|  |
| --- |
| 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: Величко В.Д. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ №1 4](#_30j0zll)

[2.1 Формулировка задачи (В списке №10, Вариант 10) 4](#_1fob9te)

[2.2 Математическая модель решения алгоритма 4](#_2et92p0)

[2.2.1 Описание выполнения и блок-схема алгоритма прямого слияния 4](#_tyjcwt)

[2.2.2 Доказательство корректности циклов алгоритма прямого слияния 6](#_3dy6vkm)

[2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма прямого слияния 7](#_qsh70q)

2.3 Реализация алгоритма на языке C++ и проведение тестирования 8

[2.3.1 Реализация алгоритма прямого слияния на языке C++ 8](#_1t3h5sf)

[2.3.2 Тестирование 15](#_4d34og8)

[2.4 Адаптация программы к новым данным 16](#_2s8eyo1)

[2.6 Тестирование адаптированного кода 24](#_17dp8vu)

[2.7 Практическая сложность алгоритма 27](#_3rdcrjn)

[2.8 Вывод по заданию №1 27](#_26in1rg)

[3 ЗАДАНИЕ №2 28](#_lnxbz9)

[3.1 Формулировка задачи 28](#_35nkun2)

[3.2 Математическая модель решения алгоритма 28](#_1ksv4uv)

[3.2.1 Описание выполнения и блок-схема алгоритма естественного слияния 28](#_44sinio)

[3.2.2 Доказательство корректности циклов алгоритма естественного слияния 30](#_2jxsxqh)

[3.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма естественного слияния 31](#_3as4poj)

[3.3 Реализация алгоритма на языке C++ и проведение тестирования 32](#_2jxsxqh)

[3.3.1 Реализация алгоритма естественного слияния на языке C++ 32](#_z337ya)

[2.3.2 Тестирование 39](#_3j2qqm3)

[3.4 Адаптация программы к новым данным 40](#_1y810tw)

[3.6 Тестирование адаптированного кода 50](#_4i7ojhp)

[3.7 Практическая сложность алгоритма 61](#_2xcytpi)

[3.8 Вывод по заданию №2 61](#_1ci93xb)

[5 ВЫВОДЫ 63](#_3whwml4)

[6 ЛИТЕРАТУРА 64](#_2bn6wsx)

# **1 ЦЕЛЬ**

Освоить приёмы сортировки данных из файлов

# **2 ЗАДАНИЕ №1**

## **2.1 Формулировка задачи (В списке №5, Вариант 5)**

Разработать программу и применить алгоритм внешней сортировки прямого слияния к сортировке файла данных “Сведения о книге” по значению ключевого поля “Издательство”.

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 алгоритма прямого слияния и его брок-схема**

Внешняя сортировка представляет собой метод сортировки больших объемов данных, которые не помещаются в оперативную память и хранятся. Она используется тогда, когда использование обычных методов сортировки невозможно из-за ограничений памяти.

Принцип сортировки слиянием состоит в следующем: исходный набор данных разделяется на две равные части, каждая из которых сортируется отдельно, затем отсортированные части объединяются. Этот процесс повторяется до тех пор, пока не будет получен полностью отсортированный набор данных. По сути, массив разделяется на меньшие части, которые затем сортируются и объединяются.

Алгоритм сортировки прямым слиянием является простым методом внешней сортировки, основанным на процедуре слияния последовательностей элементов. На каждом шаге длина последовательностей (или серий) увеличивается в два раза. При этом исходный файл делится на два вспомогательных файла, которые затем сливаются обратно в исходный файл, образуя упорядоченные последовательности. Этот процесс повторяется до тех пор, пока весь файл не будет упорядочен.

Алгоритм сортировки простым слиянием имеет несколько признаков завершения, таких как длина серии, которая не меньше количества элементов в файле, и количество серий, равное 1.

Визуальное представление выполнения алгоритма сортировки прямым слиянием можно увидеть на блок-схеме, которая демонстрирует последовательность шагов алгоритма.

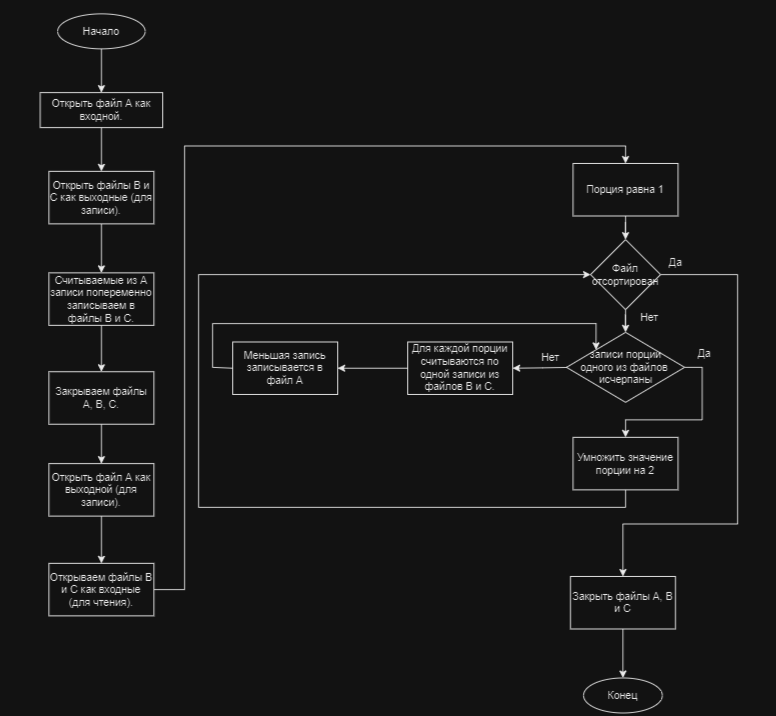


Рисунок 1 – Блок-схема алгоритма прямого слияния

## **2.3 Реализация алгоритма на языке C++ и проведение тестирования**

### **2.3.1 Реализация алгоритма прямого слияния на языке C++**

Для воплощения данного алгоритма в код на языке C++ потребуются следующие библиотеки: iostream, random, chrono, fstream, algorithm, vector и string.

Библиотека iostream предоставляет возможности для ввода и вывода данных через стандартные потоки cout и cin. Это основной инструмент для взаимодействия с пользователем через консоль.

Библиотека chrono используется для работы с временем и временными измерениями. Она позволяет измерять время исполнения кода и выполнение различных операций с точностью до наносекунд.

Библиотека fstream обеспечивает работу с файлами через потоки ввода и вывода. Она позволяет записывать данные в файлы и считывать данные из файлов, что полезно при работе с внешними источниками данных.

Библиотека algorithm содержит множество стандартных алгоритмов, таких как сортировка, поиск, преобразование контейнеров и другие. Она предоставляет удобные инструменты для работы с данными и их обработки.

Библиотека vector представляет собой динамический массив, который позволяет хранить и управлять элементами одного типа данных. Это полезный инструмент для работы с коллекциями объектов переменной длины.

Библиотека string предоставляет функции для работы с символьными строками, такие как конкатенация, поиск, сравнение и другие операции. Она упрощает работу с текстовыми данными в программе.

