



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

РТУ МИРЭА

Отчет по выполнению практического занятия 5

Тема: Алгоритмы внешних сортировок

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент

Хвостов В. В.

Группа

ИКБО-01-20

Москва 2021

Содержание

1	Преамбула	3
1.1	Сортировка массива случайных чисел методом естественного слияния	3
1.2	Сортировка случайного массива методом многофазного слияния	4
2	Задание 1	5
2.1	Постановка задачи	5
2.2	Подготовка тестовых данных	5
2.3	Алгоритм внешней сортировки "прямое слияние"	8
3	Задание 2	16
3.1	Постановка задачи	16
3.2	Алгоритм внешней сортировки "естественное слияние"	16
	Выводы	23
	Список информационных источников	23

Преамбула

1.1. Сортировка массива случайных чисел методом естественного слияния

Рассмотрим файл **A**, содержащий массив, заполненный случайными числами в диапазоне [1..15].

5 1 9 3 12 2 4 7 6 1 10 8 11 14 12 Сортировка массива методом естественного слияния будет осуществляться следующим образом:

1. Выделяем серии (упорядоченные подпоследовательности:

A: [5], [1, 9], [3, 12], [2, 4, 7], [6], [1, 10], [8, 11, 14], [12].

Получилось 8 серий.

2. Поочередно записываем эти серии в файлы **B** и **C**:

B: [5], [3, 12], [6], [8, 11, 14].

C: [1, 9], [2, 4, 7], [1, 10], [12].

3. Сольем серии в файл **A** и выделим их.

A: [1, 5, 9], [2, 3, 4, 7, 12], [1, 6, 10], [8, 11, 12, 14].

4. Поочередно записываем эти серии в файлы **B** и **C**:

B: [1, 5, 9], [1, 6, 10].

C: [2, 3, 4, 7, 12], [8, 11, 12, 14].

5. Сольем серии в файл **A** и выделим их.

A: [1, 2, 3, 4, 5, 7, 12], [1, 6, 8, 10, 11, 12, 14].

6. Поочередно записываем эти серии в файлы **B** и **C**:

B: [1, 2, 3, 4, 5, 7, 12].

C: [1, 6, 8, 10, 11, 12, 14].

7. Сольем серии в файл **A**.

A: [1, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 12, 14].

Получилась одна серия, что означает конец сортировки естественного слияния.

1.2. Сортировка случайного массива методом многофазного слияния

Рассмотрим файл А, содержащий массив, заполненный случайными числами в диапазоне [1..15].

5 1 9 3 12 2 4 7 6 1 10 8 11 14 12 Сортировка массива методом многофазного слияния будет осуществляться следующим образом:

1. Выделяем серии (упорядоченные подпоследовательности):

А: [5], [1, 9], [3, 12], [2, 4, 7], [6], [1, 10], [8, 11, 14], [12].

Получаем 8 серий.

2. Распределяем эти серия по двум файлам **В** и **С** в соотношении $\frac{5}{8}$ и $\frac{3}{8}$ (5 и 3 - числа Фибоначчи).

В: [5], [3, 12], [6], [8, 11, 14], [12].

С: [1, 9], [2, 4, 7], [1, 10].

3. Сливаем серии с парами в файл **А**.

А: [1, 5, 9], [2, 3, 4, 7, 12], [1, 6, 10].

В: [8, 11, 14], [12].

4. Сливаем серии с парами в файл **С**.

А: [1, 6, 10].

С: [1, 5, 8, 9, 11, 14], [2, 3, 4, 7, 12, 12].

5. Сливаем серии с парами в файл **В**.

В: [1, 1, 5, 6, 8, 9, 10, 11, 14].

С: [2, 3, 4, 7, 12, 12].

6. Сливаем серии с парами в файл **А**.

А: [1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 12, 14].

Получилась одна серия, что означает конец сортировки многофазного слияния.

Задание 1

2.1. Постановка задачи

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

Структура файла в соответствии с персональным вариантом : Список экспортируемых товаров. Об отдельном товаре хранятся данные: Наименование товара, Страна импортирующая товар, Количество(в штуках).

