|  |
| --- |
| 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: Гендриксон А.А. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ №1 4](#_30j0zll)

[2.1 Формулировка задачи 4](#_1fob9te)

[2.2 Описание выполнения прямого слияния и его блок-схема 4](#_2et92p0)

[2.3 Реализация алгоритма и тестирование 6](#)

[2.4 Адаптация программы 13](#_2s8eyo1)

[2.5 Практическая сложность алгоритма 23](#_3rdcrjn)

[2.6 Вывод по заданию один 23](#_26in1rg)

[3 ЗАДАНИЕ №2 25](#_lnxbz9)

[3.1 Формулировка задачи 25](#_35nkun2)

[3.2 Описание выполнения естественного слияния и его блок-схема 25](#_1ksv4uv)

[3.3 Реализация алгоритма и тестирования 26](#_2jxsxqh)

[3.4 Адаптация программы 38](#_1y810tw)

[3.5 Практическая сложность алгоритма 51](#_2xcytpi)

[3.6 Вывод по заданию два 51](#_1ci93xb)

[4 ВЫВОДЫ 53](#_3whwml4)

[5 ЛИТЕРАТУРА 54](#_2bn6wsx)

# **1 ЦЕЛЬ**

Освоить приёмы сортировки данных из файлов

# **2 ЗАДАНИЕ №1**

## **2.1 Формулировка задачи**

В списке №6

Вариант №6

Персональный вариант: Сведения об успеваемости одного студента по одной дисциплине: Номер зачетной книжки, Шифр группы, Название дисциплины, Дата получения оценки, Оценка, Фамилия преподавателя

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

1) Реализовать функцию сортировки (возможно, с вспомогательными функциями) и основную подпрограмму main.

2) Отладить программу, протестировать на примере из п.2.

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.

## **2.2 Описание выполнения прямого слияния и его блок-схема**

Алгоритм прямого слияния, также известный как "merge sort", заключается в разбиении исходного массива на две равные части, сортировке каждой из них отдельно, а затем слиянии отсортированных подмассивов в один отсортированный массив.

Разбиение исходного массива на две равные части.

Рекурсивная сортировка каждой из частей (подмассивов).

Слияние отсортированных подмассивов в один отсортированный массив:

Создание нового массива, в который будут помещаться элементы двух подмассивов.

Сравнение элементов двух подмассивов и добавление минимального элемента в новый массив.

Повторение предыдущего шага, пока все элементы из обоих подмассивов не будут добавлены в новый массив.

Результирующий отсортированный массив представляет собой результат выполнения алгоритма прямого слияния.

Реализуем данный алгоритм в виде блок схемы(рис.1).

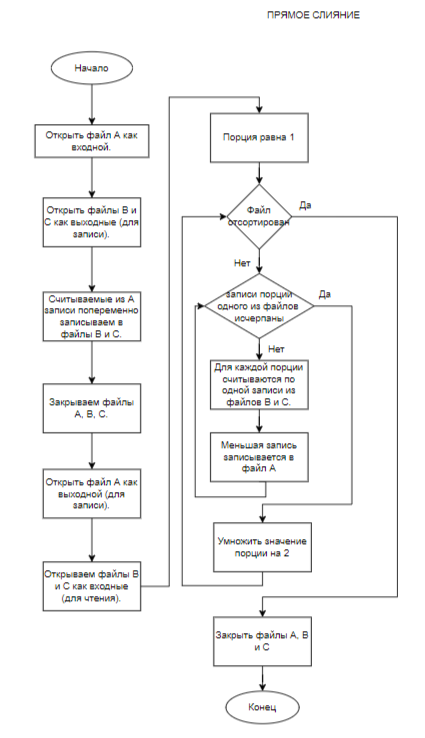


Рисунок 1 – Блок-схема алгоритма прямого слияния

## **2.3 Реализация алгоритма и тестирование**

Реализуем код на С++(блок кода 1) и проведем тестирование(рис.2), основанное на примере из файла.

Пример:  
 Пусть файл А содержит данные, которые подлежат сортировке:

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

Для реализации алгоритма будем использовать два файла, в которые будем разливать данные файла: файл В и файл С.

Сначала разбиваем по одному элементу:

B: 8 13 15 9 3 5 1 14

C: 2 4 6 11 7 10 12

Сливаем в упорядоченные двойки:

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

Разливаем по два:

B: 2 8 6 15 3 7 1 12

C: 4 13 9 11 5 10 14

Сливаем в упорядоченные четверки и т. д. пока длина порции не станет равной длине массива.

