МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

Кафедра информационных технологий и экономической информатики

ОТЧЕТ

по лабораторной работе №4

Выполнил:

Студент группы ПрИ-202 Приходько Даниил Александрович

Студент группы ПрИ-202 Саламатин Алексей Юрьевич

Студент группы ПрИ-202 Скоробогатов Максим Дмитриевич

Принял:

Преподаватель ИИТ Николаев Иван Евгеньевич

Отчет защищен \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата оценка

Челябинск 2024 г.

Содержание

Введение..................................................................................................................3

Задание 1………......................................................................................................4

Задание 2................................................................................................................17

Задание 3................................................................................................................29

Введение

Цель работы: понять и доступно объяснить Шестикласснику, как работают различные сортировки данных. Написать приложение с визуальной частью, благодаря которой будет видно весь процесс сортировок.

Задание 1: Реализовать демонстрацию работы 4 алгоритмов внутренней сортировки. А конкретно, нами были выбраны: Select Sort, Shell Sort, Quick Sort, Merge Sort.

Задание 2: Дана таблица, состоящая из нескольких полей. Таблица находится в файле. Реализовать программу, которая одним из методов (прямым, естественным или многопутевым слиянием) сортирует записи и записывает их в результирующий файл. Метод внешней сортировки и ключевой атрибут, по которому осуществляется сортировка выбирается пользователем. Количество атрибутов в таблица может быть произвольным.

Задание 3: Пусть имеется текст, состоящий из слов. Необходимо разбить текст на отдельные слова и провести их сортировку в лексикографическом порядке. Причем одинаковые слова будут в отсортированной последовательности идти друг за другом. Надо выбрать два алгоритма сортировки, один из которых базовый или усовершенствованный, и другой алгоритм ABC- сортировка или redix сортировка. После получения отсортированного массива необходимо пройтись по нему и для каждого слова подсчитать, сколько раз оно встречается. Результаты подсчетов вывести на экран. Далее необходимо провести эксперименты с использованием обоих алгоритмов по сортировке текстов различной длины (100, 500, 1000, 2000, 5000 слов или более). Причем для каждого эксперимента произвести замеры времени сортировки. Получившиеся

данные оформить в таблицу.

Задание 1

В рамках этой части были реализованы алгоритмы внутренней сортировки Select Sort, Shell Sort, Quick Sort, Merge Sort.

Сортировка выбором – это алгоритм, суть которого заключается в постоянном сравнении элементов неотсортированой части. Наименьший элемент в неотсортированой части меняется местами с первым ее элементом и сразу становится отсортированой частью.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 1.1.1 – 1.1.3.

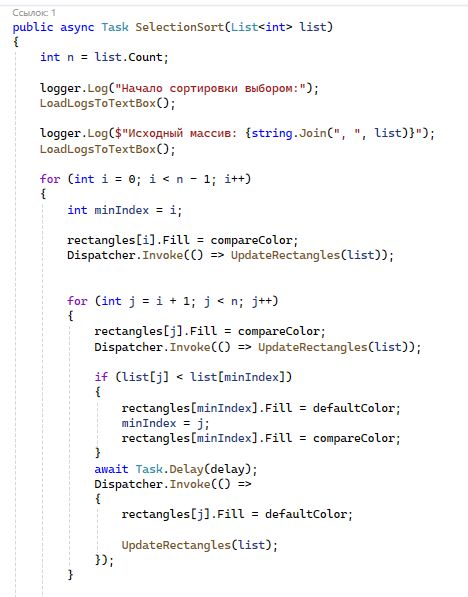


Рис.1.1.1 Код сортировки выбором

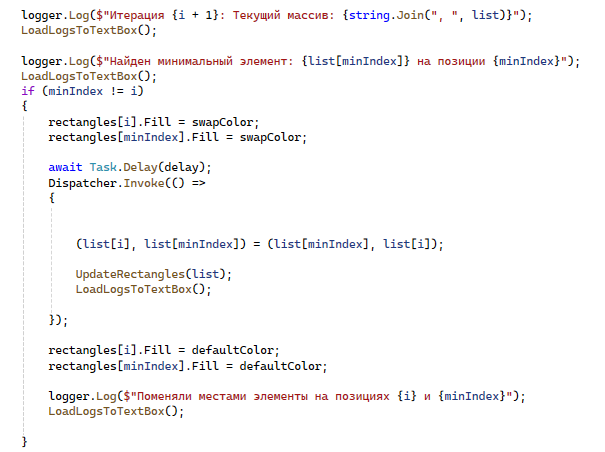


Рис.1.1.2 Код сортировки выбором

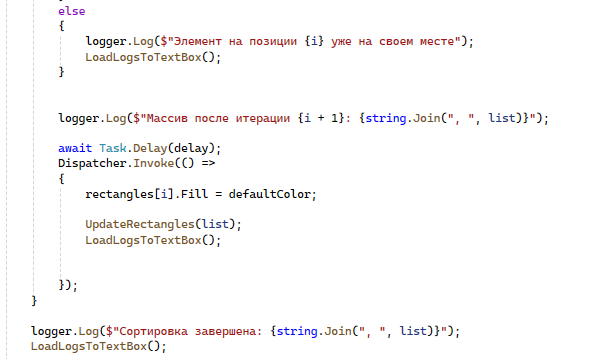


Рис.1.1.3 Код сортировки выбором

Сортировка Шелла – это модифицированая сортировка вставками. Ее суть в том, что массив данных разбивается на так называемые Gap’ы (зазоры). Каждая итерация сортировки – это сравнение элементов массива на расстоянии текущего зазора друг от друга, а затем упорядочивание этих элементов. С каждым проходом по массиву зазор уменьшается вдвое и эти действия повторяются до окончания сортировки.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 1.2.1 – 1.2.2.

