**Содержание**

[Введение 4](#_Toc184060835)

[1. Теоретическая основа 5](#_Toc184060836)

[1.1 Принцип сортировки слиянием 5](#_Toc184060837)

[1.2 Основные понятия 6](#_Toc184060838)

[1.3 Внешняя сортировка прямым слиянием 7](#_Toc184060839)

[1.4 Внешняя сортировка естественным слиянием 8](#_Toc184060840)

[2. Анализ поставленной задачи 10](#_Toc184060841)

[3. Проектирование используемых типов данных 11](#_Toc184060842)

[4. Реализация алгоритмов 12](#_Toc184060843)

[4.1 Сортировка прямым слиянием 12](#_Toc184060844)

[4.2 Сортировка естественным слиянием 15](#_Toc184060845)

[5. Анализ эффективности алгоритмов 18](#_Toc184060846)

[Заключение 20](#_Toc184060847)

[Список использованной литературы 21](#_Toc184060848)

[Приложение 1 3](#_Toc184060849)

# Введение

Современные информационные системы требуют эффективного управления большими объемами данных, особенно когда речь идет о работе с массивными наборами информации, которые невозможно разместить целиком в оперативной памяти компьютера. В таких случаях применяются алгоритмы внешней сортировки, позволяющие упорядочивать данные, хранящиеся на внешних носителях, такими как жесткие диски или твердотельные накопители. Одним из наиболее популярных методов внешней сортировки является метод прямого и естественного слияния.

Прямое и естественное слияние представляют собой эффективные подходы к сортировке больших объемов данных, обеспечивая высокую производительность даже при ограниченных ресурсах оперативной памяти. Эти методы позволяют разбить исходный набор данных на небольшие части, которые затем сортируются отдельно и объединяются в единое упорядоченное множество. Данный подход широко используется в различных областях, включая базы данных, обработку больших данных и вычислительную математику.

Целью данной курсовой работы является изучение принципов работы алгоритмов прямой и естественной сортировки методом слияния, а также анализ их эффективности.

Для достижения поставленной цели необходимо решить следующие задачи:

— Изучить теоретическую основу для применения алгоритмов внешней сортировки слиянием.

— Разработать алгоритм внешней сортировки прямым и естественным слиянием.

— Реализовать алгоритмы на практике.

— Произвести анализ эффективности алгоритмов и сделать выводы.

# 1. Теоретическая основа

## 1.1 Принцип сортировки слиянием

Сортировка слиянием базируется на слиянии нескольких отсортированных последовательностей в одну. Это достигается путём копирования минимального элемента из входных последовательностей в выходную до тех пор, пока во входных последовательностях остаются элементы. Это позволяет получить отсортированную выходную последовательность, включающую в себя все элементы входных за линейное время.

Для того, чтобы осуществить сортировку, для начала необходимо определиться, каким образом получить из неупорядоченной последовательности упорядоченные серии элементов. В каждом из рассматриваемых далее методов это достигается по-разному.

После разбиения изначальной неупорядоченной последовательности на выходе должны получиться последовательности содержащие упорядоченные серии, но еще не полностью отсортированные. Затем из каждой из получившихся последовательностей необходимо взять по одной такой серии и методом слияния получить более длинную упорядоченную серию. Все серии, получившиеся в итоге слияния образуют новую последовательность. Данную новую последовательность необходимо подвергать тем же манипуляциям до тех пор, пока все элементы изначальной последовательности не будут находиться в одной единственной упорядоченной серии.

## 1.2 Основные понятия

Внешняя сортировка – это сортировка данных, которые расположены на внешних устройствах и не вмещающихся в оперативную память.

Сортировка слиянием обычно осуществляется в 2 фазы – распределение(разбиение) и слияние.

Распределение - это процесс разделения упорядоченных серий на два и более вспомогательных файла.

Слияние – это процесс объединения двух (или более) упорядоченных серий в одну упорядоченную последовательность при помощи циклического выбора элементов доступных в данный момент.

Серия (упорядоченный отрезок) – это последовательность элементов, которая упорядочена по ключу.

Сортировка слиянием предполагает помимо сортируемого файла создание и использование двух и более вспомогательных. Сортировка осуществляемая на двух вспомогательных файлах называется двухпутевой. Многопутевая сортировка – это, в свою очередь, сортировка, выполняемая на N (N > 2) количестве вспомогательных файлов. В данном случае число N называется количеством путей слияния.

