**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: Типы данных, определяемые пользователем. Структуры

|  |  |  |
| --- | --- | --- |
| Студент гр. 4373 |  | Шепелев Д.Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение и организация массивов данных; получение практических навыков работы со массивами; изучение сортировок.

**Основные теоретические положения.**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами. Чаще всего ограничиваются тем, как структуры представлены в языке С:

struct [имя\_типа] {

тип\_1 элемент\_1;

тип \_2 элемент\_2;

…

тип\_k элемент\_k;

} [ список\_описателей ];

Описание структуры начинается ключевым словом struct. Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных. Их количество не лимитировано.

Вся эта конструкция является инструкцией языка программирования, поэтому после нее всегда должен ставиться символ ‘;’.

При описании структуры память для размещения данных не выделяется. Работать с описанной структурой можно только после того, как будет определена переменная (переменные) этого типа данных, только при этом компилятор выделит необходимую память.

Для инициализации структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

struct complex{

float real, im;

} data [2][2] = {

{{1,1}, {2,2}},

{{3,3}, {4,4}}

};

Все поля структурных переменных располагаются в непрерывной области памяти одно за другим. Общий объем памяти, занимаемый структурой, равен сумме размеров всех полей структуры. Для определения размера структуры следует использовать инструкцию sizeof().

Для того чтобы записать данные в структурную переменную, необходимо каждому полю структуры присвоить определенное значение. Для этого необходимо использовать оператор ‘’ («точка»):

struct Stack { // Cтек

float arr[100];

short topIndex;

};

…

Stack stack; // Объявляем переменную типа Stack

Stack.arr[0] = 1;

…

При доступе к определенному полю его следует рассматривать как обычную переменную, тип данных которой соответствует типу этого поля. Поля структур могут участвовать в качестве операндов любых выражений, допускающих использование операндов соответствующего типа данных.

Копирование данных из одной структурной переменной в другую осуществляется простой операцией присваивания, независимо от количества полей и размера структуры (это можно делать только в том случае, когда обе переменные одного и того же типа).

В программировании очень часто используются такие конструкции, как массивы структур. Например, сведения о студентах некоторой учебной группы можно хранить в массиве студентов:

t\_Student Gruppa\_N [30];

Был определен 30-элементный массив, каждый элемент которого предназначен для хранения данных одного студента. Получение доступа к данным некоторого студента из группы *N* осуществляется обычной индексацией переменной массива. Поскольку поля структуры могут быть любого типа данных, то они в свою очередь могут быть другой структурой или массивом других структур:

struct Stud

{

char FN[100];

short listNumber;

};

struct Group

{

int groupNumber;

short students;

Stud stud[30];

};

Но в структуре поля нельзя использовать элемент, тип которого совпадает с типом самой структуры, так как рекурсивное использование структур запрещено.

Любая структурная переменная занимает в памяти определенное положение, характеризующееся конкретным адресом. Для работы с адресами структурных переменных (как и для простых переменных) можно использовать указатели. Указатели на структурные переменные определяются точно так же, как и для обычных переменных. Разыменование указателя (обращение к данным по адресу, хранящемуся в указателе) осуществляется также обычным образом.

Через указатели можно работать с отдельными полями структур. Для доступа к полю структуры через указатель используется оператор ‘’ («стрелка»), а не «точка».

Структуры можно использовать в качестве параметров функций, как и обычные переменные. Для структур поддерживаются все три механизма передачи данных: по значению, через указатели и по ссылке.

Передачу структур в функции по значению необходимо использовать аккуратно:

void WriteStudent ( t\_Student S )

{

cout << "Фамилия: " << S.Fam << endl;

cout << "Имя: " << S.Name << endl;

cout << "Год рождения: " << S.Year << endl;

if ( S.Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S.Grade << endl;

}

Вызов такой функции сопровождается дополнительным расходом памяти для создания локальной переменной *S*и дополнительными затратами времени на физическое копирование данных из аргумента в параметр *S*. Учитывая то, что объем структур может быть очень большим, эти дополнительные затраты вычислительных ресурсов могут быть чрезмерными.

Предпочтительно использование передачи структуры по указателю или ссылке:

void WriteStudent ( t\_Student \*S )

{

cout << "Фамилия: " << S -> Fam << endl;

cout << "Имя: " << S -> Name << endl;

cout << "Год рождения: " << S -> Year << endl;

if ( S -> Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S -> Grade << endl;

}

Фактической передачи данных в функцию не осуществляется. Дополнительные затраты памяти для создания локальной переменной небольшие – это адрес памяти (4 байта, независимо от размера самой структуры). Вызов такой функции будет происходить быстрее, а расход памяти будет существенно меньше, чем при передаче данных по значению.

