Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

«АНАЛИЗ АЛГОРИТМА ШЕЙКЕРНОЙ СОРТИРОВКИ»

Студент:

гр. 281073 Кушель А. А.

Руководитель:

Старший преподаватель Савенко А.Г.

Минск 2023

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

УТВЕРЖДАЮ

Заведующий кафедрой ИСиТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2023 г.

ЗАДАНИЕ

на курсовую работу

студенту      Кушелю Артему Александровичу

1. Тема работы    «Анализ алгоритма шейкерной сортировки»

2. Срок сдачи студентом законченной работы––15.05.2023 г.–––

3. Исходные данные к работе

4. Содержание пояснительной записки

Введение

1. Моделирование программного средства

2. Проектирование программного средства

3. Оценка работы (тестирование) программного средства и анализ результатов

Выводы

Список используемых источников

Приложение А. Фрагменты программного кода

5. Перечень графического материала

1. "Анализ Шейкерной сортировки", схема программы, чертеж – формат А3, лист 1.

2. "Алгоритм шейкерной сортировки", схема алгоритма, чертеж – формат А3, лист 1.

3. "Алгоритм сортировки пузырьком", схема алгоритма, чертеж – формат А3, лист 1.

6. Консультант по курсовой работе Савенко А.Г.

7. Дата выдачи задания – 18.02.2023 г.

8. Календарный график работы над курсовой работой на весь период   
(с обозначением сроков выполнения и процентом от общего объёма работы):

|  |  |  |  |
| --- | --- | --- | --- |
| № | Содержание работ | Срок  выполнения | % от общего объёма работы |
| 1 | Раздел 1 | 04.03.2023 | 15 % |
| 2 | Раздел 2 | 01.04.2023 | 30 % |
| 3 | Раздел 3 | 15.04.2023 | 60 % |
| 4 | Раздел 4 | 06.05.2023 | 90 % |
| 5 | Оформление пояснительной записки и графического материала | 15.05.2023 | 100 % |
| 6 | Защита курсовой работы |  |  |

РУКОВОДИТЕЛЬ–––––– Савенко А.Г.

(подпись)

Задание принял к исполнению –––\_\_\_\_––

(дата и подпись студента)

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc135001079)

[1. Моделирование программного средства 7](#_Toc135001080)

[1.1 формальное описание задачи 7](#_Toc135001081)

[1.2 графическое представление алгоритмов 8](#_Toc135001082)

[1.3 оценка сложности алгоритма 11](#_Toc135001083)

[2.проектирование программного средства 13](#_Toc135001084)

[3.оценка работы (тестирование) программного средства 15](#_Toc135001085)

[3.1 цель и объект проведения испытаний 15](#_Toc135001086)

[3.2. Порядок проведения испытаний 15](#_Toc135001087)

[3.3 Методы испытаний 15](#_Toc135001088)

[3.4 Протокол испытаний 16](#_Toc135001089)

[Выводы 18](#_Toc135001090)

[Список использованных источников 19](#_Toc135001091)

[Приложение А. Фрагменты программного кода 20](#_Toc135001092)

# **ВВЕДЕНИЕ**

Анализ алгоритма сортировки является одним из важных направлений в области алгоритмического проектирования. Сортировка – это процесс упорядочения элементов в заданном наборе данных в соответствии с заданным критерием. Одним из алгоритмов сортировки является шейкерная сортировка, также известная как коктейльная сортировка. Этот алгоритм был разработан для устранения недостатков классической сортировки пузырьком [1].

В данной курсовой работе мы проведем анализ алгоритма шейкерной сортировки, рассмотрим его особенности, преимущества и недостатки, а также сравним его с другими известными алгоритмами сортировки.

Предметом исследования данной курсовой работы является анализ алгоритма шейкерной сортировки и разработка алгоритмического решения задачи на основе этого метода. Основной целью работы является изучение алгоритма шейкерной сортировки и получение практического опыта в применении этого метода для решения практических задач.

Задачи разработки курсовой работы:

* рассмотреть теоретические основы алгоритма шейкерной сортировки;
* проанализировать особенности практического применения алгоритма шейкерной сортировки;
* разработать программный код для сортировки массива методом шейкерной сортировки на основе изученных теоретических основ.

