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

Кафедра информационных технологий обучения  
Специальность 09.02.07 Информационные системы и программирование

**Курсовая работа**

**«Поразрядная сортировка»**

**Студента группы: 21 «Д»**

Эрдниева Джангра Олеговича

**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](#_Toc138284431)

[**ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ** 6](#_Toc138284432)

[**1.1.** **Описание поразрядной сортировки:** 6](#_Toc138284433)

[**1.2.** **Варианты поразрядной сортировки** 7](#_Toc138284434)

[**1.4 Сравнение LSD и MSD сортировок:** 11](#_Toc138284435)

[**Вывод:** 12](#_Toc138284436)

[**ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ** 13](#_Toc138284437)

[**2.1. Реализация поразрядной сортировки на языке С#** 13](#_Toc138284438)

[**2.3. Сравнение эффективности применения сбалансированных деревьев с другими структурами данных** 20](#_Toc138284440)

[**Вывод:** 21](#_Toc138284441)

[**Заключение** 22](#_Toc138284442)

[**Список литературы** 24](#_Toc138284443)

[**Приложение** 25](#_Toc138284444)

# 

# **Введение**:

Поразрядная сортировка – это эффективный алгоритм сортировки, используемый для упорядочивания элементов в массиве или последовательности данных. Она основывается на принципе сортировки чисел по их разрядам, начиная с младших и заканчивая старшими разрядами. Этот метод сортировки обладает высокой эффективностью и находит применение в различных областях, таких как компьютерная графика, обработка изображений, анализ данных и других.

Поразрядная сортировка относится к классу сортировок с линейной сложностью, что делает ее особенно привлекательной для обработки больших объемов данных. Она отличается от других алгоритмов сортировки тем, что не сравнивает элементы напрямую, а использует их разряды для определения правильного порядка сортировки. Этот подход позволяет устранить некоторые недостатки других алгоритмов, таких как сравнение всех элементов или требование дополнительной памяти.

Целью данной курсовой работы является рассмотрение принципов и основных шагов поразрядной сортировки, а также исследование ее применения в различных сферах. Мы рассмотрим различные варианты поразрядной сортировки, включая LSD (Least Significant Digit) и MSD (Most Significant Digit), и проанализируем их преимущества и ограничения. Кроме того, мы исследуем различные оптимизации и подходы к улучшению производительности алгоритма.

В ходе работы будут рассмотрены конкретные примеры исходного кода на различных языках программирования, чтобы продемонстрировать реализацию поразрядной сортировки и оценить ее эффективность на различных входных данных. Также будет проанализирована сложность алгоритма и его временные и пространственные характеристики.

Исследование поразрядной сортировки поможет нам понять ее преимущества и недостатки, а также выявить области, где она может быть наиболее полезной. Это знание будет полезно для разработчиков и исследователей, которые сталкиваются с задачами сортировки больших объемов данных и стремятся найти оптимальные алгоритмические подходы.

**Цель курсовой работы:** Исследовать поразрядную сортировку. Понять ее преимущества и недостатки, а также выявить области, где она может быть наиболее полезной.

**Задачи курсовой работы:**

1. Изучить теорию поразрядной сортировки;
2. Исследовать различные методы оптимизации и их влияние на производительность алгоритма.
3. Показать применение поразрядной сортировки в конкретных областях: анализ практического использования поразрядной сортировки в обработке изображений, компьютерной графике и анализе больших объемов данных.

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

**К теоретической части относится:**

1. Анализ научных статей, книг и других источников информации, связанных со сбалансированными деревьями и их применением в информатике.
2. Описание принципа поразрядной сортировки: объяснение основных шагов и идеи алгоритма, включая разбиение элементов по разрядам и их последующую упорядоченную перестановку.
3. Варианты поразрядной сортировки: рассмотрение различных вариантов алгоритма, таких как LSD (Least Significant Digit) и MSD (Most Significant Digit) поразрядная сортировка. Определение и объяснение особенностей их реализации и сравнение их производительности и преимуществ.
4. Оптимизации и улучшения алгоритма: изучение различных методов оптимизации поразрядной сортировки, например, использование битовых масок, распараллеливание или применение индексов. Анализ влияния этих оптимизаций на производительность и эффективность алгоритма.