Передача по ссылке по эффективности эквивалентна передаче данных через указатель. Однако, поскольку при передаче данных по ссылке все адресные преобразования берет на себя компилятор, существенно упрощается программирование действий со структурами. При использовании ссылочных параметров структурных типов доступ к членам структуры осуществляется обычным способом – с помощью оператора «точка».

Недостатком этих способов является то, что случайные изменения значений полей структуры внутри функции отразятся на значении аргумента после окончания работы функции. Если необходимо предотвратить изменения переданных по адресу аргументов, можно при определении соответствующего параметра объявить его константой (использовать спецификатор const).

**Постановка задачи.**

Необходимо написать программу, которая:

1)    Создает целочисленный массив размерности *N* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2)    Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

3)    Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

4)    Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитайте время поиска.

5)    Выводит количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

6)    Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

7)    Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. (\*)

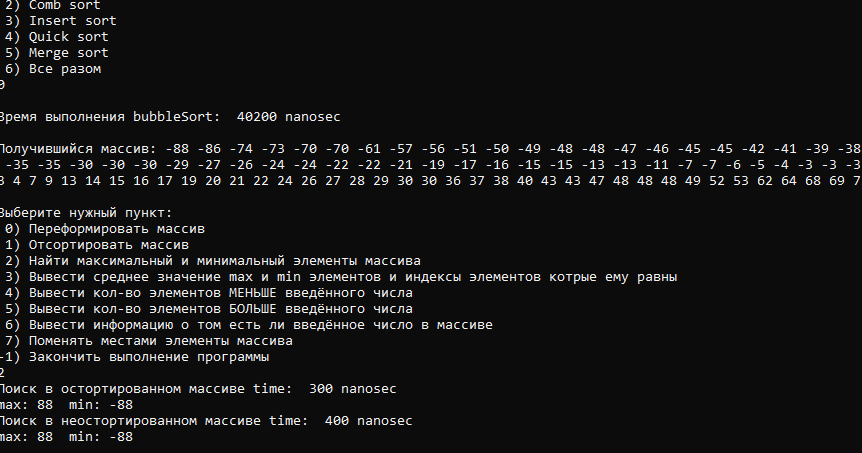
8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

Должна присутствовать возможность запуска каждого пункта многократно.

**Выполнение работы.**

Код программы представлен в приложении А.

Пример работы кода :



**Выводы.**

Я изучил массивы, получил практические навыки работы с ними и определил преимущества и недостатки разных сортировок.

Приложение А

#include <iostream>

#include <chrono>

using namespace std;

void equatingArr(int arr[], int sourceArr[], const int sizeArr) {

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

arr[i] = sourceArr[i];

}

}

void printArr(int arr[], const int sizeArr) {

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

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

}

}

void createArr(int arr[], const int sizeArr) {

srand(time(0));

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

arr[i] = (rand() % sizeArr) - (rand() % sizeArr);

}

}

void findMaxAndMinElArr(int arr[], int unsortArr[], const int sizeArr) {

int max, min;

auto timeStart1 = chrono::steady\_clock::now();

max = arr[sizeArr - 1];

min = arr[0];

auto timeEnd1 = chrono::steady\_clock::now();

cout << "Поиск в остортированном массиве time: " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd1 - timeStart1).count() << " nanosec\n" << "max: " << max << " min: " << min << endl;

auto timeStart2 = chrono::steady\_clock::now();

max = -sizeArr;

min = sizeArr;

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

if (unsortArr[i] > max) {

max = unsortArr[i];

}

if (unsortArr[i] < min) {

min = unsortArr[i];

}

}

auto timeEnd2 = chrono::steady\_clock::now();

cout << "Поиск в неостортированном массиве time: " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd2 - timeStart2).count() << " nanosec\n" << "max: " << max << " min: " << min << endl;

}

void findAverageValue(int arr[], int unsortArr[], const int sizeArr) {

int max, min, average, count = 0, flag = 0;

auto timeStart1 = chrono::steady\_clock::now();

max = arr[sizeArr - 1];

min = arr[0];

average = (max + min) / 2;

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

if (arr[i] == average) {

//cout << i << " ";

++count;

flag = 1;

}

else if (flag == 1) {

break;

}

}

auto timeEnd1 = chrono::steady\_clock::now();

flag = 0;

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

if (arr[i] == average) {

cout << i << " ";

flag = 1;

}

else if (flag == 1) {

break;

