|  |
| --- |
| 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: Харченко А.А. |
|  | 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](#_1t3h5sf)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 8](#)

[2.3.1 Реализация алгоритма прямого слияния на языке C++ 8](#_4d34og8)

[2.3.2 Тестирование 15](#_2s8eyo1)

[2.4 Адаптация программы к новым данным 16](#_17dp8vu)

[2.6 Тестирование адаптированного кода 24](#_ziao7tuaw113)

[2.7 Практическая сложность алгоритма 27](#_1ci93xb)

[2.8 Вывод по заданию №1 27](#_dslhzqc2xurl)

[3 ЗАДАНИЕ №2 28](#_2bn6wsx)

[3.1 Формулировка задачи 28](#_qsh70q)

[3.2 Математическая модель решения алгоритма 28](#_u290pxmzmczz)

[3.2.1 Описание выполнения и блок-схема алгоритма естественного слияния 28](#_ujf0wkarvz46)

[3.2.2 Доказательство корректности циклов алгоритма естественного слияния 30](#_vg00rxuqjnjx)

[3.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма естественного слияния 31](#_foeym3wsnfua)

[3.3 Реализация алгоритма на языке C++ и проведение тестирования 32](#_44wcxh77t5yt)

[3.3.1 Реализация алгоритма естественного слияния на языке C++ 32](#_fweoj2zgp349)

[2.3.2 Тестирование 39](#_qzyyeyy25qpm)

[3.4 Адаптация программы к новым данным 40](#_tztmvu7f1qy2)

[3.6 Тестирование адаптированного кода 50](#_omt9yh4vlatz)

[3.7 Практическая сложность алгоритма 61](#_80lopqn0frwv)

[3.8 Вывод по заданию №2 61](#_fc3bwntftrcf)

[5 ВЫВОДЫ 63](#_ihv636)

[6 ЛИТЕРАТУРА 64](#_32hioqz)

# **1 ЦЕЛЬ**

Освоить приёмы сортировки данных из файлов

# **2 ЗАДАНИЕ №1**

## **2.1 Формулировка задачи (В списке №10, Вариант 10)**

Разработать программу и применить алгоритм внешней сортировки прямого слияния к сортировке файла данных “Магазин игрушек” по значению ключевого поля “название”.

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 и блок-схема алгоритма прямого слияния**

Прямое слияние - это метод сортировки, который заключается в разделении массива на две равные части, сортировке каждой из них отдельно, а затем слиянии двух отсортированных частей в один упорядоченный массив.

Алгоритм сортировки прямого слияния для файлов

Фаза разделения:

1. Открыть файл А как входной.

2. Открыть файлы В и С как выходные (для записи).

3. Считываемые из А записи попеременно записываем в файлы В и С.

4. Закрываем файлы А, В, С.

Фаза слияния:

1. Открыть файл А как выходной (для записи).

2. Открываем файлы В и С как входные (для чтения).

3. Установить размер порции сливаемых данных: 1, 2, 4, 8 и т.д. для этого

и следующих этапов.

4. Для каждой порции считываются по одной записи из файлов В и С.

5. Меньшая запись записывается в файл А, и считывается очередная запись

из того файла, запись которого была переписана в файл А.

6. Пункты 4 и 5 повторяются до тех пор, пока записи очередной порции

одного из файлов не будут исчерпаны.

7. Оставшиеся записи из порции другого файла переписываются в файл А.

8. Пункты с 4 по 7 повторяются до тех пор, пока не будет достигнут конец

одного из файлов В и С. Тогда оставшиеся записи из другого файла переписываются в файл А.

9. Закрываются файлы А В С.

Сортировка завершается тогда, когда длина порции достигнет n.

Реализуем данное описания выполнения алгоритма в виде блок-схемы (рис.1).

