МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки

Студент(ка) гр. 9382 _____ Русинов Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить алгоритмы сортировки массива.

Задание.

17. Нитевидная сортировка.

Основные теоретические положения.

Сортировка – последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

Функции и структуры данных.

В качестве структуры данных был использован односвязный линейный список, поскольку нитевидная сортировка – сортирующий алгоритм для сортировки элементов списка.

struct List;

Поля:

1) List* next – указатель на следующий элемент списка
2) int value – значение узла

Методы:

1) void append(int element) – добавляет элемент в конец списка
2) int last() – возвращает значение последнего элемента спика
3) explicit List(int value) – конструктор
4) ~List() – деструктор

void showList(List* list) – функция для визуализации списка. Принимает сипсок.

List* merge(List* first, List* second) – функция для слияния двух списков, принимает два спика.

List* strandSort(List* list) – функция для нитевидной сортировки списка.

List* insert(int file = 0) – функция для считывания списка, аргумент – ввод из файла или из консоли. Если 0, то из консоли.

Описание алгоритма.

Проходим список и по пути отбираем элементы, формирующие упорядоченный подсписок. Производим слияние текущего упорядоченного подсписка с ранее полученным.

В качестве первого элемента подсписка выбирается первый элемент изначального списка. Затем каждый элемент изначального списка сравнивается с последним элементом подсписка. Если был найден элемент, который больше, чем последний элемент подсписка, то в подсписок добавляется в конец этот элемент и удаляется из начального списка. Затем сравнение идет уже с ним, и так до конца изначального списка.

После этого необходимо вызвать еще раз функцию сортировки над изначальным списком, поскольку из него исключены некоторые элементы. Затем нужно произвести слияние подсписка и отсортированного изначального списка. Алгоритм выполняется рекурсивно.

Эффективность – в среднем O(n^2). Однако в случае почти упорядоченного списка достигает O(n).

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	5 5 3 9 2 6	Ввод из файла? 0 - если нет: 1 [Список до сортировки] 5 3 9 2 6 [Слияние] 3 6 | 2 [Результат слияния] 2 3 6 [Слияние] 5 9 | 2 3 6 [Результат слияния] 2 3 5 6 9	

		[Результат сортировки] 2 3 5 6 9	
2.	10 5 3 9 2 6 0 -1 3 4 1	Ввод из файла? 0 - если нет: 1 [Список до сортировки] 5 3 9 2 6 0 -1 3 4 1 [Слияние] 0 1 | -1 [Результат слияния] -1 0 1 [Слияние] 2 3 4 | -1 0 1 [Результат слияния] -1 0 1 2 3 4 [Слияние] 3 6 | -1 0 1 2 3 4 [Результат слияния] -1 0 1 2 3 3 4 6 [Слияние] 5 9 | -1 0 1 2 3 3 4 6 [Результат слияния] -1 0 1 2 3 3 4 5 6 9 [Результат сортировки] -1 0 1 2 3 3 4 5 6 9	
3.	1 1	Ввод из файла? 0 - если нет: 0 Введите кол-во чисел: 1 [1] Введите число: 1 [Список до сортировки] 1 [Результат сортировки] 1	
4.	-1	Ввод из файла? 0 - если нет: 0 Введите кол-во чисел: -1 Кол-во чисел должно быть больше 0	Проверка на некорректных данных
5.	5 1 1 1 1 1	Ввод из файла? 0 - если нет: 0 Введите кол-во чисел: 5 [1] Введите число: 1 [2] Введите число: 1 [3] Введите число: 1 [4] Введите число: 1	

		[5] Введите число: 1 [Список до сортировки] 1 1 1 1 1 [Слияние] 1 | 1 [Результат слияния] 1 1 [Слияние] 1 | 1 1 [Результат слияния] 1 1 1 [Слияние] 1 | 1 1 1 [Результат слияния] 1 1 1 1 [Слияние] 1 | 1 1 1 1 [Результат слияния] 1 1 1 1 1 [Результат сортировки] 1 1 1 1 1	
6.	5 1 -1 1 -1 1	Ввод из файла? 0 - если нет: 0 Введите кол-во чисел: 5 [1] Введите число: 1 [2] Введите число: -1 [3] Введите число: 1 [4] Введите число: -1 [5] Введите число: 1 [Список до сортировки] 1 -1 1 -1 1 [Слияние] -1 1 | -1 1 [Результат слияния] -1 -1 1 1 [Слияние] 1 | -1 -1 1 1 [Результат слияния] -1 -1 1 1 1 [Результат сортировки] -1 -1 1 1 1	

Выводы.

