Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский химико-технологический университет имени Д.И. Менделеева»

Факультет цифровых технологий и химического инжиниринга

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

ПО КУРСУ

«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

Ведущий преподаватель

Ассистент Крашенинников Р. С.

СТУДЕНТ группы КС-36 Лупинос А. В.

Москва

2025

Задание

Написать свою реализацию двусвязного списка:

- Добавление элемента в начало, в конец, в произвольное место;
- Удаление элемента по из списка.

В рамках лабораторной работы необходимо изучить и реализовать двусвязный список, при этом структура должна:

- Использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными;
- Реализовывать свой итератор, предоставляющий стандартный для языка механизм работы с ним;
- Обеспечивать работу стандартных библиотек и конструкции for each, если она есть в языке, если их нет, то реализовать собственную функцию использующую итератор;
- Проверку на пустоту и подсчет количества элементов.

Для демонстрации работы структуры необходимо создать набор тестов (под тестом понимается функция, которая создает структуру, проводит операцию или операции над структурой и удаляет структуру):

- Заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального;
- Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов;
- Заполнение контейнера 100 структур содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнения необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры содержащие результат фильтрации, проверить выполнение на правильность подсчетом количества элементов, не подходящих под условие в новых структурах.

- Тесты для списка:
    - Перемешать все элементы;
    - Выполнить серию тестирования сортировки из первой лабораторной работы на реализованном списке и сравнить производительность с полученной на массиве.

В первой лабораторной работе была выполнена сортировка перемешиванием.

Описание алгоритма

Двусвязный список — это линейная структура данных, в которой каждый элемент (узел) содержит данные и два указателя: на предыдущий и следующий узлы, что позволяет перемещаться по списку в обоих направлениях. В отличие от односвязного списка, где доступ возможен только вперед, двусвязный список обеспечивает более гибкую навигацию, что упрощает такие операции, как вставка и удаление элементов.

Основным преимуществом является возможность быстрого доступа к соседним элементам с обеих сторон узла, что делает его эффективным для алгоритмов, требующих обратного обхода, например, коктейльной сортировки с указателями. Однако это достигается ценой увеличения объема памяти, так как каждый узел хранит дополнительный указатель, а также усложняет реализацию по сравнению с массивом или односвязным списком.

К минусам можно отнести отсутствие прямого доступа к элементам по индексу (в отличие от массива), что требует линейного времени для достижения произвольного элемента, если не использовать указатели напрямую.

Сортировка перемешиванием, или Шейкерная сортировка, или двунаправленная (англ. Cocktail sort) — разновидность пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства.

Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, ее можно исключить из рассмотрения.

Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

Лучший случай для сортировки перемешиванием — отсортированный массив O(n), худший — отсортированный в обратном порядке $O(n^2)$. Усредненным случаем также будет являться $O(n^2)$.

То есть, подытожив, сортировка перемешиванием является измененной версией сортировки пузырьком, в которой также руководствуются идеей постоянного обмена местами двух элементов, только в этот раз не просто с проходом по массиву от начала в сторону конца, смещая все большие элементы к концу, но еще и добавлением обратного хода, смещая малые элементы к началу.

