**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: Одномерные статические массивы

|  |  |  |
| --- | --- | --- |
| Студент гр. 2372 |  | Тубшинов В.Т. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение массивов: объявление массивов, ввод-вывод массивов, различные сортировки массивов; получение практических навыков работы с одномерными статическими массивами; определение времени, затраченного на сортировку массива разными способами.

**Основные теоретические положения.**

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение. Каждый элемент в этих массивах определяется значением индекса элемента.

Все массивы можно разделить на две группы: *одномерные* и *многомерные*. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

<Базовый тип элементов> <Идентификатор массива> [<Количество элементов>]

Значения индексов элементов массивов всегда начинается с 0. Поэтому максимальное значение индекса элемента в массиве всегда на единицу меньше количества элементов в массиве.

Ввод и вывод массивов реализуются с помощью циклов.

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке.

Сортировка обменом (bubble sort). Вся суть метода заключается в попарном сравнении элементов и последующем обмене. Таким образом, если следующий элемент меньше текущего, то они меняются местами, максимальный элемент массива постепенно смещается в конец массива, а минимальный – в начало. Один полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент. Затем процесс повторяется до тех пор, пока вся последовательность не будет упорядочена. Важно заметить, что после первого прохода по массиву, уже имеется один упорядоченный элемент, он стоит на своем месте, и менять его не надо. Таким образом на следующем шаге будут сравниваться *N*-1 элемент.

Шейкер-сортировка массива (shaker sort)– модификация пузырьковой сортировки. Принцип работы этой сортировки аналогичен bubble sort: попарное сравнение элементов и последующий обмен местами. Но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм не начинает новую итерацию с первого элемента, а запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно.Затем процесс повторяется до тех пор, пока массив не будет отсортирован.

Сортировка расческой (comb sort) – алгоритм был разработан специально для случаев, когда минимальные элементы стоят слишком далеко, или максимальные – слишком близко к началу массива. В сортировке расческой переставляются элементы, стоящие на расстоянии. Оптимально изначально взять расстояние равным длине массива, а далее уменьшать его на определенный коэффициент, который примерно равен 1.247. Когда расстояние станет равно 1, выполняется обычная сортировка пузырьком.

Сортировка вставками (insert sort) – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов. Сортировка вставками делить массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться. Причем найти нужное место для очередного элемента в отсортированном массиве достаточно легко.

Алгоритм бинарного поиска – классический алгоритм поиска элемента в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден.

**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)    Отсортировывает заданный в пункте 1 массив сортировкой (bubble sort, shaker sort, insert sort, comb sort) от меньшего к большему. Определяет время, затраченное на сортировки, используя библиотеку chrono.

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

4)    Выводит среднее значение максимального и минимального значения в отсортированном и неотсортированном массивах. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитывает время поиска.

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

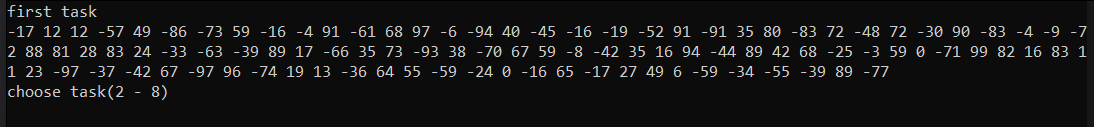
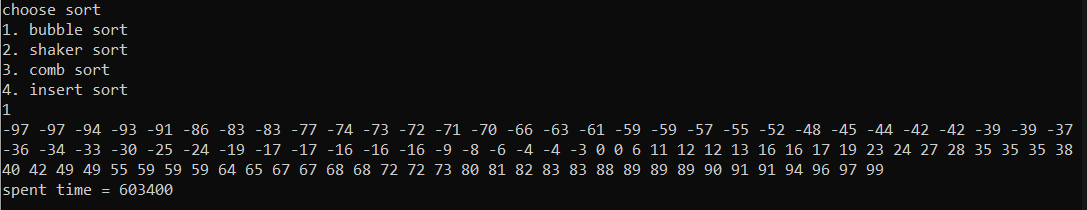
6)    Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

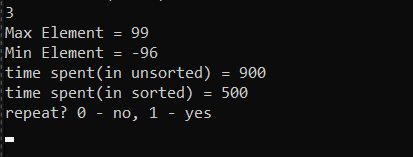
7)    Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве (реализация бинарным поиском). Сравнивает скорость его работы с обычным перебором.

