**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 И ИХ ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ В ПАМЯТИ"

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. | Тимошенков И.М. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2022

**Цель работы.**

Изучение работы массивов, работа с алгоритмами сортировки, бинарным поиском и библиотекой chrono.

**Основные теоретические положения.**

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.

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.

Comb sort (сортировка расческой) – ещё одна модификация сортировки пузырьком. 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.

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.

Общая суть сортировки вставками такова:

1)    Перебираются элементы в неотсортированной части массива.

2)    Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находится.

Быстрая сортировка (quick sort) – одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки.

Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного.

**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. 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.**

Код программы представлен в приложении А.

#include <iostream>

#include<ctime> // подключение ctime для работы с srand

#include<cmath> // подключение cmath для работы с логарифмами

#include<chrono> // подключение chrono

using namespace std;

using namespace chrono; // подключение пространства имен chrono

void binarySearch(int\* A, int size, int number) //алгоритм бинарного поиска

{

int step\_size = size / 2; // часть размера на которую сдвигается область поиска числа

int bin\_size = size / 2; // середина массива

int flagNumber = 0; // флаг, который отслеживает найденное число

float log = log2(size); // количество шагов, необходимое для поиска числа

for (int i = 0; i < int(ceil(log)); i++)

{

if (step\_size != 1) // так как шаг постепенно стримиться к нулю, то тут реализована проверка

{

step\_size = step\_size / 2; // если шаг больше 1, то он делиться, если нет - то он так и остается равен 1

}

if (number > A[bin\_size]) // если число больше числа, на которое в данный момент указывает bin\_size,

{

bin\_size = bin\_size + step\_size; // то прибавляем некоторую часть размера

}

else if (number < A[bin\_size])

{

bin\_size = bin\_size - step\_size; // иначе - отнимаем

}

if (number == A[bin\_size]) // число найдено и алгоритм прекращает работу

{

flagNumber = A[bin\_size];

break;

}

}

if (flagNumber != 0)

{

cout << "Ваше число есть в массиве \n";

}

else

{

cout << "Вашего числа нет в массиве \n";

}

}

void changeElements(int\* A) // работа по смене чисел через функцию swap

{

int indx1;

int indx2;

cout << "Введите индексы элементов, которые хотели бы поменять местами \n";

cout << "Индекс\_1 - ";

cin >> indx1;

cout << '\n' << "Индекс\_2 - ";

cin >> indx2;

steady\_clock::time\_point s = steady\_clock::now();

cout << '\n' << "Дело сделано \n";

swap(A[indx1], A[indx2]);

steady\_clock::time\_point e = steady\_clock::now();

int t = duration\_cast<nanoseconds>(e - s).count();

cout << t << " наносекунд \n";

}

int stupidSearch(int \*A, int size) // обычный линейный поиск

{

cout << "Загружается алгоритм линейного поиска... \n";

cout << "Введите число для поиска при помощи линейного \n";

int number;

cin >> number;

steady\_clock::time\_point st = steady\_clock::now();

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

{

if (A[i] == number)

{

cout << "Ваше число в массиве \n";

steady\_clock::time\_point en = steady\_clock::now();

cout << duration\_cast<nanoseconds>(en - st).count() << " наносекунд \n";

return 0;

}

}

cout << "Вашего числа нет в массиве \n";

steady\_clock::time\_point end = steady\_clock::now();

cout << duration\_cast<nanoseconds>(end - st).count() << " наносекунд \n";

return 0;

}

void quicksort(int\* A, int end, int begin) // алгоритм быстрой сортировки

{

int mid;

int f = begin;

int l = end;

mid = A[(f + l) / 2];

while (f < l)

{

while (A[f] < mid)

{

f++;

}

while (A[l] > mid)

{

l--;

}

if (f <= l)

{

swap(A[f], A[l]);

f++;

l--;

}

}

if (begin < l) quicksort(A, l, begin);

if (f < end) quicksort(A, end, f);

