**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(ка) гр. |  | Саевич И. И. 4372 |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение одномерных статических массивов; получение практических навыков работы с ними (заполнение массива, различные алогритмы сортировки массивов, поиск максимального и минимального элемента, алгоритм бинарного поиска);

**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: ai при i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных.

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];

char Chars [50];

bool B [200];

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:

A1 [8] = -2000;

cout << A1 [8]; // На экран выведено -2000

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:

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

или так:

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

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:

S[0] = 1;

S[1] = 4;

S[2] = 9;

S[3] = 16;

S[4] = 25;

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

**Постановка задачи.**

Необходимо:

1) Написать программу, которая cоздает 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) Вывести среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Вывести индексы всех элементов, которые равны этому значению, и их количество. Подсчитать время поиска.

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

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

7) Вывести информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализовать алгоритм бинарного поиска. Сравнить скорость его работы с обычным перебором.

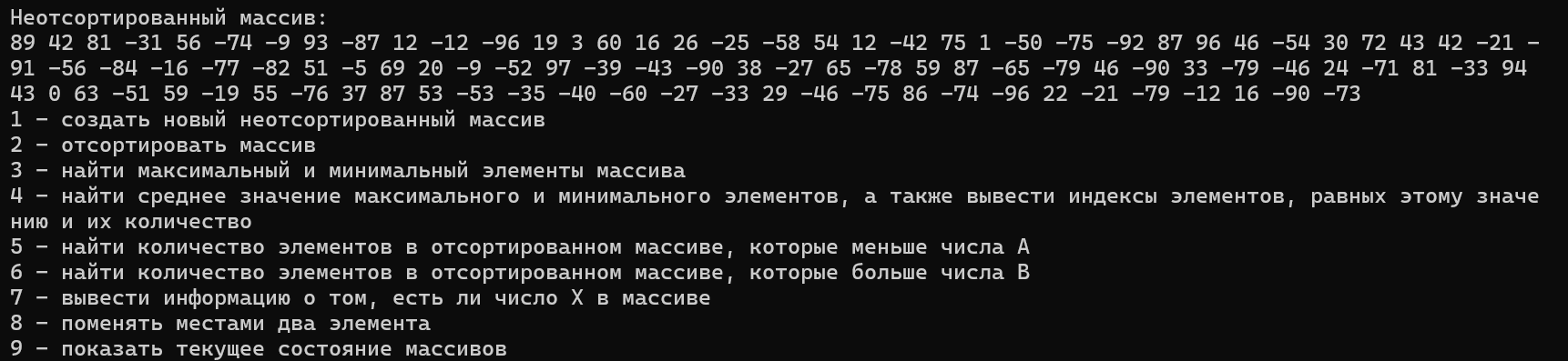
8) Поменять местами элементы массива, индексы которых вводит пользователь. Вывести скорость обмена, используя библиотеку chrono.

Должна присутствовать возможность запуска каждого пункта многократно.

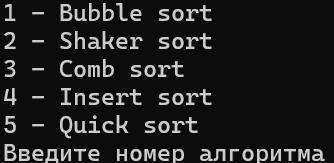
**Выполнение работы.**

Код программы представлен в приложении A.

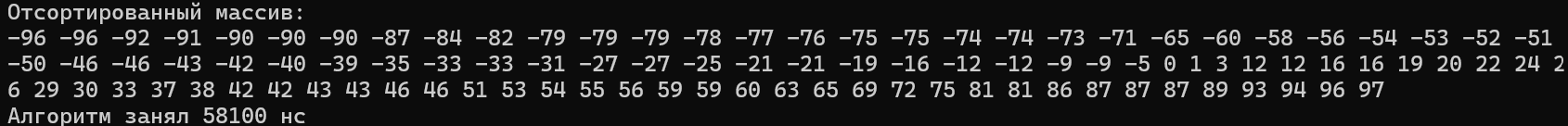
Создается массив случайных чисел от -100 до 100.



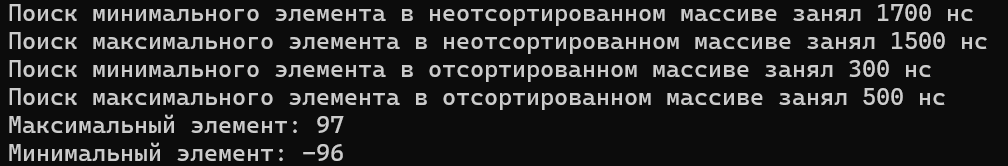
Пользователь может выбрать алгоритм сортировки этого массива.