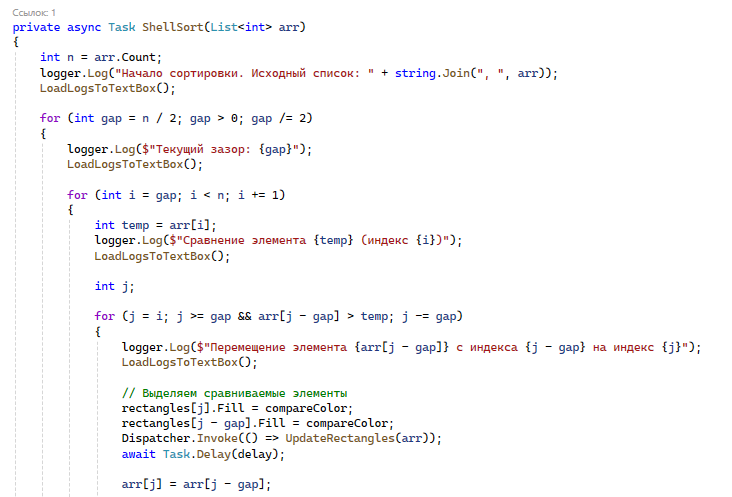


Рис.1.2.1 Код сортировки Шелла

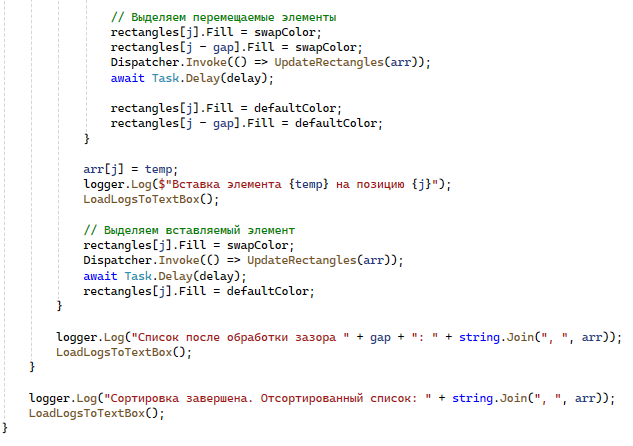


Рис.1.2.2 Код сортировки Шелла

Быстрая сортировка - это алгоритм "разделяй и властвуй", который рекурсивно сортирует массив, разделяя его на части относительно опорного элемента и объединяя отсортированные части.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 1.3.1 – 1.3.3.

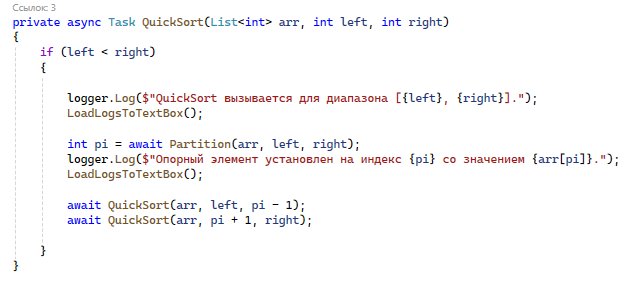


Рис.1.3.1 Код быстрой сортировки

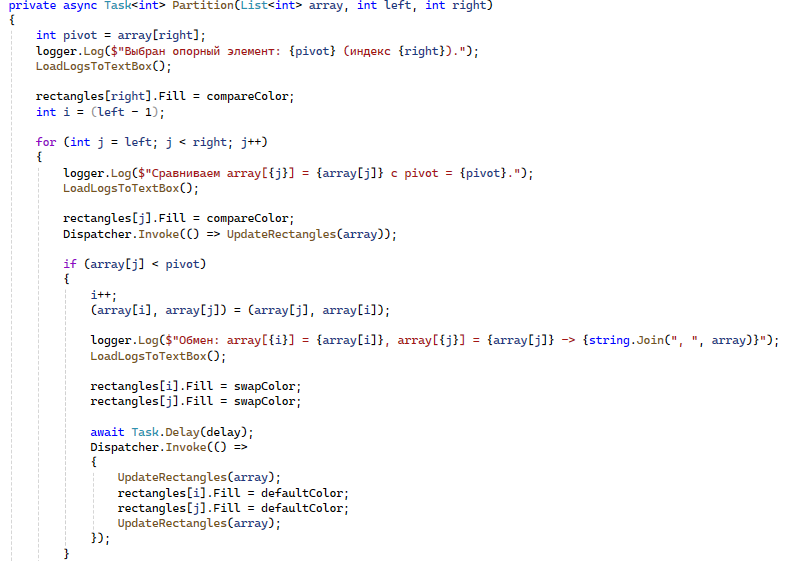


Рис.1.3.2 Код быстрой сортировки

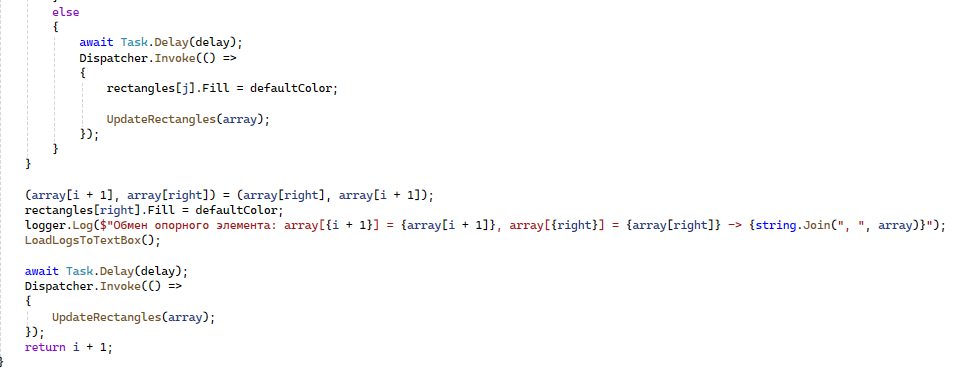


Рис.1.3.3 Код быстрой сортировки

Сортировка слиянием – это алгоритм который разделяет массив данных на две равные части, потом вызывает сам себя для каждой из этих частей и так до тех пор, пока массивы не станут единичными. Дальше эти подмассивы сравниваются со своей «половинкой» и в упорядоченом виде объединяются в новый отсортированый массив.

Данный алгоритм является внутренним, несмотря на то что есть его аналоги в виде внешних сортировок.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 1.4.1 – 1.4.4.

