|  |
| --- |
| 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: Елисеев И.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 Алгоритм решения алгоритма 4](#_2et92p0)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 5](#)

[2.3.1 Реализация алгоритма прямого слияния на языке C++ 5](#_4d34og8)

[2.3.2 Тестирование 10](#_2s8eyo1)

[2.4 Адаптация программы к новым данным 11](#_17dp8vu)

[2.5 Тестирование адаптированного кода 17](#_3rdcrjn)

[2.6 Практическая сложность алгоритма 19](#_26in1rg)

[2.7 Вывод по заданию №1 20](#_lnxbz9)

[3 ЗАДАНИЕ №2 21](#_35nkun2)

[3.1 Формулировка задачи 21](#_1ksv4uv)

[3.2 Алгоритм решения алгоритма 21](#_44sinio)

[3.3 Реализация алгоритма на языке C++ и проведение тестирования 23](#_1y810tw)

[3.3.1 Реализация алгоритма естественного слияния на языке C++ 23](#_4i7ojhp)

[2.3.2 Тестирование 28](#_2xcytpi)

[3.4 Адаптация программы к новым данным 29](#_1ci93xb)

[3.5 Тестирование адаптированного кода 36](#_3whwml4)

[3.6 Практическая сложность алгоритма 47](#_2bn6wsx)

[3.7 Вывод по заданию №2 47](#_qsh70q)

[4 ВЫВОДЫ 49](#_3as4poj)

[5 ЛИТЕРАТУРА 50](#_1pxezwc)

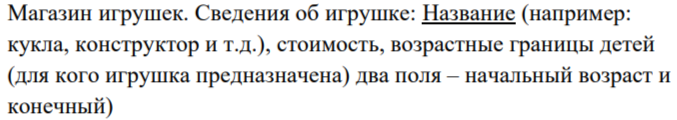
# **1 ЦЕЛЬ**

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 Алгоритм решения алгоритма**

Алгоритм сортировки простым слиянием:

1. Исходный файл f разбивается на два вспомогательных файла f1 и f2.

2. Вспомогательные файлы f1 и f2 сливаются в файл f, при этом одиночные элементы образуют упорядоченные пары.

3. Полученный файл f вновь обрабатывается, как указано в шагах 1 и 2. При этом упорядоченные пары переходят в упорядоченные четверки.

4. Повторяя шаги, сливаем четверки в восьмерки и т.д., каждый раз удваивая длину слитых последовательностей до тех пор, пока не будет упорядочен целиком весь файл.

После выполнения i проходов получаем два файла, состоящих из серий длины 2i. Окончание процесса происходит при выполнении условия 2i>=n. Следовательно, процесс сортировки простым слиянием требует порядка O (log n) проходов по данным.

Признаками конца сортировки простым слиянием являются следующие условия:

1. длина серии не меньше количества элементов в файле (определяется после фазы слияния);

2. количество серий равно 1 (определяется на фазе слияния).

3. количество серий равно 1 (определяется на фазе слияния).

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

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

Реализуем данный алгоритм на языке C++(блок кода 1). Для реализации понадобятся такие библиотеки, как iostream, random, chrono, fstream, algorithm, vector, string.

