Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

Кафедра САПР ВС

К защите
Руководитель работы:
_____
дата, подпись

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
по дисциплине
«Структуры и алгоритмы обработки данных»
Тема:
«Сортировка слиянием»

Выполнил студент группы 045
Вашкулатов Н.А.

_____
дата сдачи на проверку, подпись

Руководитель работы
д.т.н., профессор кафедры САПР ВС
Скворцов С.В.

_____ _____
оценка дата защиты, подпись

Рязань 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Рязанский государственный радиотехнический университет

имени В.Ф. Уткина»

Кафедра САПР ВС

ЗАДАНИЕ
на курсовой проект по дисциплине
«Структуры и алгоритмы обработки данных»

Студенту Вашкулатову Никите Александровичу_____ группа 045_____

Тема: Сортировка слиянием_____

_____

Вариант № 15

Примерное содержание пояснительной записки

Введение
1. Постановка задачи
2. Описание и анализ алгоритмов решения задачи
   2.1. Описание обобщенного алгоритма
   2.2. Анализ особенностей, достоинств и недостатков
   2.3. Описание используемых структур данных
3. Разработка программного обеспечения
   3.1. Структура программы
   3.2. Основные переменные, массивы и структуры данных
   3.3. Основные подпрограммы
4. Экспериментальная часть
   4.1. Ручное решение тестовых задач
   4.2. Машинное решение тестовых задач
   4.3. Результаты экспериментальных исследований программы
5. Программная документация

Заключение
Список использованной литературы
Приложения:
- текст программы;
-листинг с результатами машинного решения

Дата выдачи задания_____ Срок сдачи_____

Руководитель _____ Скворцов С.В.

Студент _____Вашкулатов Н.А.

Вариант 15

Сортировка слиянием

Выполнить программную реализацию и анализ эффективности алгоритма сортировки слиянием [1, 2, 3]. Для оценки эффективности выполнить сравнение с прямыми методами сортировки. Получить зависимости времени работы программ (реализующих разные алгоритмы сортировки) от размера сортируемого массива.

При оформлении программной документации учитывать требования, изложенные в работе [4]

Литература

1. Левитин А. Алгоритмы: введение в разработку и анализ. М.: Издательский дом «Вильямс», 2006. 576 с.

2. Дасгупта С., Пападимитриу Х., Вазирани У. Алгоритмы. М.: МЦНМО, 2014. 320 с.

3. Кормен Т. Алгоритмы: вводный курс. Вводный курс. М.: ООО «И.Д. Вильямс», 2014. 208 с.

4. Структуры и алгоритмы обработки данных: методические указания к курсовому проектированию / Рязан. гос. радиотехн. ун-т; сост. С.В. Скворцов, В.И. Хрюкин. Рязань, 2021. 16 с. (номер в каталоге 5982)

Содержание

Введение

В мире современной разработки программного обеспечения, где огромные объемы данных становятся нормой, вопросы эффективной сортировки данных играют важную роль. Способность эффективно упорядочивать информацию становится неотъемлемой частью различных алгоритмов и приложений, где скорость обработки данных существенно влияет на общую производительность системы.

Один из популярных методов сортировки, метод слияния, предоставляет решение основанное на идее «разделяй и властвуй», обеспечивая стабильность и приемлемую сложность O(n log n). Эти характеристики делают метод слияния привлекательным выбором для эффективной сортировки данных в различных сценариях.

Сравнение метода слияния с прямыми методами сортировки, такими как пузырьковая или сортировка вставками, предоставляет ценное понимание того, какие методы могут быть наилучшими для конкретных ситуаций в реальных приложениях. В зависимости от размера и характера данных разработчики могут выбирать между различными методами для достижения оптимальной производительности.

Сортировка данных является неотъемлемой частью широкого спектра областей в реальной разработке. От управления базами данных до алгоритмов поиска, от обработки изображений до событийной обработки - эффективная сортировка данных определяет производительность и отзывчивость приложений.

1. Постановка задачи

Данная курсовая работа нацелена на исследование и анализ эффективности алгоритма слияния при помощи сравнения времени работы с прямыми методами сортировки. Задачу можно сформулировать следующим образом:

Рассмотреть алгоритм работы сортировки слиянием, построить схему алгоритма, вручную промоделировать работу алгоритма.

Разработать программу, которая должна подсчитывать и выводить время, которое занимает сортировка слиянием, вставками и выбором на примере массивов различных размеров.

Построить графики зависимости времени сортировки от размеров массива для сравниваемых алгоритмов на основе данных, полученных в программе.

Выполнить анализ полученных результатов.

Составить программную документацию.

2. Описание и анализ алгоритмов решения задачи

2.1. Описание обобщенного алгоритма

Метод декомпозиции (он же метод «разделяй и властвуй») один из наиболее популярных методов разработки алгоритмов. Ряд очень эффективных алгоритмов представляют собой реализации этой общей стратегии. Алгоритмы, основанные на методе декомпозиции работают в соответствии со следующим планом:

1. Экземпляр задачи разбивается на несколько меньших экземпляров той же задачи, в идеале – одинакового размера.
2. Решаются меньшие экземпляры задачи (обычно рекурсивно).
3. При необходимости решение исходной задачи находится путем комбинации решений меньших экземпляров.