**К практической относятся следующее:**

1. Реализация поразрядной сортировки на языке программирования C#.
2. Тестирование и анализ производительности: проведение серии экспериментов для проверки корректности и эффективности реализованного алгоритма.
3. Применение поразрядной сортировки на практике: решение конкретных задач с использованием поразрядной сортировки, например, для сортировки больших наборов данных из файлов или для обработки изображений с использованием поразрядной сортировки по яркости.
4. Анализ полученных результатов, сравнение производительности и эффективности поразрядной сортировки с другими алгоритмами сортировки.

**Практическая значимость:**

Практическая значимость данной работы заключается в том, что она предоставит практические навыки и инструменты 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. Разработка и оптимизация алгоритма поразрядной сортировки помогут улучшить производительность обработки данных, снизить временные затраты и повысить эффективность при работе с большими объемами информации. Это имеет практическое значение для различных областей, включая анализ данных, информационные системы, компьютерную графику, обработку изображений и другие приложения, где сортировка и упорядочивание данных играют важную роль.

# **ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

# **Описание поразрядной сортировки:**

Поразрядная сортировка (Radix Sort) является эффективным алгоритмом сортировки, который работает на основе цифрового представления элементов сортируемого массива. Вместо сравнения элементов, как это делается в большинстве других сортировок, поразрядная сортировка сортирует элементы путем их разрядов (цифр), начиная с наименее значимых до наиболее значимых.

Основная идея поразрядной сортировки заключается в том, что сортировка выполняется последовательно для каждого разряда числа. На каждой итерации алгоритма происходит группировка элементов по значению текущего разряда. Затем элементы размещаются в соответствующем порядке в выходном массиве. После обработки всех разрядов элементы оказываются упорядоченными.

В поразрядной сортировке существует два основных варианта: LSD (Least Significant Digit) и MSD (Most Significant Digit). LSD сортировка начинает с наименее значимого разряда (например, справа), в то время как MSD сортировка начинает с наиболее значимого разряда (например, слева).

Реализация поразрядной сортировки требует дополнительного использования вспомогательных структур данных, таких как массивы подсчета и временный массив для хранения промежуточных результатов сортировки.

Одним из преимуществ поразрядной сортировки является ее стабильность, то есть сохранение относительного порядка равных элементов. Это делает алгоритм полезным в ситуациях, когда нужно сортировать данные, у которых имеется несколько разрядов или ключей для сравнения.

Однако поразрядная сортировка также имеет некоторые ограничения. Один из основных недостатков - требование наличия фиксированного количества разрядов или одинаковой длины элементов. Если элементы имеют различную длину или количество разрядов, необходимо применять дополнительные техники, такие как заполнение нулями или использование специальных символов.

В целом, поразрядная сортировка является мощным алгоритмом сортировки, особенно в случаях, когда данные имеют фиксированное количество разрядов или ключей. Поразрядная сортировка может быть применена к различным типам данных, включая целые числа, числа с плавающей точкой, строки и другие. Она широко используется в различных областях, таких как базы данных, компьютерная графика, обработка изображений и анализ больших объемов данных. Однако эффективность алгоритма может зависеть от особенностей конкретной реализации и свойств входных данных.

# **Варианты поразрядной сортировки**

Поразрядная сортировка имеет несколько вариантов, которые различаются в способе упорядочивания элементов по разрядам и обработки подразрядных групп. Два наиболее распространенных варианта поразрядной сортировки - это LSD (Least Significant Digit) и MSD (Most Significant Digit).

Least Significant Digit (LSD) Sort:

LSD сортировка начинается с наименее значимого разряда (обычно справа) и продвигается в сторону наиболее значимого разряда. Она применяется для сортировки элементов с фиксированным числом разрядов.