| #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> using namespace std;  *// Перегрузка оператора вывода для вывода вектора целых чисел* ostream& operator << (ostream& lhv, const vector<int>& rhv) {  const auto size = rhv.size();  lhv << "(";  for (int i = 0; i < size; ++i) {  lhv << rhv[i];  if(i+1 < size) lhv << " ";  }  lhv << ")";  return lhv; }  *// Класс для внешней сортировки слиянием* class SExternalMergeSort { public:  *// Конструктор класса, принимающий имя файла и количество "путей" сортировки по умолчанию*  explicit SExternalMergeSort(const string& filename, unsigned int way\_count = 2) : seriesLength\_{ 0 }, wayCount\_ { way\_count }, inputFilename\_{ filename } {  if (wayCount\_ < 2) wayCount\_ = 2;   };   *// Виртуальный деструктор класса*  virtual ~SExternalMergeSort() = default;    *// Метод сортировки*  void sort(fstream& input);  private:  *// Метод для вычисления длины серии чисел во входном файле*  void computeSeriesLength(fstream& input);   *// Метод для получения внешних файлов*  vector<fstream> getExternalFiles();   *// Метод для повторного открытия внешних файлов*  void reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode);   *// Метод для удаления внешних файлов*  void removeExternalFiles();   *// Метод для повторного открытия входного файла*  void reopenInputFile(fstream& input, ios\_base::openmode mode);   *// Метод для разделения серий чисел*  void split(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length);   *// Метод для слияния серий чисел*  void merge(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length);   unsigned int seriesLength\_; *// Длина серии чисел*  unsigned int wayCount\_; *// Количество "путей" сортировки*  string inputFilename\_; *// Имя входного файла* };  *// Главная функция программы* int main() {  const auto input\_file = "A\_example\_simple"; *// Имя входного файла*  fstream input(input\_file, ios\_base::in | ios\_base::out); *// Открытие входного файла для чтения и записи*   *// Проверка успешности открытия файла*  if (!input) {  perror("Error with input file");  return EXIT\_FAILURE;  }   *// Создание объекта класса SExternalMergeSort и запуск сортировки*  SExternalMergeSort(input\_file).sort(input);   input.close();   system("pause");     return EXIT\_SUCCESS;  }  *// Метод для вычисления длины серии чисел во входном файле* void SExternalMergeSort::computeSeriesLength(fstream& input\_series) {  this -> seriesLength\_ = 0; *// Инициализация длины серии чисел*  int value = 0; *// Переменная для хранения считанного значения*   *// Подсчёт числа элементов в серии*  while (input\_series >> value) {  this -> seriesLength\_++;  } }  *// Метод для получения внешних файлов* vector<fstream> SExternalMergeSort::getExternalFiles() {  vector<fstream> ext(this->wayCount\_); *// Вектор внешних файлов*   *// Создание внешних файлов*  for (unsigned int i = 0; i < this -> wayCount\_; ++i) {  const auto filename = to\_string(i + 1);  ofstream f(filename);  ext[i] = fstream(filename);  }  return ext; }  *// Метод для повторного открытия внешних файлов* void SExternalMergeSort::reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode) {  *// Повторное открытие каждого внешнего файла*  for (int i = 0; i < ext.size(); ++i) {  ext[i].close(); *// Закрытие файла*  const auto filename = to\_string(i + 1);  ext[i].open(filename, mode); *// Открытие файла в указанном режиме*  } }  *// Метод для удаления внешних файлов* void SExternalMergeSort::removeExternalFiles() {  *// Удаление каждого внешнего файла*  for (unsigned int i = 0; i < this->wayCount\_; ++i) {  const auto filename = to\_string(i + 1);  const auto removeResult = remove(filename.c\_str());  cout << "Remove external file (" << filename << "): " << boolalpha << removeResult << "\n"; *// Вывод результата удаления файла*  } }  *// Метод для повторного открытия входного файла* void SExternalMergeSort::reopenInputFile(fstream& input, ios\_base::openmode mode) {  input.close(); *// Закрытие входного файла*  input.open(this->inputFilename\_, mode); *// Открытие входного файла в указанном режиме* }  *// Метод для разделения серий чисел* void SExternalMergeSort::split(fstream& input\_series, vector<fstream>& ext, const unsigned int subgroup\_length) {  *// Пока не достигнут конец входного файла*  while (!input\_series.eof()) {  *// Разделение чисел на подгруппы и запись во внешние файлы*  for (auto&& f: ext) {  int value = 0; *// Переменная для хранения считанного значения*  unsigned int step = 0; *// Шаг разделения*   *// Чтение чисел и запись во внешний файл*  while (input\_series >> value) {  f << value << " ";  cout << value << "\t"; *// Вывод числа в консоль*  ++step; *// Увеличение шага*   *// Проверка достижения конца подгруппы*  if (step == subgroup\_length) break;  }   cout << " | "; *// Вывод разделителя*  }   cout << "\n"; *// Переход на новую строку*  }   *// Сброс буфера для каждого внешнего файла*  for (auto&& f : ext) {  f.flush();  } }  *// Метод для слияния серий чисел* void SExternalMergeSort::merge(fstream& input\_series, vector<fstream>& ext, const unsigned int subgroup\_length) {  if (subgroup\_length == 0) return; *// Если длина подгруппы равна 0, завершить метод*   vector<int> buf; *// Буфер для хранения чисел для слияния*  buf.reserve(subgroup\_length \* this->wayCount\_); *// Резервирование места в буфере для чисел*   const unsigned int fullGroupCount = this -> seriesLength\_ / subgroup\_length / this -> wayCount\_; *// Полное число групп*  const unsigned int add = this -> seriesLength\_ % this -> wayCount\_ == 0 ? 0 : 1; *// Дополнительная группа, если нечётное число элементов*  const unsigned int stepsCount = fullGroupCount + add; *// Общее число шагов слияния*   unsigned int step = 0; *// Текущий шаг*   *// Пока не достигнут конец слияния*  while (step < stepsCount) {   *// Слияние чисел из внешних файлов в буфер*  for (auto& f: ext) {  int value = 0; *// Переменная для хранения считанного значения*  unsigned int substep = 0; *// Шаг слияния*   *// Считывание чисел из файла в буфер*  while (substep < subgroup\_length && f >> value) {  buf.push\_back(value); *// Добавление числа в буфер*  ++substep; *// Увеличение шага*  }  }    std::sort(begin(buf), end(buf)); *// Сортировка буфера*    const auto bufLength = buf.size(); *// Длина буфера*   *// Запись отсортированных чисел из буфера во входной файл*  for(int i = 0; i < bufLength; ++i) {  input\_series << buf[i] << " "; *// Запись числа во входной файл*  cout << buf[i] << " "; *// Вывод числа в консоль*  }   cout << "\t"; *// Вывод разделителя*   buf.clear(); *// Очистка буфера*  buf.resize(0); *// Установка размера буфера в 0*   step++; *// Увеличение шага*   }   cout << "\n"; *// Переход на новую строку* }  *// Метод для запуска сортировки* void SExternalMergeSort::sort(fstream& input\_series) {  this -> computeSeriesLength(input\_series); *// Вычисление длины серии чисел*   *// Если длина серии меньше или равна 1, завершить метод*  if (this->seriesLength\_ <= 1) return;   vector<fstream> ext = this -> getExternalFiles(); *// Получение внешних файлов*    *// Цикл сортировки*  for (unsigned int subgroup\_length = 1; subgroup\_length < this->seriesLength\_; subgroup\_length \*= 2) {  *// Повторное открытие входного и внешних файлов для чтения и записи*  reopenInputFile(input\_series, ios\_base::in);  reopenExternalFiles(ext, ios\_base::out);   cout << "Subgroup with length " << to\_string(subgroup\_length) << ":\n"; *// Вывод сообщения о длине подгруппы*   *// Разделение серий чисел на подгруппы и запись во внешние файлы*  this -> split(input\_series, ext, subgroup\_length);   *// Повторное открытие входного и внешних файлов для чтения и записи*  reopenInputFile(input\_series, ios\_base::out);  reopenExternalFiles(ext, ios\_base::in);   cout << "Merge to groups with length " << to\_string(subgroup\_length) << "\n"; *// Вывод сообщения о слиянии подгрупп*   *// Слияние подгрупп чисел*  this -> merge(input\_series, ext, subgroup\_length);   }  input\_series.flush(); *// Сброс входного файла*   this -> removeExternalFiles(); *// Удаление внешних файлов* } |
| --- |

Блок кода 1 – Прямое слияние

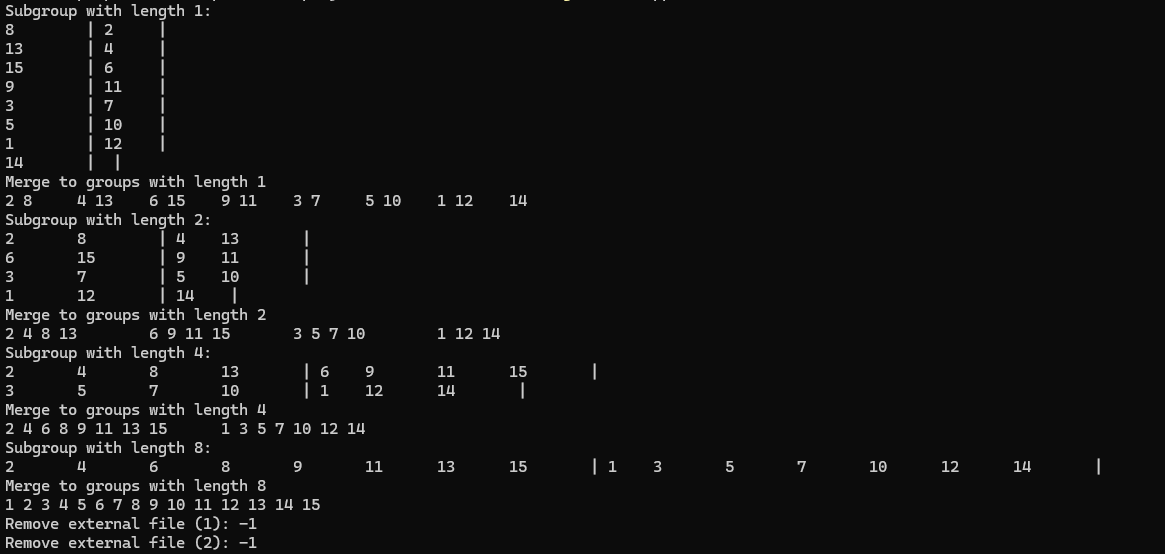


Рисунок 2 - Тестирование программы с входными данными из примера

## **2.4 Адаптация программы**

Адаптируем программу к данным по условию индивидуального варианта. Для этой цели будет использован пользовательский заголовочный файл "s.h"(блок кода 2) и main.cpp(блок кода 3).