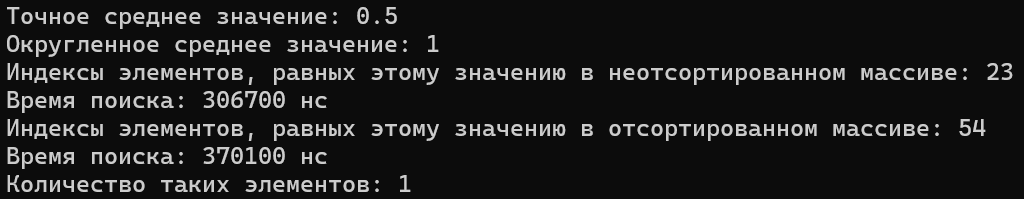
Выводится отсортированный массив и время, которое занял алгоритм.



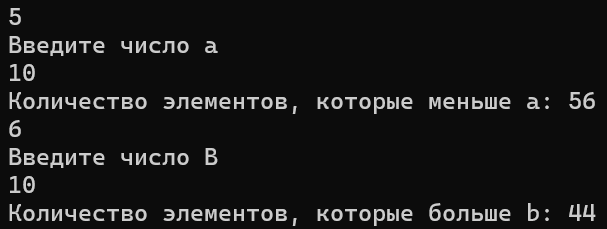
Производится поиска максимального и минимального элемента в отсортированном массиве и в неотсортированном.



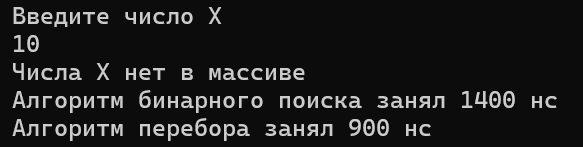
Находится среднее значение максимального и минимального элементов. Проверяется наличие элемента с таким значением в массиве.



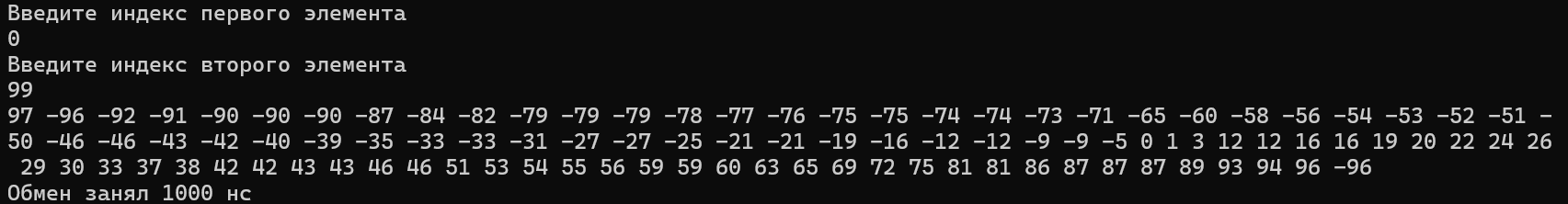
Пользователь вводит число. Находится количество элементов в массиве, которые строго больше этого и числа и которые строго меньше этого числа.



Проверяется наличие введенного пользователем числа в массиве.



Меняет местами два элемента, индексы которых вводит пользователь.



**Выводы.**

В ходе работы мы изучили одномерные статические массивы, а также получили практические навыки работы с ними (заполнение массива, различные алогритмы сортировки массивов, поиск максимального и минимального элемента, алгоритм бинарного поиска);

Приложение А

рабочий код

﻿#include <iostream>

#include <chrono>

using namespace std;

void InputCheck(int &number)

{

while (cin.fail())

{

cin.clear();

cin.sync();

cin.ignore(1000, '\n');

cout << "Введите ЧИСЛО" << endl;

cin >> number;

}

}

void BubbleSort(int Array[], int N)

{

bool swapped = true;

int j = 0;

while (swapped)

{

swapped = false;

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

{

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

{

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

swapped = true;

}

}

j++;

}

}

void ShakerSort(int Array[], int N)

{

int start = 0, end = N - 1;

bool swapped = true;

while (swapped)

{

swapped = false;

for (int i = start; i != end; ++i)

{

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

{

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

swapped = true;

}

}

if (!swapped)

break;

swapped = false;

--end;

for (int i = end; i != start; --i)

{

if (Array[i] < Array[i - 1])

{

std::swap(Array[i], Array[i - 1]);

swapped = true;

}

}

++start;

