**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Алгоритмы и структуры данных»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

«Шейкерная сортировка с использованием кольцевой очереди на базе связанного списка. Оценка сложности сортировки с использованием подсчёта количества элементарных операций в зависимости от количества чисел в файле»

**Выполнила:**

Чу Ван Доан, студент группы N3247

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Ерофеев С. А., преподаватель ФБИТ

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 3

1 ОПИСАНИЕ ФУНКЦИОНАЛА ПРОГРАММЫ 4

2 БЛОК-СХЕМА 6

3 КОД ПРОГРАММЫ 7

4 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 11

ЗАКЛЮЧЕНИЕ 12

**ВВЕДЕНИЕ**

Задача работы – Работать программу шейкерной сортировки, использует кольцевую очередь на базе связного списка. Оценить сложность сортировки посредством подсчета количества элементарных операций в зависимости от количества чисел в файле.

Шейкерная сортировка с использованием кольцевой очереди на базе связанного списка является алгоритмом сортировки, который может быть применен для упорядочивания набора чисел в файле. Этот алгоритм оперирует с кольцевой очередью, которая реализована на базе связанного списка.

Сложность сортировки может быть оценена путем подсчета количества элементарных операций, которые выполняются в процессе сортировки. Элементарные операции могут включать сравнение двух чисел, перестановку элементов и доступ к элементам в памяти.

Оценка сложности шейкерной сортировки с использованием подсчета элементарных операций зависит от количества чисел в файле. Оценка сложности сортировки посредством подсчета количества элементарных операций позволяет определить эффективность алгоритма и сравнить его с другими алгоритмами сортировки.

Для реализации был выбран язык программирования C++.

# ОПИСАНИЕ ФУНКЦИОНАЛА ПРОГРАММЫ

Данный код представляет собой реализацию алгоритма шейкер-сортировки с использованием кольцевой очереди (CircularQueue).

Описание алгоритма:

1. Создается класс CircularQueue, который представляет кольцевую очередь. Он содержит структуру Node для представления элементов очереди и основные операции, такие как добавление элемента (push), удаление элемента (pop), получение первого элемента (front), проверка на пустоту (empty), получение размера очереди (size) и очистка очереди (clear).
2. В функции shakerSort происходит сортировка массива с помощью шейкер-сортировки. Создаются две кольцевые очереди - queue и sorted\_queue. Переменная swapped используется для отслеживания наличия обменов элементов. Переменные start и end определяют границы текущего просмотра массива.
3. Внутри цикла происходит проход слева направо. Элементы массива добавляются в очередь queue. Затем элементы из очереди извлекаются по одному, сравниваются и добавляются в отсортированную очередь sorted\_queue в нужном порядке, пока queue не станет пустой.
4. Элементы из sorted\_queue возвращаются в исходный массив arr.
5. После прохода слева направо происходит проход справа налево, аналогичный предыдущему, но с обратным порядком сравнения.
6. Границы start и end обновляются, сужая просматриваемую область массива.
7. Алгоритм повторяется до тех пор, пока не будет выполнен полный проход без обменов элементов, что означает, что массив отсортирован.
8. Функция printQueue выводит отсортированную очередь на экран.
9. В функции main происходит чтение данных из файла input.txt, создание массива, вызов функции shakerSort для его сортировки и вывод отсортированного массива на экран.

Сложность шейкерной сортировки на базе кольцевой очереди может быть оценена следующим образом:

* **Анализ сложности shakerSort**

**1. Инициализация переменных и внешний цикл**

bool swapped = true;

int start = 0;

int end = n;

Инициализация переменных: 3 операции присваивания (3 операции).

**2. Внешний цикл**

while (swapped) {

swapped = false;

Условие и обновление swapped: 1 операция сравнения и 1 операция присваивания (2 операции для каждой итерации).

**3. Проход слева направо**

for (int i = start; i < end; i++) {

queue.push(arr[i]);

}

Цикл for от start до end (n итераций):

Каждая итерация: 1 операция сравнения, 1 операция увеличения счетчика, 1 операция доступа к массиву и 1 операция push.

Всего: 4𝑛 операций

**4. Сравнение и сортировка в очереди**

first\_element = queue.front();

queue.pop();

while (!queue.empty()){

second\_element = queue.front();

if (first\_element > second\_element){

swapped = true;

sorted\_queue.push(second\_element);

queue.pop();

} else {

sorted\_queue.push(first\_element);

first\_element = second\_element;

queue.pop();

}

}

sorted\_queue.push(first\_element);

* Установка first\_element и pop: 2 операции.
* Цикл while выполняется n раз
* Каждая итерация: 1 операция сравнения (!queue.empty()), 1 операция получения значения (front), 1 операция сравнения (if), операция push и операция pop.
* Если происходит обмен: 4 операции.
* Если обмен не происходит: 5 операций.
* В среднем: около 4.5 операций для каждой итерации
* Всего для цикла while: 𝑛 × 4.5 = 4.5 𝑛 n×4.5=4.5n операций.
* Добавляем 1 операцию push: 1 операция.

