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

Кафедра информационных технологий обучения  
Специальность 09.02.07 Информационные системы и программирование

**Курсовая работа**

**«Принципы анализа алгоритмов»**

**Студента группы: 21 «Д»**

Матевосян Давида Грайровича

**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](#_Toc138245503)

[**ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ** 5](#_Toc138245504)

[**1.1 Основные принципы и концепции анализа алгоритмов** 5](#_Toc138245505)

[**1.2 Методы измерения и оценки временной и пространственной сложности алгоритмов** 5](#_Toc138245506)

[**1.3 Анализ стабильности и устойчивости алгоритмов** 7](#_Toc138245507)

[**Вывод:** 8](#_Toc138245508)

[**ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ** 9](#_Toc138245509)

[**2.1. Реализация алгоритма сортировки пузырьком:** 9](#_Toc138245510)

[**2.2 Сравнение и эффективность сортировки пузырьком с другими методами** 12](#_Toc138245511)

[**Вывод** 14](#_Toc138245512)

[**Заключение** 15](#_Toc138245513)

[**Список литературы** 16](#_Toc138245514)

# **Введение**

В настоящее время информационные технологии прочно вошли во все сферы жизни человека. Компьютерные программы используются в банковской сфере, медицине, транспорте, торговле, научных исследованиях, развлечениях и многих других областях. Одним из ключевых элементов программного обеспечения являются алгоритмы, которые определяют последовательность шагов для решения задачи.

Актуальность темы "Принципы анализа алгоритмов" обусловлена несколькими факторами. Во-первых, существует постоянная потребность в создании более эффективных алгоритмов, способных обрабатывать большие объемы данных и решать сложные задачи в кратчайшие сроки. Оптимизация алгоритмов позволяет улучшить производительность программного обеспечения, снизить затраты ресурсов и повысить удобство использования.

**Цель курсовой работы:** Изучение и анализ принципов анализа алгоритмов.

**Задачи курсовой работы:**

1. Изучить основные принципы и концепции анализа алгоритмов: для достижения этой задачи необходимо ознакомиться с основными понятиями, терминологией и принципами, связанными с анализом алгоритмов. Будут изучены различные подходы к анализу, методы оценки временной и пространственной сложности, а также понятия стабильности и устойчивости алгоритмов.
2. Исследовать методы измерения и оценки временной сложности алгоритмов: в рамках этой задачи будет проведен анализ различных методов измерения временной сложности алгоритмов. Будут изучены основные подходы к оценке временной сложности, такие как асимптотический анализ, экспериментальное сравнение, рекурсивные уравнения и др. Также будут изучены основные классы сложности алгоритмов, такие как O-нотация и О-символика.
3. Исследовать методы измерения и оценки пространственной сложности алгоритмов: в рамках данной задачи будет проведен анализ методов измерения пространственной сложности алгоритмов. Будут рассмотрены основные подходы к оценке использования памяти, такие как оценка объема используемой памяти, анализ структур данных, использование битовых операций и др. Также будет изучена связь между временной и пространственной сложностью алгоритмов.
4. Применить полученные знания на практике: для достижения этой задачи будет проведен практический анализ алгоритмов на реальных примерах. Будут рассмотрены конкретные задачи, где необходимо провести анализ алгоритмов, выбрать наиболее эффективный вариант и оценить его производительность. В работе будут рассмотрены примеры из различных областей, таких как сортировка, поиск, обработка данных и другие.

# **ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

# **1.1 Основные принципы и концепции анализа алгоритмов**

Рассмотрим основные понятия и терминологию, связанные с анализом алгоритмов. Ознакомление с этими понятиями является важной основой для понимания функционирования алгоритмов и их анализа.

Алгоритмы представляют собой последовательность инструкций, выполняемых для решения определенной задачи. Они имеют структуру, которая включает входные данные, операции и выходные результаты. Корректность алгоритма означает, что он дает правильные результаты во всех возможных случаях, при условии, что входные данные удовлетворяют определенным требованиям.

Гарантированное окончание алгоритма означает, что он завершит свою работу после конечного числа шагов. Однозначность алгоритма означает, что каждая его инструкция имеет однозначное и единственное значение и не допускает неоднозначности в интерпретации.

