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. 0324 _____ Жигалова Д.А.

Преподаватель _____ Глущенко А.Г

Санкт-Петербург
2020

Цель работы.

Знакомство с массивами и способами работы с ними, а также исследование различных способов сортировки элементов массивов.

Основные теоретические положения.

Массивы.

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.

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

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

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: A[8]. 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 A является тип данных int, с любым элементом этого массива можно выполнять любые операции, которые можно выполнять над значениями типа int.

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>

Виды сортировок.

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.

Очевидно, что хуже всего алгоритм будет работать, когда на вход подается массив, отсортированный в обратную сторону (от большего к меньшу). Быстрее же всего алгоритм работает с уже отсортированным массивом.

Но стандартный алгоритм пузырьковой сортировки предполагает полный циклический проход по массиву. Если изначально подается упорядоченная последовательность, то работа алгоритма все равно продолжиться. Исправить это можно, добавив условие проверки: если на текущей итерации ни один элемент не изменил свой индекс, то работа алгоритма прекращается.

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.

Кажется, что bubble sort теряет свою эффективность по сравнению с shaker sort. Сортировка проходит в массиве в обоих направлениях, а не только от его

начала к концу. Но в работе с большими массивами преимущество шейкер-сортировки уменьшается как раз из-за использования двух циклов.

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 находиться.

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.

Существует множество модификаций сортировки вставками, некоторые из них затрагивают именно способ вставки элемента в отсортированную часть. Одна из лучших модификаций – сортировка простыми вставками с бинарным поиском. Бинарный поиск будет описан позже.

Лучше всего сортировка вставками работает при обработке почти отсортированных массивов. В таком случае insert sort работает быстрее других сортировок.

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.

Постановка задачи.

Нужно разработать алгоритм и 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 решения поставленных была создана программа на языке программирования C++.

В первой задаче был использован цикл for, который заполнял созданный пустой массив случайными числами.

Во второй задаче для сортировки массива использовался метод Quick sort.

В третьей задаче для поиска максимального и минимального элементов в неотсортированном массиве использовался метод перебора всех элементов массива, а в отсортированном вывод первого и последнего элементов массива.

В четвертой задаче производится сложение первого и последнего элементов отсортированного массива и деление их суммы на два. После чего с помощью перебора ищутся идентичные элементы массива.

В пятой задаче перебираются все элементы отсортированного массива и подсчитывается сколько из них меньше числа, введённого пользователем.

В шестой задаче перебираются все элементы отсортированного массива и подсчитывается сколько из них больше числа, введённого пользователем.

В седьмой задаче реализуется алгоритм бинарного поиска и сравнивается его скорость работы со скорость простого перебора всех элементов.

В восьмой задаче мы запрашиваем у пользователя индексы элементов массива которые требуется поменять местами и используем для этого функцию swap.

