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 №2

«Сортировка слиянием на основе дека»

Выполнила:

Дынина Е.А., студент группы N3249

_____
(подпись)

Проверил:

Ерофеев С.А.

_____
(отметка о выполнении)

_____
(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Цель работы — разработать программу сортировки нисходящим слиянием для чисел из файла, которые записываются в дек на базе связного списка. Оценить сложность.

Для выполнения цели поставлены следующие задачи:

- Изучить класс деков;
- Создать блок-схему алгоритма с обработкой ошибок;
- Написать программу на языке C++ с интерфейсом для пользователей;
- Оценить сложность программы.

ТЕХНИЧЕСКАЯ ЗАДАЧА

Дек — абстрактный тип данных, экземпляр которого является двусторонней или двунаправленной очередью. Базовые операции с деком:

- Добавление элемента в начало очереди;
- Добавление элемента в конец очереди;
- Удаление первого элемента;
- Удаление последнего элемента;
- Определение размера дека;
- Проверка дека на пустоту.

Сортировка слиянием заключается в переборе срезов исходного дека размером i с шагом 2i, где i от 1 до n, где n — размер дека. Берутся по два соседних среза, поэлементно сравниваются, и в исходный дек сначала добавляются меньшие значения, а потом, если сравнивать больше нечего, то все остальные. Таким образом, перебрав все срезы, весь дек отсортируется.

Входные данные

На вход в программу через указанный заранее файл подается дек с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Промежуточные данные

В ходе работы программы задействованы промежуточные переменные, указанные в таблице 1.

Таблица 1 – Промежуточные данные

Название переменной	Тип в Python	Диапазон типа	Значение
$file\_path$	string	—	Путь к файлу с числами
num	long double	$[-1{,}2 \cdot 10^{4932}, 1{,}2 \cdot 10^{4932}]$	Переменная, в которую считываем числа из файла
cur	int	$[-2147483648, 2147483647]$	Позиция указателя
$left$	int	$[-2147483648, 2147483647]$	Левый индекс среза

mid	int	$[-2147483648, 2147483647]$	Середина среза
$right$	int	$[-2147483648, 2147483647]$	Правый индекс среза
$it1$	int	$[-2147483648, 2147483647]$	Индекс для отслеживания текущего положения в левой части среза
$it2$	int	$[-2147483648, 2147483647]$	Индекс для отслеживания текущего положения в правой части среза
i	int	$[-2147483648, 2147483647]$	Переменная для реализации перебора в цикле
j	int	$[-2147483648, 2147483647]$	Переменная для реализации перебора в цикле

Выходные данные

В результате работы программы в консольное приложение выводится отсортированный дек с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Используемые функции

- **ifstream —** открывает файл для чтения;
- **.is_open() —** проверяет, открыт ли файл;
- **numeric_limits <тип>::lowest —** возвращает наименьшее отрицательное конечное значение типа;
- **numeric_limits <тип>::max —** возвращает максимальное конечное значение типа;
- **.clear() —** сбрасывает состояние потока ifstream;
- **.tellg() —** возвращает текущую позицию указателя в файле;
- **.seekg() —** устанавливает указатель на определенную позицию в файле;
- **.close() —** закрывает файл;
- **merge(Deque& dq, int left, int mid, int right) —** сравнивает два среза дека, добавляет меньшие значения в дек, а потом добавляет оставшиеся;
- **sorting(Deque& dq) —** определяет срезы и проводит сортировку;

Функции класса Deque

- **.pushFront(double x) —** добавление элемента x в начало дека;

- **.pushBack(double x) —** добавление элемента x в конец дека;
- **.popFront() —** удаление элемента из начала дека;
- **.popBack() —** удаление элемента из конца дека;
- **.get(int index) —** возвращение значения элемента дека по индексу;
- **.set(int index, double value) —** установление элемента по индексу;
- **.size() —** возвращение размера дека;
- **.empty() —** проверка дека на пустоту;
- **.print() —** выводит все элементы дека.

БЛОК-СХЕМА

Рассмотрим блок-схему придуманного алгоритма, где реализована сортировка слиянием на основе дека на двухсвязном списке и учтены всевозможные ошибки.