Различные типы алгоритмов широко применяются в программировании. Алгоритмы сортировки позволяют упорядочить элементы в заданной последовательности, алгоритмы поиска осуществляют поиск заданного элемента в наборе данных, а алгоритмы обработки данных выполняют определенные операции над данными.

Примеры известных алгоритмов включают алгоритмы сортировки, такие как сортировка пузырьком, сортировка вставками и быстрая сортировка. Также существуют алгоритмы поиска, включая линейный поиск и двоичный поиск. Кроме того, существуют алгоритмы обработки данных, такие как алгоритмы обхода графов и обработки строк.

Алгоритмы играют важную роль в программировании. Они являются ключевым инструментом разработки программных решений, позволяя решать различные задачи эффективно и оптимально. Анализ алгоритмов позволяет оценить их временную и пространственную сложность, выбрать наиболее эффективные варианты и оптимизировать работу программы.

# **1.2 Методы измерения и оценки временной и пространственной сложности алгоритмов**

В данной части курсовой работы мы рассмотрим методы измерения и оценки как временной, так и пространственной сложности алгоритмов. Понимание и оценка этих двух аспектов являются важным шагом в анализе алгоритмов и позволяют выбирать наиболее эффективные решения для конкретных задач.

1. **Временная сложность алгоритмов:**

Временная сложность алгоритма определяет количество операций, необходимых для выполнения алгоритма в зависимости от размера входных данных. Она позволяет оценить скорость работы алгоритма. Для измерения и оценки временной сложности существуют различные подходы:

* Асимптотический анализ: Один из наиболее распространенных методов оценки временной сложности. Он позволяет выразить временную сложность алгоритма с использованием математических функций, таких как O-нотация, Ω-нотация и Θ-нотация. Это позволяет сравнивать алгоритмы и определять их эффективность.
* Экспериментальное сравнение: Этот метод основан на выполнении алгоритма на реальных данных и измерении времени выполнения. Путем повторения испытаний с разными размерами входных данных можно получить эмпирические данные о временной сложности алгоритма.
* Рекурсивные уравнения: Некоторые алгоритмы могут быть определены с помощью рекурсивных уравнений, которые описывают зависимость временной сложности от размера задачи.

1. **Пространственная сложность алгоритмов:**

Пространственная сложность алгоритма определяет количество памяти, необходимой для его выполнения. Она позволяет оценить объем памяти, который будет использован алгоритмом. Для измерения и оценки пространственной сложности также используются различные методы:

* Оценка объема памяти: Этот метод основан на анализе используемых алгоритмом структур данных, массивов, переменных и других ресурсов памяти. Он позволяет определить общий объем памяти, занимаемый алгоритмом.
* Анализ структур данных: Сложность алгоритма может зависеть от выбранной структуры данных. Изучение использования памяти в различных структурах данных, таких как массивы, списки, деревья и графы, позволяет оценить пространственную сложность.
* Использование битовых операций: в некоторых случаях можно оптимизировать использование памяти, используя битовые операции и битовые поля для хранения информации более компактным образом.

Анализ временной и пространственной сложности алгоритмов позволяет выбирать оптимальные решения для конкретных задач. Знание этих методов позволяет программистам прогнозировать производительность алгоритмов, оптимизировать их работу и выбирать наиболее эффективные алгоритмы для решения различных задач.

# **1.3 Анализ стабильности и устойчивости алгоритмов**

Анализ стабильности и устойчивости алгоритмов является важной частью анализа алгоритмов. Он позволяет оценить поведение алгоритма при изменении входных данных или при наличии небольших возмущений. Эти понятия имеют значение для оценки надежности и предсказуемости работы алгоритмов в различных условиях.

Стабильность алгоритма связана с его способностью сохранять относительный порядок элементов входных данных. Если алгоритм является стабильным, то равные элементы во входных данных будут сохранять свой относительный порядок после выполнения алгоритма. Это особенно важно для алгоритмов сортировки и поиска. Анализ стабильности алгоритма включает изучение его внутренней логики и алгоритмических операций. Рассмотрение таких факторов, как обмен, сравнение и перемещение элементов, позволяет оценить, сохраняется ли порядок элементов при выполнении алгоритма.