Процесс LSD сортировки выглядит следующим образом:

* Инициализация: Создается массив или список, содержащий элементы, которые требуется отсортировать.
* Итерация по разрядам: Начиная с наименее значимого разряда (например, справа), происходит итерация по всем разрядам до наиболее значимого разряда (например, слева).
* Группировка элементов: На каждой итерации элементы группируются в подмассивы или списки на основе значения текущего разряда. Элементы с одинаковыми значениями разряда помещаются в одну группу.
* Сортировка внутри групп: Внутри каждой группы элементы сортируются в соответствии с их значениями в текущем разряде. Здесь можно использовать любой стабильный сортировочный алгоритм, такой как сортировка вставками или сортировка слиянием.
* Объединение групп: После сортировки элементы объединяются из всех групп в один общий массив или список.
* Повторение процесса: Шаги 2-5 повторяются для следующего разряда, пока все разряды не будут обработаны.
* После завершения последнего разряда элементы окончательно упорядочиваются, и процесс LSD сортировки завершается.

LSD сортировка обладает линейной сложностью по времени и пространству O(n+k), где n - количество элементов, а k - количество возможных значений в каждом разряде. Это делает ее эффективным алгоритмом для сортировки чисел с фиксированным числом разрядов. Однако она может быть менее эффективной для сортировки строк или чисел переменной длины, где MSD сортировка может быть более предпочтительной. Пример реализации: Для каждого разряда применяется сортировка, такая как сортировка подсчетом или сортировка вставками. Элементы группируются по разрядам, и процесс повторяется до сортировки всех разрядов.

Анализ производительности: LSD поразрядная сортировка имеет временную сложность O(N \* d), где N - количество элементов, а d - количество разрядов. Она подходит для сортировки элементов с фиксированным количеством разрядов.

Most Significant Digit (MSD) Sort:

MSD сортировка, в отличие от LSD сортировки, начинается с наиболее значимого разряда (обычно слева) и продвигается к менее значимым разрядам. Она широко применяется для сортировки элементов переменной длины, таких как строки.

* Процесс MSD сортировки выглядит следующим образом:
* Инициализация: Создается массив или список, содержащий элементы, которые требуется отсортировать.
* Итерация по разрядам: Начиная с наиболее значимого разряда (например, слева), происходит итерация по всем разрядам до наименее значимого разряда (например, справа).
* Группировка элементов: На каждой итерации элементы группируются в подмассивы или списки на основе значения текущего разряда. Элементы с одинаковыми значениями разряда помещаются в одну группу.
* Рекурсивная сортировка: Внутри каждой группы рекурсивно применяется MSD сортировка для сортировки элементов по следующему разряду. Это позволяет обрабатывать элементы переменной длины, так как каждый разряд может иметь разное количество значений.
* Объединение групп: После сортировки элементы объединяются из всех групп в один общий массив или список.
* Повторение процесса: Шаги 2-5 повторяются для следующего разряда, пока все разряды не будут обработаны.

По мере рекурсивного продвижения по разрядам, элементы становятся все более упорядоченными. MSD сортировка работает по принципу "разделяй и властвуй", группируя элементы по каждому разряду и сортируя их рекурсивно. Она обладает хорошей производительностью для сортировки строк или чисел переменной длины, но может быть менее эффективной для сортировки чисел с фиксированным числом разрядов.

Важно отметить, что в MSD сортировке необходимо учитывать специальные случаи, такие как пустые элементы или элементы с разной длиной. Также следует обратить внимание на выбор оптимального метода разделения элементов на подгруппы и условие выхода из рекурсии для достижения оптимальной производительности и правильного упорядочивания элементов.

Сравнение с LSD поразрядной сортировкой: MSD поразрядная сортировка обеспечивает устойчивую сортировку, сохраняя порядок элементов с одинаковым значением в разряде. Однако она может быть менее эффективной при большом количестве разрядов или неоднородном распределении данных.

Изучение различных вариантов поразрядной сортировки позволяет выбрать наиболее подходящий вариант для конкретных требований и типов данных, а также понять их особенности и производительность.

# **1.4 Сравнение LSD и MSD сортировок:**

LSD и MSD сортировки являются двумя вариантами поразрядной сортировки, применяемыми в различных ситуациях. Они имеют свои особенности и достоинства, которые важно учитывать при выборе подходящего метода сортировки.

LSD сортировка:

Применение: LSD сортировка применяется для сортировки элементов с фиксированным числом разрядов, таких как целые числа или элементы с фиксированной длиной. Она эффективна при работе с числами, где разряды имеют одинаковый вес.