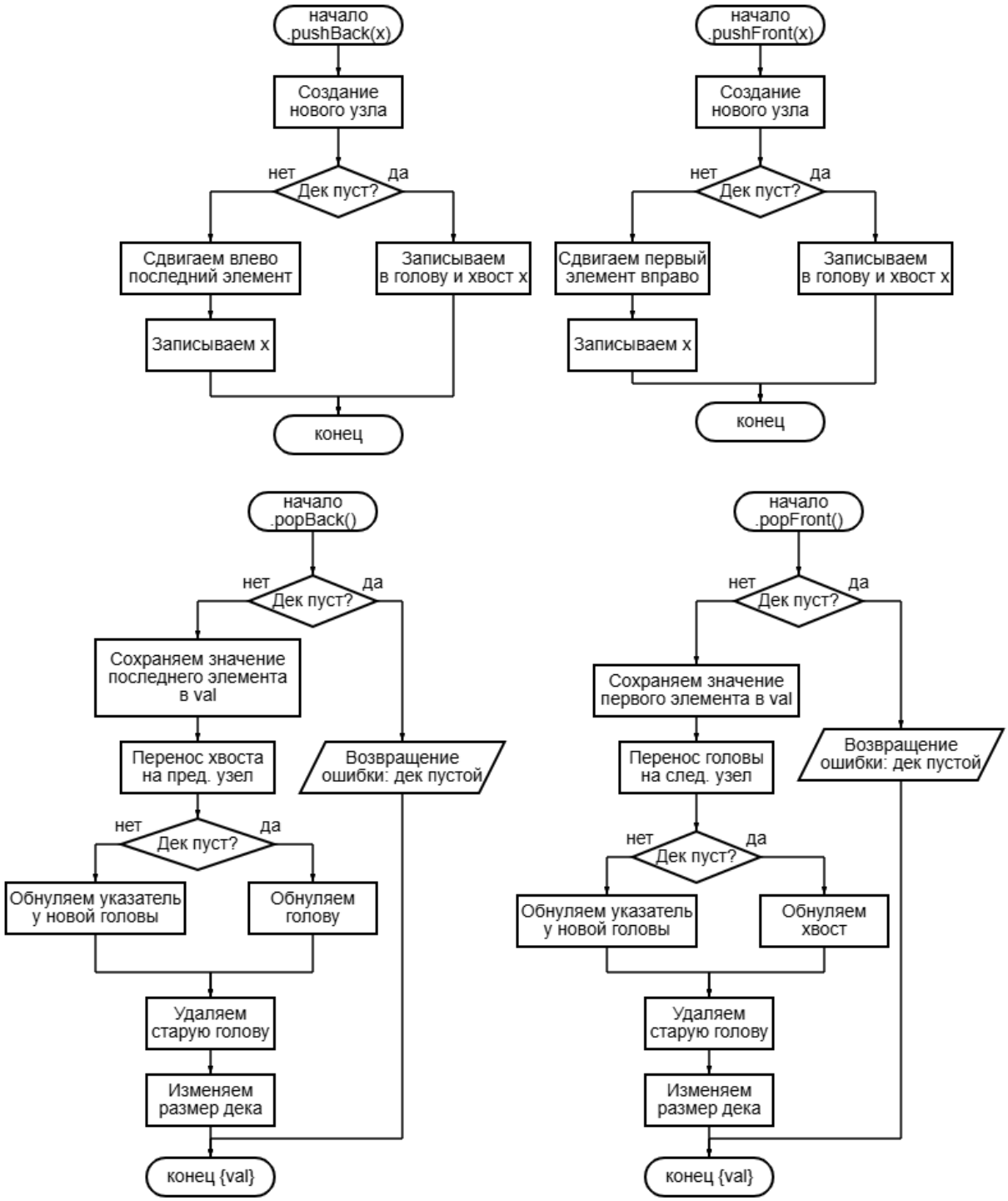

Рисунок 1 – Блок-схема функций добавления и удаления класса Deque

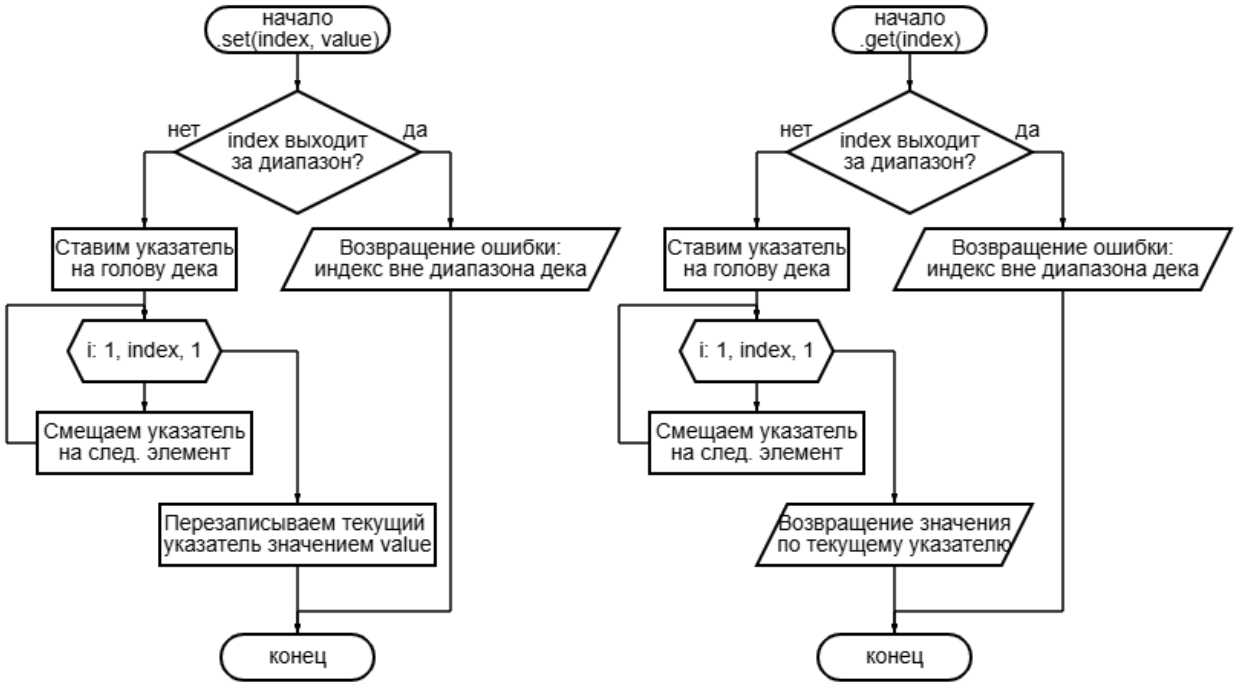

Рисунок 2 – Блок-схема функций возвращения и установления элемента класса Deque

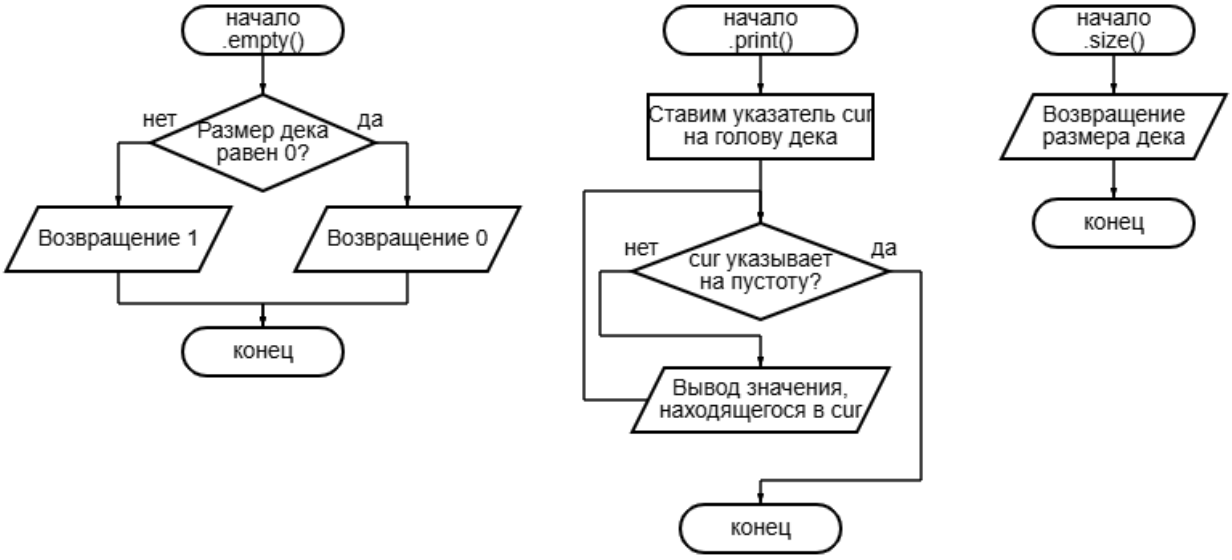

Рисунок 3 – Блок-схема функций вывода, проверки на пустоту и размера Deque

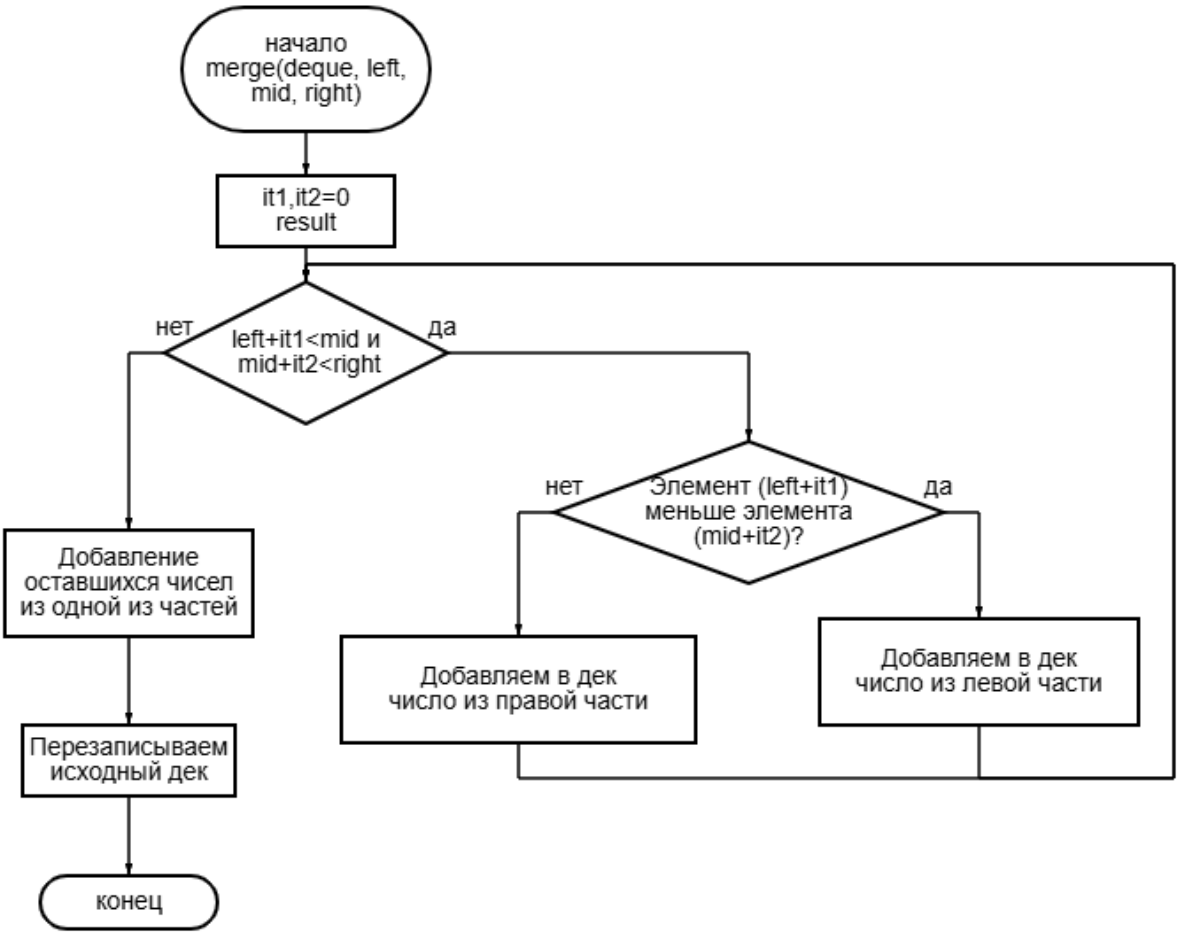

Рисунок 4 – Блок-схема функции слияния

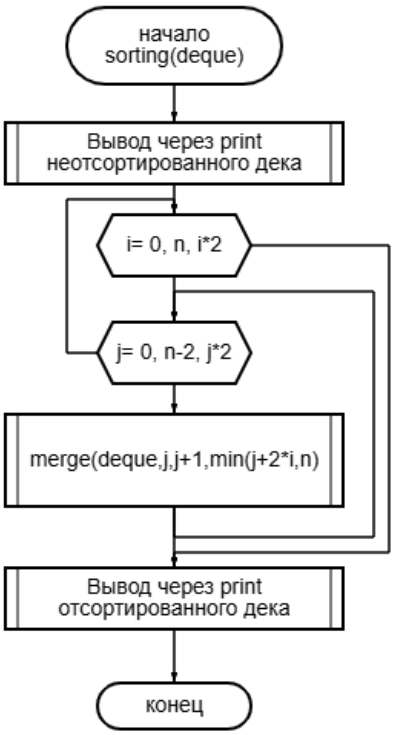

Рисунок 5 – Блок-схема функций добавления и удаления элементов класса Deque

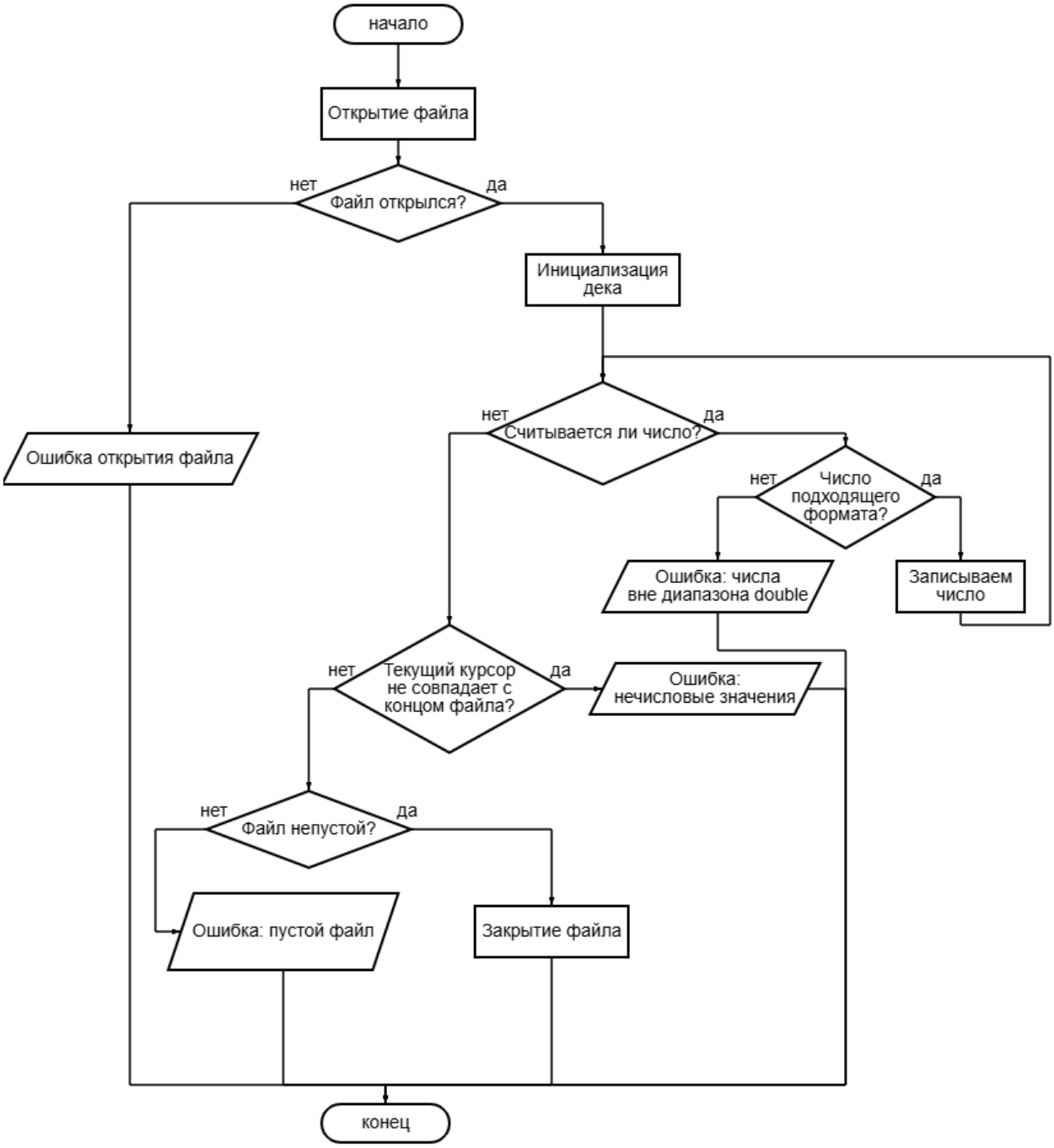

Рисунок 6 – Блок-схема основной части программы

КОД

Ниже приведен код, написанный для сортировки слиянием чисел, подаваемых через файл в дек на основе связного списка, по указанной выше блок-схеме. Программа реализована на языке C++.

```
#include <iostream>
#include <fstream>
#include <limits>
using namespace std;
struct Node { // Структура узла
        double value; // присваивание: 1
        Node* next; // присваивание: 1
        Node* prev; // присваивание: 1
        Node(double val) : value(val), next(nullptr), prev(nullptr) {}
}; // На структуру: 3
class Deque { // Класс дека
private:
        Node* head;
        Node* tail;
        int size_;
public:
        Deque(): head(nullptr),tail(nullptr),size_(0){}//присваиваний:3. Deque: 3