Устойчивость алгоритма, в свою очередь, оценивает его поведение при наличии возмущений или изменении входных данных. Устойчивый алгоритм способен обрабатывать небольшие изменения во входных данных без существенных изменений в результате. Это важно в случаях, когда входные данные могут содержать шум, ошибки или иные несущественные возмущения. Анализ устойчивости алгоритма требует оценки его реакции на изменения входных данных и выявления предельных случаев, в которых алгоритм может давать неадекватный результат.

В обоих случаях, как при анализе стабильности, так и при анализе устойчивости алгоритма, важно учитывать особенности конкретных алгоритмов и их применение. Некоторые алгоритмы могут быть более устойчивыми или стабильными в определенных ситуациях, в то время как другие могут быть более чувствительными к изменениям. Эти аспекты нужно учитывать при выборе и применении алгоритмов в конкретных задачах.

# **Вывод:**

Первая глава курсовой работы "Принципы анализа алгоритмов" позволила нам ознакомиться с основными принципами, концепциями и методами анализа алгоритмов. Мы изучили понятия и терминологию, связанные с анализом алгоритмов, а также различные подходы к оценке и измерению их временной и пространственной сложности.

Кроме того, мы рассмотрели анализ стабильности и устойчивости алгоритмов. Понимание этих аспектов позволяет нам создавать надежные алгоритмы, которые способны обрабатывать различные входные данные и сохранять стабильность своей работы.

В результате изучения первой главы мы приобрели необходимые знания и инструменты для проведения анализа алгоритмов. Это позволит нам более глубоко понимать и оценивать производительность и эффективность алгоритмов при их применении в реальных задачах.

**В первой главе были выполнены следующие задачи:**

1. Изучить основные принципы и концепции анализа алгоритмов: для достижения этой задачи необходимо ознакомиться с основными понятиями, терминологией и принципами, связанными с анализом алгоритмов. Будут изучены различные подходы к анализу, методы оценки временной и пространственной сложности, а также понятия стабильности и устойчивости алгоритмов.
2. Исследовать методы измерения и оценки временной сложности алгоритмов: в рамках этой задачи будет проведен анализ различных методов измерения временной сложности алгоритмов. Будут изучены основные подходы к оценке временной сложности, такие как асимптотический анализ, экспериментальное сравнение, рекурсивные уравнения и др. Также будут изучены основные классы сложности алгоритмов, такие как O-нотация и О-символика.
3. Исследовать методы измерения и оценки пространственной сложности алгоритмов: в рамках данной задачи будет проведен анализ методов измерения пространственной сложности алгоритмов. Будут рассмотрены основные подходы к оценке использования памяти, такие как оценка объема используемой памяти, анализ структур данных, использование битовых операций и др. Также будет изучена связь между временной и пространственной сложностью алгоритмов.

# **ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ**

# **2.1. Реализация алгоритма сортировки пузырьком:**

Алгоритм сортировки пузырьком является простым и популярным методом сортировки элементов массива. Он основывается на сравнении и обмене соседних элементов до тех пор, пока весь массив не будет отсортирован. В этой подтеме мы представим код, реализующий алгоритм сортировки пузырьком.

﻿using System;

class SortingAlgorithm

{

static void Main()

{

// Создаем массив чисел для сортировки

int[] array = ReadArrayFromInput();

Console.WriteLine("Исходный массив:");

PrintArray(array);

// Вызываем алгоритм сортировки

BubbleSort(array);

Console.WriteLine("Отсортированный массив:");

PrintArray(array);

Console.ReadLine();

}

// Метод для ввода значений массива с клавиатуры

static int[] ReadArrayFromInput()

{

Console.WriteLine("Введите элементы массива, разделенные пробелом:");

string input = Console.ReadLine();

string[] numbers = input.Split(' ');

int[] array = new int[numbers.Length];

for (int i = 0; i < numbers.Length; i++)

{

array[i] = int.Parse(numbers[i]);

}

return array;

}

// Метод для печати массива

static void PrintArray(int[] array)

{

foreach (int num in array)

{

Console.Write(num + " ");

}

Console.WriteLine();

}

// Алгоритм сортировки пузырьком

static void BubbleSort(int[] array)

{

int n = array.Length;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (array[j] > array[j + 1])

{

// Обмен элементов

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

}

}

Данный код представляет собой реализацию алгоритма сортировки пузырьком на языке программирования C#. Рассмотрим его работу более подробно:

1. В методе Main():

* Создается массив array для сортировки путем вызова метода ReadArrayFromInput().
* Выводится сообщение "Исходный массив:".
* Вызывается метод PrintArray() для вывода исходного массива на экран.
* Вызывается алгоритм сортировки пузырьком BubbleSort().
* Выводится сообщение "Отсортированный массив:".
* Вызывается метод PrintArray() для вывода отсортированного массива на экран.