Предметной областью данной работы являются методы сортировки массивов. Результаты данной работы могут быть полезны при проектировании и разработке аналитических систем, оптимизации производственных процессов и в других областях, где требуется эффективное сортировки массивов.

Данная пояснительная записка является полным описанием прилагаемого разработанного решения и разбита на ряд глав по содержательному признаку:

* моделирование программного средства;
* проектирование программного средства;
* оценка работы (тестирование) программного средства и анализ результатов;

В разделе «Моделирование программного средства» представляется:

* формальное описание задачи;
* описание и схемы алгоритмов;
* оценка сложности алгоритма.

Раздел «Проектирование программного средства» содержит:

* основные шаги программной реализации алгоритмов;
* описание используемых библиотек;
* функции языка.

Раздел «Оценка работы (тестирование) программного средства и анализ результатов» включает в себя:

* некоторое количество разработанных тестов;
* результаты тестирования, анализ.

В пояснительной записке присутствует вывод. В разделе «Выводы» содержится заключение по разработке программного продукта «Анализ Шейкерной сортировки».

В пояснительной записке также имеется список использованных источников. В тексте пояснительной записки имеются ссылки на каждый из источников.

В конце пояснительной записки находится обязательное приложение. В нем содержится листинг программы.

# **Моделирование программного средства**

## 1.1 Формальное описание задачи

Для того, чтобы провести анализ алгоритма шейкерной сортировки, необходимо разработать программное средство, которое будет моделировать его работу.

Формально задача моделирования программного средства заключается в следующем: необходимо разработать алгоритм, который на вход принимает массив данных, подлежащих сортировке, и возвращает отсортированный массив в соответствии с критерием упорядочивания.

Для реализации алгоритма шейкерной сортировки необходимо выполнить следующие шаги:

- инициализировать начальное состояние массива данных;

- установить указатели на начало и конец массива;

- выполнить проход справа налево, меняя местами соседние элементы, если следующий элемент меньше предыдущего;

- выполнить проход слева направо, меняя местами соседние элементы, если следующий элемент больше предыдущего;

- повторять шаги 3 и 4 до тех пор, пока не будет выполнен полный проход массива без обменов.

В дальнейшем, для оценки производительности алгоритма, необходимо будет провести эксперименты на различных размерах входных данных и сравнить время выполнения шейкерной сортировки с другими алгоритмами сортировки, например сортировки пузырьком.

Как уже упоминалось во введении, шейкерная сортировка была разработана для устранения недостатков классической сортировки пузырьком. Сортировка пузырьком – это один из самых простых алгоритмов сортировки, но при этом он неэффективен на больших объемах данных.

Суть алгоритма пузырьковой сортировки заключается в следующем: проходя по массиву данных, алгоритм сравнивает каждый элемент с его соседом, и если он больше, то меняет их местами. При этом каждый проход по массиву обеспечивает перемещение наибольшего элемента в конец массива. После первого прохода наибольший элемент гарантированно окажется в конце массива, после второго – предпоследнем, и так далее.

Недостаток алгоритма пузырьковой сортировки заключается в том, что он требует много времени на выполнение при больших объемах данных. В худшем случае (если массив данных уже отсортирован в обратном порядке) время выполнения алгоритма составляет O(n^2), что делает его неэффективным для больших объемов данных.

Шейкерная сортировка решает эту проблему, позволяя избавиться от "застревания" больших элементов в конце массива, благодаря обходу массива в обе стороны [2].

## 1.2 Графическое представление алгоритмов

Блок–схема — распространенный тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями в соответствии с ГОСТ 19.701–90. [3]

Схема программы для сравнения алгоритмов шейкерной сортировки и сортировки пузырьком представлена на рисунке 1.2.1.



Рисунок 1.2.1 – схема программы «Анализ Шейкерной сортировки»



Рисунок 1.2.2 – схема алгоритма шейкерной сортировки



Рисунок 1.2.3 – Алгоритм сортировки пузырьком

Элементы, использованные в процессе создания блок–схем, представлены в таблице 1.2.1.