| #include <iostream> #include <fstream> #include <algorithm> #include <vector> #include <string> using namespace std;   //Оператор перегрузки вывода для потока ostream и вектора целых чисел ostream& operator<< (ostream& lhv, const vector<int>& rhv) {  const auto size = rhv.size();// Получение размера вектора  lhv << "(";  for (int i = 0; i < size; ++i)// Цикл вывода элементов вектора с пробелами между ними  {  lhv << rhv[i];  if(i+1 < size) lhv << " ";  }  lhv << ")";  return lhv;// Возврат потока вывода } class SExternalMergeSort// Объявление класса 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;// Проверка и коррекция значения wayCount, если оно меньше 2  };  virtual ~SExternalMergeSort() = default;// Закрытие виртуального деструктора по умолчанию  void sort(fstream& input);// Определение функции sort класса SExternalMergeSort 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);  // 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  void merge(fstream& input, vector<fstream>& ext, const unsigned int subgroup\_length);   unsigned int seriesLength\_; // длина серии вычисляется при запуске сортировки   unsigned int wayCount\_; // количество путей (вспомогательных файлов) сортировки   string inputFilename\_; }; int main(int argc, char\*\* argv) {  const auto input\_file = "A";// Определение строки inputfile с именем "A"  fstream input(input\_file, ios\_base::in | ios\_base::out);//Открытие файла input в режиме чтения и записи  if (!input) //Проверка, удалось ли открыть файл  {  perror("Error with input file");  return EXIT\_FAILURE;  }  SExternalMergeSort(input\_file).sort(input);//Создание объекта SExternalMergeSort и вызов метода sort с передачей input в качестве параметра  input.close();//Закрытие файла input  system("pause");//Выполнение команды system("pause")  return EXIT\_SUCCESS;//Возврат значения EXITSUCCESS } void SExternalMergeSort::computeSeriesLength(fstream& input\_series) {  this->seriesLength\_ = 0;//Обнуление переменной seriesLength  int value = 0;//Объявление переменной value для хранения числа из файла  while (input\_series >> value)//Цикл чтения чисел из файла и увеличение значения seriesLength  {  this->seriesLength\_++;  } } //Определение функции getExternalFiles класса SExternalMergeSort vector<fstream> SExternalMergeSort::getExternalFiles() {  vector<fstream> ext(this->wayCount\_);//Создание вектора ext с размером wayCount  for(unsigned int i = 0; i < this->wayCount\_; ++i)//Цикл создания вспомогательных файлов и добавления их в вектор  {  const auto filename = to\_string(i + 1);  ofstream f(filename);  ext[i] = fstream(filename);  }  return ext; } //Определение функции reopenExternalFiles класса SExternalMergeSort void SExternalMergeSort::reopenExternalFiles(vector<fstream>& ext, ios\_base::openmode mode) {  for (int i = 0; i < ext.size(); ++i)//Цикл закрытия и переоткрытия вспомогательных файлов с учетом режима mode  {  ext[i].close();   const auto filename = to\_string(i + 1);  ext[i].open(filename, mode);  } } // Определение функции removeExternalFiles класса SExternalMergeSort void SExternalMergeSort::removeExternalFiles() {  for (unsigned int i = 0; i < this->wayCount\_; ++i)//Цикл удаления вспомогательных файлов с выводом результата  {  const auto filename = to\_string(i + 1);  const auto removeResult = remove(filename.c\_str());  cout << "Remove external file (" << filename << "): " << boolalpha << removeResult << endl;  } } // закрыть и открыть файл входных данных для корректного ввода-вывода void SExternalMergeSort::reopenInputFile(fstream& input, ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); } // Определение функции split класса SExternalMergeSort void SExternalMergeSort::split(fstream& input\_series, vector<fstream>& ext, const unsigned int subgroup\_length) {  while(!input\_series.eof())//Цикл чтения данных из входного файла и записи в вспомогательные файлы по подгруппам  {  for(auto&& f: ext)//Цикл для сброса буфера каждого вспомогательного файла после записи данных  {  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();  } } //Определение функции merge класса SExternalMergeSort, принимающей файловый поток inputseries, вектор файловых потоков ext и длину подгруппы subgrouplength void SExternalMergeSort::merge(fstream& input\_series, vector<fstream>& ext, const unsigned int subgroup\_length) {  if (subgroup\_length == 0) return;//Проверка, если длина подгруппы равна 0, то выход из функции  vector<int> buf;//Создание вектора buf для временного хранения данных с резервированием необходимого размера  buf.reserve(subgroup\_length \* this->wayCount\_);//Вычисление количества полных групп в путях  // 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)//Цикл по шагам выполнения алгоритма  {  for (auto& f: ext)//Цикл по каждому файловому потоку вектора 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));//Сортировка данных в буфере buf  const auto bufLength = buf.size();//Получение длины буфера buf  for(int i = 0; i < bufLength; ++i)//Цикл вывода отсортированных данных из буфера в файл inputseries и на консоль  {  input\_series << buf[i] << " ";  cout << buf[i] << " ";  }  cout << "\t";  buf.clear();//Очистка буфера buf и увеличение значения шага на единицу  buf.resize(0);//Вывод символа новой строки на консоль  step++;  }  cout << endl; } //Определение функции sort класса SExternalMergeSort void SExternalMergeSort::sort(fstream& input\_series) {   this->computeSeriesLength(input\_series);//Вычисление длины серии данных inputseries  if (this->seriesLength\_ <= 1)//Проверка, если длина серии меньше или равна 1, то выход из функции  {  return;  }  vector<fstream> ext = this->getExternalFiles();//Получение вектора вспомогательных файлов ext  //Цикл сортировки по подгруппам, удваивая длину подгруппы на каждой итерации  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;// Переоткрытие входного файла и вспомогательных файлов для чтения и записи  this->split(input\_series, ext, subgroup\_length);//Вывод информации о текущей длине подгруппы на консоль  reopenInputFile(input\_series, ios\_base::out);// Разделение данных на подгруппы и запись в вспомогательные файлы  reopenExternalFiles(ext, ios\_base::in);//Переоткрытие входного файла и вспомогательных файлов для обратного чтения и записи  cout << "Merge to groups with length " << to\_string(subgroup\_length) << endl;//Вывод информации о слиянии групп с текущей длиной подгруппы  this->merge(input\_series, ext, subgroup\_length);// Вызов функции merge для слияния данных из вспомогательных файлов входного файла  }  input\_series.flush();//Сброс буфера входного файла  this->removeExternalFiles();//Удаление вспомогательных файлов после завершения сортировки. } |
| --- |

Блок кода 1 – Программа main.cpp

### **2.3.2 Тестирование**

Стоит задача протестировать программу с заданным данными в файле А: 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 и 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 и 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.