Описание выполнения задачи

Для реализации двусвязного списка и алгоритма сортировки перемешиванием была написана программа на языке программирования C++:

```
#include <iostream>
#include <string>
#include <random>
#include <ctime>
#include <stdexcept>
#include <algorithm>
#include <utility>
#include <vector>
#include <chrono>
#include <limits>

using namespace std;

// Структура для даты
struct Date {
    int day, month, year;
    Date() : day(1), month(1), year(1980) {}
```

```
    Date(int d, int m, int y) : day(d), month(m), year(y) {}
};

// Структура для человека
struct Person {
    string surname, name, patronymic;
    Date birthDate;
};

// Структура для статистики сортировки
struct SortStats {
    long long swap_count;
    int full_passes;
};

// Узел двусвязного списка
template<typename T>
struct Node {
    T data;
    Node* prev;
    Node* next;
    explicit Node(T  value) : data(std::move(value)), prev(nullptr), next(nullptr) {}
};

// Шаблонный класс двусвязного списка
template<typename T>
class DoublyLinkedList {
private:
    Node<T>* head;
    Node<T>* tail;
    size_t size;

public:
    class Iterator {
    private:
        Node<T>* current;
    public:
        explicit Iterator(Node<T>* node) : current(node) {}
        T& operator*() { return current->data; }
        Iterator& operator++() {
            current = current->next;
            return *this;
        }
        bool operator!=(const Iterator& other) const {
            return current != other.current;
        }
    };

    DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}

    ~DoublyLinkedList() {
        while (head) {
            Node<T>* temp = head;
```

```
        head = head->next;
        delete temp;
    }
}

void push_front(const T& value) {
    auto* newNode = new Node<T>(value);
    size++;
    if (!head) {
        head = tail = newNode;
        return;
    }
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
}

void push_back(const T& value) {
    auto* newNode = new Node<T>(value);
    size++;
    if (!head) {
        head = tail = newNode;
        return;
    }
    newNode->prev = tail;
    tail->next = newNode;
    tail = newNode;
}

void insert(size_t index, const T& value) {
    if (index > size) throw out_of_range("Index out of range");

    if (index == 0) {
        push_front(value);
        return;
    }
    if (index == size) {
        push_back(value);
        return;
    }

    Node<T>* current = head;
    for (size_t i = 0; i < index; i++) {
        current = current->next;
    }

    auto* newNode = new Node<T>(value);
    newNode->prev = current->prev;
    newNode->next = current;
    current->prev->next = newNode;
    current->prev = newNode;
    size++;
}
```

```
void remove(size_t index) {
    if (!head) throw runtime_error("List is empty");
    if (index >= size) throw out_of_range("Index out of range");

    Node<T>* current = head;
    for (size_t i = 0; i < index; i++) {
        current = current->next;
    }

    if (current == head) {
        head = head->next;
        if (head) head->prev = nullptr;
    }
    else if (current == tail) {
        tail = tail->prev;
        tail->next = nullptr;
    }
    else {
        current->prev->next = current->next;
        current->next->prev = current->prev;
    }
    delete current;
    size--;
}

void print() const {
    Node<T>* current = head;
    cout << "List (" << size << " elements): ";
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << "\n";
}

[[nodiscard]] bool empty() const { return size == 0; }
[[nodiscard]] size_t getSize() const { return size; }
Iterator begin() { return Iterator(head); }
Iterator end() { return Iterator(nullptr); }

T& get(size_t index) {
    if (index >= size) throw out_of_range("Index out of range");
    Node<T>* current = head;
    for (size_t i = 0; i < index; i++) {
        current = current->next;
    }
    return current->data;
}

void shuffle() {
    random_device rd;
    mt19937 gen(rd());
```

```
        for (size_t i = size - 1; i > 0; i--) {
            uniform_int_distribution<> dis(0, i);
            size_t j = dis(gen);
            swap(get(i), get(j));
        }
    }

    // Оптимизированная коктейльная сортировка с указателями
    SortStats cocktailSort() {
        if (size <= 1) return {0, 0};

        Node<T>* left = head;
        Node<T>* right = tail;
        bool flag;
        long long swap_count = 0;
        int full_passes = 0;

        while (left != right && left->prev != right) {
            flag = false;
            ++full_passes;

            // Проход справа налево
            Node<T>* current = right;
            while (current != left) {
                if (current->prev->data > current->data) {
                    swap(current->prev->data, current->data);
                    flag = true;
                    ++swap_count;
                }
                current = current->prev;
            }
            left = left->next;

            // Проход слева направо
            current = left;
            while (current != right) {
                if (current->data > current->next->data) {