Кроме того, в коде будут использованы классы, которые представляют собой абстракции, описывающие методы и свойства объектов, и объекты, которые являются конкретным представлением этих абстракций. Созданные объекты на основе классов называются экземплярами этих классов.

| #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string>   std::ostream& operator<< (std::ostream& lhv, const std::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; }  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;   void sort(std::fstream& input);  private:  // вычисление длины серии   void computeSeriesLength(std::fstream& input);   // 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  void merge(std::fstream& input, std::vector<std::fstream>& ext, const unsigned int subgroup\_length);   unsigned int seriesLength\_; // длина серии вычисляется при запуске сортировки   unsigned int wayCount\_; // количество путей (вспомогательных файлов) сортировки   std::string inputFilename\_; };  int main(int argc, char\*\* argv) {  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(input\_file).sort(input);  input.close();  system("pause");  return EXIT\_SUCCESS; }  void SExternalMergeSort::computeSeriesLength(std::fstream& input\_series) {  this->seriesLength\_ = 0;  int value = 0;  while (input\_series >> value)  {  this->seriesLength\_++;  } }  std::vector<std::fstream> SExternalMergeSort::getExternalFiles() {  std::vector<std::fstream> ext(this->wayCount\_);  for (unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = std::to\_string(i + 1);  std::ofstream f(filename);  ext[i] = std::fstream(filename);  }  return ext; }  void SExternalMergeSort::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 = std::to\_string(i + 1);  ext[i].open(filename, mode);  } }  void SExternalMergeSort::removeExternalFiles() {  for (unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = std::to\_string(i + 1);  const auto removeResult = std::remove(filename.c\_str());  std::cout << "Remove external file (" << filename << "): " << std::boolalpha << removeResult << std::endl;  } }  // закрыть и открыть файл входных данных для корректного ввода-вывода void SExternalMergeSort::reopenInputFile(std::fstream& input, std::ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); }  void SExternalMergeSort::split(std::fstream& input\_series, std::vector<std::fstream>& ext, const unsigned int subgroup\_length) {  while (!input\_series.eof())  {  for (auto&& f : ext)  {  int value = 0;  unsigned int step = 0;  while (input\_series >> value)  {  f << value << " ";  std::cout << value << "\t";  ++step;  if (step == subgroup\_length) break;  }  std::cout << " | ";  }  std::cout << std::endl;  }   for (auto&& f : ext)  {  f.flush();  } }  void SExternalMergeSort::merge(std::fstream& input\_series, std::vector<std::fstream>& ext, const unsigned int subgroup\_length) {  if (subgroup\_length == 0) return;   std::vector<int> 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 последней группы в некоторых путях  const unsigned int add = this->seriesLength\_ % 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)  {  int value = 0;  unsigned int substep = 0;  while (substep < subgroup\_length && f >> value)  {  buf.push\_back(value);  ++substep;  }  }   std::sort(std::begin(buf), std::end(buf));   const auto bufLength = buf.size();  for (int i = 0; i < bufLength; ++i)  {  input\_series << buf[i] << " ";  std::cout << buf[i] << " ";  }  std::cout << "\t";   buf.clear();  buf.resize(0);  step++;  }  std::cout << std::endl; }  void SExternalMergeSort::sort(std::fstream& input\_series) {  this->computeSeriesLength(input\_series);  if (this->seriesLength\_ <= 1)  {  return;  }   std::vector<std::fstream> ext = this->getExternalFiles();   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);  }   input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 1 – Программа main.cpp

### **2.3.2 Тестирование**

Необходимо выполнить тестирование программы, используя определенные данные, которые находятся в файле:

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

2

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.

Для более точных результатов в программе будем рассматривать микросекунды. Результаты тестирования представлены на рисунке 2.

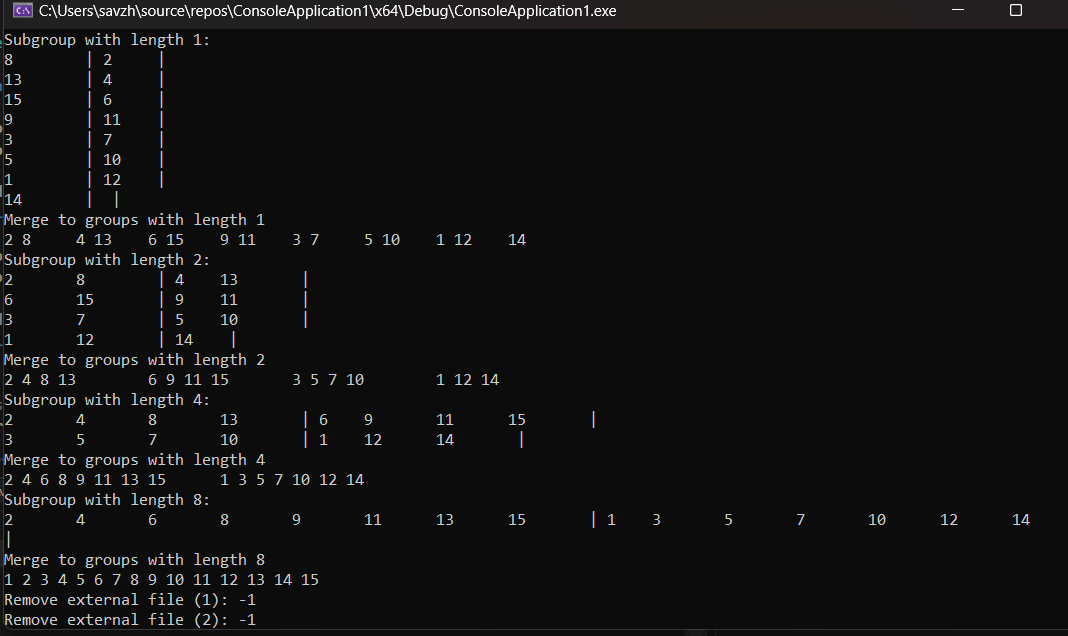


Рисунок 2 - Тестирование программы с входными данными из примера

## **2.4 Адаптация программы к новым данным**

Адаптируем программу к данным по условию индивидуального варианта. Магазин игрушек. Сведения о книге: Автор, Название, Инвентарный номер, Издательство, Количество страниц, Цена. То, что выделено, играет важную роль в процессе сортировки. Определим структуру файла, которая содержит строки (набор символов, хранящихся в массиве char) и три целых числа. Для этой цели будет использован пользовательский заголовочный файл "prost.h".

Код разбит на две части: заголовочный файл "prost.h" (блок кода 2) и основной файл main.cpp (блок кода 3).