| #pragma once #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> #include <functional>  *// SType - тип сортируемых данных* *// DSep - разделитель данных в файле* template<class SType, char DSep = ' '> class SExternalMergeSort { public:  *// way\_count -- количество путей (вспомогательных файлов) сортировки*  explicit SExternalMergeSort(const std::string& filename, unsigned int way\_count = 2) : seriesLength\_{ 0 }, wayCount\_{ way\_count }, inputFilename\_{ filename } {  if (wayCount\_ < 2) wayCount\_ = 2;  };  virtual ~SExternalMergeSort() = default;   template<class Compare>  void sort(std::fstream& input, Compare&& cmp = [](const SType& a, const SType& b) { return a < b; });  private:  *// вычисление длины серии*   void computeSeriesLength(std::fstream& input);   *// имя для вспомогательного файла*  std::string getExternalFilename(const unsigned int way);   *// получить список из wayCount\_ вспомогательных файлов*  std::vector<std::fstream> getExternalFiles();   *// закрыть и открыть вспомогательные файлы для корректного ввода-вывода*  void reopenExternalFiles(std::vector<std::fstream>& ext, std::ios\_base::openmode mode);   *// удалить вспомогательные файлы*  void removeExternalFiles();   *// закрыть и открыть файл входных данных для корректного ввода-вывода*  void reopenInputFile(std::fstream& input, std::ios\_base::openmode mode);   *// разделение серии во вспомогательные файлы подгруппами длинны subgroup\_length*  void split(std::fstream& input, std::vector<std::fstream>& ext, const unsigned int subgroup\_length);   *// слияние из вспомогательных файлов подгруппами длинны subgroup\_length*  template<class Compare>  void merge(std::fstream& input, std::vector<std::fstream>& ext, const unsigned int subgroup\_length, Compare& cmp);   unsigned int seriesLength\_; *// длина серии вычисляется при запуске сортировки*   unsigned int wayCount\_; *// количество путей (вспомогательных файлов) сортировки*   std::string inputFilename\_; };  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::computeSeriesLength(std::fstream& input\_series) {  this->seriesLength\_ = 0;  SType value{};  while (input\_series >> value)  {  this->seriesLength\_++;  } }  template<class SType, char DSep> std::string SExternalMergeSort<SType, DSep>::getExternalFilename(const unsigned int way) {  return "/" + std::to\_string(way + 1); }  template<class SType, char DSep> std::vector<std::fstream> SExternalMergeSort<SType, DSep>::getExternalFiles() {  std::vector<std::fstream> ext(this->wayCount\_);  for (unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = this->getExternalFilename(i);  std::ofstream f(filename);  ext[i] = std::fstream(filename);  }  return ext; }  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::reopenExternalFiles(std::vector<std::fstream>& ext, std::ios\_base::openmode mode) {  for (int i = 0; i < ext.size(); ++i)  {  ext[i].close();   const auto filename = getExternalFilename(i);  ext[i].open(filename, mode);  } }  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::removeExternalFiles() {  for (unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = getExternalFilename(i);  const auto removeResult = std::remove(filename.c\_str());  std::cout << "Remove external file (" << filename << "): " << std::boolalpha << removeResult << std::endl;  } }  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::reopenInputFile(std::fstream& input, std::ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); }  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::split(std::fstream& input\_series, std::vector<std::fstream>& ext, const unsigned int subgroup\_length) {  while (!input\_series.eof())  {  for (auto&& f : ext)  {  SType value{};  unsigned int step = 0;  while (input\_series >> value)  {  f << value << DSep;  std::cout << value << "\t";  ++step;  if (step == subgroup\_length) break;  }  std::cout << " | ";  }  std::cout << std::endl;  } }  template<class SType, char DSep> template<class Compare> void SExternalMergeSort<SType, DSep>::merge(  std::fstream& input\_series, std::vector<std::fstream>& ext,  const unsigned int subgroup\_length,  Compare& cmp) {  if (subgroup\_length == 0) return;   std::vector<SType> buf;  buf.reserve(subgroup\_length \* this->wayCount\_);   *// количество полных групп в путях*  const unsigned int fullGroupCount = this->seriesLength\_ / subgroup\_length / this->wayCount\_;   *// учет возможного отсутствия полной последней группы в некоторых путях*  const unsigned int add = (this->seriesLength\_ / subgroup\_length) % this->wayCount\_ == 0 ? 0 : 1;   *// количество шагов алгоритма*  const unsigned int stepsCount = fullGroupCount + add;  unsigned int step = 0;  while (step < stepsCount)  {  for (auto& f : ext)  {  SType value{};  unsigned int substep = 0;  while (substep < subgroup\_length && f >> value)  {  buf.push\_back(value);  ++substep;  }  }   std::sort(std::begin(buf), std::end(buf), cmp);   const auto bufLength = buf.size();  for (int i = 0; i < bufLength; ++i)  {  input\_series << buf[i] << DSep;  std::cout << buf[i] << DSep;  }  std::cout << "\t";   buf.clear();  buf.resize(0);  step++;  }  std::cout << std::endl; }  template<class SType, char DSep> template<class Compare> void SExternalMergeSort<SType, DSep>::sort(std::fstream& input\_series, Compare&& cmp) {  this->computeSeriesLength(input\_series);  if (this->seriesLength\_ <= 1)  {  return;  }   auto&& cmp\_ = std::forward<Compare>(cmp);  std::vector<std::fstream> ext = this->getExternalFiles();   *// Сортируем пока длина подгруппы меньше длины серии*  for (unsigned int subgroup\_length = 1; subgroup\_length < this->seriesLength\_; subgroup\_length \*= 2)  {  reopenInputFile(input\_series, std::ios\_base::in);  reopenExternalFiles(ext, std::ios\_base::out);   std::cout << "Subgroup with length " << std::to\_string(subgroup\_length) << ":" << std::endl;  this->split(input\_series, ext, subgroup\_length);   reopenInputFile(input\_series, std::ios\_base::out);  reopenExternalFiles(ext, std::ios\_base::in);   std::cout << "Merge to groups with length " << std::to\_string(subgroup\_length) << std::endl;  this->merge(input\_series, ext, subgroup\_length, cmp\_);  }   input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 2 - заголовочный файл "s.h"

| #include <fstream> #include <vector> #include "s.h" #include <string> #include <sstream> using namespace std; *// Функция разделения строки по символу-разделителю* std::vector<std::string>& split(const std::string& s, char delim, std::vector<std::string>& elems) {  std::stringstream ss(s);  std::string item;  while (std::getline(ss, item, delim)) {  elems.push\_back(item);  }  return elems; }  *//*  struct Consumer {  std::string number;  std::string group;  std::string name;  std::string data;  std::string score;  std::string surname; };  istream& operator>>(istream& lhv, Consumer& rhv) {  string tmp;  getline(lhv, tmp);   vector<string> e;  split(tmp, ',', e);  const auto elementsCount = e.size();   if (elementsCount >= 6)  {  rhv.number = e[0];  rhv.group = e[1];  rhv.name = e[2];  rhv.data = e[3];  rhv.score = e[4];  rhv.surname = e[5];  }   return lhv; }   std::ostream& operator<<(std::ostream& lhv, Consumer& rhv) {  lhv << rhv.number << "," << rhv.group << "," << rhv.name << "," << rhv.data << "," << rhv.score << "," << rhv.surname;  return lhv; }  int main(int argc, char\*\* argv) {  setlocale(LC\_CTYPE, "Rus");   const auto input\_file = "A";  std::fstream input(input\_file, std::ios\_base::in | std::ios\_base::out);  if (!input)  {  perror("Error with input file");  return EXIT\_FAILURE;  }   SExternalMergeSort<Consumer, '\n'>(input\_file).sort(input, [](const Consumer& a, const Consumer& b) { return a.number < b.number; });  input.close();  system("pause");  return EXIT\_SUCCESS; } |
| --- |

Блок кода 3 - Файл main.cpp

Протестируем адаптированный код. Результаты тестирования будут продемонстрированы на рисунке 3-7.