Эффективность такого подхода определяется тем, насколько быстро мы можем (1) делить задачу на подзадачи; (2) решать подзадачи, не поддающиеся дальнейшему делению и (3) собирать решение задачи из решений подзадач.

Метод декомпозиции можно применить для сортировки массива:

- Разделим массив на две части.
- Рекурсивно отсортируем каждую из них.
- Сольем отсортированные части.

Данный алгоритм сортировки называется алгоритмом сортировки слиянием. Рассмотрим более подробно принцип его работы.

В основе этого алгоритма стоит процесс слияния двух отсортированных массивов в один отсортированный. Слияние можно выполнить следующим образом. Два указателя (индексы массивов) после инициализации указывают на первые элементы сливаемых массивов. Затем элементы, на которые указывают указатели, сравниваются, и меньший из них добавляется в новый массив. После этого индекс меньшего элемента увеличивается, и он указывает на элемент, непосредственно следующий за только что скопированным. Эта операция повторяется до тех пора, пока не будет

исчерпан один из сливаемых массивов, после чего оставшиеся элементы второго массива добавляются в конец нового массива.

Рассмотрим процесс слияния на примере в таблице 1.

Таблица 1 – Пример процесса слияния.

Номер Итерации	**Значения переменных**					
	A	**B**	**C**	**i**	**j**	**k**
1	(1,3,5,7)	(2,4)	()	0	0	0
2	(1,3,5,7)	(2,4)	(1)	1	0	1
3	(1,3,5,7)	(2,4)	(1,2)	1	1	2
4	(1,3,5,7)	(2,4)	(1,2,3)	2	1	3
5	(1,3,5,7)	(2,4)	(1,2,3,4)	2	2	4
6	(1,3,5,7)	(2,4)	(1,2,3,4,5)	3	2	5
7	(1,3,5,7)	(2,4)	(1,2,3,4,6,7)	4	2	6

В алгоритме сортировки слиянием мы используем разделение задачи на меньшие подзадачи. В данном случае мы делим массив на две части до тех пор, пока не дойдем до базового случая - отсортированного массива единичного размера, после чего выполняем слияние всех полученных массивов, начиная с базового случая.

Пусть дан неотсортированный массив A[0..n]. Тогда алгоритм сортировки слиянием можно представить следующим образом:

1) Разделение: исходный массив разделяется на left = A[0..n/2] и right = A[n/2 + 1 .. n].

2) Сортировка: сортируем каждую из половин. То есть рекурсивно повторяем шаг 1 и 2 пока размер массива больше 1.

3) Слияние: отсортированные подмассивы объединяются в новый отсортированный массив, используя процесс слияния.

2.2. Анализ особенностей, достоинств и недостатков

Определим вычислительную сложность алгоритма сортировки слиянием. Вычислительная сложность состоит из трех частей:

1. Разделение: состоит в вычислении индекса mid = n/2. O(1).
2. Два рекурсивных вызова: состоит в двух рекурсивных вызовах подзадач размера n/2. O(logn).
3. Объединение результатов двух рекурсивных вызовов. O(n).

Поскольку объединение вызывается после каждого разделения получаем вычислительную сложность всего алгоритма O(nlogn), что значительно лучше, чем у прямых методов ($O(n^2)$).

Однако алгоритм сортировки слиянием обладает значительным недостатком – для выполнения сортировки слиянием необходимо минимум n дополнительных ячеек памяти. В полученной реализации требуется $]\log_2(n)[* n$ ячеек памяти.

2.3. Описание используемых структур данных

Единственная структура данных, которая используется в алгоритме сортировки слиянием – массив. Массив – это последовательность однотипных данных в памяти, имеющее общее имя, доступ к элементам которой осуществляется по смещению относительно первого (нулевого) элемента.

Для анализа требуются классы, встроенные в стандартную библиотеку разработки на Java – Instant и Duration, при помощи которых можно измерить время и промежуток времени. Так же используется класс Random для заполнения массива случайными числами.

3. Разработка программного обеспечения

3.1. Структура программы

Процедура слияния двух массивов представлена на рисунке 1. В данной процедуре происходит слияние массивов left и right в массив array.

