**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.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 2373 |  | Басова К. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Создание и сортировка массива, поиск и изменение значений элементов массива, реализация алгоритма бинарного поиска; изучение и применение библиотеки chrono.

**Основные теоретические положения.**

Массив – это индексированная последовательность однотипных элементов с заранее определенным количеством элементов. Все массивы можно разделить на 2 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 0. При описании массива используются те же модификаторы, что и для простых переменных.

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

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

Например:

int ArrInt [10], A1 [20];

double D [100];

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

A1 [8] = -2000;

При обращении к элементу массива его можно рассматривать как обычную переменную, тип которой соответствует базовому типу элементов массива.

При объявлении массива его можно инициализировать определенными значениями:

short S [5] = {1, 4, 9, 16, 25};

short S [ ] = {1, 4, 9, 16, 25};

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:

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

Многомерные массивы определяются аналогично одномерным массивам.

int  A1 [5] [3];

double D [10] [15] [3];

Общее число элементов в многомерном массиве определяется как произведение количества элементов по каждому измерению. Так, например, массив D содержит 10 \* 15 \* 3 = 450  элементов типа double, а объем памяти, требующийся для этого массива, будет равен 450 \* 8 = 3600 байтам.

Простейший циклический алгоритм вывода значений элементов некоторого одномерного массива выглядит так:

const int n = 10;

short A[n];

…

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

       cout << setw(8) << left << A[i];

cout << endl;

На каждом шаге этого цикла в поток вывода отправляется i-й элемент массива, устанавливается ширина поля вывода (8 позиций), выравнивание по левому краю.

Вывод двумерных массивов обычно осуществляется в табличной форме:

const int n = 10, m = 10;

short A [n] [m];

…

for (int i = 0; i < n; ++i) //  Выводим i-ю строку массива

{

       for (int j = 0; j < m; ++j) //  Выводим j-й элемент i-й строки массива

             cout << setw(7) << right << A [i] [j];

       cout << endl;

}

Здесь используются вложенные циклы.

Ввод массивов также осуществляется с помощью циклических алгоритмов.

const int n = 10;

short A[n];

…

cout  << "Введите " << n << " целых чисел: \n"

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

       cin  >> A [i];

cin.sync();  //  Очистка потока ввода от возможных лишних введенных чисел

При заполнении двумерных массивов ввод значений элементов лучше осуществлять в табличной форме:

const int n = 10, m = 5;

short A [n] [m];

…

cout  << "Введите " << n << " строк из "<< m << " целых чисел: \n"

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

{

       cout << "Строка " <<  i + 1 << ": ";

       //  Вводим поэлементно значения i-й строки массива

       for (int j = 0; j < m; ++j)

             сin  >>  A [i] [j];

       cin.sync();

}

Обмен элементов массива осуществляется через буферную переменную либо через функцию swap (a, b).

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Простейшая из сортировок – сортировка обменом (bubble sort). Суть метода заключается в попарном сравнении элементов и последующем обмене. Если следующий элемент меньше текущего, то они меняются местами. 1 полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент. Процесс повторяется до тех пор, пока вся последовательность не будет упорядочена.

Shaker sort – модификация пузырьковой сортировки. Принцип работы: попарное сравнение элементов и последующий обмен местами. Как только максимальный элемент становится на свое место, алгоритм запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно. Процесс повторяется до тех пор, пока массив не будет отсортирован. За счет того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее.

Comb sort (сортировка расческой) – ещё одна модификация сортировки пузырьком. Алгоритм был разработан для случаев, когда минимальные элементы стоят слишком далеко, или максимальные – слишком близко к началу массива. Переставляются элементы, стоящие на расстоянии. Оптимально изначально взять расстояние равным длине массива, а далее уменьшать его на 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.

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.

Сортировка вставками делит массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться.

