Министерство образования и науки Российской Федерации

#### ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

##### Кафедра «Математическое обеспечение и применение ЭВМ»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| «Утверждаю» | | | | |
| Зав. кафедрой «МО и ПЭВМ» | | | | |
|  | | | Макарычев П.П. | |
| « » |  |  | | 2018 |

### Пояснительная записка

к курсовой работе по дисциплине

«Структуры и алгоритмы обработки данных»

на тему: «Сравнение и анализ алгоритмов внешней сортировки»

|  |  |
| --- | --- |
| Автор работы: | Копылова О.Ю. |
| Направление бакалавриата | 09.03.04 («Программная инженерия») |
| Обозначение курсовой работы | ПГУ 09.03.04 - 04КР161.17 ПЗ |
| Группа | 16ВП1 |
| Руководитель работы | Самуйлов С.В, к.т.н., доцент |
| Работа защищена «\_\_» \_\_\_\_ 2018 г. | Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Пенза 2018 г.

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.

ЗАДАНИЕ

на курсовое проектирование по курсу

«Структуры и алгоритмы обработки данных»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студенту | Копыловой О.Ю. | | Группа | 16ВП1 |
| Тема проекта | | Сравнение и анализ алгоритмов внешней сортировки | | |

Исходные данные (технические требования) на проектирование

|  |
| --- |
| Выполнить сравнение времени сортировки алгоритмов сортировки |
| простым слиянием, сортировки естественным слиянием, сортировки |
| методом поглощения и сортировки многофазным слиянием. Данные |
| отобразить в табличной и графической формах. |
| Для разработки приложений использовать по выбору одну из сред |
| визуального программирования Borland Delphi, Borland Builder C++, |
| Lazarus, MS Visual C++ и т.п. |
| Пользовательский интерфейс должен обеспечивать привычное и |
| удобное представление информации, простое и эффективное выполнение |
| основных функций приложения. |
| Программное обеспечение должно быть полностью отлажено и |
| протестировано, функционировать под управлением ОС Windows XP и |
| выше. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Объем работы по курсу

1. Расчетная часть

|  |
| --- |
| 1. Анализ требований к разработке программного обеспечения |
| 1. Проектирование программы |
| 1. Разработка программного обеспечения |
| 1. Тестирование программного обеспечения |
|  |
|  |
|  |

2. Графическая часть

|  |
| --- |
| 1. Диаграмма вариантов использования |
| 1. Диаграмма классов |
| 1. Диаграмма деятельности |
| 1. Диаграмма компонентов |
|  |

3. Экспериментальная часть

|  |
| --- |
| 1. Подготовка набора тестовых данных |
| 1. Отладка и тестирование программного обеспечения |
|  |
|  |

Срок выполнения проекта по разделам

|  |  |  |
| --- | --- | --- |
| 1. Анализ требований к разработке ПО | к | 15.03.18 |
| 1. Проектирование программы | к | 01.04.18 |
| 1. Разработка программного обеспечения | к | 01.05.18 |
| 1. Тестирование программного обеспечения | к | 15.05.18 |
| 1. Оформление пояснительной записки | к | 28.05.18 |
| 1. Защита курсовой работы | к | 24.05.18 |
|  | к |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Дата выдачи задания | « 16» |  | февраля 2018 |
| Дата защиты проекта | « » |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель | |  | |
| Задание получил | | | «16» февраля 2018 г. |
| Студент |  | | |

# **Оглавление**

[Реферат 5](#_Toc510639554)

[Введение 6](#_Toc510639555)

[1. Постановка задачи и анализ предметной области 8](#_Toc510639556)

[1.1. Основные понятия и определения 8](#_Toc510639557)

[1.2. Постановка задачи на разработку программы 10](#_Toc510639558)

[1.3. Анализ требований 10](#_Toc510639559)

[1.4. Технология разработки программного обеспечения 16](#_Toc510639560)

[2. Проектирование программы 17](#_Toc510639561)

[2.1. Модель интерфейса 17](#_Toc510639562)

[2.2. Проектирование структур данных 18](#_Toc510639563)

[2.3. Структура программного обеспечения 20](#_Toc510639564)

[3. Реализация программы 29](#_Toc510639565)

[3.1. Кодирование 29](#_Toc510639566)

[3.2. Диаграмма компонентов 29](#_Toc510639567)

[4. Тестирование программы 30](#_Toc510639568)

[4.1. Виды тестирования программных средств 31](#_Toc510639569)

[4.2. Функциональное тестирование программы 32](#_Toc510639570)

[5. Анализ результатов 40](#_Toc510639571)

[Заключение 41](#_Toc510639572)

[Список использованной литературы 42](#_Toc510639573)

[Приложение A. Листинг программы 43](#_Toc510639574)

# Реферат

Пояснительная записка содержит 59 листов, 20 рисунков, 19 таблиц,   
 10 использованных источников, 1 приложение.

СОРТИРОВКА, ВНЕШНЯЯ СОРТИРОВКА, ПРОСТОЕ СЛИЯНИЕ, ЕСТЕСТВЕННОЕ СЛИЯНИЕ, МЕТОД ПОГЛОЩЕНИЯ, МНОГОФАЗНОЕ СЛИЯНИЕ.

Объектом исследования являются наиболее известные алгоритмы внешней сортировки.

Целью курсовой работы является разработка приложения для сравнения и анализа алгоритмов внешней сортировки.

Разработка проводилась на языке программирования C# в среде программирования Visual Studio 2017.

Осуществлено функциональное тестирование разработанного программного обеспечения, которое показало корректность его работы.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ПГУ 09.03.04 - 04КР161.17 ПЗ | | | | | |
|  |  |  |  |  |
| Изм. | Листт | № докум. | Подп. | Дата |
| Разраб. | | Копылова О.Ю. |  |  | «Сравнение и анализ алгоритмов внешней сортировки»  Пояснительная записка. | Лит. | | | Лист | Листов |
| Пров. | | Самуйлов С.В. |  |  |  |  |  | 4 | 59 |
|  | |  |  |  | Группа 16ВП1 | | | | |
| Н. контр. | |  |  |  |
| Утв. | |  |  |  |

# **Введение**

В программировании довольно часто перед тем как начать работу с данными их необходимо предварительно отсортировать. Однако следует учитывать, что метод сортировки должен быть подобран самых эффективный и удачно реализованный, иначе попытка оптимизации может привести к заметному понижению эффективности алгоритма в целом.

Все задачи сортировки делятся на две группы: задачи внутренней и внешней сортировки. Внутренняя сортировка применяется если все данные можно разместить в оперативной памяти одновременно, а внешняя, когда этого сделать нельзя.

На практике часто приходится иметь дело с неупорядоченными данными большого объема находящимися на внешних носителях с последовательным доступом. Для работы с ними необходимо преобразовать записи в упорядоченную последовательность, затратив как можно меньше времени и памяти, то есть для выполнения этой задачи необходимо совершить как можно меньше сравнений, чтений из файла и записей обратно. Однако эти файлы имеют большой объем, поэтому нельзя переместить их полностью в оперативную память, произвести сортировку и вернуть обратно, поэтому чтение и запись в файл происходит по частям с использованием различных алгоритмов внешней сортировки.

Скорость выполнения внешних сортировок будет зависеть от размера буфера или буферов основной памяти, которая может быть использована. Наиболее часто внешняя сортировка используется в СУБД при выполнении запросов и ее производительность зависит от эффективности применяемых методов.

Каждая сортировка имеет свои ключевые особенности из которых складываются достоинства и недостатки данного алгоритма, поэтому возникает вопрос: какой из методов быстрее выполняет сортировку данных?

**Актуальность рассматриваемой темы:** Сортировка применяется во всех областях программирования. Универсального метода сортировки нет, но для решения каждой задачи можно выбрать более эффективный. Соответственно выбор оптимального алгоритма для упорядочения данных сложная задача. Поэтому важно серьезно подойти к выбору внешней сортировки для каждой задачи в отдельности.

**Цель разрабатываемой программы:** Сравнить время сортировки данных разными алгоритмами внешней сортировки.

**Перечень подзадач, которые необходимо решить для достижения поставленной цели:**

1. Изучить принципы работы алгоритмов внешней сортировки различными методами (простым слиянием, естественным слиянием, методом поглощения, многофазным слиянием);
2. Реализовать эти алгоритмы;
3. Измерить время выполнения каждого метода;
4. Построить по полученным данным таблицы и диаграммы;
5. Сравнить полученные результаты.

# **1. Постановка задачи и анализ предметной области**

### **Основные понятия и определения**

Внешняя сортировка — сортировка данных, расположенных на периферийных устройствах и не вмещающихся в оперативную память.

Данные хранящиеся на внешних носителях занимают достаточно много памяти, поэтому нельзя перенести все сразу в оперативную память компьютера, отсортировать, а затем вернуть обратно. Для этого были разработаны различные алгоритмы внешней сортировки, каждый из которых имеет свои преимущества и недостатки.

К наиболее известным алгоритмам относятся:

* Сортировка простым слиянием
* Сортировка естественным слиянием
* Сортировка методом поглощения
* Сортировка многофазным слиянием

Основное понятие, которое используется в алгоритмах внешней сортировки – серия. Она представляет собой последовательность упорядоченных элементов. Длина серии варьируется от N (нет упорядоченным элементов) до одного (все элементы упорядочены).

Методы внешней сортировки состоят из многократно повторяющихся фаз слияния и распределения. Слияние – это процесс объединения двух и более упорядоченных серий в одну. Распределение - процесс разделения упорядоченных серий на вспомогательные файлы.

Рассмотрим алгоритмы внешней сортировки более подробно.

**1.** Алгоритм сортировки простым слиянием является простейшим. Он основан на процедуре слияния серией. Их длина фиксируется на каждом шаге. Сначала в файле все серии имеют длину 1, затем после каждого шага она увеличивается в 2 раза. Сортировка заканчивается когда , где – длина серии, – количество элементов в файле.

Основные шаги:

Шаг 1. Исходный файл делится на два вспомогательных и . В файл F1 записывается первый элемент, а в второй и так далее.

Шаг 2. Вспомогательные файлы и сливаются обратно в файл , причем одиночные элементы образуют упорядоченные пары.

Шаг 3. Полученный файл вновь обрабатывается согласно п.1 и п.2. упорядоченные пары переходят в упорядоченные четверки.

Шаг 4. Длина серии с каждым шагом увеличивается в 2 раза до тех пор, пока весь файл не будет упорядочен.

После выполнения k проходов образуются два файла с сериями длиной . Процесс сортировки прекратиться когда , следовательно она требует около O проходов, а исходный и вспомогательные файлы прочитаны и записаны O раз [2].

**2.** Алгоритм естественного слияния в отличии от предыдущего учитывает частично упорядоченные последовательности. То есть длина серии не ограничивается и зависит от уже упорядоченных элементов при каждом проходе.

