# ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ «ВОЛГОГРАДСКИЙ СОЦИАЛЬНО – ПЕДАГОГИЧЕСКИЙ КОЛЛЕДЖ»

Кафедра информационных технологий обучения  
Специальность 09.02.07 Информационные системы и программирование

**Курсовая работа**

**«Слияние и сортировка слиянием»**

**Студента группы: 21 «Д»**

Лиджиев Сяп Дорджиевич

**Специальность:** 09.02.07 «Информационные системы и

программирование»

**Руководитель:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/** Бетиров А. М.

**Работа допущена к защите:**

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(роспись руководителя)

Волгоград, 2023 г.

**Оглавление**

[**Введение**: 3](#_Toc132838756)

[**ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ** 5](#_Toc132838757)

[1.1. Основные алгоритмы слияния 5](#_Toc132838758)

[1.2. Алгоритм сортировки слиянием 5](#_Toc132838759)

[1.3. Оптимизации алгоритма сортировки слиянием 7](#_Toc132838760)

[**Вывод:** 9](#_Toc132838761)

[**ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ** 10](#_Toc132838762)

[2.1. Реализация алгоритма сортировки слиянием на языке C# 10](#_Toc132838763)

[2.2. Анализ производительности сортировки слиянием 19](#_Toc132838764)

[2.3. Расширение и оптимизация алгоритма сортировки слиянием 20](#_Toc132838765)

[**Вывод:** 21](#_Toc132838766)

[**Заключение** 22](#_Toc132838767)

[**Список литературы** 23](#_Toc132838768)

[**Приложение** 24](#_Toc132838769)

# 

# **Введение**:

Современный мир данных стал свидетелем взрывного роста объемов информации, создаваемой и накапливаемой в различных сферах жизни. Обработка и анализ таких огромных объемов данных требует эффективных алгоритмов и методов. Одной из основных операций при работе с большими наборами данных является сортировка.

Сортировка является фундаментальной операцией обработки данных и находит широкое применение во многих областях, включая базы данных, анализ данных, информационные системы и многое другое. Однако, сортировка больших объемов данных может быть сложной задачей из-за ограничений на доступ к памяти и вычислительные ресурсы.

Одним из наиболее эффективных алгоритмов сортировки больших объемов данных является алгоритм слияния и сортировки слиянием. Он основывается на принципе "разделяй и властвуй", который позволяет эффективно разбить задачу сортировки на более маленькие подзадачи и затем объединить результаты сортировки.

**Цель курсовой работы:** Изучить алгоритм слияния и сортировки слиянием, а также исследование его применимости и эффективности при обработке больших объемов данных.

**Задачи курсовой работы:**

1. Исследование алгоритма сортировки слиянием.
2. Сравнительный анализ алгоритмов сортировки, включая сортировку слиянием.
3. Применение алгоритма слияния и сортировки слиянием в реальных задачах.

**Методы исследования** включают в себя теоретическую и практическую часть.

**К теоретической части относится:**

1.Обзор алгоритма сортировки слиянием: В этом пункте следует представить общую идею и принципы работы алгоритма сортировки слиянием.

2.Анализ временной сложности алгоритма сортировки слиянием: В данном разделе следует провести анализ временной сложности алгоритма сортировки слиянием в лучшем, среднем и худшем случаях.

3.Исследование пространственной сложности алгоритма сортировки слиянием: В этом пункте проведите анализ пространственной сложности алгоритма сортировки слиянием. Определите, сколько дополнительной памяти требуется для хранения временных структур данных и стека вызовов рекурсии при выполнении алгоритма.

4.Оптимизации и вариации алгоритма сортировки слиянием: Рассмотрите различные оптимизации и вариации алгоритма сортировки слиянием. Это может включать использование буферов для минимизации операций записи на диск, оптимизацию объединения подмассивов или реализацию итеративной версии алгоритма.

5.Сравнительный анализ алгоритма сортировки слиянием с другими алгоритмами сортировки: В этом разделе проведите сравнительный анализ алгоритма сортировки слиянием с другими популярными алгоритмами сортировки, такими как быстрая сортировка (QuickSort), сортировка вставками (Insertion Sort) и сортировка пузырьком (Bubble Sort).