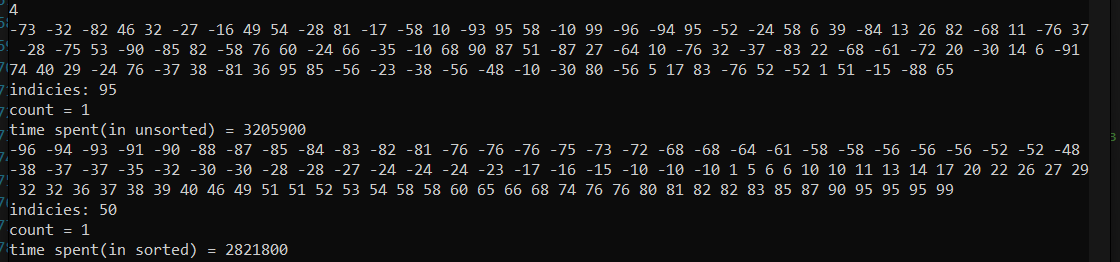
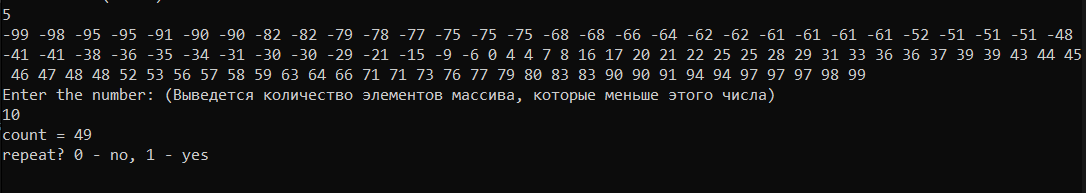
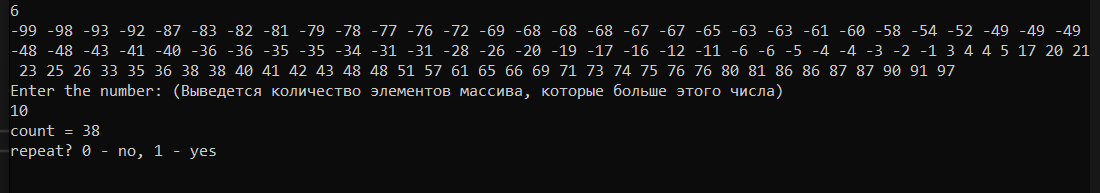
8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выводит скорость обмена, используя библиотеку chrono.

**Выполнение работы.**

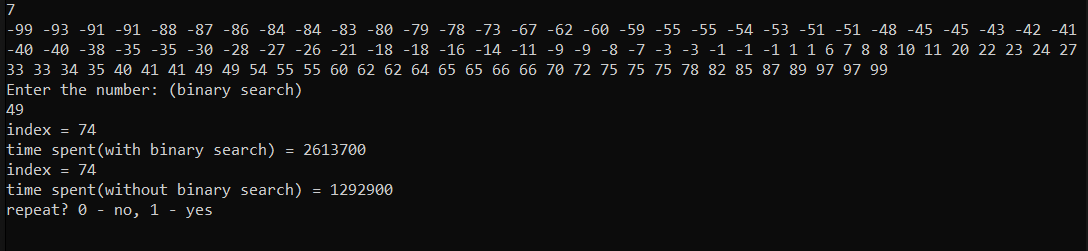
Код программы представлен в приложении А.