                    swap(current->data, current->next->data);
                    flag = true;
                    ++swap_count;
                }
                current = current->next;
            }
            right = right->prev;

            if (!flag) break;
        }

        return {swap_count, full_passes};
    }
};

// Тест 1: Работа с числами
```

```
void testNumbers() {
    DoublyLinkedList<int> list;
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(-1000, 1000);

    for (int i = 0; i < 1000; i++) {
        list.push_back(dis(gen));
    }

    int sum = 0, min = 1000, max = -1000;
    for (int num : list) {
        sum += num;
        min = std::min(min, num);
        max = std::max(max, num);
    }

    double avg = static_cast<double>(sum) / 1000;
    cout << "Test 1 (Numbers):\n";
    cout << "Sum: " << sum << "\nAverage: " << avg
         << "\nMin: " << min << "\nMax: " << max << "\n";
    cout << "Is empty: " << (list.empty() ? "Yes" : "No") << "\n";
    cout << "Size: " << list.getSize() << "\n\n";
}

// Тест 2: Работа со строками
void testStrings() {
    DoublyLinkedList<string> list;
    string initial[] = {"One", "Two", "Three", "Four", "Five",
                        "Six", "Seven", "Eight", "Nine", "Ten"};
    for (const auto & i : initial) {
        list.push_back(i);
    }

    cout << "Test 2 (Strings) - Interactive\n";
    cout << "Initial list:\n";
    list.print();

    int choice;
    while (true) {
        cout << "\nOperations:\n";
        cout << "1. Add to front\n";
        cout << "2. Add to back\n";
        cout << "3. Insert at index\n";
        cout << "4. Remove at index\n";
        cout << "5. Check if empty\n";
        cout << "6. Get size\n";
        cout << "7. Print list\n";
        cout << "8. Exit test\n";
        cout << "Enter choice (1-8): ";
        cin >> choice;
        cin.ignore();
```

```
    if (choice == 8) break;

    string input;
    size_t index;
    switch (choice) {
        case 1:
            cout << "Enter string to add to front: ";
            getline(cin, input);
            list.push_front(input);
            list.print();
            break;
        case 2:
            cout << "Enter string to add to back: ";
            getline(cin, input);
            list.push_back(input);
            list.print();
            break;
        case 3:
            cout << "Enter index (0-" << list.getSize() << "): ";
            cin >> index;
            cin.ignore();
            cout << "Enter string to insert: ";
            getline(cin, input);
            try {
                list.insert(index, input);
                list.print();
            } catch (const out_of_range& e) {
                cout << "Error: " << e.what() << "\n";
            }
            break;
        case 4:
            cout << "Enter index to remove (0-" << (list.getSize()-1) << "): ";
            cin >> index;
            try {
                list.remove(index);
                list.print();
            } catch (const exception& e) {
                cout << "Error: " << e.what() << "\n";
            }
            break;
        case 5:
            cout << "List is " << (list.empty() ? "empty" : "not empty") << "\n";
            break;
        case 6:
            cout << "List size: " << list.getSize() << "\n";
            break;
        case 7:
            list.print();
            break;
        default:
            cout << "Invalid choice!\n";
    }
}
```

```
}

// Тест 3: Работа с персонами
void testPersons() {
    DoublyLinkedList<Person> list;
    string surnames[] = {"Ivanov", "Petrov", "Sidorov", "Kuznetsov"};
    string names[] = {"Alexey", "Boris", "Sergey", "Dmitry"};
    string patronymics[] = {"Ivanovich", "Petrovich", "Sergeevich"};

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> yearDist(1980, 2019);
    uniform_int_distribution<> monthDist(1, 12);
    uniform_int_distribution<> dayDist(1, 28);
    uniform_int_distribution<> nameDist(0, 3);

    for (int i = 0; i < 100; i++) {
        Person p;
        p.surname = surnames[nameDist(gen)];
        p.name = names[nameDist(gen)];
        p.patronymic = patronymics[nameDist(gen) % 3];
        p.birthDate = Date(dayDist(gen), monthDist(gen), yearDist(gen));
        list.push_back(p);
    }

    DoublyLinkedList<Person> under20, over30;
    int currentYear = 2025;

    for (const Person& p : list) {
        int age = currentYear - p.birthDate.year;
        if (age < 20) under20.push_back(p);
        if (age > 30) over30.push_back(p);
    }

    cout << "Test 3 (Persons):\n";
    cout << "People under 20: " << under20.getSize() << "\n";
    cout << "People over 30: " << over30.getSize() << "\n";

    size_t expectedRemainder = 100 - under20.getSize() - over30.getSize();
    size_t actualRemainder = 0;
    for (const Person& p : list) {
        int age = currentYear - p.birthDate.year;
        if (age >= 20 && age <= 30) actualRemainder++;
    }

    cout << "Verification (people aged 20-30): " << actualRemainder
         << " (expected: " << expectedRemainder << ")\n";
    cout << "Is empty: " << (list.empty() ? "Yes" : "No") << "\n";