2.2. Подготовка тестовых данных

Код класса для генерации тестовых данных (файл countries - список всех существующих стран)

```
#include <fstream>
#include <random>
#include <string>
#include <vector>

class Generator {
public :
    Generator () {
        std :: ifstream  fin ;
        fin .open(" countries ");
        std :: string  field ;
        while ( std :: getline ( fin ,  field )){
            countries .push_back( field );
        }
        fin .close ();
    }

    void generate ( int64_t  data_size , const std :: string & file_name = " test . txt ") {
        std :: ofstream  fout ;
        fout .open( file_name );
        for ( int  i = 0; i < data_size ; ++i) {
            fout << getRandomProductName() << " "
                << getRandomCountry() << " "
                << getRandomAmountOfTheProduct() << '\n';
        }
        fout .close ();
    }

private :
```

```

const int64_t BIG_NUM=10'000'000;
std :: vector<std :: string > countries ;

const std :: vector<std :: string > product_names = {
    "Apple", "Banana", "Cucumber", "Tomato",
    " Garlic ", "Lemon", "Orange", "Peach",
    " Grapefruit ", "Avocado", "Pear", " Black_beans ",
    " White_beans ", " Chickpeas ", "Corn", " Beetroot ",
    "Pumpkin", " Butter_squash ", " Pear ", " Carrot ",
    " Cilantro ", "Lime", " Phig ", " Potato ",
    " Sweet_potato ", " Broccoli ", " Cauliflower ", " Brussel_sprouts ",
    "Tomato", "Cumin", " Turnip ", " Zucchini ", "Coconut",
    " Parsley ", " Artichok ", "Peas", "Onion", " Lettuce ",
    "Red_cabbage", " Spinach ", " Green_beans ", " Strawberry ",
    " Blueberry ", " Rasberry ", " Ginger ", " Cranberry ",
    "Plume", "Prun", " Abricot ", "Grape", " Raisins ",
    " White_rice ", "Wheat", "Rye", "Quinoa", "Bulgur",
    "Brown_rice", " Oats ", " Radish ", "Cinnamon",
    " Vanilla ", "Sugar", " Salt ", " Vinegar ",
    " Olive_oil ", " Sunflower_oil ", " Coconut_oil ",
    " Sesame_oil ", " Canola_oil ", " Avocado_oil ",
    "Milk", " Kefir ", "Heavy_cream", " Half_and_half ",
    " Yogurt ", " Parmesan_cheese ", " Cottage_cheese ",
    " Mozzarella_cheese ", " Cheddar_cheese ",
    " Gorgonzolla_cheese ", "Tofu", "Soy_beans",
    "Edamami", "Soy_sauce", " Mayonaisse ", " Pumpkin_seeds ",
    " Sesame_seeds ", " Butter ", "White_wine", "Red_wine",
    "Maple_syrup", "Jam", "Musli", "Granola",
    "Buckwheat", " All_purpose_wheat_flour ", " Bread_wheat_flour ",
    " Cake_wheat_flour ", " Whole_wheat_flour ", " Rice_flour ",
    "Whole_chicken", " Whole_turkey ", "Beef_back",
    "Lamb_ribs", " Pork_tenderloin ", " Chicken_eggs ",
    " Pineapple ", " Asparagus ", " Duck_breast ", "Leek",
    " Bell_pepper ", " Saurekrat ", "Red_onion", " Chili_pepper ",
    "Mango", "Watermelon", "Melon", " Scallion "
};

std :: string getRandomCountry() {
    std :: random_device rd;
    std :: mt19937 gen(rd());
    std :: uniform_int_distribution <int> dist (0, countries . size () - 1);
    return countries [ dist (gen)];
}

std :: string getRandomProductName() {
    std :: random_device rd;
    std :: mt19937 gen(rd());
    std :: uniform_int_distribution <int> dist (0, product_names . size () - 1);
    return product_names [ dist (gen)];
}

```

```

}

int64_t getRandomAmountOfTheProduct() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<int> dist(0, BIG_NUM);
    return dist(gen);
}
};

```

Пример сгенерированного файла показан на рисунке 1.

```

Mozarella_cheese Mexico 3499730
Prun Maldives 4742575
White_beans Australia 8185518
Abricot Marshall Islands 4089959
Jam Bangladesh 6035050
Peach Honduras 2179793
White_rice Bahrain 9551296
Lettuce India 9945985
Duck_breast Serbia 9735483
Onion Hungary 6684117
Vanilla Tanzania 8306341
Beef_back Paraguay 8036060
Whole_chicken Israel 9895919
Phig Madagascar 4163915
Cheddar_cheese Tanzania 5329267
Plume St Kitts & Nevis 1913244
Brown_rice Thailand 7056164
Butter_squash Korea North 2529781
Whole_turkey Yemen 9115103
Strawberry Congo 3105226
Potato Malawi 6441410
Tomato Grenada 2251310
Tomato Belarus 4884907

```

Рис. 1 - Пример сгенерированного входного файла

2.3. Алгоритм внешней сортировки "прямое слияние"

Описание алгоритма

Внешняя сортировка прямого слияния требует наличие нескольких вспомогательных файлов (в нашем случае двух). Каждый шаг сортировки состоит из фазы разделения и слияния.