| #pragma once #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> #include <functional> using namespace std;  // 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 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(fstream& input, Compare&& cmp = [](const SType& a, const SType& b) { return a < b; });  private:  // вычисление длины серии   void computeSeriesLength(fstream& input);   // имя для вспомогательного файла  string getExternalFilename(const unsigned int way);   // получить список из wayCount\_ вспомогательных файлов  vector<fstream> getExternalFiles();   // закрыть и открыть вспомогательные файлы для корректного ввода-вывода  void reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode);   // удалить вспомогательные файлы  void removeExternalFiles();   // закрыть и открыть файл входных данных для корректного ввода-вывода  void reopenInputFile(fstream& input, ios\_base::openmode mode);   // разделение серии во вспомогательные файлы подгруппами длинны subgroup\_length  void split(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length);   // слияние из вспомогательных файлов подгруппами длинны subgroup\_length  template<class Compare>  void merge(fstream& input, vector<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   string inputFilename\_; };  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::computeSeriesLength(fstream& input\_series) {  this->seriesLength\_ = 0;  SType value{};   while (input\_series >> value)  {  this->seriesLength\_++;  } }  template<class SType, char DSep> string SExternalMergeSort<SType, DSep>::getExternalFilename(const unsigned int way) {  return "/" + to\_string(way + 1); }  template<class SType, char DSep> vector<fstream> SExternalMergeSort<SType, DSep>::getExternalFiles() {  vector<fstream> ext(this->wayCount\_);   for (unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = this->getExternalFilename(i);  ofstream f(filename);  ext[i] = fstream(filename);  }  return ext; }  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::reopenExternalFiles(vector<fstream>& ext, 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 = remove(filename.c\_str());  cout << "Remove external file (" << filename << "): " << boolalpha << removeResult << endl;  } }  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::reopenInputFile(fstream& input, ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); }  template<class SType, char DSep> void SExternalMergeSort<SType, DSep>::split(fstream& input\_series, vector<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;  cout << value << "\t";  ++step;  if (step == subgroup\_length) break;  }  cout << " | ";  }  cout << endl;  } }  template<class SType, char DSep> template<class Compare> void SExternalMergeSort<SType, DSep>::merge(  fstream& input\_series, vector<fstream>& ext,  const unsigned int subgroup\_length,  Compare& cmp) {  if (subgroup\_length == 0) return;   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(begin(buf), end(buf), cmp);   const auto bufLength = buf.size();   for (int i = 0; i < bufLength; ++i)  {  input\_series << buf[i] << DSep;  cout << buf[i] << DSep;  }  cout << "\t";   buf.clear();  buf.resize(0);  step++;  }  cout << endl; }  template<class SType, char DSep> template<class Compare> void SExternalMergeSort<SType, DSep>::sort(fstream& input\_series, Compare&& cmp) {  this->computeSeriesLength(input\_series);   if (this->seriesLength\_ <= 1)  {  return;  }   auto&& cmp\_ = forward<Compare>(cmp);  vector<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, ios\_base::in);  reopenExternalFiles(ext, ios\_base::out);   cout << "Subgroup with length " << to\_string(subgroup\_length) << ":" << endl;  this->split(input\_series, ext, subgroup\_length);   reopenInputFile(input\_series, ios\_base::out);  reopenExternalFiles(ext, ios\_base::in);   cout << "Merge to groups with length " << to\_string(subgroup\_length) << endl;  this->merge(input\_series, ext, subgroup\_length, cmp\_);  }   input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 2 - заголовочный файл "prost.h"

| #include <fstream> #include <vector> #include "prost.h" #include <string> #include <sstream> using namespace std;  // Функция разделения строки по символу-разделителю vector<string>& split(const string& s, char delim, vector<string>& elems) {  stringstream ss(s);  string item;  while (getline(ss, item, delim))  {  elems.push\_back(item);  }  return elems; }  //  struct Consumer {  string author;  string name;  string number ;  string published;  string pages;  string price; };  istream& operator>>(istream& lhv, Consumer& rhv) {  string tmp;  getline(lhv, tmp);   vector<string> e;  split(tmp, ',', e);  const auto elementsCount = e.size();   if(elementsCount > 0)  {  vector<string> fio;  split(e[0], ' ', fio);   if(fio.size())  {  rhv.author = fio[0];  rhv.name = fio[1];  rhv.number = fio[2];  }   if(elementsCount >= 2) rhv.published = e[1].substr(1) + ", " + e[2].substr(1);  if(elementsCount >= 3) rhv.pages = e[3].substr(1);  if(elementsCount >= 4) rhv.price = e[4].substr(1);  }   return lhv; }  ostream& operator<<(ostream& lhv, Consumer& rhv) {  lhv << rhv.author << " " << rhv.name << " " << rhv.number  << ", " << rhv.published << ", " << rhv.pages << ", " << rhv.price;  return lhv; }  int main(int argc, char\*\* argv) {  //выбирает категорию классификации символов языка С  setlocale(LC\_CTYPE, "rus");   const auto input\_file = "A";  fstream input(input\_file, ios\_base::in | ios\_base::out);   if (!input)  {  perror("Ошибка с входным файлом");  return EXIT\_FAILURE;  }   SExternalMergeSort<Consumer, '\n'>(input\_file).sort(input, [](const Consumer& a, const Consumer& b) { return a.author < b.author; });  input.close();  system("Пауза");  //успешное выполнение программы  return EXIT\_SUCCESS; } |
| --- |