**К практической относятся следующее:**

1. Реализовать алгоритм сортировки слиянием на языке C#.
2. Проверить его работоспособность на различных наборах данных.
3. Измерить время выполнения сортировки на разных размерах входных массивов и построить график зависимости времени от размера массива.
4. Сравнить производительность сортировки слиянием с другими алгоритмами сортировки, такими как быстрая сортировка или сортировка вставками.

**Практическая значимость:**

Практическая значимость данной работы заключается в том, что она предоставляет инструменты и знания для решения задач, связанных с обработкой больших объемов данных, поиском и сортировкой информации, а также хранением и поиском данных в базах данных и других приложениях. Алгоритм сортировки слиянием является эффективным инструментом для сортировки данных, особенно когда имеется дело с большими объемами информации. Он обеспечивает стабильность порядка элементов и обладает предсказуемой производительностью.

# **ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

# **Основные алгоритмы слияния**

Метод двух указателей (или "Two Pointer") - это один из основных алгоритмов слияния, используемых для объединения двух отсортированных массивов или списков. Он базируется на использовании двух указателей, которые последовательно просматривают элементы в обоих массивах (или списках) и выполняют сравнение для определения порядка слияния.

Описание алгоритма метода двух указателей:

1. Инициализируем два указателя, указывающих на начало каждого массива (или списка).
2. Сравниваем элементы, на которые указывают указатели. Если элемент первого массива (или списка) меньше или равен элементу второго массива (или списка), добавляем этот элемент в результирующий массив (или список) и переходим к следующему элементу в первом массиве (или списке), сдвигая первый указатель вправо.
3. Если элемент второго массива (или списка) меньше, чем элемент первого массива (или списка), добавляем этот элемент в результирующий массив (или список) и переходим к следующему элементу второго массива (или списка), сдвигая второй указатель вправо.
4. Повторяем шаги 2-3 до тех пор, пока один из указателей не достигнет конца своего массива (или списка).
5. Когда один из массивов (или списков) закончится, добавляем оставшиеся элементы из другого массива (или списка) в конец результирующего массива (или списка).
6. Получаем отсортированный массив (или список) в результате слияния.

Преимущества метода двух указателей:

* Временная сложность алгоритма составляет O(n), где n - общая длина двух массивов (или списков).
* Не требуется дополнительная память для хранения промежуточных результатов.
* Метод подходит для слияния отсортированных массивов или списков различных размеров.

Недостатки метода двух указателей:

* Алгоритм работает только с отсортированными массивами или списками.
* Метод неэффективен, если требуется слияние большого количества массивов или списков.

Метод двух указателей является простым и эффективным способом слияния двух отсортированных коллекций и находит широкое применение в задачах сортировки и поиска.

Рекурсивный подход является одним из основных алгоритмов слияния, используемых для сортировки и объединения двух отсортированных массивов или списков. Этот подход основан на принципе разделения задачи на более мелкие подзадачи и их последующем слиянии.

Описание рекурсивного подхода к слиянию:

## Разделяем исходные массивы (или списки) пополам, создавая две меньшие задачи.

## Рекурсивно применяем алгоритм слияния к каждой половине, пока не достигнем базового случая.

## Базовый случай: когда размер массива (или списка) становится равным 1, он уже отсортирован.

## Выполняем слияние отсортированных половинок в один массив (или список):

## Сравниваем элементы в начале каждой половинки.

## Добавляем меньший элемент в результирующий массив (или список) и продвигаем указатель этой половинки вперед.

## Повторяем шаг 3 до тех пор, пока все элементы не будут добавлены в результирующий массив (или список).

## Возвращаем отсортированный массив (или список) в качестве результата.

## Разделяем исходные массивы (или списки) пополам, создавая две меньшие задачи.

## Рекурсивно применяем алгоритм слияния к каждой половине, пока не достигнем базового случая.

## Базовый случай: когда размер массива (или списка) становится равным 1, он уже отсортирован.

## Выполняем слияние отсортированных половинок в один массив (или список):

## Сравниваем элементы в начале каждой половинки.

## Добавляем меньший элемент в результирующий массив (или список) и продвигаем указатель этой половинки вперед.

## Повторяем шаг 3 до тех пор, пока все элементы не будут добавлены в результирующий массив (или список).

## Возвращаем отсортированный массив (или список) в качестве результата.

## Преимущества рекурсивного подхода:

## Простота и понятность кода.

## Временная сложность алгоритма составляет O(n log n), где n - общая длина массивов (или списков).

## Позволяет решать задачу сортировки и слияния с помощью небольшого количества кода.

## Недостатки рекурсивного подхода:

## Рекурсивные вызовы могут потребовать дополнительную память для хранения промежуточных результатов, особенно при работе с большими массивами (или списками).