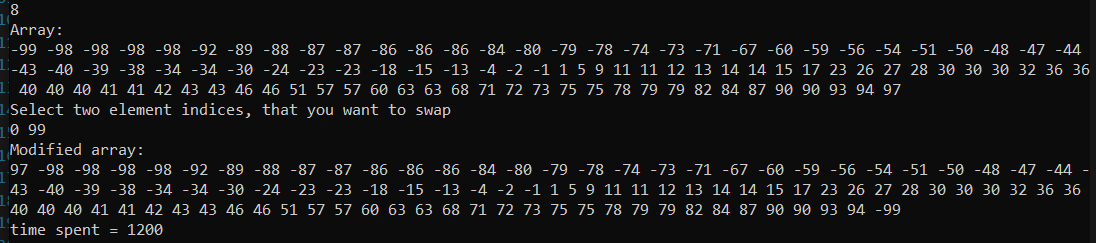
1. При запуске программы пользователю выводится неотсортированный массив из 100 элементов, которые принимают случайное значение в диапазоне от -99 до 99.
2. Далее на экран выводится отсортированный массив из п.1., с помощью выбранной сортировки и потраченной на эту сортировку время
3. Далее выводится максимальный и минимальный элементы массива и время поиска этих элементов в отсортированном и неотсортированном массивах