Быстрая сортировка (quick sort) – существенно улучшенная версия алгоритма пузырьковой сортировки. Общая идея алгоритма: сначала выбирается из массива элемент, который называется опорным. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на 3 непрерывных отрезка, следующие друг за другом: меньше опорного, равны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы. На практике массив обычно делят на 2 части: «меньше опорного» и «равные и большие» или «меньше опорного или равные» и «большие». Такой поход в общем случае эффективнее, ведь упрощается алгоритм разделения.

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. Сам алгоритм имеет вид:

1) Определение значения в середине массива. 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 массива должны принимать случайное значение в диапазоне от -99 до 99.

2) Сортирует заданный в пункте 1 массив от меньшего к большему и определяет время, затраченное на сортировку, используя библиотеку chrono.

3) Находит максимальный и минимальный элемент массива и подсчитывает время поиска этих элементов в отсортированном массиве и неотсортированном.

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. 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, с помощью алгоритма бинарного поиска. Сравнивает скорость его работы с обычным перебором.

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

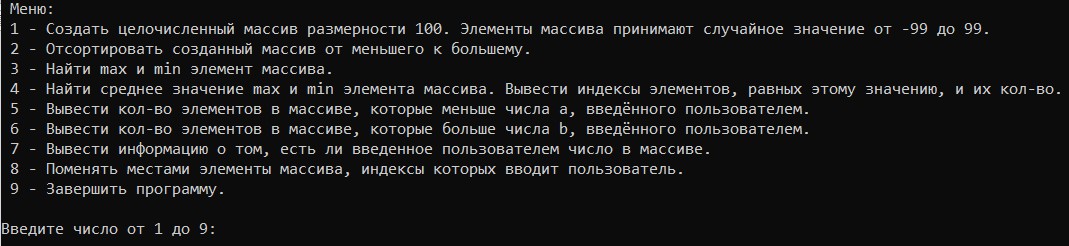
Должна присутствовать возможность запуска каждого пункта многократно.

**Выполнение работы.**

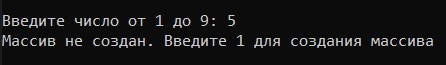
Код программы представлен в приложении А.

Описание кода и использованных алгоритмов.

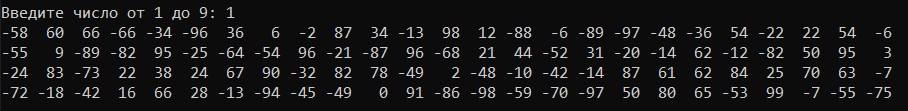
1. При запуске программы появляется окно, в котором выводится меню допустимых операций под номерами. Пользователю предлагается ввести любой номер операции от 1 до 9:



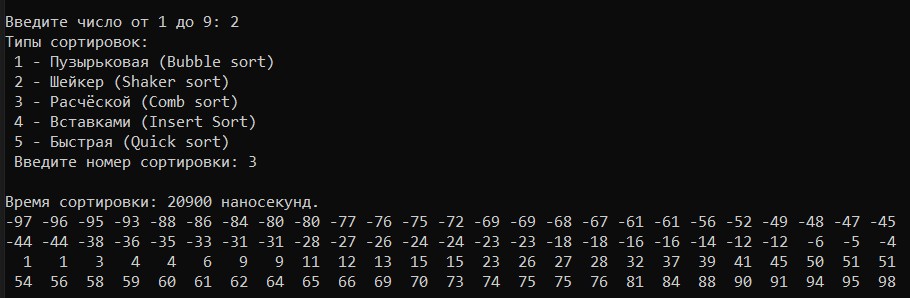
2. При попытке пользователя работать с массивом до его создания (т.е. при вводе чисел 2 – 8 до 1) программа выдаёт следующее сообщение:



3. При вводе 1 создаётся и выводится на экран массив размера 100 из случайных чисел от -99 до 99:



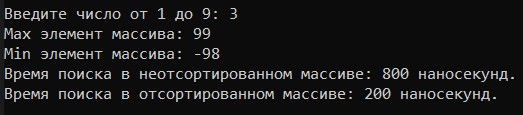
4. При вводе 2 пользователю предлагается ввести число от 1 до 5 – номер типа сортировки. После этого массив сортируется выбранным способом от меньшего к большему и выводится на экран. Также подсчитывается время, затраченное на сортировку (с помощью библиотеки chrono).