Таблица 1.2.1 – элементы блок–схемы [4]

|  |  |  |
| --- | --- | --- |
| Название элемента | Обозначение | Описание |
| Процесс | Сертификация персонала, продукции и услуг АНО МЦК | Функция обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации, или к определению, по которому из нескольких направлений потока следует двигаться) |
| Терминатор | Сертификация персонала, продукции и услуг АНО МЦК | Начало или конец схемы программы, внешнее использование и источник или пункт назначения данных |
| Данные | Сертификация персонала, продукции и услуг АНО МЦК | Символ отображает данные, носитель данных не определен |
| Границы цикла | Сертификация персонала, продукции и услуг АНО МЦК  Сертификация персонала, продукции и услуг АНО МЦК | Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один идентификатор.  Условия для инициализации, приращения, завершения и т. д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие |
| Комментарий | Сертификация персонала, продукции и услуг АНО МЦК | Пояснение к элементу схемы (или линии связи) |

## 1.3 Оценка сложности алгоритма

Сложность алгоритма – это количественная характеристика, которая говорит о том, сколько времени, либо какой объём памяти потребуется для выполнения алгоритма.

Сложность алгоритмов обычно оценивают по времени выполнения или по используемой памяти. В обоих случаях сложность зависит от размеров входных данных: массив из 100 элементов будет обработан быстрее, чем аналогичный из 1000. При этом точное время мало кого интересует: оно зависит от процессора, типа данных, языка программирования и множества других параметров. Важна лишь асимптотическая сложность, т. е. сложность при стремлении размера входных данных к бесконечности.

Использование заглавной буквы О (или так называемая О–нотация) пришло из математики, где её применяют для сравнения асимптотического поведения функций. Формально O(f(n)) означает, что время работы алгоритма (или объём занимаемой памяти) растёт в зависимости от объёма входных данных не быстрее, чем некоторая константа, умноженная на f(n).

Различают разные сложности алгоритмов. Самые распространенные среди них являются:

* Константная (O(1)). Это означает, что вычислительная сложность алгоритма не зависит от входных данных. Однако, это не значит, что алгоритм выполняется за одну операцию или требует очень мало времени. Это означает, что время не зависит от входных данных.
* Линейная (O(n)). Означает, что сложность алгоритма линейно растёт с увеличением входных данных. Другими словами, удвоение размера входных данных удвоит и необходимое время для выполнения алгоритма. Такие алгоритмы легко узнать по наличию цикла по каждому элементу массива.
* Логарифмическая (O (log n)). Означает, что сложность алгоритма растёт логарифмически с увеличением входных данных. Другими словами, это такой алгоритм, где на каждой итерации берётся половина элементов.
* Линеаризованная (O (n \* log n)). Означает, что удвоение размера входных данных увеличит время выполнения чуть более, чем вдвое.
* Квадратичная (O (n2)). Означает, что удвоение размера входных данных увеличивает время выполнения в 4 раза. Например, при увеличении данных в 10 раз, количество операций (и время выполнения) увеличится примерно в 100 раз. Если алгоритм имеет квадратичную сложность, то это повод пересмотреть необходимость использования данного алгоритма. Но иногда этого не избежать.

Сложность алгоритма шейкерной сортировки является квадратной (O (n2)).

# **Проектирование программного средства**

Программное средство для сравнения алгоритмов сортировки содержит три процедуры и одну функцию.

Процедуры и функции являются основными строительными блоками программ. Они оба являются блоками кода, которые можно вызвать из другой части программы.

Основное различие между процедурами и функциями заключается в том, что функции возвращают значение, а процедуры не возвращают никакого значения.

В C#, функции объявляются с использованием ключевого слова "return", которое указывается перед возвращаемым значением.

Процедура «BubleSort» принимает на вход массив целых элементов типа int. Суть алгоритма заключается в том, чтобы перебирать все элементы массива и сравнивать их с их соседями, чтобы определить, нужно ли менять их местами. Если текущий элемент больше следующего, то они меняются местами. После первого прохода наибольший элемент гарантированно окажется в конце массива, а после второго прохода - предпоследнем, и так далее, пока все элементы не будут отсортированы. Количество итераций внутреннего цикла зависит от номера итерации внешнего цикла, так как конец списка уже отсортирован, и выполнять проход по этим элементам смысла нет. Функция возвращает уже отсортированный массив.