Несмотря на то, что в сортировка слиянием предполагает 2 фазы, их можно объединить в одну. Такая сортировка будет называться однофазной. В такой сортировке после слияния нескольких серий полученная упорядоченная подпоследовательность сразу записывается в один из вспомогательных файлов, после чего происходит переключение выходного файла. Такой подход позволяет вдвое уменьшить количество операций копирования данных, однако увеличивает количество необходимых вспомогательных файлов до 2N.

## 1.3 Внешняя сортировка прямым слиянием

Внешняя сортировка предполагает размещение последовательностей в файлах, а не в оперативной памяти. Алгоритм сортировки прямым слиянием предполагает фиксированную длину упорядоченных серий на каждом шаге. В исходном файле все серии имеют длину 1, после первого прохода она равна 2, после второго – 4, после третьего – 8, после k -го шага – 2k (Рисунок 1).

Сначала неотсортированный файл f0 разбивается на вспомогательные файлы f1 и f2 по принципу чередования серий в каждый из выходных файлов. Затем вспомогательные файлы сливаются в f0. При этом одиночные элементы образуют упорядоченные пары, пары образуют упорядоченные четверки и т.д. Эти шаги повторяются, каждый раз удваивая фиксированную длину серии.

После выполнения i проходов получаем два файла, состоящих из серий длины 2i. Окончание процесса происходит при выполнении условия 2i>=n, где n - длина сортируемой последовательности. Следовательно, процесс сортировки простым слиянием требует порядка O(log2 n) проходов по данным.

Признаками конца сортировки простым слиянием являются следующие условия:

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

— количество серий равно 1.

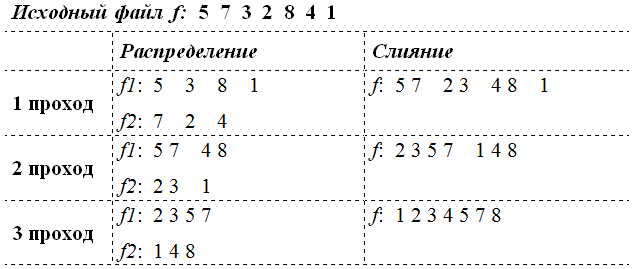


Рисунок 1 – Пример сортировки прямым слиянием

## 1.4 Внешняя сортировка естественным слиянием

В случае простого слияния частичная упорядоченность сортируемых данных не дает никакого преимущества. Это объясняется тем, что на каждом проходе сливаются серии фиксированной длины. При естественном слиянии длина серий не ограничивается, а определяется количеством элементов в уже упорядоченных подпоследовательностях, выделяемых на каждом проходе.

Алгоритм сортировки естественным слиянием начинается с того, что файл f разбивается на два вспомогательных файла f1 и f2. Распределение происходит следующим образом: поочередно считываются записи ai исходной последовательности (неупорядоченной) таким образом, что если значения ключей соседних записей удовлетворяют условию f(ai)<=f(ai+1), то они записываются в первый вспомогательный файл f1. Как только встречаются f(ai)>f(ai+1), то записи ai+1 копируются во второй вспомогательный файл f2. Процедура повторяется до тех пор, пока все записи исходной последовательности не будут распределены по файлам. Далее вспомогательные файлы f1 и f2 сливаются в файл f, при этом серии образуют упорядоченные последовательности. Эти шаги повторяются до тех пор, пока весь файл не будет отсортирован (Рисунок 2).

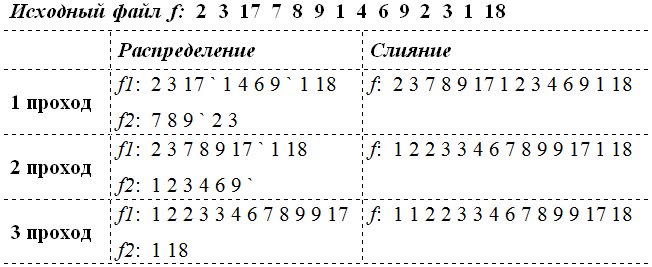


Рисунок 2 – Пример сортировки естественным слиянием

Важно отметить, что, в отличие от метода прямым слиянием, в этом методе сливаемые друг с другом серии могут быть разной длины, что требует от алгоритма дополнительных проверок при каждом считывании. Они нужны для проверки того, кончилась ли текущая серия. В случае, если одна из серий кончилась раньше другой, в соответствие с алгоритмом слияния, неоконченная последовательность дописывается без изменений.

Признаками конца сортировки естественным слиянием являются следующие условия:

* количество серий равно 1 (определяется на фазе слияния).
* при однофазной сортировке второй по счету вспомогательный файл после распределения серий остался пустым.