Блок кода 3 - Файл main.cpp

## **2.5 Тестирование адаптированного кода**

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

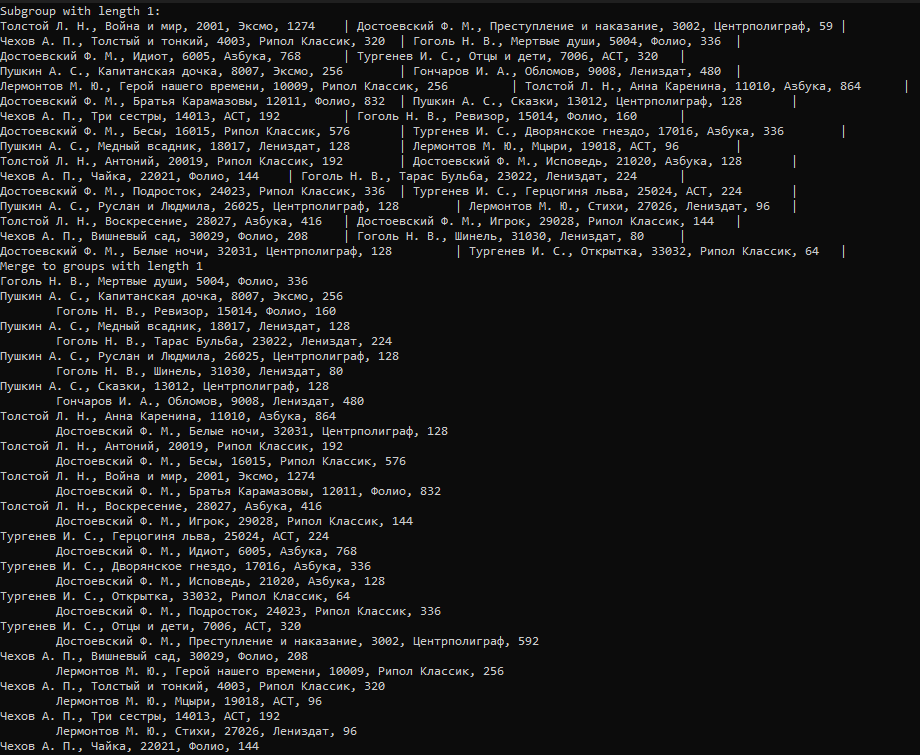


Рисунок 3 - Тестирование адаптированной программы

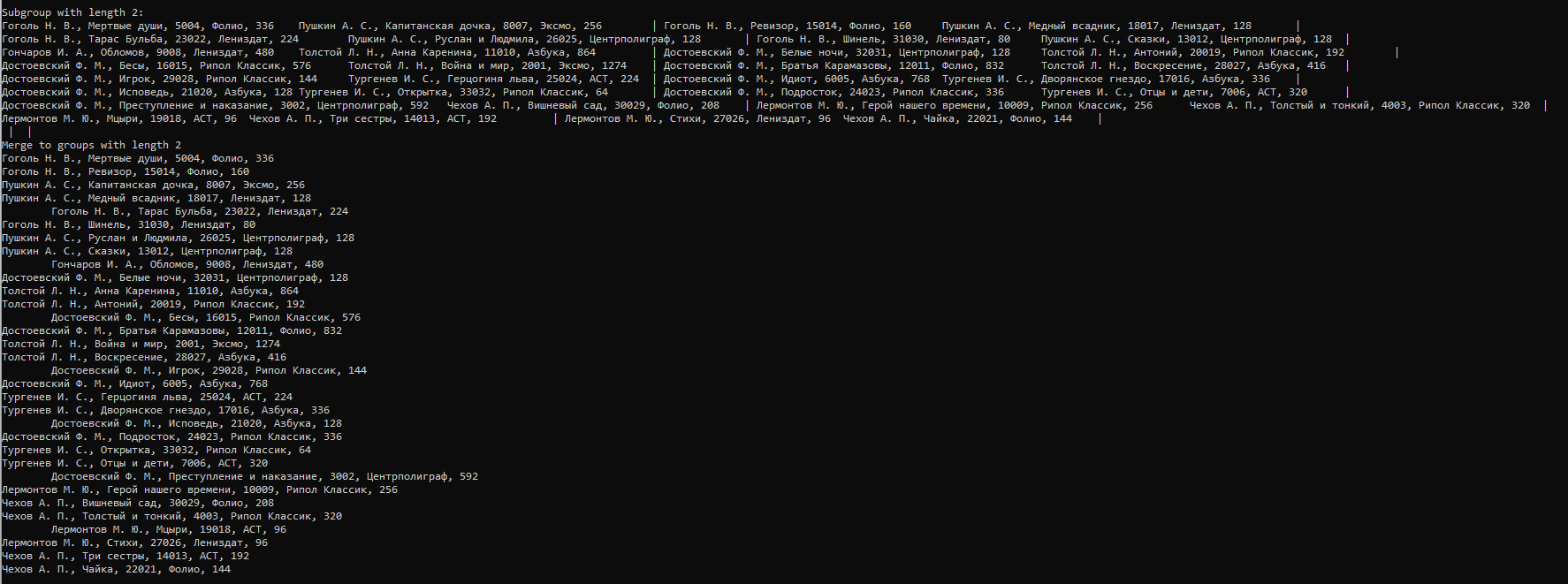


Рисунок 4 - Тестирование адаптированной программы

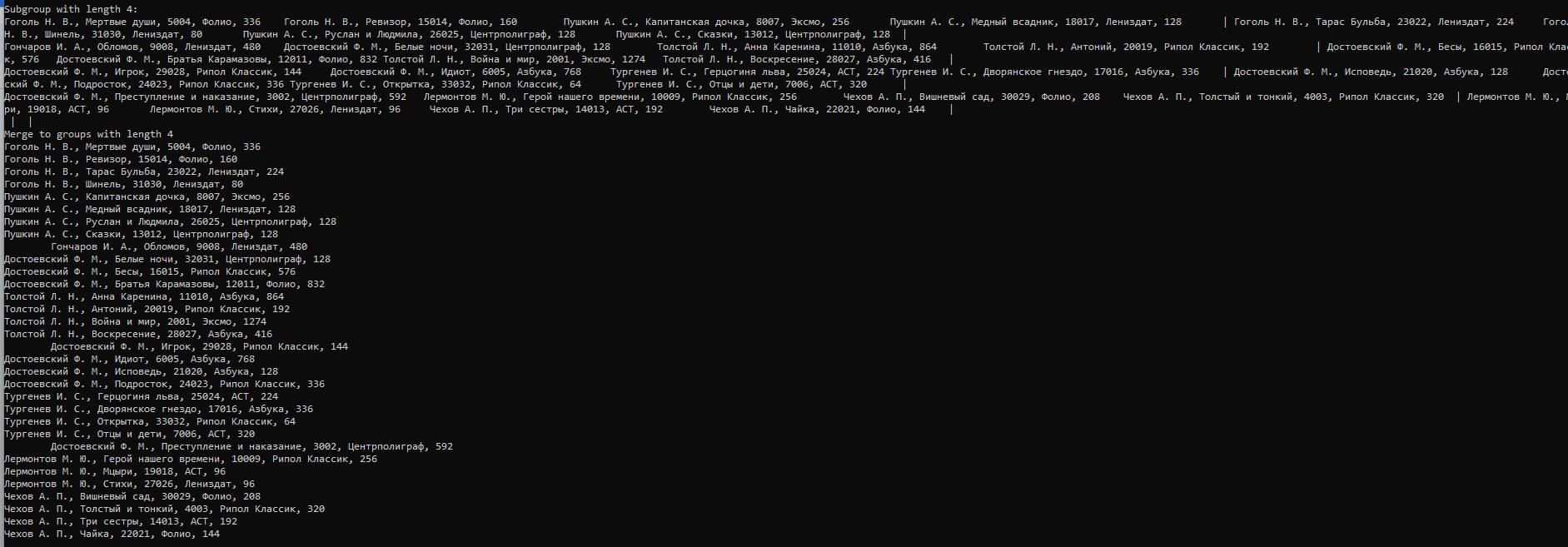


Рисунок 5 - Тестирование адаптированной программы

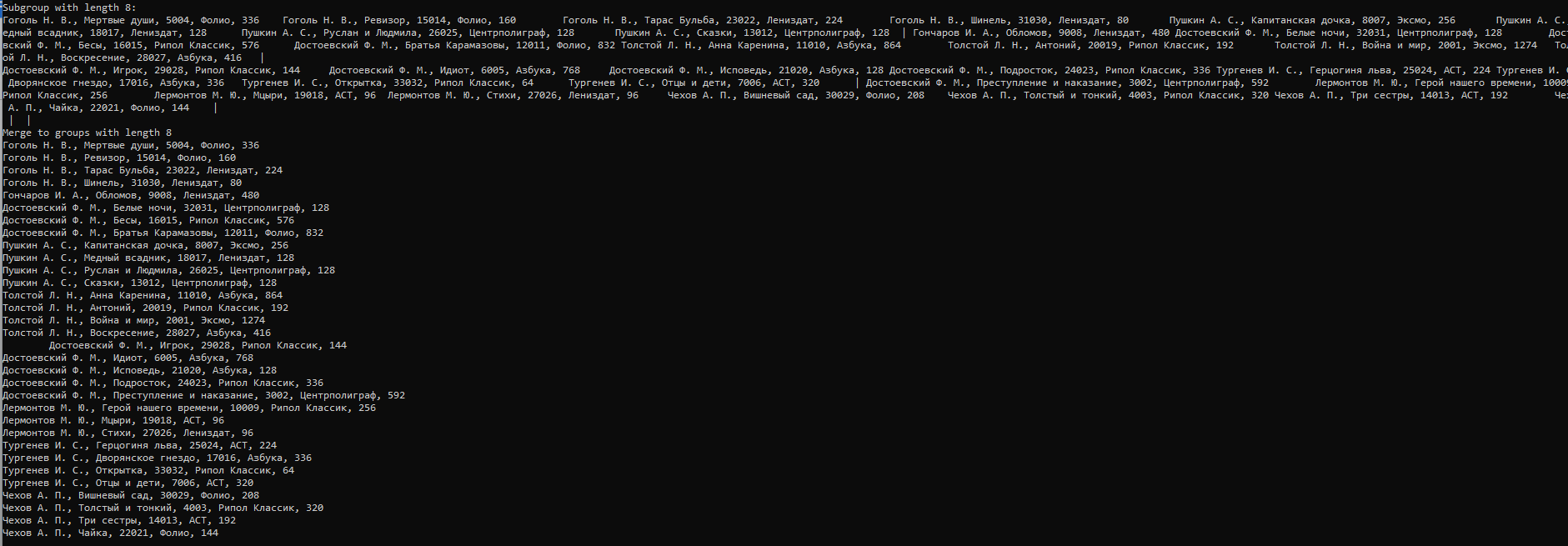


Рисунок 6 - Тестирование адаптированной программы

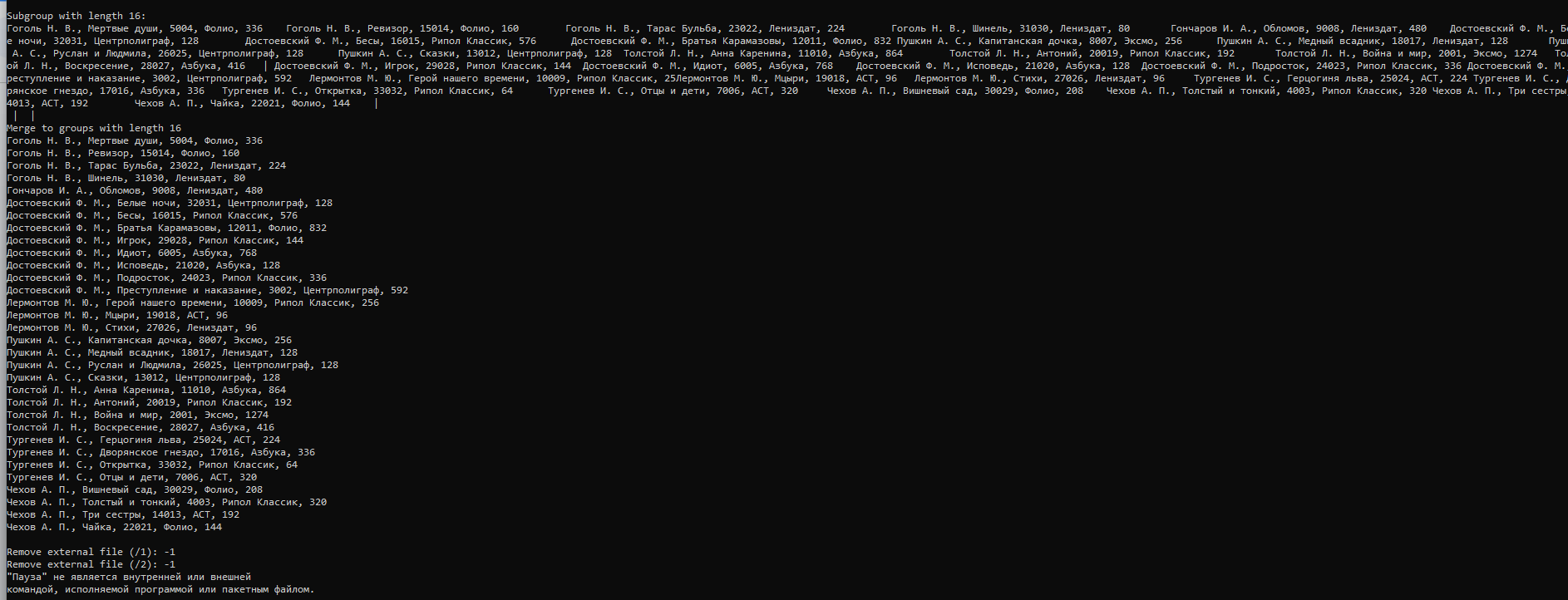


Рисунок 7 - Тестирование адаптированной программы

## **2.6 Практическая сложность алгоритма**

При увеличении количества записей в файле, увеличивается количество операций чтения и записи на диск, что сказывается на времени выполнения алгоритма. Докажем это утверждение с помощью тестирования на разном количестве записей(таб.1). Таким образом, практическая сложность алгоритма прямого слияния будет увеличиваться пропорционально увеличению количества записей в файле.

Таблица 1 - Сводная таблица результатов

| **Количество записей** | **Время(мкс)** |
| --- | --- |
| 8 | 203093 |
| 16 | 599637 |
| 32 | 2603692 |

## **2.7 Вывод по заданию №1**

Сортировка прямым слиянием - это эффективный метод внешней сортировки, особенно при работе с большими объемами данных, которые не помещаются в оперативную память. Алгоритм разделяет данные на подсписки, сортирует их и объединяет в отсортированный список. Его эффективность зависит от нескольких факторов:

1. Объем данных

2. Структура данных

3. Объем оперативной памяти

4. Оптимизации алгоритма

В общем, сортировка прямым слиянием имеет свои преимущества, такие как устойчивость к большим данным и работа с внешними носителями. Однако, в некоторых случаях она может быть менее эффективной по сравнению с другими методами сортировки.