## В некоторых случаях рекурсивный подход может быть менее эффективным по сравнению с итеративным подходом из-за накладных расходов на вызовы функций.

## Рекурсивный подход к слиянию позволяет эффективно сортировать и объединять отсортированные массивы или списки, и он широко применяется в различных алгоритмах сортировки, таких как сортировка слиянием (Merge Sort).

# **1.2Алгоритм сортировки слиянием**

Алгоритм сортировки слиянием (Merge Sort) основан на разделяй и властвуй принципе. В кратком описании алгоритма происходит следующее:

1.Разделение: Исходный массив (или список) рекурсивно разделяется пополам, пока не достигнется базовый случай, когда размер массива (или списка) становится равным 1 или менее. Рекурсивное разделение выполняется путем вычисления середины массива (или списка) и создания двух новых подмассивов (или подсписков) из элементов слева и справа от середины.

2.Слияние: Отсортированные подмассивы (или подсписки) рекурсивно сливаются в один отсортированный массив (или список). Создается новый пустой массив (или список), который будет содержать отсортированные элементы. Элементы из двух подмассивов (или подсписков) сравниваются, и меньший элемент добавляется в новый массив (или список). Этот процесс продолжается, пока все элементы из обоих подмассивов (или подсписков) не будут добавлены в новый массив (или список). Если после завершения слияния остались непройденные элементы в одном из подмассивов (или подсписков), они просто добавляются в конец нового массива (или списка).

3.Рекурсивный вызов: Процесс разделения и слияния рекурсивно повторяется для каждой половины массива (или списка), пока не достигнется базовый случай сортировки одного элемента.

4.Получение отсортированного массива (или списка): После завершения всех рекурсивных вызовов получается отсортированный массив (или список). Алгоритм сортировки слиянием обладает преимуществами стабильности (порядок равных элементов сохраняется), эффективности для больших массивов или списков, а также гарантированной временной сложности O(n log n), где n - количество элементов в массиве (или списке).

Временная сложность алгоритма сортировки слиянием (Merge Sort) составляет O(n log n), где n - количество элементов в массиве (или списке). Это означает, что время выполнения алгоритма растет логарифмически с увеличением размера входных данных. Merge Sort является одним из самых эффективных алгоритмов сортировки для больших наборов данных.

Пространственная сложность алгоритма Merge Sort составляет O(n), где n - количество элементов в массиве (или списке). В процессе выполнения алгоритма требуется дополнительная память для создания временных массивов (или списков) при слиянии подмассивов. Однако, при правильной реализации, пространственная сложность может быть сведена к O(1) при использовании ин-плейс сортировки, где сортировка выполняется внутри исходного массива (или списка) без дополнительных временных структур данных.

Таким образом, алгоритм сортировки слиянием обладает эффективностью как по времени, так и по пространству, особенно при работе с большими объемами данных.

# **1.3. Оптимизации алгоритма сортировки слиянием**

Многопоточное выполнение (или параллельное выполнение) относится к ситуации, когда несколько потоков исполнения выполняются одновременно и независимо друг от друга. В контексте сортировки слиянием, многопоточное выполнение означает распараллеливание процесса сортировки на несколько потоков, каждый из которых работает с разными частями массива и выполняет сортировку слиянием независимо.

При многопоточном выполнении сортировки слиянием, исходный массив разделяется на подмассивы, которые могут быть сортированы параллельно. Затем отсортированные подмассивы сливаются вместе для получения итогового отсортированного массива.

Многопоточное выполнение может привести к ускорению выполнения сортировки слиянием, особенно при обработке больших объемов данных. Однако требуется правильное управление потоками и синхронизация для избежания состояний гонки и обеспечения корректности сортировки.

Использование оптимального размера блока при слиянии является важным аспектом в процессе сортировки слиянием. Оптимальный размер блока влияет на производительность и эффективность алгоритма.

