МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра «Информационные системы»

**отчет**

по практической работе №2

по дисциплине «Программирование»

Тема: "Одномерные статические массивы"

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Голубцов В.В. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

Цель работы.

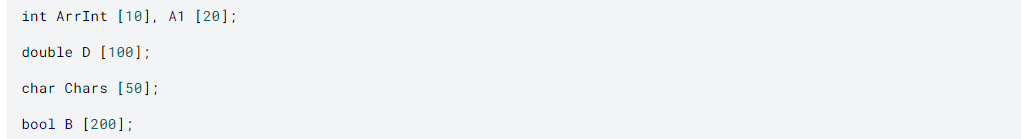
Познакомиться с массивами, действиями над ними и различными типами их сортировки.

Основные теоретические положения.

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

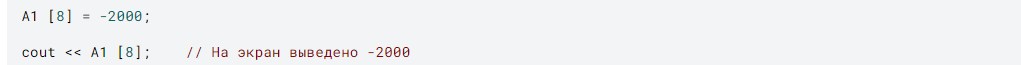
**<Базовый тип элементов> <Идентификатор массива> [<Количество элементов>]**

Например:

****

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

Обращение к определенному элементу массива осуществляется с помощью указания значения индекса этого элемента:



В этом примере, обратившись к элементу массива **A1** с индексом 8, мы, фактически, обратились к его 9-му элементу.

При обращении к конкретному элементу массива этот элемент можно рассматривать как обычную переменную, тип которой соответствует базовому типу элементов массива, и осуществлять со значением этого элемента любые операции, которые характерны для базового типа. Например, поскольку базовым типом массива **A1** является тип данных **int**, с любым элементом этого массива можно выполнять любые операции, которые можно выполнять над значениями типа **int**.

При объявлении массива его можно инициализировать определенными значениями:

 или так:

 Во втором случае мы не указываем количество элементов массива **S**. Автоматически создается массив на 5 элементов в соответствии с инициализирующими значениями.

Эти инициализации будут эквивалентны следующим операциям присваивания:

Количество значений, указанных в фигурных скобках (инициализирующих значений) не должно превышать количества элементов в массиве (в нашем примере - 5).

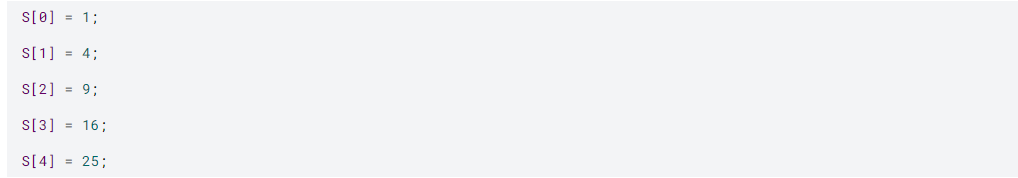
Значения всех элементов массива в памяти располагаются в непрерывной области одно за другим. Общий объем памяти, выделяемый компилятором для массива, определяется как произведение объема одного элемента массива на количество элементов в массиве и равно:

**sizeof( <Базовый тип> )  \* <Количество элементов>**

Для предыдущего примера объем массива **S** будет равен **sizeof( short) \* 5 = 2 \* 5 = 10** байтам.

Поскольку все элементы массивов располагаются в памяти один за другим без разрывов, обращение к элементам массива по их индексам (какой бы длины не был этот массив) осуществляется очень эффективно путем вычисления адреса нужного элемента. Пусть, например, адрес памяти, где начинается массив  **S,**равен **100**, тогда адрес элемента этого массива с индексом **3** будет равен **100 +** **sizeof( short) \* 3 = 100 + 2 \* 3 = 106**. Обращаемся по этому адресу и считываем **2** байта. Это и будет значением элемента с индексом **3** массива **S**.

**В языке C++ не осуществляется проверка выхода за границы массивов**. То есть, вполне корректно (с точки зрения компилятора) будет обращение к элементу массива **S**, индекс которого равен **10**. Это может привести к возникновению весьма серьезных отрицательных последствий. Например, если выполнить присвоение **S[10] = 1000** будут изменены данные, находящиеся за пределами массива, а это может быть значение какой-нибудь другой переменной программы. После этого предсказать поведение программы будет невозможно. Единственный выход – быть предельно внимательным при работе с индексами элементов массивов.