**5. Обновление массива**

for (int i = start; i < end; i++) {

arr[i] = sorted\_queue.front();

sorted\_queue.pop();

}

Цикл for от start до end (n итераций):

Каждая итерация: 1 операция сравнения, 1 операция увеличения счетчика, 1 операция присваивания из front и 1 операция pop.

Всего: 4𝑛 операций.

6. Проход справа налево

for (int i = end - 1; i >= start; i--) {

queue.push(arr[i]);

}

Цикл for от end - 1 до start (n итераций):

Каждая итерация: 1 операция сравнения, 1 операция уменьшения счетчика, 1 операция доступа к массиву и 1 операция push.

Всего: 4𝑛 операций.

**7. Сравнение и сортировка в очереди (обратный)**

Похоже на шаг 4, всего: 4.5𝑛 + 1 операций.

**8. Обновление массива (обратный)**

Похоже на шаг 5, всего: 4𝑛 операций.

**9. Обновление границ**

start++;

end--;

Обновление start и end: 2 операции.

Общая сложность для каждой итерации внешнего цикла:

2 + 4n + 4.5n + 1+ 4n + 4.5n + 1+ 4n + 2 = 22n + 6

В худшем случае внешний цикл выполняется до 𝑛 n раз.

Общее количество операций в худшем случае:

n(22n + 6) = 22+ 6n

Максимальная сложность: 𝑂 ( )

* Анализ сложности функции main

Чтобы вычислить сложность всей программы, нужно рассмотреть каждую часть функции main и объединить её с вычислением сложности функции shakerSort.

**1. Открытие файла и проверка**

ifstream file("input.txt");

if (!file.is\_open()) {

cout << "The file cannot be opened for reading." << endl;

return -1;

}

* Открытие файла: 1 операция.
* Проверка открытия файла: 1 операция.
* Вывод сообщения и завершение, если файл не открылся: 1 операция (предположим, что файл всегда открывается, поэтому этот случай пропускаем).

**2. Чтение количества элементов**

int n; file >> n;

Чтение n: 1 операция.

**3. Инициализация и чтение массива**

int \*arr = new int[n];

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

file >> arr[i];

}

* Инициализация динамического массива: 1 операция.
* Цикл for для чтения n элементов:
* Каждая итерация: 1 операция сравнения, 1 операция увеличения счетчика и 1 операция чтения из файла.
* Всего: 3 𝑛 операций.

**4. Закрытие файла**

file.close();

Закрытие файла: 1 операция.

**5. Вызов функции shakerSort**

shakerSort(arr, n);

Сложность shakerSort: 𝑂 ( )

**6. Вывод отсортированного массива**

cout << "Sorted array: ";

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

cout << arr[i] << " ";

}

cout << endl;

* Вывод сообщения: 1 операция.
* Цикл for для вывода n элементов:
* Каждая итерация: 1 операция сравнения, 1 операция увеличения счетчика и 1 операция вывода.
* Всего: 3 𝑛 операций.
* Вывод новой строки: 1 операция.

**Суммарная сложность функции main**

* Открытие файла и проверка: 2 операции.
* Чтение n: 1 операция.
* Инициализация и чтение массива: 3 𝑛 + 1 операций.
* Закрытие файла: 1 операция.
* Вызов shakerSort: n( 2 + 4n + 4.5n + 1+ 4n + 4.5n + 1+ 4n + 2)
* Вывод отсортированного массива: 3 𝑛 + 2 операций.

**Общее количество операций:**

2 + 1 + (3n + 1) + 1 +n(2 + 4n + 4.5n + 1+ 4n + 4.5n + 1+ 4n + 2) + (3n + 2) = 22+ 9n +8

=> Сложность : 𝑂 ( )

# БЛОК-СХЕМА

Изображение выглядит как текст, диаграмма, План, линия

Автоматически созданное описание

1. Блок-схема алгоритма Shakersort

# КОД ПРОГРАММЫ

#include <iostream>

#include <fstream>

using namespace std;

struct Node {

    int data;

    Node \*next;

};

class CircularQueue {

private:

    Node \*head, \*tail;

    int \_size;

public:

    CircularQueue(){

        head = nullptr;

        tail = nullptr;

        \_size = 0;

    }

    bool empty() {

        return head == nullptr;

    }

    void push(int data){

        Node\* p = new Node;

        p->data = data;

        p->next = nullptr;

        if (head == nullptr){

            head = p;

            tail = p;

            tail->next = head;