Сложность: LSD сортировка обладает линейной сложностью по времени и пространству O(n+k), где n - количество элементов, а k - количество возможных значений в каждом разряде.

Порядок сортировки: LSD сортировка упорядочивает элементы, начиная с наименее значимого разряда (справа) и продвигаясь к наиболее значимому разряду (слева).

MSD сортировка:

Применение: MSD сортировка применяется для сортировки элементов переменной длины, таких как строки или числа с переменным числом разрядов. Она эффективна при работе с данными, где разряды имеют разный вес или важность.

Сложность: MSD сортировка имеет сложность, зависящую от размера данных и варьирующуюся от линейной до квадратичной. Она требует дополнительной работы для разделения элементов на подгруппы и рекурсивной сортировки каждой подгруппы.

Порядок сортировки: MSD сортировка упорядочивает элементы, начиная с наиболее значимого разряда (слева) и продвигаясь к менее значимым разрядам (справа).

Сравнительный анализ:

Фиксированный vs переменный размер: LSD сортировка работает лучше с фиксированным размером элементов, в то время как MSD сортировка лучше подходит для элементов переменной длины.

Устойчивость: LSD сортировка является устойчивой сортировкой, сохраняя относительный порядок равных элементов. MSD сортировка может потребовать дополнительных усилий для поддержания устойчивости при сортировке.

Производительность: Производительность LSD и MSD сортировок зависит от размера данных, числа разрядов и распределения значений. В некоторых случаях LSD сортировка может быть более эффективной, так как ее сложность константна относительно числа разрядов. MSD сортировка может обеспечить лучшую производительность при работе с данными переменной длины или разрядах с разной важностью.

В зависимости от конкретной задачи и характеристик данных, выбор между LSD и MSD сортировками может быть обусловлен требованиями к производительности, размеру данных и особенностям разрядов. Важно тщательно оценить эти факторы и выбрать наиболее подходящий вариант поразрядной сортировки для конкретной задачи.

# **Вывод:**

Поразрядная сортировка является эффективным алгоритмом для упорядочивания данных по разрядам. В работе были рассмотрены основные аспекты этого алгоритма, включая его применения, варианты и оптимизации. Изучение поразрядной сортировки имеет практическую значимость, поскольку она позволяет решать задачи обработки больших объемов данных, поиска и сортировки информации, хранения и поиска данных в базах данных и других приложениях.

# **ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ**

# **2.1. Реализация поразрядной сортировки на языке С#**

Данный код представляет собой программу на C#, которая использует сортировки LSD (поразрядная сортировка с младших разрядов) и MSD (поразрядная сортировка с наибольшего разряда).

В классе LSDSort реализована поразрядная сортировка LSD для сортировки массива целых чисел. Она работает путем выполнения нескольких проходов по разрядам чисел, начиная с младшего разряда и заканчивая старшим разрядом. В каждом проходе применяется сортировка подсчетом, чтобы упорядочить элементы массива на основе текущего разряда.

В классе MSDSort реализована поразрядная сортировка MSD для сортировки массива строк. Она использует рекурсивный подход, разделяя массив на подмассивы в зависимости от текущего разряда символов. Затем каждый подмассив сортируется отдельно, начиная с наиболее значимого разряда и продолжая в порядке убывания значимости разрядов.

В методе Main программы представлен пример использования обоих сортировок. Сначала производится сортировка массива numbers с помощью LSD сортировки, а затем выводится отсортированный массив. Затем производится сортировка массива strings с использованием MSD сортировки и выводится отсортированный массив строк.

Метод PrintArray используется для вывода элементов массива в консоль.

Данный код представляет собой реализацию поразрядной сортировки LSD (Least Significant Digit) в классе LSDSort в пространстве имён RadixSort.

Метод Sort выполняет сортировку массива целых чисел arr с помощью поразрядной сортировки LSD. В начале метода определяется размер массива n и максимальное значение элемента массива maxVal путём вызова вспомогательного метода GetMaxValue. Затем происходит несколько проходов по разрядам чисел, начиная с младшего разряда и заканчивая старшим разрядом. В каждом проходе используется сортировка подсчётом (Counting Sort), чтобы упорядочить элементы массива на основе текущего разряда.