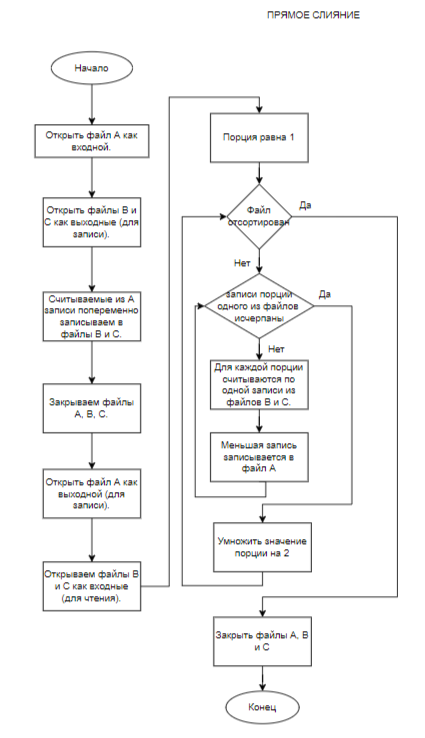


Рисунок 1 – Блок-схема алгоритма прямого слияния

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

Алгоритм будет реализован на языке C++, где будут использоваться библиотеки iostream, fstream, vector, algorithm, string(блок-кода 1)

| #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> using namespace std;   ostream& operator<< (ostream& lhv, const vector<int>& rhv) {  const auto size = rhv.size();*// размер вектора*  lhv << "(";  for (int i = 0; i < size; ++i)*// вывод элементов вектора*  {  lhv << rhv[i];  if(i+1 < size) lhv << " ";  }  lhv << ")";  return lhv; }  class SExternalMergeSort// Объявление класса SExternalMergeSort { public:  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;  void sort(fstream& input) private:  *// длина серии*   void computeSeriesLength(fstream& input);  vector<fstream> getExternalFiles();  void reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode);  *// удалить дополнительные файлы*  void removeExternalFiles();  void reopenInputFile(fstream& input, ios\_base::openmode mode);  void split(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length);  *// слияние*  void merge(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length);  unsigned int seriesLength\_; *// длина серии*  unsigned int wayCount\_;*// количество путей*  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;  }  SExternalMergeSort(input\_file).sort(input);  input.close();  system("pause");  return EXIT\_SUCCESS; }  void SExternalMergeSort::computeSeriesLength(fstream& input\_series) {  this->seriesLength\_ = 0;  int value = 0;  while (input\_series >> value)  {  this->seriesLength\_++;  } }  vector<fstream> SExternalMergeSort::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 SExternalMergeSort::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 SExternalMergeSort::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 SExternalMergeSort::reopenInputFile(fstream& input, ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); }  void SExternalMergeSort::split(fstream& input\_series, vector<fstream>& ext, const unsigned int subgroup\_length) {  while(!input\_series.eof())*//запись в файлы по подгруппам*  {  for(auto&& f: ext)  {  int value = 0;  unsigned int step = 0;  while(input\_series >> value)  {  f << value << " ";  cout << value << "\t";  ++step;  if (step == subgroup\_length) break;  }  cout << " | ";  }  cout << endl;  }  for (auto&& f : ext)  {  f.flush();  } }  void SExternalMergeSort::merge(fstream& input\_series, vector<fstream>& ext, const unsigned int subgroup\_length) {  if (subgroup\_length == 0) return;  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(begin(buf), end(buf));  const auto bufLength = buf.size();  for(int i = 0; i < bufLength; ++i)  {  input\_series << buf[i] << " ";  cout << buf[i] << " ";  }  cout << "\t";  buf.clear();  buf.resize(0);  step++;  }  cout << endl; }  void SExternalMergeSort::sort(fstream& input\_series) {   this->computeSeriesLength(input\_series);  if (this->seriesLength\_ <= 1)  {  return;  }  vector<fstream> ext = this->getExternalFiles();  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);*// Разделение данных на подгруппы и запись в доп файлы*  reopenExternalFiles(ext, ios\_base::in);  cout << "Merge to groups with length " << to\_string(subgroup\_length) << endl;  this->merge(input\_series, ext, subgroup\_length);  }  input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 1 – Программа main.cpp

Тестирование проводилось на основе данных представленных в примере задания. 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.

Результаты тестирования представлены на рисунке 2.

