МИНОБРНАУКИ РОСИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №1.4

Тема:

Алгоритмы внешних сортировок

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Павлов Н.С.

Группа: ИКБО-30-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

Освоить приёмы сортировки данных из файлов

1 ЗАДАНИЕ №1

1.1 ФОРМУЛИРОВКА ЗАДАЧИ

Разработать программу и применить алгоритм внешней сортировки прямого слияния к сортировке файла данных индивидуального варианта по значению ключевого поля.

1. Реализовать функцию сортировки (возможно, с вспомогательными функциями) и основную подпрограмму main.
2. Отладить программу, протестировать на примере.
3. Предварительно подготовить файл данных в соответствии с вариантом (не менее 32 записей).
4. Адаптировать программу для сортировки файла с записями, протестировать на подготовленном ранее файле.
5. Определить практическую сложность алгоритма для файлов с увеличивающимся количеством записей (8, 16, 32). Сформировать таблицу результатов, указав количество записей и время сортировки.

Персональный вариант: Сведения о спортсмене: Фамилия, Имя, Дата рождения, Вес, Рост, Пол

Ключевое поле: Рост

1.2 РАБОТА С АЛГОРИТМОМ

1.2.1 Математическая модель решения

Фаза разделения:

1. Открыть файл A как входной.
2. Открыть файлы B и C как выходные (для записи).
3. Считываемые из A записи попеременно записываем в файлы B и C.
4. Закрываем файлы A, B, C.

Фаза слияния:

1. Открыть файл A как выходной (для записи).
2. Открываем файлы B и C 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 B и C.
5. Меньшая запись записывается в файл A, и считывается очередная запись из того файла, запись которого была переписана в файл A.
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 A.
8. Пункты с 4 по 7 повторяются до тех пор, пока не будет достигнут конец одного из файлов B и C. Тогда оставшиеся записи из другого файла переписываются в файл A.
9. Закрываются файлы A B C.