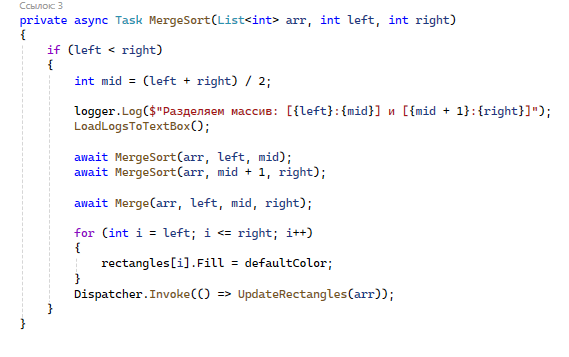


Рис.1.4.1 Код сортировки слиянием

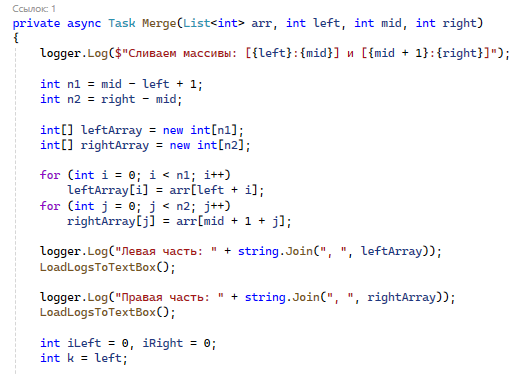


Рис.1.4.2 Код сортировки слиянием

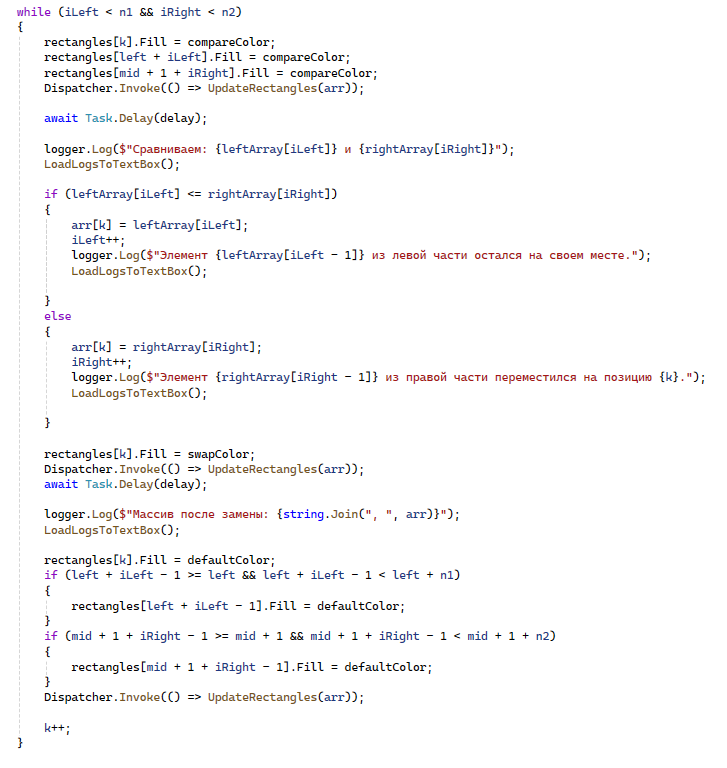


Рис.1.4.3 Код сортировки слиянием

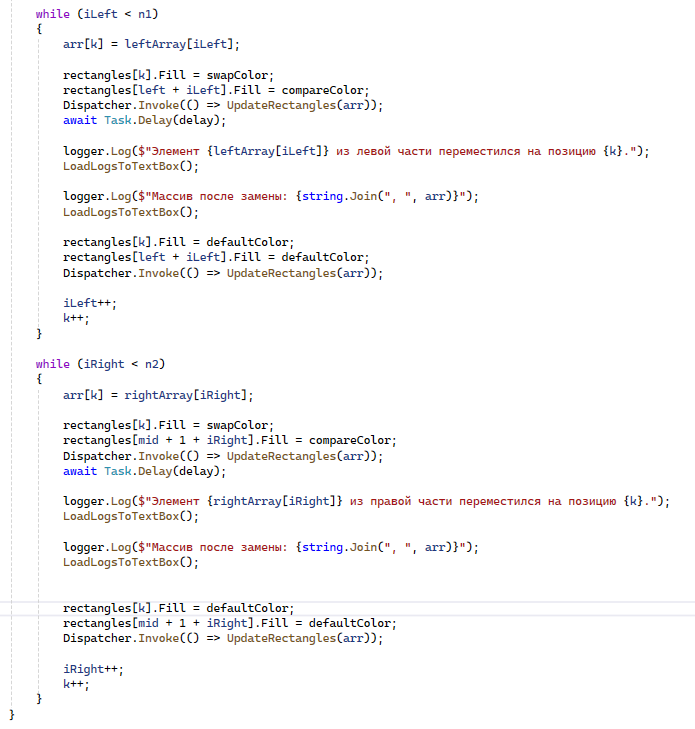


Рис.1.4.4 Код сортировки слиянием

Чтобы наглядно посмотреть на процесс сортировки был написан GUI для данной части (рис.1.5), где видно как проходит сортировка элементов, которые представлены в виде столбцов с соответствующей величиной. Так же есть окно, куда с определенной задержкой, которую выбирает пользователь, выводятся сопутствующие объяснения. Сам массив пользователь может ввести в верхнем окне ввода, через запятую или пробел. Выбрать желаемую сортировку можно в панели управления, в правой части окна.

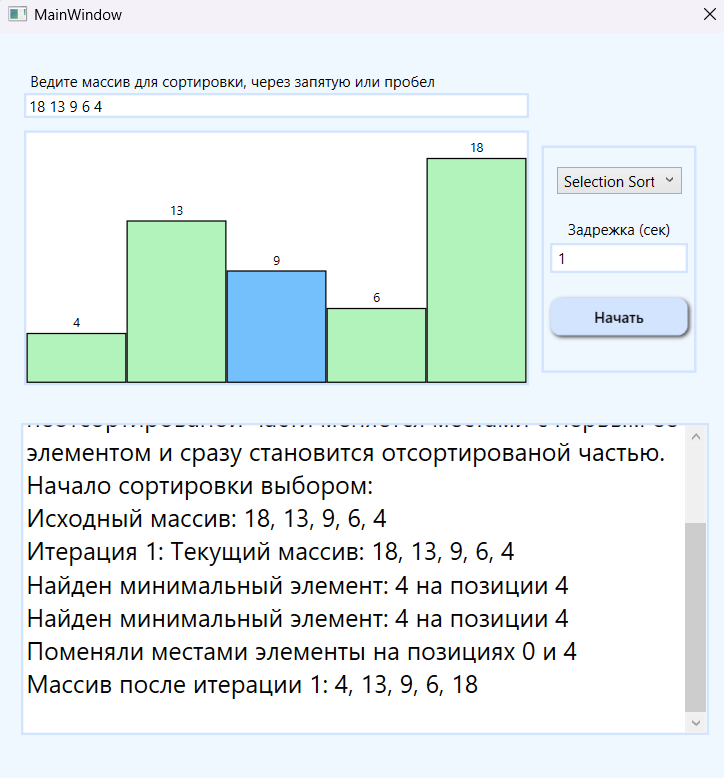


Рис.1.5 визуализация первого задания

Задание 2

Эта часть работы заключается в том, чтобы показать наглядно, и объяснить, как работают внешние сортировки, такие как прямое, естественное и многопутевое слияния.

Прямая сортировка слиянием:

Алгоритм внешней сортировки прямым слиянием предназначен для упорядочивания больших объемов данных, которые не помещаются в оперативную память компьютера. Этот метод основан на процедуре слияния серий и включает несколько ключевых шагов, которые повторяются до полного упорядочивания данных.

Основные шаги алгоритма:

1. Инициализация:
   * Исходный файл *f* разбивается на два вспомогательных файла *f*1 и *f*2. На этом этапе каждый элемент исходного файла рассматривается как отдельная серия длиной 1.
2. Слияние серий:
   * Вспомогательные файлы *f*1 и *f*2 сливаются в файл *f*, при этом одиночные элементы образуют упорядоченные пары. Например, если в *f*1 и *f*2 содержатся элементы [a1, a2, a3, a4] и [b1, b2, b3, b4] соответственно, то после слияния в *f* будут упорядоченные пары [min(a1, b1), max(a1, b1)], [min(a2, b2), max(a2, b2)] и так далее.
3. Повторение процесса:
   * Полученный файл *f* вновь обрабатывается, как указано в шагах 1 и 2. При этом упорядоченные пары переходят в упорядоченные четверки. Например, если в *f* содержатся упорядоченные пары [c1, c2] и [d1, d2], то после повторного слияния в *f* будут упорядоченные четверки [min(c1, d1), min(c2, d2), max(c1, d1), max(c2, d2)].
4. Удвоение длины серий:
   * Повторяя шаги, сливаем четверки в восьмерки и так далее, каждый раз удваивая длину слитых последовательностей до тех пор, пока не будет упорядочен целиком весь файл. Например, если в *f* содержатся упорядоченные четверки [e1, e2, e3, e4] и [f1, f2, f3, f4], то после очередного слияния в *f* будут упорядоченные восьмерки [min(e1, f1), min(e2, f2), min(e3, f3), min(e4, f4), max(e1, f1), max(e2, f2), max(e3, f3), max(e4, f4)].