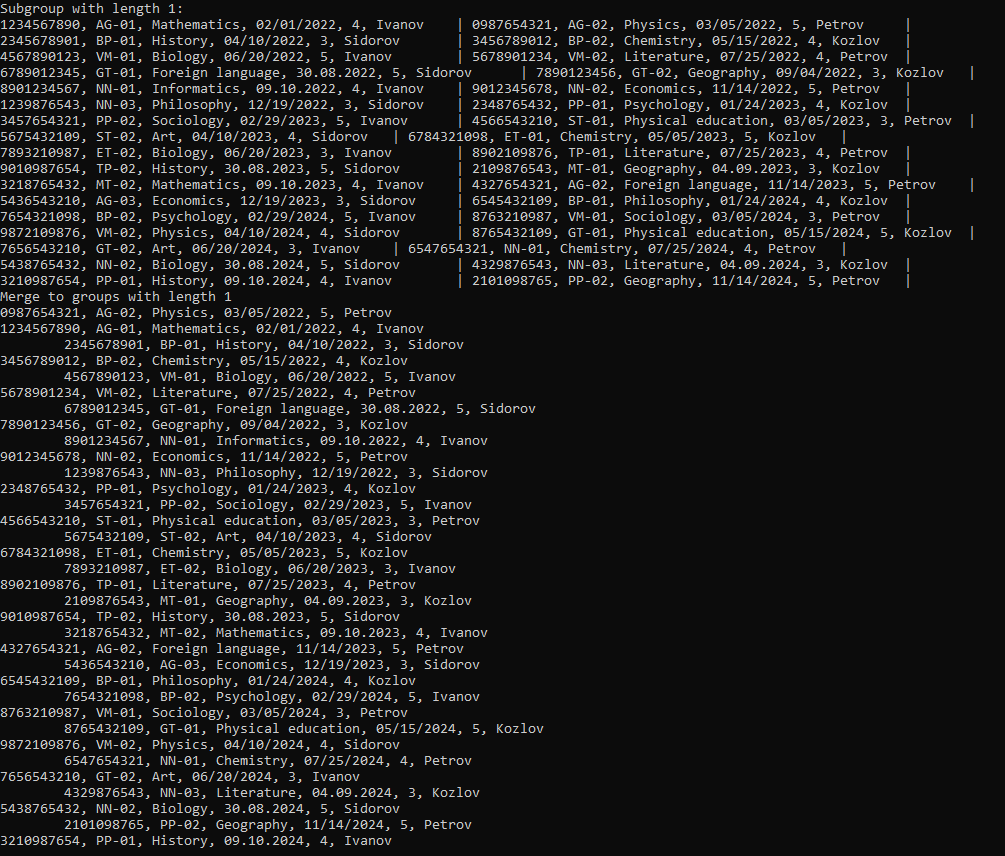


Рисунок 3 - Тестирование адаптированной программы

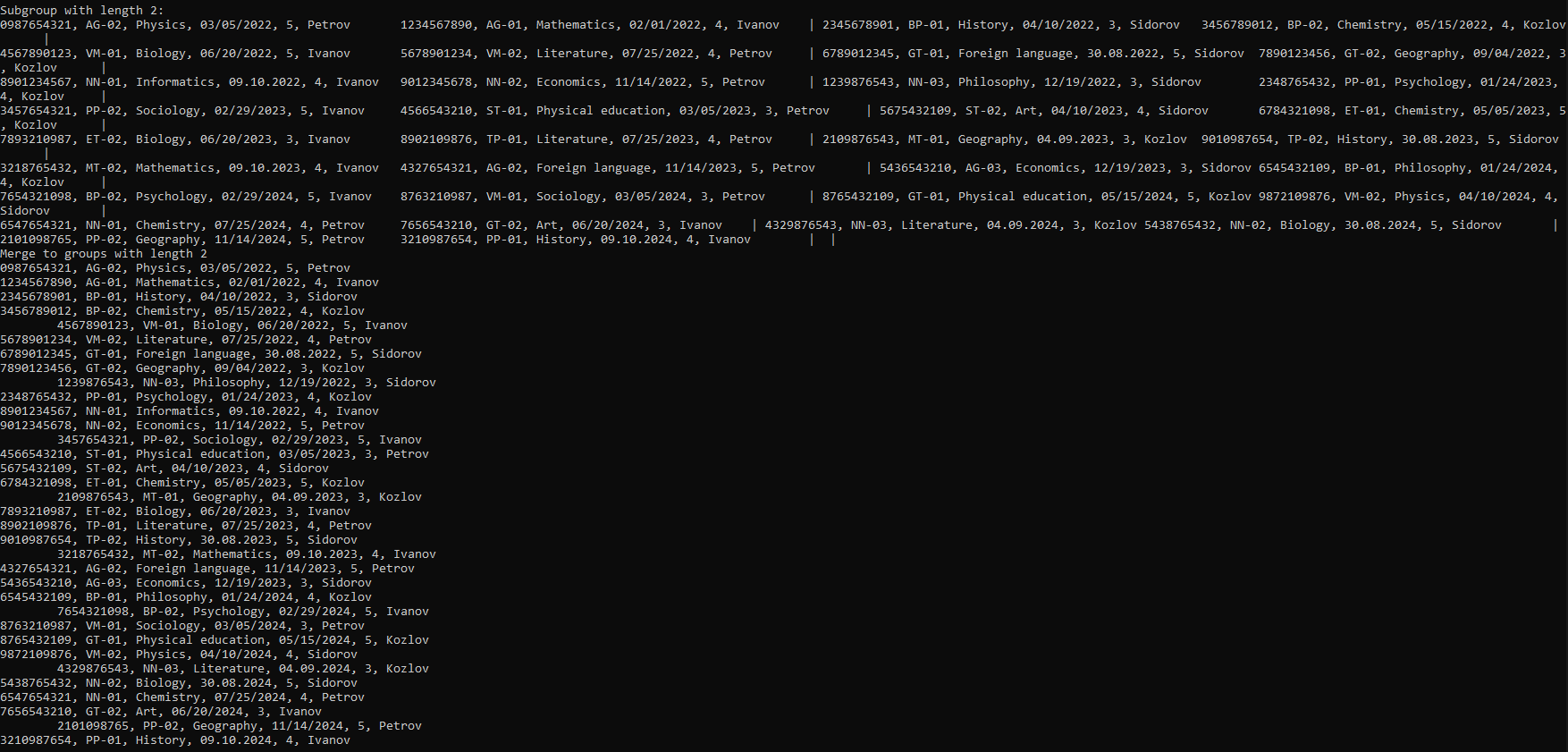
****

Рисунок 4 - Тестирование адаптированной программы

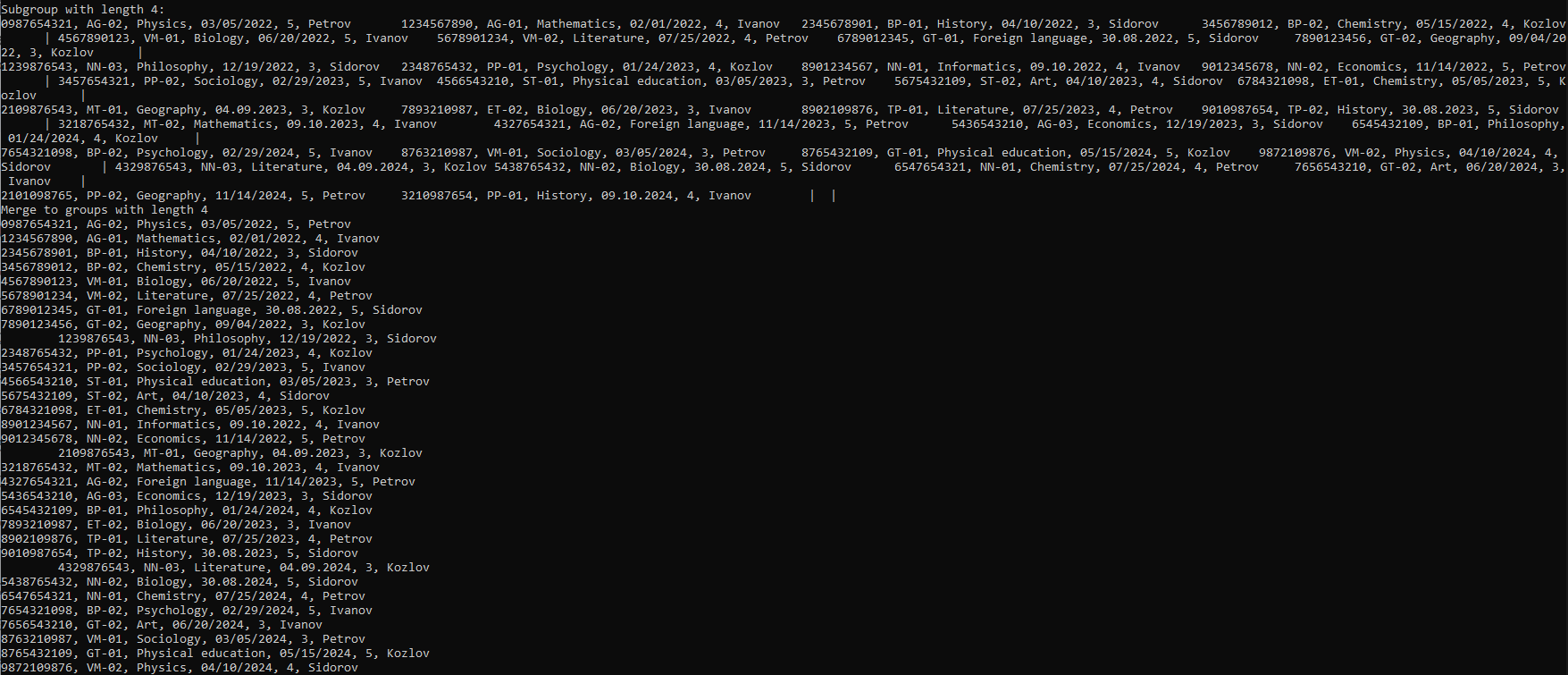


Рисунок 5 - Тестирование адаптированной программы



Рисунок 6 - Тестирование адаптированной программы



Рисунок 7 - Тестирование адаптированной программы

## **2.5 Практическая сложность алгоритма**

Продемонстрируем подсчёт времени на разном количестве записей в файле данных в таблице 1.