        // Добавление в начало
        void pushFront(double x) {
        Node* new_node = new Node(x);
        if (!head) { // Если дек пуст
                head = tail = new_node;
        }
        else { // Сдвиг первого элемента вправо
                new_node->next = head;
                head->prev = new_node;
                head = new_node;
        }
        size_++;    }
```

```
// Добавление в конец
void pushBack(double x) {
Node* new_node = new Node(x); // создание node: 3
if (!tail) { // Если дек пуст. Обращение: 1
        head = tail = new_node; // присваивания: 2
}
else { // Сдвиг конца на новый элемент
        new_node->prev = tail; // обращение и присваивание: 2
        tail->next = new_node; // обращение и присваивание: 2
        tail = new_node; // присваивание: 1
}
size_++;}Инкремент: 2. Push_back: 11
// Удаление из начала
double popFront() {
if (!head) { // Проверка на пустоту
        throw runtime_error("Deque is empty");
}
double val = head->value;
Node* temp = head;
head = head->next; // Перенос головы на след. узел
if (head) {  // Обнуляем старую голову
        head->prev = nullptr;
}
else {  // Обнуляем хвост
        tail = nullptr;
}
delete temp; // Удаляем старую голову
size_--;
return val;
}
// Удаление с конца
double popBack() {
if (!tail) { // Проверка на пустоту. Обращение: 1