}

void maxMinSorted(int\* A, int size) // поиск в отсортированном массиве

{

cout << "Отсортированный массив \n";

steady\_clock::time\_point s = steady\_clock::now();

cout << A[0] << ' ' << A[size - 1] << ' '; // программа обращается к последнему и первому элементу

steady\_clock::time\_point e = steady\_clock::now();

cout << duration\_cast<nanoseconds>(e - s).count() << " наносекунд \n";

}

void maxMinNoSorted(int\* A, int size) // поиск в неотсортированном массиве

{

cout << "Неотсортированный массив \n";

int max = 0;

int min = 10000000;

steady\_clock::time\_point s = steady\_clock::now();

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

{

if (A[i] > max)

{

max = A[i];

}

if (A[i] < min)

{

min = A[i];

}

}

cout << min << ' ' << max << ' ';

steady\_clock::time\_point e = steady\_clock::now();

cout << duration\_cast<nanoseconds>(e - s).count() << " наносекунд \n";

}

void averageVal(int\* A, int size) // кол-во элементов и их индексы, которые равны среднему max и min

{

int count = 0;

int average = (A[0] + A[size - 1]) / 2;

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

{

if (A[i] == average)

{

count++;

cout << "Ин." << i << ' ';

}

}

cout << "Кол - во: " << count << '\n';

}

void lessA(int\* A, int size) // кол-во чисел меньших a

{

int a;

cout << "Введите число: ";

cin >> a;

int count = 0;

cout << '\n';

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

{

if (A[i] < a)

{

count++;

}

}

cout << count << '\n';

}

void bigger(int\* A, int size) // кол-во чисел больших b

{

int a;

cout << "Введите число: ";

cin >> a;

int count = 0;

cout << '\n';

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

{

if (A[i] > a)

{

count++;

}

}

cout << count << '\n';

}

int main()

{

setlocale(0, "Rus");

while (1)

{

srand(time(0));

cout << "------Выберите необходимую операцию------ \n";

const int size = 100;

char input;

int A[size] = {};

int flagarray = 0;

int flagsorted = 0;

int Ansort[size] = {};

int flag = 0;

cout << "Нажмите 1, чтобы создать массив длины 100 \n";

cout << "Нажмите 2, чтобы отсортировать массив длины 100 \n";

cout << "Нажмите 3, чтобы найти максимальный и минимальный элемент массива\n";

cout << "Нажмите 4, чтобы найти количество элементов массива, которые равны среднему значению максимального и минимального элементов\n";

cout << "Нажмите 5, чтобы найти кол-во элементов меньше числа а \n";

cout << "Нажмите 6, чтобы найти кол-во элементов больше числа b\n";

cout << "Нажмите 7, чтобы узнать, есть ли ваше число в массиве \n";

cout << "Нажмите 8, чтобы поменять местами элементы массива \n";

cout << "Нажмите 9, чтобы прекратить работу программы \n";

cout << "Нажмите клавишу: ";

cin >> input;

cout << '\n';

if (input == '9')

break;

else

{

cout << "-----Для выхода обратно в меню, введите любую клавишу, кроме клавишь управления функциями-----\n";

while (1)

{

switch (input)

{

case '1':

flagarray = flagarray + 1;

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

{

A[i] = rand() % (100 - (-100)) + (-100);

}

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

{

Ansort[i] = A[i];

}

cout << "Массив создан, хотите его вывести?(y/n)\n";

char chose;

cin >> chose;

if (chose == 'y')

{

cout << "{";

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

{

cout << A[i] << " ";

}

cout << "}\n";

}

cout << "Операция 1 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

cout << '\n';

break;

case '2':

flagsorted = flagsorted + 1;

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

quicksort(A, size - 1, 0);

cout << "Массив отсортирован, хотите его вывести?(y/n)\n";

cin >> chose;

if (chose == 'y')

{

cout << "{";

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

{

cout << A[i] << " ";

}

cout << "}\n";