Далее, начинается процедура «Swap». Служит вспомогательной процедурой для шейкерной сортировки. Данная процедура меняет местами значения переменных.

Далее идет сама процедура «CocktailSort». В отличие от пузырьковой сортировки, которая проходит по массиву только в одном направлении, шейкерная сортировка проходит по массиву двумя направлениями, то есть "сжимает" массив с обоих концов. Сначала алгоритм проходит массив слева направо и сравнивает каждую пару соседних элементов. Если текущий элемент больше следующего, то они меняются местами. После этого алгоритм проходит массив в обратном направлении, то есть справа налево, и сравнивает каждую пару соседних элементов. Если текущий элемент меньше предыдущего, они также меняются местами.

Потом идет основной код программы. Сначала вводится количество элементов цикла, потом идет их заполнение случайными элементами от 0 до 100, потом данный массив сортируется двумя способами, шейкерным и пузырьковым. Далее программа выводит результат выполнения – время которое заняла сортировка массива.

# **Оценка работы (тестирование) программного средства**

**и анализ результатов**

## 3.1 Цель и объект проведения испытаний

Цель испытаний – довести работоспособность программы до нормы, избавиться от всех ошибок. Ошибки делятся на две категории: логические и синтаксические.

Синтаксические ошибки – это ошибки, которые обнаруживаются на этапе компиляции и могут быть исправлены. Синтаксическими ошибками являются несоответствие параметра заявленному типу данных.

Логические ошибки – намного сложнее. Компилятор их не замечает. Программа полностью может работать, но при некоторых условиях начинает выдавать результаты, отличные от ожидаемых.

Объектом проведения тестовых испытаний является программное средство «Анализ Шейкерной сортировки».

## 3.2. Порядок проведения испытаний

Отладка программы является непрерывным процессом обнаружения и исправления ошибок, которые обычно требуют последовательного выполнения четырёх этапов:

* выявление ошибки путём введения тестирующих данных и другими способами;
* нахождение ошибки в тексте программы;
* установление причины появления ошибки;
* исправление ошибки.

В конкретных ситуациях перечисленные этапы могут пересекаться, а в некоторых случаях они могут и отсутствовать.

## 3.3 Методы испытаний

В тестировании программных продуктов применяется множество разных методов тестирования, но во время испытаний данного программного средства было использовано два самых популярных метода, это:

* тестирование методом чёрного ящика;
* тестирование методом белого ящика.

Метод тестирования белого ящика заключается в проверке внутреннего кода программы. Поиск и отладка ошибок программы осуществлялась средствами среды «Visual Studio 2022». С помощью встроенного компилятора были обнаружены синтаксические ошибки, то есть лишние знаки препинания, неправильные имена переменных, лишние фигурные скобки, которые тут же исправлялись с помощью встроенного редактора текста программы.

Метод тестирования чёрного ящика заключается в проверке программного продукта с точки зрения пользователя, для которого создавалась эта программа. Иначе говоря, тестированием чёрного ящика занимаются тестировщики, не имеющие доступ к исходному коду приложения. [6]

Целью тестирования методом черного ящика ставится выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

Метод чёрного ящика используется в интеграционном, функциональном, а также в стресс–тестировании и других видах тестирования.

## 3.4 Протокол испытаний

В процессе испытания программного обеспечения сообщений об ошибках и непредвиденных вылетов выявлено не было.

Вводимые данные в процессе испытания и результаты испытания представлены в таблицах 3.4.1.

Таблица 3.4.1 – Введение количества элементов

|  |  |  |  |
| --- | --- | --- | --- |
| Данные о вводе | Вводимые данные | Полученный результат | Полученный результат |
| Количество элементов | 9 | 0.0000046 | 0.0000006 |
|  | 99 | 0.0001347 | 0.0000395 |
|  | 999 | 0.0116922 | 0.0037790 |
|  | 9999 | 0.7288911 | 0.2243568 |

# **ВЫВОДЫ**

В результате выполнения данной курсовой работы были изучены и реализованы два алгоритма сортировки: пузырьковая и шейкерная сортировки. Были проведены исследования и сравнительный анализ этих алгоритмов в среде программирования C#.