Сортировка завершается тогда, когда длина порции достигнет n

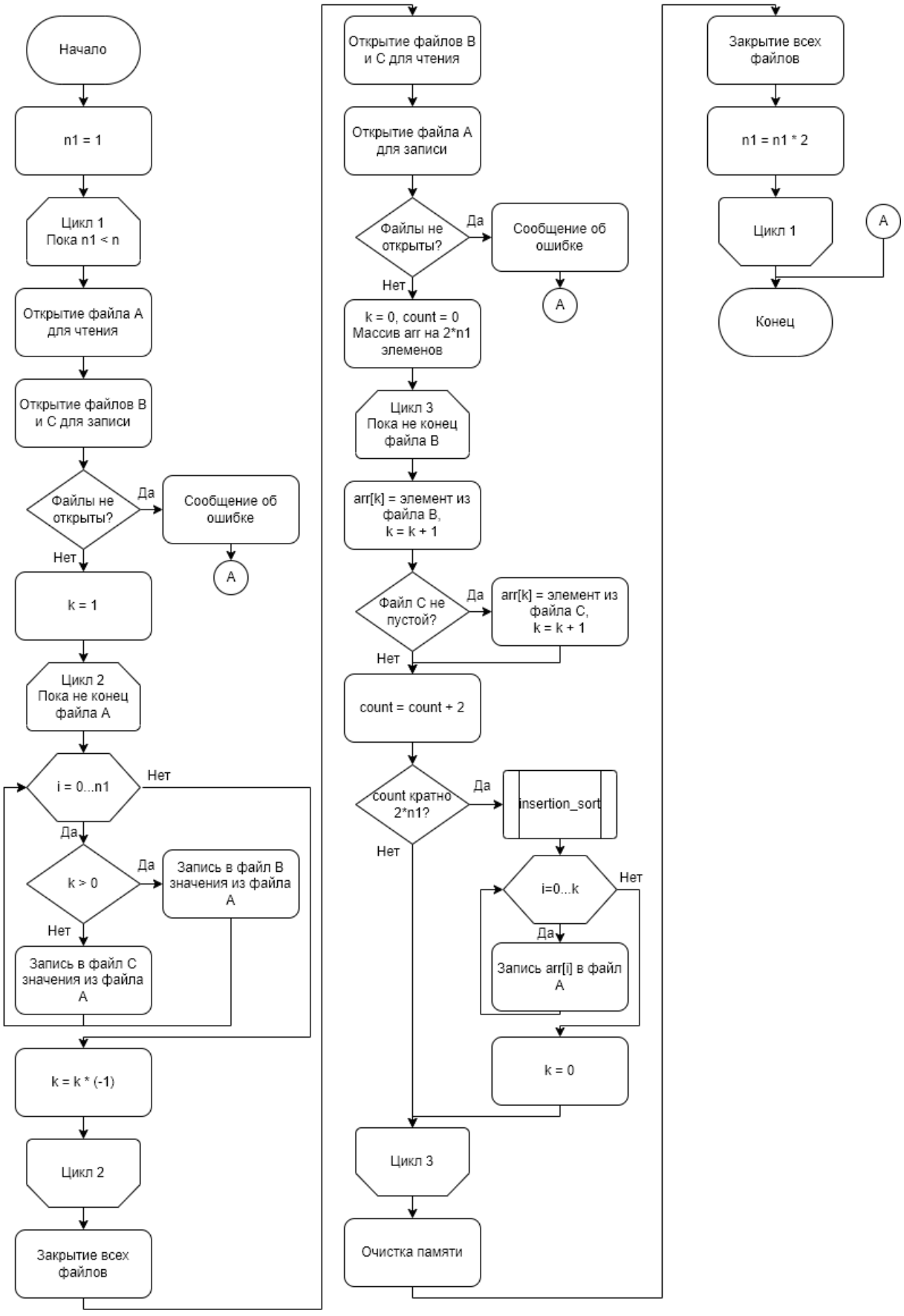

Рисунок 1 – Блок-схема алгоритма сортировки прямого слияния

1.2.2 Реализация для сортировки чисел

Реализуем алгоритм сортировки прямого слияния на языке C++ (рис.2-4) и проведем тестирование на примере (рис.5).

```
int n1 = 1;
while (n1 < n) {
    ifstream finA1;
    ofstream foutB, foutC;
    finA1.open(A1);
    foutB.open(B);
    foutC.open(C);
    if (!finA1.is_open() || !foutB.is_open() || !foutC.is_open()) {
        cout << "Ошибка открытия файла" << endl;
        exit(0);
    }
    else {
        int k = 1;
        while (!finA1.eof()) {
            string x;
            for (int i = 0; i < n1; i++) {
                if (finA1 >> x) {
                    if (k > 0) {
                        foutB << " " << x;
                    }
                    else {
                        foutC << " " << x;
                    }
                }
            }
            k *= -1;
        }
    }
    finA1.close();
    foutB.close();
    foutC.close();
```

Рисунок 2 – Реализация сортировки. Часть 1

```
ifstream finB, finC;
ofstream foutA;
foutA.open(A);
finB.open(B);
finC.open(C);
if (!foutA.is_open() || !finB.is_open() || !finC.is_open()) {
    cout << "Ошибка открытия файла" << endl;
    exit(0);
}
else {
    string b, c;
    int k = 0, count = 0;
    int* arr = new int[n1 * 2];
    while (!finB.eof()) {
        finB >> b;
        arr[k] = stoi(b);
        k++;
        if (finC >> c) {
            arr[k] = stoi(c);
            k++;
        }
        count += 2;
        if (count % (n1 * 2) == 0) {
            insertion_sort(arr, k);
            for (int i = 0; i < k; i++) {
                foutA << arr[i] << " ";
            }
            k = 0;
        }
    }
    delete[] arr;
}
```

Рисунок 3 – Реализация сортировки. Часть 2

```
    foutA.close();
    finB.close();
    finC.close();
    n1 *= 2;
    A1=A;
}
```

Рисунок 4 – Реализация сортировки. Часть 3

```
8 2 13 4 15 6 9 11 3 7 5 10 1 12 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Рисунок 5 – Исходный и отсортированный файл

1.2.3 Адаптация под файл варианта

Путем небольших преобразований адаптируем нашу функцию сортировки под персональный вариант (рис.6-9) и протестируем ее (рис.10-11).