Итоговый код программы:

```
#include <iostream>
#include <ctime>
#include <chrono>

using namespace std;

void quicksort(int* arr, int  end, int begin) {
    int mid;
    int f = begin;
    int l = end;
    mid = arr[(f + l) / 2];
    while (f < l)
    {
        while (arr[f] < mid) f++;
        while (arr[l] > mid) l--;
        if (f <= l)
        {
            swap(arr[f], arr[l]);
            f++;
            l--;
        }
    }
    if (begin < l) quicksort(arr, l, begin);
    if (f < end) quicksort(arr, end, f);
}

int binarySearch(int array[], int x, int low, int high) {

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (array[mid] == x)
            return mid;

        if (array[mid] < x)
            low = mid + 1;

        else
            high = mid - 1;
    }

    return -1;
}

int main()
{
    const int N = 100;
    int unsortArr[N];
    int arr[N];

    cout << "\nLet's make some array and sort it to realize other tasks\n";

    int answer;
    answer = 0;

    while (answer != 2) {
        cout << "\nTo make new array, enter 1\n";
        cout << "To exit, enter 2 \n";
        cin >> answer;
        system("CLS");

        switch (answer)
        {
```

```
        case (1):
        {
            // создаем рандомный массив из 100 чисел от -99 до 99
            for (int i = 0; i < N; i++) {
                unsortArr[i] = -99 + (rand() % 200);
            }

            for (int i = 0; i < N; ++i)
                arr[i] = unsortArr[i];

            // сортируем массив с помощью quick sort
            auto startQuick = std::chrono::high_resolution_clock::now();
            int end, begin;
            end = N - 1;
            begin = 0;
            quicksort(arr, end, begin);
            auto endQuick = std::chrono::high_resolution_clock::now();
            auto timeQuick = std::chrono::duration_cast<std::chrono::nanoseconds>(endQuick -
startQuick);

            cout << "\nTime spent on sorting (nanoseconds): " << std::fixed <<
timeQuick.count() << endl;

            int answer2;
            answer2 = 0;

            while (answer2 != 9) {
                cout << "\nUnsorted array: \n";
                for (int i = 0; i < N; i++)
                    cout << unsortArr[i] << " ";

                cout << "\n\nSorted array: \n";
                for (int i = 0; i < N; i++)
                    cout << arr[i] << " ";

                cout << "\n\nWhat tasks do you wanna check? (Write task's number)\n";
                cout << "3. Find the maximum and minimum element of the array. Calculate the
search time in sorted and unsorted arrays using the chrono library;\n";
                cout << "4. Output the average value of the maximum and minimum values.
Print the indexes of all elements that are equal to this value, and their number.\n";
                cout << "5. Print the number of elements in the sorted array that are less
than the number that is initialized by the user;\n";
                cout << "6. Print the number of elements in the sorted array that are
greater than the number that is initialized by the user;\n";
                cout << "7. Output information about whether the number entered by the user
is in the sorted array;\n";
                cout << "8. Swaps array elements whose indexes are entered by the user.\n";
                cout << "To make new array or exit, enter 9 \n";
                cin >> answer2;
                system("CLS");

                switch (answer2)
                {
                case (3):
                {
                    int minInUnsorted, maxInUnsorted;
                    minInUnsorted = 0;
                    maxInUnsorted = 0;

                    auto startFindingUn = std::chrono::high_resolution_clock::now();
                    for (int i = 0; i < N; i++)
                    {
                        if (unsortArr[i] <= minInUnsorted)
                        {
```

```
                        minInUnsorted = unsortArr[i];
                    }
                    if (unsortArr[i] >= maxInUnsorted)
                    {
                        maxInUnsorted = unsortArr[i];
                    }
                }
                cout << "\nMinimum element of an unsorted array: " << minInUnsorted;
                cout << "\nMaximum element of an unsorted array: " << maxInUnsorted;
                auto endFindingUn = std::chrono::high_resolution_clock::now();
                auto timeFindingUn =
std::chrono::duration_cast<std::chrono::nanoseconds>(endFindingUn - startFindingUn);
                cout << "\nTime spent on unsorted finding (nanoseconds): " << std::fixed
<< timeFindingUn.count() << endl;

                auto startFindingSo = std::chrono::high_resolution_clock::now();
                cout << "\nMinimum element of the sorted array: " << arr[0];
                cout << "\nMaximum element of the sorted array: " << arr[99];
                auto endFindingSo = std::chrono::high_resolution_clock::now();
                auto timeFindingSo =
std::chrono::duration_cast<std::chrono::nanoseconds>(endFindingSo - startFindingSo);
                cout << "\nTime spent on sorted finding (nanoseconds): " << std::fixed
<< timeFindingSo.count() << endl;

                break;
            }

            case (4):
                int average;

                average = round((arr[0] + arr[99]) / 2);
                cout << "\nThe average value of the maximum and minimum values: " <<
average;
                cout << "\nIndexes of all elements that are equal to this value: ";
                int numberInArray;
                numberInArray = 0;
                for (int i = 0; i < 100; i++) {
                    if (unsortArr[i] == average) {
                        numberInArray += 1;
                        cout << "\n" << i << "\n";