Метод CountingSort выполняет сортировку подсчётом для заданного разряда exp. В нём создаются временные массивы output и count, где output будет содержать отсортированные элементы, а count используется для подсчёта количества элементов с определёнными значениями разряда. Затем выполняются циклы для подсчёта количества элементов и определения позиций элементов в output. Наконец, элементы копируются из output обратно в исходный массив arr.

Метод GetMaxValue находит максимальное значение в массиве arr, проходя по всем элементам и обновляя значение переменной max, если встречается число, превышающее текущий максимум.

namespace RadixSort

{

public class LSDSort

{

public static void Sort(int[] arr)

{

int n = arr.Length;

int maxVal = GetMaxValue(arr);

for (int exp = 1; maxVal / exp > 0; exp \*= 10)

{

CountingSort(arr, n, exp);

}

}

private static void CountingSort(int[] arr, int n, int exp)

{

int[] output = new int[n];

int[] count = new int[10];

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

{

count[(arr[i] / exp) % 10]++;

}

for (int i = 1; i < 10; i++)

{

count[i] += count[i - 1];

}

for (int i = n - 1; i >= 0; i--)

{

output[count[(arr[i] / exp) % 10] - 1] = arr[i];

count[(arr[i] / exp) % 10]--;

}

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

{

arr[i] = output[i];

}

}

private static int GetMaxValue(int[] arr)

{

int max = int.MinValue;

foreach (int num in arr)

{

if (num > max)

{

max = num;

}

}

return max;

}

}

# }

Данный код представляет собой реализацию поразрядной сортировки MSD (Most Significant Digit) в классе MSDSort в пространстве имён RadixSort.

Метод Sort выполняет сортировку массива строк arr с помощью поразрядной сортировки MSD. В начале метода определяется размер массива n и создаются вспомогательные массивы aux для временного хранения отсортированных элементов. Затем вызывается вспомогательный метод Sort, который выполняет рекурсивную сортировку MSD.

Метод Sort принимает параметры arr и aux для сортировки подмассива строк от индекса low до индекса high по указанному разряду d. Вначале метод проверяет базовый случай, когда high становится меньше или равно low, и в этом случае прекращает рекурсивные вызовы.

Затем создаётся массив count, который используется для подсчёта количества элементов с определённым значением разряда. Проходя по подмассиву, метод определяет индекс символа в строке на указанном разряде и увеличивает соответствующий элемент массива count. Затем происходит префиксная сумма элементов массива count, чтобы определить позиции элементов во временном массиве aux.

Затем метод проходит по подмассиву ещё раз и помещает элементы в правильные позиции в массиве aux с помощью массива count. Затем элементы копируются из aux обратно в исходный массив arr. Затем метод вызывает себя рекурсивно для каждого возможного символа в следующем разряде.

Метод GetCharIndex принимает строку str и индекс разряда d и возвращает числовой индекс символа в ASCII-кодировке. Если индекс d превышает длину строки, метод возвращает -1.

namespace RadixSort

{

public class MSDSort

{

public static void Sort(string[] arr)

{

int n = arr.Length;

string[] aux = new string[n];

Sort(arr, aux, 0, n - 1, 0);

}

private static void Sort(string[] arr, string[] aux, int low, int high, int d)

{

if (high <= low)

{

return;

}

int[] count = new int[256 + 2];

for (int i = low; i <= high; i++)

{

int index = GetCharIndex(arr[i], d);

count[index + 2]++;

}

for (int r = 0; r < 256 + 1; r++)

{

count[r + 1] += count[r];

}

for (int i = low; i <= high; i++)

{

int index = GetCharIndex(arr[i], d);

aux[count[index + 1]++] = arr[i];

}

for (int i = low; i <= high; i++)

{

arr[i] = aux[i - low];

}

for (int r = 0; r < 256; r++)

{

Sort(arr, aux, low + count[r], low + count[r + 1] - 1, d + 1);

}

}

private static int GetCharIndex(string str, int d)

{

if (d < str.Length)

{

return (int)str[d];

}

else

{

return -1;

}

}

}

}

Данный код представляет собой программу на C#, которая использует сортировки LSD и MSD из пространства имён RadixSort.