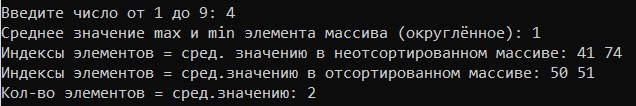
5. При попытке пользователя работать с массивом до его сортировки (т.е. при вводе чисел 3 – 8 до 2) программа выдаёт следующее сообщение:



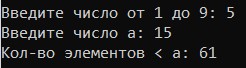
6. При вводе 3 выводится максимальный и минимальный элемент массива, а также рассчитывается время поиска этих элементов в отсортированном и неотсортированном массиве.



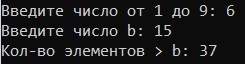
7. При вводе 4 подсчитывается и выводится на экран среднее значение максимального и минимального элемента массива, индексы элементов в отсортированном и неотсортированном массиве, которые равны этому значению, и количество этих элементов.



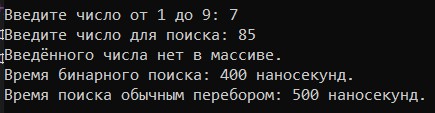
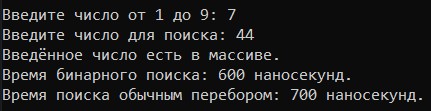
8. При вводе 5 пользователю предлагается ввести ещё одно число (а). Затем на экран выводится количество элементов массива, значения которых меньше введенного числа.



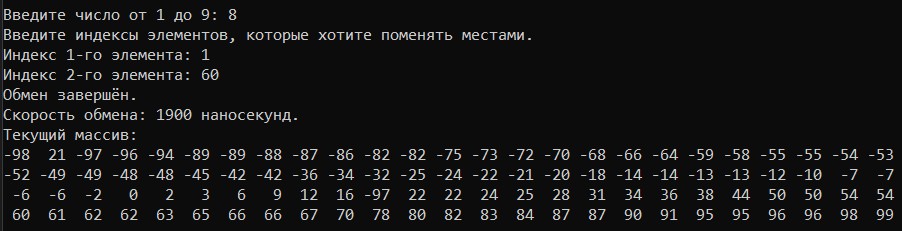
9. При вводе 6 предлагается ввести ещё одно число (b), после чего на экран выводится количество элементов массива, значения которых больше этого числа.



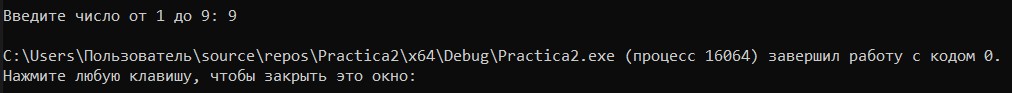
10. При вводе 7 пользователю предлагается ввести ещё одно число – для поиска в массиве. Программа выдаёт информацию о том, есть ли число в массиве, а также время бинарного поиска и поиска обычным перебором.

11. При вводе 8 пользователю предлагается ввести ещё 2 числа – индексы двух любых элементов массива, которые он хочет поменять местами. После того, как обмен завершён, выводится получившийся массив.



12. При вводе 9 работа программы завершается.



**Выводы.**

Для работы с массивом его следует сначала отсортировать, так как искать ту или иную информацию о его элементах проще, когда они упорядочены. Например, это даёт возможность использовать алгоритм бинарного поиска. Кроме того, ускоряется процесс выполнения некоторых операций (для поиска минимального и максимального значения в отсортированном массиве достаточно узнать значения его первого и последнего элемента).