                    }
                }
                cout << "Total of these in the array: " << numberInArray << "\n";

                break;

            case (5):
                cout << "To search for elements in the sorted array that are less than
the number a, enter a: ";
                int a;
                int lowerThanA;
                lowerThanA = 0;
                cin >> a;
                for (int i = 0; i < N; i++) {
                    if (arr[i] < a) {
                        lowerThanA += 1;
                    }
                }
                cout << "\nElements less than a: " << lowerThanA << endl;
                break;

            case (6):
                cout << "To search for elements in the sorted array that are greater
than the number b, enter b: ";
```

```
                    int b;
                    int higherThanB;
                    higherThanB = 0;
                    cin >> b;
                    for (int i = 0; i < N; i++) {
                        if (arr[i] > b) {
                            higherThanB += 1;
                        }
                    }
                    cout << "\nElements greater than  b: " << higherThanB << endl;
                    break;

                case (7):
                {
                    int x;
                    bool answer7;
                    bool isExists = false;

                    cout << "\nUnsorted array: \n";
                    for (int i = 0; i < N; i++)
                        cout << unsortArr[i] << " ";

                    cout << "\n\nSorted array: \n";
                    for (int i = 0; i < N; i++)
                        cout << arr[i] << " ";

                    while (true) {
                        cout << "\nWhat integer do you want to find? \n";
                        cin >> x;

                        auto startSort = std::chrono::high_resolution_clock::now();
                        for (int i = 0; i < N; i++) {
                            if (arr[i] == x) {
                                isExists = true;
                                break;
                            }
                        }
                        auto endSort = std::chrono::high_resolution_clock::now();
                        auto timeSort =
std::chrono::duration_cast<std::chrono::nanoseconds>(endSort - startSort);
                        cout << "\nTime spent on regular sorted finding (nanoseconds): " <<
std::fixed << timeSort.count() << endl;

                        auto startBinary = std::chrono::high_resolution_clock::now();
                        int result = binarySearch(arr, x, 0, N - 1);
                        auto endBinary = std::chrono::high_resolution_clock::now();
                        auto timeBinary =
std::chrono::duration_cast<std::chrono::nanoseconds>(endBinary - startBinary);
                        cout << "\nTime spent on binary sorted finding (nanoseconds): " <<
std::fixed << timeBinary.count() << endl;

                        if (result == -1)
                            cout << "Not found" << "\n";
                        else
                            cout << "Element is found at index " << result << "\n";

                        cout << fixed << "\nDifference between Binary search and regular
search (nanoseconds): " << timeSort.count() - timeBinary.count() << "\n";

                        cout << "\nExit? 1/0";
                        cin >> answer7;
                        if (answer7) {
                            system("CLS");
                            break;
```

```
                        }
                    }
                    break;
                }

                case (8):
                {
                    bool answer8;
                    cout << "\nUnsorted array: \n";
                    for (int i = 0; i < N; i++)
                        cout << unsortArr[i] << " ";

                    while (true) {
                            cout << "\nTo swap two array elements, enter their indexes: \n";
                            int first;
                            int second;
                            cout << "First: ";
                            cin >> first;
                            cout << "Second: ";
                            cin >> second;
                            cout << "\n";

                            auto startSwap = std::chrono::high_resolution_clock::now();
                            swap(unsortArr[first], unsortArr[second]);
                            auto endSwap = std::chrono::high_resolution_clock::now();
                            auto timeSwap =
std::chrono::duration_cast<std::chrono::nanoseconds>(endSwap - startSwap);
                            cout << "\nTime spent on swaping(nanoseconds): " << std::fixed
<< timeSwap.count() << endl;

                            cout << "\nUnsorted array: \n";
                            for (int i = 0; i < N; i++)
                                cout << unsortArr[i] << " ";

                            cout << "\nExit? 1/0";
                            cin >> answer8;
                            if (answer8) {
                                system("CLS");
                                break;
                            }
                    }
                    break;
                }

                case (9):
                    break;

                default:
                    cout << "Oh, no! This task doesn't exist. Let's try again.\n";
                    break;
                }
            }
            break;
        }

        case (2):
            cout << "Have a nice day!\n";
            break;

        default:
            cout << "Oh, no! This task doesn't exist. Let's try again.\n";
            break;
        }
    }
```

```
    return 0;
}
```