Основные шаги [2]:

Шаг 1. Исходный файл разбивается на два вспомогательных и . Из файла поочередно считываются записи. Пока выполняется условие   
, то элементы записываются в . Как только встречаются   
 записи копируются в . Процедура повторяется пока все записи не будут распределены.

Шаг 2. Вспомогательные файлы и сливаются в , образуя упорядоченные последовательности.

Шаг 3. Файл обрабатывают согласно п1. и п.2.

Шаг 4. Повторяя шаги, сливание происходит до тех пор пока весь файл не будет упорядочен.

При использовании этого метода количество чтений и записей будет таким же, как и в предыдущем методе, однако количество сравнений увеличится, так как придется проводить сравнения для проверки конца серии. Кроме этого размер вспомогательных файлов может быть близок к размеру исходного, поскольку длина серий произвольная [4].

**3.** Для сортировки методом поглощения так же, как и для сортировки с внешним слиянием мы используем внутреннюю память компьютера.

Алгоритм действий для данной сортировки следующий:

Шаг 1. Из конца исходного файла считываем первую часть данных, упорядочиваем и записываем обратно.

Шаг 2. Далее берем следующую часть записей сортируем и сливаем с предыдущей упорядоченной, затем записываем на месте поглощенной части отсортированную последовательность и так до конца файла.

4. Многофазная сортировка появилась из сбалансированного многофазного слияния. В этом алгоритме примерно половина вспомогательных файлов используется для разделения и столько же для их слияния. Поэтому появилась идея многофазной сортировки. Она состоит в том, что из m файлов для распределения серий используется m-1, а оставшийся для слияния. Как только один из вводных файлов становится пустым его начинают использовать для слияния. Этот процесс происходит до тех пор, пока в одном из файлов не останется одна серия.

Многофазное слияние даёт ожидаемый результат и на каждом этапе сливает максимальное количество серий если начальное распределение серий по вспомогательным файлам описывается соседними числами Фибоначчи.

**1.2. Постановка задачи на разработку программы**

1. Запрограммировать следующие алгоритмы внешней сортировки:
2. Сортировка простым слиянием
3. Сортировка естественным слиянием
4. Сортировка методом поглощения
5. Сортировка многофазным слиянием
6. Выполнить сравнение времени сортировки перечисленными выше методами сортировки. Результаты представить в табличной и графической формах.

### **1.3. Анализ требований**

#### **1.3.1. Требования к интерфейсу пользователя**

Пользовательский интерфейс – это комплекс средств для взаимодействия пользователя с технической системой. В понятие пользовательского интерфейса компьютерной системы входят следующие составляющие: графическая среда, набор управляющих элементов пользовательского интерфейса и их расположение на экране, технологии взаимодействия пользователя с системой.

Основные требования к пользовательскому интерфейсу [9]:

1. Понятность и логичность;
2. Быстрое обучение пользователя;
3. Соответствие задачам пользователя;
4. Обеспечение высокой скорости работы пользователя;
5. Обеспечение защиты от человеческих ошибок;
6. Минимальные затраты ресурсов пользователя при вводе и просмотре данных;
7. Субъективное удовлетворение пользователя;
8. Максимальное взаимодействие пользователя и программы.

Приложение «Курсовая работа. Внешняя сортировка данных» должно представлять пользователю удобный интерфейс, позволяющий:

1. Сортировать данные разными методами;
2. Отображать работу разных алгоритмов внешней сортировки;
3. Выбирать для сравнения разные методы внешней сортировки;
4. Задавать размер массива входных данных;
5. Сравнивать время работы каждого метода между собой;
6. Отображать результаты сравнения в виде таблицы;
7. Отображать результаты сравнения в виде графика.

#### **1.3.2. Требования к структуре данных**

В разрабатываемой программе используются следующие структуры данных:

1) Array;

2) BinaryWriter

3) BinaryReader

Массив(Array) - [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом. При этом доступ к отдельным элементам массива осуществляется с помощью [индексации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), то есть через ссылку на массив с указанием номера (индекса) нужного элемента [6].

Основные операции для работы с массивами:

1. Установление размерности;
2. Заполнение массива по индексу;
3. Очищение массива и т.д.

BinaryWriter и BinaryReader – это классы, предназначенные для работы с бинарными файлами. Они позволяют читать и записывать данные в двоичном формате.

Данные на внешних носителях хранятся в виде файлов, которые состоят их записей. Запись характеризуется длиной, которая выражается в байтах и состоит из данных, которые передаются между оперативной и внешней памятью за одну операцию чтения и записи данных.

Чтение – это процесс передачи данных из внешней памяти в оперативную, а запись – это процесс передачи из оперативной во внешнюю.

Порядок записи в двоичный файл:

1. Создание объекта BinaryWrite;
2. Последовательная запись данных в файл;
3. Закрытие потока записи файла.

Порядок считывания данных из двоичного файла:

1. Создание объекта BinaryReader;
2. Последовательное чтение записей из файла;
3. Закрытие потока чтения файла.

Основные методы класса BinaryWriter [7]:

1. Close(): закрывает поток и освобождает ресурсы;
2. Flush(): очищает буфер, дописывая из него оставшиеся данные в файл;
3. Seek(): устанавливает позицию в потоке;
4. Write(): записывает данные в поток.

Основные методы класса BinaryReader [8]:

1. Close(): закрывает поток и освобождает ресурсы;
2. ReadInt32(), ReadChar(), ReadBoolean() и т.д.: соответствующий метод считывает данные определенного типа и перемещает указатель на размер этого типа в байтах;
3. PeekChar(): возвращает следующий доступный для чтения символ, не перемещая позицию байта или символа вперед.

#### **1.3.3. Требования к программным средствам**

Разрабатываемая программа должна производить сортировку данных выбранными методами и замерять время их работы. Анализ задания на разработку позволяет выделить следующие варианты использования (Рисунок 1).



Рисунок 1 - Диаграмма вариантов использования

Опишем спецификацию нескольких прецедентов. Сначала рассмотрим спецификацию прецедента «Провести сортировку файла методом однофазного простого слияния» (Таблица 1):

|  |
| --- |
| Прецедент: Провести сортировку файла методом однофазного простого слияния |
| ID: 1 |
| Краткое описание: Подсчитать время работы выполнения метода однофазного простого слияния, представить результаты в виде таблицы и графика. |
| Действующие лица: пользователь |
| Предусловия: Пользователь вошел в систему. |
| Основной поток:   1. Пользователь вводит количество элементов в файле 2. Пользователь выбирает метод однофазного простого слияния 3. Система формирует файл с заданным количеством элементов 4. Пока шаг не больше общего количества элементов    1. Система запускает подсчет времени    2. Система заполняет файл    3. Пока не конец файла       1. Система делит исходный файл на два вспомогательных, поочередно по одному элементу    4. Пока длина серии не равна длине всего файла       1. Система производит посерийное слияние двух файлов       2. Слитые серии поочередно распределяет по двум файлам    5. Система останавливает подсчет времени    6. Система заносит время в массив для дальнейшего вывода на графике 5. Система выводит результаты в таблицу 6. Система рисует график по полученным данным |
| Постусловия: Сформирована таблица и нарисован график |

Таблица 1 – Спецификация прецедента «Провести сортировку файла методом однофазного простого слияния»

Теперь опишем спецификацию прецедента «Провести сортировку файла методом однофазного естественного слияния» (Таблица 2):

|  |
| --- |
| Прецедент: Провести сортировку файла методом однофазного естественного слияния |
| ID: 2 |
| Краткое описание: Подсчитать время работы выполнения метода однофазного естественного слияния, представить результаты в виде таблицы и графика. |
| Действующие лица: пользователь |
| Предусловия: Пользователь вошел в систему. |
| 1. Пользователь вводит количество элементов в файле 2. Пользователь выбирает метод однофазного естественного слияния 3. Система формирует файл с заданным количеством элементов 4. Пока шаг не больше общего количества элементов    1. Система запускает подсчет времени    2. Система заполняет файл    3. Пока не конец файла       1. Система делит исходный файл на два вспомогательных, поочередно распределяя упорядоченные серии, которые уже есть в исходном файле    4. Пока длина серии не равна длине всего файла       1. Система производит посерийное слияние двух файлов       2. Слитые серии поочередно распределяет по двум файлам    5. Система останавливает подсчет времени    6. Система заносит время в массив для дальнейшего вывода на графике 5. Система выводит результаты в таблицу    1. Система рисует график по полученным данным |
| Постусловия: Сформирована таблица и нарисован график |

Таблица 2 - Спецификация прецедента «Провести сортировку файла методом однофазного естественного слияния»

Представим спецификацию прецедента «Провести сортировку файла методом поглощения» (Таблица 3):

|  |
| --- |
| Прецедент: Провести сортировку файла методом поглощения |
| ID: 3 |
| Краткое описание: Подсчитать время работы выполнения метода поглощения, представить результаты в виде таблицы и графика. |
| Действующие лица: пользователь |
| Предусловия: Пользователь вошел в систему. |
| Основной поток:   1. Пользователь вводит количество элементов в файле 2. Пользователь вводит размер оперативной памяти 3. Пользователь выбирает метод поглощения 4. Пока шаг не равен общему количеству элементов   6.1. Система запускает подсчет времени  6.2. Система заполняет файл  6.3. Система начинает чтение файла с конца  6.4. Система записывает первую серию в оперативную память  6.4.1. Система производит внутреннюю сортировку  6.4.2. Система возвращает серию в исходный файл  6.5. Пока не прочитан весь файл  6.5.1. Система увеличивает смещение от конца файла  6.5.2. Система считывает последовательность от начала установленного смещения до начала уже упорядоченной последовательности в оперативную память  6.5.2.1. Система производит внутреннюю сортировку последовательности  6.5.2.2. Система сливает полученную серию с предыдущей отсортированной  6.6. Система останавливает подсчет времени  6.7. Система заносит время в массив для дальнейшего вывода на графике  5. Система выводит результаты в таблицу  6. Система рисует график по полученным данным |
| Постусловия: Сформирована таблица и нарисован график |

Таблица 3 - Спецификация прецедента «Провести сортировку файла методом поглощения»

### **1.4. Технология разработки программного обеспечения**

Технологии разработки программного обеспечения – это комплекс мер, операций и приемов, направленных на разработку программного продукта. В процессе выполнения курсовой работы были использованы следующие технологии:

1. RAD
2. Каскадная модель жизненного цикла

RAD (Rapid Application Development) – это концепция создания средств разработки программных продуктов, уделяющая особое внимание быстроте и удобству программирования, созданию технологического процесса, позволяющего программисту максимально быстро создавать компьютерные программы. Концепцию RAD также часто связывают с концепцией визуального программирования.

Принципы данной технологии направлены на обеспечение трех основных преимуществ – высокая скорость разработки, низкая стоимость и высокое качество. Для достижения этих целей инструментарий должен быть нацелен на минимизацию времени разработки, цикличность разработки.