```
int n1 = 1;
while (n1 < n) {
    ifstream finA1;
    ofstream foutB, foutC;
    finA1.open(A1);
    foutB.open(B);
    foutC.open(C);
    if (!finA1.is_open() || !foutB.is_open() || !foutC.is_open()) {
        cout << "Ошибка открытия файла" << endl;
    }
    else {
        int k = 1, l = 0;
        while (!finA1.eof()) {
            string x;
```

Рисунок 6 – Реализация сортировки для персонального варианта (Часть 1)

```
            for (int i = 0; i < n1; i++) {
                getline(finA1, x);
                l++;
                if (k > 0) {
                    if (l == 1) {
                        foutB << x;
                    }
                    else {
                        foutB << endl << x;
                    }
                }
                else {
                    if (l == n) {
                        foutC << x;
                    }
                    else {
                        foutC << x << endl;
                    }
                }
            }
            k *= -1;
        }
    }
    finA1.close();
    foutB.close();
    foutC.close();
```

Рисунок 7 – Реализация сортировки для персонального варианта (Часть 2)

```
ifstream finB, finC;
ofstream foutA;
foutA.open(A);
finB.open(B);
finC.open(C);
if (!foutA.is_open() || !finB.is_open() || !finC.is_open()) {
    cout << "Ошибка открытия файла" << endl;
}
else {
    string b, c;
    int k = 0, count = 0;
    Person* person = new Person[n1 * 2];
    while (!finB.eof() && count != n) {
        finB >> person[k].name >> person[k].surname >> person[k].birthday
            >> person[k].weight >> person[k].height >> person[k].gender;
        k++;
        finC >> person[k].name >> person[k].surname >> person[k].birthday
            >> person[k].weight >> person[k].height >> person[k].gender;
        k++;
        count += 2;
        if (count % (n1 * 2) == 0) {
            insertion_sort_2(person, k);
            bool f = true;
            for (int i = 0; i < k; i++) {
                foutA << person[i].name << " " << person[i].surname << " " << person[i].birthday << " "
                    << person[i].weight << " " << person[i].height << " " << person[i].gender << endl;
            }
            k = 0;
        }
    }
    delete[] person;
}
```

Рисунок 8 – Реализация сортировки для персонального варианта (Часть 3)