Результат работы программы.

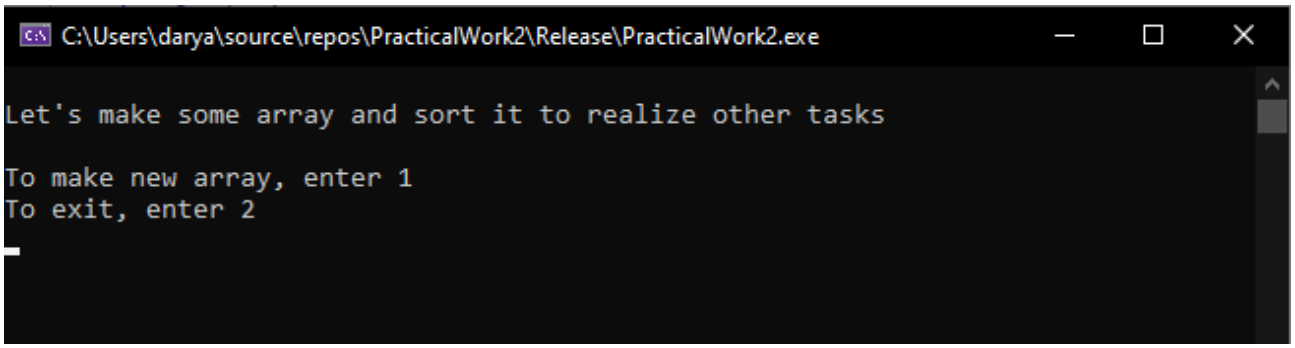

Рисунок 1 - Создание массива

