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: Алгоритмы сортировки

Студент гр. 9382 _____ Кузьмин Д. И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить принципы алгоритмов сортировки массива. Освоить навыки разработки программ, осуществляюших сортировку.

Основные теоретические положения.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. Задача сортировки состоит в перестановке членов последовательности таким образом, чтобы выполнялось условие: ai <= ai+1, для всех i от 0 до n.

Задание.

Вариант 12. Быстрая сортировка – итеративная реализация.

Описание функций и структур данных

1) В качестве сортируемого объекта используется вектор типа int std:vector<int>.

2) Описаны некоторые шаблонные функции. Такие как

3) int divide(vector<T>& a, int start, int end) — где a — сортируемый вектор, start – начало сортируемой части, end – ее конец. Возвращает индекс «опорного элемента»(см. Описание алгоритма)

4) void iterativeQuickSort(vector<T>& a) — функция, реализующая быструю сортировку. В качестве аргумента принимает вектор объектов любого типа. Возвращает void.

5) void readVector(vector<T>& v, FILE* f = stdin) — считывание вектора. V – ссылка на вектор, f – указатель на поток ввода(по умолчанию стандартный поток ввода). Возвращает void.

6) void printArray(vector<T> a, int start, int end) – вывод вектора. A - сам вектор. Start и end – границы вывода, то есть с какого по какой элемент вывести. Возвращает void.

Описание алгоритма

1) Для быстрой сортировки вводится понятие опорного элемента. То есть элеменета, с которым будет проводится сравнения в сортируемых частях.

В данной реализации опорным элементом будет выбираться последний из сортируемой части.

2) В функции для перестановки элемента проводятся сравнения с опорным элементов и в зависимости от результата этих сравнений формируется новый подмассив. В нем все элемнеты, стоящие перед опорным — меньше(или равны) его, а стояще после него — больше.

3) В самой функции сортировки неотсортированные части хранятся в виде стека (типа std::stack) из объектов типа std::pair, в которых лежат начало и конец частей.

4) Первым шагом в стек помещается сам массив, то есть пара начало — 0 и конец — индекс последнего элемента. Далее проводится перестановка.

5) Затем, если после(или до) опорного лежат >= 2 элементов, то эти части считаются неотсортированными и индексы их начала и конца кладутся в стек.

6) На каждой итерации проводятся шаги 2,3,4,5 для элементов стека. Если стек становится пустым — цикл и алгоритм завершается.

7) Среднее время — O(n*log(n))

8) Плюс алгоритма по сравнению с рекурсивной реализацией состоит в том, что здесь исключается переполнение стека засчет многочисленного вызова рекурсивных функций.

Исходный код см. в приложении А.

Тестирование.

Результаты тестирование представлены в табл. 1.

Таблица 1 — результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарий
1	6 -1 3 14 9 -11 5	-11 -1 3 5 6 9 14	Совпадает с результатом библиотечной сортировки
2	0 6 -6 2 14 33	-6 0 2 6 14 33	Совпадает с результатом библиотечной сортировки
3	13 22 0 4 -4 3 100	-4 0 3 4 13 22 100	Совпадает с результатом библиотечной сортировки
4	17 6 -2 34 0 0 0	-2 0 0 0 6 17 34	Совпадает с результатом библиотечной сортировки
5	0 1 2 3	0 1 2 3	Совпадает с результатом библиотечной сортировки
6	16 -20 34 0 11 55	-20 0 11 16 34 55	Совпадает с результатом библиотечной сортировки
7	0 0 0	0 0 0	Совпадает с результатом библиотечной сортировки
8	34 -2 19 0 6 -30 1 2 3	-30 -2 0 1 2 3 6 19 34	Совпадает с результатом библиотечной сортировки

Выводы.

Был изучен алгоритм быстрой сортировки. Получены навыки разработки программы, реализующей этот алгоритм.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <stack>
#include <vector>
#include <algorithm>
#include <cstring>
#include <sstream>

using namespace std;

template <typename T>
void printArray(vector<T> a, int start, int end) {
	cout << "{";
	for (int i = start; i < end; i++) {
		cout << a[i] << ", ";
	}
	cout << a[end] << "}";
}
template <typename T>
int divide(vector<T>& a, int start, int end) //функция перестановки
элементов относительно некоторого опорного
{
	static int count = 0;
	T pivot = a[end]; //опорный элемент для последующих сравнений


	int pIndex = start; //вспомогательный индекс, который показывает
положение опорного элемента

	cout << "Перестановка элементов подмассива: ";
	printArray(a, start, end);
	cout << " относительно " << pivot << "\n";

	for (int i = start; i < end; i++)
	{
		count++;
		cout << "Сравнение " << a[i] << " и " << pivot << "\n";
		if (a[i] <= pivot)
		{
			cout << a[i] << " <= " << pivot << ", поэтому элемент
переходит в левую половину.\n";
			swap(a[i], a[pIndex]);

			pIndex++;
		}
		else {

			cout << a[i] << " > " << pivot << ", поэтому элемент
переходит в правую половину.\n";