**Ввод-вывод массивов.**

Ранее были рассмотрены приемы ввода-вывода простых предопределенных типов данных (**int, double, char**и **bool**) с помощью потоков ввода и вывода. Стандартные потоки ввода и вывода не “умеют” работать с массивами, поэтому ввод и вывод массивов необходимо реализовывать самостоятельно, обрабатывая массивы поэлементно.

Большинство алгоритмов по обработке массивов реализуются с помощью циклов. Ввод и вывод массивов не являются исключением.

Начнем с рассмотрения операций вывода значений элементов массивов на экран.

**Бинарный поиск**

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.

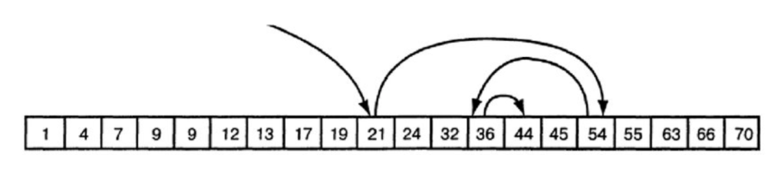
Бинарный поиск работает только в топ случае, если список отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

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.



Постановка задачи.

Разработать алгоритм и написать программу, которая позволяет:

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. (\*)

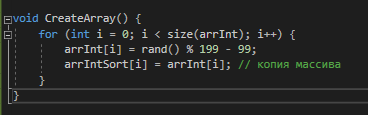
8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

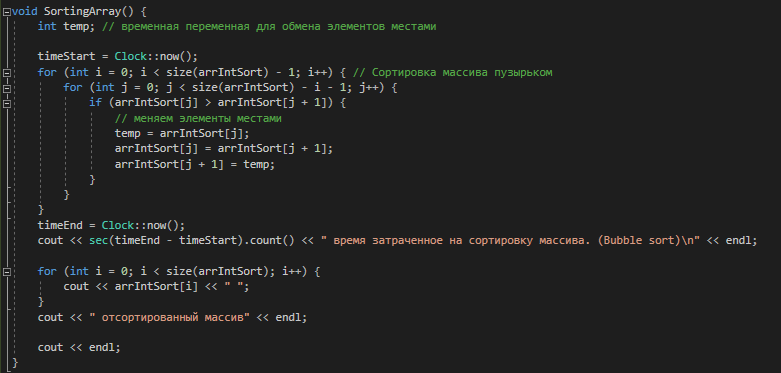
Должна присутствовать возможность запуска каждого пункта многократно.

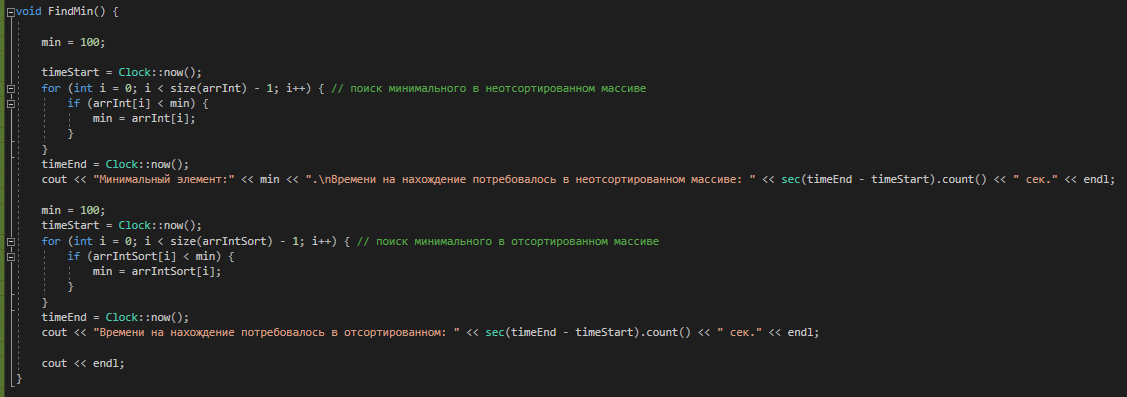
Выполнение работы.