Был изучен алгоритм нитевидной сортировки, была создана программа, которая сортирует список данным алгоритмом.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include "fstream"


struct List { // односвязный линейный список
    List* next = nullptr;
    int value;

    void append(int element) { // добавить элемент в конец
        List* current = this;
        while(current->next) current = current->next;
        current->next = new List(element);
    }

    int last() { // получить значение последнего элемента
        List* current = this;
        while (current->next) current = current->next;
        return current->value;
    }

    explicit List(int value) : value(value) {} // конструктор

    ~List() { // деструктор
        delete next;
    }
};

// функция для вывода списка
void showList(List* list) {
    while (list) {
        std::cout << list->value << " ";
        list = list->next;
    }
}


List* merge(List* first, List* second) { // слияние двух списков
    if (!first) return second; // если первый пустой, то вернуть второй
    if (!second) return first; // если второй пустой, то вернуть первый

    List* mergedList;

    List* headFirst = first; // фиксируем головы списков, чтобы в конце
функции очистить память
    List* headSecond = second;

    std::cout << "[Слияние] ";
    showList(first);
    std::cout << "| ";
    showList(second);
    std::cout << std::endl;
```

```
    if (first->value < second->value) { // инициализация головы списка
        mergedList = new List(first->value);
        first = first->next; // переход к следующему элементу
    } else {
        mergedList = new List(second->value);
        second = second->next;
    }

    while (first || second) { // пока списки не пустые
        if (first && second) { // если оба списка не пустые
            if (first->value < second->value) { // если значение первого
списка < второго
                mergedList->append(first->value); // тогда в результатив-
ный список добавляем значение первого
                first = first->next; // обрезаем первый список
            } else {
                mergedList->append(second->value); // аналогично
                second = second->next;
            }
        } else if (first) { // если непустой только первый, то добавляем
в результативный список элементы первого
            mergedList->append(first->value);
            first = first->next;
        } else {
            mergedList->append(second->value); // аналогия
            second = second->next;
        }
    }

    delete headFirst; // очистка памяти
    delete headSecond;

    std::cout << "[Результат слияния] ";
    showList(mergedList);
    std::cout << std::endl;

    return mergedList; // результат слияния
}

// Strand sort is a recursive sorting algorithm that sorts items of a
list into increasing order
// сортирущий алгоритм для сортировки элементов списка
// поэтому была реализована модель односвязного линейного списка
List* strandSort(List* list) { // нитевидная сортировка
    // в эту функцию нельзя передать пустой список
    if (!list->next) return list; // если список из одного элемента, то
возвращаем его

    List* sublist = new List(list->value); // промежуточный список
    list = list->next;
    // в него передается первый элемент списка
    // затем добавляются элементы из оригинального списка, которые больше
последнего элемента промежуточного списка

    List* source = nullptr;
    // данный список необходим, чтобы хранить элементы, которые <= по-
следнего элемента промежуточного списка
    // данный шаг упрощает работу с односвязным списком
```

```
    while (list) { // идем по элементам списка
        // если последний элемент промежуточного списка < элемента
списка, то добавляем элемент списка
        // в промежуточный список
        if (sublist->last() < list->value) sublist->append(list->value);
        else {
            // элемент списка <= последнего элемента промежуточного
списка
            // значит нужно добавить в source
            if (!source) source = new List(list->value); // инициализа-
ция, если source пустой
            else source->append(list->value);
        }
        // след элемент списка
        list = list->next;
    }

    // теперь есть промежуточный список и оригинальный без элементов про-
межуточного
    // оригинальный без элементов промежуточного нужно еще раз отсортиро-
вать
    // затем сделать слияние
    if (source) return merge(sublist, strandSort(source)); // если source
не пустой, то проводим его сортировку
    else return sublist; // иначе вернем промежуточный, он уже отсортиро-
ван
}

// ввод из консоли или из файла
List* insert(int file = 0) {
    int N;
    List* source = nullptr;

    if (!file) {
        std::cout << "Введите кол-во чисел: ";
        std::cin >> N; // получаем кол-во элементов
        if (N <= 0) {
            std::cout << "Кол-во чисел должно быть больше 0";
            return source;
        }

        for (int i = 0; i < N; ++i) { // считываем N элементов
            int k;
            std::cout << "[" << i+1 << "] " << "Введите число: ";
            std::cin >> k;

            if (!source) source = new List(k);
            else source->append(k); // Добавляем в список
        }
    } else {
        std::ifstream input("file.txt"); // открываем файл
        if (!input.is_open()) { // проверяем на доступность
            input.close();
            std::cout << "Не удалось открыть файл file.txt" << std::endl;
            return source;
        }
        else {
```

```
            if(!(input >> N)) { // считываем N
                std::cout << "Не удалось считать кол-во элементов" <<
std::endl;
                return nullptr;
            }

            if (N <= 0) {
                std::cout << "Кол-во чисел должно быть больше 0";
                return source;
            }

            for (int i = 0; i < N; ++i) { // Считываем N элементов
                int k;
                if(!(input >> k)) { // проверяем, получилось ли считать
                    std::cout << "Задано N чисел, но было введено меньше
N чисел" << std::endl;
                    delete source;
                    return nullptr; // если не удалось, вызываем деструк-
тор списка
                }
                if (!source) source = new List(k); // инициализация
списка
                else source->append(k); // добавляем элемент
            }
        }
    }
    return source;
}

int main() {
    int file;
    std::cout << "Ввод из файла? 0 - если нет: ";
    std::cin >> file;
    List* source = insert(file); // считывание списка
    if (!source) return 0;

    std::cout << "[Список до сортировки] ";
    showList(source);
    std::cout << std::endl;
    List* result = strandSort(source); // сортировка
    if (result != source) delete source;
    std::cout << "[Результат сортировки] ";
    showList(result); // отображение списка
    delete result;

    return 0;
}
```