        }

        else {

            tail->next = p;

            tail = p;

            tail->next = head;

        }

        \_size++;

    }

    void pop(){

        if(empty()) return;

        if (head == tail){

            Node\* temp = head;

            head = tail = nullptr;

            delete temp;

            \_size--;

        }

        else {

            Node\* temp = head;

            head = head->next;

            delete temp;

            \_size--;

        }

    }

    int front(){

        return head->data;

    }

    int size(){

        return \_size;

    }

    void clear(){

        while (!empty()){

            pop();

        }

    }

};

void shakerSort(int arr[], int n) {

    CircularQueue queue, sorted\_queue;

    bool swapped = true;

    int start = 0;

    int end = n;

    int first\_element;

    int second\_element;

    while (swapped) {

        swapped = false;

        // Проход слева направо

        for (int i = start; i < end; i++) {

            queue.push(arr[i]);

        }

        first\_element = queue.front();

        queue.pop();

        while (!queue.empty()){

            second\_element = queue.front();

            if (first\_element > second\_element){

                swapped = true;

                sorted\_queue.push(second\_element);

                second\_element = queue.front();

                queue.pop();

            }

            else {

                sorted\_queue.push(first\_element);

                first\_element = second\_element;

                second\_element = queue.front();

                queue.pop();

            }

        }

        sorted\_queue.push(first\_element);

        for (int i = start; i < end; i++) {

            arr[i] = sorted\_queue.front();

            sorted\_queue.pop();

        }

        // Проход справа налево

        for (int i = end - 1; i >= start; i--) {

            queue.push(arr[i]);

        }

        first\_element = queue.front();

        queue.pop();

        while (!queue.empty()){

            second\_element = queue.front();

            if (first\_element < second\_element){

                swapped = true;

                sorted\_queue.push(second\_element);

                second\_element = queue.front();

                queue.pop();

            }

            else {

                sorted\_queue.push(first\_element);

                first\_element = second\_element;

                second\_element = queue.front();

                queue.pop();

            }

        }

        sorted\_queue.push(first\_element);

        for (int i = end - 1; i >= start; i--) {

            arr[i] = sorted\_queue.front();

            sorted\_queue.pop();

        }

        // Обновление границ

        start++;

        end--;

    }

}

void printQueue(CircularQueue q) {

    cout << "Sorted queue: ";

    while (!q.empty()) {

        cout << q.front() << " ";

        q.pop();

    }

    cout << endl;

}

int main() {

    ifstream file("input.txt");

    if (!file.is\_open()) {

        cout << "The file cannot be opened for reading." << endl;

        return -1;

    }

    int n; file >> n;

    int \*arr = new int[n];

    for (int i = 0; i < n; i++){

        file >> arr[i];

    }

    file.close();

    shakerSort(arr, n);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++){

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

# РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

1. Массив чисел вводится в программу.



1. После сортировки массив чисел с использованием кольцевой очереди на базе связного списка с помощью алгоритма Shakersort

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

1. Массив чисел вводится в программу.



1. После сортировки массив чисел с использованием кольцевой очереди на базе связанного списка с помощью алгоритма Shakersort

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения данной лабораторной работы, я разработал и реализовал алгоритм шейкерной сортировка с использованием кольцевой очереди на базе связанного списка.

Шейкерная сортировка — это алгоритм сортировки, который является вариацией сортировки пузырьком. Алгоритм шейкерной сортировки выполняет проходы по списку в обоих направлениях, сравнивая и переставляя соседние элементы, если это необходимо.

Оценка сложности сортировки с использованием подсчета количества элементарных операций в зависимости от количества чисел в файле позволяет оценить эффективность алгоритма. В данной реализации шейкерной сортировки сложность составляет примерно O(n^2), где n - количество чисел в файле. Это означает, что время выполнения алгоритма будет увеличиваться квадратично с увеличением размера входных данных. Оценка сложности основана на том, что внешний цикл выполняется примерно n/2 раз, а внутренние циклы проходят каждый по n элементов в худшем случае. В целом, шейкерная сортировка является улучшенной версией пузырьковой сортировки, которая обеспечивает более эффективную работу за счет двустороннего прохода по массиву.

В целом, алгоритм шейкерной сортировки с использованием кольцевой очереди на базе связанного списка представляет собой эффективный способ сортировки массива чисел. Однако, при работе с большими объемами данных или в случае необходимости оптимизации времени выполнения, возможно рассмотреть другие алгоритмы сортировки с лучшей асимптотической сложностью. Лабораторная работа успешно реализует шейкерную сортировку с использованием кольцевой очереди на базе связанного списка на языке программирования C++.