**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных Систем и Технологий (ИСИТ)**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: «ОДНОМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ»

|  |  |  |
| --- | --- | --- |
| Студент гр. XXXX |  | Стамбровский Т.С |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

20XX

**Цель работы.**

Ознакомление с основными особенностями одномерных статических массивов, изучение правил описания массивов, выполнение простейших действий с массивами, реализация различных алгоритмов, связанных с массивами.

**Основные теоретические положения.**

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов.

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.

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.

Аналогом одномерного массива из математики может служить последовательность некоторых элементов с одним индексом: a\_i*ai*​ при  i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных. Наглядно одномерный массив можно представить как набор пронумерованных ячеек, в каждой из которых содержится определенное значение:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 3.02 | 1.5 | 7.0 | -2.3 | 12.0 |

Это пример одномерного массива из 5 элементов, каждый из которых представляет собой некоторое вещественное значение и каждое из этих значений имеет индекс от 0 до 4.

А вот пример одномерного массива из десяти элементов, представляющих собой одиночные символы:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| ‘a’ | ‘b’ | ‘c’ | ‘+’ | ‘1’ | ‘2’ | ‘!’ | ‘#’ | ‘@’ | ‘&’ |

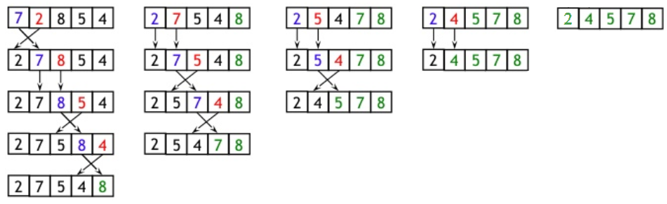
Каждый элемент в этих массивах определяется значением индекса элемента. Например, в последнем массиве элемент с индексом 5 равен символу  ‘2’.

### ****Пузырьковая сортировка массива (bubble sort)****

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Простейшая из сортировок – сортировка обменом (пузырьковая сортировка). 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.



Очевидно, что хуже всего алгоритм будет работать, когда на вход подается массив, отсортированный в обратную сторону (от большего к меньшу). Быстрее же всего алгоритм работает с уже отсортированным массивом.

Но стандартный алгоритм пузырьковой сортировки предполагает полный циклический проход по массиву. Если изначально подается упорядоченная последовательность, то работа алгоритма все равно продолжиться. Исправить это можно, добавив условие проверки: если на текущей итерации ни один элемент не изменил свой индекс, то работа алгоритма прекращается.

Алгоритм бинарного поиска – классический алгоритм поиска 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.

Рассмотрим простой пример: имеется массив из 100 элементов, упорядоченных по возрастанию от 1 до 100. Было загадано какое-то число, необходимо его назвать. Компьютер имеет три ответа на ваше предположение: верно, число больше, число меньше. Сколько попыток нужно, чтобы ответить правильно? Обычный перебор – наихудшая стратегия. Можно назвать правильный ответ лишь с 100-ой попытки.

Но если начать спрашивать с середины, то ситуация кардинально меняется. Если число больше 50, то необходимо делить правую половину, и следующее предположение – 75, если меньше – 25. Так необходимо продолжать до тех пор, пока не будет названо правильное число. Наибольшее число предположений равняется:

k= ⌈ log\_2 100⌉=7*k*=⌈*log*2​100⌉=7

Этот результат явно лучше простого перебора. Схожий принцип работы имеет алгоритм бинарного поиска.

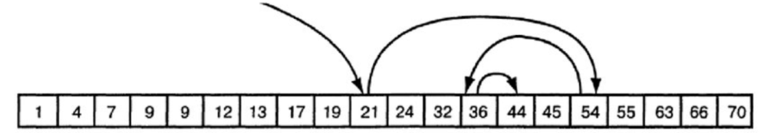
Бинарный поиск работает только в топ случае, если список отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1)    Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2)    Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3)    Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4)    Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.



Чтобы уменьшить количество шагов поиска, можно сразу смещать границы поиска на элемент, следующий за серединой отрезка.

**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 элементы массива […] 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. Выводит индексы всех элементов, которые равны этому значению, и их количество.

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.