Чтобы определить время, за которое выполняется операция или цикл, используется библиотека chrono. Благодаря ей можно сравнить скорости работы определённых алгоритмов и сделать выводы об их эффективности.

Приложение А

код Программы

#include <iostream>

#include <chrono>

using namespace std;

using namespace chrono;

const int N = 100; // Кол-во элементов в массиве

void BubbleSort(int\* array) { // Пузырьковая сортировка

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

int sorted = 1;

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

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

sorted = 0;

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

}

}

if (sorted) // Если sorted = 1, значит массив отсортирован

break;

}

}

void ShakerSort(int\* array) { // Шейкер-сортировка

int left = 0; // Левая граница

int right = N - 1; // Правая граница

int sorted = 1;

while ((left < right) and (sorted == 1)) { // Цикл выполняется, пока границы не сомкнулись или массив не отсортирован

sorted = 0;

for (int i = left; i < right; i++) { // Сортировка слева

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

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

sorted = 1;

}

}

right --; // Сдвиг правой границы

for (int i = right; i > left; i--) { // Сортировка справа

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

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

sorted = 1;

}

}

left++; // Сдвиг левой границы

}

}

void CombSort(int\* array) { // Сортировка расчёской

const float k = 1.247; // Константа для определения расстояния между сравниваемыми элементами

int gap = N;

while (gap > 1) { // Цикл выполняется, пока расстояние между элементами больше 1

gap /= k;

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

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

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

}

}

}

void InsertSort(int\* array) { // Сортировка вставками

for (int i = 1; i < N; i++) { // Сравнение элемента с предыдущими

for (int j = i; j > 0; j--) {

if (array[j - 1] > array[j])

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

}

}

}

void QuickSort(int\* array, int begin, int end) { // Быстрая сортировка

int left = begin;

int right = end;

int mid = array[(left + right) / 2]; // Опорный элемент

while (left < right) {

while (array[left] < mid) left++; // Пока левый элемент меньше опорного, сдвигается вправо

while (array[right] > mid) right--; // Пока правый элемент больше опорного, сдвигается влево

if (left <= right) {

swap(array[left], array[right]);

left++;

right--;

}

}

if (begin < right)

QuickSort(array, begin, right); // Сортировка правой части

if (left < end)

QuickSort(array, left, end); // Сортировка левой части

}

int main() {

setlocale(0, "Rus");

int arr[N], arr\_sort[N], i, chislo;

int op1 = 0, op2 = 0;

int MaxNum = -100;

int MinNum = 100;

cout << " Меню: \n\

1 - Создать целочисленный массив размерности 100. Элементы массива принимают случайное значение от -99 до 99. \n\

2 - Отсортировать созданный массив от меньшего к большему. \n\

3 - Найти max и min элемент массива. \n\

4 - Найти среднее значение max и min элемента массива. Вывести индексы элементов, равных этому значению, и их кол-во. \n\

5 - Вывести кол-во элементов в массиве, которые меньше числа a, введённого пользователем. \n\

6 - Вывести кол-во элементов в массиве, которые больше числа b, введённого пользователем. \n\

7 - Вывести информацию о том, есть ли введенное пользователем число в массиве. \n\

8 - Поменять местами элементы массива, индексы которых вводит пользователь. \n\

9 - Завершить программу. \n";

cout << "\nВведите число от 1 до 9: ";

cin >> chislo; // Вводим номер операции

while (chislo != 9) { // Пока номер не = 9, программа выполняется

if ((chislo < 1) or (chislo > 9)) { // Проверяем, есть ли введённое число среди номеров допустимых операций

cout << "Такой операции не существует.";

}

else if (chislo == 1) {

op1++; // Признак выполнения 1-ой операции. Подсчитывает, сколько раз она была вызвана

for (i = 0; i < N; i++) {

arr[i] = (rand() % 199) - 99; // Присваиваем каждому элементу массива случайное число от -99 до 99

cout.width(3);

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

if (((i + 1) % 25 == 0) and (i != 99)) // Оформление вывода

cout << "\n";