Для удобства и читаемости кода все выполняемые части программы я разделил на отдельные исполняемые методы.

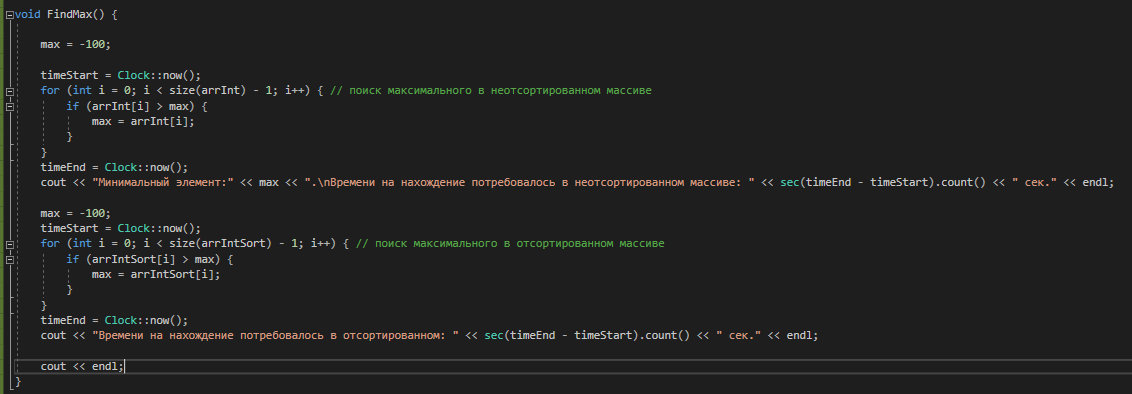
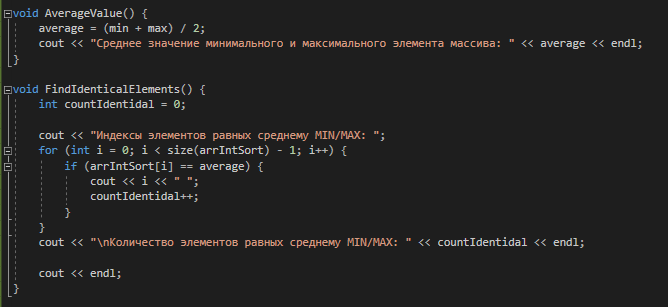
В начале я создаю массив из 100 элементов которые принимают случайное число в диапазоне от -99 до 99 включительно используя функцию rand().



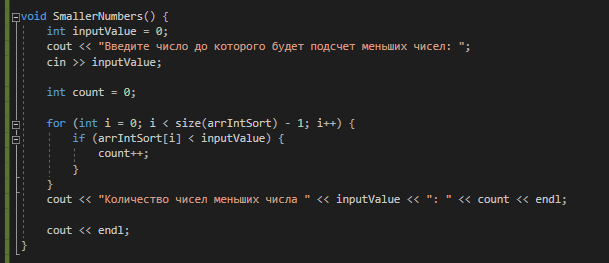
Далее нужно было отсортировать массив, я выбрал метод сортировки пузырьком.  


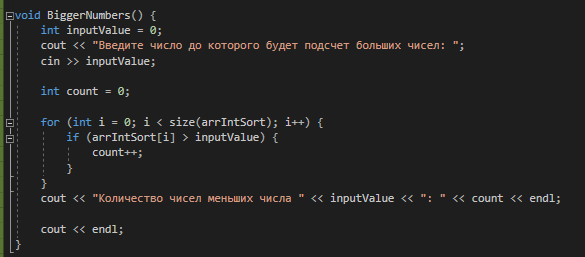
Для подсчета времени использовал две переменные типа Clock используя библиотеку "hrono". Первая принимает время до начала сортировки, вторая после, в выводе выводится их разность. Данная сортировка занимает в среднем 0,00025 сек.