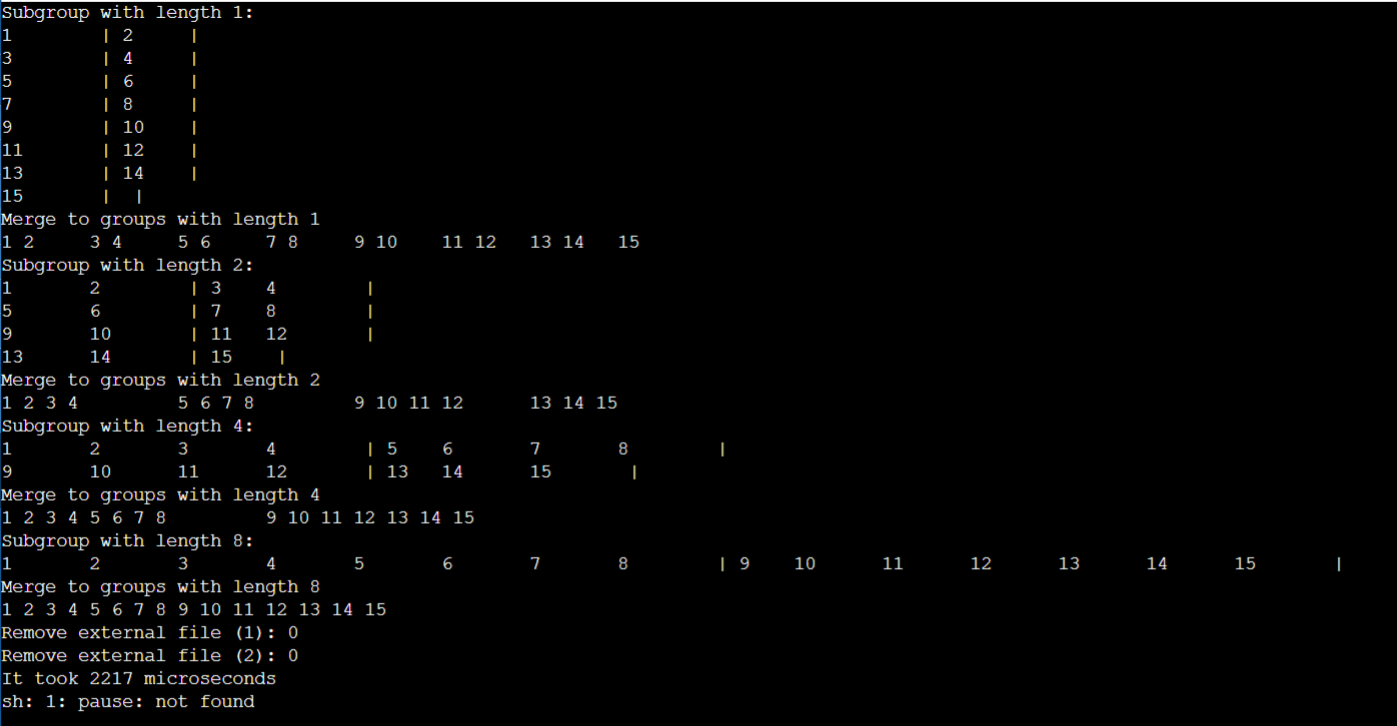


Рисунок 2 - Тестирование программы с входными данными из примера

## **2.4 Адаптация программы к индивидуальному варианту задания**

Переделаем программу так, чтобы она была адаптирована под данные из файла по индивидуальному варианту. Индивидуальный вариант: Магазин игрушек. Сведения об игрушке: Название (например: кукла, конструктор и т.д.), стоимость, возрастные границы детей (для кого игрушка предназначена) два поля – начальный возраст и конечный). Предоставим реализацию кода в блоке кода 2 и 3, а тестирование в рисунках 3-5.

| #pragma once #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> #include <functional> using namespace std;  template<class SType, char DSep = ' '> class SExternalMergeSort { public:  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(fstream& input, Compare&& cmp = [](const SType& a, const SType& b) { return a < b; });  private:  void computeSeriesLength(fstream& input);  string getExternalFilename(const unsigned int way);  vector<fstream> getExternalFiles();  void reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode);  void removeExternalFiles();  void reopenInputFile(fstream& input, ios\_base::openmode mode);  void split(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length);  template<class Compare>  void merge(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length, Compare& cmp);  unsigned int seriesLength\_;  unsigned int wayCount\_;  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();  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\_);  }  *//Сбрасываем буфер в файл и удаляем внешние файлы.*  input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 2 - заголовочный файл "ems\_simple.h"