}

}

cout << " : индексы элементов равных среднему, всего их: " << count << endl;

cout << "Поиск и вывод индексов в остортированном массиве time: " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd1 - timeStart1).count() << " nanosec\n" << "среднее от max и min: " << average << endl;

auto timeStart2 = chrono::steady\_clock::now();

count = 0;

average = 0;

max = -sizeArr;

min = sizeArr;

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

if (unsortArr[i] > max) {

max = unsortArr[i];

}

if (unsortArr[i] < min) {

min = unsortArr[i];

}

}

average = (max + min) / 2;

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

if (unsortArr[i] == average) {

++count;

}

}

auto timeEnd2 = chrono::steady\_clock::now();

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

if (unsortArr[i] == average) {

cout << i << " ";

}

}

cout << " : индексы элементов равных среднему, всего их: " << count << endl;

cout << "Поиск и вывод индексов в неостортированном массиве time: " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd2 - timeStart2).count() << " nanosec\n" << "среднее от max и min: " << average << endl;

}

void findLessThenNumb(int arr[], int number, const int sizeArr) {

int count = 0;

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

if (arr[i] < number) {

++count;

}

else {

break;

}

}

cout << "\nКол-во элементов меньших чем введёное число: " << count;

}

void findMoreThenNumb(int arr[], int number, const int sizeArr) {

int count = 0;

for (int i = sizeArr - 1; i >= 0; --i) {

if (arr[i] > number) {

++count;

}

else {

break;

}

}

cout << "\nКол-во элементов больших чем введёное число: " << count;

}

int binarySearch(int arr[], int number, int start , int end) {

int rem = 0;

if ((end - start) / 2 == 0) {

rem = 1;

}

int middleNumb = arr[start + (end - start) / 2 + rem];

if (middleNumb == number) {

return start + (end - start) / 2 + rem;

}

else if (start + 1 == end || start == end + 1) {

return -1;

}

if (middleNumb > number) {

end = start + (end - start) / 2 + rem;

binarySearch(arr, number, start, end);

}

else{

start = start + (end - start) / 2 + rem;

binarySearch(arr, number, start, end);

}

}

int simpleSearch(int arr[], int number, const int sizeArr) {

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

if (arr[i] == number) {

return i;

}

}

return -1;

}

void tryFindEl(int arr[], int number, const int sizeArr) {

int idEl;

auto timeStart1 = chrono::steady\_clock::now();

idEl = binarySearch(arr, number, 0, sizeArr - 1);

auto timeEnd1 = chrono::steady\_clock::now();

if (idEl == -1) {

cout << "Такого элемента нет в массиве\n";

}

else{

cout << "ID этого элемента: " << idEl << endl;

}

cout << "Бинарный поиск time: " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd1 - timeStart1).count() << " nanosec\n";

auto timeStart2 = chrono::steady\_clock::now();

idEl = simpleSearch(arr, number, sizeArr);

auto timeEnd2 = chrono::steady\_clock::now();

if (idEl == -1) {

cout << "Такого элемента нет в массиве\n";

}

else{

cout << "ID этого элемента: " << idEl << endl;

}

cout << "Обычный поиск time: " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd2 - timeStart2).count() << " nanosec\n";

}

void swapElTime(int arr[], int firstElId, int secondElId) {

auto timeStart = chrono::steady\_clock::now();

swap(arr[firstElId], arr[secondElId]);

auto timeEnd = chrono::steady\_clock::now();

cout << "swap time: " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd - timeStart).count() << " nanosec\n";

}

template <const int sizeArr>

void bubbleSort(int arr[]) {

bool isChanged = true;

int end = sizeArr - 1;

while (isChanged == true) {

isChanged = false;

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

if (arr[i + 1] < arr[i]) {

swap(arr[i + 1], arr[i]);

isChanged = true;

}

}

--end;

}

}

template <const int sizeArr>

void shakerSort(int arr[]) {

bool isChanged = true;

int start = 0, end = sizeArr - 1;

while (isChanged == true) {

isChanged = false;

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

if (arr[i + 1] < arr[i]) {

swap(arr[i + 1], arr[i]);

isChanged = true;

}

}

--end;

if (!isChanged) {

break;

}

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

if (arr[i - 1] > arr[i]) {

swap(arr[i - 1], arr[i]);

isChanged = true;

}

}

++start;

}

}

template <const int sizeArr>

void combSort(int arr[]) {

float k = 1.247;

bool isChanged = true;

int end = sizeArr - 1, gap = sizeArr - 1;

while (gap != 1) {

for (int i = 0; i < sizeArr - gap; ++i) {

if (arr[i] > arr[i + gap]) {

swap(arr[i + gap], arr[i]);