Завершение процесса:

После выполнения *i* проходов получаем два файла, состоящих из серий длины 2*i*. Окончание процесса происходит при выполнении условия 2*i*≥*n*, где *n* — количество элементов в исходном файле. Следовательно, процесс сортировки простым слиянием требует порядка O(logn) проходов по данным.

Признаки завершения сортировки:

* Длина серии не меньше количества элементов в файле (определяется после фазы слияния).
* Количество серий равно 1 (определяется на фазе слияния).
* При однофазной сортировке второй по счету вспомогательный файл после распределения серий остался пустым.

Преимущества алгоритма:

* Эффективность при работе с большими объемами данных, которые не помещаются в оперативную память.
* Минимизация количества проходов по данным, что особенно важно при работе с внешними носителями информации.

Таким образом, алгоритм внешней сортировки прямым слиянием является мощным инструментом для упорядочивания больших массивов данных, обеспечивая эффективное использование ресурсов и минимизацию времени выполнения.

Код сортировки представлен на рисунке 2.1.1 – 2.1.2.



Рис.2.1.1 Код сортировки прямым слиянием

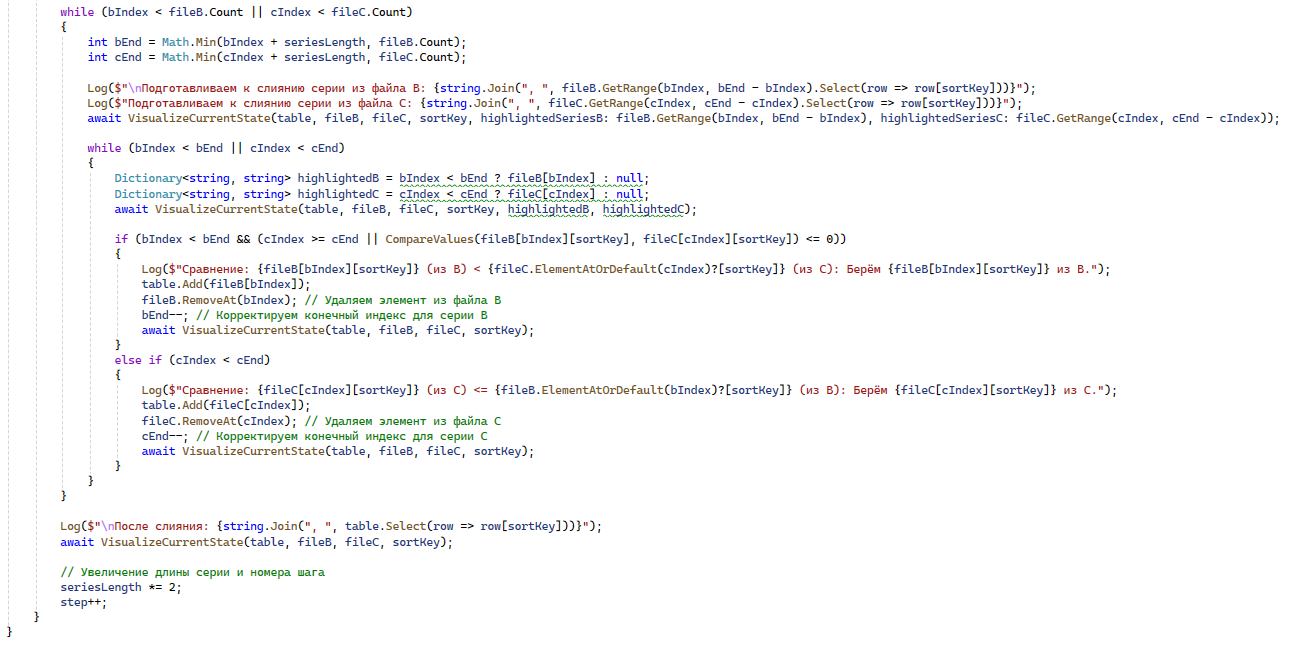


Рис.2.1.2 Код сортировки прямым слиянием

Естественное слияние:  
  
Алгоритм внешней сортировки естественным слиянием представляет собой метод упорядочивания больших объемов данных, которые не помещаются в оперативную память компьютера. В отличие от прямого слияния, где длина серий фиксируется на каждом шаге, естественное слияние использует естественные последовательности, которые уже упорядочены в исходных данных. Это позволяет более эффективно обрабатывать данные, особенно если они уже частично упорядочены.

Основные шаги алгоритма:

1. Инициализация:
   * Исходный файл *f* разбивается на серии естественных последовательностей. Естественная последовательность — это наибольшая возможная упорядоченная подпоследовательность в исходных данных. Например, если исходный файл содержит элементы [3, 5, 7, 2, 4, 6, 8, 1], то естественные последовательности будут [3, 5, 7], [2, 4, 6, 8] и [1].
2. Распределение серий:
   * Естественные последовательности распределяются по двум вспомогательным файлам *f*1 и *f*2. Например, последовательности [3, 5, 7] и [1] могут быть распределены в *f*1, а последовательность [2, 4, 6, 8] — в *f*2.
3. Слияние серий:
   * Вспомогательные файлы *f*1 и *f*2 сливаются в файл *f*, при этом серии объединяются в более длинные упорядоченные последовательности. Например, если в *f*1 содержатся последовательности [3, 5, 7] и [1], а в *f*2 — [2, 4, 6, 8], то после слияния в *f* будут последовательности [2, 3, 4, 5, 6, 7, 8, 1].
4. Повторение процесса:
   * Полученный файл *f* вновь обрабатывается, как указано в шагах 1-3. При этом упорядоченные последовательности объединяются в еще более длинные серии. Процесс повторяется до тех пор, пока не будет упорядочен целиком весь файл.

Завершение процесса:

Процесс сортировки естественным слиянием завершается, когда все элементы исходного файла объединены в одну упорядоченную последовательность. В отличие от прямого слияния, количество проходов по данным может варьироваться в зависимости от степени естественной упорядоченности исходных данных. В худшем случае, если данные полностью не упорядочены, алгоритм потребует порядка O(logn)проходов, где *n* — количество элементов в исходном файле.