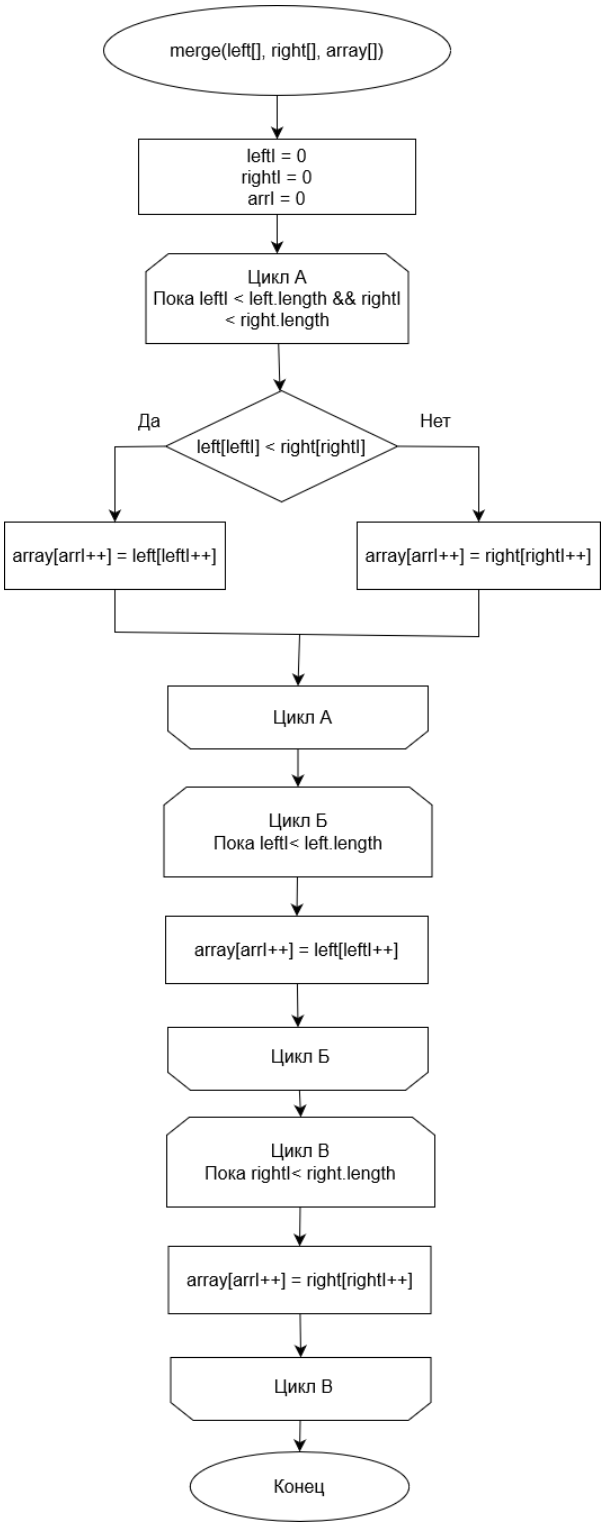

Рисунок 1 – Процедура слияния (merge)

Функция копирования массива представлена на рисунке 2. Данная функция принимает копируемый массив, начало и конец промежутка копирования. Возвращает новый массив.

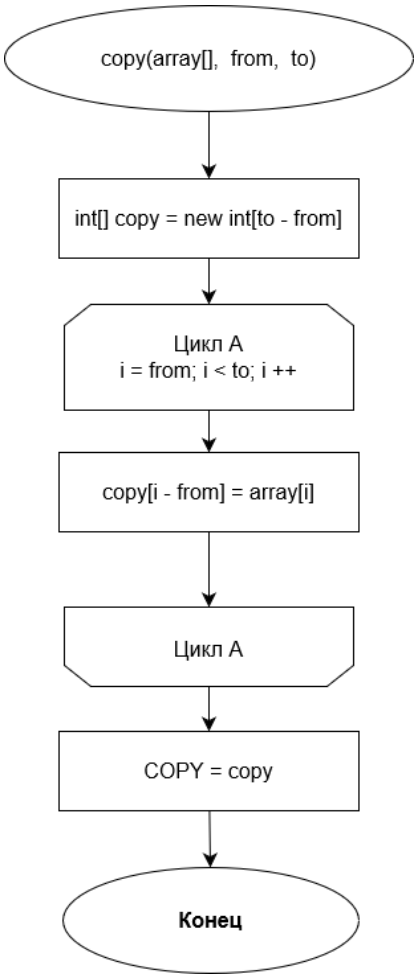

Рисунок 2 – Функция copy

Процедура сортировки слиянием показана на рисунке 3.

