**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 гр. 3372 |  | Матвиенко А. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Научиться пользоваться массивами, применять сортировки, узнавать время на затраченную операцию, научиться проводить операции с элементами внутри массива, узнать преимущества и недостатки определенных сортировок, а также сравнить отсортированный и неотсортированный массивы.

**Основные теоретические положения.**

### Понятие массива

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. 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 |

### Вывод массивов

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

const int n = 10;

short A[n];

…

//  Для использования setw() необходимо включить #include <iomanip>

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

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

cout << endl;

На каждом шаге этого цикла в поток вывода отправляется очередной **i**-й элемент массива, при этомустанавливается ширина поля вывода, равная 8 позициям, выравнивание по левому краю. После окончания цикла вывода всех **n** элементов массива осуществляется переход на следующую строку экрана.

Обратим внимание на то, что в программах выгоднее задавать размеры массивов через именованные константы (в данном примере – константа **n**), для того чтобы использовать эти же константы для управления работой циклов. При необходимости изменить размеры массива достаточно будет поменять значение этой константы. При этом все циклы, использующие для управления своей работой эту константу, автоматически приспособятся к изменившимся размерам обрабатываемого массива.

Вывод двумерных массивов, как правило, осуществляется в табличной форме. Реализация такого алгоритма может быть, например, такой:

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;

}

### Ввод массивов

Ввод массивов также осуществляется с помощью циклических алгоритмов.

Простейший алгоритм ввода значений элементов одномерного массива может выглядеть так:

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

В этом фрагменте ввод значений элементов массива можно осуществлять и в виде строки целых чисел, разделенных пробелами или символами табуляции, и по одному значению с помощью клавиши **Enter**. Обратите внимание на последнюю строку этого фрагмента. Эта инструкция очищает поток ввода от оставшихся в нем данных. Если в потоке ввода после окончания цикла останутся какие-либо данные (например, мы ввели не 10 требуемых значений, а ошибочно 11 чисел), то следующие за циклом инструкции ввода возьмут их из потока, что может привести к неправильной работе программы.

Следующий фрагмент обеспечивает ввод  значений элементов массива с указанием номеров вводящихся элементов:

const int n = 10;

short A[n];

…

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

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

{

     cout << i + 1 << ": ";

     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)

             cin  >>  A [i] [j];

       //  Очищаем поток ввода от возможных лишних введенных чисел

       cin.sync();

### ****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.

### ****Шейкер-сортировка массива (shaker sort)****

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.

Затем процесс повторяется до тех пор, пока массив не будет отсортирован. За счет того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее. Очевидным примером этого был бы случай, когда в начале массива стоит максимальный элемент, а в конце массива – минимальный. Shaker sort справится с этим за 1 итерацию, при условии, что другие элементы стоят на правильном месте.

### ****Сортировка массива расчёской (comb sort)****

Очевидный недостаток bubble и shaker sort заключается в том, что элементы переставляются максимум на одну позицию.

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.

Оптимально изначально взять расстояние равным длине массива , а далее уменьшать его на определенный коэффициент, который примерно равен 1.247. Когда расстояние станет равно 1, выполняется обычная сортировка пузырьком.

### ****Сортировка массива вставками (insert sort)****

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 делить массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться. Причем найти нужное место для очередного элемента в отсортированном массиве достаточно легко.

Рассмотрим самый простой способ (рис. 3.5). Необходимо пройти массив слева направо и обработать каждый элемент. Слева будет наращиваться отсортированная часть массива, а справа – уменьшаться неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, что освобождает место в массиве и позволяет сдвинуть элементы и освободить точку вставки.

### ****Быстрая сортировка массива (quick sort)****

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. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

## 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.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая позволяет создавать целочисленный массив размерности N  = 100, где элементы массивы должны принимать случайное значение в диапазоне от -99 до 99. Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono. Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono. 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. Вывести количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем. Вывести количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем. Вывести информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализовать алгоритм бинарного поиска и сравнить скорость его работы с обычным перебором. Поменять местами элементы массива, индексы которых вводит пользователь. Вывести скорость обмена, используя библиотеку chrono.