Признаки завершения сортировки:

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

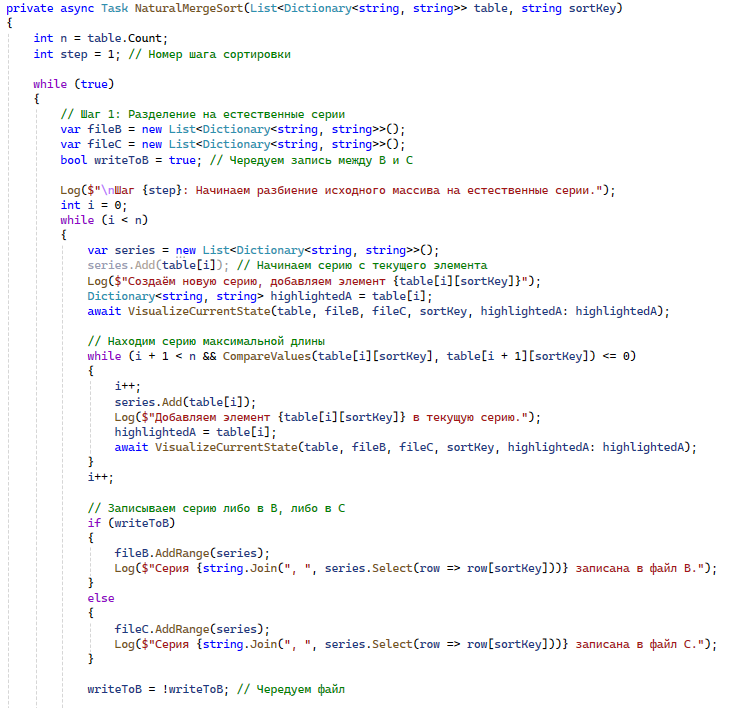
Преимущества алгоритма:

* Эффективность при работе с частично упорядоченными данными, что позволяет сократить количество проходов по данным.
* Минимизация времени выполнения за счет использования естественных последовательностей.
* Универсальность для различных типов данных и их распределений.

Таким образом, алгоритм внешней сортировки естественным слиянием является мощным инструментом для упорядочивания больших массивов данных, обеспечивая эффективное использование ресурсов и минимизацию времени выполнения, особенно при наличии частично упорядоченных данных.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода, который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 2.2.1 – 2.2.3.

  
Рис.2.2.1 Код сортировки естественным слиянием

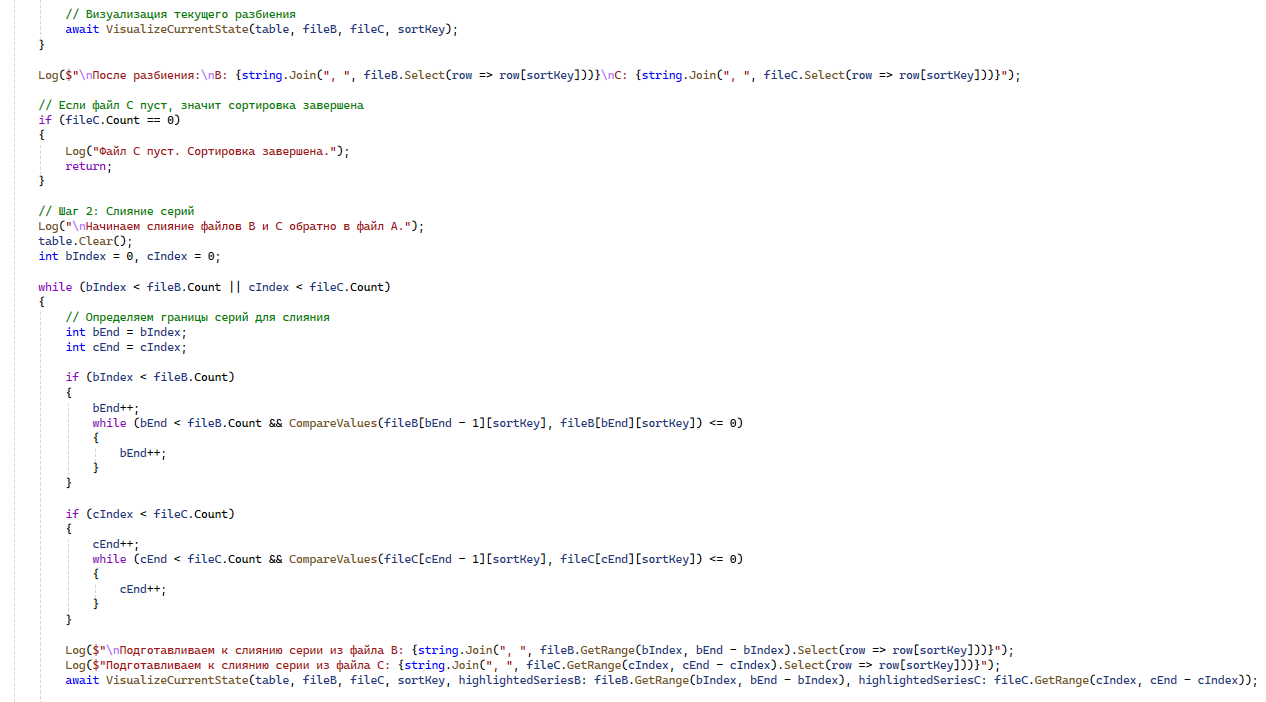


Рис.2.2.2 Код сортировки естественным слиянием

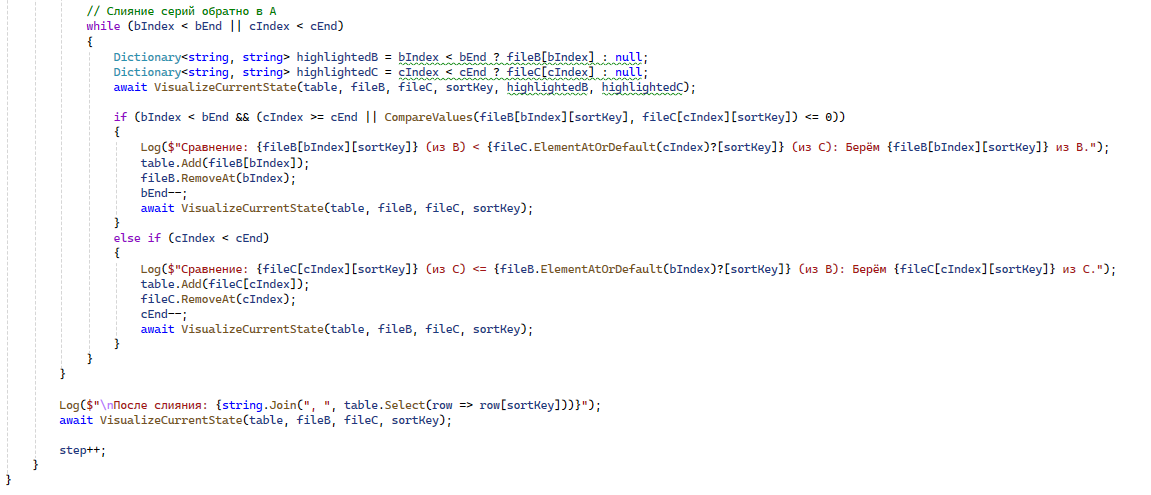


Рис.2.2.3 Код сортировки естественным слиянием

Многопутевое слияние:

Алгоритм внешней сортировки многопутевым слиянием представляет собой расширение метода двухпутевого, позволяя обрабатывать большие объемы данных, которые не помещаются в оперативную память компьютера. В этом алгоритме данные разбиваются на несколько вспомогательных файлов, которые затем сливаются в упорядоченные последовательности. Количество путей слияния (k) может быть любым натуральным числом, что позволяет гибко настраивать алгоритм в зависимости от доступных ресурсов и объема данных.