}

cout << '\n';

cout << "Операция 2 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

break;

case '3':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

maxMinSorted(A, size);

maxMinNoSorted(Ansort, size);

cout << "Операция 3 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

break;

case '4':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

averageVal(A, size);

cout << "Операция 4 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

break;

case '5':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

lessA(A, size);

cout << "Операция 5 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

break;

case '6':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

bigger(A, size);

cout << "Операция 6 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

break;

case '7':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

{

cout << "Загружается алгоритм бинарного поиска... \n";

cout << "Введите число для поиска при помощи бинарного алгоритма \n";

int number;

cin >> number;

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

binarySearch(A, size, number);

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

cout << duration\_cast<nanoseconds>(e - s).count() << " наносекунд \n";

stupidSearch(A, size);

cout << "Операция 7 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

case '8':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

cin >> input;

break;

}

changeElements(A);

cout << "Операция 8 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

cin >> input;

break;

default:

cout << "==========Прекращение работы подпрограммы==========\n\n";

flag = flag + 1;

break;

}

if (flag == 1)

{

flagsorted = flagsorted - 1;

flagarray = flagarray - 1;

flag = flag - 1;

break;

}

}

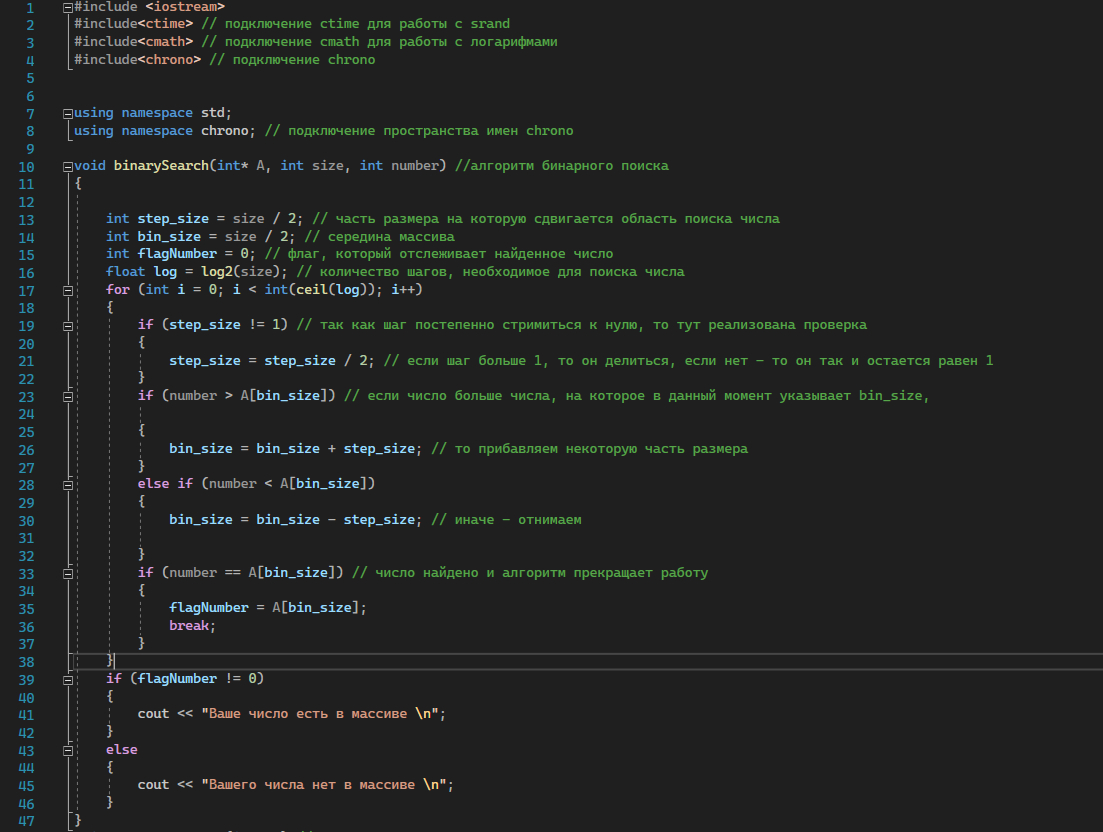
}

}

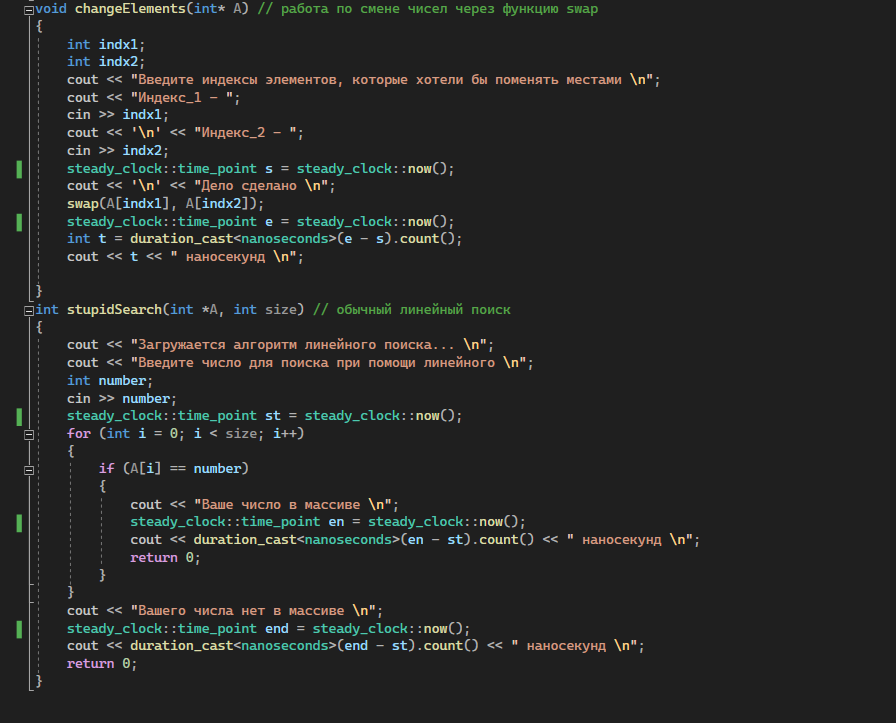
}

**Блок описания кода и использованных алгоритмов**

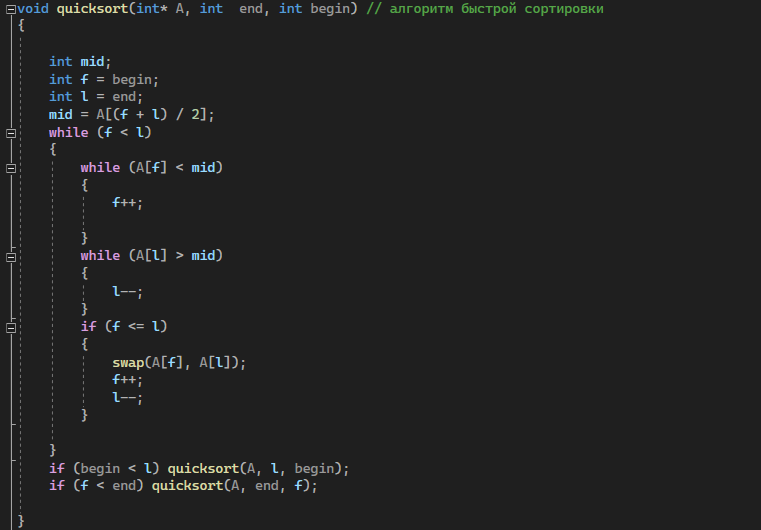
Алгоритм бинарного поиска.