Таблица 1 - Сводная таблица результатов

| **Количество записей** | **Время(мкс)** |
| --- | --- |
| 8 | 31647 |
| 16 | 425478 |
| 32 | 1905464 |

## **2.6 Вывод по заданию один**

Описание: Алгоритм прямого слияния, также известный как сортировка слиянием, является эффективным алгоритмом сортировки, который основан на методе "разделяй и властвуй". Основная идея состоит в том, чтобы разделить исходный массив на более мелкие подмассивы до тех пор, пока каждый подмассив не будет состоять из одного элемента. Затем происходит поэлементное слияние этих подмассивов в порядке возрастания или убывания, что приводит к получению отсортированного массива.

Реализация: Для реализации алгоритма прямого слияния необходимо написать функции для разделения массива, слияния подмассивов и сортировки. Рекурсивно вызывая эти функции, мы добьемся правильной сортировки массива.

Эффективность: Алгоритм прямого слияния обладает временной сложностью O(n log n) в худшем, лучшем и среднем случаях. Он является стабильным алгоритмом сортировки, что означает, что порядок равных элементов не меняется в отсортированном массиве. Этот алгоритм эффективен для больших массивов, так как не зависит от начального порядка элементов и имеет оптимальную сложность. Однако, он требует дополнительной памяти для хранения промежуточных подмассивов.

# **3 ЗАДАНИЕ №2**

## **3.1 Формулировка 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 из п.4.

3) Адаптировать программу для сортировки файла с записями, протестировать на подготовленном ранее файле.

4) Сформировать таблицу результатов, указав количество записей и время сортировки.

## **3.2 Описание выполнения естественного слияния и его блок-схема**

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.8).

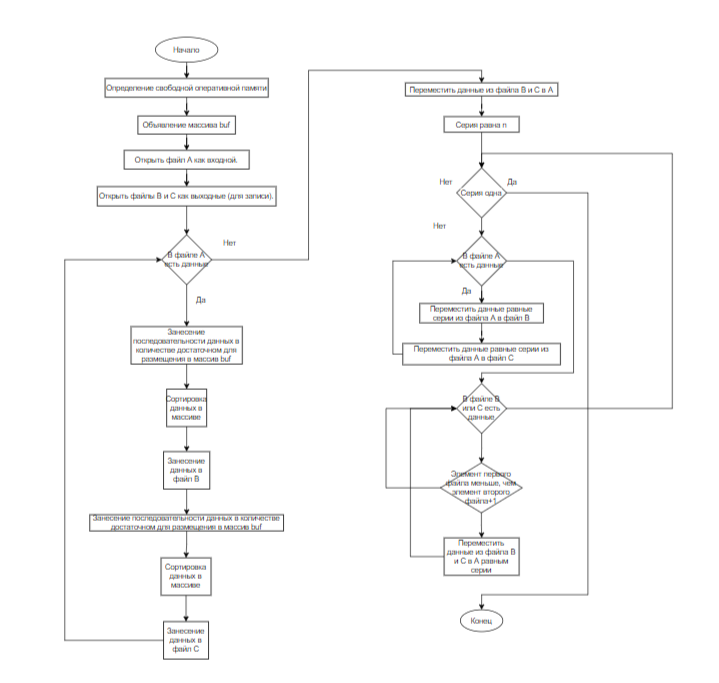


Рисунок 8 – Блок-схема алгоритма естественного слияния

## **3.3 Реализация алгоритма и тестирования**

Пусть есть файл А, содержащий записи с ключами:

17 31 5 59 13 41 43 67 11 23 29 47 3 7 71 2 19 57 37 61

Выделим серии, завершая запятой, чтобы было нагляднее:

17 31’ 5 59’ 13 41 43 67’ 11 23 29 47’ 3 7 71’ 2 19 57’ 37 61

Получилось 7 серий.

Разделим файл на два файла В и С, переписывая в них поочередно по серии:

B: 17 31’ 13 41 43 67’ 3 7 71’ 37 61

C: 5 59’ 11 23 29 47’ 2 19 57

Сольем файлы в файл А, сливая серии в упорядоченные серии

А:5 17 31 59’ 11 13 23 29 41 43 47 67’2 3 7 19 57 71’37 61

Опять разольем в В и С поочередно переписывая серии

B: 5 17 31 59’2 3 7 19 57 71

C: 11 13 23 29 41 43 47 67’37 61

Сливаем в файл А по сериям

А: 5 11 13 17 23 29 31 41 43 47 59 67’2 3 7 19 37 57 61 71

Разливаем …..

и продолжаем до тех пор, пока в массив А не будет переписана серия длины n.