    cout << "Size: " << list.getSize() << "\n\n";
}

// Тест 4: Перемешивание чисел
void testShuffle() {
```

```
    DoublyLinkedList<int> list;
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(1, 100);

    for (int i = 0; i < 10; i++) {
        list.push_back(dis(gen));
    }

    cout << "Test 4 (Shuffle):\n";
    cout << "Before shuffling:\n";
    list.print();

    list.shuffle();

    cout << "After shuffling:\n";
    list.print();
    cout << "Is empty: " << (list.empty() ? "Yes" : "No") << "\n";
    cout << "Size: " << list.getSize() << "\n\n";
}

// Тест 5: Оптимизированная коктейльная сортировка
void testCocktailSort() {
    int sizes[8] = {1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000};
    vector<vector<double>> sec_times(8);
    vector<vector<long long>> swap_counts(8);
    vector<vector<int>> full_passes(8);

    vector<double> best_time(8, numeric_limits<double>::max());
    vector<double> worst_time(8, numeric_limits<double>::lowest());
    vector<double> avg_time(8, 0);
    vector<double> avg_swaps(8, 0);
    vector<double> avg_passes(8, 0);

    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> distribution(-1, 1);

    cout << "Test 5 (Optimized Cocktail Sort on Doubly Linked List):\n";
    for (int s = 0; s < 8; ++s) {
        for (int k = 0; k < 20; ++k) {
            int M = sizes[s];
            DoublyLinkedList<double> list;

            // Заполнение списка случайными числами
            for (int i = 0; i < M; ++i) {
                list.push_back(distribution(gen));
            }

            // Замер времени и сортировка
            chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now();
            SortStats stats = list.cocktailSort();
            chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now();
```

```
        chrono::duration<double> sec_diff = end - start;
        double time_s = sec_diff.count();
        sec_times[s].push_back(time_s);
        swap_counts[s].push_back(stats.swap_count);
        full_passes[s].push_back(stats.full_passes);

        best_time[s] = min(best_time[s], time_s);
        worst_time[s] = max(worst_time[s], time_s);
        avg_time[s] += time_s;
        avg_swaps[s] += (double)stats.swap_count;
        avg_passes[s] += stats.full_passes;
        cout << "END OF " << k + 1 << " TRY for size " << M << endl;
    }
    avg_time[s] /= 20.0;
    avg_swaps[s] /= 20.0;
    avg_passes[s] /= 20.0;
    cout << "==END OF " << sizes[s] << " SIZE OF LIST==" << endl;
}

cout << "\n=== Sorting Times (Seconds) ===" << endl;
for (int s = 0; s < 8; ++s) {
    cout << "Size " << sizes[s] << ": ";
    for (double time : sec_times[s]) {
        cout << time << " s, ";
    }
    cout << endl;
}

cout << "\n=== Swap Counts ===" << endl;
for (int s = 0; s < 8; ++s) {
    cout << "Size " << sizes[s] << ": ";
    for (auto swaps : swap_counts[s]) {
        cout << swaps << ", ";
    }
    cout << endl;
}

cout << "\n=== Full Passes ===" << endl;
for (int s = 0; s < 8; ++s) {
    cout << "Size " << sizes[s] << ": ";
    for (int passes : full_passes[s]) {
        cout << passes << ", ";
    }
    cout << endl;
}

cout << "\n=== Best Times (Seconds) ===" << endl;
for (int s = 0; s < 8; ++s) {
    cout << sizes[s] << "," << best_time[s] << endl;
}

cout << "\n=== Worst Times (Seconds) ===" << endl;
```

```
    for (int s = 0; s < 8; ++s) {
        cout << sizes[s] << "," << worst_time[s] << endl;
    }

    cout << "\n=== Average Times (Seconds) ===" << endl;
    for (int s = 0; s < 8; ++s) {
        cout << sizes[s] << "," << avg_time[s] << endl;
    }

    cout << "\n=== Average Swap Counts ===" << endl;
    for (int s = 0; s < 8; ++s) {
        cout << sizes[s] << "," << avg_swaps[s] << endl;
    }

    cout << "\n=== Average Full Passes ===" << endl;
    for (int s = 0; s < 8; ++s) {
        cout << sizes[s] << "," << avg_passes[s] << endl;
    }
    cout << "\n";
}

// Перегрузка оператора << для вывода Person
ostream& operator<<(ostream& os, const Person& p) {
    os << p.surname << " " << p.name << " " << p.patronymic
       << " (" << p.birthDate.day << "." << p.birthDate.month
       << "." << p.birthDate.year << ")";
    return os;
}

int main() {
    int choice;
    while (true) {
        cout << "Select test to run:\n";
        cout << "1. Numbers test\n";
        cout << "2. Strings test\n";
        cout << "3. Persons test\n";
        cout << "4. Shuffle test\n";
        cout << "5. Optimized Cocktail sort test\n";
        cout << "6. Exit\n";
        cout << "Enter choice (1-6): ";
        cin >> choice;

        if (choice == 6) break;

        switch (choice) {
            case 1:
                testNumbers();
                break;
            case 2:
                testStrings();
                break;
            case 3:
                testPersons();
```

```
            break;
        case 4:
            testShuffle();
            break;
        case 5:
            testCocktailSort();
            break;
        default:
            cout << "Invalid choice!\n";
    }
  }

  return 0;
}
```