Его практическая сложность зависит от объема данных, структуры данных и доступной оперативной памяти. Вне зависимости от ситуации, данный алгоритм демонстрирует сложность O(n log n), что подтверждает его высокую эффективность в сортировке. Время работы алгоритма связано с количеством элементов в списке, но оно никогда не превышает n log n операций. Ёмкостная сложность этого метода оценивается как O(n).

# **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**

Сортировка натуральным слиянием - это метод упорядочивания данных, который применяет принципы слияния для распределения и последующего объединения элементов. В этом методе исходные данные разделяются на упорядоченные части, которые затем сливаются вместе для получения окончательно отсортированного списка.

Основные этапы сортировки натуральным слиянием следующие:

1. Разделение: исходный список делится на упорядоченные подсписки.

2. Слияние: упорядоченные подсписки сливаются в более крупные упорядоченные подсписки.

3. Повторение: процесс разделения и слияния продолжается до тех пор, пока не будет получен окончательно отсортированный список.

Преимущество сортировки натуральным слиянием заключается в том, что она эффективно обрабатывает данные, которые уже частично упорядочены, и не требует дополнительной памяти для временных массивов, так как сортировка выполняется на месте.

Для улучшения эффективности сортировки был предложен метод предварительного разделения данных на серии одинаковой длины, их сортировки в оперативной памяти и записи обратно в файл. Чем больше серию можно обработать в памяти, тем более эффективным становится алгоритм сортировки.

Рассмотрим алгоритм и его фазы. Он также состоит из двух этапов:

1. Фаза предварительного разделения включает чтение данных из файла, их сортировку и запись обратно в файлы.

2. Фаза слияния состоит из последовательного чтения и слияния данных из файлов.

Таким образом, сортировка натуральным слиянием является эффективным методом для обработки больших объемов данных, особенно когда данные уже частично отсортированы.