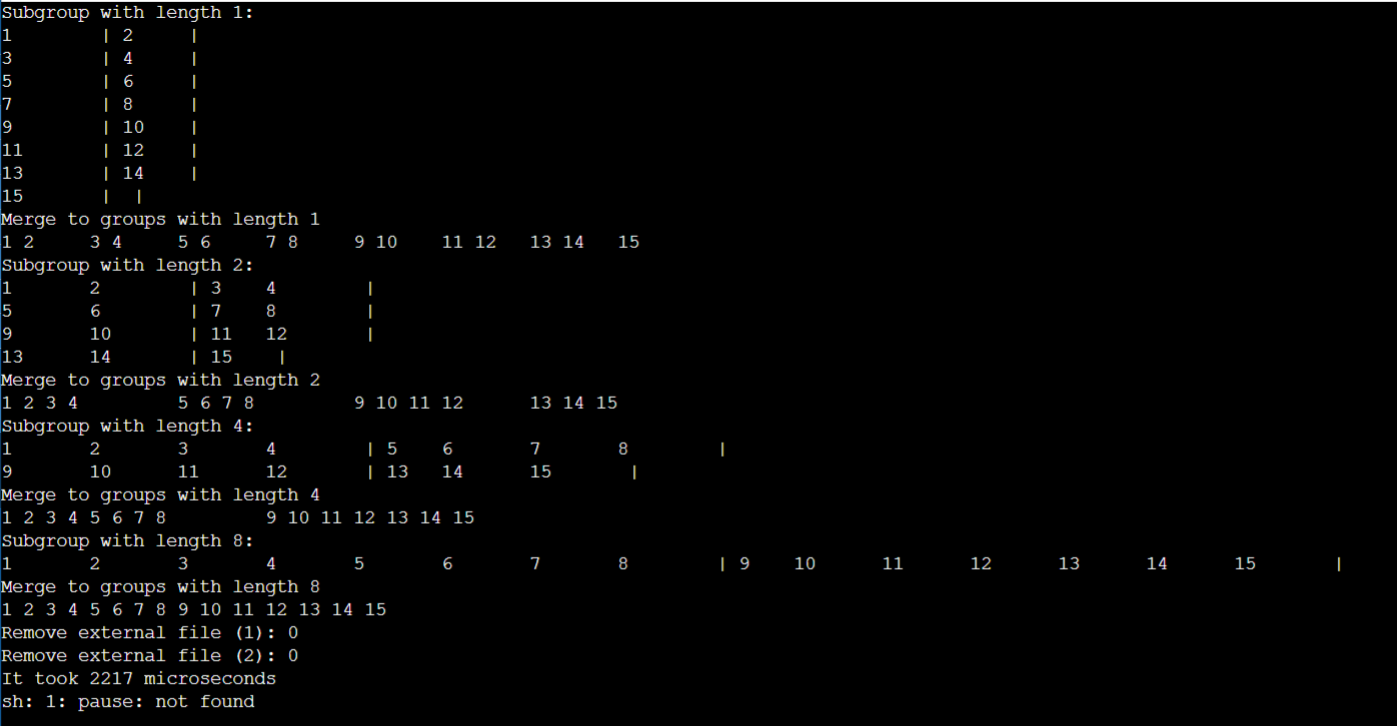


Рисунок 2 - Тестирование программы с входными данными из примера

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

Адаптируем программу к данным по условию индивидуального варианта. Магазин игрушек. Реализация кода состоит из двух частей. Из заголовочного файла "simple.h"(блок кода 2) и исходного файла 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 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:  // вычисление длины серии   void computeSeriesLength(std::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   std::string inputFilename\_; }; template<class SType, char DSep> // Вычисляет длину серии данных, считывая их из файла input\_series до конца. Кол-во прочитанных элементов сохраняется в переменной seriesLength\_. void SExternalMergeSort<SType, DSep>::computeSeriesLength(fstream& input\_series) {  this->seriesLength\_ = 0;  SType value{};  while (input\_series >> value)  {  this->seriesLength\_++;  } }  template<class SType, char DSep> // Возвращает строку, представляющую внешний файл с данными, в зависимости от переданного параметра way. Файлы нумеруются с единицы. string SExternalMergeSort<SType, DSep>::getExternalFilename(const unsigned int way) {  return "/" + to\_string(way + 1); }  template<class SType, char DSep> // Создает и открывает вектор внешних файлов нужного размера и возвращает его. Названия файлов формируются с помощью функции getExternalFilename. 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> //Закрывает и повторно открывает все внешние файлы из вектора ext с заданным режимом mode. Названия файлов берутся из функции getExternalFilename. 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> //Удаляет внешние файлы, используемые для сортировки. 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> //Повторно открывает входной файл с указанным режимом. void SExternalMergeSort<SType, DSep>::reopenInputFile(fstream& input, ios\_base::openmode mode) {  input.close();  input.open(this->inputFilename\_, mode); }  template<class SType, char DSep> //Разделяет данные из входного файла на подгруппы и записывает их во внешние файлы. 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> //Выполняет слияние подгрупп и сортировку данных по возрастанию. 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";  //Очищаем буфер и увеличиваем счетчик шага.  buf.clear();  buf.resize(0);  step++;  }  cout << endl; } template<class SType, char DSep> //Сортирует данные во внешних файлах с использованием алгоритма внешней сортировки template<class Compare> void SExternalMergeSort<SType, DSep>::sort(fstream& input\_series, Compare&& cmp) {  this->computeSeriesLength(input\_series);  if (this->seriesLength\_ <= 1)//Если длина меньше или равна 1, то возвращаемся.  {  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\_);  }  //Сбрасываем буфер в файл и удаляем внешние файлы.  input\_series.flush();  this->removeExternalFiles(); } |
| --- |

Блок кода 2 - заголовочный файл "simple.h"

| #include <fstream> #include <vector> #include <chrono> //Подключение пользовательского заголовочного файла "ems\_simple.h" #include "ems\_simple.h" using namespace std; //Определение структуры файла A, содержащего поля n, a, b, c. struct A {  string n;//Название игрушки  int a = 0;//цена игрушки  int b = 0;//возраст начальный  int c = 0;//возраст конечный }; //Перегрузка оператора >> для чтения значений типа 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;// Присваиваем свойству "n" объекта "rhv" значение переменной "n" rhv.n = n  int value = 0;// Создаем переменную "value" и считываем значение из потока "lhv" int value = 0; lhv >> value;  lhv >> value;// Считываем значение а переменную "value" из потока "lhv" lhv >> value  rhv.a = value;// Присваиваем свойству "a" объекта "rhv" значение переменной "value", считанной из потока "lhv" rhv.a = value  lhv.get();// Пропускаем символ '\n' в потоке "lhv" lhv.get()  lhv >> value;// Снова считываем значение в переменную "value" из потока "lhv" lhv >> value  rhv.b = value;// Присваиваем свойству "b" объекта "rhv" значение переменной "value" rhv.b = value  lhv.get();// Пропускаем символ '\n' в потоке "lhv" lhv.get()  lhv >> value;// Снова считываем значение в переменную "value" из потока "lhv" lhv >> value  rhv.c = value;// Присваиваем свойству "c" объекта "rhv" значение переменной "value" rhv.c = value  lhv.get(); // считывание  return lhv; } //Перегрузка оператора << для записи значений типа A 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();// Начало сортировки  //Открытие файла "A" в режиме чтения и записи.  const auto input\_file = "A";  fstream input(input\_file, ios\_base::in | ios\_base::out);  //Проверка успешности открытия файла  if (!input)  {  perror("Error with input file");  return EXIT\_FAILURE;  }  //Использование шаблонной функции NExternalMergeSort из заголовочного файла "simple.h"  SExternalMergeSort<A, '\n'>(input\_file).sort(input, [](const A& a, const A& b) { return a.n < b.n; });  input.close();//Закрытие входного файла  auto end = chrono::high\_resolution\_clock::now();// Конец сортировки  auto d = chrono::duration\_cast<chrono::microseconds>(end - start).count();// затраченное время  cout << "It took " << d << " microseconds" << endl;  system("pause");//Приостановка выполнения программы для ожидания ввода пользователя   return EXIT\_SUCCESS;//Возврат успешного завершения программы. } |
| --- |

Блок кода 3 - Файл main.cpp

## **2.5 Тестирование адаптированного кода**

Стоит задача протестировать программу с заданным данными в файле А. Результаты тестирования будут продемонстрированы на рисунке 3-7.