Основные шаги алгоритма:

1. Инициализация:
   * Исходный файл *f* разбивается на *k* вспомогательных файлов *f*1,*f*2,…,*fk*. На этом этапе каждый элемент исходного файла рассматривается как отдельная серия длиной 1.
2. Слияние серий:
   * Вспомогательные файлы *f*1,*f*2,…,*fk* сливаются в файл *f*, при этом одиночные элементы образуют упорядоченные последовательности длиной *k*. Например, если в *f*1,*f*2,…,*fk* содержатся элементы [a1, a2, … , ak], то после слияния в *f* будут упорядоченные последовательности [min(a1, a2, … , ak), mid(a1, a2, … , ak), … , max(a1, a2, … , ak)].
3. Повторение процесса:
   * Полученный файл *f* вновь обрабатывается, как указано в шагах 1 и 2. При этом упорядоченные последовательности длиной *k* переходят в упорядоченные последовательности длиной *k*2. Например, если в *f* содержатся упорядоченные последовательности [b1, b2, … , bk] и [c1, c2, … , ck], то после повторного слияния в *f* будут упорядоченные последовательности длиной *k*2.
4. Увеличение длины серий:
   * Повторяя шаги, сливаем последовательности длиной *k*2 в последовательности длиной *k*3 и так далее, каждый раз увеличивая длину слитых последовательностей до тех пор, пока не будет упорядочен целиком весь файл. Например, если в *f* содержатся упорядоченные последовательности длиной *k*2 [d1, d2, … , dk2] и [e1, e2, … , ek2], то после очередного слияния в *f* будут упорядоченные последовательности длиной *k*3.

Завершение процесса:

После выполнения *i* проходов получаем *k* файлов, состоящих из серий длины *ki*. Окончание процесса происходит при выполнении условия *ki*≥*n*, где *n* — количество элементов в исходном файле. Следовательно, процесс сортировки многопутевым слиянием требует порядка O(log(k)n)проходов по данным.

Признаки завершения сортировки:

* Длина серии не меньше количества элементов в файле (определяется после фазы слияния).
* Количество серий равно 1 (определяется на фазе слияния).
* При однофазной сортировке все вспомогательные файлы, кроме одного, после распределения серий остались пустыми.

Преимущества алгоритма:

* Эффективность при работе с большими объемами данных, которые не помещаются в оперативную память.
* Гибкость в настройке количества путей слияния (k), что позволяет оптимизировать процесс сортировки в зависимости от доступных ресурсов.
* Минимизация количества проходов по данным, что особенно важно при работе с внешними носителями информации.

Таким образом, алгоритм внешней сортировки многопутевым слиянием является универсальным и мощным инструментом для упорядочивания больших массивов данных, обеспечивая эффективное использование ресурсов и минимизацию времени выполнения.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода, который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 2.3.1 – 2.3.2.

