МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФГБОУ ВО «Курганский государственный университет»

Кафедра «Программное обеспечение автоматизированных систем»

РФ КГУ 09.03.04.КР24.3600091 02

**КУРСОВАЯ РАБОТА**

**по дисциплине**

**«Алгоритмы и структуры данных»**

**Пояснительная записка**

**АННОТАЦИЯ**

Проектируемая программная система предназначена для анализа качества алгоритмов сортировки данных, расположенных на внешнем носителе.

Основные задачи, решаемые разработчиком в процессе выполнения курсовой работы:

1. Программная реализация алгоритма сортировки данных, расположенных на внешнем носителе.
2. Проведение оценки качества реализованного алгоритма.
3. Программная реализация экспериментального исследования алгоритма и анализ его результатов.
4. Документирование проекта в соответствии с установленными требованиями.

Программа состоит из набора программных средств. Программа включает формы, описывающие программное и информационное обеспечение оптимальной сортировки массивов, сгенерированных случайно с помощью вероятностного распределительного закона Брэдфорда, методом Timsort.

Программный комплекс реализован с использованием языка программирования Microsoft Visual C++.

# **СОДЕРЖАНИЕ**

[**СОДЕРЖАНИЕ** 3](#_Toc183953776)

[**ВВЕДЕНИЕ** 4](#_Toc183953777)

[**1 АНАЛИТИЧЕСКИЙ ОБЗОР** 5](#_Toc183953778)

[**2 ОПИСАНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ** 6](#_Toc183953779)

# **ВВЕДЕНИЕ**

Необходимость сортировки данных возникает в программировании достаточно часто. Одной из важнейших задач является поиск эффективных методов сортировки для работы с большими массивами данных. В рамках данной курсовой работы рассматривается реализация программного приложения для сортировки массива методом Timsort.

Метод Timsort является адаптивным алгоритмом сортировки, который сочетает в себе элементы сортировки слиянием и сортировки вставками. Он был разработан для эффективной работы с реальными данными, в особенности с частично отсортированными массивами.

Для достижения цели необходимо решить следующие задачи:

1. Изучить теоретическое понятие сортировки и алгоритма сортировки.
2. Ознакомиться с критериями оценки эффективности алгоритмов сортировки
3. Изучить особенности алгоритма Timsort и его применения.
4. Разработать программу, реализующую сортировку массива данных методом Timsort.
5. Провести экспериментальное исследование и анализ эффективности алгоритма на различных данных, сгенерированных с помощью датчика случайных величин.

# **1 АНАЛИТИЧЕСКИЙ ОБЗОР**

Данное приложение формализует алгоритм сортировки массива данных методом Timsort. Программное приложение разработано с использованием технологии визуального проектирования и событийного программирования. Алгоритм сортировки, использующий метод Timsort, сочетает в себе элементы сортировки вставками и слиянием, что обеспечивает его высокую эффективность при работе с различными типами данных. В программе обеспечен ввод исходных данных, ввод коэффициентов вероятностного распределительного закона, сортировка данных с использованием алгоритма Timsort и вывод отсортированных данных в форме таблицы. Также реализовано оценивание коэффициентов уравнения связи между временем выполнения программы и объемом входных данных, а также анализ этих коэффициентов для оценки эффективности работы алгоритма.

Приложение написано на языке C++ в программной среде Microsoft Visual Studio 2022 Community

# **2 ОПИСАНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ**

Алгоритм сортировки Timsort был изобретен Тимом Петерсом в 2002 году. С тех пор стал стандартным алгоритмом сортировки в таких языках программирования, как Python и Java. Идея, лежащая в основе Timsort, заключается в том, что на реальной выборке данных часто встречаются уже упорядоченные участки. Осознавая это, Тим Петерс предложил использовать комбинацию нескольких алгоритмов сортировки, что позволяет эффективно работать с уже частично отсортированными массивами.

Основной принцип работы Timsort заключается в разбиении исходного массива на run, которые затем сливаются с помощью сортировки слиянием. Эти run - это небольшие отсортированные подмассивы, которые затем объединяются в один большой отсортированный массив.

В процессе разработки Timsort было введено понятие minrun - минимального размера отсортированных подмассивов, которые будут слияться. Минимальный размер run - это число, которое определяет размер подмассива, на котором будет работать сортировка вставками. minrun рассчитывается экспериментально и выбирается таким образом, чтобы обеспечить наилучшую производительность алгоритма. Как показали эксперименты, алгоритм работает наиболее эффективно, когда размер minrun находится в диапазоне от 32 до 64. Если значение minrun слишком маленькое, то потребуется больше времени на слияние, а если слишком большое - сортировка вставками будет слишком медленной из-за своей квадратичной сложности .

Для сбалансированного слияния количество run должно быть степенью двойки. Это важно, потому что алгоритм слияния наиболее эффективен, когда подмассивы имеют приблизительно одинаковый размер. Таким образом, минимальный размер каждого run определяется через minrun, которое в среднем должно быть в диапазоне от 32 до 64. Если размер массива меньше 64, то minrun устанавливается равным размеру всего массива, и алгоритм работает как простая сортировка вставками. В нашей сортировке минимальный размер подмассива minrun равен 64. Этот размер выбран исходя из предполагаемого диапазона размеров массива от 9000 до 50 000 элементов, что является оптимальным для данного случая.