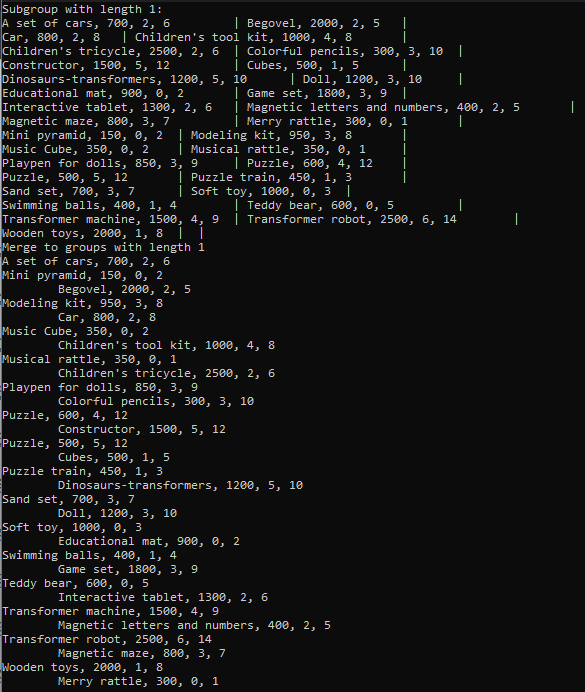


Рисунок 3 - Тестирование адаптированной программы

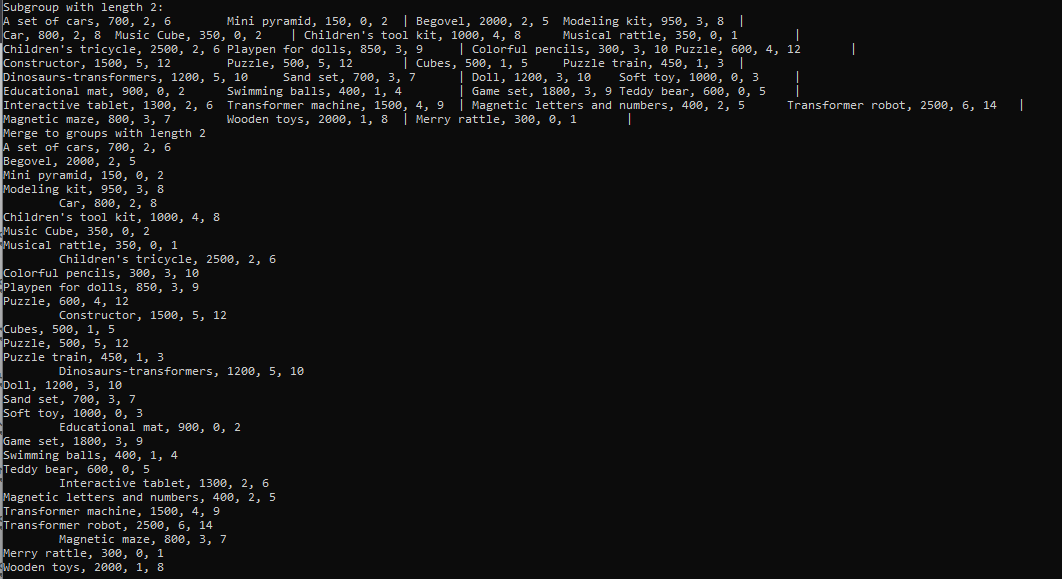


Рисунок 4 - Тестирование адаптированной программы

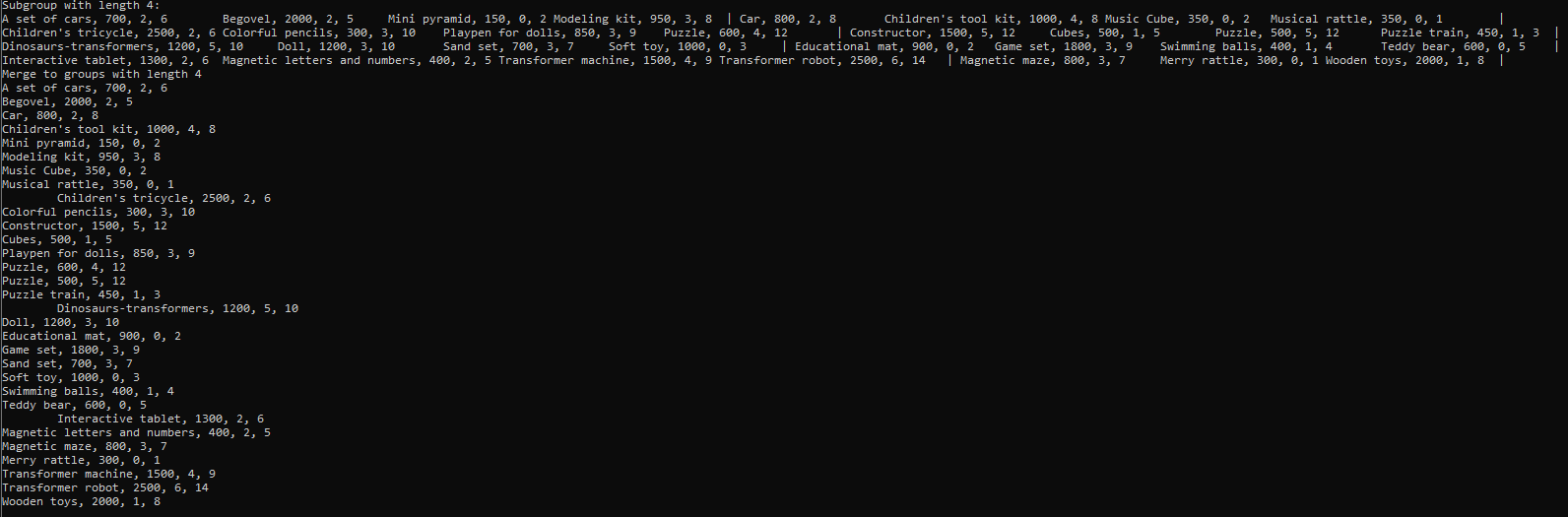


Рисунок 5 - Тестирование адаптированной программы

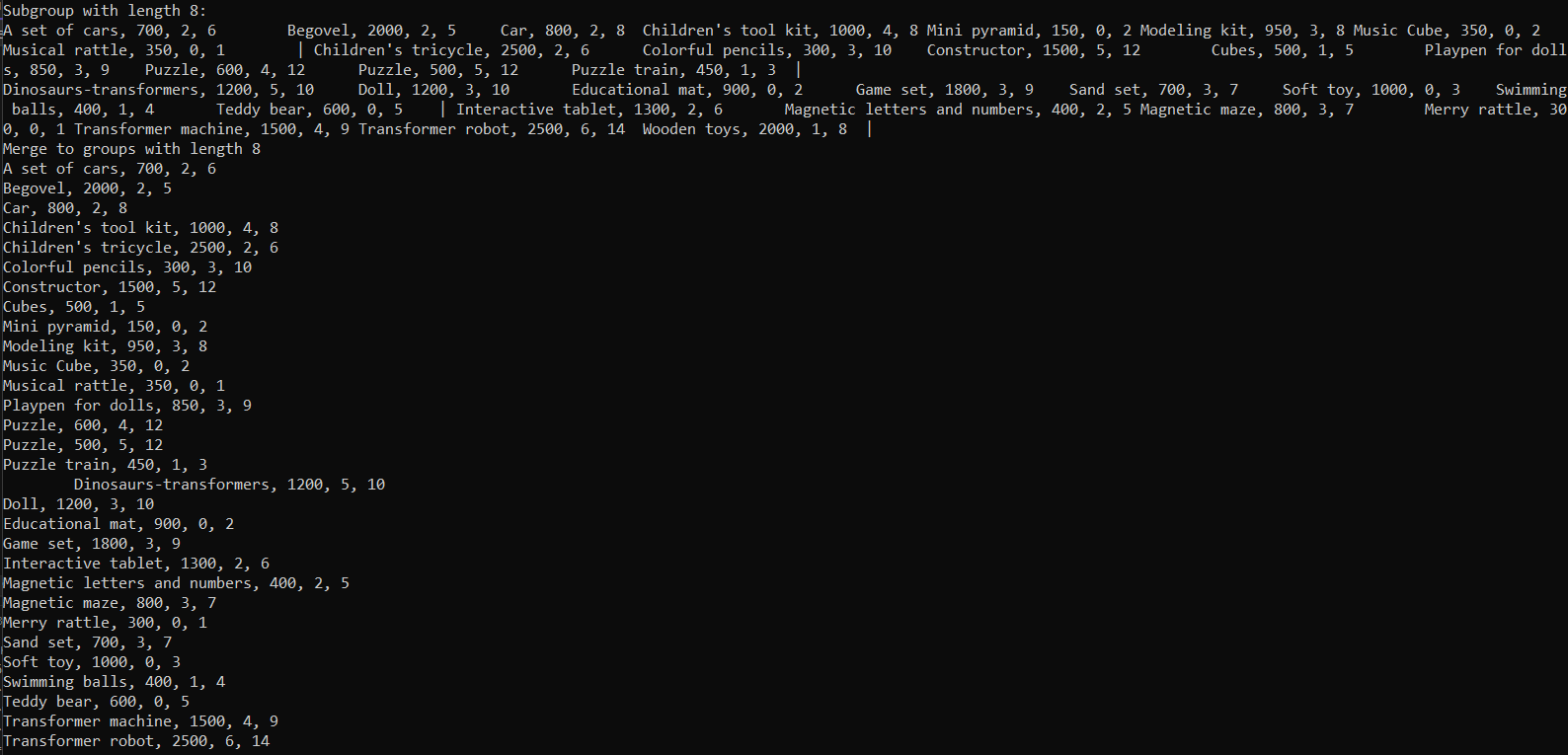


Рисунок 6 - Тестирование адаптированной программы

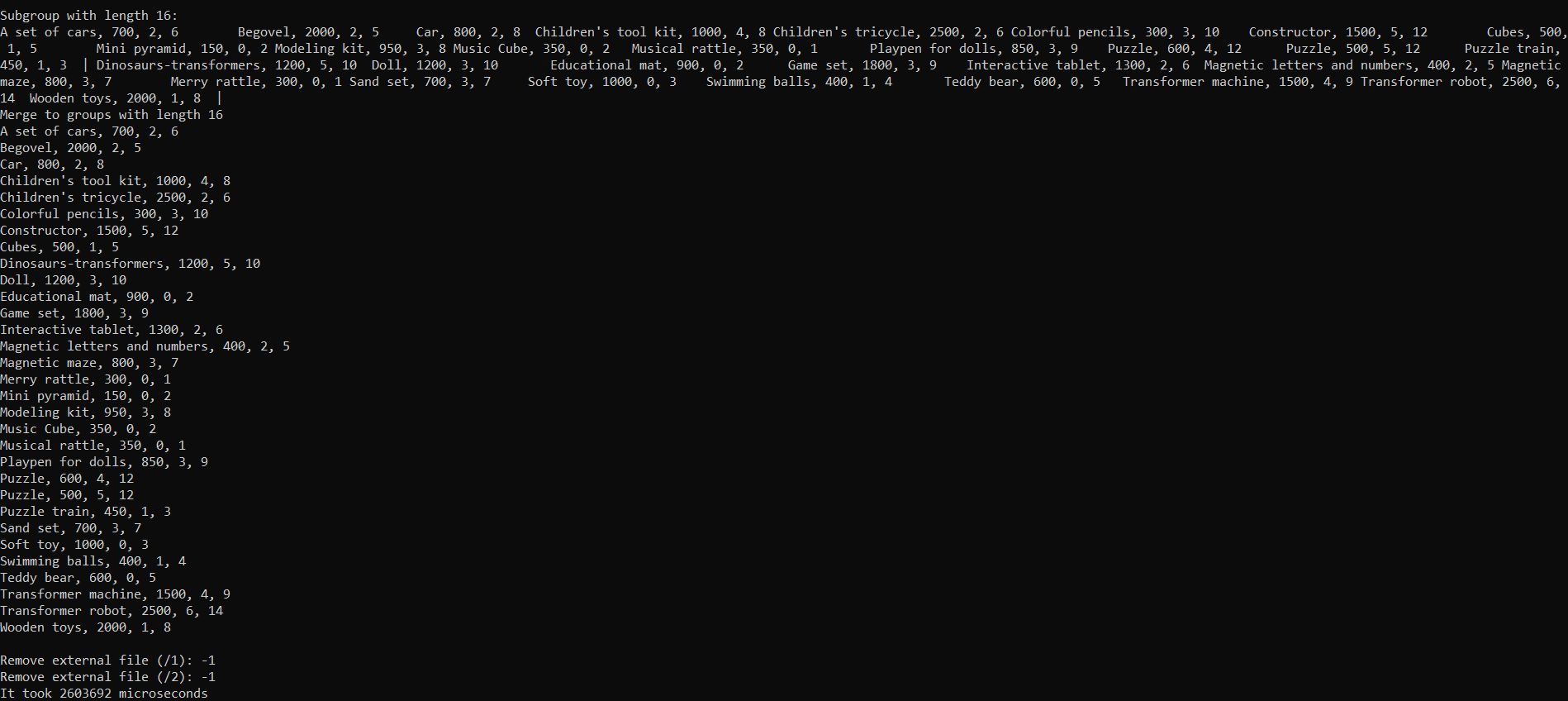


Рисунок 7 - Тестирование адаптированной программы

## **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 | 24375 |
| 16 | 423473 |
| 32 | 3285528 |

## **2.7 Вывод по заданию №1**

Сортировка прямым слиянием – это эффективный и стабильный алгоритм сортировки, который разделяет список на две части, сортирует каждую часть отдельно, а затем объединяет отсортированные части в один упорядоченный список. Этот алгоритм имеет временную сложность O(n log n) в любом случае, что делает его одним из самых эффективных методов сортировки. Время выполнения зависит от количества элементов в списке, но гарантированно не превышает n log n операций. Пространственная сложность алгоритма равна O(n).