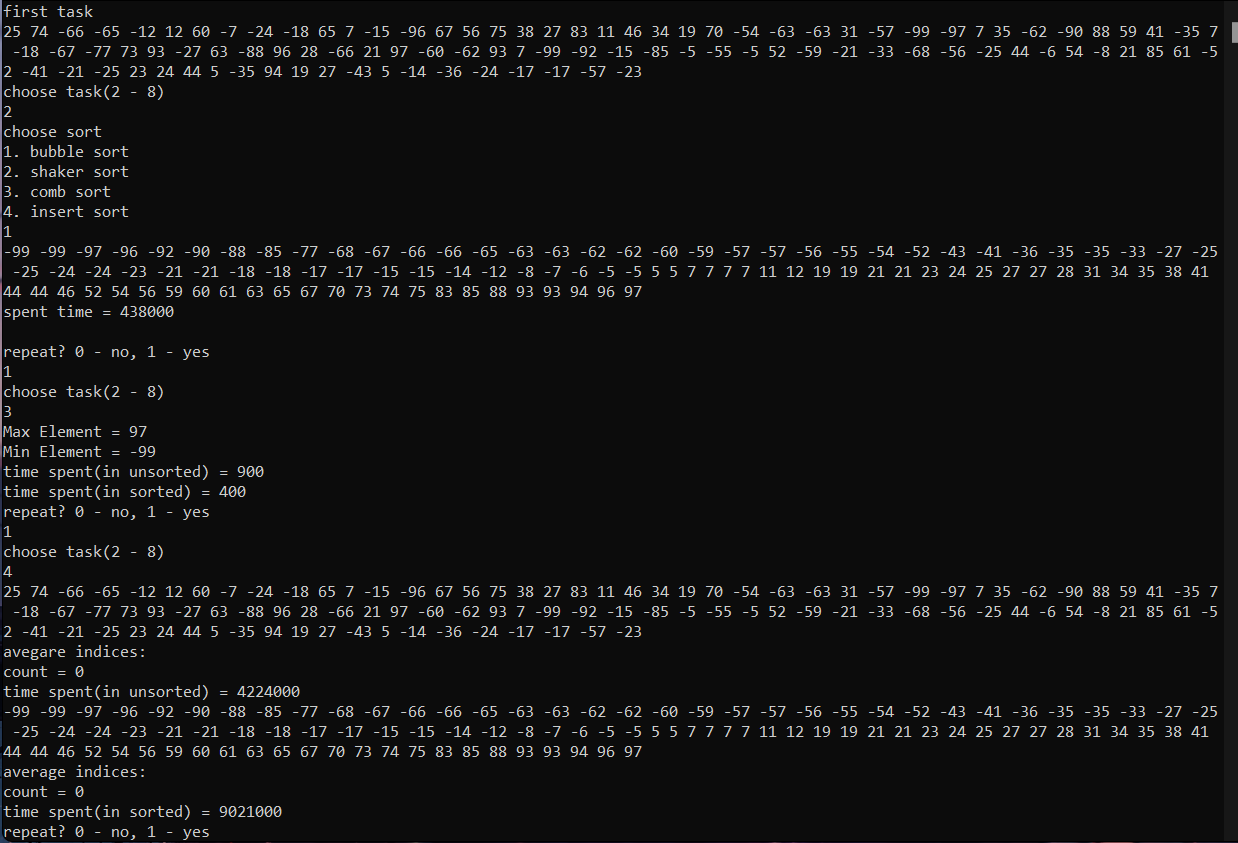


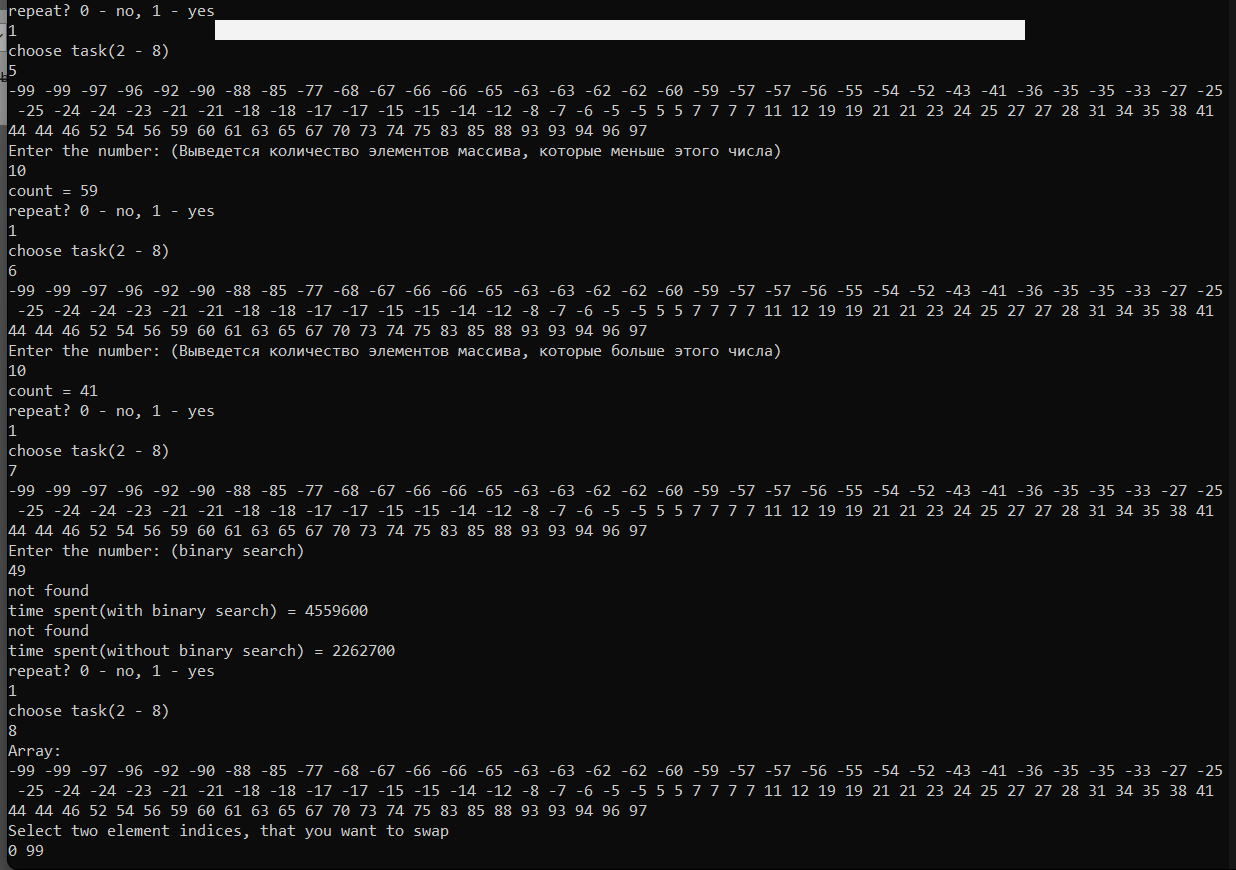
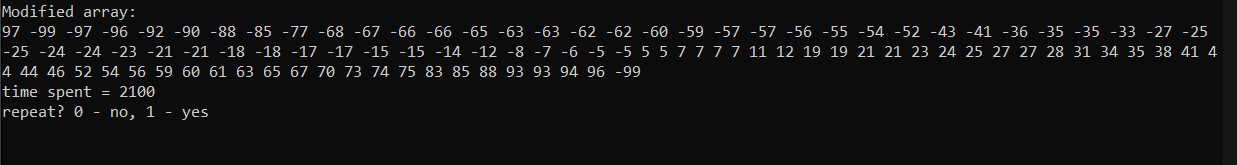
1. После этого выводятся среднее значение максимального и минимального значения в отсортированном и неотсортированном массивах. А также выводятся индексы всех элементов, которые равны этому значению, и их количество, подсчёт времени поиска.
2. Далее пользователю предлагается ввести число. Программа выводит количество элементов в отсортированном массиве, которые меньше этого числа.
3. После, пользователю предлагается ввести другое число. Программа выводит количество элементов в отсортированном массиве, которые больше этого числа.
4. Дальше пользователю предлагается ввести число. Программа с помощью бинарного поиска определяет, есть ли это число в массиве, и выводит индекс этого элемента в отсортированном массиве (при наличии элемента в массиве).