        throw runtime_error("Deque is empty");} //Возвращение ошибки:1
```

```
double val = tail->value; // Обращение и присваивание: 2
Node* temp = tail; // Присваивание: 1
tail = tail->prev; // Перенос хвоста на пред. узел. Обращение и присваивание: 2
if (tail) { // Обнуляем старый хвост. Обращение: 1
        tail->next = nullptr; // Обращение и присваивание: 2
}
else { // Обнуляем  голову
        head  = nullptr; // присваивание: 1
}
delete temp; // Удаляем старый хвост: 1
size_--; // Декремент: 2
return val; // Возвращение: 1
} // pop_back: 13
// Возвращение значения по индексу
double get(int index) const {
if (index < 0 || index >= size_) { // сравнения: 3
        throw out_of_range("Index out of range"); // возвращение ошибки: 1
}
Node* cur = head; // Присваивание: 1
for (int i = 0; i < index; ++i) { // Присваивание, сравнение и инкремент: 1+3n
        cur = cur->next; // обращение и присваивание: 2
}//1+5n
return cur->value; // возвращение и обращение: 2
}// get: 7+5n
// Установление значения по индексу
void set(int index, double value) {
if (index < 0 || index >= size_) { // сравнения: 3
        throw out_of_range("Index out of range"); // возвращение ошибки: 1
}
Node* cur = head; // Присваивание: 1
for (int i = 0; i < index; ++i) { // Присваивание, сравнение и инкремент: 1+3n
        cur = cur->next; // обращение и присваивание: 2
}// 1+5n