Код реализует шаблонный класс DoublyLinkedList<T> — двусвязный список, где каждый узел содержит данные и указатели на предыдущий и следующий элементы. Класс включает методы для добавления (push_front, push_back, insert), удаления (remove), проверки пустоты (empty), получения размера (getSize) и итератор для обхода. Особенностью является оптимизированная коктейльная сортировка (cocktailSort) с использованием указателей, что ускоряет проходы по списку, и метод shuffle для перемешивания элементов.

Код содержит пять тестов: работа с числами, интерактивное управление строками, фильтрация персон, перемешивание чисел и измерение производительности сортировки для списков от 1000 до 128000 элементов. Реализация проста, надежна благодаря исключениям и деструктору, который освобождает память, и поддерживает любые типы данных.

Специфические моменты реализации:

- Оптимизированная коктейльная сортировка: Использование указателей left и right вместо индексов позволяет напрямую перемещаться по списку, что особенно эффективно для двусвязной структуры.
- Управление памятью: Деструктор автоматически освобождает все узлы, а локальная область видимости тестов гарантирует очистку памяти после каждого теста.

- Шаблонность: Класс поддерживает любые типы данных благодаря template<typename T>, что проверяется в тестах с int, string и Person.
- Исключения: Методы insert, remove и get выбрасывают исключения при некорректных индексах, обеспечивая надежность.
- Итератор: Простая реализация итератора позволяет использовать список в циклах for-range, что упрощает обход.
- Перемешивание: Метод shuffle опирается на get, что менее эффективно, чем прямое использование указателей, но выбрано для читаемости кода.