Каскадная модель (Waterfall model) – классический жизненный цикл, предполагающий последовательное выполнение этапов создания программного обеспечения. Основные этапы разработки:

1. Анализ;
2. Проектирование;
3. Разработка;
4. Тестирование;
5. Сопровождение.

Данная методология характеризуется тем, что переход на следующую стадию осуществляется только после завершения текущей, а возвращение на предыдущие стадии не предусматривается.

# **2. Проектирование программы**

### **2.1. Модель интерфейса**

Работа приложения начинается с запуска главной формы. Модель ее интерфейса представлена на рисунке 2.

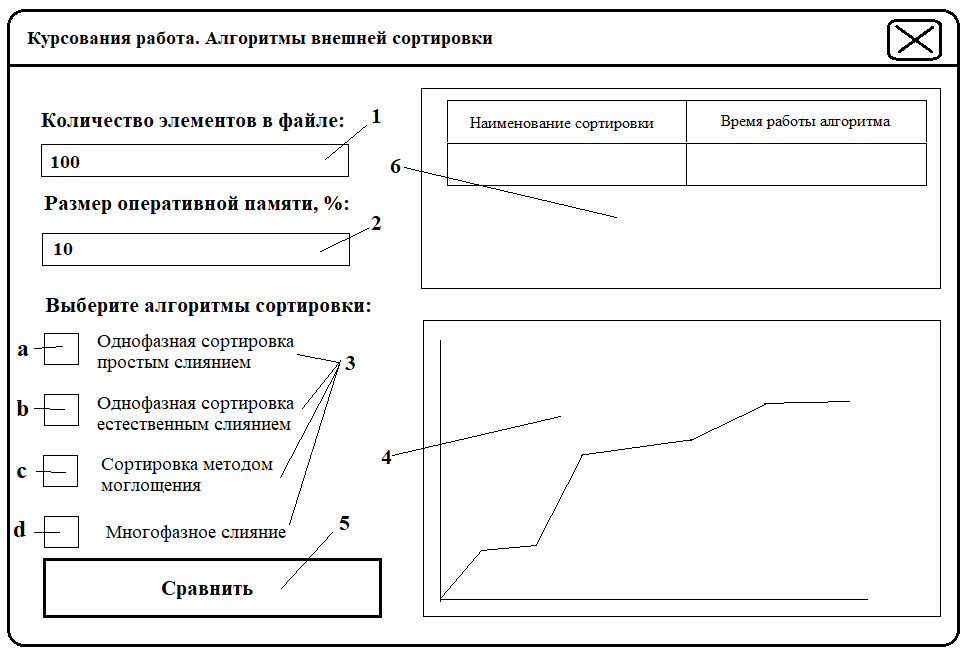


Рисунок 2 - Модель интерфейса программы

На ней:

1. Поле задания количества элементов в файле;
2. Поле задания объема оперативной памяти;
3. Выбор сравниваемых методов внешней сортировки;
4. График с результатами времени работы выбранных алгоритмов
   1. Метод однофазной сортировки простым слиянием
   2. Метод однофазной сортировки естественным слиянием
   3. Метод поглощения
   4. Метод многофазного слияния
5. Кнопка сравнить, при нажатии которой проводится сортировка выбранными методами. Результат отображается в таблице и на графике;
6. Таблица с результатами;

### **2.2. Проектирование структур данных**

Рассмотрим синтаксис и основные методы структур данных, использованных для создания приложения.

Структура данных Array (массив).

Синтаксис [10]:

public abstract class Array :

[SerializableAttribute]

[ComVisibleAttribute(true)]

public abstract class Array : ICloneable, IList, ICollection,

IEnumerable, IStructuralComparable, IStructuralEquatable

Основные методы представлены в таблице 4 [6]:

|  |  |
| --- | --- |
| Метод | Описание |
| Clear(Array, Int32, Int32) | Присваивает элементам массива значение 0, false или null, в зависимости от типа элементов. |
| Copy(Array, Int64, Array, Int64, Int64) | Копирует диапазон элементов из массива Array, начиная с заданного индекса источника, и вставляет его в другой массив Array, начиная с заданного индекса назначения. |
| Find<T>(T[], Predicate<T>) | Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает первое найденное вхождение в пределах всего списка Array. |
| GetValue(Int32) | Возвращает значение, хранящееся в указанной позиции одномерного массива Array. |

Таблица 4 - Основные методы класса Array

Структура класса BinaryWriter.

[SerializableAttribute]

[ComVisibleAttribute(true)]

public class BinaryWriter : IDisposable

Основные методы представлены в таблице 5 [7]:

|  |  |
| --- | --- |
| Метод | Описание |
| Close() | Закрывает текущий BinaryWriter и основной поток. |
| Flush() | Очищает все буферы текущего модуля записи и вызывает немедленную запись всех буферизованных данных на базовое устройство. |
| Dispose() | Освобождает все ресурсы, используемые текущим экземпляром класса BinaryWriter. |
| PeekChar() | Возвращает следующий доступный для чтения символ, не перемещая позицию байта или символа вперед. |
| ReadInt32() | Считывает целое число со знаком длиной 4 байта из текущего потока и перемещает текущую позицию в потоке на четыре байта вперед. |
| Finalize() | Позволяет объекту попытаться освободить ресурсы и выполнить другие операции очистки, перед тем как он будет уничтожен во время сборки мусора |
| Seek(Int32, SeekOrigin) | Задает позицию в текущем потоке |
| Write(<тип данных>) | Записывает в текущий поток и перемещает позицию в потоке. |

Таблица 5 - Основные методы класса BinaryWriter

Структура класса BinaryReader

Синтаксис [10]:

[ComVisibleAttribute(true)]

public class BinaryReader : IDisposable

Основные методы представлены в таблице 6 [8]:

|  |  |
| --- | --- |
| Метод | Описание |
| Close() | Закрывает текущий поток чтения и связанный с ним базовый поток. |
| Dispose() | Освобождает все ресурсы, используемые текущим экземпляром класса BinaryReader. |
| PeekChar() | Возвращает следующий доступный для чтения символ, не перемещая позицию байта или символа вперед. |
| ReadInt32() | Считывает целое число со знаком длиной 4 байта из текущего потока и перемещает текущую позицию в потоке на четыре байта вперед. |
| Finalize() | Позволяет объекту попытаться освободить ресурсы и выполнить другие операции очистки, перед тем как он будет уничтожен во время сборки мусора |

Таблица 6 - Основные методы класса BinaryReader

### **2.3. Структура программного обеспечения**

Структура приложения представлены на диаграмме классов (Рисунок 3).

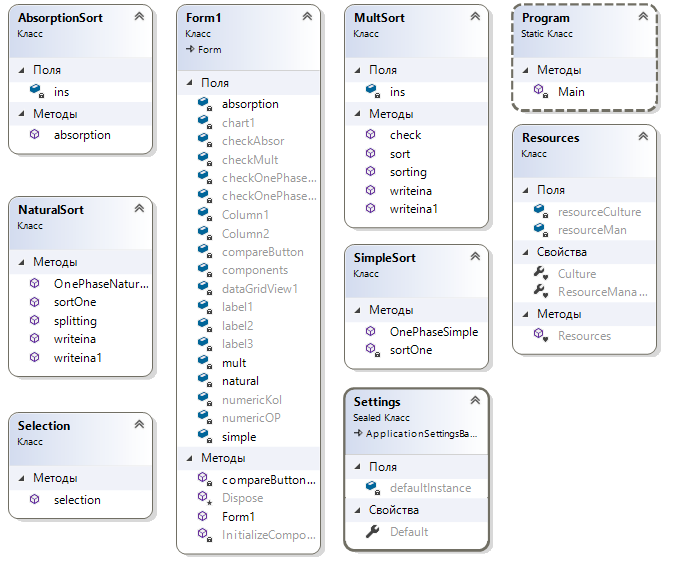


Рисунок 3 - Диаграмма классов

**Спецификация класса «Form1».**

**Назначение класса:** главная форма приложения. Методы и свойства класса изображены на рисунке 4.

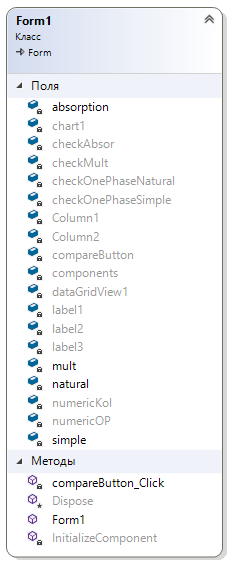


Рисунок 4 - Методы и свойства класса Form1

**Описание свойств класса представлено в таблице 7.**

|  |  |
| --- | --- |
| **Свойство класса** | **Назначение свойства** |
| chart1 | График |
| checkAbsor | Элементы дизайна (CheckBox), отвечающие за выбор метода внешней сортировки |
| checkMult |
| checkOnePhaseNatural |
| checkOnePhaseSimple |
| dataGridView1 | Таблица |
| column1 | Элементы дизайна (DataGridView), отвечающие за вывод результатов на экран |
| column2 |
| compareButton | Кнопка, запускающая сравнение |
| label1 | Надписи |
| label2 |
| label3 |
| numericKol | Поле для ввода количества элементов в файле |
| numericOP | Поле для ввода размера оперативной памяти |
| absorption | Экземпляр класса AbsorptionSort |
| mult | Экземпляр класса MultSort |
| natural | Экземпляр класса NaturalSort |
| simple | Экземпляр класса SimpleMerge |

Таблица 7 - Свойства класса Form1

**Описание методов класса представлено в таблице 8**.

|  |  |
| --- | --- |
| **Метод класса** | **Назначение метода** |
| Form1() | Конструктор |
| compareButton\_Click(sender : ModelingProject1::System::Object, e : ModelingProject1::System::EventArgs) | Обработчик события нажатие на кнопку compareButton. Запускает и останавливает подсчет времени, производит сортировку файла методом поглощения |
| InitializeComponent() | Инициализирует объект |
| Dispose(disposing : Boolean) | Выполняет определяемые приложением задачи, связанные с удалением, высвобождением или сбросом неуправляемых ресурсов |

Таблица 8 - Методы класса Form1

**Спецификация класса «SimpleMerge».**

**Назначение класса:** производит сортировку файла методом простого однофазного слияния. Методы класса изображены на рисунке 5.

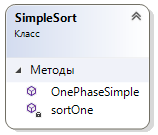


Рисунок 5 - Методы и свойства класса SimpleSort

**Описание методов класса представлено в таблице 9**.

|  |  |
| --- | --- |
| **Метод класса** | **Назначение метода** |
| OnePhaseSimple(a: String, kol: Integer, time: Ref Long) | Запускает и останавливает подсчет времени, производит начальное разделение исходного файла по двум вспомогательным, запускает сортировку файла |
| sortOne(noots: Ref Boolean, b: FileStream, c: FileStream, d: FileStream, e: FileStream) | Производит слияние двух файлов A и B, при этом сразу разделяет между файлами C и D. |

Таблица 9 - Методы класса SimpleSort

**Спецификация класса «NaturalSort».**

**Назначение класса:** производит сортировку файла методом естественного однофазного слияния. Методы и свойства класса изображены на рисунке 6.