| #include <iostream>  #include <fstream> #include <algorithm> #include <vector> #include <string> #include <chrono> using namespace std;  ostream& operator << (ostream& lhv, const vector<int>& rhv) { *// Перегрузка оператора вывода для вектора целых чисел*   const auto size = rhv.size(); *// Получение размера вектора*   lhv << "("; *// Вывод открывающей скобки*   for (int i = 0; i < size; ++i) { *// Цикл по всем элементам вектора*  lhv << rhv[i]; *// Вывод текущего элемента*   if(i+1 < size) lhv << " "; *// Если не последний элемент, добавляем пробел*  }   lhv << ")"; *// Вывод закрывающей скобки*   return lhv; *// Возвращаем ссылку на поток вывода*   }  constexpr auto kBlockSeparator = "|"; *// Определение разделителя блоков*  class NExternalMergeSort { *// Класс для внешней сортировки методом слияния*    public:    explicit NExternalMergeSort(const string& filename, unsigned int way\_count = 2) : seriesLength\_{ 0 }, wayCount\_{ way\_count }, inputFilename\_{ filename } {  if (wayCount\_ < 2) wayCount\_ = 2; *// Установка количества путей сортировки (по умолчанию 2)*  };   virtual ~NExternalMergeSort() = default; *// Виртуальный деструктор класса*   void sort(fstream& input); *// Метод для сортировки*   private:    void computeSeriesLength(fstream& input); *// Метод для вычисления длины серии чисел во входном файле*    vector<fstream> getExternalFiles(); *// Метод для создания временных файлов*    void reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode); *// Метод для повторного открытия временных файлов*    void removeExternalFiles(); *// Метод для удаления временных файлов*    void reopenInputFile(fstream& input, ios\_base::openmode mode); *// Метод для повторного открытия входного файла*    bool fullEOF(vector<fstream>& ext); *// Метод для проверки, достигнут ли конец всех временных файлов*    bool isSorted(fstream& input); *// Метод для проверки отсортированности файла*    void split(fstream& input, vector<fstream>& ext); *// Метод для разделения исходного файла на серии чисел*    void merge(fstream& input, vector<fstream>& ext); *// Метод для слияния отсортированных серий чисел*    unsigned int seriesLength\_; *// Длина серии чисел в исходном файле*    unsigned int wayCount\_ = 2; *// Количество путей сортировки (по умолчанию 2)*   string inputFilename\_; *// Имя входного файла*  };  int main() { *// Главная функция программы*   const auto input\_file = "A\_example\_natural"; *// Имя входного файла*   fstream input(input\_file, ios\_base::in | ios\_base::out); *// Открытие входного файла для чтения и записи*   if (!input) { *// Проверка успешности открытия файла*  perror("Error with input file"); *// Вывод сообщения об ошибке*  return EXIT\_FAILURE; *// Возвращение значения, указывающего на неудачное завершение программы*  }   NExternalMergeSort(input\_file).sort(input); *// Создание объекта класса сортировки и вызов метода сортировки*   input.close(); *// Закрытие входного файла*   system("pause"); *// Ожидание ввода пользователя перед закрытием консоли*  return EXIT\_SUCCESS; *// Возвращение значения, указывающего на успешное завершение программы*  }  void NExternalMergeSort::computeSeriesLength(fstream& input\_series) { *// Метод для вычисления длины серии чисел во входном файле*   this -> seriesLength\_ = 0; *// Инициализация длины серии чисел*   int value = 0; *// Переменная для считывания чисел из файла*   while (input\_series >> value) { *// Чтение чисел из файла и подсчет их количества*  this -> seriesLength\_++; *// Увеличение счетчика чисел*  }  }  vector<fstream> NExternalMergeSort::getExternalFiles() { *// Метод для создания временных файлов*   vector<fstream> ext(this->wayCount\_); *// Создание вектора временных файлов*   for(unsigned int i = 0; i < this->wayCount\_; ++i) { *// Цикл по количеству путей сортировки*   const auto filename = to\_string(i + 1); *// Формирование имени временного файла*   ofstream f(filename); *// Создание временного файла*   ext[i] = fstream(filename); *// Добавление временного файла в вектор*   }   return ext; *// Возвращение вектора временных файлов*  }  void NExternalMergeSort::reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode) { *// Метод для повторного открытия временных файлов*   for (int i = 0; i < ext.size(); ++i) { *// Цикл по всем временным файлам*   ext[i].close(); *// Закрытие текущего временного файла*   const auto filename = to\_string(i + 1); *// Формирование имени временного файла*   ext[i].open(filename, mode); *// Открытие временного файла в указанном режиме*   }  }  void NExternalMergeSort::removeExternalFiles() { *// Метод для удаления временных файлов*   for (unsigned int i = 0; i < this->wayCount\_; ++i) { *// Цикл по всем временным файлам*   const auto filename = to\_string(i + 1); *// Формирование имени временного файла*  const auto removeResult = remove(filename.c\_str()); *// Удаление временного файла*   cout << "Remove external file (" << filename << "): " << boolalpha << removeResult << "\n"; *// Вывод информации о результате удаления*   }  }  void NExternalMergeSort::reopenInputFile(fstream& input, ios\_base::openmode mode) { *// Метод для повторного открытия входного файла*  input.close(); *// Закрытие входного файла*  input.open(this->inputFilename\_, mode); *// Открытие входного файла в указанном режиме* }  bool NExternalMergeSort::fullEOF(vector<fstream>& ext) { *// Метод для проверки, достигнут ли конец всех временных файлов*   bool fullEOF = true; *// Флаг, указывающий на достижение конца всех файлов*   for (auto& f : ext) { *// Цикл по всем временным файлам*   fullEOF &= f.eof(); *// Проверка, достигнут ли конец текущего файла*   if (!fullEOF) break; *// Если не достигнут конец, выход из цикла*   }   return fullEOF; *// Возвращение значения флага*  }  bool NExternalMergeSort::isSorted(fstream& input) { *// Метод для проверки отсортированности файла*   int value = 0; *// Переменная для считывания чисел из файла*  input >> value; *// Считывание первого числа из файла*   int oldValue = value; *// Сохранение предыдущего значения для сравнения*   bool sorted = false; *// Флаг отсортированности файла*   while (input >> value) { *// Чтение чисел из файла и проверка их отсортированности*   sorted = oldValue <= value; *// Проверка, чтобы текущее значение было больше или равно предыдущему*   oldValue = value; *// Обновление предыдущего значения*   if (!sorted) break; *// Если не отсортировано, выход из цикла*   }   return sorted; *// Возвращение значения флага отсортированности*  }  void NExternalMergeSort::split(fstream& input\_series, vector<fstream>& ext) { *// Метод для разделения исходного файла на серии чисел*   int value = 0; *// Переменная для считывания чисел из файла*   int oldValue = 0; *// Предыдущее считанное значение*   unsigned int way = 0; *// Индекс текущего временного файла*   bool firstStep = true; *// Флаг первого шага разделения*   while(input\_series >> value) { *// Чтение чисел из исходного файла*   if (firstStep) { *// Если первый шаг*   firstStep = false; *// Сброс флага первого шага*   ext[way] << value << " "; *// Запись числа в текущий временный файл*  oldValue = value; *// Сохранение текущего значения*  cout << value << " "; *// Вывод числа на экран*   continue; *// Пропуск оставшейся части цикла*   }   if (oldValue > value) { *// Если предыдущее значение больше текущего*   ext[way].seekg(-1, ios\_base::cur); *// Перемещение указателя на один символ назад*  ext[way] << kBlockSeparator << " "; *// Запись разделителя блоков в текущий временный файл*  cout << kBlockSeparator; *// Вывод разделителя блоков на экран*  way = (way + 1) % wayCount\_; *// Переход к следующему временному файлу*   }   ext[way] << value << " "; *// Запись числа в текущий временный файл*  oldValue = value; *// Сохранение текущего значения*  cout << value << " "; *// Вывод числа на экран*   }   cout << "\n"; *// Переход на новую строку*  }  void NExternalMergeSort::merge(fstream& input\_series, vector<fstream>& ext) { *// Метод для слияния отсортированных серий чисел*   while(!this->fullEOF(ext)) { *// Пока не достигнут конец всех временных файлов*   vector<int> oldValues(this->wayCount\_, { 0 }); *// Вектор для хранения текущих значений из каждого временного файла*  vector<bool> eogs(this->wayCount\_, { false }); *// Вектор для хранения флагов достижения конца временных файлов*   for (unsigned int way = 0; way < this->wayCount\_; ++way) { *// Цикл по всем временным файлам*   int value = 0; *// Переменная для считывания чисел из временного файла*   if (ext[way] >> value) oldValues[way] = value; *// Если чтение прошло успешно, сохраняем значение*  else eogs[way] = true; *// Если достигнут конец файла, устанавливаем флаг*   }   bool fullEOG = false; *// Флаг завершения всех временных файлов*    while (!fullEOG) { *// Пока не завершены все временные файлы*   int wayWithMin = -1; *// Индекс временного файла с минимальным значением*   for (unsigned int checkedWay = 0; checkedWay < this->wayCount\_; ++checkedWay) { *// Цикл по всем временным файлам*   if (eogs[checkedWay] || ext[checkedWay].eof()) continue; *// Если достигнут конец файла, пропускаем текущий файл*   wayWithMin = checkedWay; *// Устанавливаем индекс минимального файла*   }   if (-1 == wayWithMin) break; *// Если все файлы завершены, выходим из цикла*   for (unsigned int checkedWay = 0; checkedWay < this->wayCount\_; ++checkedWay) { *// Цикл по всем временным файлам*   if (eogs[checkedWay] || ext[checkedWay].eof()) continue; *// Если достигнут конец файла, пропускаем текущий файл*   if (oldValues[checkedWay] < oldValues[wayWithMin]) wayWithMin = checkedWay; *// Если текущее значение меньше, обновляем индекс минимального файла*   }    input\_series << oldValues[wayWithMin] << " "; *// Запись минимального значения в исходный файл*  cout << oldValues[wayWithMin] << " "; *// Вывод минимального значения на экран*   int value = 0; *// Переменная для считывания чисел из временного файла*   if (ext[wayWithMin] >> value) { *// Если чтение прошло успешно*  oldValues[wayWithMin] = value; *// Сохраняем значение*  } else { *// Если достигнут конец файла*  eogs[wayWithMin] = true; *// Устанавливаем флаг*  ext[wayWithMin].clear(); *// Сбрасываем флаги ошибок*  }    fullEOG = true; *// Устанавливаем флаг завершения всех файлов*   for (auto eog : eogs) { *// Проверка, завершены ли все файлы*   fullEOG &= eog; *// Обновление значения флага*   if (!fullEOG) break; *// Если не завершены, выходим из цикла*   }   }   for (unsigned int way = 0; way < this->wayCount\_; ++way) { *// Цикл по всем временным файлам*   if (eogs[way]) { *// Если файл завершен*  ext[way].get(); *// Пропускаем разделитель блоков*  ext[way].get(); *// Пропускаем пробел*  }   }   }   cout << "\n"; *// Переход на новую строку*  }  void NExternalMergeSort::sort(fstream& input\_series) { *// Метод для сортировки серий чисел*   this -> computeSeriesLength(input\_series); *// Вычисление длины серии чисел*   if (this->seriesLength\_ <= 1) return; *// Если серия содержит одно число или менее, выход из метода*   vector<fstream> ext = this -> getExternalFiles(); *// Создание временных файлов*   reopenInputFile(input\_series, ios\_base::in); *// Повторное открытие входного файла для чтения*   bool sorted = isSorted(input\_series); *// Проверка отсортированности файла*   while(!sorted) { *// Пока файл не отсортирован*   reopenInputFile(input\_series, ios\_base::in); *// Повторное открытие входного файла для чтения*  reopenExternalFiles(ext, ios\_base::out); *// Повторное открытие временных файлов для записи*   cout << "Subgroup: \n"; *// Вывод информации о начале разделения*   this -> split(input\_series, ext); *// Разделение исходного файла на серии чисел*   reopenInputFile(input\_series, ios\_base::out); *// Повторное открытие входного файла для записи*  reopenExternalFiles(ext, ios\_base::in); *// Повторное открытие временных файлов для чтения*   cout << "Merge: \n"; *// Вывод информации о начале слияния*   this -> merge(input\_series, ext); *// Слияние отсортированных серий чисел*    reopenInputFile(input\_series, ios\_base::in); *// Повторное открытие входного файла для чтения*  sorted = isSorted(input\_series); *// Проверка отсортированности файла*   cout << "Sorted: " << boolalpha << sorted << "\n"; *// Вывод информации о завершении сортировки*   }   input\_series.flush(); *// Сброс буфера входного файла*   this -> removeExternalFiles(); *// Удаление временных файлов*  } |
| --- |

