на языке C++ и проведены замеры времени выполнения на различных объемах данных.

Результаты работы показали, что сортировка прямым слиянием имеет сложность $O(n \cdot \log n)$ и хорошо работает на больших объемах данных, в то время как естественная сортировка предназначена для уже отсортированных массивов и может работать медленнее в общем случае. Однако, при работе с небольшими массивами естественная сортировка может быть более эффективной.

Таким образом, цель работы была успешно достигнута, были освоены приемы сортировки данных из файлов, рассмотрены два алгоритма сортировки и проведены замеры времени выполнения. Полученные результаты могут быть полезны для дальнейшей работы с данными из файлов и выбора наиболее подходящего алгоритма сортировки в зависимости от задачи.

4 СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин М.Л. и др. Введение в структуры и алгоритмы обработки данных. Ч. 1 - учебное пособие, 2022.pdf
2. ПР-1.1 (Теоретическая сложность алгоритма).
3. Лекционные материалы - СиАОД-1Файл
4. <https://metanit.com/>
5. <https://ru.cppreference.com/w/>
6. Оценка сложности алгоритмов <https://habr.co/>
7. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.
8. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.
9. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.
10. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.
11. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL:<http://algotlist.manual.ru/> (дата обращения 15.03.2022).
12. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: <https://habr.com/ru/hub/algorithms/> (дата обращения 15.03.2022).
13. НОУ ИНТУИТ | Технопарк Mail.ru Group: Алгоритмы и структуры данных[Электронныйресурс].URL:<https://intuit.ru/studies/courses/3496/738/info> (дата обращения 15.03.2022).