}

}

gap /= k;

}

while (isChanged == true) {

isChanged = false;

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

if (arr[i + 1] < arr[i]) {

swap(arr[i + 1], arr[i]);

isChanged = true;

}

}

--end;

}

}

template <const int sizeArr>

void insertSort(int arr[]) {

int end = sizeArr - 1, stepCount;

for (int start = 1; start <= end; ++start) {

stepCount = 0;

while (arr[start - stepCount] < arr[start - 1 - stepCount]) {

swap(arr[start - stepCount], arr[start - 1 - stepCount]);

++stepCount;

}

}

}

void quickSortSep(int arr[], int start, int end){

int pivot = start;

++start;

if (start >= end) {

return;

}

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

if (arr[i] < arr[pivot]) {

swap(arr[start], arr[i]);

++start;

}

}

swap(arr[pivot], arr[start-1]);

quickSortSep(arr, pivot, start - 2);

quickSortSep(arr, start, end);

}

template <const int sizeArr>

void quickSortMain(int arr[]) {

quickSortSep(arr, 0, sizeArr - 1);

}

void mergeSortStep(int arr[], const int sizeArr, int start, int end, int twoSortArr[]) {

int parity = (end - start + 1) % 2;

if (end - start <= 1) {

if (end - start == 1 && arr[end] < arr[start]) {

swap(arr[start], arr[end]);

}

return;

}

mergeSortStep(arr, sizeArr, start, (end + start) / 2 - parity, twoSortArr);

mergeSortStep(arr, sizeArr, (end + start) / 2 + 1 - parity, end, twoSortArr);

int ind = 0;

int lArrIndex = start, rArrIndex = (end + start) / 2 + 1 - parity;

for (int i = 0; i < (end - start + 1);++i) {

if (lArrIndex <= (end + start) / 2 - parity && rArrIndex <= end) {

if (arr[lArrIndex] > arr[rArrIndex]) {

twoSortArr[ind] = arr[rArrIndex];

++rArrIndex;

++ind;

}

else {

twoSortArr[ind] = arr[lArrIndex];

++lArrIndex;

++ind;

}

}

else if (rArrIndex > end) {

twoSortArr[ind] = arr[lArrIndex];

++lArrIndex;

++ind;

}

else {

twoSortArr[ind] = arr[rArrIndex];

++rArrIndex;

++ind;

}

}

ind = 0;

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

arr[i] = twoSortArr[ind];

++ind;

}

}

template <const int sizeArr>

void mergeSortMain(int arr[]) {

int twoSortArr[sizeArr];

mergeSortStep(arr, sizeArr, 0, sizeArr - 1, twoSortArr);

}

void leadTime(void (\*func)(int[]), const char name[], int arr[]) {

auto timeStart = chrono::steady\_clock::now();

func(arr);

auto timeEnd = chrono::steady\_clock::now();

cout << "Время выполнения " << name << ": " << chrono::duration\_cast<chrono::nanoseconds>(timeEnd - timeStart).count() << " nanosec\n";

}

int main() {

setlocale(0, "");

constexpr int sizeArr = 100;

int choise, arr[sizeArr], unsortArr[sizeArr], inputNumber, firstElId, secondElId;

bool isSorted = false;

cout << "Исходный массив: ";

createArr(unsortArr, sizeArr);

printArr(unsortArr, sizeArr);

equatingArr(arr, unsortArr, sizeArr);

do {

if (isSorted) {

cout << "\n\nВыберите нужный пункт:"

<< "\n 0) Переформировать массив"

<< "\n 1) Отсортировать массив"

<< "\n 2) Найти максимальный и минимальный элементы массива"

<< "\n 3) Вывести среднее значение max и min элементов и индексы элементов котрые ему равны"

<< "\n 4) Вывести кол-во элементов МЕНЬШЕ введённого числа"

<< "\n 5) Вывести кол-во элементов БОЛЬШЕ введённого числа"

<< "\n 6) Вывести информацию о том есть ли введённое число в массиве"

<< "\n 7) Поменять местами элементы массива"

<< "\n-1) Закончить выполнение программы\n";

cin >> choise;

switch (choise)

{

case(0): // Переформировать массвив

cout << "Новый исходный массив: ";

isSorted = false;

createArr(unsortArr, sizeArr);

printArr(unsortArr, sizeArr);

break;

case(1): // Отсортировать массив

cout << "\nВыберите выриант сортировки:"

<< "\n 0) Bubble sort"

<< "\n 1) Shaker sort"

<< "\n 2) Comb sort"