```
    foutA.close();
    finB.close();
    finC.close();
    n1 *= 2;
    A1 = A;
}
```

Рисунок 9 – Реализация сортировки для персонального варианта (Часть 4)

Рисунок 10-11 – Файл персонального варианта до и после сортировки

1.2.4 Оценка эмпирической сложности

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 1.

Таблица 1 – Сводная таблица результатов

n	**T(n), мс**
8	6
16	17
32	33

2 ЗАДАНИЕ №2

2.1 ПОСТАНОВКА ЗАДАЧИ

Разработать программу и применить алгоритм сортировки естественного слияния к сортировке файла с данными варианта (файл уже должен быть подготовлен в задании 1).

1. Реализовать функцию сортировки (возможно, с вспомогательными функциями) и основную подпрограмму main.
2. Отладить программу, протестировать на примере.
3. Адаптировать программу для сортировки файла с записями, протестировать на подготовленном ранее файле.
4. Сформировать таблицу результатов, указав количество записей и время сортировки.

2.2 РАБОТА С АЛГОРИТМОМ

2.2.1 Математическая модель решения

Алгоритм:

1. Определить размер свободной оперативной памяти для выгрузки в нее серии из файла. В программе создаем массив для хранения серии mas.
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 mas. Отсортировать в массиве методом внутренней сортировки и записать в файл B.
5. Считать следующую последовательность данных в количестве достаточном для размещения в массиве mas. 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 ai<ai+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.

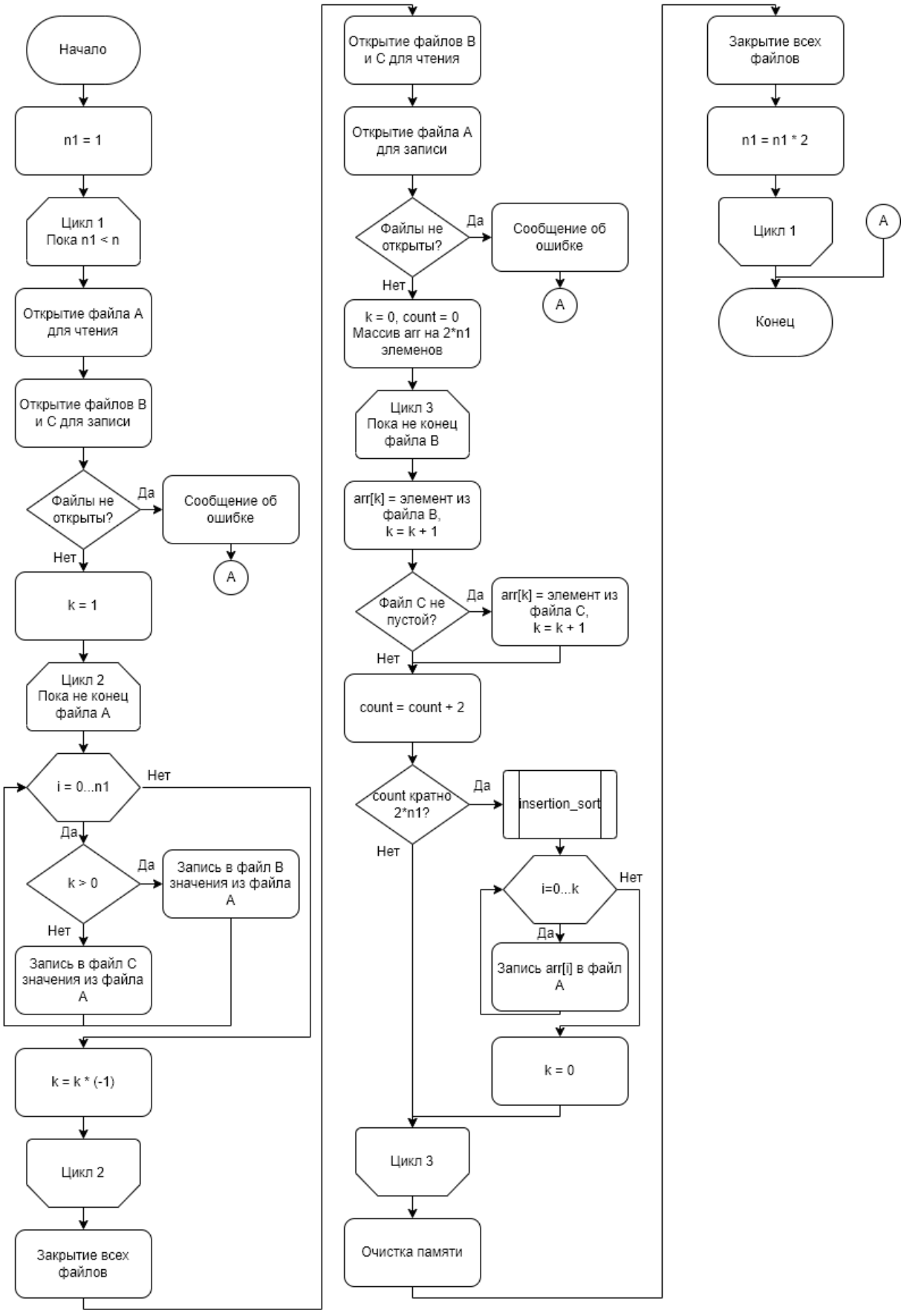

Рисунок 12 – Блок-схема алгоритма сортировки прямого слияния

2.2.2 Реализация для сортировки чисел

Реализуем алгоритм сортировки естественного слияния на языке C++ (рис.13-15) и проведем тестирование на примере (рис.16).

```
int n1 = 1;
while (n1 < n) {
    ifstream finA1;
    ofstream foutB, foutC;
    finA1.open(A1);
    foutB.open(B);
    foutC.open(C);
    if (!finA1.is_open() || !foutB.is_open() || !foutC.is_open()) {
        cout << "Ошибка открытия файла" << endl;
        exit(0);
    }
    else {
        int k = 1;
        while (!finA1.eof()) {
            string x;
            for (int i = 0; i < n1; i++) {
                if (finA1 >> x) {
                    if (k > 0) {
                        foutB << " " << x;
                    }
                    else {
                        foutC << " " << x;
                    }
                }
            }
            k *= -1;
        }
    }
    finA1.close();
    foutB.close();
    foutC.close();
```

Рисунок 13 – Реализация сортировки. Часть 1