		}

	}
```

```
	// После выполнения всех сравнений элемент со вспомогательный индексом меняется местами с опорным элементом
	// В полученном подмассиве все элементы, стоящие до опорного меньше его, а после опорного больше.
	swap(a[pIndex], a[end]);
	cout << "Результат перестановки: ";
	printArray(a, start, end);
	cout << "\n\nИсходный массив: ";
	printArray(a, 0, a.size() - 1);
	cout << "\n";
	// Функция возвращает индекс опорного элемента.

	return pIndex;
}
template <typename T>
// Непосредственно функция сортировки
void iterativeQuickSort(vector<T>& a)
{

	//для хранения подмассивов используется стек. элементы в нем имеют тип пары, в которой хранится
	//начала и конец сортируемых подмассивов
	stack<pair<int, int>> stk;

	int start = 0;
	int end = a.size() - 1;
	stk.push(make_pair(start, end));

	// цикл будет выполняться, пока стек не станет пустым
	while (!stk.empty())
	{

		// на каждой итерации обрабытвается верхний элемент стека
		start = stk.top().first;
		end = stk.top().second;
		stk.pop();

		// перестановка элементов подмассива и получение индекса опорного элемента
		int pivot = divide(a, start, end);
		if ( end - start <= 1){
			cout << "Часть ";
			printArray(a, start, end);
			cout << " отсортирована\n\n";
			}
		else {
			//если в сортируемом подмассиве до(или после) опорного элемента находятся
			//минимум два элемента - подмассив, содержащий их также кладется в стек
			bool c1 = pivot - start >= 2;
			bool c2 = end - pivot >= 2;


			if (c1) {
				stk.push(make_pair(start, pivot - 1));
				cout << "Часть ";
				printArray(a, start, pivot - 1);
```

```
                    cout << " может быть не отсортирована\n";
                }
                else {
                    cout << "Часть ";
                    printArray(a, start, pivot);
                    cout << " отсортирована\n";
                }
                //помещение в стек подмассива левее опорного

                if (c2) {
                    cout << "Часть ";
                    printArray(a, pivot + 1, end);
                    cout << " может быть не отсортирована\n";
                    stk.push(make_pair(pivot + 1, end));
                }
                else {
                    cout << "Часть ";
                    printArray(a, pivot, end);
                    cout << " отсортирована\n\n";
                }
            }
                //правее опорного


    }
}
template <typename T>
void readVector(vector<T>& v, FILE* f = stdin) { //считывание массива
    stringstream ss;
    char c = fgetc(f);
    int spacecount = 1;
    while (c != '\n' && c != EOF) {
        ss << c;
        if (isspace(c))spacecount++;
        c = fgetc(f);
    }

    for (int i = 0; i < spacecount; i++) {
        T a;
        ss >> a;
        v.push_back(a);
    }

}

int main()
{
    vector<int> v;
    setlocale(LC_ALL, "Russian");
    char* a = new char[20];
    char* c = new char[300];
    while (strcmp(a, "1\n") && strcmp(a, "2\n")) {
        std::cout << "Сортировка массива. Введите 1 для ввода с
консоли и 2 для ввода с файла.\n";
        fgets(a, 20, stdin);
    }
    if (strcmp(a, "1\n") == 0) {
        std::cout << "Введите элементы массива через пробел:";
```

```
            readVector<int>(v,  stdin);

        }
        else {
            FILE* f1 = fopen("test.txt", "r+");
            std::cout << "Содержимое файла: ";

            while (fgets(c, 300, f1)) std::cout << c;
            f1 = fopen("test.txt", "r+");
            readVector<int>(v, f1);
            std::cout << "\n";
            fclose(f1);

        }
        iterativeQuickSort(v);
        cout << "Отсортированный массив: ";
        for (int i = 0; i < v.size(); i++) {
            cout << v[i] << " ";
        }
        cout << "\n";
        return 0;
}
```