Таким образом, число чтений или перезаписей файлов при использовании метода естественного слияния будет не хуже, чем при применении метода простого слияния, а в среднем – даже лучше. Но в этом методе увеличивается число сравнений за счет тех, которые требуются для распознавания концов серий. Помимо этого, максимальный размер вспомогательных файлов может быть близок к размеру исходного файла, так как длина серий может быть произвольной.

# 2. Анализ поставленной задачи

Внешняя сортировка требует углубленной работы с памятью. Из этого следует, что алгоритм должен выполняться на достаточно низкоуровневом языке программирования.

Программа должна будет работать не с числами, как в примерах, а с записями, имеющими целочисленное ключевое поле. Записи имеют размер в 200 байт.

Для анализа эффективности реализуемых сортировок должны быть произведены эксперименты, фиксирующие количество записей в неотсортированном файле и время, затраченное на их упорядочивание. Результаты экспериментов будут выведены в виде графиков.

Под выдвинутые условия подходит язык программирования С++. На нем будут закодированы алгоритмы внешней сортировки прямым и естественным слиянием.

# 3. Проектирование используемых типов данных

Исходя из требований к записям в файлах будет практично создать структуру данных, которая будет имитировать такую запись с целочисленным ключом. Для этого создадим структуру под названием Record, содержащую переменную key типа данных int весом 4 байта. Остальные 196 байт записи сымитируем массивом типа данных int на 49 элементов.

Для замеров времени выполнения алгоритма используем пространство имен chrono языка С++. Создадим класс Timer, имеющий 2 переменные start и end. Эти переменные будут являться беззнаковыми точками во времени. Переменная start будет инициализироваться в конструкторе класса. В методе класса будет инициализироваться переменная end и возвращать разницу между двумя точками во времени в миллисекундах. Класс Timer будет создаваться в самом начале каждой из функций сортировок, а метод подсчета времени перед выходом из функций.

Данные измерений будут помещены в файл с расширением .csv, после чего по ним будет построен график.

# 4. Реализация алгоритмов

## 4.1 Сортировка прямым слиянием

Создадим описанные выше структуры данных, а именно структуру записи Record и класс для измерения времени Timer (Рисунок 3).

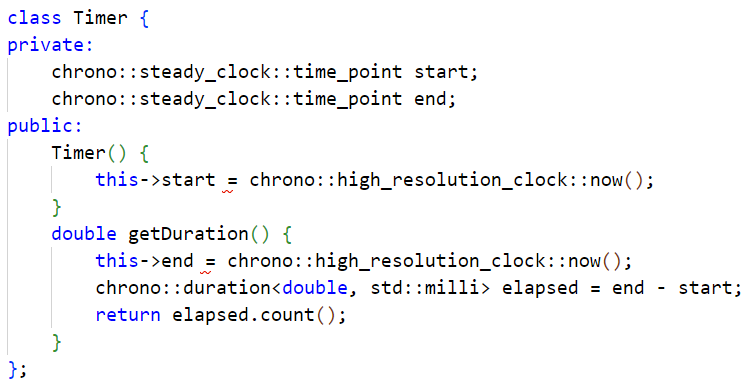
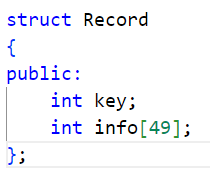


Рисунок 3 – Создание вспомогательных типов данных

Для реализации алгоритма понадобятся следующие переменные:

* + Дескрипторы файлов f0 (изначальная последовательность), f1, f2;
  + Переменные для считывания в них записей record1, record2;
  + Переменная для хранения длины файла len;
  + Переменные итераторы i, j;
  + Переменная для хранения актуальной длины серии series.