1. Метод ReadArrayFromInput():

* Пользователю предлагается ввести элементы массива, разделенные пробелом.
* Введенная строка input разделяется на отдельные числа с помощью метода Split(), и сохраняется в массив строк numbers.
* Создается массив array с длиной, равной количеству введенных чисел.
* Циклом for каждое число из массива строк numbers преобразуется в целое число и сохраняется в массив array.
* Метод возвращает введенный пользователем массив чисел.

1. Метод PrintArray():

* Выводит на экран каждый элемент массива, разделенный пробелом, с помощью цикла foreach.

1. Метод BubbleSort():

* Алгоритм сортировки пузырьком.
* Инициализируется переменная n с длиной массива array.
* Внешний цикл for выполняется n-1 раз, чтобы обеспечить сравнение всех пар элементов.
* Внутренний цикл for проходит от первого элемента до n-i-1, где i - номер текущей итерации внешнего цикла.
* Если текущий элемент больше следующего, выполняется обмен значений между ними.
* Таким образом, на каждой итерации внешнего цикла наибольший элемент "всплывает" к концу массива.
* По завершении алгоритма массив оказывается отсортированным по возрастанию.

# **2.2 Сравнение и эффективность сортировки пузырьком с другими методами**

Алгоритм сортировки пузырьком, хотя прост в реализации, имеет квадратичную временную сложность, то есть его время выполнения зависит от квадрата размера входного массива. Это означает, что с увеличением размера массива время выполнения алгоритма сортировки пузырьком значительно увеличивается. Также стоит отметить, что этот алгоритм неэффективен для уже отсортированных или почти отсортированных массивов.

В данной подтеме мы будем проводить сравнительный анализ алгоритма сортировки пузырьком с другими методами сортировки, а также анализировать их временную сложность и эффективность для разных наборов данных.

1. **Сортировка пузырьком:**

* Алгоритм сортировки пузырьком основан на последовательном сравнении и обмене соседних элементов. Он проходит по массиву несколько раз, пока все элементы не будут упорядочены.
* Временная сложность алгоритма сортировки пузырьком составляет O(n^2), где n - размер массива. Это означает, что время выполнения алгоритма будет увеличиваться квадратично с увеличением размера массива.
* Алгоритм сортировки пузырьком прост в реализации, но неэффективен для больших массивов данных и уже отсортированных или почти отсортированных массивов.

1. **Другие методы сортировки:**

* Сортировка вставками: Этот метод основан на вставке каждого элемента на своё место в уже отсортированной части массива. Временная сложность составляет O(n^2) в худшем случае, но в среднем случае и для почти отсортированных массивов может иметь линейную сложность O(n).
* Сортировка выбором: Этот метод выбирает наименьший элемент и перемещает его в начало массива. Затем он выбирает следующий наименьший элемент и перемещает его на следующую позицию, и так далее. Временная сложность составляет O(n^2), что делает его неэффективным для больших массивов данных.
* Быстрая сортировка: Этот метод основан на принципе "разделяй и властвуй". Он разбивает массив на подмассивы, сортирует их отдельно, а затем объединяет весь массив в отсортированном порядке. Временная сложность в среднем составляет O(n log n), что делает его одним из самых эффективных методов сортировки.

1. **Сравнительный анализ:**

* Для проведения сравнения алгоритма сортировки пузырьком с другими методами мы можем использовать различные метрики, такие как время выполнения, количество сравнений и обменов элементов.
* При сравнении временной сложности видно, что сортировка пузырьком имеет квадратичную сложность, в то время как другие методы, такие как быстрая сортировка, имеют более линейную или логарифмическую сложность.
* В зависимости от конкретной задачи и объема данных, выбор конкретного метода сортировки может быть обоснованным. Если нам необходима простота реализации, и мы имеем дело с небольшими массивами данных, то сортировка пузырьком может быть приемлемым выбором. Однако, для больших массивов данных, где эффективность играет решающую роль, следует обратить внимание на более эффективные методы, такие как быстрая сортировка.