Разделение предполагает разбиение текущей последовательности на несколько подпоследовательностей (порций) и поочередную запись в вспомогательные файлы. Количество элементов в порции на каждом этапе увеличивается в 2 раза.

Слияние состоит в чтении вспомогательных файлов и упорядочивании соответствующих порций. Для каждой порции сравниваются 2 записи, меньшая из них записывается в исходный файл, и считывается новая запись из файла, в котором содержалась меньшая. При достижении конца порции одного файла оставшиеся записи другого файла переносятся без изменений. Данные операции повторяются пока не будет достигнут конец одного из файлов, после чего переписываются незатронутые записи из другого.

Код сортировки

```
#include <iostream>
#include <fstream>
#include <string>
// string , string , int64_t
#include "generator .cpp"
#include "parser .cpp"
#include "sort_utilities .cpp"

namespace defined {
    constexpr char main_file_name [] = " test . txt ";
    constexpr char buffer1_file_name [] = "b. txt ";
    constexpr char buffer2_file_name [] = "c. txt ";
    constexpr int64_t amount_of_lines = 1'000'000; // '000;
    constexpr long double some_strange_constant = 1.188 ;
}

void forwardDivision (const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ,
    int64_t coef);
void forwardMerge (const std :: string & output_name ,
    const std :: string & input_name1 , const std :: string & input_name2 ,
    int64_t coef);
```



```

void forwardMergeSort ( const std :: string & input_name = defined :: main_file_name ,
    const std :: string & buffer_name1 = defined :: buffer1_file_name ,
    const std :: string & buffer_name2 = defined :: buffer2_file_name ,
    int64_t line_amount = defined :: amount_of_lines ) {
    for ( int64_t coef = 1; coef < line_amount ; coef <= 1) {
        forwardDivision ( input_name , buffer_name1 , buffer_name2 , coef );
        forwardMerge ( input_name , buffer_name1 , buffer_name2 , coef );
    }
}

void forwardDivision ( const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ,
    int64_t coef ) {
    std :: ifstream fin ;
    std :: ofstream buff1 , buff2 ;
    int counter = 0;
    std :: string line , key;
    bool first_buff = true ;

    fin . open ( input_name );
    buff1 . open ( output_name1 );
    buff2 . open ( output_name2 );
    while ( std :: getline ( fin , line ) ) {
        if ( counter ++ == coef ) {
            first_buff = ! first_buff ;
            counter = 1;
        }
        ( first_buff ? buff1 : buff2 ) << line << '\n';
    }
    fin . close () ; buff1 . close () ; buff2 . close () ;
}

void forwardMerge ( const std :: string & output_name ,
    const std :: string & input_name1 , const std :: string & input_name2 ,
    int64_t coef ) {
    std :: ofstream fout ;
    std :: ifstream buff1 , buff2 ;
    std :: string line1 , line2 , country_name1 , country_name2 ;
    bool buff1_end = false , buff2_end = false ;
    ProductParser parser ;

    fout . open ( output_name );
    buff1 . open ( input_name1 );
    buff2 . open ( input_name2 );
    std :: getline ( buff1 , line1 ) ; std :: getline ( buff2 , line2 );
    for ( ;; ) {
        int64_t counter1 = 0 , counter2 = 0;

```

```

while ( counter1 < coef && counter2 < coef) {
    if (! line1 .empty() &&! line2 .empty()) {
        country_name1 = parser .getCountryName( line1 );
        country_name2 = parser .getCountryName( line2 );
    }
    if ( country_name1 < country_name2 ) {
        ++counter1 ;
        fout << line1 << '\n';
        std :: getline ( buff1 , line1 );
        if ( buff1 .eof()) {
            buff1_end = true;
            break;
        }
    } else {
        ++counter2 ;
        fout << line2 << '\n';
        std :: getline ( buff2 , line2 );
        if ( buff2 .eof()) {
            buff2_end = true;
            break;
        }
    }
}
if (! buff1_end ) {
    for (; counter1 < coef; ++counter1 ) {
        fout << line1 << '\n';
        std :: getline ( buff1 , line1 );
        if ( buff1 .eof()) {
            buff1_end = true;
            break;
        }
    }
}
if (! buff2_end ) {
    for (; counter2 < coef; ++counter2 ) {
        fout << line2 << '\n';
        std :: getline ( buff2 , line2 );
        if ( buff2 .eof()) {
            buff2_end = true;
            break;
        }
    }
}
if ( buff1_end || buff2_end ) {
    break;
}
if (! buff1_end ) {