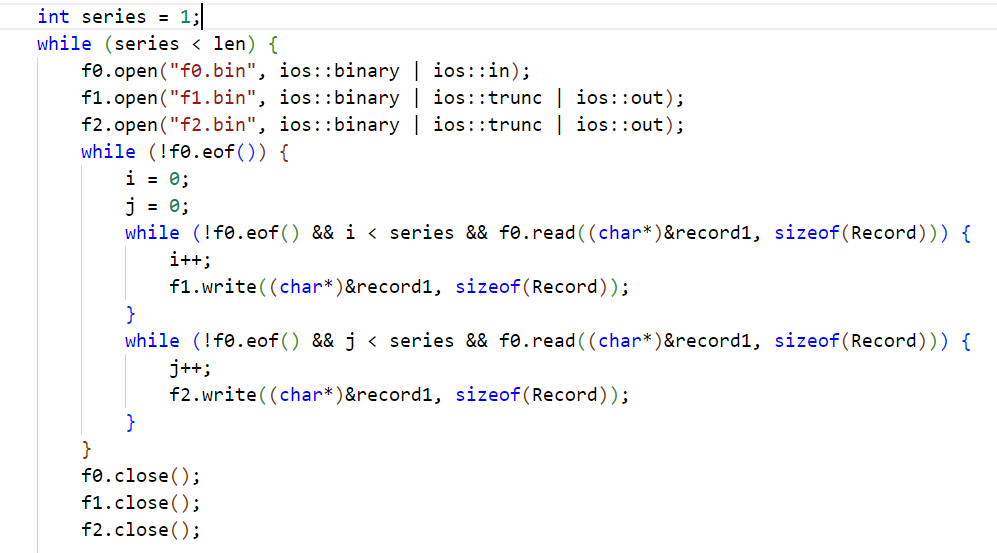


Рисунок 4 – Начало основного цикла программы

После инициализации переменных начинается основной цикл программы, в начале которого происходит распределение записей по файлам. В каждый файл поочередно попадает количество записей series. Изначально series инициализируется единицей (Рисунок 4). Файл изначальной последовательности f0 открывается для чтения, f1 и f2 – для записи.

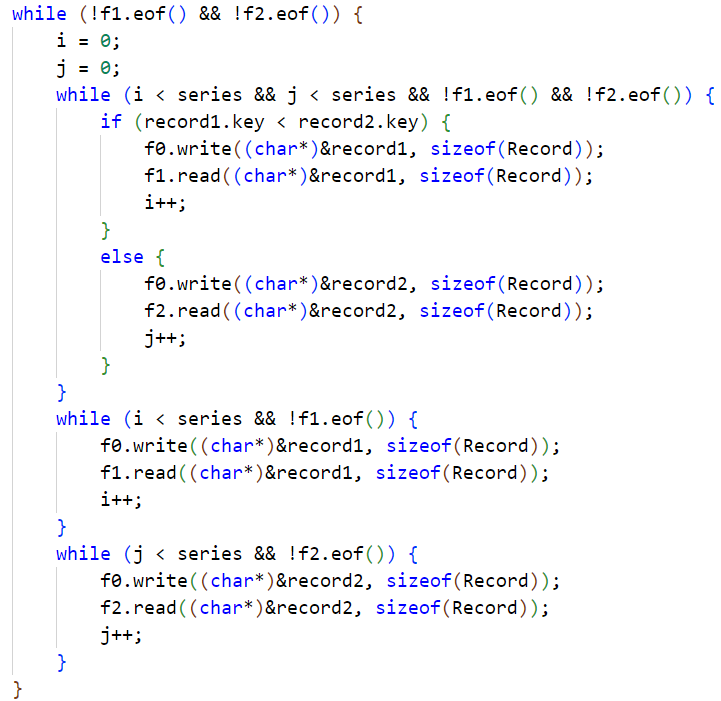


Рисунок 5 – Цикл слияния

После распределения наступает фаза слияния. Файлы f1 и f2 открываются для чтения, f0 для записи. Далее до окончания одного из файлов происходит слияние серий и запись в f0. Пока в обоих файлах серия не окончена – выполняется первый внутренний цикл, считывающий из соответствующего файла новую запись, записывающий запись в f0 и инкрементирующий переменную итератор. Если закончилась серия либо весь файл – остаток второй серии либо второго файла дописывается в f0 (Рисунок 5).

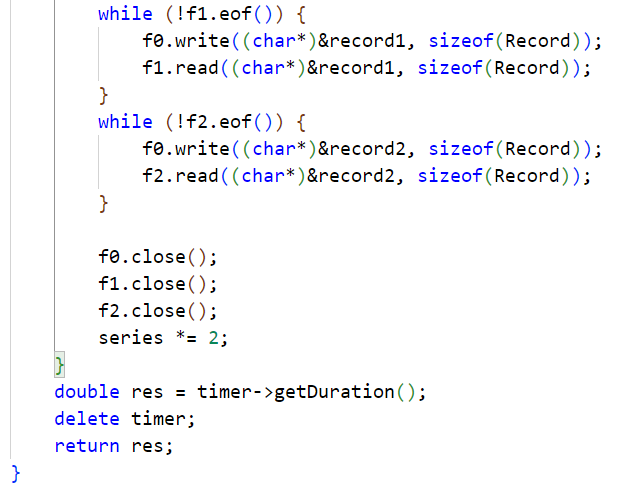


Рисунок 6 – Завершение функции слияния

Далее, в случае окончания одного из файлов он дописывается в f0. После этого используемые файлы закрываются и series увеличивается в 2 раза следуя алгоритму. После выполнения основного цикла вызывается функция класса Timer для фиксации затраченного времени и возвращается как результат работы функции (Рисунок 6).