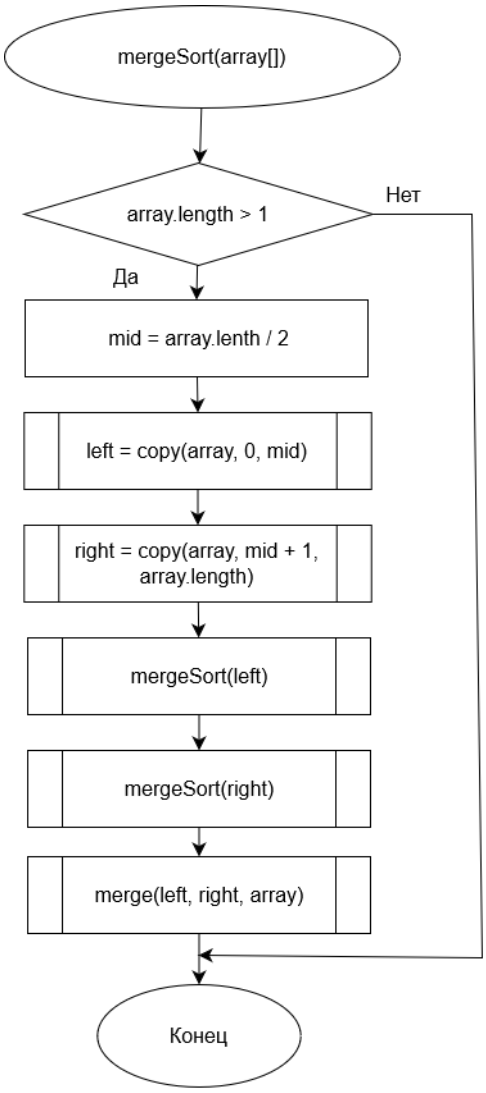

Рисунок 3 – Процедура сортировки слиянием

3.2. Основные переменные, массивы и структуры данных

При сортировке слиянием используются следующие переменные, массивы и структуры данных:

- *array* – сортируемый массив;
- *left* – левая половина сортируемого массива;
- *right* – правая половина сортируемого массива;
- **.length* – длина массива. Например array.length;
- *leftI, rigntI, arrI* – указатели на текущий элемент при слиянии массивов;
- *copy* – вспомогательный массив при копировании половины;

3.3. Основные подпрограммы

При сортировке слиянием используются следующие подпрограммы:

- `void merge(int[] left, int[] right, int[] array)` – выполняет слияние массивов *left* и *right* в массив array.
- `int[] copy(int[] array, int from, int to)` – создает и возвращает копию массива *array* начиная с *from* заканчивая *to*.
- `void mergeSort(int[] array)` – сортирует массив *array* с использованием алгоритма слияния.

При анализе используются следующие подпрограммы:

- `Instant Instant.now()` – возвращает текущее время системных часов.
- `Duration Duration.between(Instant start, Instant end)` – возвращает промежуток времени.
- `long Duration.toNanos()` – возвращает количество наносекунд из промежутка.
- `int[] createArray(int size, int min, int max)` – возвращает ссылку на массив, заполненный случайным образом с значениями от *min* до *max*.
- `void printSortTime(int arr[], int n, int count)` – выводит среднее время сортировки массива для алгоритмов сортировки вставками, выбором, слиянием. Сортируются копии массива *arr* count количество раз, ограничивая его размер до *n*

4. Экспериментальная часть

4.1. Ручное решение тестовых задач

Рассмотрим сортировку массива размером 8 элементов. Визуализация представлена на рисунке 4. Так же на рисунке приведен идентификатор массива, который разделяется или в который происходит слияние в программной реализации.