```

```

while ( std :: getline ( buff1 , line1 )) {
    fout << line1 << '\n';
}
}
if ( ! buff2_end ) {
    while ( std :: getline ( buff2 , line2 )) {
        fout << line2 << '\n';
    }
}

fout . close (); buff1 . close (); buff2 . close ();
}

```

Тестирование алгоритма

Для тестирования алгоритма использовался класс TimeCounter.

```

class TimeCounter {
public :
    TimeCounter() {
//      std :: clog << " Start  clock \n";
        start = std :: chrono :: steady_clock :: now();
    }
    ~TimeCounter() {
        auto end = std :: chrono :: steady_clock :: now();
        std :: chrono :: duration <double, std :: milli> diff = end - start ;
        std :: clog << "End clock , time = " << diff . count () << " ms" << std :: endl ;
    }
private :
    std :: chrono :: time_point <std :: chrono :: steady_clock > start ;
};

```

Также использовалась отладочная версия данной сортировки.

```

void forwardDivisionLog ( const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ,
    int64_t coef , int64_t & comparisons , int64_t & moves);
void forwardMergeLog( const std :: string & output_name ,
    const std :: string & input_name1 , const std :: string & input_name2 ,
    int64_t coef , int64_t & comparisons , int64_t & moves);

void forwardMergeSortLog ( const std :: string & input_name = defined :: main_file_name ,
    const std :: string & buffer_name1 = defined :: buffer1_file_name ,
    const std :: string & buffer_name2 = defined :: buffer2_file_name ,
    int64_t line_amount = defined :: amount_of_lines ) {
    int64_t comparisons = 0, moves = 0;
    for ( int64_t coef = 1; coef < line_amount ; coef <= 1) {
        forwardDivisionLog ( input_name , buffer_name1 , buffer_name2 , coef ,
            comparisons , moves);
    }
}

```

```

        forwardMergeLog(input_name, buffer_name1, buffer_name2, coef,
                        comparisons, moves);
    }
    std::clog << "Comparisons: " << comparisons << std::endl;
    std::clog << "Moves: " << moves << std::endl;
    std::clog << "T_prac: " << comparisons + moves << std::endl;
}

void forwardDivisionLog(const std::string & input_name,
    const std::string & output_name1, const std::string & output_name2,
    int64_t coef, int64_t & comparisons, int64_t & moves) {
    std::ifstream fin;
    std::ofstream buff1, buff2;
    int counter = 0;
    std::string line, key;
    bool first_buff = true;

    fin.open(input_name);
    buff1.open(output_name1);
    buff2.open(output_name2);
    while (std::getline(fin, line)) {
        if (counter++ == coef) {
            first_buff = !first_buff;
            counter = 1;
        }
        (first_buff ? buff1 : buff2) << line << '\n';
    }
    fin.close(); buff1.close(); buff2.close();
}

void forwardMergeLog(const std::string & output_name,
    const std::string & input_name1, const std::string & input_name2,
    int64_t coef, int64_t & comparisons, int64_t & moves) {
    std::ofstream fout;
    std::ifstream buff1, buff2;
    std::string line1, line2, country_name1, country_name2;
    bool buff1_end = false, buff2_end = false;
    ProductParser parser;

    fout.open(output_name);
    buff1.open(input_name1);
    buff2.open(input_name2);
    std::getline(buff1, line1); std::getline(buff2, line2);
    for (;;) {
        int64_t counter1 = 0, counter2 = 0;
        while (counter1 < coef && counter2 < coef) {
            comparisons += 2; ++moves;
            if (!line1.empty() && !line2.empty()) {

```

```

country_name1 = parser .getCountryName( line1 );
country_name2 = parser .getCountryName( line2 );
}
if ( country_name1 < country_name2 ) {
    ++counter1 ;
    fout << line1 << '\n';
    std :: getline ( buff1 , line1 );
    if ( buff1 .eof() ) {
        buff1_end = true;
        break;
    }
} else {
    ++counter2 ;
    fout << line2 << '\n';
    std :: getline ( buff2 , line2 );
    if ( buff2 .eof() ) {
        buff2_end = true;
        break;
    }
}
}
if ( ! buff1_end ) {
    for ( ; counter1 < coef; ++counter1 ) {
        fout << line1 << '\n';
        ++moves;
        std :: getline ( buff1 , line1 );
        if ( buff1 .eof() ) {
            buff1_end = true;
            break;
        }
    }
}
if ( ! buff2_end ) {
    for ( ; counter2 < coef; ++counter2 ) {
        fout << line2 << '\n';
        ++moves;
        std :: getline ( buff2 , line2 );
        if ( buff2 .eof() ) {
            buff2_end = true;
            break;
        }
    }
}
if ( buff1_end || buff2_end ) {
    break;
}
if ( ! buff1_end ) {