В этой части кода я нахожу минимальный элемент массива с помощью перебора в двух массивах, отсортированном и обычном. Время требуемое на нахождение таких элементов занимает в среднем 1,2е-6 и 4,6е-6 соответственно.

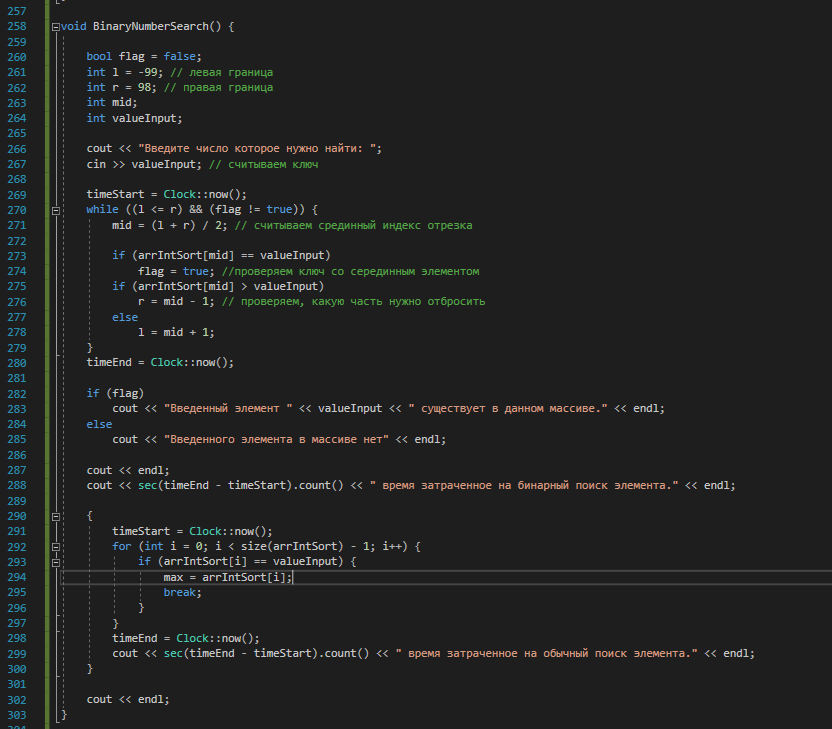
Аналогично методу поиска минимального элемента я нашел их максимальные. Времени занимает примерно как и с минимальными элементами.

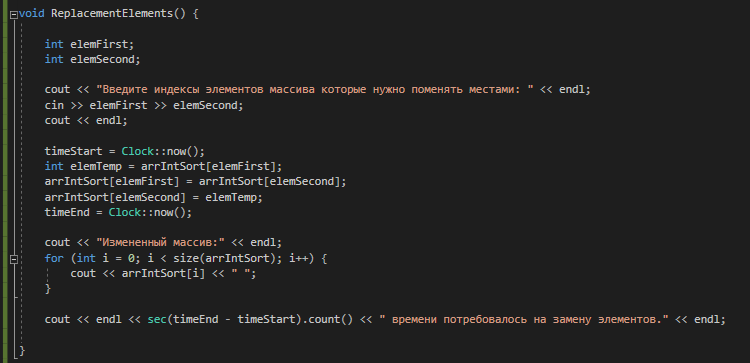
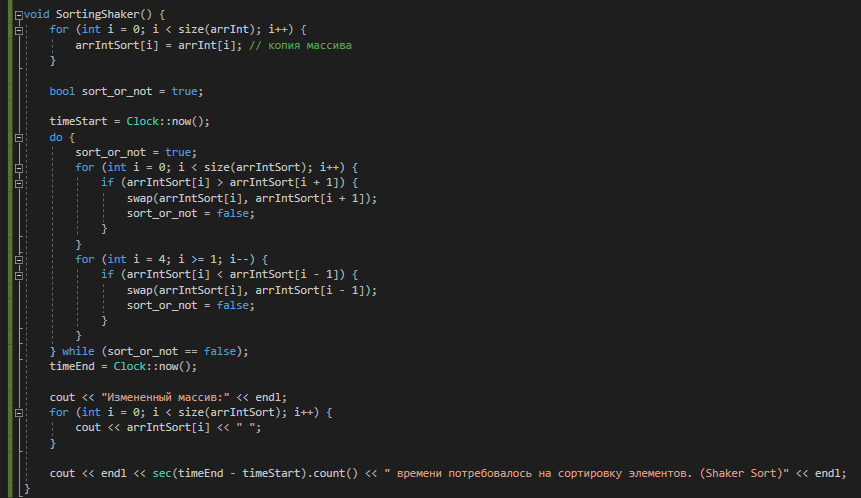
Далее я посчитал среднее значение минимального и максимального элемента и нашел сколько элементов в массиве соответствует данному среднему.