cur->value = value; }//Обращение и присваивание: 2. set: 7+5n
```

```
        // Размер дека
        int size() const { return size_; }// Обращение и возвращение: 2. Size: 2
         // Проверка на пустоту
        bool empty() const { return size_ == 0; }
        // Вывод дека
        void print() const{
        Node* cur=head; // Обращение и присваивание: 2
        while (cur != nullptr) { // Сравнение: n
                cout << cur->value << " "; // Обращение и вывод:2
                cur=cur->next; //Обращение и присваивание: 2
        }// O(5n)
        cout << endl; //Вывод: 1
        }// print: 3+5n
};
void merge(Deque& dq, int left, int mid, int right){// Слияние двух частей
        int it1=0, it2=0; // Присваивание: 2
        // Вспомогательный дек
        Deque result; // Инициализация: 3
        // Добавление меньших чисел из двух частей
        while(left+it1<mid and mid+it2<right){ // Сложения, сравнения и булевая операция: 5n
                // Меньшее число из левой части
                if(dq.get(left+it1)<dq.get(mid+it2)){ // Сложения, сравнение, get: 17+10n
                        result.pushBack(dq.get(left+it1)); //Сложение, get, pushBack: 19+5n
                        it1++;}// Инкремент: 2
                // Меньшее число из правой части
                else {
                        result.pushBack(dq.get(mid+it2)); //Сложение, get, pushBack: 19+5n