Прямое слияние требует дополнительной памяти для сортировки, так как создает временные массивы для слияния. Это делает его не очень эффективным для больших объемов данных из-за потребления дополнительной памяти. Однако, несмотря на это, сортировка прямым слиянием более эффективна, чем простые, быстрые или усовершенствованные сортировки, так как позволяет эффективно сортировать как большие, так и маленькие списки. Это делает её универсальным методом для различных 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 Алгоритм решения алгоритма**

Сортировка естественного слияния (natural merge sort) - это способ сортировки данных, который использует принципы слияния (merge) для упорядочивания элементов. В этом алгоритме данные разделяются на отсортированные подсписки, которые затем сливаются вместе для получения отсортированного списка.

Основные шаги выполнения сортировки естественного слияния:

1. Разделение: исходный список делится на отсортированные подсписки.

2. Слияние: подсписки сливаются в большие отсортированные подсписки.

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

Преимущество сортировки естественного слияния состоит в том, что она эффективно сортирует данные, которые уже частично отсортированы. Кроме того, она не требует дополнительной памяти для хранения временных массивов, так как сортировка происходит на месте.

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.

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.

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

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

Реализуем данный алгоритм на языке C++(блок кода 4). Для реализации понадобятся такие библиотеки, как iostream, random, chrono, fstream, algorithm, vector, string.

| #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

### **2.3.2 Тестирование**

Стоит задача протестировать программу с заданным данными в файле А: 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

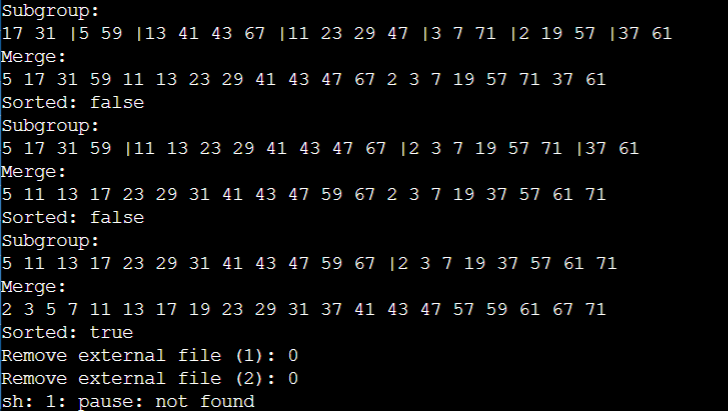


Рисунок 9 - Тестирование программы с входными данными из примера

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

Адаптируем программу к данным по условию индивидуального варианта. Магазин игрушек. Реализация кода состоит из двух частей. Из заголовочного файла "natural.h"(блок кода 5) и исходного файла main.cpp(блок кода 6).