```

```

while ( std :: getline ( buff1 , line1 )) {
    fout << line1 << '\n';
    ++moves;
}
}
if ( ! buff2_end ) {
    while ( std :: getline ( buff2 , line2 )) {
        fout << line2 << '\n';
        ++moves;
    }
}

fout . close (); buff1 . close (); buff2 . close ();
}

```

Результаты тестирования алгоритма на файле из миллиона записей можно увидеть на рисунке 2.



```

Milk Afghanistan 5773249
Garlic Afghanistan 4834644
Carrot Afghanistan 4586629
Whole_chicken Afghanistan 1629261
Lemon Afghanistan 7540876
Duck_breast Afghanistan 1982477
Soy_beans Afghanistan 8552818
White_beans Afghanistan 5077215
Sunflower_oil Afghanistan 7063354
Pumpkin Afghanistan 9841011
Ginger Afghanistan 4024652
Spinach Afghanistan 2079022
Heavy_cream Afghanistan 3669070
Plume Afghanistan 2322613
Cumin Afghanistan 543275
Garlic Afghanistan 1851948
Coconut Afghanistan 1890212
Pork_tenderloin Afghanistan 9889524
Lime Afghanistan 3892506
Bulgur Afghanistan 8671577

```

Рис. 2 - Первые строки файла после применения сортировки прямого слияния

Сводная таблица тестирования

```

Enter the size of an array: 100
Comparisons: 568
Moves: 2040
T_prac: 2608
End clock, time = 5.63553 ms
vitalir@swiftly:~/Documents/education
Enter the size of an array: 1000
Comparisons: 8710
Moves: 20022
T_prac: 28732
End clock, time = 14.6317 ms
vitalir@swiftly:~/Documents/education
Enter the size of an array: 10000
Comparisons: 123530
Moves: 199946
T_prac: 323476
End clock, time = 144.395 ms
vitalir@swiftly:~/Documents/education
Enter the size of an array: 100000
Comparisons: 1564578
Moves: 1999935
T_prac: 3564513
End clock, time = 1755.91 ms
vitalir@swiftly:~/Documents/education
Enter the size of an array: 1000000
Comparisons: 18690351
Moves: 19999922
T_prac: 38690273
End clock, time = 24018.9 ms
vitalir@swiftly:~/Documents/education

```

Рис. 3 - Результаты тестирования сортировки слиянием

Таблица 1 - Сводная таблица тестирования сортировки прямым слиянием

n	$T(n)$	$T_T = f(C + M)$	$T_{\Pi} = C_{\Phi} + M_{\Phi}$
100	5.63553 мс	$\Theta(n \log n)$	2608
1000	14.6317 мс		28732
10000	144.395 мс		323476
100000	1755.91 мс		3564513
1000000	24018.9 мс		38690273

Задание 2

3.1. Постановка задачи

Реализовать алгоритм внешней сортировки «естественное слияние» для данных, в соответствии с персональным вариантом (указан в постановке задачи задания No1).

3.2. Алгоритм внешней сортировки "естественное слияние"

Описание сортировки

Внешняя сортировка естественного слияния работает над упорядоченными последовательностями – сериями. Фазы разделения и слияния осуществляются над сериями, не всегда имеющими одинаковый размер (в отличие от прямого слияния).

Фаза разделения включает поиск и попеременную запись серий максимальной длины из исходного файла в вспомогательные. Процедура повторяется до конца файла.

Фаза слияния включает упорядочивание серий, получаемых из вспомогательных файлов. Алгоритм фазы слияния идентичен сортировке прямого слияния. В итоге, из двух серий образуется одна, она и является отсортированной последовательностью.