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

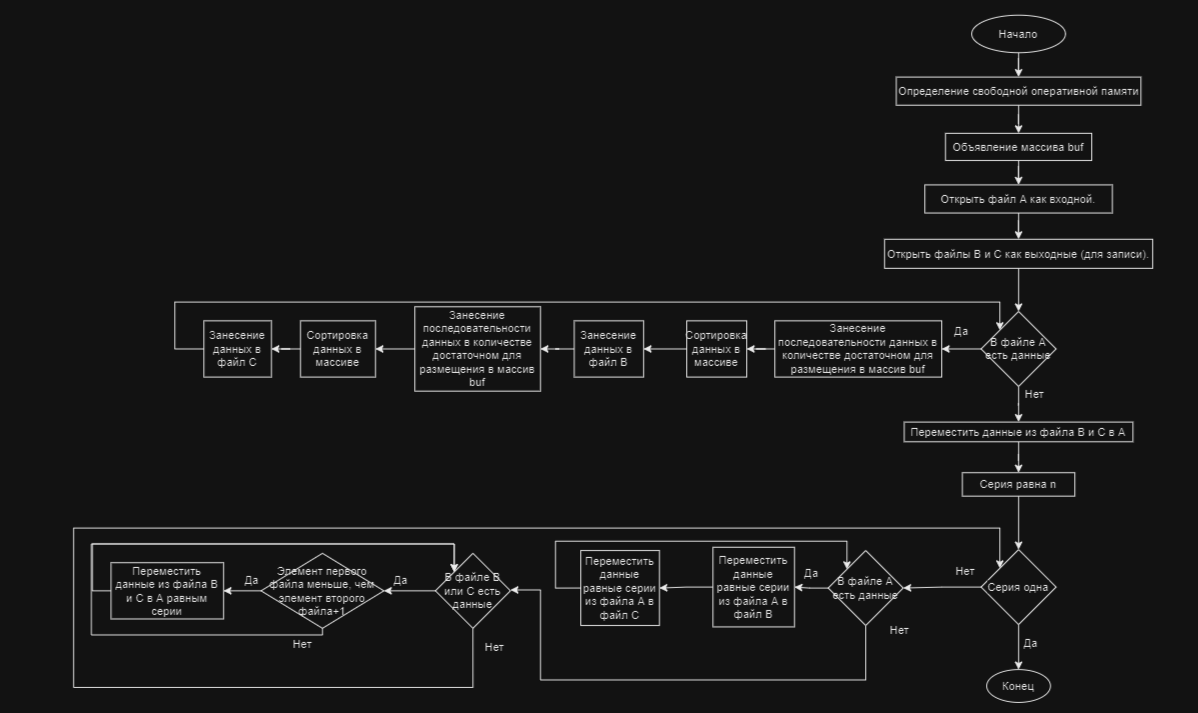


Рисунок 8 – Блок-схема алгоритма естественного слияния

## **3.3 Реализация алгоритма на языке C++ и проведение тестирования**

### **3.3.1 Реализация алгоритма естественного слияния на языке C++**

Для создания этого алгоритма на языке C++ нам потребуются несколько библиотек, включая iostream, random, chrono, fstream, algorithm, vector и string.

Библиотека iostream в C++ представляет собой инструмент для ввода и вывода данных через стандартные потоки cout и cin. Она содержит классы, функции и переменные для управления этими процессами.

Библиотека chrono используется для работы с временем и измерения времени выполнения кода. Она включает функции для точного измерения времени выполнения программы.

Файловая библиотека fstream предоставляет возможности для работы с файлами через потоки ввода и вывода. Она позволяет записывать данные в файлы и читать данные из файлов.

Библиотека algorithm содержит широкий набор стандартных алгоритмов, таких как сортировка, поиск и преобразование контейнеров. Она обеспечивает удобный доступ к этим алгоритмам для работы с данными.

Векторная библиотека vector представляет собой динамический массив, который может хранить и управлять элементами одного типа данных. Она обеспечивает эффективное хранение и доступ к данным.

Строковая библиотека string предоставляет функции для работы с символьными строками, включая конкатенацию, поиск и сравнение. Она упрощает работу с текстовыми данными.

Директива препроцессора pragma once используется для включения заголовочного файла только один раз в процессе компиляции, предотвращая конфликты имен и ускоряя процесс компиляции.

В программе также будут использоваться классы и объекты для абстрагирования функциональности и создания экземпляров этих классов.

| #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> #include <chrono> using namespace std; 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; } constexpr auto kBlockSeparator = "|"; class NExternalMergeSort { public:  // way\_count -- количество путей (вспомогательных файлов) сортировки  explicit NExternalMergeSort(const string& filename, unsigned int way\_count = 2) : seriesLength\_{ 0 }, wayCount\_{ way\_count }, inputFilename\_{ filename } {  if (wayCount\_ < 2) wayCount\_ = 2;  };  virtual ~NExternalMergeSort() = default;  void sort(fstream& input); private:  // вычисление длины серии   void computeSeriesLength(fstream& input);  // получить список из wayCount\_ вспомогательных файлов  vector<fstream> getExternalFiles();  // закрыть и открыть вспомогательные файлы для корректного ввода-вывода  void reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode);  // удалить вспомогательные файлы  void removeExternalFiles();  // закрыть и открыть файл входных данных для корректного ввода-вывода  void reopenInputFile(fstream& input, ios\_base::openmode mode);  bool fullEOF(vector<fstream>& ext);  // проверка отсортированности входных данных  bool isSorted(fstream& input);  // разделение серии в сортированные подгруппы по порядку следования входных данных  void split(fstream& input, vector<fstream>& ext);  // естественное слияние из вспомогательных файлов  void merge(fstream& input, vector<fstream>& ext);  unsigned int seriesLength\_; // длина серии вычисляется при запуске сортировки   unsigned int wayCount\_ = 2; // количество путей (вспомогательных файлов) сортировки   string inputFilename\_; }; int main(int argc, char\*\* argv) {  const auto input\_file = "A";  fstream input(input\_file, ios\_base::in | ios\_base::out);  if (!input)  {  perror("Error with input file");  return EXIT\_FAILURE;  }  NExternalMergeSort(input\_file).sort(input);  input.close();  system("pause");  return EXIT\_SUCCESS; } void NExternalMergeSort::computeSeriesLength(fstream& input\_series) {  this->seriesLength\_ = 0;  int value = 0;  while (input\_series >> value)  {  this->seriesLength\_++;  } } vector<fstream> NExternalMergeSort::getExternalFiles() {  vector<fstream> ext(this->wayCount\_);  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; } void NExternalMergeSort::reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode) {  for (int i = 0; i < ext.size(); ++i)  {  ext[i].close();  const auto filename = to\_string(i + 1);  ext[i].open(filename, mode);  } } void NExternalMergeSort::removeExternalFiles() {  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 << endl;  } } void NExternalMergeSort::reopenInputFile(fstream& input, ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); } bool NExternalMergeSort::fullEOF(vector<fstream>& ext) {  bool fullEOF = true;  for (auto& f : ext)  {  fullEOF &= f.eof();  if (!fullEOF) break;  }  return fullEOF; } bool NExternalMergeSort::isSorted(fstream& input) {  int value = 0;  input >> value;  int oldValue = value;  bool sorted = false;  while (input >> value)  {  sorted = oldValue <= value;  oldValue = value;  if (!sorted) break;  }  return sorted; }  void NExternalMergeSort::split(fstream& input\_series, vector<fstream>& ext) {  int value = 0;  int oldValue = 0;  unsigned int way = 0;  bool firstStep = true;  while (input\_series >> value)  {  if (firstStep)  {  firstStep = false;   ext[way] << value << " ";  oldValue = value;  cout << value << " ";  continue;  }   if (oldValue > value)  {  ext[way].seekg(-1, ios\_base::cur);  ext[way] << kBlockSeparator << " ";  cout << kBlockSeparator;  way = (way + 1) % wayCount\_;  }  ext[way] << value << " ";  oldValue = value;  cout << value << " ";  }  cout << endl; } void NExternalMergeSort::merge(fstream& input\_series, vector<fstream>& ext) {  while (!this->fullEOF(ext))  {  vector<int> oldValues(this->wayCount\_, { 0 });  vector<bool> eogs(this->wayCount\_, { false }); // end of group (аналог end of file)  for (unsigned int way = 0; way < this->wayCount\_; ++way)  {  int value = 0;  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 (oldValues[checkedWay] < oldValues[wayWithMin]) wayWithMin = checkedWay;  }  // слияние  input\_series << oldValues[wayWithMin] << " ";  cout << oldValues[wayWithMin] << " ";  // след. значение  int value = 0;  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;  }  }  for (unsigned int way = 0; way < this->wayCount\_; ++way)  {  if (eogs[way])  {  ext[way].get();  ext[way].get();  }  }  }  cout << endl; } void NExternalMergeSort::sort(fstream& input\_series) {  this->computeSeriesLength(input\_series);  if (this->seriesLength\_ <= 1)  {  return;  }  vector<fstream> ext = this->getExternalFiles();  reopenInputFile(input\_series, ios\_base::in);  bool sorted = isSorted(input\_series);  while (!sorted)  {  reopenInputFile(input\_series, ios\_base::in);  reopenExternalFiles(ext, ios\_base::out);  cout << "Subgroup: " << endl;  this->split(input\_series, ext);  reopenInputFile(input\_series, ios\_base::out);  reopenExternalFiles(ext, ios\_base::in);  cout << "Merge: " << endl;  this->merge(input\_series, ext);  reopenInputFile(input\_series, ios\_base::in);  sorted = isSorted(input\_series);  cout << "Sorted: " << boolalpha << sorted << endl;  }  input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 4 – Программа main.cpp