Если числа в массиве нет, то выводит «not found». То же самое с обычным перебором элементов. Сравнивается скорость работы этих алгоритмов.



1. После этого пользователю предлагается ввести два индекса, элементы которых в массиве (отсортированном) нужно поменять местами. Для наглядности на экран выводится весь массив с уже перемещёнными элементами. Также выводится время скорости обмена элементов.

 Общий результат кода с примерами:



**Выводы.**

В ходе работы было изучено: объявление массивов, ввод-вывод массивов, сортировки массивов: bubble sort, shaker sort, insert sort, comb sort; определение времени, затраченного на сортировку массива разными способами.

Приложение А

рабочий код

#include <iostream>

#include <chrono>

#include <ctime>

using namespace std;

using namespace chrono;

void MakeArray(int \*array, int size) {

srand(time(NULL));

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

array[i] = rand() % 199 - 99;

}

}

void PrintArray(int array[], int size) {

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

cout << array[i] << " ";

}

cout << "\n";

}

void SaveValues(int \*array, int \*constArray, int size) {

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

array[i] = constArray[i];

}

}

void BubbleSort(int array[], int size) {

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

for (int j = 0; j < size - 1 - i; j++) {

if (array[j] > array[j + 1]) {

swap(array[j], array[j + 1]);

}

}

}

}

void ShakerSort(int array[], int size) {

int low = 0, up = size - 1, last;

while (low < up) {

last = -1;

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

if (array[i] > array[i + 1]) {

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

last = i;

}

}

up = last;

if (last == -1) {

break;

}

last = size;

for (int i = up; i >= low; i--) {

if (array[i] > array[i + 1]) {

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

last = i;

}

}

low = last + 1;

}

}

void InsertionSort(int array[], int size) {

for (int step = 1; step < size; step++) {

int key = array[step];

int j = step - 1;

while (key < array[j] && j >= 0) {

array[j + 1] = array[j];

--j;

}

array[j + 1] = key;

}

}

void CombSort(int array[], int size) {

float k = 1.247, S = size - 1;

int count = 0;

while (S >= 1)

{

for (int i = 0; i + S < size; i++)

{

if (array[i] > array[int(i + S)])

{

swap(array[i], array[int(i + S)]);

}

}

S /= k;

}

while (true)

{

for (int i = 0; i < size - 1; i++)

{

if (array[i] > array[i + 1])

{

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

}

else count++;

}

if (count == size - 1)

break;

else

count = 0;

}

}

int main() {

setlocale(0, "");

auto start = steady\_clock::now(); //таймер

auto end = steady\_clock::now();

auto result = duration\_cast <nanoseconds> (end - start);

const int size = 100; //размер массива

int array[size]; //массив

int answer = 1; // для цикла

int maxElement, minElement; //максимальный и минимальный элементы массива для задания ..