# **Вывод**

Во второй главе нашей курсовой работы мы изучили и реализовали алгоритм сортировки пузырьком. Мы подробно рассмотрели этот алгоритм, который основывается на сравнении и обмене соседних элементов в массиве. Реализация алгоритма включала создание массива чисел, ввод значений с клавиатуры, вывод исходного и отсортированного массивов.

Затем мы провели сравнение и анализ эффективности сортировки пузырьком с другими методами сортировки. Были рассмотрены методы сортировки вставками, выбором и быстрая сортировка. Сравнительный анализ показал, что сортировка пузырьком имеет квадратичную временную сложность, что делает его неэффективным для больших массивов данных. В то же время, другие методы, такие как быстрая сортировка, обладают более линейной или логарифмической сложностью, что делает их более эффективными для работы с большими объемами данных.

**Во второй главе мы выполнили следующую задачу:**

1. Применить полученные знания на практике.

**Для достижения этой задачи, я выполнил следующие действия:**

Сначала я разобрал, как работает алгоритм сортировки пузырьком. Он основан на сравнении соседних элементов массива и их последующем обмене, если нужно. Процесс сортировки продолжается до тех пор, пока все элементы не будут расположены в правильном порядке.

Затем я реализовал этот алгоритм в коде, на языке C#, который позволяет ввести массив чисел с клавиатуры, выполнить сортировку пузырьком и вывести исходный и отсортированный массивы на экран.

# **Заключение**

Исследование принципов анализа алгоритмов предоставляет нам ценные знания и инструменты для разработки эффективных программных решений. Оно помогает нам лучше понимать внутреннюю структуру алгоритмов, их возможности и ограничения, а также выбирать наиболее подходящие алгоритмы для конкретных задач.

В ходе выполнения данной курсовой работы мы получили глубокое понимание основных принципов анализа алгоритмов, изучили методы измерения и оценки временной и пространственной сложности, а также овладели навыками реализации и анализа алгоритмов. Мы успешно реализовали алгоритм сортировки пузырьком и провели его сравнительный анализ с другими методами сортировки. В результате, мы приобрели ценный опыт в области анализа и разработки алгоритмов, что будет полезно для нашей дальнейшей работы и исследований в данной области.

В результате работы мы получили ценный опыт в области анализа и разработки алгоритмов, а также расширили свои знания о методах измерения и оценки сложности. Этот опыт будет полезен для нашей дальнейшей работы и исследований в области алгоритмов и их анализа.

В заключении данной курсовой работы мы успешно достигли поставленных задач и изучили основные принципы и методы анализа алгоритмов.

# **Список литературы**

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. "Алгоритмы: построение и анализ". Москва: Вильямс, 2013.
2. Седжвик, Р., Уэйн, К. "Алгоритмы на Java". Санкт-Петербург: БХВ-Петербург, 2019.
3. Грокаем алгоритмы: иллюстрированное пособие для программистов и любопытствующих. - М.: ДМК Пресс, 2018.
4. Макконнелл, С. "Анализ алгоритмов: время работы и структуры данных". Москва: Техносфера, 2018.
5. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. "Введение в алгоритмы". Москва: МЦНМО, 2009.
6. Левитин, А. "Алгоритмы: введение в разработку и анализ". Москва: Вильямс, 2006.
7. Ахо, А., Хопкрофт, Д., Ульман, Д. "Структуры данных и алгоритмы". Москва: Вильямс, 2002.
8. Зедж, Дж. "Мастерство программирования на алгоритмическом языке". Москва: Издательство "ЛКИ", 2017.
9. Росси, Ф., Вагнер, К., Вайсли, Д. "Фундаментальные алгоритмы на С++". Москва: Диалектика, 2011.
10. Уоррен, Г., Хачатуров, А. "Алгоритмы и структуры данных". Москва: БИНОМ, 2007.