Код сортировки

```
void naturalDivision (const std::string & input_name ,
    const std::string & output_name1 , const std::string & output_name2 );

void naturalMerge (const std::string & output_name ,
    const std::string & input_name1 , const std::string & input_name2 ,
    int64_t n , int64_t & groups );

void naturalMergeSort (const std::string & input_name ,
    const std::string & output_name1 , const std::string & output_name2 ,
    int64_t n) {
    int64_t series ;
    firstNaturalDivision (input_name , output_name1 , output_name2 , n);
    for (;;) {
        series = 0;
```



```

        naturalMerge ( input_name , output_name1 , output_name2 , n , series );
        if ( series == 1 ) {
            break;
        }
        naturalDivision ( input_name , output_name1 , output_name2 );
    }
}

void firstNaturalDivision ( const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ,
    int64_t n ) {
    std :: ifstream fin ;
    std :: ofstream buff1 , buff2 ;
    std :: string line , key;
    ProductParser parser ;
    std :: vector<std :: string> buf;
    buf.reserve ( n );

    fin.open( input_name );
    buff1.open( output_name1 );
    buff2.open( output_name2 );

    for ( int i = 0; i < n / 2; ++i ) {
        std :: getline ( fin , line );
        buf.push_back( line );
    }
    std :: sort ( buf.begin() , buf.end() , [ parser ]( const std :: string & line1 , const std :: string & line2 )
        { return parser.getCountryName( line1 ) < parser.getCountryName( line2 ); } );
    for ( const std :: string & line : buf ) {
        buff1 << line << '\n';
    }
    buf.clear ();

    for ( int i = 0; i < n / 2; ++i ) {
        std :: getline ( fin , line );
        buf.push_back( line );
    }
    std :: sort ( buf.begin() , buf.end() , [ parser ]( const std :: string & line1 , const std :: string & line2 )
        { return parser.getCountryName( line1 ) < parser.getCountryName( line2 ); } );
    for ( const std :: string & line : buf ) {
        buff2 << line << '\n';
    }

    fin.close (); buff1.close (); buff2.close ();
}

void naturalDivision ( const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ) {

```

```

std :: ifstream  fin ;
std :: ofstream  buff1 , buff2 ;
std :: string  line , key ;
bool  first_buff  = true ;

fin . open ( input_name ) ;
buff1 . open ( output_name1 ) ;
buff2 . open ( output_name2 ) ;
while ( std :: getline ( fin , line ) &&! line . empty () ) {
    ( first_buff ? buff1 : buff2 ) << line << '\n' ;
    first_buff  = ! first_buff ;
}
fin . close () ; buff1 . close () ; buff2 . close () ;
}

void  naturalMerge ( const  std :: string & output_name ,
    const  std :: string & input_name1 , const  std :: string & input_name2 ,
    int64_t  n , int64_t & groups ) {
    std :: ofstream  fout ;
    std :: ifstream  buff1 , buff2 ;
    std :: string  line1 , line2 , country_name1 , country_name2 ;
    ProductParser  parser ;

    fout . open ( output_name ) ;
    buff1 . open ( input_name1 ) ;
    buff2 . open ( input_name2 ) ;

    for ( int64_t  counter  = 0 ; counter < n ; ++groups ) {
        std :: getline ( buff1 , line1 ) ;
        std :: getline ( buff2 , line2 ) ;

        for ( ; ! line1 . empty () &&! line2 . empty () ; ++counter ) {
            country_name1  = parser . getCountryName ( line1 ) ;
            country_name2  = parser . getCountryName ( line2 ) ;
            if ( country_name1 < country_name2 ) {
                fout << line1 << '\n' ;
                std :: getline ( buff1 , line1 ) ;
            } else {
                fout << line2 << '\n' ;
                std :: getline ( buff2 , line2 ) ;
            }
        }
    }

    for ( ; ! line1 . empty () ; ++counter ) {
        fout << line1 << '\n' ;
        std :: getline ( buff1 , line1 ) ;
    }

    for ( ; ! line2 . empty () ; ++counter ) {
        fout << line2 << '\n' ;

```

```

        std :: getline ( buff2 , line2 );
    }
}
fout . close (); buff1 . close (); buff2 . close ();
}

```

Тестирование алгоритма

Для тестирования алгоритма использовался класс TimeCounter (код представлен в задании 1). Также использовалась отладочная версия данной сортировки.

```

void naturalDivisionLog ( const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ,
    int64_t & moves);
void naturalMergeLog ( const std :: string & output_name ,
    const std :: string & input_name1 , const std :: string & input_name2 ,
    int64_t n , int64_t & groups , int64_t & comparisons , int64_t & moves);

void naturalMergeSortLog ( const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ,
    int64_t n ) {
    int64_t series ;
    int64_t comparisons = 0 , moves = 0;

    // Compensation for quick sort two times in firstNaturalDivision
    long double coeff = defined :: some_strange_constant *n*std :: log(n);
    comparisons += 2 * static_cast < int64_t >( coeff );
    moves += 2 * static_cast < int64_t >( coeff / 5);

    firstNaturalDivision ( input_name , output_name1 , output_name2 , n);
    for ( ;; ) {
        series = 0;
        naturalMergeLog ( input_name , output_name1 , output_name2 , n ,
            series , comparisons , moves);
        if ( series == 1 ) {
            break;
        }
        naturalDivisionLog ( input_name , output_name1 , output_name2 , moves);
    }
    std :: clog << "Comparisons: " << comparisons << std :: endl ;
    std :: clog << "Moves: " << moves << std :: endl ;
    std :: clog << "T_prac: " << comparisons + moves << std :: endl ;
}

void naturalDivisionLog ( const std :: string & input_name ,
    const std :: string & output_name1 , const std :: string & output_name2 ,
    int64_t & moves) {