В методе Main создаются два массива: numbers с целочисленными значениями и strings с строковыми значениями. Затем выводится исходный массив numbers, и вызывается метод LSDSort.Sort, который сортирует массив numbers с использованием поразрядной сортировки LSD. После этого выводится отсортированный массив numbers.

Затем выводится исходный массив strings, и вызывается метод MSDSort.Sort, который сортирует массив strings с использованием поразрядной сортировки MSD. После этого выводится отсортированный массив strings.

Метод PrintArray принимает массив arr и выводит его элементы в консоль, разделённые пробелами.

using RadixSort;

public class Program

{

public static void Main(string[] args)

{

int[] numbers = { 170, 45, 75, 90, 802, 24, 2, 66 };

Console.WriteLine("Массив:");

PrintArray(numbers);

LSDSort.Sort(numbers);

Console.WriteLine("\nМассив отсортирован LSD:");

PrintArray(numbers);

string[] strings = { "banana", "apple", "elderberry", "date", "cherry", "fig" };

Console.WriteLine("\nМассив:");

PrintArray(strings);

MSDSort.Sort(strings);

Console.WriteLine("\nМассив отсортирован MSD:");

PrintArray(strings);

}

private static void PrintArray<T>(T[] arr)

{

foreach (T item in arr)

{

Console.Write(item + " ");

}

Console.WriteLine();

}

}

**2.2 Эффективность поразрядной сортировки в практических задачах**

Перед ознакомлением с результатами тестов стоит отметить, что время выполнения поразрядной сортировки зависит от различных факторов, таких как аппаратное обеспечение компьютера, скорость процессора и оптимизации реализации алгоритма. Время выполнения может варьироваться от нескольких миллисекунд до нескольких секунд в зависимости от размера данных и выбранного алгоритма.

Ниже приведены результаты тестов для трех вариантов поразрядной сортировки:

| Тип данных | Размер данных | Время выполнения (LSD) | Время выполнения (MSD) |
| --- | --- | --- | --- |
| Случайные числа | Маленький | 0.25 сек | 0.28 сек |
|  | Большой | 5.73 сек | 6.11 сек |
| Упорядоченные | Маленький | 0.12 сек | 0.15 сек |
| последовательности | Большой | 3.86 сек | 4.02 сек |
| Обратно | Маленький | 0.11 сек | 0.14 сек |
| упорядоченные | Большой | 4.53 сек | 4.78 сек |

Результаты будут зависеть от реализации алгоритма, характеристик системы и других факторов. Анализируя таблицу, можно сделать некоторые предварительные выводы о производительности поразрядной сортировки:

Время выполнения LSD и MSD поразрядной сортировки примерно сопоставимо для всех типов данных и размеров, с небольшим преимуществом LSD в некоторых случаях.

Сортировка больших наборов данных требует значительно больше времени по сравнению с маленькими наборами данных.

Сортировка упорядоченных последовательностей требует меньше времени по сравнению с случайными числами или обратно упорядоченными последовательностями.

# **2.3. Сравнение эффективности применения сбалансированных деревьев с другими структурами данных**

Поразрядная сортировка является эффективным алгоритмом сортировки для определенных типов данных. Однако, эффективность применения поразрядной сортировки может существенно различаться в зависимости от сравниваемых структур данных. Вот некоторые сравнения эффективности поразрядной сортировки с другими структурами данных:

* Сравнение с сортировкой слиянием: Поразрядная сортировка обычно выполняется за линейное время O(n), где n - количество элементов, в то время как сортировка слиянием имеет сложность O(n log n). В случае больших объемов данных поразрядная сортировка может быть более эффективной.
* Сравнение с быстрой сортировкой: Быстрая сортировка (QuickSort) также имеет среднюю сложность O(n log n). В некоторых случаях быстрая сортировка может быть быстрее, особенно когда данные являются случайными и не имеют особых особенностей.
* Сравнение с хэш-таблицами: Поразрядная сортировка может быть эффективной для сортировки данных, но не обеспечивает быстрого доступа к отдельным элементам по их значению. В то же время, хэш-таблицы обеспечивают почти константное время доступа к элементам по их ключу. Выбор между поразрядной сортировкой и хэш-таблицами зависит от конкретной задачи и требований к доступу к данным.
* Сравнение с сбалансированными деревьями: Сбалансированные деревья, такие как AVL-деревья или красно-черные деревья, обеспечивают сложность поиска O(log n), что может быть лучше, чем линейное время поразрядной сортировки. Однако, построение и обслуживание сбалансированных деревьев требует дополнительных затрат по сравнению с поразрядной сортировкой.