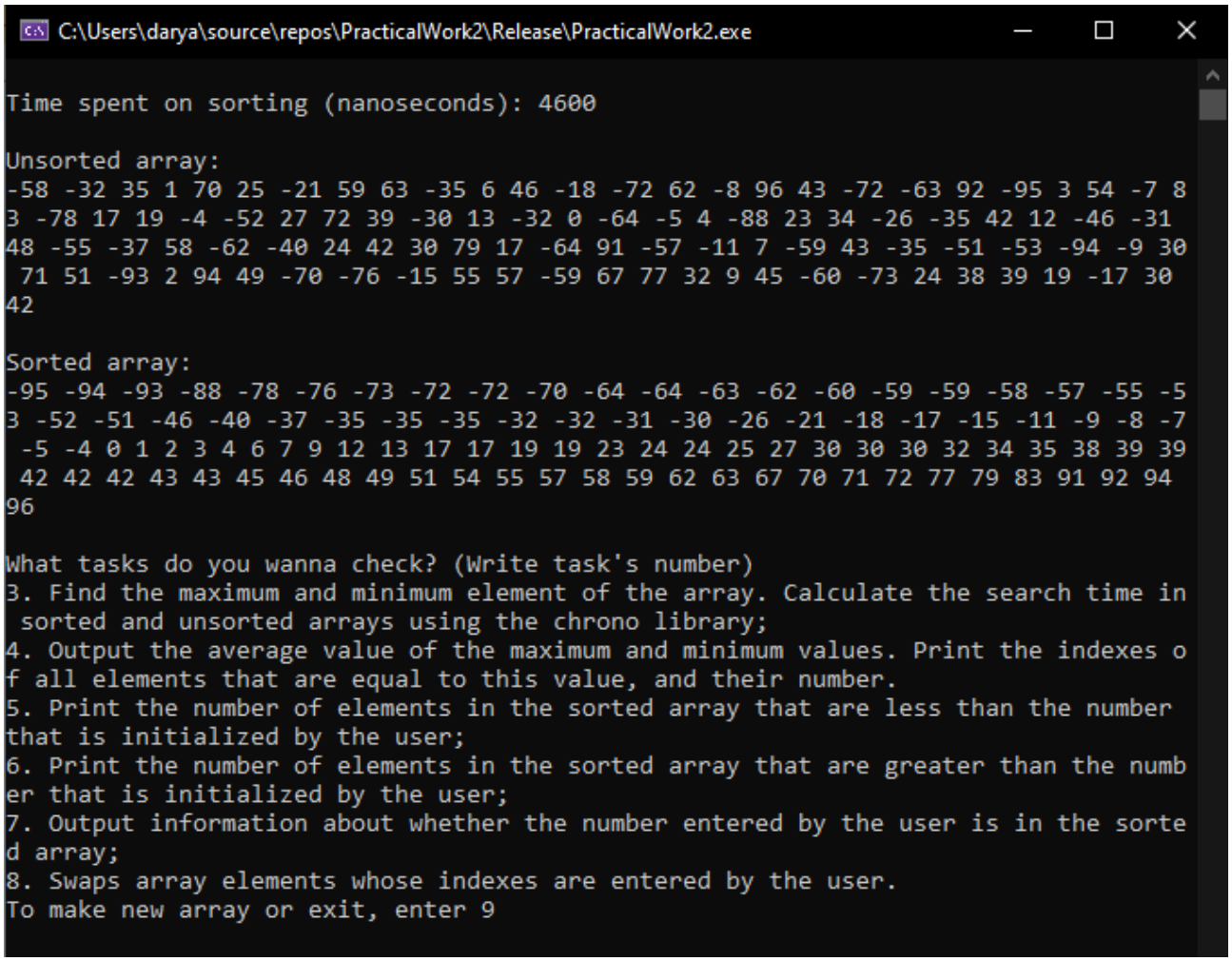

Рисунок 2 - Реализация первого и второго задания, а также меню