Блок кода 4 - Естественное слияние

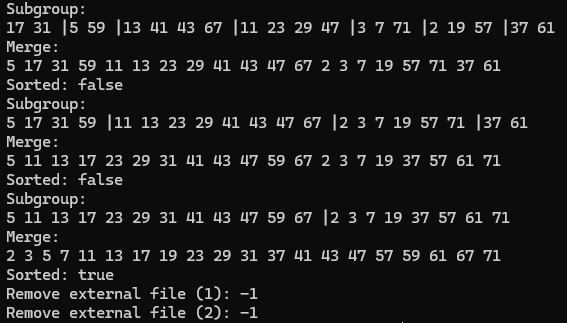


Рисунок 9 - Тестирование программы с входными данными из примера

## **3.4 Адаптация программы**

Адаптируем программу к данным по условию индивидуального варианта. Для этой цели будет использован пользовательский заголовочный файл "a.h"(блок кода 5) и main.cpp(блок кода 6).

| #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string>   constexpr auto kBlockSeparator = "|";  *// SType - тип сортируемых данных* *// DSep - разделитель данных в файле* template<class SType, char DSep = ' '> class NExternalMergeSort { public:  *// way\_count -- количество путей (вспомогательных файлов) сортировки*  explicit NExternalMergeSort(const std::string& filename, unsigned int way\_count = 2) : seriesLength\_{ 0 }, wayCount\_{ way\_count }, inputFilename\_{ filename } {  if (wayCount\_ < 2) wayCount\_ = 2;  };  virtual ~NExternalMergeSort() = default;   template<class Compare>  void sort(std::fstream& input, Compare&& cmp = [](const SType& a, const SType& b) { return a < b; });  private:  *// вычисление длины серии*   void computeSeriesLength(std::fstream& input);   *// имя для вспомогательного файла*  std::string getExternalFilename(const unsigned int way);   *// получить список из wayCount\_ вспомогательных файлов*  std::vector<std::fstream> getExternalFiles();   *// закрыть и открыть вспомогательные файлы для корректного ввода-вывода*  void reopenExternalFiles(std::vector<std::fstream>& ext, std::ios\_base::openmode mode);   *// удалить вспомогательные файлы*  void removeExternalFiles();   *// закрыть и открыть файл входных данных для корректного ввода-вывода*  void reopenInputFile(std::fstream& input, std::ios\_base::openmode mode);   bool fullEOF(std::vector<std::fstream>& ext);   *// проверка отсортированности входных данных*  template<class Compare>  bool isSorted(std::fstream& input, Compare& cmp);   *// разделение серии в сортированные подгруппы по порядку следования входных данных*  template<class Compare>  void split(std::fstream& input, std::vector<std::fstream>& ext, Compare& cmp);   *// естественное слияние из вспомогательных файлов*  template<class Compare>  void merge(std::fstream& input, std::vector<std::fstream>& ext, Compare& cmp);   unsigned int seriesLength\_; *// длина серии вычисляется при запуске сортировки*   unsigned int wayCount\_ = 2; *// количество путей (вспомогательных файлов) сортировки*   std::string inputFilename\_; };  template<class SType, char DSep> void NExternalMergeSort<SType, DSep>::computeSeriesLength(std::fstream& input\_series) {  this->seriesLength\_ = 0;  SType value{};  while (input\_series >> value)  {  this->seriesLength\_++;  } }  template<class SType, char DSep> std::string NExternalMergeSort<SType, DSep>::getExternalFilename(const unsigned int way) {  return "/" + std::to\_string(way + 1); }  template<class SType, char DSep> std::vector<std::fstream> NExternalMergeSort<SType, DSep>::getExternalFiles() {  std::vector<std::fstream> ext(this->wayCount\_);  for(unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = getExternalFilename(i);  std::ofstream f(filename);  ext[i] = std::fstream(filename);  }  return ext; }  template<class SType, char DSep> void NExternalMergeSort<SType, DSep>::reopenExternalFiles(std::vector<std::fstream>& ext, std::ios\_base::openmode mode) {  for (int i = 0; i < ext.size(); ++i)  {  ext[i].close();   const auto filename = getExternalFilename(i);  ext[i].open(filename, mode);  } }  template<class SType, char DSep> void NExternalMergeSort<SType, DSep>::removeExternalFiles() {  for (unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = getExternalFilename(i);  const auto removeResult = std::remove(filename.c\_str());  std::cout << "Remove external file (" << filename << "): " << std::boolalpha << removeResult << std::endl;  } }  template<class SType, char DSep> void NExternalMergeSort<SType, DSep>::reopenInputFile(std::fstream& input, std::ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); }  template<class SType, char DSep> bool NExternalMergeSort<SType, DSep>::fullEOF(std::vector<std::fstream>& ext) {  bool fullEOF = true;  for (auto& f : ext)  {  fullEOF &= f.eof();  if (!fullEOF) break;  }   return fullEOF; }  template<class SType, char DSep> template<class Compare> bool NExternalMergeSort<SType, DSep>::isSorted(std::fstream& input, Compare& cmp) {  SType value{};  input >> value;  SType oldValue = value;   bool sorted = false;  while (input >> value)  {  const auto cmp1 = cmp(oldValue, value);  const auto cmp2 = cmp(value, oldValue);  sorted = cmp1 || (cmp1 == false && cmp2 == false);   oldValue = value;  if (!sorted) break;  }   return sorted; }  template<class SType, char DSep> template<class Compare> void NExternalMergeSort<SType, DSep>::split(  std::fstream& input\_series, std::vector<std::fstream>& ext,  Compare& cmp  ) {  SType value{};  SType oldValue{};  unsigned int way = 0;   if (input\_series >> value)  {  ext[way] << value << DSep;  oldValue = value;  std::cout << value << DSep;  }   while(input\_series >> value)  {  if (!cmp(oldValue, value) && cmp(value, oldValue))  {  way = (way + 1) % wayCount\_;  }   ext[way] << value << DSep;  oldValue = value;  std::cout << value << DSep;  }  std::cout << std::endl; }  template<class SType, char DSep> template<class Compare> void NExternalMergeSort<SType, DSep>::merge(  std::fstream& input\_series, std::vector<std::fstream>& ext,  Compare& cmp ) {  while(!this->fullEOF(ext))  {  std::vector<SType> oldValues(this->wayCount\_);  std::vector<bool> eogs(this->wayCount\_, { false }); *// end of group (аналог end of file)*  for (unsigned int way = 0; way < this->wayCount\_; ++way)  {  SType value{};  if(ext[way] >> value) oldValues[way] = value;  else eogs[way] = true;  }   bool fullEOG = false; *// флаг достижения конца групп во всех путях*   *// выбираем путь, с которого будем начинать слияние*  while (!fullEOG)  {  int wayWithMin = -1;   for (unsigned int checkedWay = 0; checkedWay < this->wayCount\_; ++checkedWay)  {  if (eogs[checkedWay] || ext[checkedWay].eof()) continue;  wayWithMin = checkedWay;  }  if (-1 == wayWithMin) break;   for (unsigned int checkedWay = 0; checkedWay < this->wayCount\_; ++checkedWay)  {  if (eogs[checkedWay] || ext[checkedWay].eof()) continue;  if (cmp(oldValues[checkedWay], oldValues[wayWithMin])) wayWithMin = checkedWay;  }   *// слияние*  input\_series << oldValues[wayWithMin] << DSep;  std::cout << oldValues[wayWithMin] << DSep;   *// след. значение*  SType value{};  if (ext[wayWithMin] >> value)   {  oldValues[wayWithMin] = value;  }  else  {  eogs[wayWithMin] = true;  ext[wayWithMin].clear(); *// очищаем сигнальные биты потока, чтобы иметь возможность читать дальнейшие группы*  }   *// проверка достижения конца групп*  fullEOG = true;  for (auto eog : eogs)  {  fullEOG &= eog;  if (!fullEOG) break;  }  }   *// переход через символы завершения группы*  for (unsigned int way = 0; way < this->wayCount\_; ++way)  {  if (eogs[way])   {  ext[way].get();  ext[way].get();  }  }  }   std::cout << std::endl; }  template<class SType, char DSep> template<class Compare> void NExternalMergeSort<SType, DSep>::sort(std::fstream& input\_series, Compare&& cmp) {   this->computeSeriesLength(input\_series);  if (this->seriesLength\_ <= 1)  {  return;  }   auto&& cmp\_ = std::forward<Compare>(cmp);  std::vector<std::fstream> ext = this->getExternalFiles();   reopenInputFile(input\_series, std::ios\_base::in);  bool sorted = isSorted(input\_series, cmp\_);  std::cout << "Sorted: " << std::boolalpha << sorted << std::endl;  while(!sorted)  {  reopenInputFile(input\_series, std::ios\_base::in);  reopenExternalFiles(ext, std::ios\_base::out);   std::cout << "Subgroup: " << std::endl;  this->split(input\_series, ext, cmp\_);   reopenInputFile(input\_series, std::ios\_base::out);  reopenExternalFiles(ext, std::ios\_base::in);   std::cout << "Merge: " << std::endl;  this->merge(input\_series, ext, cmp\_);   reopenInputFile(input\_series, std::ios\_base::in);  sorted = isSorted(input\_series, cmp\_);   std::cout << "Sorted: " << std::boolalpha << sorted << std::endl;  }   input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 5 - заголовочный файл "a.h"