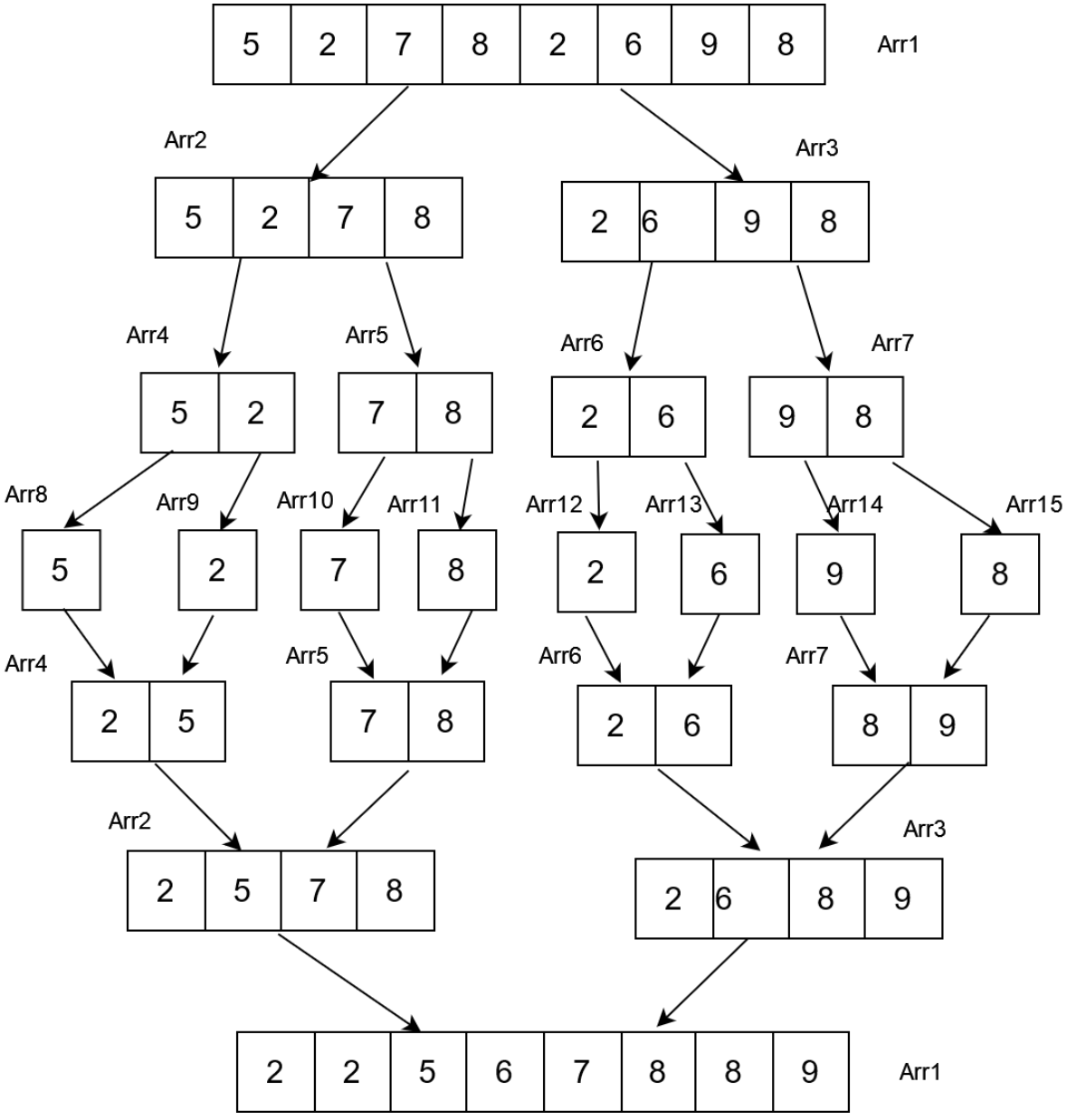

Рисунок 4 – Визуализация сортировки слиянием

4.2. Машинное решение тестовых задач

Ввод массива и вывод отсортированного массива осуществляется через консоль. Результат выполнения программы:

```
Введите размер исходного массива:
8
Введите массив:
5 2 7 8 2 6 9 8
Исходный массив:
5 2 7 8 2 6 9 8
Отсор тированный массив:
2 2 5 6 7 8 8 9
```

Сравнивая результаты ручного и машинного решения тестовой задачи, можно сделать вывод о том, что реализация верна.

4.3. Результаты экспериментальных исследований программы

Для анализа работы программы добавим процедуру, которая выводит время работы сортировки в милисекундах (рисунок 5).

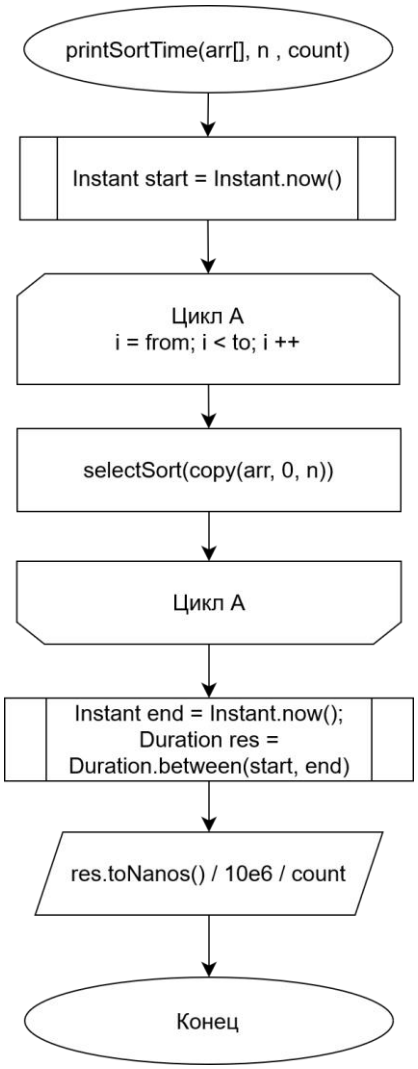

Рисунок 5 – Процедура измерения времени сортировки.

В программе есть возможность производить сортировку несколько раз и получать время одной сортировки исходя из того, сколько времени заняли все сортировки. Это необходимо для того, чтобы убрать особенности языка, на котором написана программа. Это связано с тем как работает язык Java и JVM на которой выполняется код, а конкретнее с работой JIT компилятора. Для того, чтобы время работы было стабильным необходимо «прогреть» программу.

Так же добавим процедуру для создания массива заданного размера с случайными значениями из заданного промежутка (рисунок 6).

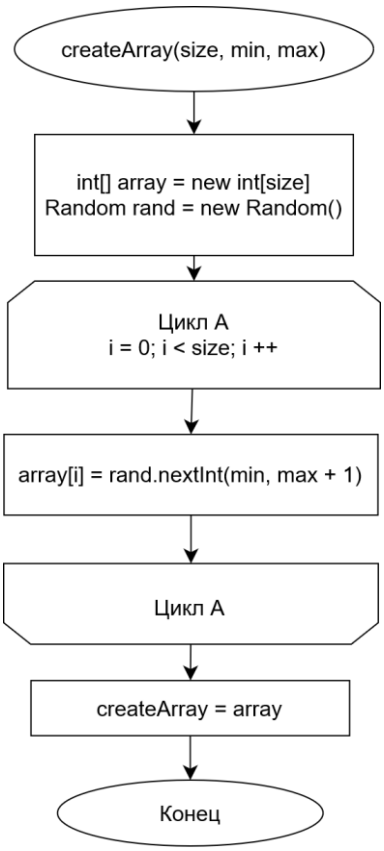

Рисунок 6 – Процедура измерения времени сортировки.