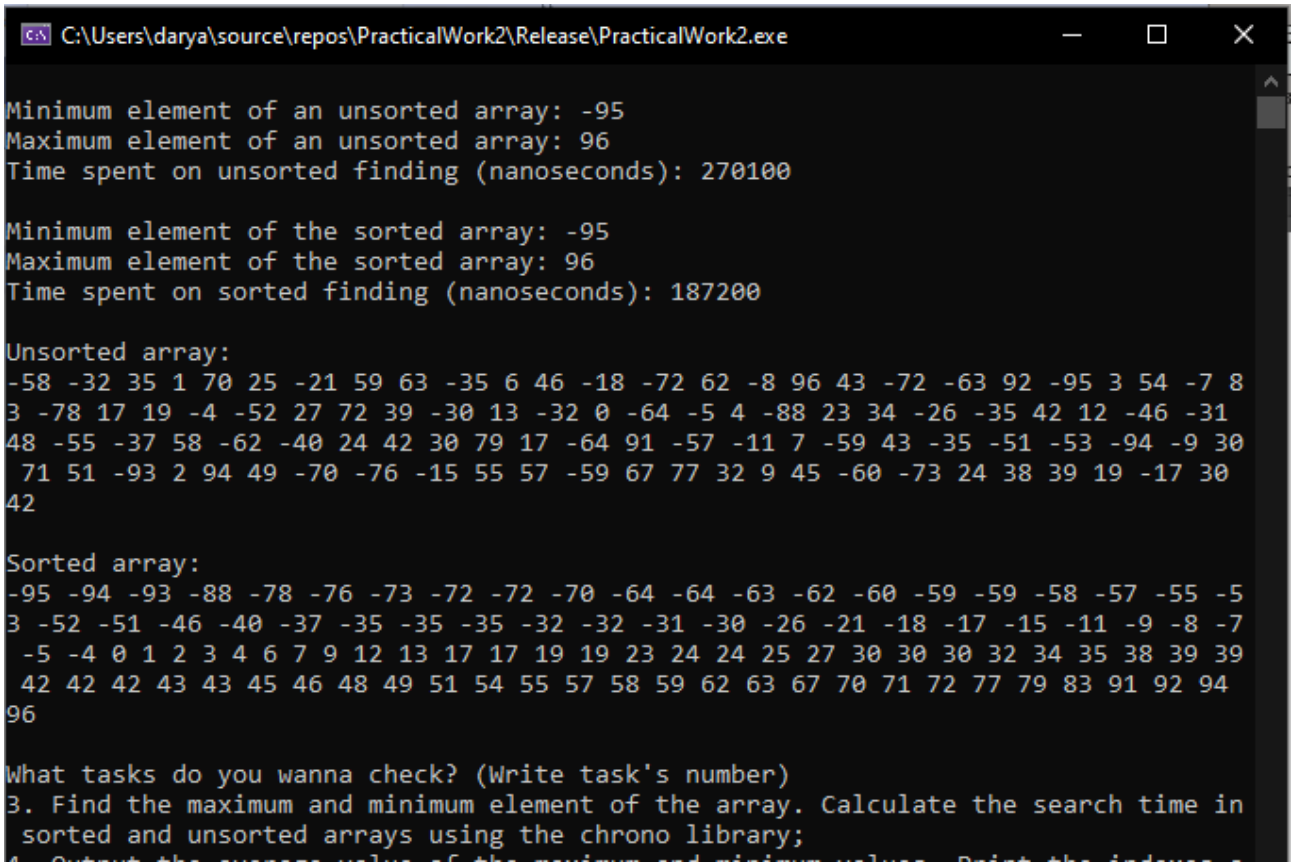

Рисунок 3 - Реализация 3 задания

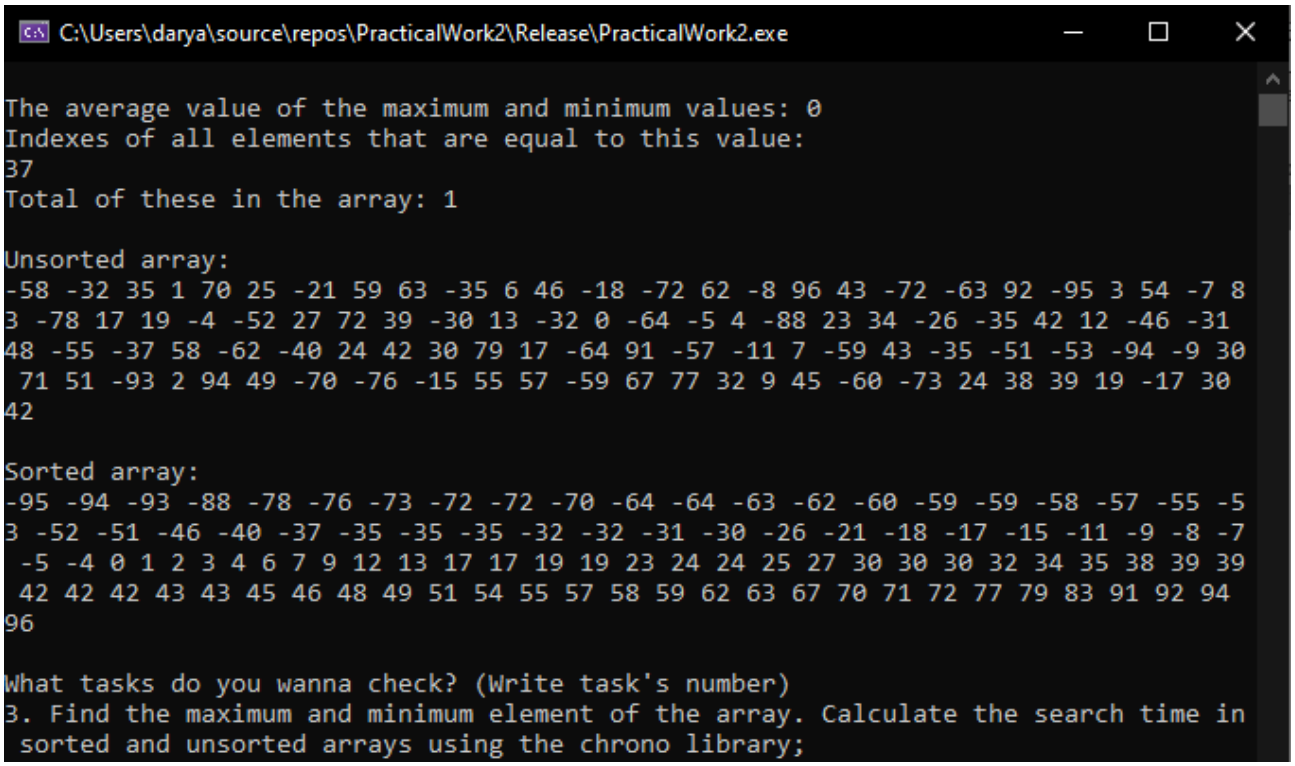