}

}

void CombSort(int Array[], int N)

{

bool swapped = true;

int j = 0;

float S = N - 1, k = 1.247;

while (S >= 1)

{

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

{

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

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

}

S /= k;

}

while (swapped)

{

swapped = false;

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

{

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

{

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

swapped = true;

}

}

j++;

}

}

void InsertSort(int Array[], int N)

{

int key, j;

for (int i = 1; i != N; ++i)

{

key = Array[i];

j = i - 1;

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

{

Array[j + 1] = Array[j];

j -= 1;

}

Array[j + 1] = key;

}

}

void QuickSort(int Array[], int start, int end)

{

int mid, l = start, r = end;

mid = Array[(l + r) / 2];

while (l < r)

{

while (Array[l] < mid)

l++;

while (Array[r] > mid)

r--;

if (l <= r)

{

swap(Array[l], Array[r]);

l++;

r--;

}

}

if (start < r)

QuickSort(Array, start, r);

if (end > l)

QuickSort(Array, l, end);

}

int BinarySearch(int Array[], int value, int start, int end)

{

if (end >= start)

{

int mid = start + (end - start) / 2;

if (Array[mid] == value)

return mid;

if (Array[mid] > value)

return BinarySearch(Array, value, start, mid - 1);

return BinarySearch(Array, value, mid + 1, end);

}

return -1;

}

int BinarySearchFirstOccurrence(int Array[], int value, int start, int end)

{

if (end >= start)

{

int mid = start + (end - start) / 2;

if (Array[mid] == value)

{

while (Array[mid - 1] == value && mid - 1 >= 0)

mid -= 1;

return mid;

}

if (Array[mid] > value)

return BinarySearchFirstOccurrence(Array, value, start, mid - 1);

return BinarySearchFirstOccurrence(Array, value, mid + 1, end);

}

return start;

}

int BinarySearchLastOccurrence(int Array[], int value, int start, int end)

{

int mid = start + (end - start) / 2;

if (end >= start)

{

if (Array[mid] == value)

{

while (Array[mid + 1] == value && mid + 1 <= end)

mid += 1;

return mid;

}

if (Array[mid] > value)

return BinarySearchLastOccurrence(Array, value, start, mid - 1);

return BinarySearchLastOccurrence(Array, value, mid + 1, end);

}

return start - 1;

}

int ElemSearch(int Array[], int value, int N)

{

int X = -1;

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

{

if (Array[i] == value)

{

X = value;

break;

}

}

return X;

}

int MinSearch(int Array[], int N)

{

int min = 101;

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

min = Array[i] < min ? Array[i] : min;

return min;

}

int MaxSearch(int Array[], int N)

{

int max = 0;

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

max = Array[i] > max ? Array[i] : max;

return max;

}

int main()

{

srand(time(0));

setlocale(0, "");

const int N = 100;

int Array[N], sortedArray[N], select = 0;

bool isSorted = false;

chrono::nanoseconds duration;

chrono::steady\_clock::time\_point start, end;

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

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

{

Array[i] = -99.5 + rand() % 200;

cout << Array[i] << " ";

}

cout << endl;

cout << "1 - создать новый неотсортированный массив\n" <<

"2 - отсортировать массив\n" <<

"3 - найти максимальный и минимальный элементы массива\n" <<

"4 - найти среднее значение максимального и минимального элементов, а также вывести индексы элементов, равных этому и значению и их количество\n" <<

"5 - найти количество элементов в отсортированном массиве, которые меньше числа A\n" <<

"6 - найти количество элементов в отсортированном массиве, которые больше числа B\n" <<

"7 - вывести информацию о том, есть ли число X в массиве\n" <<

"8 - поменять местами два элемента\n" <<

"9 - показать текущее состояние массивов" << endl;

while (true)

{

cin >> select;

InputCheck(select);

switch (select)

{

case 1:

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

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

{

Array[i] = -99.5 + rand() % 200;

cout << Array[i] << " ";

}

cout << endl;

isSorted = false;

break;

case 2:

{

isSorted = false;

cout << "1 - Bubble sort\n2 - Shaker sort\n3 - Comb sort\n4 - Insert sort\n5 - Quick sort" << endl;

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

sortedArray[i] = Array[i];

cout << "Введите номер алгоритма" << endl;

int numberOfAlgorithm;

cin >> numberOfAlgorithm;

InputCheck(numberOfAlgorithm);

while (!isSorted)

{

switch (numberOfAlgorithm)

{

case 1:

isSorted = true;