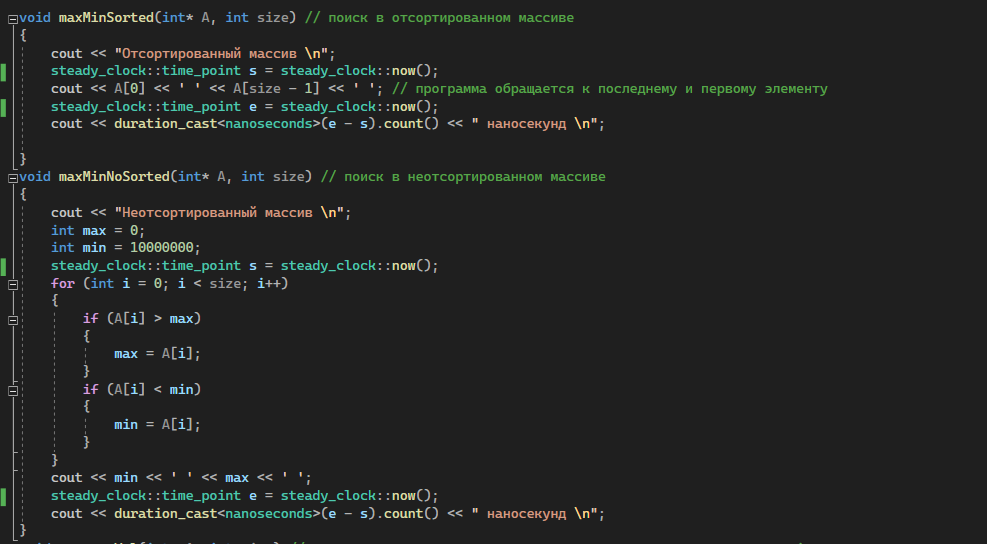
Алгоритм смены чисел местами и алгоритм линейного поиска.



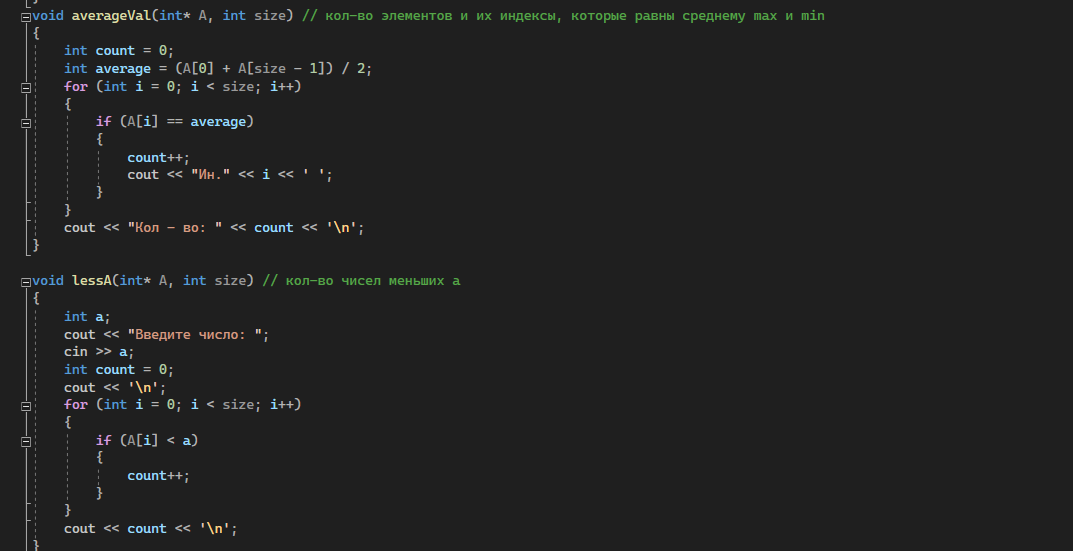
Алгоритм быстрой сортировки (рекурсивное решение)



Алгоритмы поиска минимального и максимального значения



Алгоритм поиска числа **< а или > b** и числа равному среднему максимального и минимального.



Основная программа с формированием меню.