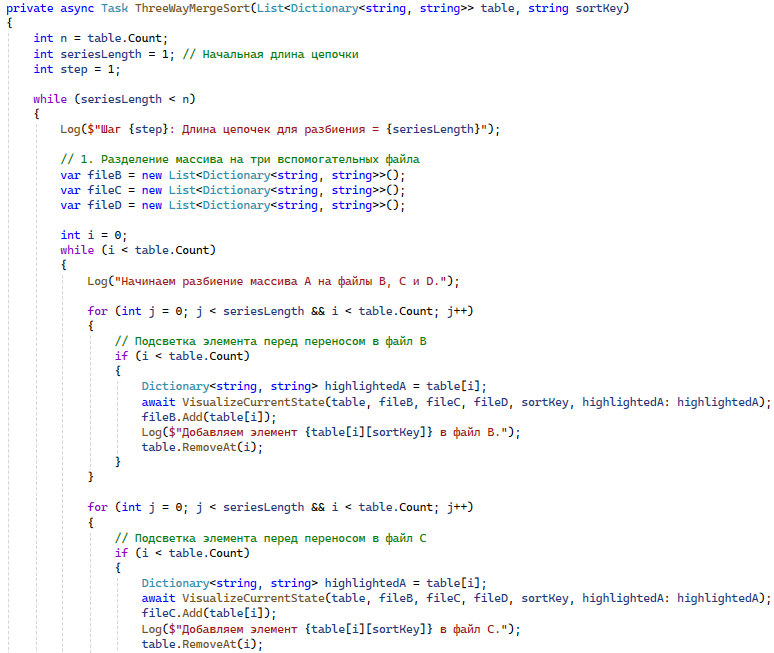


Рис.2.3.1 Код сортировки многопутевым слиянием

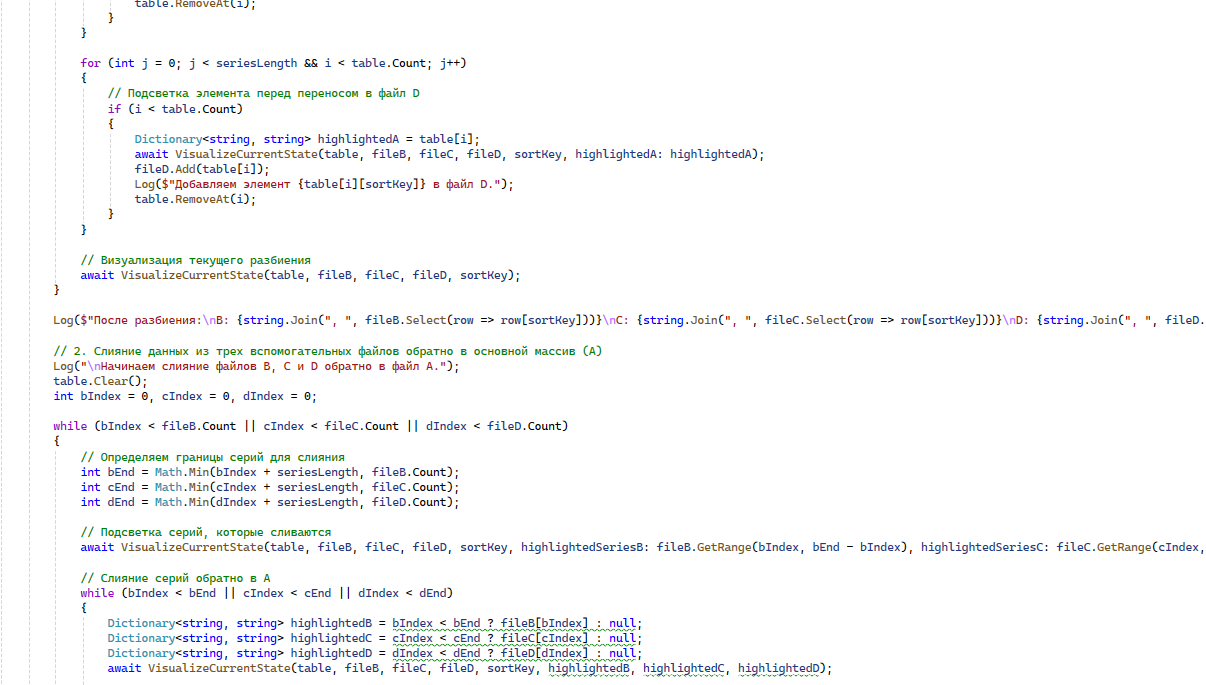


Рис.2.3.2 Код сортировки многопутевым слиянием

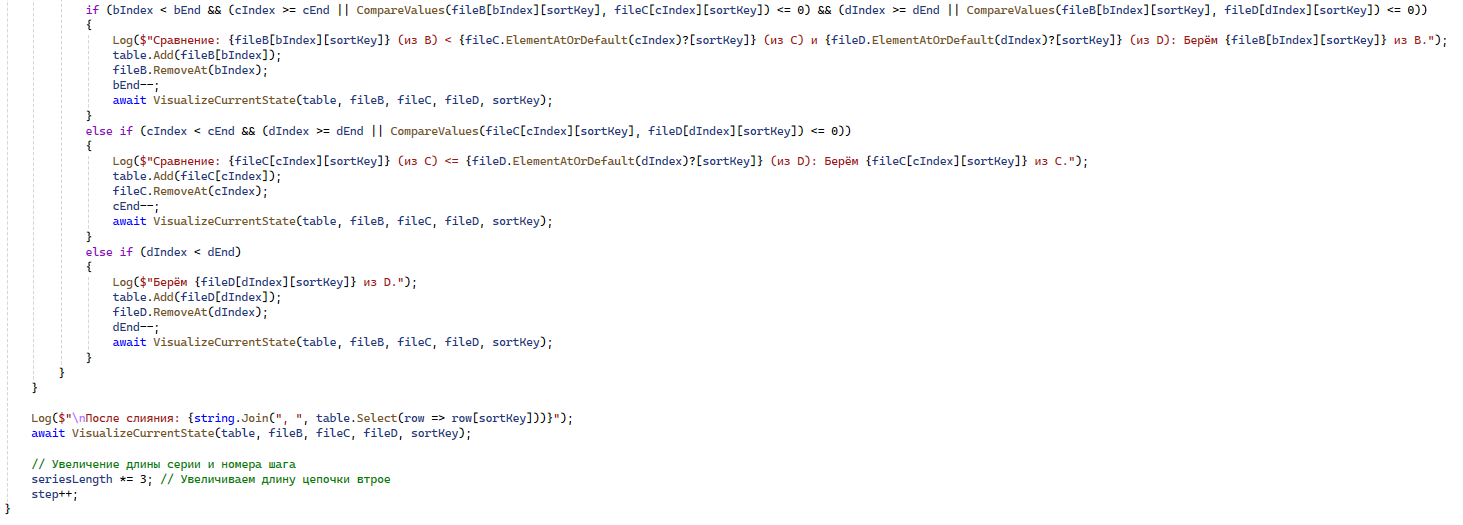


Рис.2.3.2 Код сортировки многопутевым слиянием

Чтобы наглядно посмотреть на процесс сортировки был написан GUI для данной части (рис.2.4), где видно, как проходит сортировка элементов таблицы. Пользователю представлен выбор сортировки, ключа. Так же он может пошагово отслеживать изменения во время процесса, благодаря кнопке автопроигрывание.

В отдельных окнах выводится визуализация сортировки в виде состояния таблицы в текущий момент времени, текстовая документация процесса и начальная таблица.

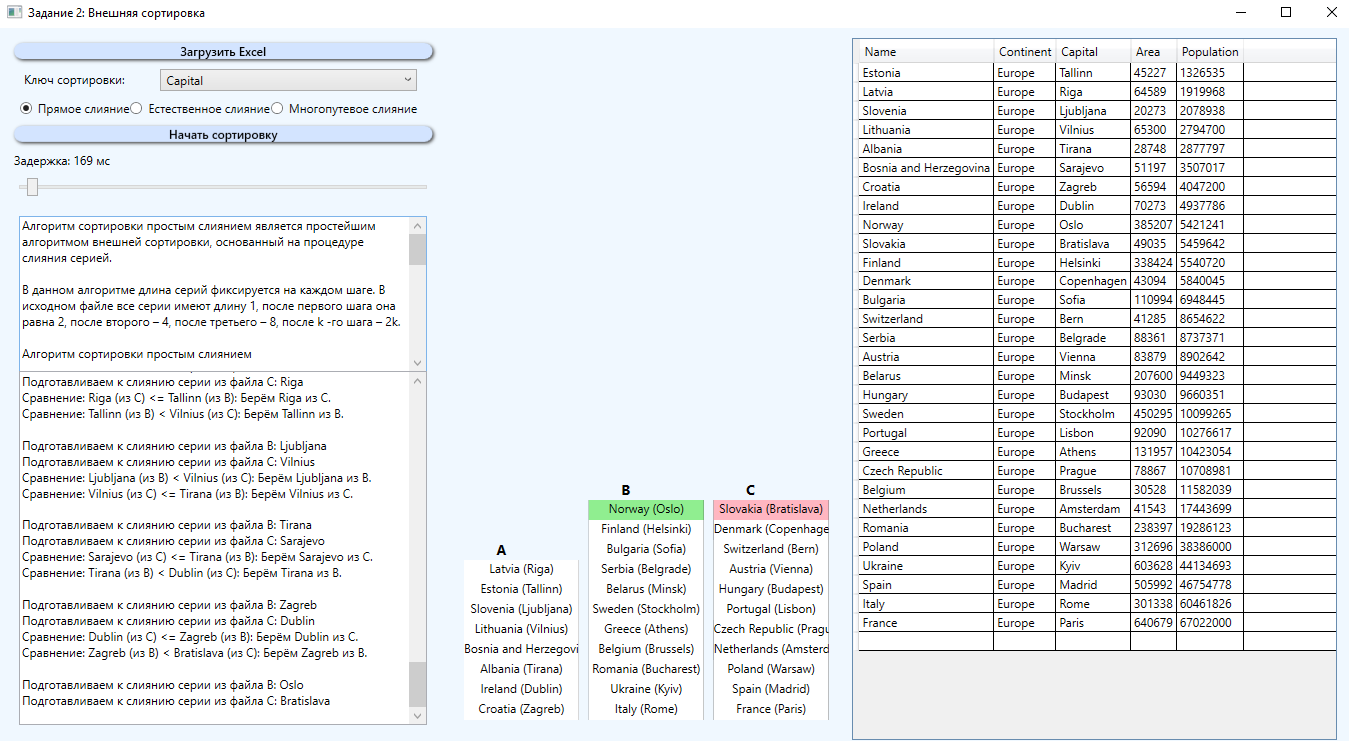


Рис.2.4 Часть 2 GUI

Задание 3

В рамках этой части были реализованы алгоритмы сортировки Modificated Buble Sort и Radix Sort.