Алгоритм сортировки методом Timsort:

Шаг 1. Разбиение массива на run и их сортировка вставками

1. Алгоритм начинается с установления указателя на начало массива. Далее начинается поиск упорядоченных подмассивов, называемых run. Эти подмассивы будут сортироваться с использованием алгоритма сортировки вставками, если их размер меньше minrun.
2. Каждый подмассив будет отсортирован по возрастанию, так как сортировка вставками работает именно с этим порядком.
3. Если найденный подмассив имеет размер меньше minrun, он сортируется с помощью сортировки вставками, которая инициализирует его как упорядоченный подмассив.
4. Алгоритм продолжает разбиение массива на run, пока весь массив не будет разделен на упорядоченные подмассивы. На каждом шаге размер подмассива будет увеличиваться, и подмассивы размером minrun или больше будут сортироваться вставками.

Шаг 2. Слияние подмассивов

1. Для слияния используется модификация сортировки слиянием, где два отсортированных подмассива сливаются в один. Слияние происходит по принципу «меньшее из двух» — элементы двух подмассивов сравниваются, и минимальный элемент добавляется в итоговый массив.
2. Слияние продолжается поочередно, пока все подмассивы не будут объединены в один отсортированный массив. Алгоритм увеличивает размер сливаемых подмассивов с каждым шагом, начиная с размера minrun, затем удваивая его: 64, 128, 256 и т. д.
3. Когда все подмассивы объединены в один массив, алгоритм завершает выполнение, и результатом является отсортированный массив.

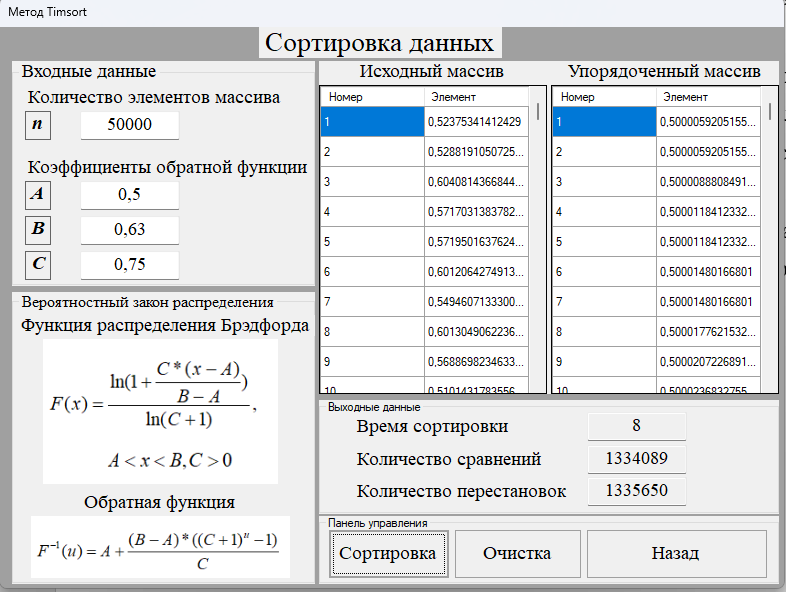
Пример использования приведен на рисунке 1.

Рисунок 1. – Сортировка массива с помощью Timsort

# **3 АНАЛИЗ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧА**

Анализ алгоритма – определение объема ресурсов (памяти и времени выполнения), требуемых алгоритмом для успешной обработки данных.

Для оценки качества алгоритма вводятся понятия сложность и эффективность алгоритма. Чем большее время и объем памяти требуются для реализации алгоритма, тем больше сложность и ниже эффективность.

## **Характеристики алгоритма**

Алгоритм обладает набором численных характеристик, сравнивая значения, которых можно произвести выбор наилучшего алгоритма. Сложность алгоритма делится на временную и ёмкостную, практическую и теоретическую.

Временная (вычислительная) сложность - критерий, характеризующий временные затраты на реализацию алгоритма. Практическая временная сложность оценивается во временных единицах (миллисекунды, секунды, количество временных тактов процессора, количество выполнения циклов).

Емкостная сложность **-** объем памяти компьютера, требуемый для реализации алгоритма. Практическая ёмкостная сложность выражается в битах, байтах, словах.

Вычислительная и емкостная сложности позволяют проводить анализ алгоритмов

## **Функции сложности алгоритмов**

Эффективность программы является важной характеристикой.

Пространственная эффективность измеряется количеством памяти, требуемой для выполнения программы. Временная эффективность программы определяется временем, необходимым для выполнения.

Сравнения алгоритмов производится сопоставлением порядков сложности. Порядок сложности – это функция, доминирующая над точечным выражением временной сложности. Функция  имеет порядок  если имеются константа  и счетчик , такие, что выполняются условия  для любых . Например, точное время обработки массива определяется по формуле



где - размер массива.