При слиянии двух отсортированных подмассивов (или подсписков), размер каждого подмассива также может варьироваться. Обычно оптимальный размер блока выбирается таким образом, чтобы достичь баланса между количеством операций слияния и использованием памяти.

Существует несколько подходов и эвристик для выбора оптимального размера блока:

1.Фиксированный размер блока: В этом подходе размер блока фиксирован и предопределен заранее. Например, можно выбрать размер блока равным 16, 32, 64 и т.д. Этот подход прост в реализации, но может не всегда быть оптимальным для разных размеров входных данных.

2.Адаптивный размер блока: В этом подходе размер блока выбирается динамически на основе размера входных данных или других параметров. Например, можно использовать эвристику, которая выбирает размер блока пропорционально размеру входного массива. Более мелкие блоки могут быть использованы для маленьких массивов, а более крупные блоки - для больших массивов. Это позволяет более гибко адаптироваться к различным сценариям.

3.Экспериментальный подход: В этом подходе размер блока выбирается на основе проведенных экспериментов. Различные размеры блоков тестируются с помощью набора данных разных размеров, и производительность алгоритма с разными размерами блоков сравнивается. По результатам экспериментов можно выбрать оптимальный размер блока.

Выбор оптимального размера блока зависит от многих факторов, таких как размер входных данных, доступная память, процессорная архитектура и другие ограничения. Важно провести тестирование и анализ производительности с разными размерами блоков, чтобы найти наилучшее сочетание для конкретной задачи и окружения.

Рекурсивное слияние и вставка в отсортированный массив - это модификация алгоритма сортировки слиянием, которая позволяет эффективно вставлять элементы в уже отсортированный массив.

Описание алгоритма:

1.Базовый случай:

* Если размер массива равен 1 или меньше, то он уже считается отсортированным, и процесс завершается.

2.Рекурсивное разделение:

* Исходный массив рекурсивно разделяется пополам до достижения базового случая.

3.Рекурсивное разделение выполняется путем вычисления середины массива и создания двух новых подмассивов: левого и правого.

* Рекурсивное слияние и вставка:
* Рекурсивно сливаем и вставляем элементы в отсортированные левый и правый подмассивы.
* При слиянии и вставке элементов мы сравниваем элементы из левого и правого подмассивов и вставляем их в итоговый массив в правильном порядке.
* Если элемент из левого подмассива меньше или равен элементу из правого подмассива, мы вставляем его в итоговый массив и продвигаем указатель левого подмассива на следующий элемент.
* Если элемент из правого подмассива меньше элемента из левого подмассива, мы вставляем его в итоговый массив и продвигаем указатель правого подмассива на следующий элемент.
* Процесс продолжается, пока все элементы из левого и правого подмассивов не будут вставлены в итоговый массив.

4.Получение отсортированного массива:

* После завершения всех рекурсивных вызовов получаем отсортированный массив.

Этот алгоритм комбинирует преимущества сортировки слиянием и вставки. Рекурсивное слияние и вставка обеспечивают стабильность порядка элементов и эффективность для малых подмассивов. В результате получается оптимизированный алгоритм, который хорошо работает как для больших, так и для малых массивов.

# **Вывод:**

Алгоритм сортировки слиянием обладает высокой эффективностью, стабильностью и гарантированной временной сложностью O(n log n). Оптимизации, такие как многопоточное выполнение, выбор оптимального размера блока и рекурсивное слияние и вставка, позволяют улучшить производительность и адаптировать алгоритм для различных сценариев и объемов данных.

# **ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ**

# **2.1. Реализация алгоритма сортировки слиянием на языке C#**

Рассмотрим реализацию двух видов слияние

* Метод двух указателей;
* Рекурсивный подход;

Начнем с Метод двух указателей т.к. на мой взгляд оно является довольно простым. В этой реализации используется метод двух указателей для сортировки слиянием. Массив разделяется на две половины, которые рекурсивно сортируются. Затем отсортированные половины сливаются в итоговый отсортированный массив с помощью двух указателей. Массив tempArray используется для временного хранения элементов во время слияния.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MergeSort

{

public class TwoPointerMergeSort

{

public static void Sort(int[] arr)

{

int[] temp = new int[arr.Length];