}

}

else if (op1 == 0) // Проверяем, был ли создан массив

cout << "Массив не создан. Введите 1 для создания массива";

else if (chislo == 2) {

op2 = op1; // Признак выполнения 2-ой операции после 1-ой

int num;

for (i = 0; i < N; i++)

arr\_sort[i] = arr[i];

cout << "Типы сортировок:\n\

1 - Пузырьковая (Bubble sort)\n\

2 - Шейкер (Shaker sort)\n\

3 - Расчёской (Comb sort)\n\

4 - Вставками (Insert Sort)\n\

5 - Быстрая (Quick sort)\n\

Введите номер сортировки: ";

cin >> num;

if (num == 1) { // Пузырьковая сортировка с подсчётом времени

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

start = steady\_clock::now();

BubbleSort(arr\_sort);

end = steady\_clock::now();

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

cout << "\nВремя сортировки: " << result.count() << " наносекунд.\n";

}

if (num == 2) { // Шейкер-сортировка с подсчётом времени

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

start = steady\_clock::now();

ShakerSort(arr\_sort);

end = steady\_clock::now();

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

cout << "\nВремя сортировки: " << result.count() << " наносекунд.\n";

}

if (num == 3) { // Сортировка расчёской с подсчётом времени

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

start = steady\_clock::now();

CombSort(arr\_sort);

end = steady\_clock::now();

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

cout << "\nВремя сортировки: " << result.count() << " наносекунд.\n";

}

if (num == 4) { // Сортировка вставками с подсчётом времени

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

start = steady\_clock::now();

InsertSort(arr\_sort);

end = steady\_clock::now();

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

cout << "\nВремя сортировки: " << result.count() << " наносекунд.\n";

}

if (num == 5) { // Быстрая сортировка с подсчётом времени

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

start = steady\_clock::now();

QuickSort(arr\_sort, 0, N - 1);

end = steady\_clock::now();

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

cout << "\nВремя сортировки: " << result.count() << " наносекунд.\n";

}

for (i = 0; i < N; i++) { // Вывод отсортированного массива

cout.width(3);

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

if (((i + 1) % 25 == 0) and (i != 99))

cout << "\n";

}

}

else if (((op2 == 0) or (op2 != op1)) and (chislo != 1)) // Проверяем, был ли отсортирован созданный массив

cout << "Массив не отсортирован. Введите 2 для сортировки массива.";

else if (chislo == 3) {

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

start = steady\_clock::now();

for (i = 0; i < N; i++) { // Находим max и min элементы в неотсортированном массиве

if (arr[i] > MaxNum) {

MaxNum = arr[i];

}

if (arr[i] < MinNum) {

MinNum = arr[i];

}

}

end = steady\_clock::now();

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

cout << "Max элемент массива: " << MaxNum;

cout << "\nMin элемент массива: " << MinNum;

cout << "\nВремя поиска в неотсортированном массиве: " << result.count() << " наносекунд.";

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start\_sort, end\_sort;

nanoseconds result\_sort;

start\_sort = steady\_clock::now();

MinNum = arr\_sort[0]; // Находим max и min элементы в отсортированном массиве

MaxNum = arr\_sort[99];

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

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

cout << "\nВремя поиска в отсортированном массиве: " << result\_sort.count() << " наносекунд.";

}

else if (chislo == 4) {

MinNum = arr\_sort[0];

MaxNum = arr\_sort[99];

int kol = 0;

float Average = float(MaxNum + MinNum) / 2.0; // Находим среднее значение max и min элемента массива

int RoundAverage = int(ceil(Average)); // Округляем среднее значение

cout << "Среднее значение max и min элемента массива (округлённое): " << RoundAverage;

cout << "\nИндексы элементов = сред. значению в неотсортированном массиве: ";

for (i = 0; i < N; i++) {

if (arr[i] == RoundAverage) { // Находим индексы элементов, = среднему значению, в неотсортированном массиве

cout << i << " ";

kol++;