                        it2++;}// Инкремент: 2
        } // 43n+15n^2
        // Добавление оставшихся чисел из одной из частей
        while(left+it1<mid){ // Сложение и сравнение: 2n
                result.pushBack(dq.get(left+it1)); //Сложение, get, pushBack: 19+5n
                it1++; // Инкремент: 2
        } // 23n+5n^2
```

```
        while(mid+it2<right){ // Сложение и сравнение: 2n
                result.pushBack(dq.get(mid+it2)); //Сложение, get, pushBack: 19+5n
                it2++; // Инкремент: 2
        } // 23n+5n^2
        // Перезаписываем исходный дек
        for(int i=0;i<it1+it2;i++){ // Присваивание, сравнение, сложение и инкремент: 1+4n
                dq.set(left+i, result.get(i)); // Set, сложение, get: 15+10n
        } // 1+19n+10n^2
} // merge: 6+108n+35n^2
void sorting(Deque& dq){
        int n=dq.size(); // Присваивание и size: 3
        // Вывод содержимого неотсортированного дека
        cout << "Unsorted deque:" << endl; // Вывод: 2
        dq.print(); // print: 3+5n
        // Сортировка
        // Определяем срезы дека и проходимся сортировкой по ним
        for(int i=1;i<n+1;i*=2){ // Присваивание, слож., сравн., умножение: 1 + 4log_2(n)
                for(int j=0;j<n-1;j+=2*i){//Присваивание, вычит., сравн., умн.: 1+5n