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню | |
| При запуске программы перед пользователем появляется окно с меню, где он может выбрать номер задания последовательности. | Меню:    Проверка на ввод символов, которые не входят в диапазон выбора: |

Продолжение Таблицы

|  |  |
| --- | --- |
| Выполнение 1 задания | |
| При вводе пользователем корректного значения пункта меню программа выполняется. | Для выполнения первого задания нужно создать массив, состоящий из 100 элементов и заполнить его случайными в диапазоне от -99 до 99 |

|  |  |
| --- | --- |
| Выполнение 2 задания | |
| Пользователь выбирает сортировку и массив распределяет элементы в правильном порядке, используя эту сортировку |  |

|  |  |
| --- | --- |
| Выполнение 3 задания | |
| Производится поиск максимального и минимального элемента, а также время, затраченное на поиск. |  |
| Выполнение 4 задания | |
| Производится поиск среднего значения максимального и минимального элементов, а также индексы элементов, равных этому значению. |  |
| Выполнение 5 задания | |
| Выводит количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем. | Пользователь вводит число а |
| Выполнение 6 задания | |
| Выводит количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем. | Пользователь вводит число b |
| Выполнение 8 задания | |
| Меняет местами элементы массива, индексы которых вводит пользователь. |  |

**Выводы.**

В процессе выполнения работы я научился пользоваться массивами, применять сортировки, узнавать время на затраченную операцию, научился проводить операции с элементами внутри массива, узнал преимущества и недостатки определенных сортировок, а также сравнил отсортированный и неотсортированный массивы.

Приложение А

рабочий код

#include <iostream>

#include <conio.h>

#include <stdlib.h>

#include <thread>

#include <chrono>

using namespace std;

using namespace chrono;

int main()

{

const int size = 100;

srand(time(0));

int arrNotSorted[size];

int arrSorted[size];

int swap = 0;

cout << "исходный массив: \n";

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

{

arrNotSorted[i] = -99 + rand() % (99 - (-99) + 1);

arrSorted[i] = arrNotSorted[i];

cout << arrNotSorted[i] << " ";

}

cout << endl;

while (true)

{

cout << "1) Создать целочисленный массив размерности N = 100.\nЭлементы массива должны принимать случайное значение в диапазоне от -99 до 99.\n\n";

cout << "2) Отсортировать заданный в пункте 1 массив[…] сортировкой(от меньшего к большему).\nОпределить время, затраченное на сортировку, используя библиотеку chrono.\n\n";

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

cout << "4) Вывести среднее значение(если необходимо, число нужно округлить) максимального и минимального значения\n в отсортированном и неотсортированном.\nВывести индексы всех элементов, которые равны этому значению, и их количество.Подсчитать время поиска.\n\n";

cout << "5) Вывести количество элементов в отсортированном массиве, которые меньше числа a,\n которое инициализируется пользователем.\n\n";

cout << "6) Вывести количество элементов в отсортированном массиве, которые больше числа b,\n которое инициализируется пользователем.\n\n";

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

cout << "8) Поменять местами элементы массива, индексы которых вводит пользователь.\nВывести скорость обмена, используя библиотеку chrono.\n\n";

int maxNotSorted = 0;

int minNotSorted = 10000000;

int maxSorted = 0;

int minSorted = 10000000;

int number;

cin >> number;

switch (number)

{

case 1:

{

cout << "Массив уже создан." << endl;

break;

}

case 2:

{

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

nanoseconds result;

cout << "Выберите сортировку: " << endl;

cout << "1) Bubble sort (пузырьковая сортировка)" << endl;

cout << "2) Shaker sort (шейкер-сортировка)" << endl;

cout << "3) Insert sort(сортировка вставками)" << endl;

cout << "4) Comb sort (сортировка расчёской)" << endl;

cout << "5) Shell sort (сортировка Шелла) - ИДЗ 4" << endl;

int vvod;

cin >> vvod;

if (vvod == 1)

{

cout << "\nотсортированный массив: \n";

start = steady\_clock::now();

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

{

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

{

if (arrSorted[j] > arrSorted[j + 1])

{

swap = arrSorted[j];

arrSorted[j] = arrSorted[j + 1];

arrSorted[j + 1] = swap;

}

}

}

end = steady\_clock::now();

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

{

cout << arrSorted[i] << " ";

}

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

cout << endl << "массив отсортировался Bubble sort за " << result.count() << " наносекунд\n" << endl;

break;

}

if (vvod == 2)

{

cout << "\nотсортированный массив: \n";

start = steady\_clock::now();

for (int i = 0; i < size / 2; i++)

{

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

{

if (arrSorted[j] > arrSorted[j + 1])

{

swap = arrSorted[j];

arrSorted[j] = arrSorted[j + 1];

arrSorted[j + 1] = swap;

}

}

for (int j = size - i - 2; j > i; j--)

{

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

{

swap = arrSorted[j];

arrSorted[j] = arrSorted[j - 1];

arrSorted[j - 1] = swap;

}

}

}

end = steady\_clock::now();

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

{

cout << arrSorted[i] << " ";

}

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

cout << endl << "массив отсортировался Shaker sort за " << result.count() << " наносекунд\n" << endl;

break;

}

if (vvod == 3)

{

cout << "\nотсортированный массив: \n";

start = steady\_clock::now();

for (int i = 1; i < size; i++)

{

int key = arrSorted[i];

int j = i - 1;

while ((j >= 0) and (arrSorted[j] > key))

{

arrSorted[j + 1] = arrSorted[j];

j = j - 1;

}

arrSorted[j + 1] = key;

}

end = steady\_clock::now();

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

{

cout << arrSorted[i] << " ";

}

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

cout << endl << "массив отсортировался Insert sort за " << result.count() << " наносекунд\n" << endl;

break;

}

if (vvod == 4)

{

cout << "\nотсортированный массив: \n";

start = steady\_clock::now();

int swap;

float k = 1.247, S = size - 1;

int count = 0;

while (S >= 1)

{

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

{

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

{

swap = arrSorted[int(i + S)];

arrSorted[int(i + S)] = arrSorted[i];

arrSorted[i] = swap;

}

}

S /= k;

}

while (true)

{

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

{

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

{

swap = arrSorted[i + 1];

arrSorted[i + 1] = arrSorted[i];

arrSorted[i] = swap;

}

else count++;

}

if (count == size - 1)

break;

else

count = 0;

}

end = steady\_clock::now();

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

{

cout << arrSorted[i] << " ";

}

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

cout << endl << "массив отсортировался Comb sort за " << result.count() << " наносекунд\n" << endl;

break;

}

if (vvod == 5)

{

cout << "\nотсортированный массив: \n";

start = steady\_clock::now();

int i, j;

int memory = 0;

for (int step = size / 2; step > 0; step /= 2)

for (i = step; i < size; i++)

{

memory = arrSorted[i];

for (j = i; j >= step; j -= step)

{

if (memory < arrSorted[j - step])

arrSorted[j] = arrSorted[j - step];

else

break;

}

arrSorted[j] = memory;

}

end = steady\_clock::now();

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

{

cout << arrSorted[i] << " ";

}

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

cout << endl << "массив отсортировался Shell sort за " << result.count() << " наносекунд\n" << endl;

}

break;

}

case 3:

{

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> startMinNotSorted, endMinNotSorted, startMaxNotSorted, endMaxNotSorted, startMinSorted, endMinSorted, startMaxSorted, endMaxSorted;

nanoseconds resultMinNotSorted, resultMaxNotSorted, resultMinSorted, resultMaxSorted;

startMinNotSorted = steady\_clock::now();