### **3.3.2 Тестирование**

Необходимо выполнить тестирование программы, используя определенные данные, которые находятся в файле:

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

2

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.

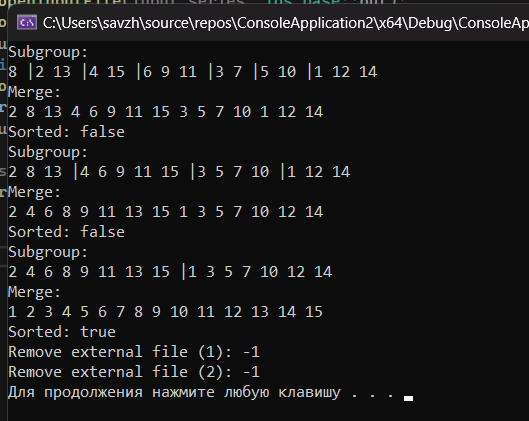


Рисунок 9 - Тестирование программы с входными данными из примера

## **3.4 Адаптация программы к новым данным**

Адаптируем программу к данным по условию индивидуального варианта. Магазин игрушек. Сведения о книге: Автор, Название, Инвентарный номер, Издательство, Количество страниц, Цена. То, что подчёркнуто, является ключом для сортировки. Зададим структуру файла, которая должна состоять из string (последовательности символов, хранящихся в массиве char) и трёх целочисленных значений.

Будем использовать индивидуальный заголовочный файл "natural.h" и библиотеку functional.

Реализация кода включает две части: заголовочный файл "natural.h" (см. блок кода 5) и исходный файл main.cpp (см. блок кода 6).

| #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> using namespace std;  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 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(fstream& input, Compare&& cmp = [](const SType& a, const SType& b) { return a < b; });  private:  // вычисление длины серии   void computeSeriesLength(fstream& input);   // имя для вспомогательного файла  string getExternalFilename(const unsigned int way);   // получить список из wayCount\_ вспомогательных файлов  vector<fstream> getExternalFiles();   // закрыть и открыть вспомогательные файлы для корректного ввода-вывода  void reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode);   // удалить вспомогательные файлы  void removeExternalFiles();   // закрыть и открыть файл входных данных для корректного ввода-вывода  void reopenInputFile(fstream& input, ios\_base::openmode mode);   bool fullEOF(vector<fstream>& ext);   // проверка отсортированности входных данных  template<class Compare>  bool isSorted(fstream& input, Compare& cmp);   // разделение серии в сортированные подгруппы по порядку следования входных данных  template<class Compare>  void split(fstream& input, vector<fstream>& ext, Compare& cmp);   // естественное слияние из вспомогательных файлов  template<class Compare>  void merge(fstream& input, vector<fstream>& ext, Compare& cmp);   unsigned int seriesLength\_; // длина серии вычисляется при запуске сортировки   unsigned int wayCount\_ = 2; // количество путей (вспомогательных файлов) сортировки   string inputFilename\_; };  template<class SType, char DSep> void NExternalMergeSort<SType, DSep>::computeSeriesLength(fstream& input\_series) {  this->seriesLength\_ = 0;  SType value{};   while (input\_series >> value)  {  this->seriesLength\_++;  } }  template<class SType, char DSep> string NExternalMergeSort<SType, DSep>::getExternalFilename(const unsigned int way) {  return "/" + to\_string(way + 1); }  template<class SType, char DSep> vector<fstream> NExternalMergeSort<SType, DSep>::getExternalFiles() {  vector<fstream> ext(this->wayCount\_);   for(unsigned int i = 0; i < this->wayCount\_; ++i)  {  const auto filename = getExternalFilename(i);  ofstream f(filename);  ext[i] = fstream(filename);  }  return ext; }  template<class SType, char DSep> void NExternalMergeSort<SType, DSep>::reopenExternalFiles(vector<fstream>& ext, 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 = remove(filename.c\_str());  cout << "Remove external file (" << filename << "): " << boolalpha << removeResult << endl;  } }  template<class SType, char DSep> void NExternalMergeSort<SType, DSep>::reopenInputFile(fstream& input, ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); }  template<class SType, char DSep> bool NExternalMergeSort<SType, DSep>::fullEOF(vector<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(fstream& input, Compare& cmp) {  SType value{};  input >> value;  SType oldValue = value;   bool sorted = false;  while (input >> value)  {  // sorted = cmp(oldValue, value); // рабочий вариант, если нет повторов  //sorted = cmp(oldValue, value);  /\*const auto cmp1 = cmp(oldValue, value);  const auto cmp2 = cmp(value, oldValue);  sorted = (cmp1 == cmp2) || (cmp1 ^ !cmp2);\*/  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(  fstream& input\_series, vector<fstream>& ext,  Compare& cmp  ) {  SType value{};  SType oldValue{};  unsigned int way = 0;   if (input\_series >> value)  {  ext[way] << value << DSep;  oldValue = value;  cout << value << DSep;  }   while(input\_series >> value)  {  if (!cmp(oldValue, value) && cmp(value, oldValue))  {  way = (way + 1) % wayCount\_;  }   ext[way] << value << DSep;  oldValue = value;  cout << value << DSep;  }  cout << endl; }  template<class SType, char DSep> template<class Compare> void NExternalMergeSort<SType, DSep>::merge(  fstream& input\_series, vector<fstream>& ext,  Compare& cmp ) {  while(!this->fullEOF(ext))  {  vector<SType> oldValues(this->wayCount\_);  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;  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();  }  }  }   cout << endl; }  template<class SType, char DSep> template<class Compare> void NExternalMergeSort<SType, DSep>::sort(fstream& input\_series, Compare&& cmp) {   this->computeSeriesLength(input\_series);  if (this->seriesLength\_ <= 1)  {  return;  }   auto&& cmp\_ = forward<Compare>(cmp);  vector<fstream> ext = this->getExternalFiles();   reopenInputFile(input\_series, ios\_base::in);  bool sorted = isSorted(input\_series, cmp\_);  cout << "Sorted: " << boolalpha << sorted << endl;  while(!sorted)  {  reopenInputFile(input\_series, ios\_base::in);  reopenExternalFiles(ext, ios\_base::out);   cout << "Subgroup: " << endl;  this->split(input\_series, ext, cmp\_);   reopenInputFile(input\_series, ios\_base::out);  reopenExternalFiles(ext, ios\_base::in);   cout << "Merge: " << endl;  this->merge(input\_series, ext, cmp\_);   reopenInputFile(input\_series, ios\_base::in);  sorted = isSorted(input\_series, cmp\_);   cout << "Sorted: " << boolalpha << sorted << endl;  }   input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 5 - заголовочный файл "natural.h"