## 4.2 Сортировка естественным слиянием

Сортировка естественным слиянием отличается тем, что в начале не создаются переменные итераторы, так как конец серий будет определяться проверками. Файлы открываются аналогично прямой сортировке. Далее в основном цикле программы создается переменная lastKey для хранения последнего считанного ключа записи, а также метка currFile, определяющая текущий выходной файл. Переменную lastKey инициализируем минимальным возможным числом. Пока f0 не закончится будем считывать записи, в случае если считанная запись меньше lastKey, значит серия закончена. В таком случае можно увеличить счетчик серий и сменить выходной файл. Далее в зависимости от currFile записываем считанную запись в f1 или f2 и меняем lastKey (Рисунок 7).

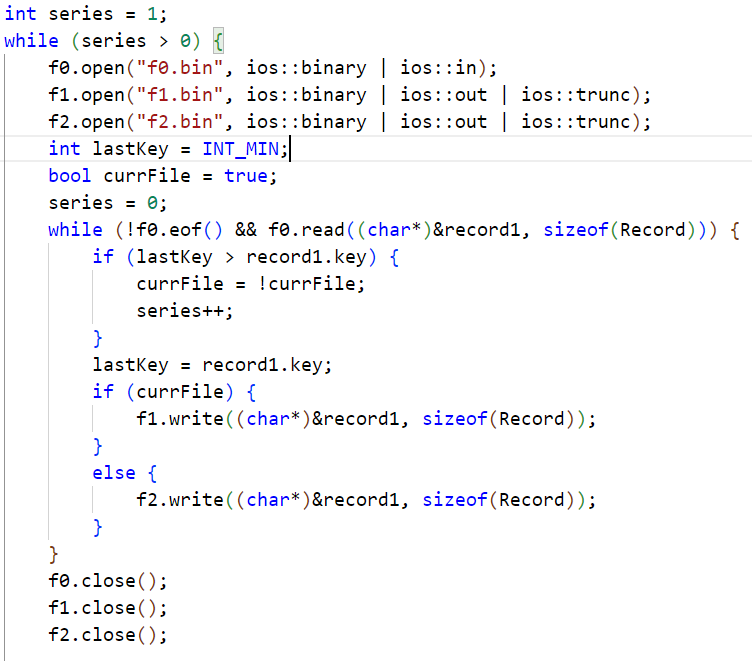


Рисунок 7 – Естественное распределение

После распределения файлы открываются в режимах аналогичных прямому слиянию. Для контроля окончания серий вводятся 2 переменных метки s1 и s2. Будем считывать записи из файлов f1 и f2 пока один из них не кончится. В случае, если запись из f1 меньше чем из f2, записываем в f0 эту запись, считываем новую из f1, сравниваем с только что записанной, меняем s1 в зависимости от того, кончилась ли серия на этом считывании. Для f2 логика симметрична.

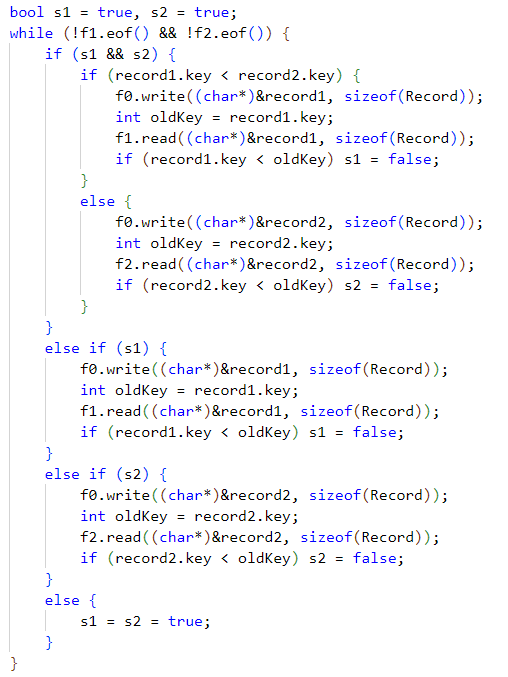


Рисунок 8 – Слияние в естественной сортировке

В случае окончания одной из серий дописываем остаток неоконченной серии в f0. В случае окончания обоих серий сбрасываем метки s1 и s2 (Рисунок 8).