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

{

if (arrNotSorted[i] < minNotSorted)

{

minNotSorted = arrNotSorted[i];

}

}

endMinNotSorted = steady\_clock::now();

startMaxNotSorted = steady\_clock::now();

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

{

if (arrNotSorted[i] > maxNotSorted)

{

maxNotSorted = arrNotSorted[i];

}

}

endMaxNotSorted = steady\_clock::now();

resultMinNotSorted = duration\_cast<nanoseconds>(endMinNotSorted - startMinNotSorted);

resultMaxNotSorted = duration\_cast<nanoseconds>(endMaxNotSorted - startMaxNotSorted);

cout << "В неотсортированном:" << endl;

cout << minNotSorted << " - минимальное, на вычисление потребовалось " << resultMinNotSorted.count() << " наносекунд" << endl;

cout << maxNotSorted << " - максимальное, на вычисление потребовалось " << resultMaxNotSorted.count() << " наносекунд\n" << endl;

startMinSorted = steady\_clock::now();

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

{

if (arrSorted[i] < minSorted)

{

minSorted = arrSorted[i];

}

}

endMinSorted = steady\_clock::now();

startMaxSorted = steady\_clock::now();

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

{

if (arrSorted[i] > maxSorted)

{

maxSorted = arrSorted[i];

}

}

endMaxSorted = steady\_clock::now();

resultMinSorted = duration\_cast<nanoseconds>(endMinSorted - startMinSorted);

resultMaxSorted = duration\_cast<nanoseconds>(endMaxSorted - startMaxSorted);

cout << "В отсортированном:" << endl;

cout << minSorted << " - минимальное, на вычисление потребовалось " << resultMinSorted.count() << " наносекунд" << endl;

cout << maxSorted << " - максимальное, на вычисление потребовалось " << resultMaxSorted.count() << " наносекунд\n" << endl;

break;

}

case 4:

{

int countNotSorted = 0;

int countSorted = 0;

int srValueNotSorted1;

int srValueSorted1;

int indexNotSorted;

int indexSorted;

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> startSrNotSorted, endSrNotSorted, startSrSorted, endSrSorted;

nanoseconds resultSrNotSorted, resultSrSorted;

startSrNotSorted = steady\_clock::now();

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

{

if (arrNotSorted[i] < minNotSorted)

{

minNotSorted = arrNotSorted[i];

}

}

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

{

if (arrNotSorted[i] > maxNotSorted)

{

maxNotSorted = arrNotSorted[i];

}

}

int srValueNotSorted;

srValueNotSorted = ((minNotSorted + maxNotSorted) \* 10) / 2;

if (srValueNotSorted < 0)

{

if ((srValueNotSorted \* (-1)) % 10 >= 5)

{

cout << srValueNotSorted / 10 - 1 << " - среднее значение макисмального и минимального в неотсортированном" << endl;

srValueNotSorted1 = srValueNotSorted / 10 - 1;

}

if (srValueNotSorted \* (-1) % 10 < 5)

{

cout << srValueNotSorted / 10 << " - среднее значение макисмального и минимального в неотсортированном" << endl;

srValueNotSorted1 = srValueNotSorted / 10;

}

}

if (srValueNotSorted >= 0)

{

if (srValueNotSorted % 10 >= 5)

{

cout << srValueNotSorted / 10 + 1 << " - среднее значение макисмального и минимального в неотсортированном" << endl;

srValueNotSorted1 = srValueNotSorted / 10 + 1;

}

if (srValueNotSorted % 10 < 5)

{

cout << srValueNotSorted / 10 << " - среднее значение макисмального и минимального в неотсортированном" << endl;

srValueNotSorted1 = srValueNotSorted / 10;