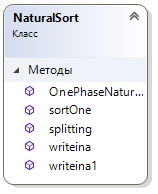


Рисунок 6 - Методы и свойства класса NaturalSort

**Описание методов класса представлено в таблице 10**.

|  |  |
| --- | --- |
| **Метод класса** | **Назначение метода** |
| OnePhaseNatural(a: String, kol: Integer, time: Ref Long) | Запускает и останавливает подсчет времени, запускает сортировку файла |
| sortOne(noots: Ref Boolean, b: FileStream, c: FileStream, d: FileStream, e: FileStream) | Производит слияние двух файлов A и B, при этом сразу разделяет между файлами C и D. |
| splitting(a: FileStream, b: FileStream, c: FileStream) | Производит начальное разделение упорядоченных последовательностей, которые уже существуют в исходном файле, по двум вспомогательным. |
| writeina1(znach: Ref Integer, nextznach: Ref Integer, zapislast: Ref Integer, zapis: Ref integer, ser1: Ref Bool, bw: BinaryWriter, brb: BinaryReader) | Запись первого элемента серии |
| writeina(znach: Ref Integer, nextznach: Ref Integer, zapislast: Ref Integer, zapis: Ref Integer, ser1: Ref Bool, bw: BinaryWriter, brb: BinaryReader) | Запись элементов серии, которые не являются первыми |

Таблица 10 - Методы класса NaturalSort

**Спецификация класса «AbsorptionSort».**

**Назначение класса:** производит сортировку файла методом поглощения. Методы и свойства класса изображены на рисунке 7.

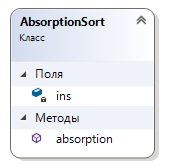


Рисунок 7 - Методы и свойства класса AbsorptionSort

**Описание свойств класса представлено в таблице 11.**

|  |  |
| --- | --- |
| **Свойство класса** | **Назначение свойства** |
| ins | Экземпляр класса Selection |

Таблица 11 - Свойства класса AbsorbtionSort

**Описание методов класса представлено в таблице 12**.

|  |  |
| --- | --- |
| **Метод класса** | **Назначение метода** |
| absorption(a: String, kol: Integer, Prop: Integer, time: Ref Long) | Запускает и останавливает подсчет времени, производит сортировку файла методом поглощения |

Таблица 12 - Методы класса AbsorbtionSort

**Спецификация класса «Selection».**

**Назначение класса:** производит сортировку серии. Методы и свойства класса изображены на рисунке 8.

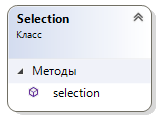


Рисунок 8 - Методы и свойства класса Selection

**Описание методов класса представлено в таблице 13**.

|  |  |
| --- | --- |
| **Метод класса** | **Назначение метода** |
| selection(a: Array Integer) | Внутренняя сортировка включением |

Таблица 13 - Методы класса Selection

**Спецификация класса «MultSort».**

**Назначение класса:** производит сортировку файла многофазным слиянием. Методы и свойства класса изображены на рисунке 9.

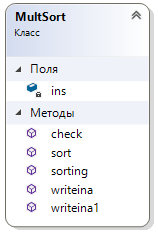


Рисунок 9 - Методы и свойства класса MultSort

**Описание свойств класса представлено в таблице 14.**

|  |  |
| --- | --- |
| **Свойство класса** | **Назначение свойства** |
| ins | Экземпляр класса Selection |

Таблица 14 - Свойства класса MultSort

**Описание методов класса представлено в таблице 15**.

|  |  |
| --- | --- |
| **Метод класса** | **Назначение метода** |
| check(noots: Ref Boolean, b: String, e: Integer) | Проверяет на пустоту файла |
| sort(w: String, kol: Integer, Prop: Integer, time: Ref Long) | Выполняет вычисление количества необходимых серий в двух вайлах, выполняет их распределение и запускает процесс слияния |
| sorting(a: Ref String, b: Ref String, c: Ref String, noots: Ref Boolean) | Сливает серии двух файлов в третий |
| writeina1(znach: Ref Integer, nextznach: Ref Integer, zapislast: Ref Integer, zapis: Ref integer, ser1: Ref Bool, bw: BinaryWriter, brb: BinaryReader) | Записывает первое значение серии |
| writeina(znach: Ref Integer, nextznach: Ref Integer, zapislast: Ref Integer, zapis: Ref Integer, ser1: Ref Bool, bw: BinaryWriter, brb: BinaryReader) | Записывает остальные значения серии |

Таблица 15 - Методы класса MultSort

**Спецификация класса «Program».**

**Назначение класса:** главный класс, точка входа в приложение. Методы класса изображены на рисунке 10.

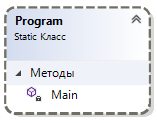


Рисунок 10 - Методы класса Program

**Описание методов класса представлено в таблице 16**.

|  |  |
| --- | --- |
| **Метод класса** | **Назначение метода** |
| Main() | Точка входа в приложение |

Таблица 16 - Методы класса Program

# **3. Реализация программы**

### **3.1. Кодирование**

Код программы приведен в приложении 1.

Приведем диаграмму деятельности метода «compareButton\_Click» (Рисунок 11). Из нее видно, что метод сначала очищает таблицу от предыдущих результатов, затем проверяет выбрана ли хотя бы один метод сортировки. Если пользователь не выбрал ни один пункт, то выдается сообщение об ошибке. Если хотя бы один метод выбран, то считывается общее количество элементов. Пока шаг не равен общему количеству элементов заполнять файл количеством элементов равным шагу. Передать созданный файл в выбранные методы и произвести соответствующую сортировку, записать замеренное время в массив. Когда шаг становится равным общему числу элементов выводится график времени и таблица с результатами.

### 

Рисунок 11 - Диаграмма вариантов использования для метода compareButton\_Click

### **3.2. Диаграмма компонентов**

Модули программы и взаимоотношения между ними представлены на диаграмме компонентов (Рисунок 12).

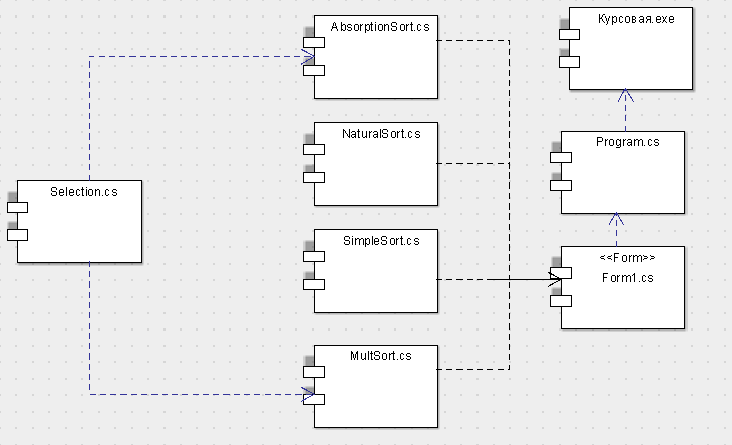


Рисунок 12 - Диаграмма компонентов

На диаграмме представлены следующие компоненты:

1. «Курсовая.exe» - исполнительный файл
2. «Program.cs» - модуль класса Program.
3. «Form1.cs» - модуль класса mainForm.
4. «SimpleMerge.cs» - модуль класса SimpleMerge.
5. «NatSort.cs» - модуль класса NatSort.
6. «InputSort.cs» - модуль класса InputSort.
7. «MultSort.cs» - модуль класса MultSort.
8. «Selection.cs» - модуль класса Selection.

# **4. Тестирование программы**

### **4.1. Виды тестирования программных средств**

Тестирование программного обеспечения — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом [5]. Цели тестирования:

1. Проверить правильность работы приложения при любых условиях;
2. Проверить соответствие приложения описанным требованиям;
3. Предоставление актуальной информации о состоянии продукта.

Существует множество видов тестирования. Рассмотрим основные:

1. Функциональное тестирование (functional testing) – проверка соответствия программного обеспечения требованиям, заявленным в спецификации
   1. Тестирование «белого ящика» - проверка на соответствие требованиям со знанием внутренней структуры системы.
   2. Тестирование «черного ящика» - проверка на соответствие требованиям без знания внутренней структуры системы.
2. Системное тестирование (system testing) – высокоуровневая проверка функционала всей системы в целом.
3. Тестирование производительности (performance testing) проводится с целью определения быстроты работы системы или её частей под определенной нагрузкой.
   1. Нагрузочное тестирование – проверка работоспособности при стандартных нагрузках.
   2. Стресс тестирование – проверка работоспособности при нестандартных нагрузках.
4. Регрессионное тестирование (regression testing) проводится для проверки поведения системы после добавления новых функций, для улучшения и исправления дефектов существующего функционала.
5. Модульное тестирование (unit testing) – проверка корректности работы каждого модуля системы в отдельности.
6. Тестирование безопасности (security testing) и анализ рисков, которые связаны с обеспечением целостного подхода к защите приложения несанкционированного доступа к данным.
7. Тестирование локализации (localization testing) – проверка правильности перевода элементов интерфейса пользователя, сопроводительной документации и т.д.
8. Юзабилити тестирование (usability testing) – метод, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности продукта для пользователя.