Таблица – Варианты сортировок

|  |  |
| --- | --- |
| Номер варианта | Название сортировки |
| 1 | Bubble sort (пузырьковая сортировка) |
| 2 | Shaker sort (шейкер-сортировка) |
| 3 | Comb sort (сортировка расчёской) |
| 4 | Insert sort (сортировка вставками) |
| 5 (\*) | Quick sort (быстрая сортировка) |

**Выполнение работы.**

Поочередно выполняем поставленные задачи, применяя на практике знания о статических одномерных массивах, при необходимости используем библиотеку chrono, определяя время, затраченное на операции. Форматируем, добавляем цвет, создаем контекстное меню, внося в него пункты выполненных задач при помощи конструкции switch.

**Выводы.**

Были освоены новые приемы работы с массивами, изучены незнакомые библиотеки, просчитано время обработки различных операций с массивами, на основании которых можно сделать определенные выводы (бинарный поиск позволяет в разы ускорить работу, по сравнению с линейным поиском и т. д.)

Приложение А

Полный код программы

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <chrono>

#include <Windows.h>

using namespace std;

int main() {

HANDLE N = GetStdHandle(STD\_OUTPUT\_HANDLE);

setlocale(LC\_ALL, "Russian");

int arr[100];

int max, min;

int answer;

answer = 0;

max = arr[0];

min = 0;

srand(time(NULL));;

for (int i = 0; i < 100; i++) {

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

}

while (answer != 9) {

SetConsoleTextAttribute(N, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "Введите номер задания: " << endl;

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "1. Создает целочисленный массив размерности N = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99." << endl;

cout << "2. Отсортировать заданный в пункте 1 элементы массива […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono." << endl;

cout << "3. Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono." << endl;

cout << "4. Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество." << endl;

cout << "5. Выводит количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем." << endl;

cout << "6. Выводит количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем." << endl;

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

cout << "8. Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono." << endl;

cout << "Чтобы выйти, введите (9)" << endl;

cin >> answer;

system("CLS");

switch (answer)

{

case (1): {

SetConsoleTextAttribute(N, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "Неотсортированный массив: " << endl;

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

for (int i = 0; i < 100; i++) {

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

}

break;

}

case (2): {

int h;

auto start2 = chrono::high\_resolution\_clock::now();

for (int i = 0; i < 100; i++) {

for (int j = 0; j < 99 - i; j++) {

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

h = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = h;

}

}

}

SetConsoleTextAttribute(N, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "Отсортированный массив: " << endl;

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

for (int i = 0; i < 100; i++)

{

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

}

auto end2 = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration2 = end2 - start2;

cout.setf(ios::fixed, ios::floatfield);

cout.precision(7);

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Время, потраченное на сортировку: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << duration2.count() << endl;

cout << endl;

break;

}

case (3): {

int h;

auto start1 = chrono::high\_resolution\_clock::now();

for (int i = 0; i < 100; i++) {

if (arr[i] > max)

{

max = arr[i];

}

if (arr[i] < min)

{

min = arr[i];

}

}

auto end1 = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration1 = end1 - start1;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Минимальный элемент: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << min << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Максимальный элемент: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << max << endl;

cout.setf(ios::fixed, ios::floatfield);

cout.precision(7);

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Время, потраченное на поиск max и min (в неотсортированном массиве): ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << duration1.count() << endl;

cout << endl;

for (int i = 0; i < 100; i++) {

for (int j = 0; j < 99 - i; j++) {

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

h = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = h;

}

}

}

auto start3 = chrono::high\_resolution\_clock::now();

for (int i = 0; i < 100; i++) {

if (arr[i] > max)

{

max = arr[i];

}

if (arr[i] < min)

{

min = arr[i];

}

}

auto end3 = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration3 = end3 - start3;

cout.setf(ios::fixed, ios::floatfield);

cout.precision(7);

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Время, потраченное на поиск max и min (в отсортированном массиве): ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << duration3.count() << endl;

cout << endl;

break;

}

case 4: {

for (int i = 0; i < 100; i++) {

if (arr[i] > max)

{

max = arr[i];

}

if (arr[i] < min)

{

min = arr[i];