Рисунок 4 - Реализация 4 задания

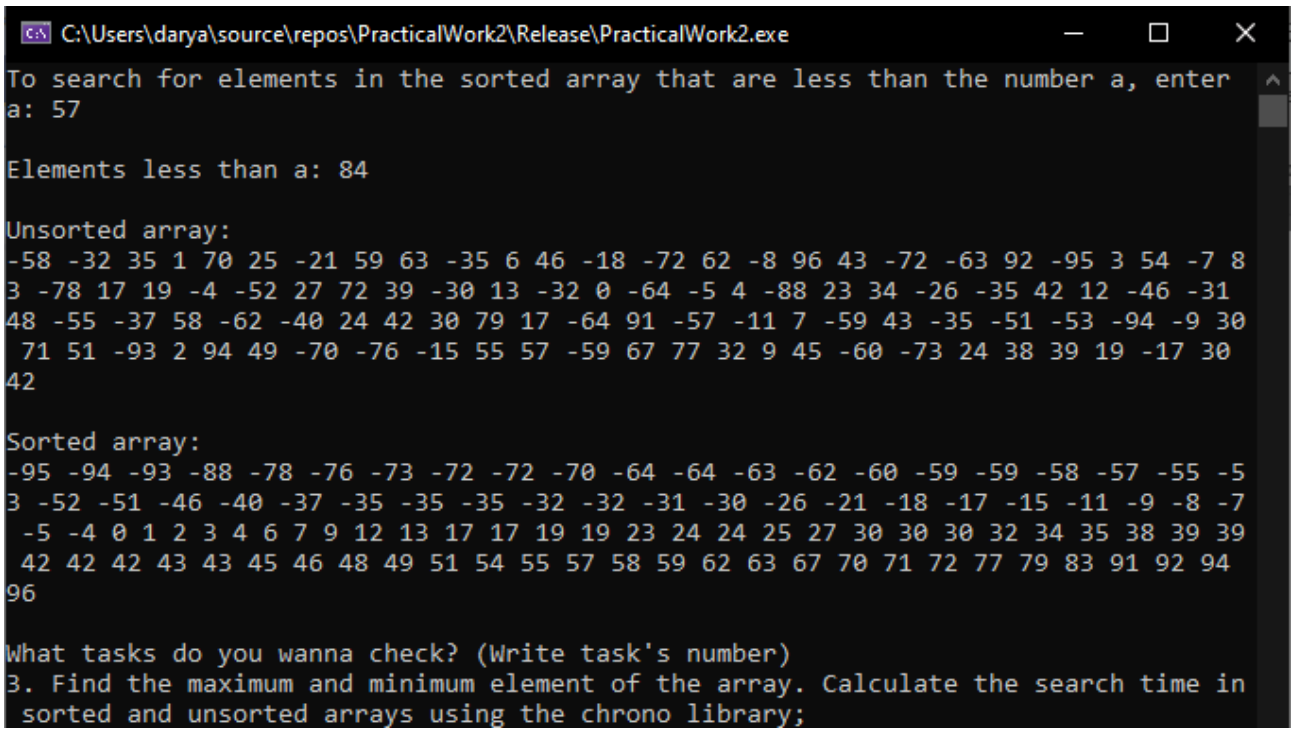

Рисунок 5 - Реализация 5 