### **4.2. Функциональное тестирование программы**

Разработка тестов основана на методе разбиения на классы эквивалентности. Рассмотрим правильные классы эквивалентности (Таблица 17).

|  |  |  |
| --- | --- | --- |
| **Режим работы** | **Входные данные** | **Выходные данные** |
| Сравнение работы алгоритмов | 100 ≤ количество элементов ≤ 100000 | Таблица с результатами и график |
| 10 ≤ объем оперативной памяти ≤ 50 |

Таблица 17 – Правильные классы эквивалентности

Неправильные классы эквивалентности таблица

|  |  |  |
| --- | --- | --- |
| **Режим работы** | **Входные данные** | **Выходные данные** |
| Сравнение работы алгоритмов | количество элементов < 100 | Количеству значений приравнивается минимальное значение |
| количество  элементов > 100000 | Количеству значений приравнивается максимальное значение |
| объем оперативной  памяти < 10 | Объему оперативной памяти присваивается минимальное значение |
| объем оперативной  памяти > 50 | Объему оперативной памяти присваивается минимальное значение |
| Ни один метод сортировки не выбран | Сообщение об ошибке: «Ни один метод сортировки не выбран!» |

Таблица 18 –Неправильные классы эквивалентности

На основе разработанных правильных классов эквивалентности составим следующие тесты (Таблица 19):

|  |  |  |
| --- | --- | --- |
| **Состав теста** | **Ожидаемый результат** | **Наблюдаемый результат** |
| Выбрана однофазная сортировка простым слиянием. Количество элементов: 1000 | В таблице показано название и время работы алгоритма. Нарисована диаграмма | В таблице показано название и время работы алгоритма. Нарисована диаграмма (Рисунок 12) |
| Выбрана однофазная сортировка естественным слиянием. Количество элементов: 2456 | В таблице показано название и время работы алгоритма. Нарисована диаграмма | В таблице показано название и время работы алгоритма. Нарисована диаграмма (Рисунок 13) |
| Выбрана сортировка методом поглощения. Количество элементов: 1892. Объем оперативной памяти: 10 | В таблице показано название и время работы алгоритма. Нарисована диаграмма | В таблице показано название и время работы алгоритма. Нарисована диаграмма (Рисунок 14) |
| Выбрана сортировка многофазным слиянием. Количество элементов: 1578 Объем оперативной памяти: 10 | В таблице показано название и время работы алгоритма. Нарисована диаграмма | В таблице показано название и время работы алгоритма. Нарисована диаграмма (Рисунок 15) |
| Выбраны все алгоритмы сортировки. Количество элементов: 100. Объем оперативной памяти: 10 | В таблице показаны названия и время работы каждого алгоритма. Нарисована диаграмма | В таблице показаны названия и время работы каждого алгоритма. Нарисована диаграмма (Рисунок 16) |
| Выбрана сортировка методом поглощения. Количество элементов: 80. Объем оперативной памяти: 2 | Количеству элементов и объему оперативной памяти присваиваются минимальные значения, выведены таблица с результатами и диаграмма | Количеству элементов и объему оперативной памяти присваиваются минимальные значения, выведены таблица с результатами и диаграмма (Рисунок 17) |
| Выбрана сортировка методом поглощения и однофазная сортировка простым слиянием. Количество элементов: 10000000. Объем оперативной памяти: 100 | Количеству элементов и объему оперативной памяти присваиваются максимальные значения, выведены таблица с результатами и диаграмма | Количеству элементов и объему оперативной памяти присваиваются максимальные значения, выведены таблица с результатами и диаграмма (Рисунок 18) |
| Ни один алгоритм сортировки не выбран | Сообщение об ошибке: «Ни один метод сортировки не выбран!» | Сообщение об ошибке: «Ни один метод сортировки не выбран!» (Рисунок 19) |

Таблица 19 - Тесты

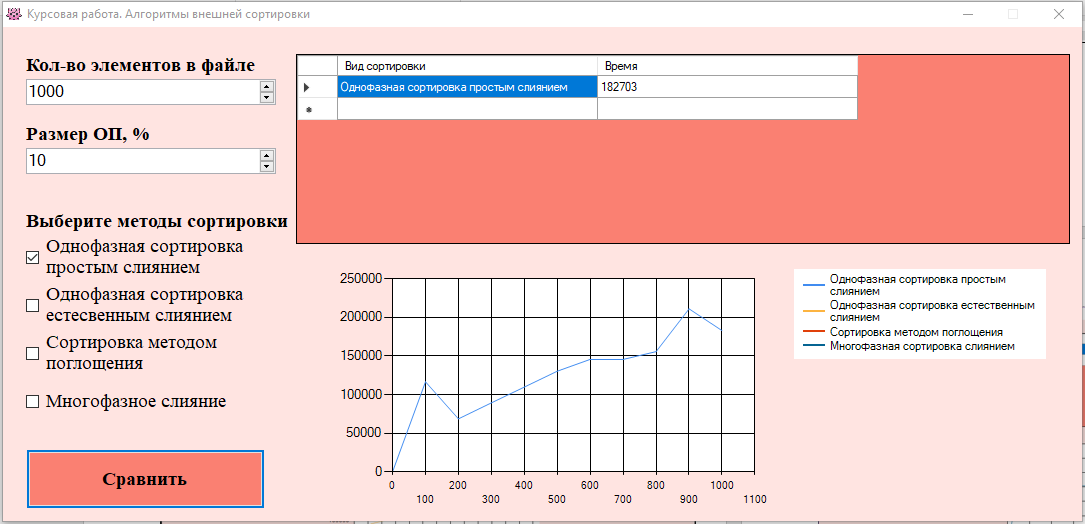


Рисунок 13 – Тест №1. Простое однофазное слияние, количество элементов - 1000

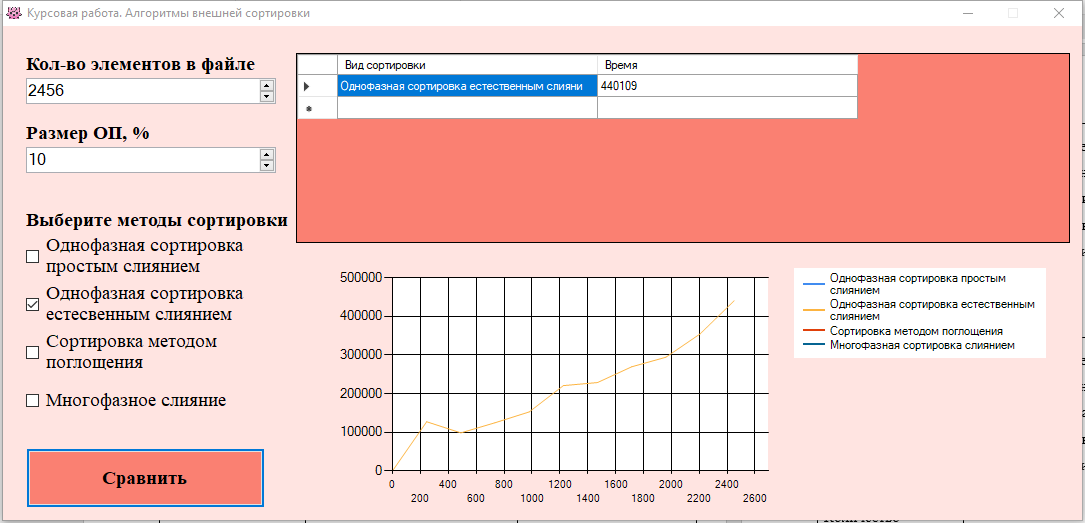


Рисунок 14 – Тест №2. Естественное однофазное слияние, количество элементов - 2456

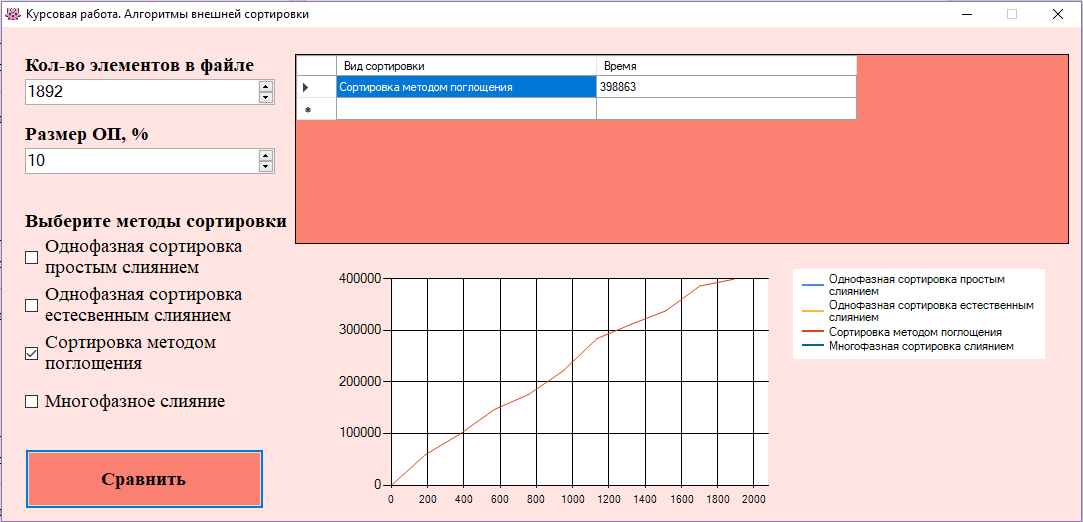


Рисунок 15 – Тест №3. Метод поглощения, количество элементов – 1892, объем оперативной памяти - 10%

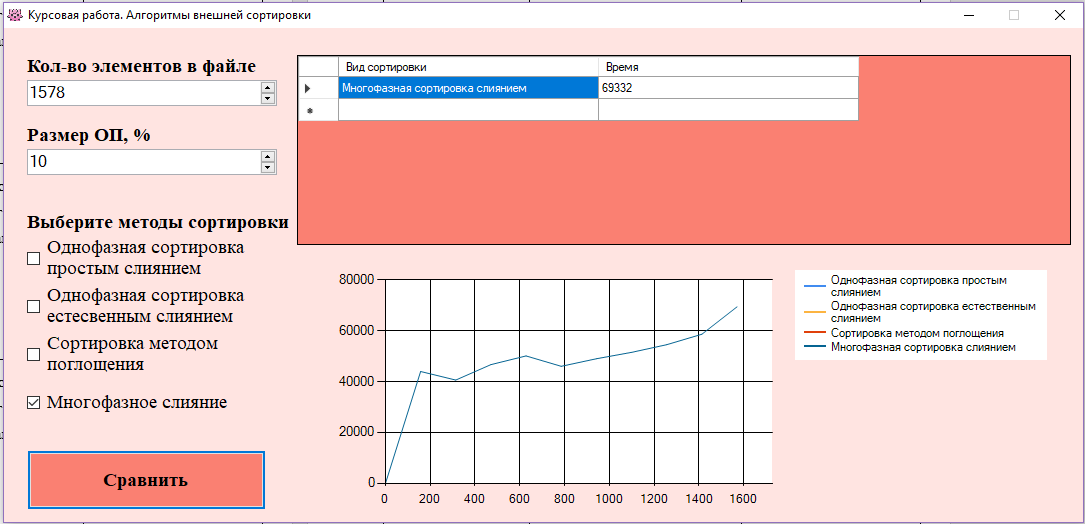


Рисунок 16 – Тест №4. Многофазное слияние, количество элементов – 1578, объем оперативной памяти - 10%

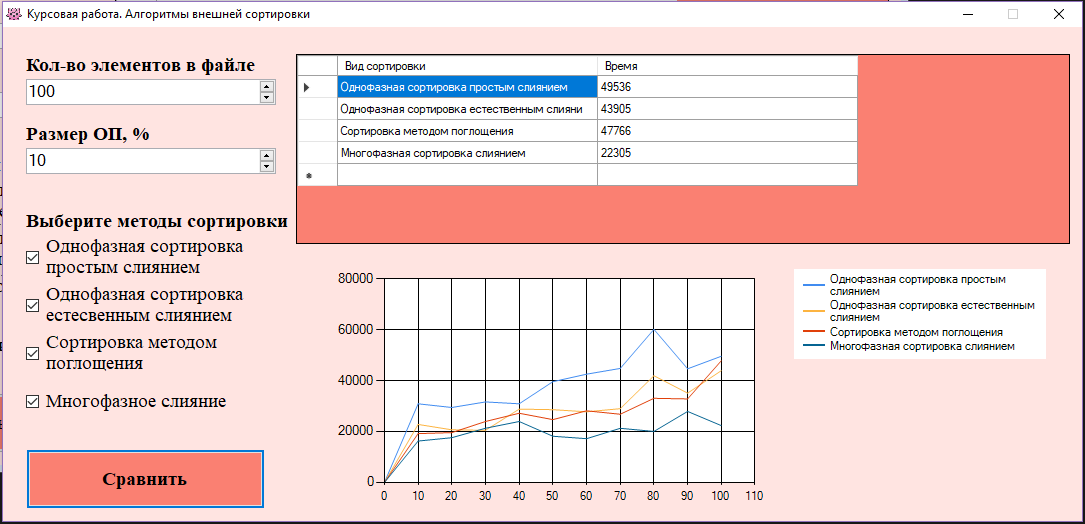


Рисунок 17 – Тест №5. Выбраны все методы сортировки, количество элементов – 100, объем оперативной памяти - 10%