Далее файлы аналогично методу прямого слияния дописываются в f0 в случае окончания одного из них. Таким же образом происходит обращение к методу класса Timer и возвращение итогового времени, затраченного на сортировку (Рисунок 9).

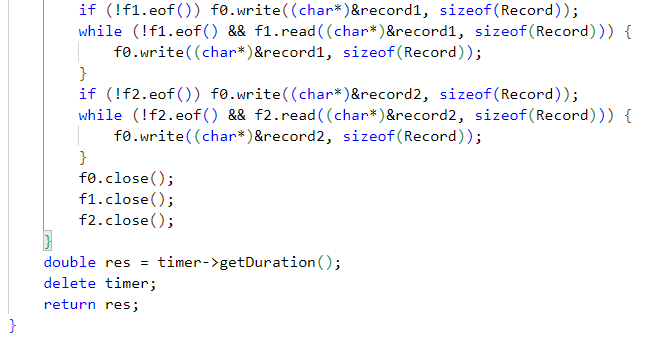


Рисунок 9 – Окончание сортировки естественным слиянием

# 5. Анализ эффективности алгоритмов

Для фиксации количества проходов по данным введем переменную mergeCount, которую будем инкрементировать вначале основного цикла каждой сортировки. В основной программе создадим цикл, создающий файл на 10000 записей со случайными ключами и отсортируем каждой из функций, фиксируя необходимые параметры.

Рисунок 10 – Сравнение времени выполнения на случайных ключах

Проведем 3 эксперимента по 100 итераций.

В первом случае будем сравнивать алгоритмы на совершенно случайных ключах (Рисунок 10). По результатам измерений обе сортировки оказались равны, некоторые отклонения спишем на погрешность измерений. Среднее время выполнения обоих алгоритмов составило приблизительно 150 миллисекунд. Количество проходов по данным в обоих сортировках в каждом из 100 случаев было равно 14, что подтверждает расчеты сложности, так как 213<10000<214.

Во втором случае сравним сортировки на частично упорядоченном файле (Рисунок 11). В данном случае вторая половина файла была упорядочена. Сортировка прямым слиянием не изменила показателей, а сортировка естественным слиянием стала выполняться в среднем на 10мс быстрее. В каждом из 100 случаев количество проходов по данным в естественном слиянии составило 13.

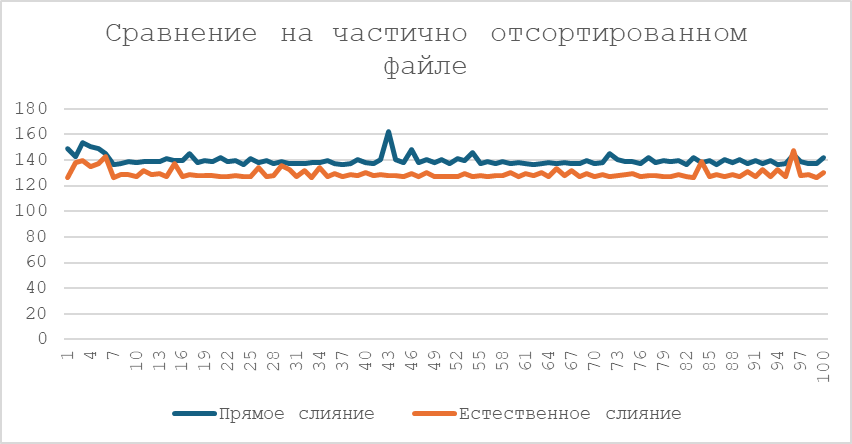


Рисунок 11 - Сравнение времени выполнения на частично упорядоченных ключах

Наконец, сравним алгоритмы на отсортированном файле (Рисунок 12). В результате измерений сортировка естественным слиянием показала среднее время выполнения в 10мс. В каждом случае был совершен 1 проход по данным.