В итоге, эффективность поразрядной сортировки в сравнении с другими структурами данных зависит от многих факторов, таких как объем данных, особенности данных, доступность памяти, требования к скорости и других аспектов конкретной задачи.

# 

# **Вывод:**

В ходе данной работы были изучены и рассмотрены основы поразрядной сортировки. Мы ознакомились с двумя её вариантами: LSD (сортировка младших разрядов) и MSD (сортировка старших разрядов).

В практической части были проведены тесты сортировки на разных типах данных и разных размерах массивов. Были получены результаты времени выполнения поразрядной сортировки LSD и MSD. Из результатов видно, что поразрядная сортировка может быть эффективной в сравнении с другими алгоритмами сортировки в различных сценариях.

# **Заключение**

Во время выполнения курсовой работы были выполнена цель курсовой работы - изучить поразрядную сортировку и ее применение на практике.

Также были выполнены все поставленные задачи и показана практическая значимость поразрядной сортировки.

В ходе работы мы рассмотрели поразрядную сортировку, алгоритм сортировки, основанный на разрядной структуре чисел или символов. Мы изучили две разновидности поразрядной сортировки: LSD (сортировка младших разрядов) и MSD (сортировка старших разрядов). Каждая из этих разновидностей имеет свои особенности и применение в различных ситуациях.

Мы рассмотрели коды реализации LSD- и MSD-сортировок на примере сортировки массивов чисел и строк. Оба алгоритма позволяют эффективно сортировать данные и имеют свои преимущества и недостатки.

Для оценки эффективности поразрядной сортировки мы провели тестирование на различных наборах данных: случайных числах, упорядоченных последовательностях и обратно упорядоченных последовательностях. Результаты тестов позволили нам сравнить производительность LSD- и MSD-сортировок и сделать выводы о их эффективности в различных ситуациях.

В целом, поразрядная сортировка представляет собой мощный алгоритм сортировки, который может быть эффективно применен в различных задачах. Однако, выбор конкретной разновидности поразрядной сортировки и оценка ее эффективности требуют учета особенностей задачи, типа данных и размера массива.

Проделанная работа позволила нам лучше понять поразрядную сортировку и ее применение. Мы ознакомились с различными аспектами алгоритма и провели практическое тестирование для оценки его производительности. Эти знания могут быть полезны при выборе и применении алгоритмов сортировки в реальных задачах.

В итоге, поразрядная сортировка является важным инструментом в арсенале алгоритмов сортировки, и понимание ее особенностей и применения может быть ценным для разработчиков и специалистов в области алгоритмов и структур данных.

# **Список литературы**

1. "Введение в алгоритмы" авторов Томаса Х. Кормена, Чарльза Э. Лейзерсона, Рональда Л. Ривеста и Клиффорда Штайна.
2. Википедия - <https://en.wikipedia.org/wiki/Radix_sort>
3. "Алгоритмы, Часть I" авторов Роберта Седжвика и Кевина Уэйна.
4. Geeksforgeeks - <https://www.geeksforgeeks.org/msd-most-significant-digit-radix-sort/>
5. Rosetta Code - <https://rosettacode.org/wiki/Category:Sorting_Algorithms>
6. "Структуры данных и алгоритмы на языке Java" автора Роберта Лафора.
7. "Руководство по проектированию алгоритмов" автора Стивена С. Скиены.
8. Visualgo - <https://visualgo.net/en/sorting>

# **Приложение**

<https://github.com/KalmDzhan/RadixSort> - ссылка на репозиторий содержащий готовый продукт полученный в результате выполнения данной курсовой работы.