```

```

std :: ifstream  fin ;
std :: ofstream  buff1 , buff2 ;
std :: string  line , key ;
bool  first_buff  = true ;

fin . open ( input_name ) ;
buff1 . open ( output_name1 ) ;
buff2 . open ( output_name2 ) ;
while ( std :: getline ( fin , line ) &&! line . empty () ) {
    ++moves;
    ( first_buff  ? buff1 : buff2 ) << line << '\n' ;
    first_buff  = ! first_buff ;
}
fin . close () ; buff1 . close () ; buff2 . close () ;
}

void  naturalMergeLog ( const  std :: string & output_name ,
    const  std :: string & input_name1 , const  std :: string & input_name2 ,
    int64_t  n , int64_t & groups , int64_t & comparisons , int64_t & moves ) {
    std :: ofstream  fout ;
    std :: ifstream  buff1 , buff2 ;
    std :: string  line1 , line2 , country_name1 , country_name2 ;
    ProductParser  parser ;

    fout . open ( output_name ) ;
    buff1 . open ( input_name1 ) ;
    buff2 . open ( input_name2 ) ;

    for ( int64_t  counter  = 0 ; counter  < n ; ++groups ) {
        std :: getline ( buff1 , line1 ) ;
        std :: getline ( buff2 , line2 ) ;

        for ( ; ! line1 . empty () &&! line2 . empty () ; ++counter ) {
            country_name1  = parser . getCountryName ( line1 ) ;
            country_name2  = parser . getCountryName ( line2 ) ;
            ++comparisons ;
            if ( country_name1  < country_name2 ) {
                fout << line1 << '\n' ;
                ++moves;
                std :: getline ( buff1 , line1 ) ;
            } else {
                fout << line2 << '\n' ;
                ++moves;
                std :: getline ( buff2 , line2 ) ;
            }
        }
    }

    for ( ; ! line1 . empty () ; ++counter ) {
        fout << line1 << '\n' ;

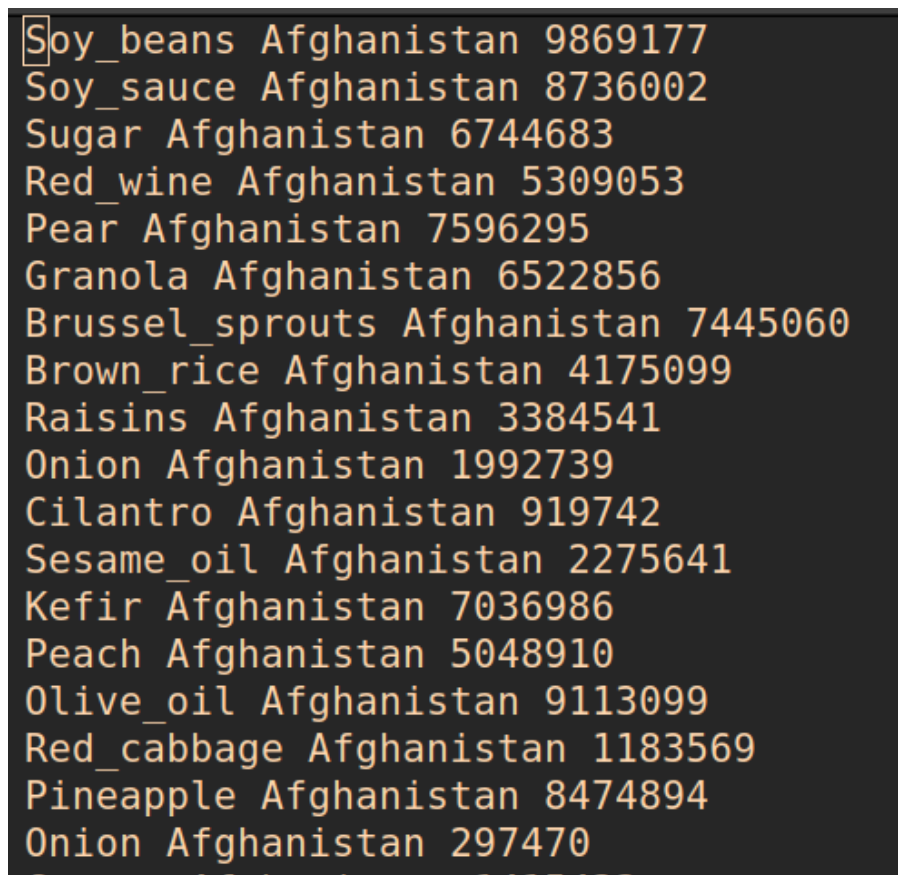
```

```

    ++moves;
    std :: getline ( buff1 , line1 );
}
for ( ; ! line2 .empty(); ++counter ) {
    fout << line2 << '\n';
    ++moves;
    std :: getline ( buff2 , line2 );
}
}
fout . close (); buff1 . close (); buff2 . close ();
}