<< "\n 3) Insert sort"

<< "\n 4) Quick sort"

<< "\n 5) Merge sort"

<< "\n 6) Все разом\n";

cin >> choise;

isSorted = true;

switch (choise) {

case 0:

cout << "\n";

leadTime(bubbleSort<sizeArr>,"bubbleSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 1:

cout << "\n";

leadTime(shakerSort<sizeArr>, "shakerSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 2:

cout << "\n";

leadTime(combSort<sizeArr>, "combSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 3:

cout << "\n";

leadTime(insertSort<sizeArr>, "insertSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 4:

cout << "\n";

leadTime(quickSortMain<sizeArr>, "quickSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 5:

cout << "\n";

leadTime(mergeSortMain<sizeArr>, "mergeSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 6:

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(bubbleSort<sizeArr>, "bubbleSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(shakerSort<sizeArr>, "shakerSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(combSort<sizeArr>, "combSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(insertSort<sizeArr>, "insertSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(quickSortMain<sizeArr>, "quickSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(mergeSortMain<sizeArr>, "mergeSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

}

break;

case(2): // Найти максимальный и минимальный элементы массива

findMaxAndMinElArr(arr, unsortArr, sizeArr);

break;

case(3): // Вывести среднее значение max и min элементов и индексы элементов котрые ему равны

findAverageValue(arr, unsortArr, sizeArr);

break;

case(4): // Вывести кол-во элементов МЕНЬШЕ введённого числа

cout << "Введите число: ";

cin >> inputNumber;

findLessThenNumb(arr, inputNumber, sizeArr);

break;

case(5): // Вывести кол-во элементов БОЛЬШЕ введённого числа

cout << "Введите число: ";

cin >> inputNumber;

findMoreThenNumb(arr, inputNumber, sizeArr);

break;

case(6): // Вывести информацию о том есть ли введённое число в массвиве

cout << "Введите число: ";

cin >> inputNumber;

tryFindEl(arr, inputNumber, sizeArr);

break;

case(7): // Поменять местами элементы массива

cout << "Введите через пробел номера двух элементов массива: ";

cin >> firstElId >> secondElId;

swapElTime(arr, firstElId, secondElId);

cout << "Полученный массив: ";

printArr(arr, sizeArr);

isSorted = false;

break;

}

}

else {

cout << "\n\nВыберите нужный пункт:"

<< "\n 0) Переформировать массив"

<< "\n 1) Отсортировать массив"

<< "\n 7) Поменять местами элементы массива"

<< "\n-1) Закончить выполнение программы\n";

cin >> choise;

switch (choise)

{

case(0): // Переформировать массвив

cout << "Новый исходный массив: ";

isSorted = false;

createArr(unsortArr, sizeArr);

printArr(unsortArr, sizeArr);

break;

case(1): // Отсортировать массив

cout << "\nВыберите выриант сортировки:"

<< "\n 0) Bubble sort"

<< "\n 1) Shaker sort"

<< "\n 2) Comb sort"

<< "\n 3) Insert sort"

<< "\n 4) Quick sort"

<< "\n 5) Merge sort"

<< "\n 6) Все разом\n";

cin >> choise;

isSorted = true;

switch (choise) {

case 0:

cout << "\n";

leadTime(bubbleSort<sizeArr>, "bubbleSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 1:

cout << "\n";

leadTime(shakerSort<sizeArr>, "shakerSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 2:

cout << "\n";

leadTime(combSort<sizeArr>, "combSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 3:

cout << "\n";

leadTime(insertSort<sizeArr>, "insertSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 4:

cout << "\n";

leadTime(quickSortMain<sizeArr>, "quickSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 5:

cout << "\n";

leadTime(mergeSortMain<sizeArr>, "mergeSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

case 6:

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(bubbleSort<sizeArr>, "bubbleSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(shakerSort<sizeArr>, "shakerSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(combSort<sizeArr>, "combSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(insertSort<sizeArr>, "insertSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(quickSortMain<sizeArr>, "quickSort", arr);

equatingArr(arr, unsortArr, sizeArr);

cout << "\n";

leadTime(mergeSortMain<sizeArr>, "mergeSort", arr);

cout << "\nПолучившийся массив: ";

printArr(arr, sizeArr);

break;

}

break;

case(7): // Поменять местами элементы массива

cout << "Введите через пробел номера двух элементов массива: ";

cin >> firstElId >> secondElId;

swapElTime(arr, firstElId, secondElId);

cout << "Полученный массив: ";

printArr(arr, sizeArr);

isSorted = false;

break;

}

}

} while (choise != -1);

return 0;

}