В качестве данных для анализа использовался массив с случайными данными размером от 10000 до 35000 элементов с шагом изменения размера 1000. Результаты измерений показаны в таблице 2.

Таблица 2 – Измерения времени разных сортировок для разных размеров

Размер массива	Выбором (мс)	Вставками (мс)	Слиянием(мс)
10000	2.535	0.700	0.097
11000	2.393	0.788	0.078
12000	2.841	0.925	0.092
13000	3.37	1.08	0.10
14000	3.85	1.25	0.11
15000	4.44	1.43	0.11
16000	5.00	1.63	0.12
17000	5.66	1.84	0.13
18000	6.29	2.07	0.14
19000	7.06	2.28	0.15
20000	7.77	2.56	0.16
21000	8.58	2.80	0.17
22000	9.41	3.08	0.18

Продолжение таблицы 2

Размер массива	Выбором (мс)	Вставками (мс)	Слиянием(мс)
23000	10.31	3.36	0.19
24000	11.23	3.65	0.20
25000	12.27	3.99	0.22
26000	13.10	4.28	0.23
27000	14.30	4.64	0.23
28000	15.19	4.97	0.23
29000	16.25	5.41	0.24
30000	17.35	5.76	0.25
31000	18.62	6.11	0.27
32000	19.82	6.51	0.27
33000	21.06	6.97	0.28
34000	22.34	7.37	0.29
35000	23.71	7.80	0.30

Зависимость времени сортировки от размеров массива для разных размеров представлена на рисунках 6 - 7.

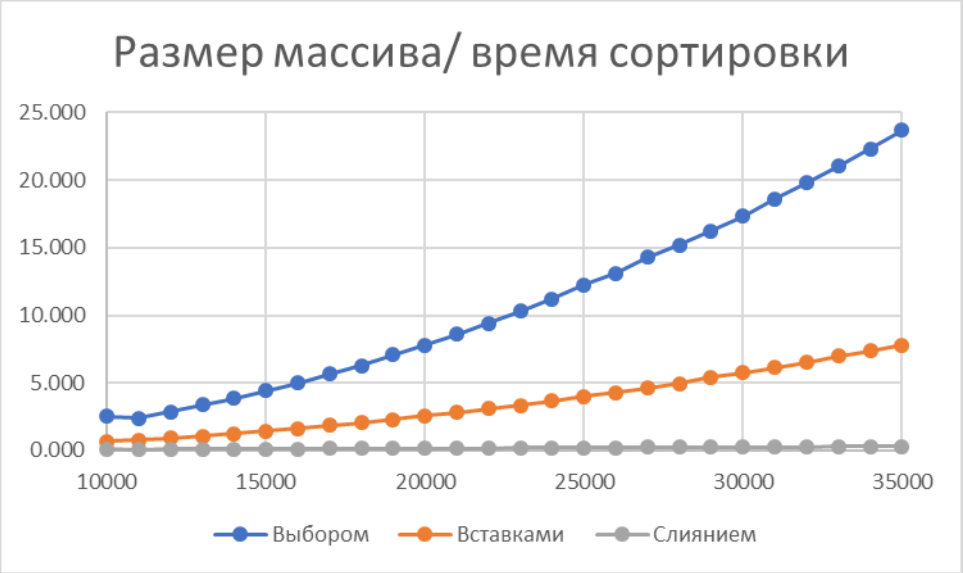

Рисунок 7 – Зависимость размера массива от времени сортировки

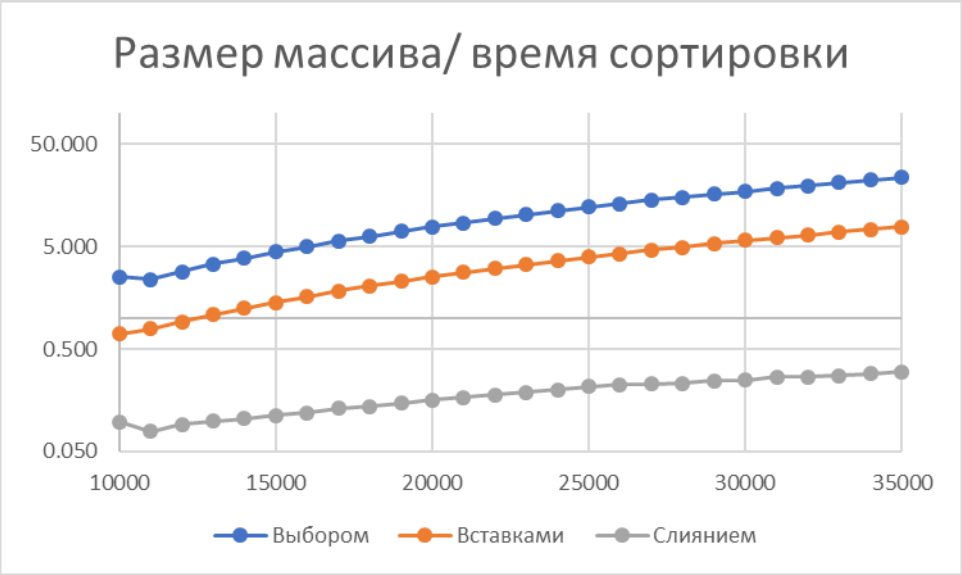

Рисунок 8 – Зависимость размера массива от времени сортировки (логарифмическая шкала времени)