| #include <fstream> #include <vector> #include <chrono> #include "ems\_simple.h" using namespace std; struct A {  *//Название, цена, начальный возраст, конечный возраст*  string n;  int a = 0;  int b = 0;  int c = 0; };  istream& operator>>(istream& lhv, A& rhv) {  string n;  while (true)  {  const auto c = lhv.get();  if (c == -1 || c == ',') break;  if (c >= 0) n += (char)c;  }  rhv.n = n;  int value = 0;  lhv >> value;  rhv.a = value;  lhv.get();  lhv >> value;  rhv.b = value;  lhv.get();  lhv >> value;  rhv.c = value;  lhv.get();  return lhv; } ostream& operator<<(ostream& lhv, const A& rhv) {  lhv << rhv.n << ", " << rhv.a << ", " << rhv.b << ", " << rhv.c;  return lhv; } int main(int argc, char\*\* argv) {  setlocale(LC\_ALL, "RUS");  auto start = chrono::high\_resolution\_clock::now();  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;  }  SExternalMergeSort<A, '\n'>(input\_file).sort(input, [](const A& a, const A& b) { return a.n < b.n; });  input.close();  auto end = chrono::high\_resolution\_clock::now();  auto d = chrono::duration\_cast<chrono::microseconds>(end - start).count();  cout << "It took " << d << " microseconds" << endl;  system("pause");  return EXIT\_SUCCESS; } |
| --- |