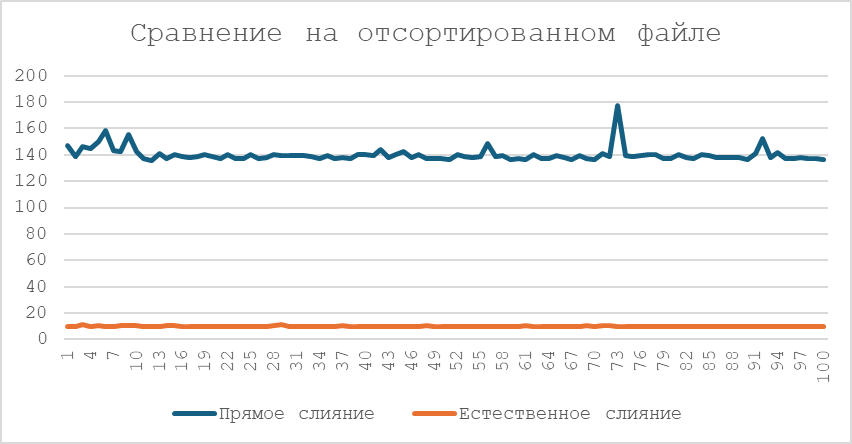


Рисунок 12 - Сравнение времени выполнения на упорядоченных ключах

# Заключение

В ходе выполнения курсовой работы были рассмотрены и реализованы алгоритмы внешней сортировки методами прямого и естественного слияний.

В процессе работы были проведены эксперименты, по результатам которых можно сказать, что сортировка естественным слиянием не хуже, а в среднем лучше сортировки прямым слиянием.

Алгоритмы внешней сортировки являются неотъемлемой частью работы с большими объемами данных, из чего следует, что их анализ и сравнение эффективности также важны. Для получения лучших результатов следует использовать многопутевые однофазные алгоритмы, не рассмотренные в рамках данной курсовой работы.

# Список использованной литературы

1. Алгоритм внешней сортировки слиянием | habr.com [Электронный ресурс]. Режим доступа: <https://habr.com/ru/companies/otus/articles/712234/> (Дата обращения: 02.12.2024)

2. Внешняя сортировка | wikipedia.org [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Внешняя_сортировка> (Дата обращения: 01.12.2024)

3. Сортировка естественным слиянием | intuit.ru [Электронный ресурс]. Режим доступа: https://intuit.ru/studies/courses/648/504/lecture/11473?page=2 (Дата обращения: 02.12.2024)

# Приложение 1

**Листинг основной программы  
(обязательное)**

#include <iostream>

#include <fstream>

#include <random>

#include <ctime>

#include <chrono>

#include <string>

#include "Record.h"

using namespace std;

class Timer {

private:

chrono::steady\_clock::time\_point start;

chrono::steady\_clock::time\_point end;

public:

Timer() {

this->start = chrono::high\_resolution\_clock::now();

}

double getDuration() {

this->end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::milli> elapsed = end - start;

//cout << "Time: " << elapsed.count() << " ms\n";

return elapsed.count();

}

};

void createFin(int amount) {

ofstream outfile("fin.bin", ios::binary | ios::trunc);

for (int i = 0; i < amount; i++)

{

Record record;

record.key = rand();

outfile.write((char\*)&record, sizeof(Record));

}

outfile.close();

}

void createF0() {

ifstream fin("fin.bin", ios::binary);

ofstream fout("f0.bin", ios::binary | ios::trunc);

Record record;

while(!fin.eof() && fin.read((char\*)&record, sizeof(Record)))

{

fout.write((char\*)&record, sizeof(Record));

}

fout.close();

fin.close();

}

int printFile(const char\* filename) {

int cntr = 0;

cout << "FILE: " << filename << endl;

ifstream infile(filename, ios::binary);

Record record;

while (infile.read((char\*)&record, sizeof(Record)) && !infile.eof()) {

cout << record.key << endl;

cntr++;

}

infile.close();

cout << "--------------------------------" << endl;

return cntr;

}

double directMerge() {

Timer \*timer = new Timer();

fstream f0("f0.bin", ios::binary | ios::in);

fstream f1, f2;

Record record1, record2;

f0.seekg(0, ios::end);

auto len = f0.tellg() / sizeof(Record);

f0.close();

int i = 0, j = 0;

int series = 1;

while (series < len) {

f0.open("f0.bin", ios::binary | ios::in);

f1.open("f1.bin", ios::binary | ios::trunc | ios::out);

f2.open("f2.bin", ios::binary | ios::trunc | ios::out);

while (!f0.eof()) {

i = 0;

j = 0;

while (!f0.eof() && i < series && f0.read((char\*)&record1, sizeof(Record))) {

i++;

f1.write((char\*)&record1, sizeof(Record));

}

while (!f0.eof() && j < series && f0.read((char\*)&record1, sizeof(Record))) {

j++;

f2.write((char\*)&record1, sizeof(Record));