Основываясь на полученных результатах можно сделать вывод, что сортировка слиянием работает значительно быстрее чем алгоритмы прямой сортировки и гораздо меньше зависит от размеров массива. Однако, в связи с затратами оперативной памяти перед использованием данного алгоритма стоит рассмотреть альтернативы, например, быструю сортировку.

Измерения производились на ПК с следующими характеристиками:

- Операционная система: Linux Manjaro
- ОЗУ: 32Гб DDR4 3600MHz
- Процессор: AMD Ryzen 7 3800X 4.1GHz

5. Программная документация

Руководство оператора.

1. Назначение программы.

Данная программа предназначена для сортировки массива при помощи алгоритма сортировки слиянием. Так же данная программа может измерять время сортировки вставками, выбором и слиянием для разных размеров массива и выводить результат в консоль.

2. Условия выполнения программы.

Для выполнения программы необходим установленный набор разработчика Java 8 с компилятором. Минимальные требования к вычислительной машине:

- Операционная система: Linux, Mac OS, Linux.
- Оперативная память: 128 МБ
- Место на диске: 200 МБ
- Процессор Pentium 2 266 МГц

Перечисленные требования являются минимальными для установки JDK 8. Скорость работы алгоритма будет увеличиваться с увеличением вычислительных характеристик машины.

3. Выполнение программы.

Для выполнения программы необходимо выполнить следующие действия:

3.1 Компиляция программы:

Необходимо открыть терминал из папки с исходным кодом программы и выполнить следующую команду:

```
javac -encoding utf8 .\Main.java
```

3.2 Использование программы:

Необходимо выполнить команду:

```
java Main
```

После чего необходимо выбрать режим работы. Введите 1 для сортировки вводимого массива. Введите любое число кроме 1 для анализа скорости сортировки.

3.2.1 Режим сортировки пользовательского массива. Необходимо ввести размер сортируемого массива:

```
Введите размер исходного массива:
8
```

Необходимо ввести массив, после чего получим отсортированный массив:

```
Введите массив:
5 2 6 8 3 1 5 6
Исходный массив:
5 2 6 8 3 1 5 6
Отсортированный массив:
1 2 3 5 5 6 6 8
```

3.2.2 Режим анализа скорости сортировки. Необходимо ввести минимальный размер массива, максимальный размер массива, шаг измерения массива, количество измерений для сортировки:

```
Анализ:
Введите минимальный размер массива:
1000
Введите максимальный размер массива:
10000
Введите шаг изменения размера массива:
1000
Введите количеств измерений для сортировки:
10
```

В консоли получим результаты измерений:

```
Размер = 1000
0.085159        0.075267        0.025055
```

…

```
Размер = 10000
2.282673        0.816694        0.115252
```

Время выводится в миллисекундах в следующем порядке: выбором, вставками, слиянием. Это сделано, чтобы можно было быстро перенести данные в таблицу.

Заключение

В ходе курсовой работы был изучен алгоритм сортировки слиянием, который является алгоритмом сортировки с сложностью O(nlogn) и является неплохой альтернативой прямым методам сортировки при размерах массивов от 3000 элементов.

Однако рассмотренный алгоритм имеет значительные недостаток – объем используемой дополнительной памяти.

Для принятия решения о выборе данного алгоритма слиянием необходимо так же сравнить его с другими сортировками с сложностью O(nlogn) не только по времени, но и по памяти и, в зависимости от решаемой задачи и доступных ресурсов, принять решения относительно возможности его использования.

Список использованной литературы

1. Левитин А. Алгоритмы: введение в разработку и анализ. М.: Издательский дом «Вильямс», 2006. 576 с.

2. Дасгупта С., Пападимитриу Х., Вазирани У. Алгоритмы. М.: МЦНМО, 2014. 320 с.

3. Кормен Т. Алгоритмы: вводный курс. Вводный курс. М.: ООО «И.Д. Вильямс», 2014. 208 с.

4. Структуры и алгоритмы обработки данных: методические указания к курсовому проектированию / Рязан. гос. радиотехн. ун-т; сост. С.В. Скворцов, В.И. Хрюкин. Рязань, 2021. 16 с. (номер в каталоге 5982)

5. Эккель Б. Философия Java, 4-е изд. – Спб.: Питер, 2021. – 1168с.