Приближенное время определяется функцией

.

Функция сложности – это функция, которая определяет количество сравнений, перестановок, временные и объемные затраты на реализацию алгоритма.

Функция сложности О выражает относительную скорость алгоритма в зависимости от переменных. Для функции сложности сформулированы три правила:

1) ****. Постоянные множители не имеют значения для определения порядка сложности, например, **;**

2) **, **. Порядок сложности произведения двух функций равен произведению их сложностей, например, **;**

3) **** равна доминанте  и . Порядок сложности суммы функций определяется как порядок доминанты первого и второго слагаемых (выбирается наибольший порядок), например, **,** где k – константа, f и g – функции.

## **Виды функции сложности алгоритмов**

Время выполнения алгоритма T зависит от объема входных данных N (рисунок 2)



где - время выполнения алгоритма, мс;

N – объем входных данных.

Для оценивания трудоемкости алгоритмов введена специальная система обозначений – О-нотация.

N

Линерифмическая

Квадратичная

Кубическая

T

Линейная

Рисунок 2 – Графики функций сложности алгоритма

О-нотация позволяет учитывать в функции f (n) значимые элементы, отбрасывая второстепенные:

* 1. кубическая функция – функция f(n), старший член которой содержит . О-нотация имеет вид ;



* 1. квадратическая функция – функция f(n), старший член которой содержит . О-нотация имеет вид ;
  2. линейная функция – функция f(n), старший член которой содержит **.** О-нотация имеет вид ****;
  3. функция линерифмическая – функция, старший член которой равен  логарифмов . О-нотация имеет вид ****.

Например, в функции **** при больших **** компонента **** превосходит остальные слагаемые. Поведение функции определяется компонентой ****. Остальные компоненты отбрасываются. Функция **** имеет оценку поведения (скорость роста значений) ****

О-оценивание позволяет описывать характер поведения функции f(n) с ростом n: насколько быстро или медленно растет эта функция.

О-оценка разбивает функции сложности на группы в зависимости от скорости роста:

1. постоянные функции ****, которые с ростом  не растут;
2. функции с логарифмической скоростью роста **;**
3. функции с линейной скоростью роста ****;
4. функции с линейно-логарифмической скоростью роста ****;
5. функции с квадратичной скоростью роста ****;
6. функции со степенной скоростью роста **** при а>2;
7. функции с показательной или экспоненциальной скоростью роста ****
8. функции с факториальной степенью роста **** /9/.

Описание О-нотаций приведено в таблице 1.

|  |  |
| --- | --- |
| О-нотация | Описание |
| О (1) | Инструкции программы запускаются независимо от n. Время выполнения программы постоянно (помещение в стек). Операции в программе выполняются один или несколько раз. Алгоритм независимо от размера данных требует одно и тоже время. |
| О (n) | Время выполнения программы линейно и зависит от n. Входной элемент обрабатывается линейное число раз. |
| О (n2) | Время выполнения программы является квадратичным. Алгоритмы используются для небольших n (цикл двойного уровня вложенности, сортировки выбором, вставками) |
| О (n3) | Алгоритм программы имеет кубическое время выполнения (цикл тройного уровня вложенности). Применяется для небольших задач. (умножение матриц) |
| О (logn) | Логарифмическая зависимость. С ростом n программа работает медленнее. Время характерно для программ, которые сводят большую задачу к набору меньших подзадач, уменьшая на каждом шаге размер подзадачи на постоянный коэффициент. Общее решение находится в одной из подзадач (бинарный поиск) |
| О (n\*logn) | Линерифмическая зависимость. Время выполнения программы пропорционально n\*logn. Возникает, когда алгоритм решает задачу, разбивая ее на меньшие подзадачи, решает независимо и затем объединяет решения подзадач (комбинация, сортировки быстрая, слиянием) |

Продолжение таблицы 1.

|  |  |
| --- | --- |
| О (2n) | Экспоненциальная зависимость. Прямое решение задач (перебор и сравнение различных решений). Для комбинаторных задач нереализуемы. Сведение к приближенному алгоритму с приближенным значением. |

## **Сравнение асимптотического поведения функций**

Математические обозначения для сравнения асимптотического поведения функций имеют вид:

1. определение О большое. Функция  если имеются положительные константы  и  такие, что , когда ;
2. определение Омега большое. Функция  если имеются положительные константы  и  такие, что , когда ;
3. определение Тета большое. Функция  только и если только  и 
4. определение о малое. Функция  если и только  и 

Определение О большое устанавливает, что имеется такая точка , что для всех значений  за этой точкой значения  ограничены значениями , уможенными на константу. Например, время выполнения алгоритма . В некоторой точке время выполнения ограничивается квадратичной функцией. Константы игнорируются.

Определение Омега большое утверждает, что что скорость роста функции  больше или равна скорости функции . Например, алгоритм основанный на анализе подпоследовательности в задаче требует определения максимальной суммы непрерывной подпоследовательности требует время