```

Результаты тестирования алгоритма на файле из миллиона записей можно увидеть на рисунке 4.



```

Soy_beans Afghanistan 9869177
Soy_sauce Afghanistan 8736002
Sugar Afghanistan 6744683
Red_wine Afghanistan 5309053
Pear Afghanistan 7596295
Granola Afghanistan 6522856
Brussel_sprouts Afghanistan 7445060
Brown_rice Afghanistan 4175099
Raisins Afghanistan 3384541
Onion Afghanistan 1992739
Cilantro Afghanistan 919742
Sesame_oil Afghanistan 2275641
Kefir Afghanistan 7036986
Peach Afghanistan 5048910
Olive_oil Afghanistan 9113099
Red_cabbage Afghanistan 1183569
Pineapple Afghanistan 8474894
Onion Afghanistan 297470

```

Рис. 4 - Первые строки файла после применения сортировки естественного слияния

Сводная таблица тестирования

```

Enter the size of an array: 100
Comparisons: 1192
Moves: 318
T_prac: 1510
End clock, time = 1.41936 ms
vitalir@swiftly:~/Documents/educatio
Enter the size of an array: 1000
Comparisons: 17408
Moves: 4282
T_prac: 21690
End clock, time = 8.63991 ms
vitalir@swiftly:~/Documents/educatio
Enter the size of an array: 10000
Comparisons: 228809
Moves: 53766
T_prac: 282575
End clock, time = 95.831 ms
vitalir@swiftly:~/Documents/educatio
Enter the size of an array: 100000
Comparisons: 2835222
Moves: 647094
T_prac: 3482316
End clock, time = 1154.2 ms
vitalir@swiftly:~/Documents/educatio
Enter the size of an array: 1000000
Comparisons: 33823067
Moves: 7565130
T_prac: 41388197
End clock, time = 15930 ms

```

Рис. 5 - Результаты тестирования сортировки естественного слияния

Таблица 2 - Сводная таблица тестирования сортировки естественным слиянием

n	$T(n)$	$T_T = f(C + M)$	$T_{\Pi} = C_{\Phi} + M_{\Phi}$
100	1.41936 мс	$\Theta(n \log n)$	1510
1000	8.63991 мс		21690
10000	95.831 мс		282575
100000	1154.2 мс		3483216
1000000	15930 мс		41388197

Выводы

В ходе выполнения работы были реализованы алгоритмы внешних сортировок "прямое слияние" и "естественное слияние". Исходя из времени выполнения алгоритмов на больших входных данных на одной и той же машине, алгоритм сортировки "естественное слияние" намного быстрее алгоритма "прямое слияние" несмотря на одинаковую вычислительную и емкостную сложность: $\Theta(n \log n)$ и $O(n)$ соответственно. Это объясняется тем, что в алгоритме "естественное слияние" задействован также и алгоритм внутренней сортировки, который работает значительно быстрее за счет работы с внутренней памятью (т.к. на работу с устройствами внешней памяти тратится значительное время по сравнению с внутренними). Из этого также можно заметить зависимость скорости работы алгоритма от размера данных, которые хранятся во внутренней памяти: чем больше таких данных и меньше внешних, тем быстрее работает алгоритм.

Список информационных источников

1. Thomas H. Cormen, Clifford Stein и другие: Introduction to Algorithms, 3rd Edition. Сентябрь 2009. The MIT Press.
2. B. Strousrup: A Tour of C++ (2nd Edition). Июль 2018. Addison-Wesley.
3. Quick sort // Wikipedia
[Электронный ресурс]. URL:
https://en.wikipedia.org/wiki/Quick_sort (Дата обращения: 26.04.2021)
4. Jon Bentley, Douglas McILROY: Engineering a sort function. Ноябрь 1993. John Wiley & Sons.
5. Курс Algorithms, part 1 // Coursera [Электронный ресурс]. URL:
<https://www.coursera.org/learn/algorithms-part1> (Дата обращения: 26.04.2021)