}

}

cout << "\nИндексы элементов = сред.значению в отсортированном массиве: ";

for (i = 0; i < N; i++) {

if (arr\_sort[i] == RoundAverage) { // Находим индексы элементов, = среднему значению, в отсортированном массиве

cout << i << " ";

}

}

cout << "\nКол-во элементов = сред.значению: " << kol;

}

else if (chislo == 5) {

cout << "Введите число а: ";

int a, k = 0;

cin >> a; // Вводим число а

for (i = 0; i < N; i++) {

if (arr\_sort[i] < a) // Находим кол-во элементов, которые < введённого числа а

k++;

}

cout << "Кол-во элементов < a: " << k;

}

else if (chislo == 6) {

cout << "Введите число b: ";

int b, k = 0;

cin >> b; // Вводим число b

for (i = 0; i < N; i++) {

if (arr\_sort[i] > b) // Находим кол-во элементов, которые > введённого числа b

k++;

}

cout << "Кол-во элементов > b: " << k;

}

else if (chislo == 7) {

int SearchNum;

cout << "Введите число для поиска: ";

cin >> SearchNum; // Вводим число для поиска в массиве

int found = 0;

int now = N / 2, move = N / 4, att = int(ceil(log2(N))); // Индекс центр. элемента, начальный шаг и max кол-во попыток

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start\_bin, end\_bin;

nanoseconds result\_bin;

start\_bin = steady\_clock::now();

for (int k = 0; k < att; k++) {

if (arr\_sort[now] == SearchNum) { // Проверяем, равен ли текущий элемент введённому числу

found = 1;

}

if (arr\_sort[now] > SearchNum) // Если элемент больше, то сдвигаемся влево на 1/2 оставшейся слева части массива

now -= move;

if (arr\_sort[now] < SearchNum) // Если элемент меньше, то сдвигаемся вправо на 1/2 оставшейся справа части массива

now += move;

move = move / 2 + 1; // Уменьшаем шаг

}

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

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

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start\_per, end\_per;

nanoseconds result\_per;

start\_per = steady\_clock::now();

for (int l = 0; l < N; l++) {

if (arr\_sort[l] == SearchNum) { // Делаем перебор всех элементов массива и сравниваем с введённым числом

found = 1;

}

}

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

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

if (found == 0)

cout << "Введённого числа нет в массиве.\n";

else

cout << "Введённое число есть в массиве.\n";

cout << "Время бинарного поиска: " << result\_bin.count() << " наносекунд.\n";

cout << "Время поиска обычным перебором: " << result\_per.count() << " наносекунд.";

}

else if (chislo == 8) {

int index1, index2;

cout << "Введите индексы элементов, которые хотите поменять местами.\n";

cout << "Индекс 1-го элемента: "; // Вводим индексы элементов, которые хотим поменять местами

cin >> index1;

while ((index1 < 0) or (index1 > 99)) {

cout << "Элемента с таким индексом нет в массиве\n" << "Индекс 1-го элемента: ";

cin >> index1;

}

cout << "Индекс 2-го элемента: ";

cin >> index2;

while ((index2 < 0) or (index2 > 99)) {

cout << "Элемента с таким индексом нет в массиве\n" << "Индекс 2-го элемента: ";

cin >> index2;

}

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

start = steady\_clock::now();

swap(arr\_sort[index1], arr\_sort[index2]); // Меняем элементы местами

end = steady\_clock::now();

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

cout << "Обмен завершён.\n" << "Скорость обмена: " << result.count() << " наносекунд.\n" << "Текущий массив: \n";

for (i = 0; i < N; i++) { // Вывод текущего массива после обмена

cout.width(3);

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

if (((i + 1) % 25 == 0) and (i != 99))

cout << "\n";

}

}

cout << "\n\nВведите число от 1 до 9: ";

cin >> chislo; // Снова вводим число после выполнения операции

}

return 0;

}