Sort(arr, temp, 0, arr.Length - 1);

}

private static void Sort(int[] arr, int[] temp, int left, int right)

{

if (left >= right)

return;

int middle = (left + right) / 2;

Sort(arr, temp, left, middle);

Sort(arr, temp, middle + 1, right);

Merge(arr, temp, left, middle, right);

}

private static void Merge(int[] arr, int[] temp, int left, int middle, int right)

{

Array.Copy(arr, left, temp, left, right - left + 1);

int i = left;

int j = middle + 1;

int k = left;

while (i <= middle && j <= right)

{

if (temp[i] <= temp[j])

{

arr[k] = temp[i];

i++;

}

else

{

arr[k] = temp[j];

j++;

}

k++;

}

while (i <= middle)

{

arr[k] = temp[i];

i++;

k++;

}

}

}

}Плюсы алгоритма сортировки слиянием с использованием метода двух указателей:

* Устойчивость: Алгоритм сортировки слиянием является устойчивым, то есть он сохраняет относительный порядок равных элементов. Это важно, если вам требуется сохранить порядок элементов с одинаковыми значениями.
* Гарантированная сложность: Сортировка слиянием всегда работает за время O(n log n) в худшем, среднем и лучшем случаях, где n - размер массива. Это гарантированная сложность, которую алгоритм достигает за счет своей логики и разделяющего-сливающего процесса.
* Хорошая производительность на больших данных: Сортировка слиянием хорошо масштабируется и проявляет хорошую производительность на больших массивах данных.
* Понятность и простота реализации: Реализация алгоритма сортировки слиянием с использованием метода двух указателей относительно проста для понимания и реализации.

Минусы алгоритма сортировки слиянием с использованием метода двух указателей:

* Дополнительное пространство: Алгоритм требует дополнительное пространство для временного массива (tempArray в коде), размер которого равен размеру сортируемого массива. В случае больших массивов это может привести к дополнительному использованию памяти.
* Неэффективность на небольших массивах: Для небольших массивов, когда сортировка выполняется за малое количество операций, алгоритм сортировки слиянием может быть несколько менее эффективным по сравнению с более простыми алгоритмами, такими как сортировка вставками или сортировка пузырьком.

Далее рассмотрим реализацию Рекурсивный подход;

namespace MergeSort

{

public class RecursiveMergeSort

{

public static void Sort(int[] arr)

{

if (arr.Length <= 1)

return;

int middle = arr.Length / 2;

int[] left = new int[middle];

int[] right = new int[arr.Length - middle];

Array.Copy(arr, 0, left, 0, middle);

Array.Copy(arr, middle, right, 0, arr.Length - middle);

Sort(left);

Sort(right);

Merge(arr, left, right);

}

private static void Merge(int[] arr, int[] left, int[] right)

{

int i = 0, j = 0, k = 0;

while (i < left.Length && j < right.Length)

{

if (left[i] <= right[j])

{

arr[k] = left[i];

i++;

}

else

{

arr[k] = right[j];

j++;

}

k++;

}

while (i < left.Length)

{

arr[k] = left[i];

i++;

k++;

}

while (j < right.Length)

{

arr[k] = right[j];

j++;

k++;

}

}

}

}Рекурсивный подход для сортировки слиянием имеет следующие преимущества:

* Простота понимания и реализации: Рекурсивный алгоритм сортировки слиянием легко понять и реализовать. Он следует прямолинейной логике разделения массива на подмассивы и их последующего слияния.
* Устойчивость: Алгоритм сортировки слиянием является устойчивым, что означает, что он сохраняет относительный порядок равных элементов в исходном массиве. Это важно для обработки массивов, содержащих дубликаты или состоящих из структурных элементов, где порядок имеет значение.
* Гарантированная сложность времени выполнения: Рекурсивный алгоритм сортировки слиянием имеет гарантированную сложность времени выполнения O(n log n) в худшем, лучшем и среднем случаях. Это делает его эффективным для сортировки больших массивов данных.
* Масштабируемость: Алгоритм сортировки слиянием легко масштабируется для обработки больших объемов данных. Рекурсивное разделение и слияние позволяют эффективно обрабатывать массивы любого размера.
* Параллелизация: Рекурсивный алгоритм сортировки слиянием может быть легко адаптирован для выполнения в параллельных потоках или процессах. Это позволяет ускорить процесс сортировки на многоядерных системах и использовать все доступные ресурсы для улучшения производительности.