                merge(dq,j, min(j+i,n), min(j+2*i,n));//merge, арифм.опер.: 11+108n+35n^2
                }//1+16n+108n^2+35n^3
        }// 1 + 5log_2(n) + 16n + 108n^2 + 35n^3
        // Вывод содержимого отсортированного дека
        cout << "Sorted deque:" << endl; // Вывод: 1
        dq.print(); // print: 3+5n
} // sorting: 12 + 5 log_2 n + 26n^2 + 108n^2 + 35n^3
int main() {
        string file_path="data.txt";
        // Открываем файл для чтения
        ifstream file(file_path);
        // Проверка на ошибку при открытии
        if (!file.is_open()) {
                cerr << "Error opening file" << endl;
                return 1;
                }
```

```
	Deque dq;
	long double num;
	while (file >> num) {
		// Проверка на корректность данных
		if(num<numeric_limits<double>::max() and num>numeric_limits<double>::
lowest( )) {
			dq.pushBack(num);
			continue;
		}
		cout<<"Error: numbers out of range";
		return 1;
	}
	// Проверка на ввод всех символов
	file.clear(); // Сброс возможных ошибок
	int cur=file.tellg(); // Сохраняем позицию курсора
	file.seekg(0, ios::end); // Перемещаем курсор в конец файла
	if(cur!=file.tellg()) { //Сравниваем сохраненный курсор с концом файла
		cout<<"Error: non-numeric data";
		return 1;
	}
	// Проверка на наличие данных
	if(dq.size()==0){
		cout<<"Error: empty file";
		return 1;
	}
	sorting(dq);
	return 0;
}
```