Нахождение количества элементов до значения введенного пользователем выполняется простым счетчиком и перебором всех элементов.



Поиск количества элементов больших введенного пользователем аналогичен предыдущему способу.

 Бинарный поиск. Он достаточно быстрый и не сложный в написании, что несоменный плюс его использования. Бинарный поиск находится в строках 270 — 279. В строке 263 мы создали переменную mid, в которой будет храниться индекс среднего элемента (из отрезка [l, r]). В строке 271 считываем средний элемент отрезка [l, r] в переменную mid, по формуле: (l + r) / 2 (в которой l — левая граница, r — правая граница). В строке 29 проверяем условие arrIntSort[mid] == valueInput:

Замена двух элементов индекс которых вводится пользователем. Сначала запоминаем первый элемент в отдельную переменную, после присваиваем ему значение второго элемента и наконец второму присваиваем первый из переменной.

В конце программы выполняется индивидуальное задание, я выбрал 7-ое.

Реализовал шейкер сортировку. Принцип ее следующий: Перебирает до самого большого или минимального элемента, а далее перемещает его вправо или влево.

Плюсы: простота исполнения.

Минусы: медленное время выполнения, около 0,0007 с.

Выводы.

Массивы очень удобные и потому часто используются в программах. Познакомился с массивами на языке С++ и методами работы с ними.

Полный код программы

#include <iostream>

#include <chrono>

#include <ctime>

using namespace std;

typedef chrono::high\_resolution\_clock Clock;

typedef chrono::duration<double> sec;

int arrInt[100]; // изначальный массив

int arrIntSort[100]; // отсортированный массив

int min;

int max;

int repeat = 0;

float average = 0;

Clock::time\_point timeStart;

Clock::time\_point timeEnd;

void CreateArray(); // создаем массив

void SortingArray(); // сортируем массив

void FindMin(); // находим минимальные элементы массивов

void FindMax(); // находим максимальные элементы массивов

void AverageValue(); // среднее значение мин/мах

void FindIdenticalElements(); // поиск одинаковых элементов равных среднему значению мин/мах

void SmallerNumbers(); // кол-во чисел меньших числа А

void BiggerNumbers(); // кол-во чисел больших числа Б

void BinaryNumberSearch(); // бинарный поиск введенного числа в массиве

void ReplacementElements(); // замена двух элементов местами

void SortingShaker(); // для индивидуального задания

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

do {

CreateArray();

SortingArray();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

do {

FindMin();

FindMax();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

do {

AverageValue();

FindIdenticalElements();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

do {

SmallerNumbers();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

do {

BiggerNumbers();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

do {

BinaryNumberSearch();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

do {

ReplacementElements();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

do {

SortingShaker();

repeat = 0;

cout << "Повторить? (0 - нет / 1 - да): ";

cin >> repeat;

cout << endl;

} while (repeat == 1);

cout << "End.";

}

void CreateArray() {

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

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

arrIntSort[i] = arrInt[i]; // копия массива

}

}

void SortingArray() {

int temp; // временная переменная для обмена элементов местами

timeStart = Clock::now();

for (int i = 0; i < size(arrIntSort) - 1; i++) { // Сортировка массива пузырьком

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

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

// меняем элементы местами

temp = arrIntSort[j];

arrIntSort[j] = arrIntSort[j + 1];

arrIntSort[j + 1] = temp;

}

}

}

timeEnd = Clock::now();

cout << sec(timeEnd - timeStart).count() << " время затраченное на сортировку массива. (Bubble sort)\n" << endl;