Выводы

В ходе выполнения лабораторной работы были реализованы двусвязный список и сортировка перемешиванием (шейкерная) на языке C++ и приведены проверки работы списка с помощью различных тестов (числовой, строковой, с использованием структур и сортировкой). Также было произведено сравнение сортировки двусвязного списка и массива.

Для наглядного представления была построена диаграмма на основе полученных значений в ходе одного тестового запуска программы, а также значений, использованных в первой лабораторной работе (рис. 1).

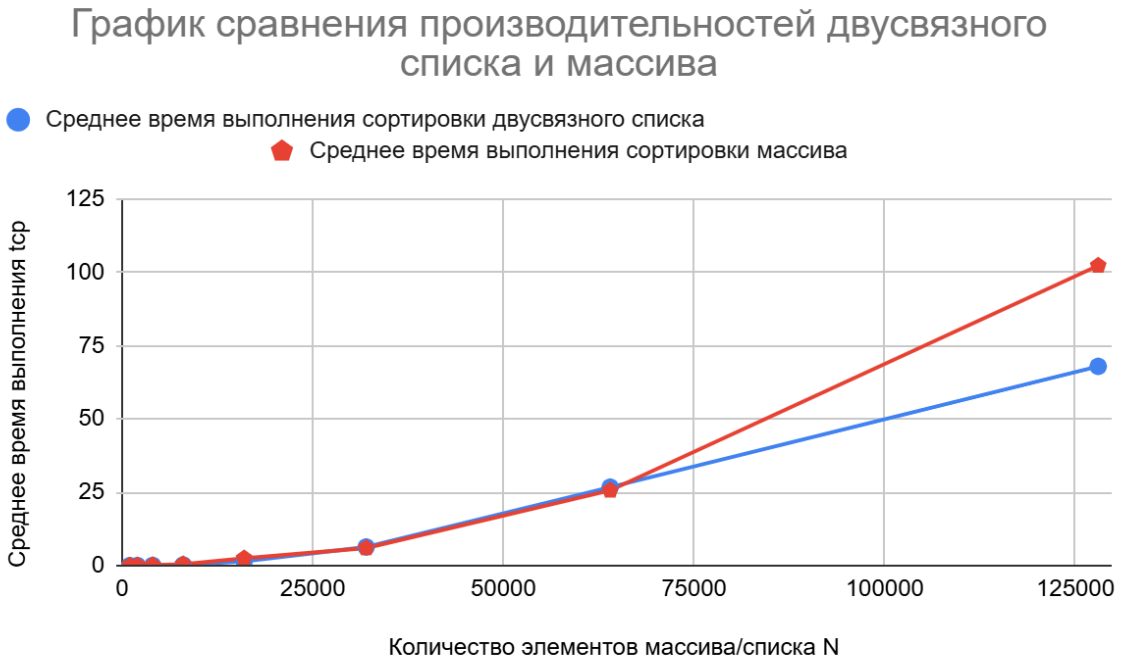

Рисунок 1 - График сравнения производительностей двусвязного списка и массива

На основании построенного графика можно подтвердить, что сортировка на двусвязном списке (синяя линия) работает быстрее, чем на массиве (красная линия), особенно когда элементов становится много. Это происходит, потому что в списке использованы указатели, которые позволяют быстро переходить между элементами, а массиву приходится дольше искать элементы по индексу.

Список сложнее сделать — нужно следить за указателями и выделять память для каждого элемента, в то время как массив проще, так как он встроен в язык и сразу готов к работе. Но список удобнее для добавления и удаления элементов, потому что не нужно копировать все данные, как в массиве. На больших данных разница в скорости становится очень заметной. В итоге список быстрее благодаря указателям, хотя его реализация требует больше усилий.

Анализ всех тестов демонстрирует, что двусвязный список надежно работает с числами, строками, структурами и справляется с сортировкой больших объемов данных. Тесты подтвердили его гибкость при добавлении и удалении элементов, а также эффективность благодаря использованию указателей. Реализация требует дополнительных усилий из-за управления памятью, но это оправдано высокой производительностью и универсальностью.