ТЕСТИРОВАНИЕ

Проведено тестирование программы с помощью различных входных данных. Был использован компилятор C++ 3.4.2 и среда разработки Embarcadero Dev-C++ 6.3

1. Дек с разнообразными числами (1e+2 -200 300 -9 4)

```
Unsorted deque:
100 -200 300 -9 4
Sorted deque:
-200 -9 4 100 300
```

Рисунок 7 – Тест №1

2. Дек, где значения по возрастанию (-9 -5 0 6 34)

```
Unsorted deque:
-9 -5 0 6 34
Sorted deque:
-9 -5 0 6 34
```

Рисунок 8 – Тест №2

3. Дек с большим количеством чисел (20 -4 52 85 6 49 1 4 28 -2 03 -2 7753 -3456 -1647 782)

```
Unsorted deque:
20 -4 52 85 6 49 1 4 28 -2 3 -2 7753 -3456 -1647 782
Sorted deque:
-3456 -1647 -4 -2 -2 1 3 4 6 20 28 49 52 85 782 7753
```

Рисунок 9 – Тест №4

4. Файл с нечисловыми символами (1e+2 -20sdf0 300 -9 4)

```
Error: non-numeric data
```

Рисунок 10 – Тест №5

5. Файл с числами, превышающими диапазон типа double (1e+3000 -200 300 -9 4)

```
Error: numbers out of range
```

Рисунок 11 – Тест №6

СЛОЖНОСТЬ ПРОГРАММЫ

Сложность основных операций программы оценена в пункте Код (с.11). В программе используются вспомогательные операции get и set, имеющие линейную сложность, и функция слияния с квадратичной сложностью. Общая сложность сортировки определяется сложностью слияния и циклов, которые перебирают срезы, и оценена как $O(n^3)$. Если реализовать слияние без вспомогательных функций, то сложность программы может понизиться.

Для оценки времени выполнения функции была использована библиотека chrono.

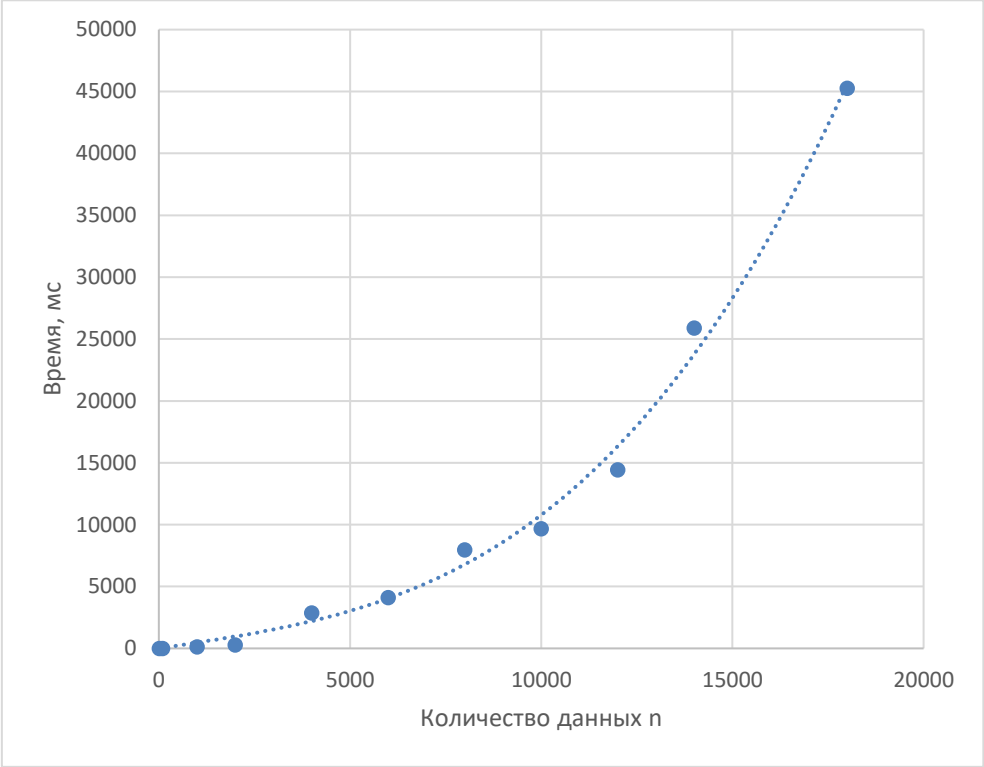

Рисунок 12 – Зависимость времени выполнения от количества данных

ЗАКЛЮЧЕНИЕ

Был изучен класс деков и написана блок-схема алгоритма программы, которая реализует сортировку слиянием чисел из файла на основе деков на связном списке. На языке C++ написан код программы с выводом неотсортированного и отсортированного дека, обработкой ошибок и интерфейсом для пользователя. Программа успешно протестирована на различных наборах данных. Сложность программы — $O(n^3)$.

Все задачи выполнены и цель достигнута.