Однако рекурсивный подход также имеет некоторые недостатки:

* Дополнительная память: Рекурсивная реализация требует дополнительной памяти для создания временных подмассивов при разделении и слиянии. Это может быть проблемой при работе с очень большими массивами, особенно если доступная память ограничена.
* Стек вызовов: Рекурсивный алгоритм требует использования стека вызовов для каждого рекурсивного вызова функции. При обработке больших массивов или глубокой рекурсии это может привести к переполнению стека и вызвать ошибку переполнения стека (stack overflow).
* Медленная производительность для малых массивов: Для очень маленьких массивов рекурсивный подход может оказаться неэффективным, так как накладные расходы на рекурсию и создание временных подмассивов могут стать существенными по сравнению с самой сортировкой.

В целом, рекурсивный алгоритм сортировки слиянием является эффективным и надежным методом сортировки, который хорошо подходит для обработки больших массивов данных. Он обладает простотой реализации и гарантированной сложностью времени выполнения O(n log n).

# **2.2. Анализ производительности сортировки слиянием**

Анализ производительности сортировки слиянием с использованием рекурсивного подхода и метода двух указателей позволяет сравнить эффективность их работы. Вот общий анализ производительности обоих методов:

Рекурсивный подход:

* Время выполнения: Рекурсивный алгоритм сортировки слиянием имеет гарантированную сложность времени выполнения O(n log n) в худшем, лучшем и среднем случаях. Это делает его одним из наиболее эффективных алгоритмов сортировки для больших массивов данных.
* Память: Рекурсивный подход требует дополнительной памяти для создания временных подмассивов при разделении и слиянии. Общий объем дополнительной памяти составляет O(n), где n - размер входного массива. Это может быть проблемой при работе с очень большими массивами или при ограниченной доступной памяти.

Метод двух указателей:

* Время выполнения: Метод двух указателей также имеет сложность времени выполнения O(n log n) в худшем, лучшем и среднем случаях. Он эффективен и работает примерно на том же уровне, что и рекурсивный подход.
* Память: Метод двух указателей не требует дополнительной памяти для создания временных подмассивов. Он работает "на месте", изменяя порядок элементов непосредственно в исходном массиве. Это позволяет сэкономить память и может быть предпочтительным в случаях, когда доступная память ограничена или при работе с большими массивами.

Оба метода обладают преимуществами и недостатками, и выбор между ними зависит от конкретной ситуации и требований проекта. Рекурсивный подход более прост в реализации и понимании, он устойчив к порядку элементов и может быть легко параллелизирован для повышения производительности на многоядерных системах. Метод двух указателей экономит память и может быть предпочтительным в случаях, когда доступная память ограничена.

Важно также учитывать особенности языка программирования и оптимизации, которые можно применить к каждому методу для улучшения производительности. Кроме того, факторы, такие как размер входного массива, характеристики системы и конкретные требования к проекту, также должны быть учтены при выборе метода сортировки слиянием.

# **2.3. Расширение и оптимизация алгоритма сортировки слиянием**

Алгоритм сортировки слиянием может быть расширен и оптимизирован для повышения его производительности и эффективности. Вот некоторые расширения и оптимизации, которые можно применить:

* Использование порогового значения: Вместо рекурсивного разделения массива до размера 1 можно установить пороговое значение, при котором переключение на более эффективный алгоритм сортировки, такой как сортировка вставками или быстрая сортировка, станет более эффективным. Это особенно полезно для небольших подмассивов, где дополнительные затраты на рекурсию могут оказаться излишними.
* Использование буферного массива: Вместо создания временных подмассивов для каждого рекурсивного вызова можно выделить один буферный массив заранее и повторно использовать его при слиянии. Это позволяет избежать множественных выделений и освобождений памяти, что может улучшить производительность и сократить использование памяти.
* Параллельное выполнение: Если доступно несколько ядер или процессоров, алгоритм сортировки слиянием может быть распараллелен для ускорения выполнения. Разделение и слияние подмассивов могут быть независимо выполнены в разных потоках или процессах, что позволяет использовать доступные ресурсы более эффективно. Однако необходимо учитывать синхронизацию и управление потоками для избежания состояний гонки и обеспечения правильного порядка слияния.
* Оптимизация операций слияния: В процессе слияния можно применить оптимизации, такие как использование битовых операций, предварительное вычисление размеров подмассивов, слияние в обратном порядке и т.д. Эти оптимизации могут уменьшить количество операций и улучшить производительность.
* Испоьзование итеративного подхода: Вместо рекурсивного алгоритма сортировки слиянием можно использовать итеративный подход, который предполагает использование циклов и стека для хранения состояния сортировки. Итеративный подход может быть более эффективным по памяти и позволяет избежать глубокой рекурсии, что может быть полезно при сортировке больших массивов.