}

}

int sr;

int am1 = 0;

sr = (max + (-min)) / 2;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Среднее значение max и min: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << sr << endl;

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Индексы элементов, равных среднему значению: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

for (int i = 0; i < 100; i++) {

if ((arr[i] == sr) || (-arr[i] == sr)) {

cout << i << " ";

am1++;

}

}

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Количество элементов, равных среднему значению: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << am1;

cout << endl;

cout << endl;

break;

}

case (5): {

int a;

int am2 = 0;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Введите число a: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cin >> a;

for (int i = 0; i < 100; i++) {

if (arr[i] < a) {

am2++;

}

}

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Количество элементов, которые меньше чем a: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << am2;

cout << endl;

break;

}

case (6): {

int b;

int am3 = 0;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Введите число b: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cin >> b;

for (int i = 0; i < 100; i++) {

if (arr[i] > b) {

am3++;

}

}

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Количество элементов, которые больше чем b: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << am3;

cout << endl;

break;

}

case(8): {

int mn1, mn2, mn3;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Введите индексы элементов, которые необходимо поменять местами: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

int h;

for (int i = 0; i < 100; i++) {

for (int j = 0; j < 99 - i; j++) {

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

h = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = h;

}

}

}

auto start4 = chrono::high\_resolution\_clock::now();

cin >> mn1;

cin >> mn2;

mn3 = arr[mn1];

arr[mn1] = arr[mn2];

arr[mn2] = mn3;

auto end4 = chrono::high\_resolution\_clock::now();

for (int i = 0; i < 100; i++) {

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

}

chrono::duration<float> duration4 = end4 - start4;

cout.setf(ios::fixed, ios::floatfield);

cout.precision(7);

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Время, потраченное на обмен переменных: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << duration4.count() << endl;

cout << endl;

break;

}

case (7): {

int h;

for (int i = 0; i < 100; i++) {

for (int j = 0; j < 99 - i; j++) {

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

h = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = h;

}

}

}

int n;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Введите искомый элемент: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cin >> n;

bool fl = false;

int left = 0;

int right = 99;

int middle;

auto start5 = chrono::high\_resolution\_clock::now();

while ((left <= right) && (fl != true)) {

middle = (left + right) / 2;

if (arr[middle] == n) fl = true;

if (arr[middle] > n) right = middle - 1;

else left = middle + 1;

}

auto end5 = chrono::high\_resolution\_clock::now();

SetConsoleTextAttribute(N, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "///////////////////////////////////" << endl;

cout << "Бинарный поиск: " << endl;

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

if (fl == 1) {

cout << "Элемент ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << n;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << " находится под номером ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << middle << endl;

}

else cout << "В массиве этого элемента нет.";

chrono::duration<float> duration5 = end5 - start5;

cout.setf(ios::fixed, ios::floatfield);

cout.precision(7);

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Время, потраченное на бинарный поиск: ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << duration5.count() << endl;

cout << endl;

int key = 0;

SetConsoleTextAttribute(N, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "///////////////////////////////////" << endl;

cout << "Линейный поиск: " << endl;

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

auto start6 = chrono::high\_resolution\_clock::now();

for (int i = 0; i < 100; i++) {

if (n == arr[i]) {

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Элемент ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << n;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << " находится под номером ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << i << endl;

key = 1;

}

}

if (key == 0) {

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "В массиве этого элемента нет.";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

}

auto end6 = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration6 = end6 - start6;

cout.setf(ios::fixed, ios::floatfield);

cout.precision(7);

cout << endl;

SetConsoleTextAttribute(N, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "Время, потраченное на линейный поиск (перебором): ";

SetConsoleTextAttribute(N, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << duration6.count() << endl;

cout << endl;

cout << endl;

break;