Блок кода 3 - Файл main.cpp

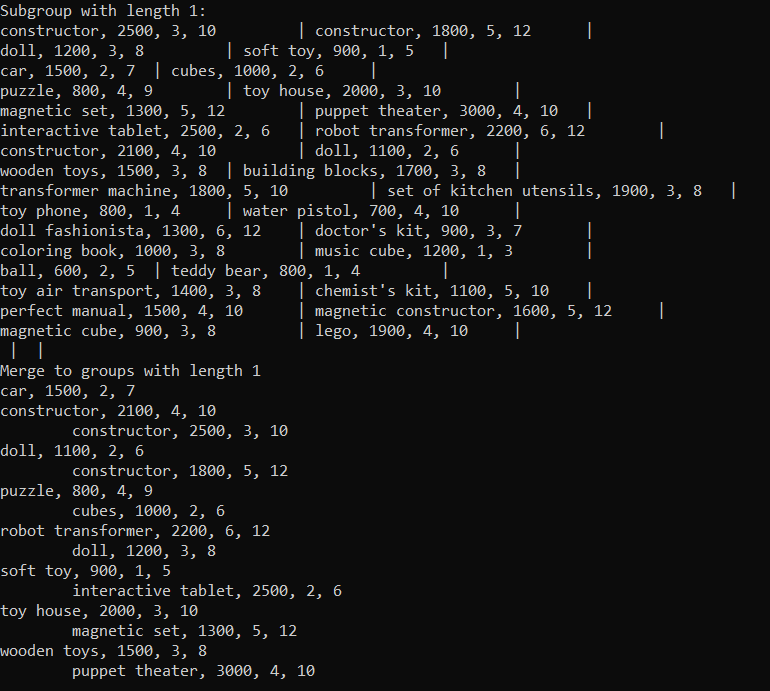


Рисунок 3 - Тестирование адаптированной программы

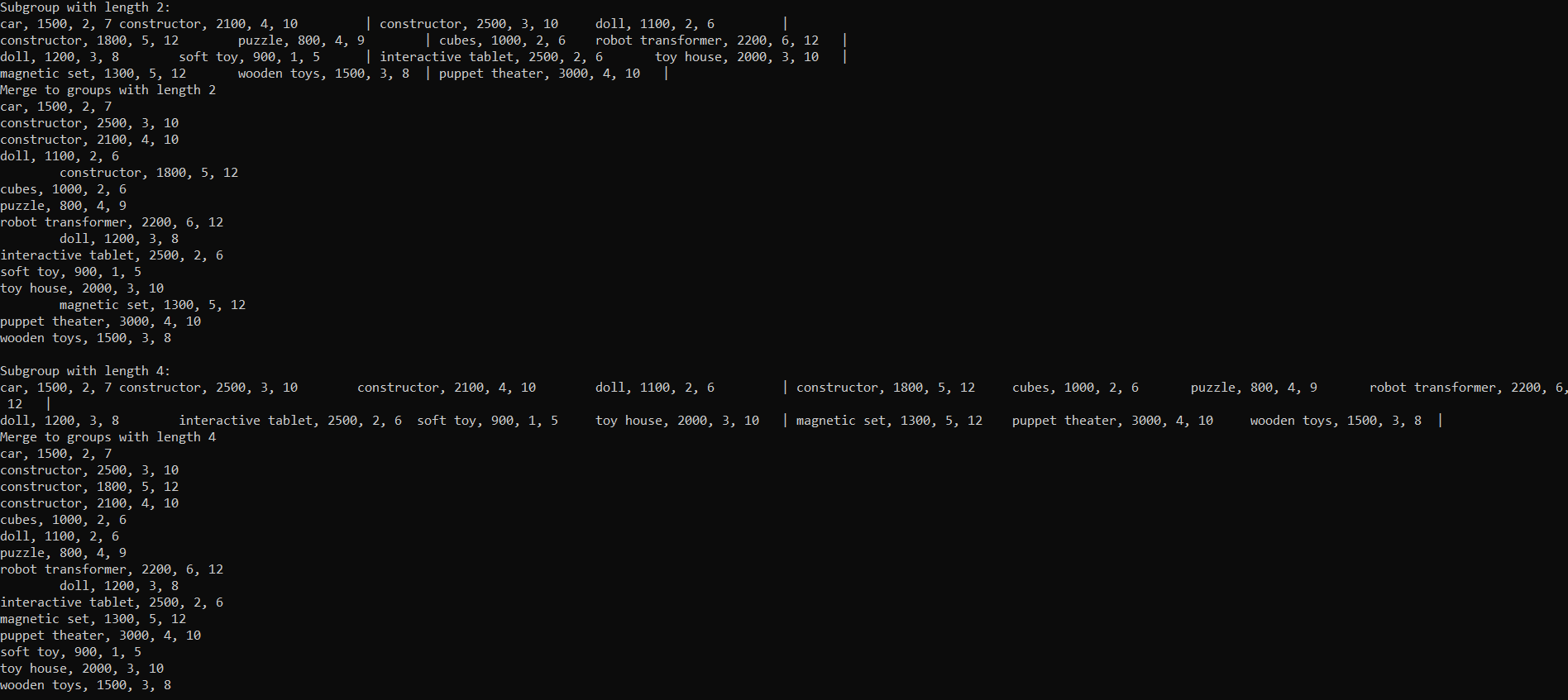


Рисунок 4 - Тестирование адаптированной программы

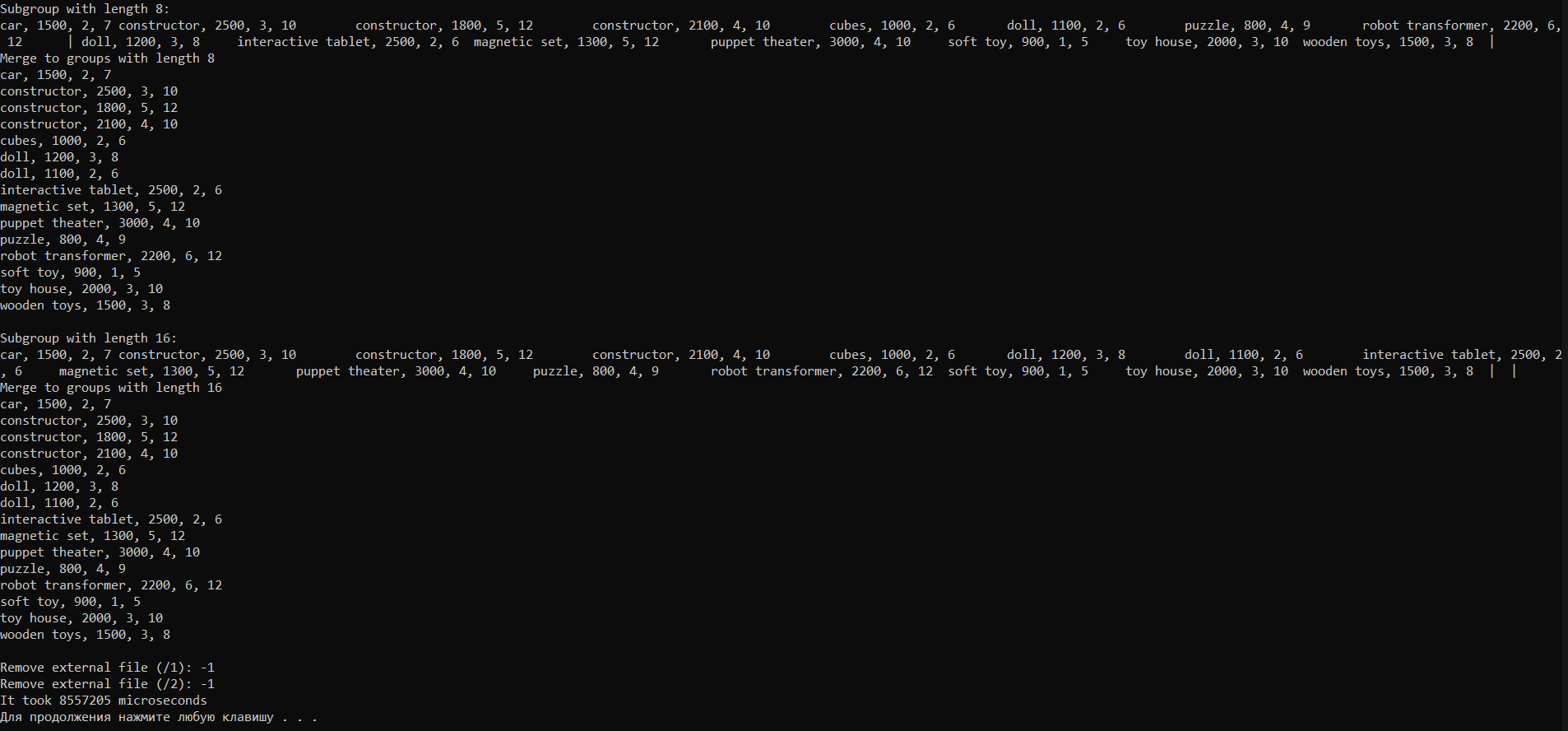


Рисунок 5 - Тестирование адаптированной программы

## **2.7 Практическая сложность алгоритма**

При увеличении количества записей в файле, количество операций чтения и записи на диск также увеличивается, что сказывается на времени выполнения алгоритма. Это утверждение можно доказать путем тестирования на разном количестве записей (табл. 1). Таким образом, практическая сложность алгоритма прямого слияния будет возрастать пропорционально увеличению количества записей в файле.

Таблица 1 - Сводная таблица результатов

| **Количество записей** | **Время(мкс)** |
| --- | --- |
| 8 | 317437 |
| 16 | 726472 |
| 32 | 3283748 |

## **2.8 Вывод по заданию №1**

Алгоритм сортировки прямого слияния является эффективным и устойчивым методом сортировки, который разбивает список на две части, сортирует их отдельно, а затем объединяет отсортированные части в один упорядоченный список. Этот алгоритм имеет сложность O(n log n) в любом случае, что делает его одним из наиболее эффективных среди сортировок. Время выполнения зависит от количества элементов в списке, но гарантированно не превысит n log n операций. Емкостная сложность данного алгоритма составляет O(n).

Прямое слияние требует дополнительной памяти для сортировки, поскольку создает временные массивы для слияния, поэтому он не слишком эффективен для больших объемов данных из-за дополнительного использования памяти. Тем не менее, сортировка прямого слияния всё же более эффективна, чем простые, быстрые или усовершенствованные сортировки, поскольку она позволяет эффективно сортировать как большие, так и маленькие списки, что делает ее универсальным методом для разных задач.

# **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 и блок-схема алгоритма естественного слияния**

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

Такое решение предлагается вам исследовать и разобраться в реализации.

Рассмотрим алгоритм и его фазы. Он так же является двухфазным.

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

2. Открыть исходный файл А, подлежащий сортировке.

3. Открыть два файла для записи В и С.

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

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

6. Пункты 4 и 5 выполнять, пока все данные из файла А не будут переписаны отсортированными во вспомогательные файлы В и С.

7. Слить данные в файл А сначала из файла В, затем из файла С.

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

8. Фаза разделения включает поочередную запись серий из А в файлы В и С.