Каждая из этих оптимизаций имеет свои преимущества и недостатки, и их применение зависит от конкретных требований проекта и характеристик системы. Необходимо проводить тестирование и анализ производительности для выбора оптимальных оптимизаций и расширений для конкретной задачи.

# **Вывод:**

В этой главе мы рассмотрели были реализованы два подхода к сортировке слиянием: рекурсивный подход и метод двух указателей. Рекурсивный подход обладает простотой и понятностью кода, в то время как метод двух указателей обеспечивает лучшую производительность. Оба подхода могут быть эффективно применены в зависимости от конкретных требований проекта. Важно проводить оптимизацию и тестирование различных вариантов реализации, чтобы найти оптимальное сочетание производительности и использования памяти для конкретного случая. Сортировка слиянием является надежным и эффективным алгоритмом сортировки, и выбор конкретного подхода должен основываться на анализе требований и контекста задачи.

# **Заключение**

В заключении можно отметить, что сортировка слиянием является одним из наиболее эффективных алгоритмов сортировки. Она обладает преимуществами стабильности, предсказуемости и масштабируемости, что делает ее привлекательным выбором для различных сценариев.

При реализации сортировки слиянием можно использовать два подхода: рекурсивный и метод двух указателей. Рекурсивный подход прост в понимании и реализации, но может быть несколько менее эффективным по времени выполнения и потреблению памяти. Метод двух указателей, с другой стороны, обеспечивает линейную временную сложность и минимальное потребление памяти.

Оптимизация алгоритма сортировки слиянием может быть достигнута через выбор оптимального размера блока при слиянии, что позволяет балансировать производительность и использование памяти в зависимости от размера входных данных.

В процессе разработки и использования алгоритма сортировки слиянием важно учитывать особенности задачи, требования к производительности, доступные ресурсы и ограничения. Тестирование и анализ производительности с разными реализациями и оптимизациями позволяют найти оптимальное сочетание для конкретной задачи.

В заключении можно подчеркнуть, что сортировка слиянием является надежным и эффективным алгоритмом, который может быть применен в различных областях, где требуется сортировка данных. Он обеспечивает стабильность, предсказуемость и возможность масштабирования, что делает его ценным инструментом для разработчиков и исследователей.

# **Список литературы**

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. (2013). Алгоритмы: построение и анализ (3-е издание). Вильямс.
2. Кнут, Д. (2007). Искусство программирования, том 3: Сортировка и поиск (2-е издание). Вильямс.
3. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих (2017). Дмитрий Лапшин. Питер.
4. Карр, С. (2014). Чистый код. Создание, анализ и рефакторинг. ДМК Пресс.
5. Романенко, А. (2011). Алгоритмы и структуры данных: учебное пособие (3-е издание). БХВ-Петербург.
6. Сафонов, И. (2013). Алгоритмы. Разработка и применение (2-е издание). Диалектика.
7. Реджепов, Р. (2015). Алгоритмы и структуры данных: учебное пособие. Горячая Линия-Телеком.
8. Лабушев, В. (2005). Алгоритмы и структуры данных. Высшая школа.
9. Гасфилд, Д. (2003). Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. МЦНМО.
10. Маньяков, А., Капустина, О. (2005). Алгоритмы на C#. Символ-Плюс.

# **Приложение**

[https://github.com/Syar3005/MergeSort (github.com)](https://github.com/VolchkovIlia/AVL-B-RB-tree)-ссылка на репозиторий содержащий готовый продукт полученный в результате выполнения данной курсовой работы.