| #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 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(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  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(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> //Вычисляет длину серии данных, считывая их из файла input\_series до конца. Кол-во прочитанных элементов сохраняется в переменной seriesLength\_. void NExternalMergeSort<SType, DSep>::computeSeriesLength(fstream& input\_series) {  this->seriesLength\_ = 0;  SType value{};  while (input\_series >> value)  {  this->seriesLength\_++;  } } template<class SType, char DSep> //Возвращает строку, представляющую внешний файл с данными, в зависимости от переданного параметра way. Файлы нумеруются с единицы.  string NExternalMergeSort<SType, DSep>::getExternalFilename(const unsigned int way) {  return "/" + to\_string(way + 1); } template<class SType, char DSep> //Создает и открывает вектор внешних файлов нужного размера и возвращает его. Названия файлов формируются с помощью функции getExternalFilename.  vector<fstream> NExternalMergeSort<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> //Закрывает и повторно открывает все внешние файлы из вектора ext с заданным режимом mode. Названия файлов берутся из функции getExternalFilename. 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> //Удаляет внешние файлы, используемые для сортировки. 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> //Повторно открывает входной файл с указанным режимом. 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 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(  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 <chrono> //Подключение пользовательского заголовочного файла "ems\_natural.h" #include "ems\_natural.h" using namespace std; //Определение структуры файла A, содержащего поля n, a, b, c. struct A {  string n;//Название игрушки  int a = 0;//цена игрушки  int b = 0;//возраст начальный  int c = 0;//возраст конечный }; //Перегрузка оператора >> для чтения значений типа 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;// Присваиваем свойству "n" объекта "rhv" значение переменной "n" rhv.n = n  int value = 0;// Создаем переменную "value" и считываем значение из потока "lhv" int value = 0; lhv >> value;  lhv >> value;// Считываем значение а переменную "value" из потока "lhv" lhv >> value  rhv.a = value;// Присваиваем свойству "a" объекта "rhv" значение переменной "value", считанной из потока "lhv" rhv.a = value  lhv.get();// Пропускаем символ '\n' в потоке "lhv" lhv.get()  lhv >> value;// Снова считываем значение в переменную "value" из потока "lhv" lhv >> value  rhv.b = value;// Присваиваем свойству "b" объекта "rhv" значение переменной "value" rhv.b = value  lhv.get();// Пропускаем символ '\n' в потоке "lhv" lhv.get()  lhv >> value;// Снова считываем значение в переменную "value" из потока "lhv" lhv >> value  rhv.c = value;// Присваиваем свойству "c" объекта "rhv" значение переменной "value" rhv.c = value  lhv.get(); // считывание  return lhv; } //Перегрузка оператора << для записи значений типа A 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();// Начало сортировки  //Открытие файла "A" в режиме чтения и записи.  const auto input\_file = "A";  fstream input(input\_file, ios\_base::in | ios\_base::out);  //Проверка успешности открытия файла  if (!input)  {  perror("Error with input file");  return EXIT\_FAILURE;  }  //Использование шаблонной функции NExternalMergeSort из заголовочного файла "ems\_simple.h"  NExternalMergeSort<A, '\n'>(input\_file).sort(input, [](const A& a, const A& b) { return a.n < b.n; });  input.close();//Закрытие входного файла  auto end = chrono::high\_resolution\_clock::now();// Конец сортировки  auto d = chrono::duration\_cast<chrono::microseconds>(end - start).count();// затраченное время  cout << "It took " << d << " microseconds" << endl;  system("pause");//Приостановка выполнения программы для ожидания ввода пользователя   return EXIT\_SUCCESS;//Возврат успешного завершения программы. } |
| --- |

Блок кода 6 - Файл main.cpp

## **3.5 Тестирование адаптированного кода**

Стоит задача протестировать программу с заданным данными в файле А. Результаты тестирования будут продемонстрированы на рисунке 10-20.