| #include <fstream> #include <vector>  #include "a.h"   #include <string> #include <sstream> *// Функция разделения строки по символу-разделителю* using namespace std; *// Функция разделения строки по символу-разделителю* std::vector<std::string>& split(const std::string& s, char delim, std::vector<std::string>& elems) {  std::stringstream ss(s);  std::string item;  while (std::getline(ss, item, delim)) {  elems.push\_back(item);  }  return elems; }  *//*  struct Consumer {  std::string number;  std::string group;  std::string name;  std::string data;  std::string score;  std::string surname; };  istream& operator>>(istream& lhv, Consumer& rhv) {  string tmp;  getline(lhv, tmp);   vector<string> e;  split(tmp, ',', e);  const auto elementsCount = e.size();   if (elementsCount >= 6)  {  rhv.number = e[0];  rhv.group = e[1];  rhv.name = e[2];  rhv.data = e[3];  rhv.score = e[4];  rhv.surname = e[5];  }   return lhv; }   std::ostream& operator<<(std::ostream& lhv, Consumer& rhv) {  lhv << rhv.number << "," << rhv.group << "," << rhv.name << "," << rhv.data << "," << rhv.score << "," << rhv.surname;  return lhv; }  int main(int argc, char\*\* argv) {  setlocale(LC\_CTYPE, "rus");   const auto input\_file = "A";  std::fstream input(input\_file, std::ios\_base::in | std::ios\_base::out);  if (!input)  {  perror("Error with input file");  return EXIT\_FAILURE;  }   NExternalMergeSort<Consumer, '\n'>(input\_file).sort(input, [](const Consumer& a, const Consumer& b) { return a.surname < b.surname; });  input.close();  system("pause");  return EXIT\_SUCCESS; } |
| --- |

Блок кода 6 - Файл main.cpp

Протестируем адаптированный код Результаты тестирования будут продемонстрированы на рисунке 10 и 11. Тестирование было сокращено до двух рисунков, которые демонстрирует начало и конец сортировки, так как происходит однотипные изменения.

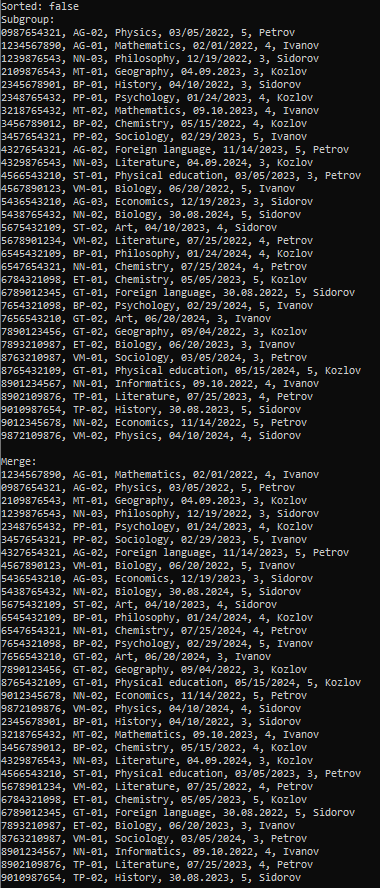


Рисунок 10 - Тестирование адаптированной программы

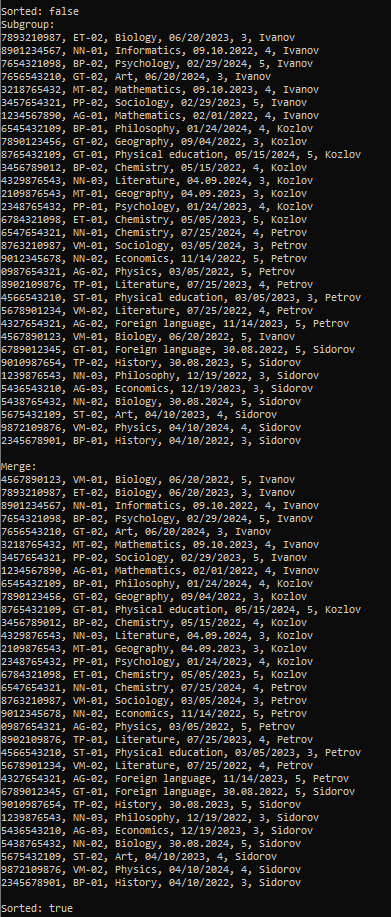


Рисунок 11 - Тестирование адаптированной программы

## **3.5 Практическая сложность алгоритма**

Продемонстрируем подсчёт времени на разном количестве записей в файле данных в таблице 2.

Таблица 2 - Сводная таблица результатов

| **Количество записей** | **Время(мкс)** |
| --- | --- |
| 8 | 327684 |
| 16 | 985365 |
| 32 | 2957563 |

## **3.6 Вывод по заданию два**

Алгоритм естественного слияния является методом сортировки, который использует процесс объединения отсортированных подсписков в один отсортированный список. Этот алгоритм эффективен благодаря учету структуры данных и упорядоченности элементов.

Реализация алгоритма естественного слияния включает в себя следующие шаги:

* Разделение входного списка на уже отсортированные подсписки.
* Постепенное слияние пар подсписков до полного объединения в один отсортированный список.

Эффективность алгоритма естественного слияния в основном зависит от исходного размера данных и структуры элементов. Из-за своей специфики, он может быть эффективен для частично отсортированных данных, поскольку не требует столько операций, как другие алгоритмы, например, слияние с использованием дополнительных структур данных.

Сравнивая эффективность естественного слияния с прямым слиянием, можно отметить, что прямое слияние имеет лучшую производительность при большом количестве данных и неупорядоченной структуре. Этот метод требует меньше операций и времени благодаря простоте и меньшей сложности реализации. Однако при частично отсортированных данных или структурах с выраженной упорядоченностью алгоритм естественного слияния может быть более эффективным в силу минимизации операций сравнения и перемещения элементов.

В итоге, выбор между алгоритмом естественного слияния и прямым слиянием зависит от конкретной задачи, данных и их структуры.

# **4 ВЫВОДЫ**

В процессе выполнения практической работы были достигнуты следующие цели:

* Изучены методы сортировки данных из файлов;
* Проведен анализ алгоритмов прямого и естественного слияния;
* Написаны программы для реализации алгоритмов;
* Проведено тестирование программ;
* Проведена адаптация под индивидуальные требования;
* Провели тестирование на разных количествах записей;
* Написан вывод о эффективности алгоритмов.

Таким образом, основную цель работы - изучение методов сортировки данных из файлов - можно считать достигнутой.

# **5 ЛИТЕРАТУРА**

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.

2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.

3. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.

4. Кораблин Ю.П. Структуры и алгоритмы обработки данных: учебно-методическое пособие / Ю.П. Кораблин, В.П. Сыромятников, Л.А. Скворцова. – М.: РТУ МИРЭА, 2020. — 219 с.

5. Кормен Т.Х. и др. Алгоритмы: построение и анализ, 3-е изд. – М.: ООО «И.Д.Вильямс», 2013. – 1328 с.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., - М.: Техносфера, 2018. – 416 с.

7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

8. Скиена С. Алгоритмы. Руководство по разработке, - 2-е изд. – СПб: БХВ-Петербург, 2011. – 720 с.

9. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд. – СПб: ООО «Альфа-книга», 2017. – 432 с.

10. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL: http://algolist.manual.ru/ (дата обращения 15.03.2022).

11. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 15.03.2022).