int number = 0; // для заданий с введенным числом пользователем

int count = 0; // счетчик

int low = 0, high = 0; //верхняя и нижняя границы для бинарного поиска и шейкер сорта

int last; // для шейкер сорта

int average = 0; // среднее

int firstIndex, secondIndex; // хз что это

int isFound = 0; // флаг, который определяет найден ли элемент 1 - да 0 - нет

float k = 1.247, S = size - 1; // дляя комб сорта

int constArr[size]; //хранит в себе значение сделанного в пункте 1 массива.

cout << "first task\n";

MakeArray(array, size); // первое задание сделать массив и вывести его на экран

PrintArray(array, size);

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

constArr[i] = array[i];

}

do {

int task; // выбора задания

cout << "choose task(2 - 8)\n";

cin >> task;

switch (task) {

case 2:

int variant; // для выбора варианта сортировки

cout << "choose sort \n";

cout << "1. bubble sort" << "\n" << "2. shaker sort" << "\n" << "3. comb sort" << "\n" << "4. insert sort" << "\n";

cin >> variant;

switch (variant) {

case 1: //бабл сорт

srand(time(0));

SaveValues(array, constArr, size);

start = steady\_clock::now();

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

for (int j = 0; j < size - 1 - i; j++) {

if (array[j] > array[j + 1]) {

swap(array[j], array[j + 1]);

}

}

}

end = steady\_clock::now();

PrintArray(array, size);

result = duration\_cast <nanoseconds> (end - start);

cout << "spent time = " << result.count() << endl;

cout << "\n";

break;

case 2: //шейкер сорт

srand(time(0));

low = 0;

high = size - 1;

SaveValues(array, constArr, size);

start = steady\_clock::now();

while (low < high) {

last = -1;

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

if (array[i] > array[i + 1]) {

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

last = i;

}

}

high = last;

if (last == -1) {

break;

}

last = size;

for (int i = high; i >= low; i--) {

if (array[i] > array[i + 1]) {

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

last = i;

}

}

low = last + 1;

}

end = steady\_clock::now();

PrintArray(array, size);

result = duration\_cast <nanoseconds> (end - start);

cout << "spent time = " << result.count() << endl;

break;

case 3: //комб сорт

srand(time(0));

SaveValues(array, constArr, size);

start = steady\_clock::now();

while (S >= 1)

{

for (int i = 0; i + S < size; i++)

{

if (array[i] > array[int(i + S)])

{

swap(array[i], array[int(i + S)]);

}

}

S /= k;

}

while (true)

{

for (int i = 0; i < size - 1; i++)

{

if (array[i] > array[i + 1])

{

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

}

else count++;

}

if (count == size - 1)

break;

else

count = 0;

}

end = steady\_clock::now();

PrintArray(array, size);

result = duration\_cast <nanoseconds> (end - start);

cout << "spent time = " << result.count() << endl;

break;

case 4: //инсерт сорт

srand(time(0));

SaveValues(array, constArr, size);

start = steady\_clock::now();

for (int step = 1; step < size; step++) {

int key = array[step];

int j = step - 1;

while (key < array[j] && j >= 0) {

array[j + 1] = array[j];

--j;

}

array[j + 1] = key;

}

end = steady\_clock::now();

PrintArray(array, size);

result = duration\_cast <nanoseconds> (end - start);

cout << "spent time = " << result.count() << endl;

break;

default: break;

}

break;

case 3: //нахождение максимального и минимального элемента в массиве

srand(time(0));

SaveValues(array, constArr, size);

maxElement = array[0], minElement = array[0];

start = steady\_clock::now();

for (int i = 1; i < size - 1; i++) {

if (array[i] > maxElement) {

maxElement = array[i];

}

if (array[i] < minElement) {

minElement = array[i];

}

}

end = steady\_clock::now();

result = duration\_cast <nanoseconds> (end - start);

cout << "Max Element = " << maxElement << "\n";

cout << "Min Element = " << minElement << "\n";

cout << "time spent(in unsorted) = " << result.count() << endl;

ShakerSort(array, size);

srand(time(0));

start = steady\_clock::now();

maxElement = array[0];

minElement = array[size - 1];

end = steady\_clock::now();

result = duration\_cast <nanoseconds> (end - start);\

cout << "time spent(in sorted) = " << result.count() << endl;

break;

case 4: //среднее значение максимального и мнимального эллементов и индексы элементов массива с таким значением и количество

srand(time(0));

count = 0;