```
ifstream finB, finC;
ofstream foutA;
foutA.open(A);
finB.open(B);
finC.open(C);
if (!foutA.is_open() || !finB.is_open() || !finC.is_open()) {
    cout << "Ошибка открытия файла" << endl;
    exit(0);
}
else {
    string b, c;
    int k = 0, count = 0;
    int* arr = new int[n1 * 2];
    while (!finB.eof()) {
        finB >> b;
        arr[k] = stoi(b);
        k++;
        if (finC >> c) {
            arr[k] = stoi(c);
            k++;
        }
        count += 2;
        if (count % (n1 * 2) == 0) {
            insertion_sort(arr, k);
            for (int i = 0; i < k; i++) {
                foutA << arr[i] << " ";
            }
            k = 0;
        }
    }
    delete[] arr;
}
```

Рисунок 14 – Реализация сортировки. Часть 2

```
    foutA.close();
    finB.close();
    finC.close();
    n1 *= 2;
    A1=A;
}
```

Рисунок 15 – Реализация сортировки. Часть 3

```
8 2 13 4 15 6 9 11 3 7 5 10 1 12 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Рисунок 16 – Исходный и отсортированный файл

2.2.3 Адаптация под файл варианта

Путем небольших преобразований адаптируем нашу функцию сортировки под персональный вариант (рис.17-20) и протестируем ее (рис.21-22).

```
int n1 = 1;
while (n1 < n) {
    ifstream finA1;
    ofstream foutB, foutC;
    finA1.open(A1);
    foutB.open(B);
    foutC.open(C);
    if (!finA1.is_open() || !foutB.is_open() || !foutC.is_open()) {
        cout << "Ошибка открытия файла" << endl;
    }
    else {
        int k = 1, l = 0;
        while (!finA1.eof()) {
            string x;
```

Рисунок 17 – Реализация сортировки для персонального варианта (Часть 1)

```
            for (int i = 0; i < n1; i++) {
                getline(finA1, x);
                l++;
                if (k > 0) {
                    if (l == 1) {
                        foutB << x;
                    }
                    else {
                        foutB << endl << x;
                    }
                }
                else {
                    if (l == n) {
                        foutC << x;
                    }
                    else {
                        foutC << x << endl;
                    }
                }
            }
            k *= -1;
        }
    }
    finA1.close();
    foutB.close();
    foutC.close();
```

Рисунок 18 – Реализация сортировки для персонального варианта (Часть 2)

```
ifstream finB, finC;
ofstream foutA;
foutA.open(A);
finB.open(B);
finC.open(C);
if (!foutA.is_open() || !finB.is_open() || !finC.is_open()) {
    cout << "Ошибка открытия файла" << endl;
}
else {
    string b, c;
    int k = 0, count = 0;
    Person* person = new Person[n1 * 2];
    while (!finB.eof() && count != n) {
        finB >> person[k].name >> person[k].surname >> person[k].birthday
            >> person[k].weight >> person[k].height >> person[k].gender;
        k++;
        finC >> person[k].name >> person[k].surname >> person[k].birthday
            >> person[k].weight >> person[k].height >> person[k].gender;
        k++;
        count += 2;
        if (count % (n1 * 2) == 0) {
            insertion_sort_2(person, k);
            bool f = true;
            for (int i = 0; i < k; i++) {
                foutA << person[i].name << " " << person[i].surname << " " << person[i].birthday << " "
                    << person[i].weight << " " << person[i].height << " " << person[i].gender << endl;
            }
            k = 0;
        }
    }
    delete[] person;
}
```

Рисунок 19 – Реализация сортировки для персонального варианта (Часть 3)

```
    foutA.close();
    finB.close();
    finC.close();
    n1 *= 2;
    A1 = A;
}
```

Рисунок 20 – Реализация сортировки для персонального варианта (Часть 4)

Рисунок 21-22 – Файл персонального варианта до и после сортировки

2.2.2 Оценка эмпирической сложности

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 2.

Таблица 2 – Сводная таблица результатов

n	**T(n), мс**
8	9
16	19
32	37

3 ВЫВОД

В результате проделанной работы были освоены приемы сортировки данных из файлов.

СПИСОК ИНФОРМАЦТОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных. Часть 1. Сложность алгоритмов. Сортировки. Линейные структуры данных. Поиск в таблице. : учеб. пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова ; МИРЭА – Российский технологический университет, 2022. – 109 с. – ISBN 978-5-7339-1612-5.

2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01-01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с.