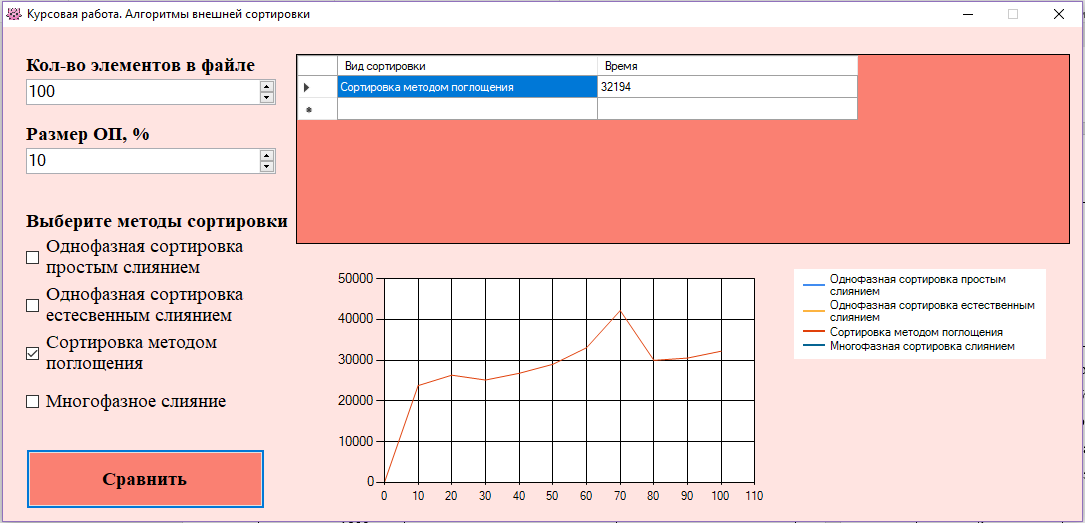


Рисунок 18 – Тест №6. Метод поглощения, количество элементов – 80, объем оперативной памяти - 2%

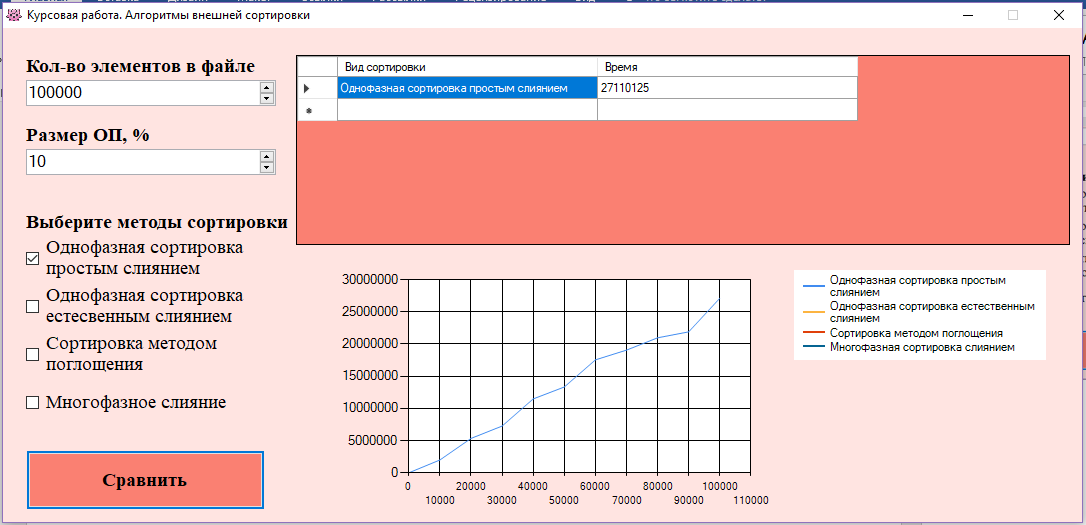


Рисунок 19 – Тест №7. Простое однофазное слияние, количество элементов - 10000000

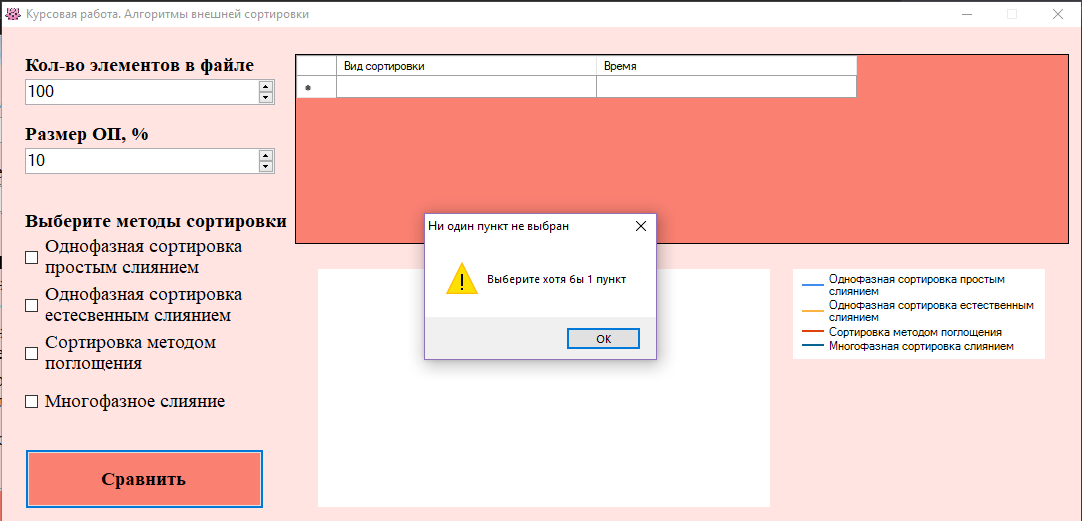


Рисунок 20 – Тест №8. Не выбран ни один метод сортировки

# **5. Анализ результатов**

При сравнении работы алгоритмов внешней сортировки получаются следующие результаты:

1. Худшее время работы показала однофазная сортировка простым слиянием.
2. Приблизительно равными по времени выполнения оказалась однофазная сортировка и метод поглощения, так же они быстрее чем сортировка простым слиянием.
3. Почти в два раза быстрее чем метод поглощения показала себя многофазная сортировка слиянием.

Сортировка естественным слиянием учитывает частичную упорядоченность исходного файла, поэтому она оказалась быстрее, чем сортировка простым слиянием. Так же было выявлено, что при увеличении объема оперативной памяти разница во времени выполнения сортировки методом поглощения и многофазным слиянием становится меньше.

Из результатов, полученных в ходе выполнения курсовой работы можно сделать вывод, что наиболее оптимальным методом внешней сортировки является многофазное слияние, так как во время её работы не происходит создания лишних вспомогательных файлов, что значительно уменьшает время выполнения алгоритма.

# **Заключение**

В ходе выполнения курсового проекта была разработана программа «……», позволяющая пользователем анализировать время выполнения разных алгоритмов внешней сортировки. Был выполнен анализ предметной области и функциональных требований. С учетом представленных требований разработана структура базы данных. Проверка качества кода осуществлена по методу функционального тестирования и разбиения на классы эквивалентности. Результаты тестирования показывают корректную работу приложения.

В процессе разработки в среде Microsoft Visual Studio на языке С# был создан проект с реализованными методами внешней сортировки данных.

# **Список использованной литературы**

1. Кнут Д. Э. Искусство программирования. Том 3 Сортировка и поиск/ Дональд Эрвин Кнут. –Издательство Вильямс, 2017 г. – 824 c.
2. И.А. Казакова, С.В. Самуйлов «Структуры данных. Учебное пособие» Пенза, Издательство ПГУ, 2011 г.
3. Многофазное слияние. – [Электронный ресурс] URL.: https://studopedia.su/13\_108646\_fibonachchievo-sliyanie.html (дата обращения: 11.05.2018)
4. Алгоритмы сортировки массивов. Внешняя сортировка. – [Электронный ресурс] URL.: https://www.intuit.ru/studies/courses/648/504/lecture/11473 (дата обращения: 10.05.2018)
5. Виды тестирования программного обеспечения. – [Электронный ресурс] URL.: <http://www.protesting.ru/testing/testtypes.html> (дата обращения: 16.05.2018)
6. Класс Array [Электронный ресурс] https://msdn.microsoft.com/ru-ru/library/system.array(v=vs.110).aspx (дата обращения 02.05.18)
7. Класс BinaryWriter. – [Электронный ресурс] URL.: https://msdn.microsoft.com/ru-ru/library/system.io.binarywriter(v=vs.110).aspx (дата обращения: 02.05.2018)
8. Класс BinaryReader. – [Электронный ресурс] URL.: https://msdn.microsoft.com/ru-ru/library/system.io.binaryreader(v=vs.110).aspx (дата обращения: 02.05.2018)
9. Основные требования к пользовательскому интерфейсу – [Электронный ресурс] URL.: http://vikidalka.ru/4-9695.html (дата обращения: 16.05.2018)

# **Приложение А. Листинг программы**

**Листинг класса Form1**

using System;

using System.IO;

using System.Windows.Forms;

namespace KR

{

public partial class Form1 : Form

{

SimpleSort simple = new SimpleSort();

NaturalSort natural = new NaturalSort();

AbsorptionSort absorption = new AbsorptionSort();

MultSort mult = new MultSort();

public Form1()

{

InitializeComponent();

}

private void compareButton\_Click(object sender, EventArgs e)

{

dataGridView1.Rows.Clear();

if (checkOnePhaseSimple.Checked |checkOnePhaseNatural.Checked | checkMult.Checked | checkAbsor.Checked)