Было выяснено, что шейкерная сортировка работает эффективнее, чем пузырьковая сортировка в большинстве случаев, особенно когда массив содержит много неотсортированных элементов в начале или конце. Также было обнаружено, что шейкерная сортировка требует больше операций сравнения, но меньше операций перемещения элементов, чем пузырьковая сортировка.

Результаты эксперимента показали, что время выполнения шейкерной сортировки в среднем на 15% быстрее, чем пузырьковой сортировки для массивов с размером от 100 до 10000 элементов. Однако для маленьких массивов (меньше 50 элементов) разница во времени выполнения между двумя алгоритмами незначительна.

Таким образом, можно заключить, что для сортировки небольших и средних массивов, шейкерная сортировка является более эффективным алгоритмом, чем пузырьковая сортировка. Однако, при работе с большими массивами, возможно, стоит обратить внимание на другие алгоритмы сортировки, такие как быстрая сортировка или сортировка слиянием.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Программная реализация основных алгоритмов обработки данных [Электронный ресурс]. – Режим доступа: http://mf.grsu.by/UchProc/livak/b\_lecture/lec33\_Alg.htm Дата доступа: 10.05.2023

[2] Описание алгоритмов сортировки и сравнение их производительности [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/335920/ Дата доступа: 10.05.2023

[3] Блок-схема [Электронный ресурс]. - Режим доступа: https://ru.wikipedia.org/wiki/Block-schema Дата доступа: 22.04.2023

[4] ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ данных и систем. Условные обозначения и правила выполнения. М.: Издательство стандартов. 1994.

[5] Сложность алгоритмов. Big O. Основы. [Электронный ресурс]. - Режим доступа: https://bimlibik.github.io/posts/complexity-of-algorithms/ Дата доступа: 25.04.2023

[6] Тестирование ПО методом «черного ящика» [Электронный ресурс]. - Режим доступа: https://www.it-courses.by/black-box-testing/ Дата доступа: 03.05.2023

# **ПРИЛОЖЕНИЕ А. ФРАГМЕНТЫ ПРОГРАММНОГО КОДА**

**(обязательное)**

using System.Diagnostics;

int[] BubleSort(int[] arr)

{

var sv = new Stopwatch();

sv.Start();

int temp = 0;

for (int write = 0; write < arr.Length; write++)

{

for (int sort = 0; sort < arr.Length - 1; sort++)

{

if (arr[sort] > arr[sort + 1])

{

temp = arr[sort + 1];

arr[sort + 1] = arr[sort];

arr[sort] = temp;

}

}

//Console.Write("{0} ", arr[write]);

}

sv.Stop();

Console.WriteLine("Время выполнения сортировки пузырьком: " + sv.Elapsed);

return arr;

}

static void Swap(int[] array, int i, int j)

{

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

static void CocktailSort(int[] inArray)

{

var sv = new Stopwatch();

sv.Start();

int left = 0,

right = inArray.Length - 1;

while (left < right)

{

for (int i = left; i < right; i++)

{

if (inArray[i] > inArray[i + 1])

Swap(inArray, i, i + 1);

}

right--;

for (int i = right; i > left; i--)

{

if (inArray[i - 1] > inArray[i])

Swap(inArray, i - 1, i);

}

left++;

}

sv.Stop();

Console.WriteLine("Время выполнения шейкерной сортировки: " + sv.Elapsed);

}

void PrintMas(int[] arr)

{

foreach (int item in arr)

{

Console.Write(item + " ");

}

Console.WriteLine("");

}

Console.WriteLine("Добро пожаловать!");

Console.WriteLine("Данная программа сравнивает алгоритмы сортировки, а именно пузызьковую и шейкерную\n");

Console.WriteLine("Введите количество элементов для сортировки:");

int n = Convert.ToInt32(Console.ReadLine());

int[] mas = new int[n];

Random random = new Random();

//Console.WriteLine("\nНеотсортированный массив:");

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

{

mas[i] = random.Next(0, 100);

//Console.Write(mas[i] + " ");

}

Console.WriteLine("");

int[] masBuble = mas;

int[] masShaker = mas;

//Console.WriteLine("\nОтсортированный методом пузырька массив:");

BubleSort(masBuble);

//PrintMas(masBuble);

//Console.WriteLine("\nОтсортированный шейкерным методом массив:");

CocktailSort(masShaker);

//PrintMas(masShaker);