Приложение А – текст программы

```
import java.time.Duration;
import java.time.Instant;
import java.util.Random;
import java.util.Scanner;

public class Main {

    public static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }



    public static int[] selectSort(int[] array) {
        int min;
        for (int i = 0; i < array.length - 1; i++) {
            min = i;
            for (int j = i + 1; j < array.length; j++) {
                if (array[j] < array[min])
                    min = j;
            }
            swap(array, i, min);
        }
        return array;
    }

    public static int[] insertionSort(int[] array) {
        for (int i = 1; i < array.length; i++) {
            if (array[i - 1] > array[i]) {
                int key = array[i];
                int j = i - 1;
                while (j >= 0 && array[j] > key) {
                    array[j + 1] = array[j];
                    j--;
                }
                array[j + 1] = key;
            }
        }
        return array;
    }
```

```
public static void mergeSort(int[] array) {
    if (array.length > 1) {
        int mid = array.length / 2;
        int[] left = copy(array, 0, mid);
        int[] right = copy(array, mid, array.length);
        mergeSort(left);
        mergeSort(right);
        merge(left, right, array);
    }
}

public static int[] copy(int[] array, int from, int to) {
    int[] copy = new int[to - from];
    for (int i = from; i < to; i++) {
        copy[i - from] = array[i];
    }
    return copy;
}

private static void merge(int[] left, int[] right, int[] array) {
    int leftI = 0;
    int rightI = 0;
    int arrI = 0;
    while (leftI < left.length && rightI < right.length) {
        if (left[leftI] < right[rightI])
            array[arrI++] = left[leftI++];
        else
            array[arrI++] = right[rightI++];
    }
    while (leftI < left.length)
        array[arrI++] = left[leftI++];

    while (rightI < right.length)
        array[arrI++] = right[rightI++];

}

private static void printSortTimes(int[] arr, int n, int count) {
    Instant start = Instant.now();
    for (int i = 0; i < count; i++) {
        selectSort(copy(arr, 0, n));
```

```
        }
        Instant end = Instant.now();
        Duration res = Duration.between(start, end);
        System.out.print(res.toNanos() / 10e3 / count + "\t");
        System.gc();

        start = Instant.now();
        for (int i = 0; i < count; i++) {
            insertionSort(copy(arr, 0, n));
        }
        end = Instant.now();
        res = Duration.between(start, end);
        System.out.print(res.toNanos() / 10e3 / count + "\t");
        System.gc();

        start = Instant.now();
        for (int i = 0; i < count; i++) {
            mergeSort(copy(arr, 0, n));
        }
        end = Instant.now();
        res = Duration.between(start, end);
        System.out.print(res.toNanos() / 10e3 / count + "\t");
        System.gc();
        System.out.println();
    }

    public static int[] createArray(int size, int min, int max) {
        int[] array = new int[size];
        Random rand = new Random();
        for (int i = 0; i < size; i++) {
            array[i] = rand.nextInt(min, max + 1);
        }
        return array;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Введите 1 для ввода массива для сортировки или
любое число для анализа: ");
        if (sc.nextInt() == 1) {
            System.out.println("Введите размер исходного массива: ");
            int size = sc.nextInt();
            System.out.println("Введите массив: ");
```

```
            int[] array = new int[size];
            for (int i = 0; i < size; i++) {
                array[i] = sc.nextInt();
            }
            System.out.println("Исходный массив:");
            for (int i = 0; i < size; i++) {
                System.out.print(array[i] + " ");
            }
            System.out.println();
            System.out.println("Отсортированный массив: ");
            mergeSort(array);
            for (int i = 0; i < size; i++) {
                System.out.print(array[i] + " ");
            }
            System.out.println();
        } else {
            System.out.println("Анализ:");

            System.out.println("Введите минимальный размер массива: ");
            int minSize = sc.nextInt();
            System.out.println("Введите максимальный размер массива: ");
            int maxSize = sc.nextInt();
            System.out.println("Введите шаг изменения размера массива: ");
            int step = sc.nextInt();
            System.out.println("Введите количеств измерений для сортировки:
");
            int count = sc.nextInt();

            int[] arr = createArray(maxSize, -100000, 100000);
            for (int i = minSize; i <= maxSize; i += step) {
                System.out.println("Размер = " + i);
                printSortTimes(arr, i, count);
            }
            sc.close();
        }

    }
}
```

Приложение Б – листинг с результатами машинного решения

```
Введите 1 для ввода массива для сортировки или любое число для анализа:
1
Введите размер исходного массива:
8
Введите массив:
10 9 8 7 6 5 4 3
Исходный массив:
10 9 8 7 6 5 4 3
Отсортированный массив:
3 4 5 6 7 8 9 10
Введите 1 для ввода массива для сортировки или любое число для анализа:
0
Анализ:
Введите минимальный размер массива:
5000
Введите максимальный размер массива:
35000
Введите шаг изменения размера массива:
5000
Введите количеств измерений для сортировки:
40
Размер = 5000
631.5125    243.79225000000002      62.4935
Размер = 10000
1987.0897499999999      816.367     111.295
Размер = 15000
4405.59325  1840.636    185.01675
Размер = 20000
7804.19725  3228.5967499999997      248.80025
Размер = 25000
12115.80725 5062.32475  334.2335
Размер = 30000
17482.6945  7179.637    565.10425
Размер = 35000
23567.10575 9779.007249999999 582.802
```