}

}

endSrNotSorted = steady\_clock::now();

resultSrNotSorted = duration\_cast<nanoseconds>(endSrNotSorted - startSrNotSorted);

cout << "На вычисление потребовалось " << resultSrNotSorted.count() << " наносекунд" << endl;

startSrSorted = steady\_clock::now();

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

{

if (arrSorted[i] < minSorted)

{

minSorted = arrSorted[i];

}

}

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

{

if (arrSorted[i] > maxSorted)

{

maxSorted = arrSorted[i];

}

}

int srValueSorted;

srValueSorted = ((minSorted + maxSorted) \* 10) / 2;

if (srValueSorted < 0)

{

if ((srValueSorted \* (-1)) % 10 >= 5)

{

cout << srValueSorted / 10 - 1 << " - среднее значение макисмального и минимального в отсортированном" << endl;

srValueSorted1 = srValueSorted / 10 - 1;

}

if (srValueSorted \* (-1) % 10 < 5)

{

cout << srValueSorted / 10 << " - среднее значение макисмального и минимального в отсортированном" << endl;

srValueSorted1 = srValueSorted / 10;

}

}

if (srValueSorted >= 0)

{

if (srValueSorted % 10 >= 5)

{

cout << srValueSorted / 10 + 1 << " - среднее значение макисмального и минимального в отсортированном" << endl;

srValueSorted1 = srValueSorted / 10 + 1;

}

if (srValueSorted % 10 < 5)

{

cout << srValueSorted / 10 << " - среднее значение макисмального и минимального в отсортированном" << endl;

srValueSorted1 = srValueSorted / 10;

}

}

endSrSorted = steady\_clock::now();

resultSrSorted = duration\_cast<nanoseconds>(endSrSorted - startSrSorted);

cout << "На вычисление потребовалось " << resultSrSorted.count() << " наносекунд" << endl;

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

{

if (arrNotSorted[i] == srValueNotSorted1)

{

cout << arrNotSorted[i] << " - элемент с индексом " << i << " в неосортированном" << endl;

countNotSorted += 1;

}

}

cout << "Всего таких элементов в неотсортированном массиве " << countNotSorted << endl;

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

{

if (arrSorted[i] == srValueSorted1)

{

cout << arrSorted[i] << " - элемент с индексом " << i << " в осортированном" << endl;

countSorted += 1;

}

}

cout << "Всего таких элементов в отсортированном массиве " << countSorted << endl;

break;

}

case 5:

{

cout << "Введите число" << endl;

int a;

int result = 0;

int i = 0;

cin >> a;

while (arrSorted[i] < a)

{

result += 1;

i += 1;

}

cout << "В массиве " << result << " меньших числа " << a << endl;

break;

}

case 6:

{

cout << "Введите число" << endl;

int b;

int result = 0;

int i = 99;

cin >> b;

while (arrSorted[i] > b)

{

result += 1;

i -= 1;

}

cout << "В массиве " << result << " больших числа " << b << endl;

break;

}

case 8:

{

int swap1 = 0;

int swap2 = 0;

cout << "Введите первый индекс элемента массива: " << endl;

int numb1;

cin >> numb1;

cout << "Введите второй индекс элемента массива: " << endl;

int numb2;

cin >> numb2;

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 100000000000000>>> startSwitchElements, endSwitchElements;

nanoseconds resultSwitchElements;

startSwitchElements = steady\_clock::now();

swap1 = arrSorted[numb1];

arrSorted[numb1] = arrSorted[numb2];

swap2 = swap1;

arrSorted[numb2] = swap2;

endSwitchElements = steady\_clock::now();

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

{

cout << arrSorted[i] << " ";

}

resultSwitchElements = duration\_cast<nanoseconds>(endSwitchElements - startSwitchElements);

cout << "\nЭлементы поменялись местами за " << resultSwitchElements.count() << " наносекунд" << endl;

break;

}

default:

cout << "Доступно только 8 заданий";

}

\_getch();

}

return 0;

}