Modificated Bubble Sort — это оптимизированная версия стандартного алгоритма сортировки пузырьком, которая позволяет завершить сортировку раньше, если массив уже отсортирован. Этот алгоритм прост в реализации и может быть эффективен для небольших массивов или частично отсортированных данных.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 3.1.1 – 3.1.2.



Рис.3.1.1 Код сортировки пузырьком.

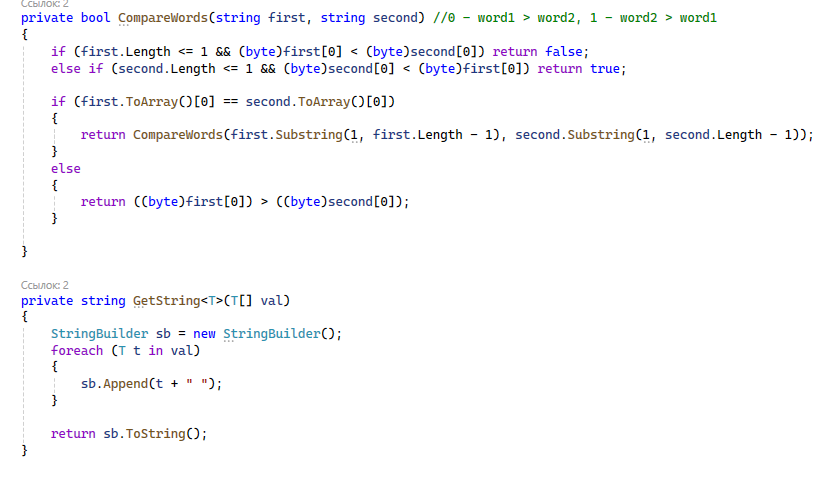


Рис.3.1.2 Код сортировки пузырьком.

Radix Sort - выполняет поразрядную сортировку, начиная с последнего символа и двигаясь к первому. Все строки выравниваются до одинаковой длины, заполняясь символом '0' (или пробелом) для корректного сравнения. На каждом этапе используется стабильная сортировка, например, CountingSort, для сортировки символов текущей позиции.

Чтобы доступно объяснить процесс сортировки, код был подвержен постоянному логированию. Так же внутри него есть множество вставок кода который отвечает за визуализацию сортировки.

Код сортировки представлен на рисунке 3.2.1 – 3.2.2.

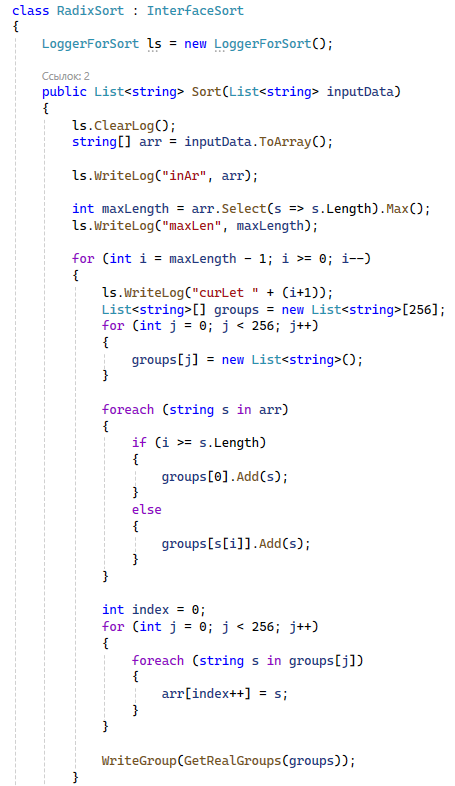


Рис.3.2.1 Код Radix Sort

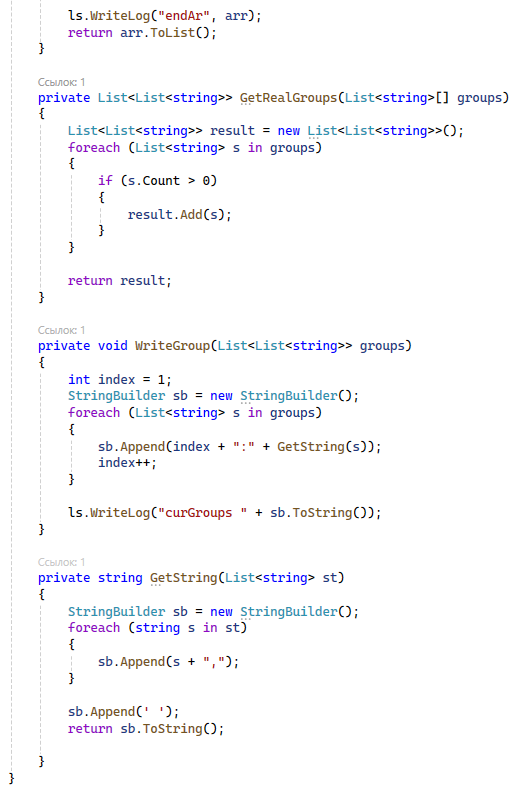


Рис.3.2.2 Код Radix Sort

Чтобы наглядно показать работу этих сортировок было создано два окна под Buble Sort (рис.3.3.1) и Radix Sort (рис.3.3.2), соответственно. В каждом окне есть возможность выгрузить файл с массивом данных с компьютера. Так же пользователь может отслеживать состояния массива прямо во время сортировки, переключая шаги

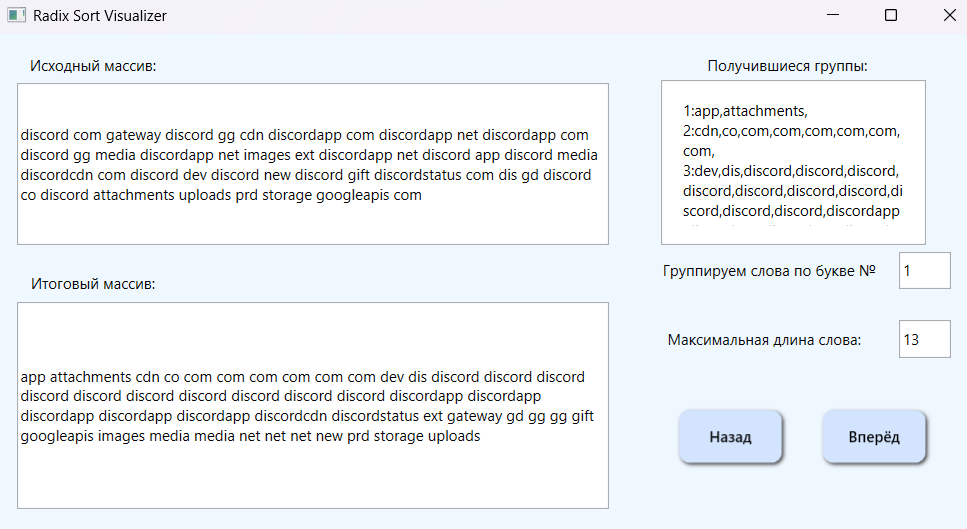


Рис.3.3.2 Визуализация Radix Sort