{

int kolEnd = Convert.ToInt32(numericKol.Text);

int op = Convert.ToInt32(numericOP.Text);

int kolElinGraph = 10;

int kolBegin = kolEnd \* 10 / 100;

int step = (0 + kolEnd) / kolElinGraph;

long[] simpleOneMas = new long[kolElinGraph + 1];

long[] naturalOneMas = new long[kolElinGraph + 1];

long[] AbsorMas = new long[kolElinGraph + 1];

long[] multMas = new long[kolElinGraph + 1];

int[] kolElemMas = new int[kolElinGraph + 1];

chart1.ChartAreas[0].AxisX.Minimum = 0;

long time1 = 0, time2 = 0, time3= 0, time4 = 0;

for (int j = 1, kol = kolBegin; kol < kolEnd + 1; j++, kol += step)

{

kolElemMas[j] = kol;

using (BinaryWriter bw = new BinaryWriter(new FileStream("a.dat", FileMode.Create)))

{

Random randNumber = new Random();

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

{

int znach = 1 + randNumber.Next(100);

bw.Write(znach);

}

bw.Close();

}

if (checkOnePhaseSimple.Checked)

{

if (File.Exists("simpleOne.dat")) File.Delete("simpleOne.dat");

File.Copy("a.dat", "simpleOne.dat");

int colread = 0;

int colwrite = 0;

int colsr = 0;

long time = 0;

simple.OnePhaseSimple("simpleOne.dat", kol, ref time);

simpleOneMas[j] = time;

time1 = time;

}

if (checkOnePhaseNatural.Checked)

{

if (File.Exists("naturalOne.dat")) File.Delete("naturalOne.dat");

File.Copy("a.dat", "naturalOne.dat");

int colread = 0;

int colwrite = 0;

int colsr = 0;

long time = 0;

natural.OnePhaseNatural("naturalOne.dat", kol, ref time);

naturalOneMas[j] = time;

time2 = time;

}

if (checkAbsor.Checked)

{

if (File.Exists("Absorption.dat")) File.Delete("Absorption.dat");

File.Copy("a.dat", "Absorption.dat");

int colread = 0;

int colwrite = 0;

int colsr = 0;

long time = 0;

absorption.absorption("Absorption.dat", kol, op, ref time);

AbsorMas[j] = time;

time3 = time;

}

if (checkMult.Checked)

{

if (File.Exists("Mult.dat")) File.Delete("Mult.dat");

File.Copy("a.dat", "Mult.dat");

int colread = 0;

int colwrite = 0;

int colsr = 0;

long time = 0;

mult.sort("Mult.dat", kol, op, ref time);

multMas[j] = time;

time4 = time;

}

}

chart1.Series["Однофазная сортировка простым слиянием"].Points.Clear();

if (checkOnePhaseSimple.Checked)

{

dataGridView1.Rows.Add("Однофазная сортировка простым слиянием", time1);

chart1.Series["Однофазная сортировка простым слиянием"].Points.DataBindXY(kolElemMas, simpleOneMas);

}

chart1.Series["Однофазная сортировка естественным слиянием"].Points.Clear();

if (checkOnePhaseNatural.Checked)

{

dataGridView1.Rows.Add("Однофазная сортировка простым слиянием", time2);

chart1.Series["Однофазная сортировка естественным слиянием"].Points.DataBindXY(kolElemMas, naturalOneMas);

}

chart1.Series["Сортировка методом поглощения"].Points.Clear();

if (checkAbsor.Checked)

{

dataGridView1.Rows.Add("Сортировка методом поглощения", time3);

chart1.Series["Сортировка методом поглощения"].Points.DataBindXY(kolElemMas, AbsorMas);

}

chart1.Series["Многофазная сортировка слиянием"].Points.Clear();

if (checkMult.Checked)

{

dataGridView1.Rows.Add("Многофазная сортировка слиянием", time4);

chart1.Series["Многофазная сортировка слиянием"].Points.DataBindXY(kolElemMas, multMas);

}

}

else

{

const string message = "Выберите хотя бы 1 пункт";

const string caption = "Ни один пункт не выбран";

var result = MessageBox.Show(message, caption, MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

}

}

**Листинг класса SimpleMerge**

using System.Diagnostics;

using System.IO;

namespace KR

{

class SimpleSort

{

public void OnePhaseSimple(string a, int kol, ref long time)

{

int k = 1;

Stopwatch stopWatch = new Stopwatch();

stopWatch.Start();

using (BinaryWriter bwc = new BinaryWriter(new FileStream("c.dat", FileMode.Create)))

using (BinaryWriter bwb = new BinaryWriter(new FileStream("b.dat", FileMode.Create)))

using (BinaryReader br = new BinaryReader(new FileStream(a, FileMode.Open)))

{

while (br.PeekChar() > -1)

{

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

{

if (br.PeekChar() > -1)

{

int znach = br.ReadInt32();

bwb.Write(znach);

}

}

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

{

if (br.PeekChar() > -1)

{

int znach = br.ReadInt32();

bwc.Write(znach);

}

}

}

br.Close();

bwb.Close();

bwc.Close();

}

while (k < kol)

{

sortOne(ref k, new FileStream("b.dat", FileMode.Open), new FileStream("c.dat", FileMode.Open), new FileStream("d.dat", FileMode.Create), new FileStream("e.dat", FileMode.Create));

k \*= 2;

sortOne(ref k, new FileStream("d.dat", FileMode.Open), new FileStream("e.dat", FileMode.Open), new FileStream("b.dat", FileMode.Create), new FileStream("c.dat", FileMode.Create));

k \*= 2;

}

stopWatch.Stop();

time = stopWatch.ElapsedTicks;

if (File.Exists(a)) File.Delete(a);

File.Move("b.dat", a);

}

private void sortOne(ref int k, FileStream b, FileStream c, FileStream d, FileStream e)

{

using (BinaryReader brb = new BinaryReader(b))

using (BinaryReader brc = new BinaryReader(c))

using (BinaryWriter bwd = new BinaryWriter(d))

using (BinaryWriter bwe = new BinaryWriter(e))

{

int znach1 = -1;

int znach2 = -1;

if (brb.PeekChar() > -1)

{

znach1 = brb.ReadInt32();

}

if (brc.PeekChar() > -1)

{

znach2 = brc.ReadInt32();

}

int ind = 1;

while (znach1 != -1 && znach2 != -1)

{

int i = 0;

int j = 0;

while (i < k && j < k && znach1 != -1 && znach2 != -1)

{

if (znach1 < znach2)

{

if (ind % 2 != 0) { bwd.Write(znach1);}

else { bwe.Write(znach1); }

if (brb.PeekChar() > -1) { znach1 = brb.ReadInt32(); }

else znach1 = -1;

i++;

}

else

{

if (ind % 2 != 0) { bwd.Write(znach2); }

else { bwe.Write(znach2); }

if (brc.PeekChar() > -1) { znach2 = brc.ReadInt32(); }

else znach2 = -1;

j++;

}

}

while (i < k && znach1 != -1)

{

if (ind % 2 != 0) bwd.Write(znach1);

else bwe.Write(znach1);

if (brb.PeekChar() > -1) { znach1 = brb.ReadInt32(); }

else znach1 = -1;

i++;

}

while (j < k && znach2 != -1)

{

if (ind % 2 != 0) bwd.Write(znach2);

else bwe.Write(znach2);

if (brc.PeekChar() > -1) { znach2 = brc.ReadInt32(); }

else znach2 = -1;

j++;

}

ind++;

}

while (znach1 != -1)

{

if (ind % 2 != 0) bwd.Write(znach1);

else bwe.Write(znach1);

if (brb.PeekChar() > -1) { znach1 = brb.ReadInt32(); }

else znach1 = -1;

}

while (znach2 != -1)

{

if (ind % 2 != 0) bwd.Write(znach2);

else bwe.Write(znach2);

if (brc.PeekChar() > -1) { znach2 = brc.ReadInt32(); }

else znach2 = -1;

}

}

}

}

}

**Листинг класса NaturalSort**

using System.Diagnostics;

using System.IO;

namespace KR

{

class NaturalSort

{

public void splitting(FileStream a, FileStream b, FileStream c)

{

int file = 1;

using (BinaryWriter bwc = new BinaryWriter(c))

using (BinaryWriter bwb = new BinaryWriter(b))

using (BinaryReader br = new BinaryReader(a))

{

int znach = br.ReadInt32();

int nextznach = br.ReadInt32();

bwb.Write(znach);

while (nextznach != -1)

{

if (znach > nextznach)

{

switch (file)

{

case 1:

file = 2;

break;

case 2:

file = 1;

break;

}

}

if (file == 1)

{

bwb.Write(nextznach);

}

else

{

bwc.Write(nextznach);

}

znach = nextznach;

if (br.PeekChar() > -1) { nextznach = br.ReadInt32(); }

else nextznach = -1;

}

br.Close();

bwb.Close();

bwc.Close();

}

}

public void writeina1(ref int znach, ref int nextznach, ref int zapislast, ref int zapis, ref bool noots, ref bool ser1, BinaryWriter bw, BinaryReader brb)

{

bw.Write(znach);

zapislast = zapis;

zapis = znach;

if (zapislast > zapis) noots = true;

if (brb.PeekChar() != -1) { nextznach = brb.ReadInt32(); }

else nextznach = -1;

if (znach > nextznach || nextznach == -1) ser1 = false;

}

public void writeina(ref int znach, ref int nextznach, ref int zapislast, ref int zapis, ref bool noots, ref bool ser1, BinaryWriter bw, BinaryReader brb)

{

bw.Write(nextznach);

znach = nextznach;

zapislast = zapis;

zapis = znach;

if (zapislast > zapis) noots = true;

if (brb.PeekChar() != -1) { nextznach = brb.ReadInt32(); }

else nextznach = -1;

if (znach > nextznach || nextznach == -1) ser1 = false;

}

public void OnePhaseNatural(string a, int kol, ref long time)

{

Stopwatch stopWatch = new Stopwatch();

stopWatch.Start();

bool noots = true;

splitting(new FileStream(a, FileMode.Open), new FileStream("b4.dat", FileMode.Create), new FileStream("c4.dat", FileMode.Create));

noots = true;

bool v = true;

while (noots)

{

noots = false;

sortOne(ref noots, new FileStream("b4.dat", FileMode.Open), new FileStream("c4.dat", FileMode.Open), new FileStream("d4.dat", FileMode.Create), new FileStream("e4.dat", FileMode.Create));

v = true;

if (!noots) break;

noots = false;

sortOne(ref noots, new FileStream("d4.dat", FileMode.Open), new FileStream("e4.dat", FileMode.Open), new FileStream("b4.dat", FileMode.Create), new FileStream("c4.dat", FileMode.Create));

v = false;

}

stopWatch.Stop();

time = stopWatch.ElapsedTicks;

if (File.Exists(a)) File.Delete(a);

if (v) File.Move("d4.dat", a);

else File.Move("b4.dat", a);

}

public void sortOne(ref bool noots, FileStream b, FileStream c, FileStream d, FileStream e)

{

using (BinaryReader brb = new BinaryReader(b))

using (BinaryReader brc = new BinaryReader(c))

using (BinaryWriter bwd = new BinaryWriter(d))

using (BinaryWriter bwe = new BinaryWriter(e))