| #include <fstream> #include <vector> #include "natural.h" #include <string> #include <sstream> using namespace std;  // Функция разделения строки по символу-разделителю vector<string>& split(const string& s, char delim, vector<string>& elems) {  stringstream ss(s);  string item;   while (getline(ss, item, delim))  {  elems.push\_back(item);  }  return elems; }  //  struct Consumer {  string author;  string name;  string number;  string published;  string pages;  string price; };  istream& operator>>(istream& lhv, Consumer& rhv) {  string tmp;  getline(lhv, tmp);   vector<string> e;  split(tmp, ',', e);  const auto elementsCount = e.size();   if (elementsCount > 0)  {  vector<string> fio;  split(e[0], ' ', fio);   if (fio.size())  {  rhv.author = fio[0];  rhv.name = fio[1];  rhv.number = fio[2];  }   if (elementsCount >= 2) rhv.published = e[1].substr(1) + ", " + e[2].substr(1);  if (elementsCount >= 3) rhv.pages = e[3].substr(1);  if (elementsCount >= 4) rhv.price = e[4].substr(1);  }   return lhv; }  ostream& operator<<(ostream& lhv, Consumer& rhv) {  lhv << rhv.author << " " << rhv.name << " " << rhv.number  << ", " << rhv.published << ", " << rhv.pages << ", " << rhv.price;  return lhv; }  int main(int argc, char\*\* argv) {  //выбирает категорию классификации символов языка С  setlocale(LC\_CTYPE, "rus");   const auto input\_file = "A";  fstream input(input\_file, ios\_base::in | ios\_base::out);  if (!input)  {  perror("Ошибка с входным файлом");  return EXIT\_FAILURE;  }   NExternalMergeSort<Consumer, '\n'>(input\_file).sort(input, [](const Consumer& a, const Consumer& b){ return a.published < b.published; });  input.close();  system("Пауза");  //успешное выполнение программы  return EXIT\_SUCCESS; } |
| --- |

Блок кода 6 - Файл main.cpp

## **3.5 Тестирование адаптированного кода**

Протестируем адаптированный код Результаты тестирования будут продемонстрированы на рисунке 10.

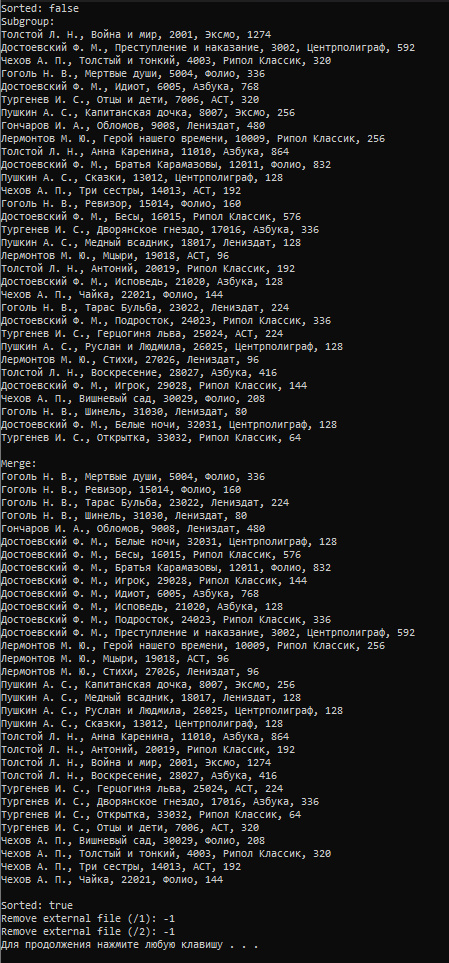


Рисунок 10 - Тестирование адаптированной программы

## **3.6 Практическая сложность алгоритма**

Для оценки практической сложности алгоритма естественного слияния при увеличении числа записей в файлах необходимо учитывать количество операций сравнения и перемещения элементов в процессе сортировки. Это утверждение будет подтверждено путем тестирования на различных объемах записей (см. Таблицу 2). Таким образом, практическая сложность алгоритма естественного слияния для файлов с растущим количеством записей остается приблизительно постоянной и составляет O(log n), где n - число записей в файле.

Таблица 2 - Сводная таблица результатов

| **Количество записей** | **Время(мкс)** |
| --- | --- |
| 8 | 328906 |
| 16 | 902492 |
| 32 | 3130237 |

## **3.7 Вывод по заданию №2**

Метод естественного слияния в сортировке делит список на упорядоченные подсписки и объединяет их, что обеспечивает эффективность при работе с большими списками за счет отсутствия необходимости в дополнительной памяти.

Практическая сложность зависит от размера списка: в лучшем случае O(n), в худшем - O(n log n), а в среднем - приближается к O(n log n). Время выполнения также увеличивается с увеличением размера списка, но остается более эффективным по сравнению с алгоритмами, требующими дополнительной памяти. Метод подходит для сортировки как больших, так и маленьких списков, что подтверждается в таблице 2, делая его универсальным. Благодаря отсутствию необходимости в дополнительной памяти и стабильной сложности с ростом числа записей, метод естественного слияния предпочтителен для сортировки больших списков. В то время как прямое слияние может быть удобнее для маленьких списков, оно менее эффективно при обработке больших данных из-за дополнительного использования памяти.

# **4 ВЫВОДЫ**

В ходе практической работы были выполнены следующие задачи:

- Освоены приёмы сортировки данных из файлов;

- Проведён анализ алгоритмов прямого и естественного слияния;

- Были реализованы программы алгоритмов прямого и естественного слияния;

- Проведено тестирование программ для алгоритмов прямого и естественного слияния;

- Программы прямого и естественного слияния были адаптированы под индивидуальный вариант;

- Проведено тестирование программ для алгоритмов прямого и естественного слияния на разных количествах записей в файлах;

-Сделан вывод об эффективности данных алгоритмов;

Таким образом, главную цель практической работы, а именно освоение приёмов сортировки данных из файлов, можно считать выполненной.

# **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).

12. НОУ ИНТУИТ | Технопарк Mail.ru Group: Алгоритмы и структуры данных [Электронный ресурс]. URL: https://intuit.ru/studies/courses/3496/738/info (дата обращения 15.03.2022).