SaveValues(array, constArr, size);

PrintArray(array, size);

maxElement = array[0];

minElement = array[0];

start = steady\_clock::now();

for (int i = 1; i < size; i++) {

if (array[i] > maxElement) {

maxElement = array[i];

}

if (array[i] < minElement) {

minElement = array[i];

}

}

average = (maxElement + minElement) / 2;

cout << "avegare indices: ";

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

if (array[i] == average) {

cout << i << " ";

count ++;

}

}

cout << "\n";

cout << "count = " << count << "\n"; // выводит количество элементов равных среднему максимального и минимального

end = steady\_clock::now();

result = duration\_cast <nanoseconds> (end - start);

cout << "time spent(in unsorted) = " << result.count() << endl;

count = 0;

ShakerSort(array, size);

PrintArray(array, size);

start = steady\_clock::now();

minElement = array[0];

maxElement = array[size - 1];

average = (maxElement + minElement) / 2;

cout << "average indices: ";

//for (int i = 0; i < size; i++) {

low = 0;

high = size - 1;

isFound = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (array[mid] == average) {

isFound = mid;

}

if (array[mid] < average)

low = mid + 1;

else

high = mid - 1;

}

if (isFound != -1) {

for (int i = isFound - 1; i >=0 && array[i] == average; i--) {

count++;

cout << " ";

}

for (int i = isFound;i < size && array[i] == average; i++) {

count++;

cout << i << " ";

}

}

cout << "\n";

cout << "count = " << count << "\n";

end = steady\_clock::now();

result = duration\_cast <nanoseconds> (end - start);

cout << "time spent(in sorted) = " << result.count() << endl;

break;

case 5: //количество элементов меньше введенного числа

InsertionSort(array, size);

PrintArray(array, size);

cout << "Enter the number: (Выведется количество элементов массива, которые меньше этого числа)\n";

cin >> number;

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

if (number > array[i]) {

count += 1;

}

else {

break;

}

}

cout << "count = ";

cout << count << endl;

break;

case 6: //количество элементов больше введенного числа

count = size;

InsertionSort(array, size);

PrintArray(array, size);

cout << "Enter the number: (Выведется количество элементов массива, которые больше этого числа)\n";

cin >> number;

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

if (number < array[i]) {

break;

}

count -= 1;

}

cout << "count = ";

cout << count << endl;

break;

case 7: //бинарный поиск

ShakerSort(array, size);

PrintArray(array, size);

cout << "Enter the number: (binary search)" << endl;

cin >> number;

start = steady\_clock::now();

low = 0;

high = 99;

isFound = 0;

while (low <= high) {

int mid = low + (high - low) / 2;

if (array[mid] == number) {

isFound = mid;

}

if (array[mid] < number)

low = mid + 1;

else

high = mid - 1;

}

if (isFound) {

cout << "index = " << isFound << endl;

}

else {

cout << "not found" << endl;

}

end = steady\_clock::now();

result = duration\_cast <nanoseconds> (end - start);

cout << "time spent(with binary search) = " << result.count() << endl;

start = steady\_clock::now();

count = 0;

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

if (array[i] == number) {

cout << "index = " << i << endl;

count += 1;

break;

}

}

if (!count) {

cout << "not found\n";

}

end = steady\_clock::now();

result = duration\_cast <nanoseconds> (end - start);

cout << "time spent(without binary search) = " << result.count() << endl;

break;

case 8: //меняет элементы массива, индексы которых вводит пользователь

srand(time(0));

InsertionSort(array, size);

cout << "Array:\n";

PrintArray(array, size);

cout << "Select two element indices, that you want to swap\n";

cin >> firstIndex >> secondIndex;

start = steady\_clock::now();

swap(array[firstIndex], array[secondIndex]);

end = steady\_clock::now();

result = duration\_cast <nanoseconds> (end - start);

cout << "Modified array:\n";

PrintArray(array, size);

cout << "time spent = " << result.count() << endl;

break;

}

cout << "repeat? 0 - no, 1 - yes\n";

cin >> answer;

} while (answer);

}