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

cout << arrIntSort[i] << " ";

}

cout << " отсортированный массив" << endl;

cout << endl;

}

void FindMin() {

min = 100;

timeStart = Clock::now();

for (int i = 0; i < size(arrInt) - 1; i++) { // поиск минимального в неотсортированном массиве

if (arrInt[i] < min) {

min = arrInt[i];

}

}

timeEnd = Clock::now();

cout << "Минимальный элемент:" << min << ".\nВремени на нахождение потребовалось в неотсортированном массиве: " << sec(timeEnd - timeStart).count() << " сек." << endl;

min = 100;

timeStart = Clock::now();

for (int i = 0; i < size(arrIntSort) - 1; i++) { // поиск минимального в отсортированном массиве

if (arrIntSort[i] < min) {

min = arrIntSort[i];

}

}

timeEnd = Clock::now();

cout << "Времени на нахождение потребовалось в отсортированном: " << sec(timeEnd - timeStart).count() << " сек." << endl;

cout << endl;

}

void FindMax() {

max = -100;

timeStart = Clock::now();

for (int i = 0; i < size(arrInt) - 1; i++) { // поиск максимального в неотсортированном массиве

if (arrInt[i] > max) {

max = arrInt[i];

}

}

timeEnd = Clock::now();

cout << "Минимальный элемент:" << max << ".\nВремени на нахождение потребовалось в неотсортированном массиве: " << sec(timeEnd - timeStart).count() << " сек." << endl;

max = -100;

timeStart = Clock::now();

for (int i = 0; i < size(arrIntSort) - 1; i++) { // поиск максимального в отсортированном массиве

if (arrIntSort[i] > max) {

max = arrIntSort[i];

}

}

timeEnd = Clock::now();

cout << "Времени на нахождение потребовалось в отсортированном: " << sec(timeEnd - timeStart).count() << " сек." << endl;

cout << endl;

}

void AverageValue() {

average = (min + max) / 2;

cout << "Среднее значение минимального и максимального элемента массива: " << average << endl;

}

void FindIdenticalElements() {

int countIdentidal = 0;

cout << "Индексы элементов равных среднему MIN/MAX: ";

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

if (arrIntSort[i] == average) {

cout << i << " ";

countIdentidal++;

}

}

cout << "\nКоличество элементов равных среднему MIN/MAX: " << countIdentidal << endl;

cout << endl;

}

void SmallerNumbers() {

int inputValue = 0;

cout << "Введите число до которого будет подсчет меньших чисел: ";

cin >> inputValue;

int count = 0;

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

if (arrIntSort[i] < inputValue) {

count++;

}

}

cout << "Количество чисел меньших числа " << inputValue << ": " << count << endl;

cout << endl;

}

void BiggerNumbers() {

int inputValue = 0;

cout << "Введите число до которого будет подсчет больших чисел: ";

cin >> inputValue;

int count = 0;

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

if (arrIntSort[i] > inputValue) {

count++;

}

}

cout << "Количество чисел меньших числа " << inputValue << ": " << count << endl;

cout << endl;

}

void BinaryNumberSearch() {

bool flag = false;

int l = -99; // левая граница

int r = 98; // правая граница

int mid;

int valueInput;

cout << "Введите число которое нужно найти: ";

cin >> valueInput; // считываем ключ

timeStart = Clock::now();

while ((l <= r) && (flag != true)) {

mid = (l + r) / 2; // считываем срединный индекс отрезка

if (arrIntSort[mid] == valueInput)

flag = true; //проверяем ключ со серединным элементом

if (arrIntSort[mid] > valueInput)

r = mid - 1; // проверяем, какую часть нужно отбросить

else

l = mid + 1;

}

timeEnd = Clock::now();

if (flag)

cout << "Введенный элемент " << valueInput << " существует в данном массиве." << endl;

else

cout << "Введенного элемента в массиве нет" << endl;

cout << endl;

cout << sec(timeEnd - timeStart).count() << " время затраченное на бинарный поиск элемента." << endl;

{

timeStart = Clock::now();

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

if (arrIntSort[i] == valueInput) {

max = arrIntSort[i];

break;