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

BubbleSort(sortedArray, N);

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

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

break;

case 2:

isSorted = true;

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

ShakerSort(sortedArray, N);

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

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

break;

case 3:

isSorted = true;

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

CombSort(sortedArray, N);

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

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

break;

case 4:

isSorted = true;

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

InsertSort(sortedArray, N);

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

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

break;

case 5:

isSorted = true;

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

QuickSort(sortedArray, 0, N - 1);

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

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

break;

default:

cout << "Алгоритма с такими номером нет. Введите другое число." << endl;

}

}

if (isSorted)

{

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

for (int i = 0; i != N; ++i)

cout << sortedArray[i] << " ";

cout << endl << "Алгоритм занял " << duration.count() << " нс" << endl;

}

break;

}

case 3:

{

if (isSorted)

{

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

int min = MinSearch(Array, N);

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

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

cout << "Поиск минимального элемента в неотсортированном массиве занял " << duration.count() << " нс" << endl;

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

int max = MaxSearch(Array, N);

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

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

cout << "Поиск максимального элемента в неотсортированном массиве занял " << duration.count() << " нс" << endl;

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

min = sortedArray[0];

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

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

cout << "Поиск минимального элемента в отсортированном массиве занял " << duration.count() << " нс" << endl;

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

max = sortedArray[N - 1];

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

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

cout << "Поиск максимального элемента в отсортированном массиве занял " << duration.count() << " нс" << endl;

cout << "Максимальный элемент: " << max << "\nМинимальный элемент: " << min << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 4:

{

if (isSorted)

{

int number = 0, mid;

double dMid = static\_cast<double>((sortedArray[N - 1] + sortedArray[0])) / 2;

cout << "Точное среднее значение: " << dMid << endl;

if (dMid > 0)

mid = dMid + 0.5;

else

mid = dMid - 0.5;

cout << "Округленное среднее значение: " << mid << endl;

cout << "Индексы элементов, равных этому значению в неотсортированном массиве: ";

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

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

{

if (Array[i] == mid)

{

cout << i << " ";

number++;

}

}

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

if (!number)

cout << "-" << endl;

else cout << endl;

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

cout << "Время поиска: " << duration.count() << " нс" << endl;

number = 0;

cout << "Индексы элементов, равных этому значению в отсортированном массиве: ";

int j = 0;

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

while (sortedArray[j] <= mid)

{

if (sortedArray[j] == mid)

{

cout << j << " ";

number++;

}

j++;

}

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

if (!number)

cout << "-" << endl;

else cout << endl;

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

cout << "Время поиска: " << duration.count() << " нс" << endl;

cout << "Количество таких элементов: " << number << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 5:

{

if (isSorted)

{

int A;

cout << "Введите число a" << endl;

cin >> A;

InputCheck(A);

int indexA = BinarySearchFirstOccurrence(sortedArray, A, 0, N - 1);

cout << "Количество элементов, которые меньше a: " << indexA << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 6:

{

if (isSorted)

{

int B;

cout << "Введите число B" << endl;

cin >> B;

InputCheck(B);

int indexB = BinarySearchLastOccurrence(sortedArray, B, 0, N - 1);

cout << "Количество элементов, которые больше b: " << N - indexB - 1 << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 7:

{

if (isSorted)

{

int X;

cout << "Введите число X" << endl;

cin >> X;

InputCheck(X);

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

int xInArray = BinarySearch(sortedArray, X, 0, N - 1);

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

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

if (xInArray == -1)

cout << "Числа X нет в массиве" << endl;

else

cout << "Число X есть в массиве" << endl;

cout << "Алгоритм бинарного поиска занял " << duration.count() << " нс" << endl;

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

xInArray = ElemSearch(Array, X, N);

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

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

cout << "Алгоритм перебора занял " << duration.count() << " нс" << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 8:

{

int p, q;

cout << "Введите индекс первого элемента" << endl;

cin >> p;

InputCheck(p);

cout << "Введите индекс второго элемента" << endl;

cin >> q;

InputCheck(q);

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

swap(sortedArray[p], sortedArray[q]);

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

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

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

cout << sortedArray[i] << " ";

cout << endl << "Обмен занял " << duration.count() << " нс" << endl;

if (p != q)

isSorted = false;

break;

}

case 9:

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

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

cout << Array[i] << " ";

cout << endl;

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

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

cout << sortedArray[i] << " ";

cout << endl;

break;

case 666:

return 0;

default:

cout << "Нет команды с таким номером" << endl;

}

}

}