9. Фаза слияния имеет теперь следующий алгоритм:

• Считываем данные из одного и другого файлов, пока ai<ai+1, меньшее из сравниваемых записывать в файл А, пока одна из серий не будет исчерпана, тогда остаток другой переписываем в файл А, пока выполняется условие ai<ai+1.

• После этого считываем следующую серию и так пока один из файлов не

станет пустым, тогда серии другого переписываются в файл А.

10. Пункты 8 и 9 повторяются пока в файл А, в результате слияния не будет переписана только одна серия

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.6).

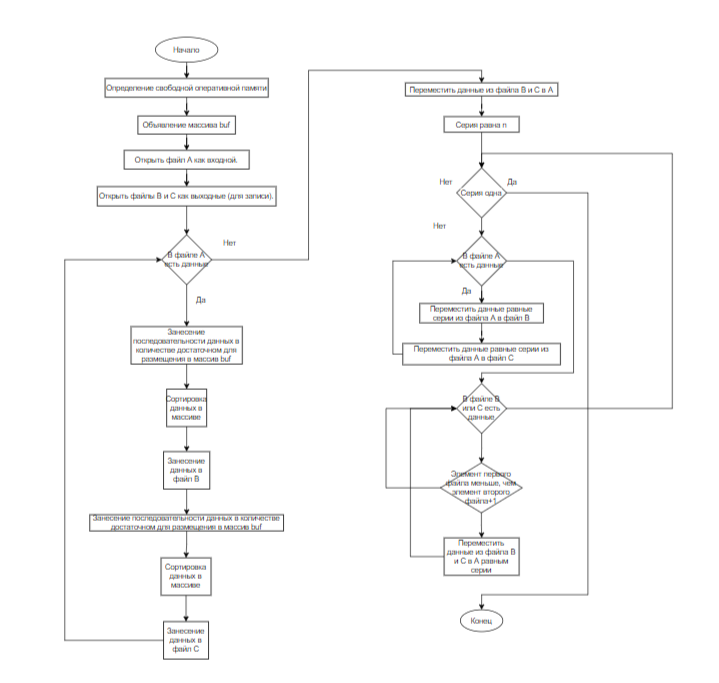


Рисунок 6 – Блок-схема алгоритма естественного слияния

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

Алгоритм будет реализован на языке C++, где будут использоваться библиотеки iostream, fstream, vector, algorithm, string(блок-кода 4)

| #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

Тестирование проводилось на основе данных представленных в примере задания. Пример, для погружения в алгоритм сортировки естественным слиянием.

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.

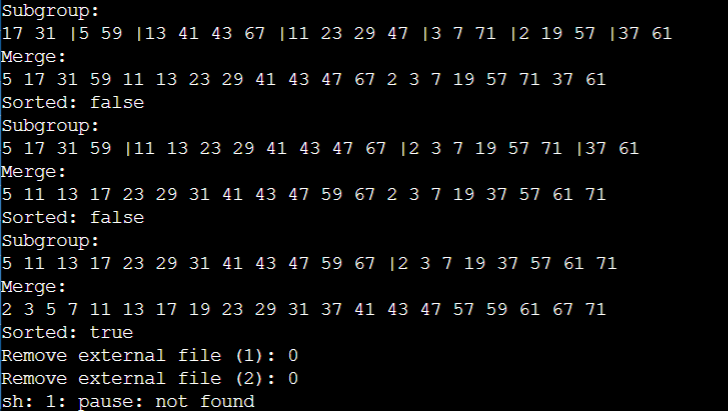


Рисунок 9 - Тестирование программы

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

Переделаем программу так, чтобы она была адаптирована под данные из файла по индивидуальному варианту. Индивидуальный вариант: Магазин игрушек. Сведения об игрушке: Название (например: кукла, конструктор и т.д.), стоимость, возрастные границы детей (для кого игрушка предназначена) два поля – начальный возраст и конечный). Предоставим реализацию кода в блоке кода 2 и 3, а тестирование в рисунках 10.