}

}

timeEnd = Clock::now();

cout << sec(timeEnd - timeStart).count() << " время затраченное на обычный поиск элемента." << endl;

}

cout << endl;

}

void ReplacementElements() {

int elemFirst;

int elemSecond;

cout << "Введите индексы элементов массива которые нужно поменять местами: " << endl;

cin >> elemFirst >> elemSecond;

cout << endl;

timeStart = Clock::now();

int elemTemp = arrIntSort[elemFirst];

arrIntSort[elemFirst] = arrIntSort[elemSecond];

arrIntSort[elemSecond] = elemTemp;

timeEnd = Clock::now();

cout << "Измененный массив:" << endl;

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

cout << arrIntSort[i] << " ";

}

cout << endl << sec(timeEnd - timeStart).count() << " времени потребовалось на замену элементов." << endl;

}

void SortingShaker() {

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

arrIntSort[i] = arrInt[i]; // копия массива

}

bool sort\_or\_not = true;

timeStart = Clock::now();

do {

sort\_or\_not = true;

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

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

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

sort\_or\_not = false;

}

}

for (int i = 4; i >= 1; i--) {

if (arrIntSort[i] < arrIntSort[i - 1]) {

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

sort\_or\_not = false;

}

}

} while (sort\_or\_not == false);

timeEnd = Clock::now();

cout << "Измененный массив:" << endl;

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

cout << arrIntSort[i] << " ";

}

cout << endl << sec(timeEnd - timeStart).count() << " времени потребовалось на сортировку элементов. (Shaker Sort)" << endl;

}

Приложение Б

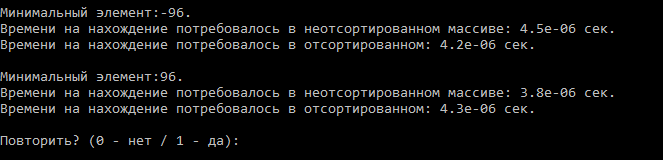
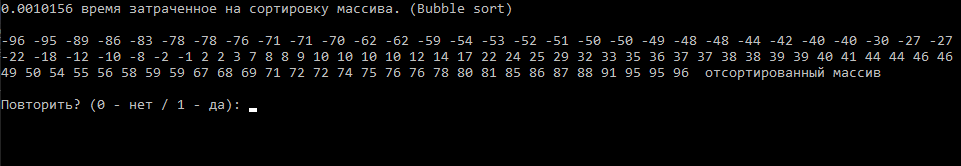
****ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

Рисунок Нахождение минимальны и максимальных элементов

Рисунок Создание массива

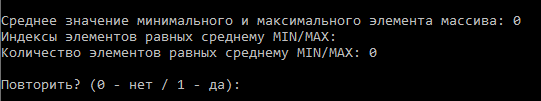
****

Рисунок Поиск элементов по среднему значению МИН/МАХ

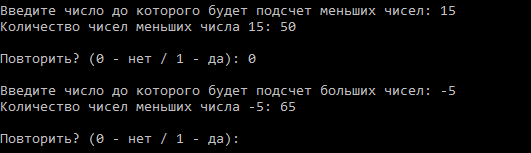
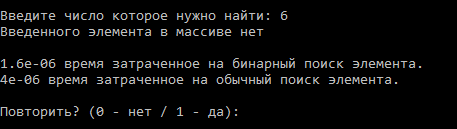


Рисунок Подстчет кол-ва элементов до и после введенного числа



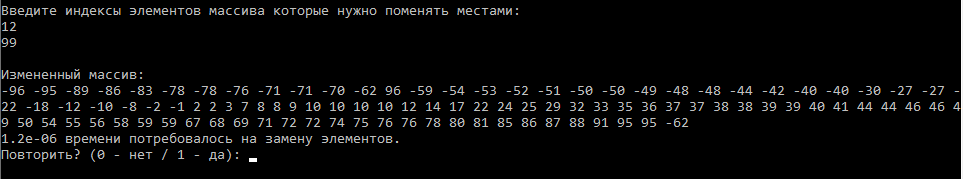
 Рисунок Поиск элемента в массиве

Рисунок 6 Замена элементов местами