}

}

f0.close();

f1.close();

f2.close();

f0.open("f0.bin", ios::binary | ios::trunc | ios::out);

f1.open("f1.bin", ios::binary | ios::in);

f2.open("f2.bin", ios::binary | ios::in);

f1.read((char\*)&record1, sizeof(Record));

f2.read((char\*)&record2, sizeof(Record));

while (!f1.eof() && !f2.eof()) {

i = 0;

j = 0;

while (i < series && j < series && !f1.eof() && !f2.eof()) {

if (record1.key < record2.key) {

f0.write((char\*)&record1, sizeof(Record));

f1.read((char\*)&record1, sizeof(Record));

i++;

}

else {

f0.write((char\*)&record2, sizeof(Record));

f2.read((char\*)&record2, sizeof(Record));

j++;

}

}

while (i < series && !f1.eof()) {

f0.write((char\*)&record1, sizeof(Record));

f1.read((char\*)&record1, sizeof(Record));

i++;

}

while (j < series && !f2.eof()) {

f0.write((char\*)&record2, sizeof(Record));

f2.read((char\*)&record2, sizeof(Record));

j++;

}

}

while (!f1.eof()) {

f0.write((char\*)&record1, sizeof(Record));

f1.read((char\*)&record1, sizeof(Record));

}

while (!f2.eof()) {

f0.write((char\*)&record2, sizeof(Record));

f2.read((char\*)&record2, sizeof(Record));

}

f0.close();

f1.close();

f2.close();

series \*= 2;

}

double res = timer->getDuration();

delete timer;

return res;

}

double naturalMerge() {

Timer \*timer = new Timer();

fstream f0, f1, f2;

Record record1, record2;

int s = 1;

while (s > 0) {

f0.open("f0.bin", ios::binary | ios::in);

f1.open("f1.bin", ios::binary | ios::out | ios::trunc);

f2.open("f2.bin", ios::binary | ios::out | ios::trunc);

int lastKey = INT\_MIN;

bool currFile = true;

s = 0;

while (!f0.eof() && f0.read((char\*)&record1, sizeof(Record))) {

if (lastKey > record1.key) {

currFile = !currFile;

s++;

}

lastKey = record1.key;

if (currFile) {

f1.write((char\*)&record1, sizeof(Record));

}

else {

f2.write((char\*)&record1, sizeof(Record));

}

}

f0.close();

f1.close();

f2.close();

f0.open("f0.bin", ios::binary | ios::out | ios::trunc);

f1.open("f1.bin", ios::binary | ios::in);

f2.open("f2.bin", ios::binary | ios::in);

f1.read((char\*)&record1, sizeof(Record));

f2.read((char\*)&record2, sizeof(Record));

bool s1 = true, s2 = true;

while (!f1.eof() && !f2.eof()) {

if (s1 && s2) {

if (record1.key < record2.key) {

f0.write((char\*)&record1, sizeof(Record));

int oldKey = record1.key;

f1.read((char\*)&record1, sizeof(Record));

if (record1.key < oldKey) s1 = false;

}

else {

f0.write((char\*)&record2, sizeof(Record));

int oldKey = record2.key;

f2.read((char\*)&record2, sizeof(Record));

if (record2.key < oldKey) s2 = false;

}

}

else if (s1) {

f0.write((char\*)&record1, sizeof(Record));

int oldKey = record1.key;

f1.read((char\*)&record1, sizeof(Record));

if (record1.key < oldKey) s1 = false;

}

else if (s2) {

f0.write((char\*)&record2, sizeof(Record));

int oldKey = record2.key;

f2.read((char\*)&record2, sizeof(Record));

if (record2.key < oldKey) s2 = false;

}

else {

s1 = s2 = true;

}

}

if (!f1.eof()) f0.write((char\*)&record1, sizeof(Record));

while (!f1.eof() && f1.read((char\*)&record1, sizeof(Record))) {

f0.write((char\*)&record1, sizeof(Record));

}

if (!f2.eof()) f0.write((char\*)&record2, sizeof(Record));

while (!f2.eof() && f2.read((char\*)&record2, sizeof(Record))) {

f0.write((char\*)&record2, sizeof(Record));

}

f0.close();

f1.close();

f2.close();

}

double res = timer->getDuration();

delete timer;

return res;

}

int main()

{

srand(static\_cast<unsigned int>(std::time(0)));

locale::global(std::locale("ru\_RU.UTF-8"));

cout.imbue(std::locale());

ofstream fout("data.csv", ios::trunc);

fout << "Прямое слияние;Естественное слияние\n";

for (int i = 0; i < 100; i++)

{

createFin(10000);

createF0();

double directRes = directMerge();

createF0();

double naturalRes = naturalMerge();

fout << directRes << ";" << naturalRes << endl;

}

fout.close();

/\*total = printFile("f0.bin");

cout << "TOTAL: " << total << endl;\*/

}