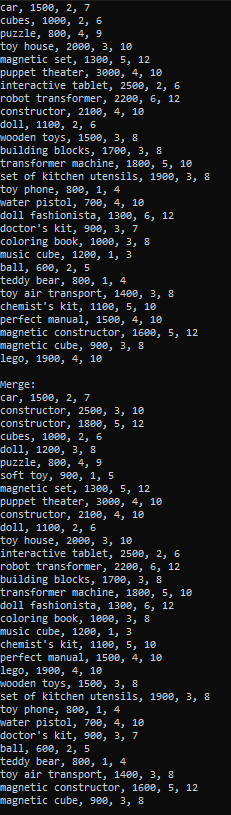


Рисунок 10 - Тестирование адаптированной программы

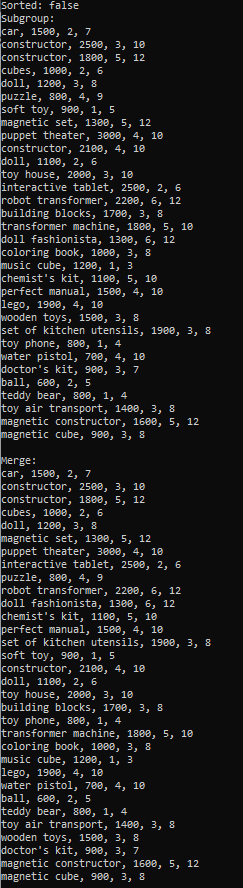


Рисунок 11 - Тестирование адаптированной программы

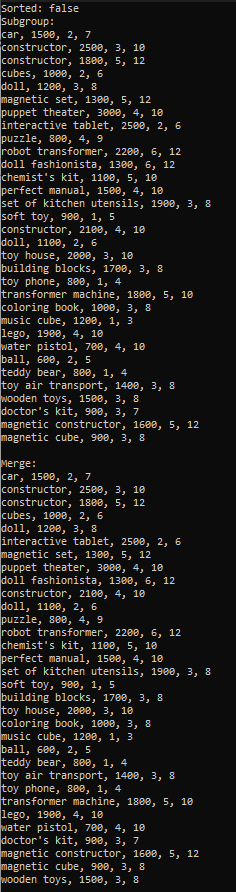


Рисунок 12 - Тестирование адаптированной программы

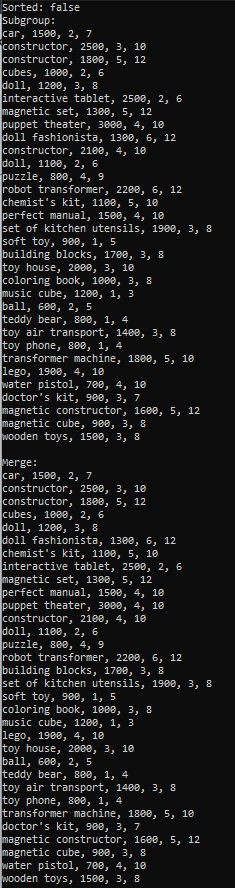


Рисунок 13 - Тестирование адаптированной программы

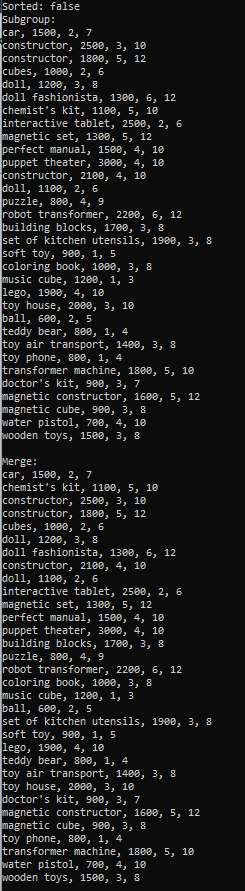


Рисунок 14 - Тестирование адаптированной программы

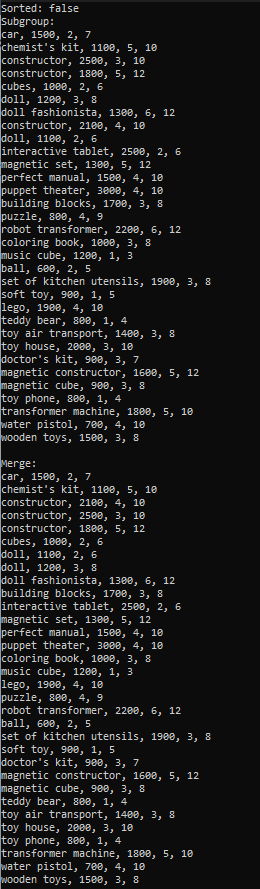


Рисунок 15 - Тестирование адаптированной программы

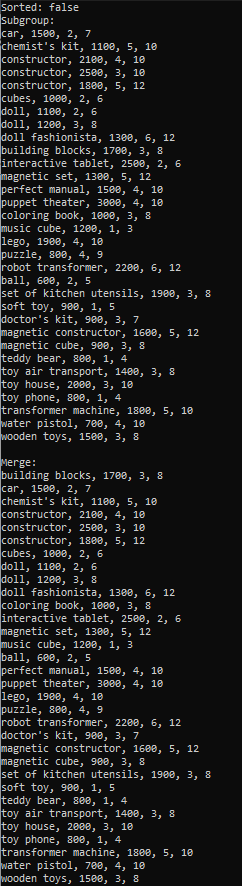


Рисунок 16 - Тестирование адаптированной программы

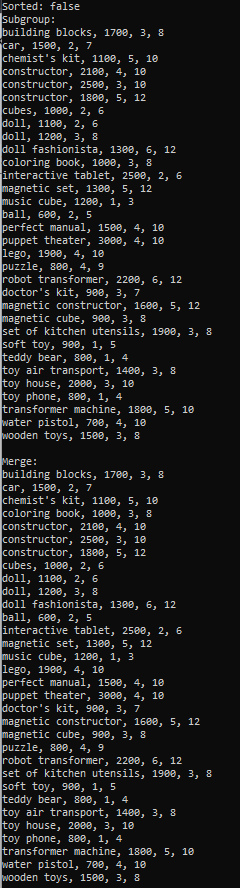


Рисунок 17 - Тестирование адаптированной программы

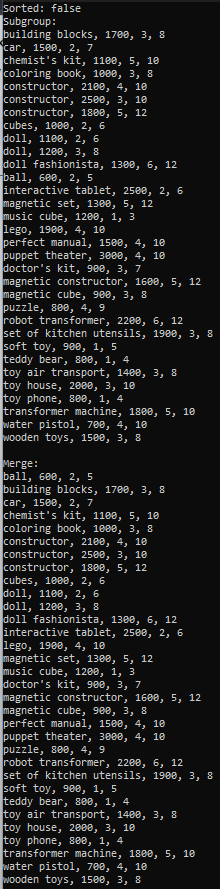


Рисунок 18 - Тестирование адаптированной программы

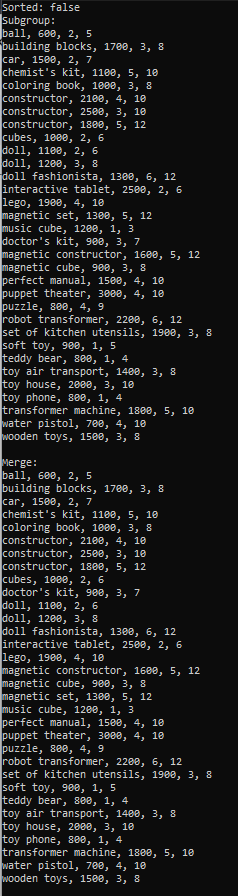


Рисунок 19 - Тестирование адаптированной программы



Рисунок 20 - Тестирование адаптированной программы

## **3.6 Практическая сложность алгоритма**

Для определения практической сложности алгоритма естественного слияния для файлов с увеличивающимся количеством записей, нужно учитывать количество операций сравнения и перемещения элементов в процессе сортировки. Докажем это утверждение с помощью тестирования на разном количестве записей(таб.2). Таким образом, практическая сложность алгоритма естественного слияния для файлов с увеличивающимся количеством записей остается примерно одинаковой и равна O(log n), где n - количество записей в файле.

Таблица 2 - Сводная таблица результатов

| **Количество записей** | **Время(мкс)** |
| --- | --- |
| 8 | 305474 |
| 16 | 824759 |
| 32 | 3184930 |

## **3.7 Вывод по заданию №2**

Сортировка естественным слиянием – это алгоритм, основанный на разделении списка на уже отсортированные подсписки с последующим их слиянием. Этот метод эффективен для сортировки больших списков, так как не требует дополнительной памяти. Практическая сложность сортировки естественным слиянием зависит от количества элементов в списке. В лучшем случае сложность составляет O(n), в худшем – O(n log n), однако в среднем случае она ближе к O(n log n). Пространственная сложность составляет O(1).

Время выполнения сортировки естественным слиянием также зависит от количества элементов. Чем больше элементов в списке, тем больше времени потребуется для выполнения алгоритма. Однако, даже при увеличении объема данных, этот метод занимает меньше времени, чем классические алгоритмы сортировки, требующие дополнительной памяти.

Сортировка естественным слиянием позволяет эффективно обрабатывать как большие, так и маленькие списки, что делает её универсальным методом для различных задач. В таблице 2 это подтверждается примерами. Этот алгоритм не требует дополнительной памяти, так как использует принцип разделения и слияния подсписков, что делает его особенно эффективным для больших списков с постоянной сложностью при увеличении количества записей.

В целом, сортировка естественным слиянием предпочтительна для больших списков, так как она не требует дополнительной памяти и обладает стабильной практической сложностью. В то время как прямое слияние может быть более удобным для небольших списков, оно становится менее эффективным при работе с большими объемами данных из-за дополнительного потребления 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).