}

}

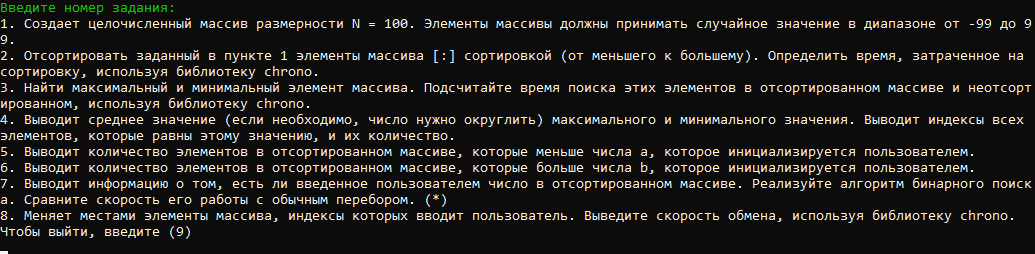
}

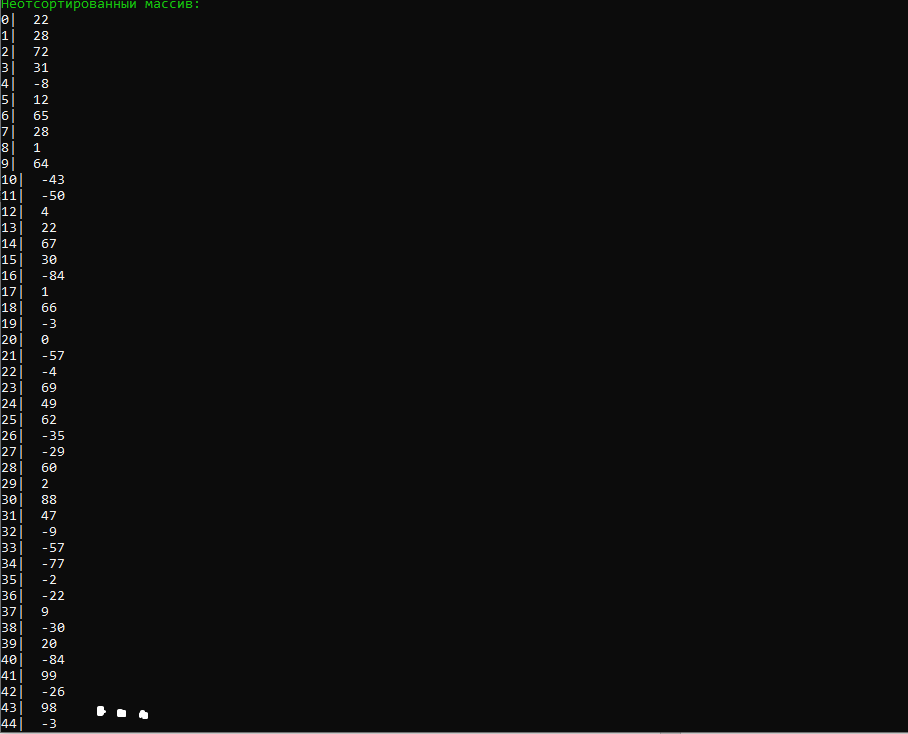
return 0;

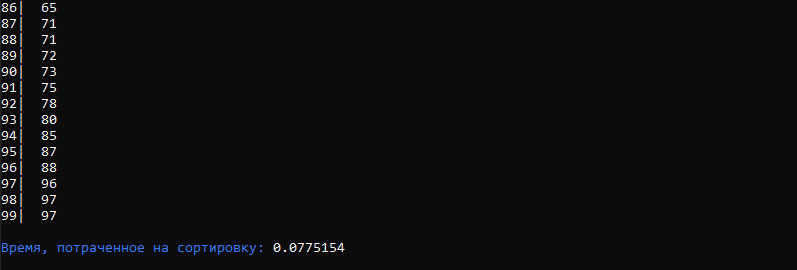
}

Приложение б

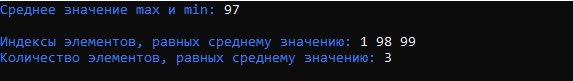
Полученные результаты

1)

2)

3) 

4)



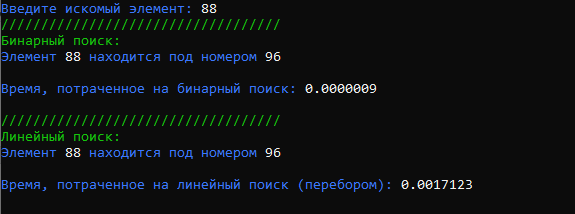
5)



6)



7)



8)