{

int switcf = 1;

int znach1 = brb.ReadInt32();

int znach2 = brc.ReadInt32();

int nextznach1 = znach1;

int nextznach2 = znach2;

int zapislast = -1;

int zapis = -1;

bool ser1 = true, ser2 = true;

if (znach1 <= znach2)

{

if (switcf == 1) writeina1(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwd, brb);

else writeina1(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwe, brb);

}

else

{

if (switcf == 1) writeina1(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwd, brc);

else writeina1(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwe, brc);

}

while ((nextznach1 != -1) && (nextznach2 != -1))

{

while (ser1 && ser2)

{

if (nextznach1 <= nextznach2)

{

if (switcf == 1) writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwd, brb);

else writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwe, brb);

}

else

{

if (switcf == 1) writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwd, brc);

else writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwe, brc);

}

}

while (ser1)

{

if (switcf == 1) writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwd, brb);

else writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwe, brb);

}

while (ser2)

{

if (switcf == 1) writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwd, brc);

else writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwe, brc);

}

if (switcf == 1) switcf = 2;

else switcf = 1;

ser1 = ser2 = true;

}

ser1 = ser2 = true;

while (nextznach1 != -1)

{

if (switcf == 1) writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwd, brb);

else writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref noots, ref ser1, bwe, brb);

if (znach1 > nextznach1)

{

if (switcf == 1) switcf = 2;

else switcf = 1;

}

}

while (nextznach2 != -1)

{

if (switcf == 1) writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwd, brc);

else writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref noots, ref ser2, bwe, brc);

if (znach2 > nextznach2)

{

if (switcf == 1) switcf = 2;

else switcf = 1;

}

}

bwd.Close();

bwe.Close();

brb.Close();

brc.Close();

}

}

}

}

**Листинг класса AbsorptionSort**

using System;

using System.Diagnostics;

using System.IO;

namespace KR

{

class AbsorptionSort

{

Selection ins = new Selection();

public void absorption(string a, int kol, int Prop, ref long time)

{

Stopwatch sw = new Stopwatch();

int op = (int)(kol \* Prop) / 100;

int len = (int)kol / op;

int[] ser = new int[op];

int ost = kol % op;

if (ost != 0)

using (FileStream fs = File.Open(a, FileMode.Open))

{

for (int i = 0; i < op - ost; i++)

{

Byte[] number = new Byte[4];

number = BitConverter.GetBytes(0);

fs.Seek(0, SeekOrigin.End);

fs.Write(number, 0, 4);

}

}

sw.Restart();

using (FileStream fs = File.Open(a, FileMode.Open))

{

Byte[] number = new Byte[4];

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

{

fs.Seek(-4 \* (i + 1), SeekOrigin.End);

fs.Read(number, 0, 4);

ser[i] = BitConverter.ToInt32(number, 0); ;

}

ins.selection(ser);

fs.Seek(-4 \* (op), SeekOrigin.End);

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

{

number = BitConverter.GetBytes(ser[i]);

fs.Write(number, 0, 4);

}

}

int serlength = op;

while (serlength < kol)

{

using (FileStream fs = File.Open(a, FileMode.Open))

{

int smesh = serlength + op;

int smechread = serlength;

fs.Seek(-4 \* (smesh), SeekOrigin.End);

Byte[] numbers = new Byte[4 \* op];

fs.Read(numbers, 0, 4 \* op);

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

{

ser[i] = BitConverter.ToInt32(numbers, 4 \* i);

}

ins.selection(ser);

Byte[] number = new Byte[4];

fs.Seek(-4 \* smechread, SeekOrigin.End);

smechread--;

fs.Read(number, 0, 4);

int znach = BitConverter.ToInt32(number, 0);

int znach2 = ser[0];

int lenfile = 0, j = 0;

while (lenfile < serlength && j < op)

{

if (znach < znach2)

{

number = BitConverter.GetBytes(znach);

fs.Seek(-4 \* (smesh), SeekOrigin.End);

fs.Write(number, 0, 4);

smesh--;

fs.Seek(-4 \* (smechread), SeekOrigin.End);

smechread--;

fs.Read(number, 0, 4);

znach = BitConverter.ToInt32(number, 0);

lenfile++;

}

else

{

number = BitConverter.GetBytes(znach2);

fs.Seek(-4 \* (smesh), SeekOrigin.End);

smesh--;

fs.Write(number, 0, 4);

j++;

if (j < op) znach2 = ser[j];

}

}

while (lenfile < serlength)

{

number = BitConverter.GetBytes(znach);

fs.Seek(-4 \* (smesh), SeekOrigin.End);

fs.Write(number, 0, 4);

smesh--;

fs.Seek(-4 \* (smechread), SeekOrigin.End);

smechread--;

fs.Read(number, 0, 4);

znach = BitConverter.ToInt32(number, 0);

lenfile++;

}

while (j < op)

{

number = BitConverter.GetBytes(znach2);

fs.Seek(-4 \* (smesh), SeekOrigin.End);

fs.Write(number, 0, 4);

smesh--;

j++;

if (j < op) znach2 = ser[j];

}

serlength += op;

}

}

sw.Stop();

time = sw.ElapsedTicks;

}

}

}

**Листинг класса Multphase**

using System.Diagnostics;

using System.IO;

namespace KR

{

Selection ins = new Selection();

public void sort(string w, int kol, int Prop, ref long time)

{

Stopwatch sw = new Stopwatch();

sw.Start();

int op = (int)((kol+1) \* Prop) / 100;

int[] ser = new int[op];

int q = 0, p = 0;

bool end = false;

int kolSer = (kol + 1) % op;

if (kolSer > 0)

{

kolSer = 1 + ((int)(kol + 1) / op);

}

else

kolSer =( kol + 1) / op;

int serA = 1, serB = 1;

if (kolSer <= 21 && kolSer > 13)

{

serA = 13; serB = 8;

}

else

{

if (kolSer <= 13 && kolSer > 8)

{

serA = 8; serB = 5;

}

else

{

if (kolSer <= 8 && kolSer > 5)

{

serA = 5; serB = 3;

}

else

{

if (kolSer <= 5 && kolSer > 3)

{

serA = 3; serB = 2;

}

else

{

if (kolSer == 3)

{

serA = 2; serB = 1;

}

}

}

}

}

using (BinaryReader br = new BinaryReader(new FileStream("a1.dat", FileMode.Open)))

using (BinaryWriter bw1 = new BinaryWriter(new FileStream("A11.dat", FileMode.Create)))

using (BinaryWriter bw2 = new BinaryWriter(new FileStream("B11.dat", FileMode.Create)))

{

while (q < serA + serB)

{

if (!(br.PeekChar() > -1))

{

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

ser[i] = 101;

}

else

{

while (p < op)

{

if (br.PeekChar() > -1)

ser[p] = br.ReadInt32();

else

{

end = true;

ser[p] = 101;

}

p++;

}

}

q++;

if (!end)

ins.selection(ser);

p = 0;

if (q <= serB \* 2)

{

if (q % 2 != 0)

{

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

bw1.Write(ser[i]);

}

else

{

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

bw2.Write(ser[i]);

}

}

else

{

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

bw1.Write(ser[i]);

}

Array.Clear(ser, 0, op);

}

br.Close();

bw1.Close();

bw2.Close();

}

string a = "A11.dat";

string b = "B11.dat";

string c = "C11.dat";

bool noots = true;

while (noots)

{

sorting(ref a, ref b, ref c, ref noots);

}

sw.Stop();

time = sw.ElapsedTicks;

}

public void check(ref bool noots, string b, int e)

{

using (BinaryReader brb = new BinaryReader(new FileStream(b, FileMode.Open)))

{

if (brb.PeekChar() > -1)

{

e = brb.ReadInt32();

}

brb.Close();

}

if (e == -1)

noots = false;

}

public void sorting(ref string a, ref string b, ref string c, ref bool noots)

{

check(ref noots, b,-1);

using (BinaryReader brb = new BinaryReader(new FileStream(a, FileMode.Open)))

using (BinaryReader brc = new BinaryReader(new FileStream(b, FileMode.Open)))

using (BinaryWriter bwd = new BinaryWriter(new FileStream(c, FileMode.Create)))

using (BinaryWriter bwa = new BinaryWriter(new FileStream("template.dat", FileMode.Create)))

{

if (brc.PeekChar() > -1)

{

int znach1 = brb.ReadInt32();

int znach2 = brc.ReadInt32();

int nextznach1 = znach1;

int nextznach2 = znach2;

int zapislast = -1;

int zapis = -1;

bool ser1 = true, ser2 = true;

if (znach1 <= znach2)

{

writeina1(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref ser1, bwd, brb);

}

else

{

writeina1(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref ser2, bwd, brc);

}

while ((nextznach1 != -1) && (nextznach2 != -1))

{

while (ser1 && ser2)

{

if (nextznach1 <= nextznach2)

{

writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref ser1, bwd, brb);

}

else

{

writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref ser2, bwd, brc);

}

}

while (ser1)

{

writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref ser1, bwd, brb);

}

while (ser2)

{

writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref ser2, bwd, brc);

}

ser1 = ser2 = true;

}

while (nextznach1 != -1)

{

writeina(ref znach1, ref nextznach1, ref zapislast, ref zapis, ref ser1, bwa, brb);

}

while (nextznach2 != -1)

{

writeina(ref znach2, ref nextznach2, ref zapislast, ref zapis, ref ser2, bwa, brc);

}

brb.Close();

brc.Close();

bwd.Close();

bwa.Close();

File.Delete(b);

File.Delete(a);

File.Move("template.dat", a);

}

}

string temp = b;

b = a;

a = c;

c = temp;

}

public void writeina1(ref int znach, ref int nextznach, ref int zapislast, ref int zapis, ref bool ser1, BinaryWriter bw, BinaryReader brb)

{

bw.Write(znach);

zapislast = zapis;

zapis = znach;

if (brb.PeekChar() != -1) { nextznach = brb.ReadInt32(); }

else nextznach = -1;

if (znach > nextznach || nextznach == -1) ser1 = false;

}

public void writeina(ref int znach, ref int nextznach, ref int zapislast, ref int zapis, ref bool ser1, BinaryWriter bw, BinaryReader brb)

{

bw.Write(nextznach);

znach = nextznach;

zapislast = zapis;

zapis = znach;

if (brb.PeekChar() != -1) { nextznach = brb.ReadInt32(); }

else nextznach = -1;

if (znach > nextznach || nextznach == -1) ser1 = false;

}

}

}

**Листинг класса Selection**

namespace KR

{

class Selection

{

public int[] selection(int[] a)

{

int j = a.Length - 1;

int kol = 0;

while (j > 0)

{

int m = a[0];

int k = 0;

int i = 1;

while (i <= j)

{

if (m < a[i])

{

m = a[i];

k = i;

}

i++;

}

a[k] = a[j];

a[j] = m;

j--;

kol++;

}

return a;

}

}

}