| #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)  {  *// sorted = cmp(oldValue, value); // рабочий варик, 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(  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 "ems\_natural.h"

| #include <fstream> #include <vector> #include <chrono> #include "ems\_natural.h" using namespace std; struct A {  *//Название, цена, начальный возраст, конечный возраст*  string n;  int a = 0;  int b = 0;  int c = 0; };  istream& operator>>(istream& lhv, A& rhv) {  string n;  while (true)  {  const auto c = lhv.get();  if (c == -1 || c == ',') break;  if (c >= 0) n += (char)c;  }  rhv.n = n;  int value = 0;  lhv >> value;  rhv.a = value;  lhv.get();  lhv >> value;  rhv.b = value;  lhv.get();  lhv >> value;  rhv.c = value;  lhv.get();  return lhv; } ostream& operator<<(ostream& lhv, const A& rhv) {  lhv << rhv.n << ", " << rhv.a << ", " << rhv.b << ", " << rhv.c;  return lhv; } int main(int argc, char\*\* argv) {  setlocale(LC\_ALL, "RUS");  auto start = chrono::high\_resolution\_clock::now();  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<A, '\n'>(input\_file).sort(input, [](const A& a, const A& b) { return a.n < b.n; });  input.close();  auto end = chrono::high\_resolution\_clock::now();  auto d = chrono::duration\_cast<chrono::microseconds>(end - start).count();  cout << "It took " << d << " microseconds" << endl;  system("pause");  return EXIT\_SUCCESS; } |
| --- |

Блок кода 6 - Файл main.cpp

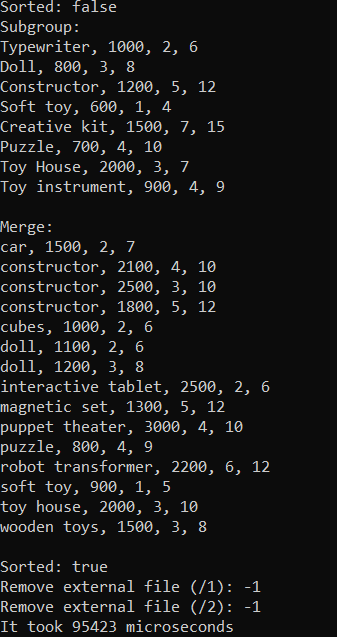


Рисунок 10 - Тестирование адаптированной программы

## **3.7 Практическая сложность алгоритма**

Для оценки практической сложности алгоритма естественного слияния при увеличении числа записей в файле важно учитывать количество операций сравнения и перемещения элементов в процессе сортировки. Доказательство этого утверждения можно провести путем тестирования на различном количестве записей (табл. 2). Таким образом, практическая сложность алгоритма естественного слияния остается примерно постоянной при увеличении числа записей в файле и составляет примерно O(log n), где n - количество записей в файле.

Таблица 2 - Сводная таблица результатов

| **Количество записей** | **Время(мкс)** |
| --- | --- |
| 8 | 367465 |
| 16 | 984365 |
| 32 | 3436538 |

## **3.8 Вывод по заданию №2**

Сортировка естественного слияния - это метод сортировки, который основан на разбиении списка на отсортированные подсписки и последующем объединении их. Этот алгоритм подходит для сортировки больших списков, поскольку не требует дополнительной памяти для сортировки.

Практическая сложность сортировки естественным слиянием зависит от объема элементов в списке. В лучшем случае сложность составляет O(n), а в худшем случае - O(n log n). В среднем случае сложность ближе к O(n log n). Емкостная сложность сортировки естественным слиянием составляет O(1).

Время выполнения сортировки естественным слиянием также зависит от количества элементов в списке. С увеличением числа элементов увеличивается и время выполнения алгоритма. Однако он требует меньше времени, чем традиционные алгоритмы сортировки, которые используют дополнительную память.

Сортировка естественным слиянием позволяет эффективно сортировать как большие, так и маленькие списки, что демонстрируется в таблице 2, что делает ее универсальным методом для различных задач.

Этот метод сортировки не требует дополнительной памяти для сортировки, так как использует принцип разделения и слияния подсписков, что делает его эффективным для больших списков и имеющим стабильную сложность при увеличении числа записей.

В общем, сортировка естественным слиянием более эффективна и предпочтительна для сортировки больших списков, так как не требует дополнительной памяти и имеет стабильную практическую сложность. В то время как прямое слияние может быть более удобным для небольших списков, но может оказаться менее эффективным при работе с большими объемами данных из-за использования дополнительной памяти.

# **5 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.

# **6 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).