задания, где a = 57

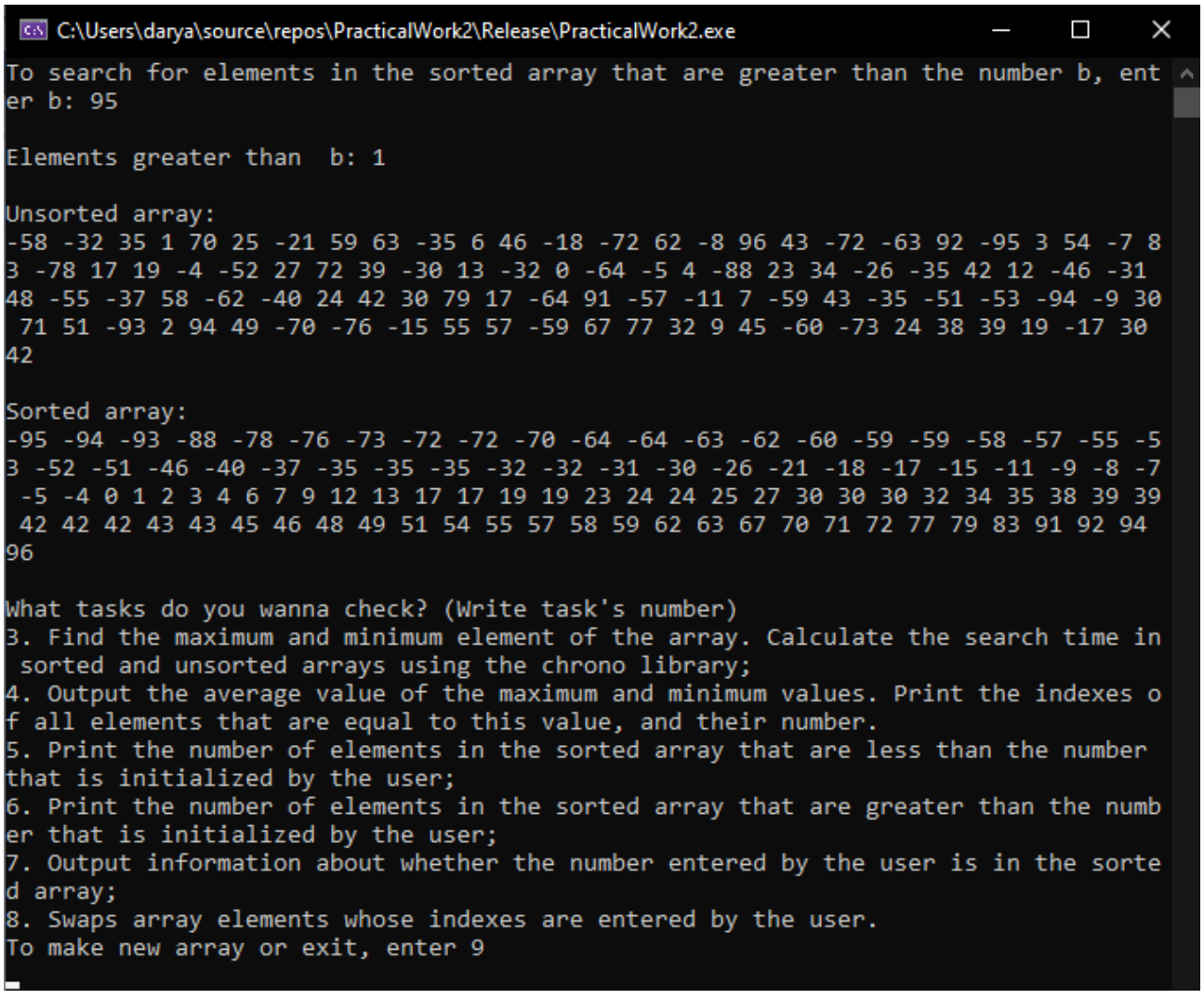

Рисунок 6 - Реализация 6 задания, где b = 95

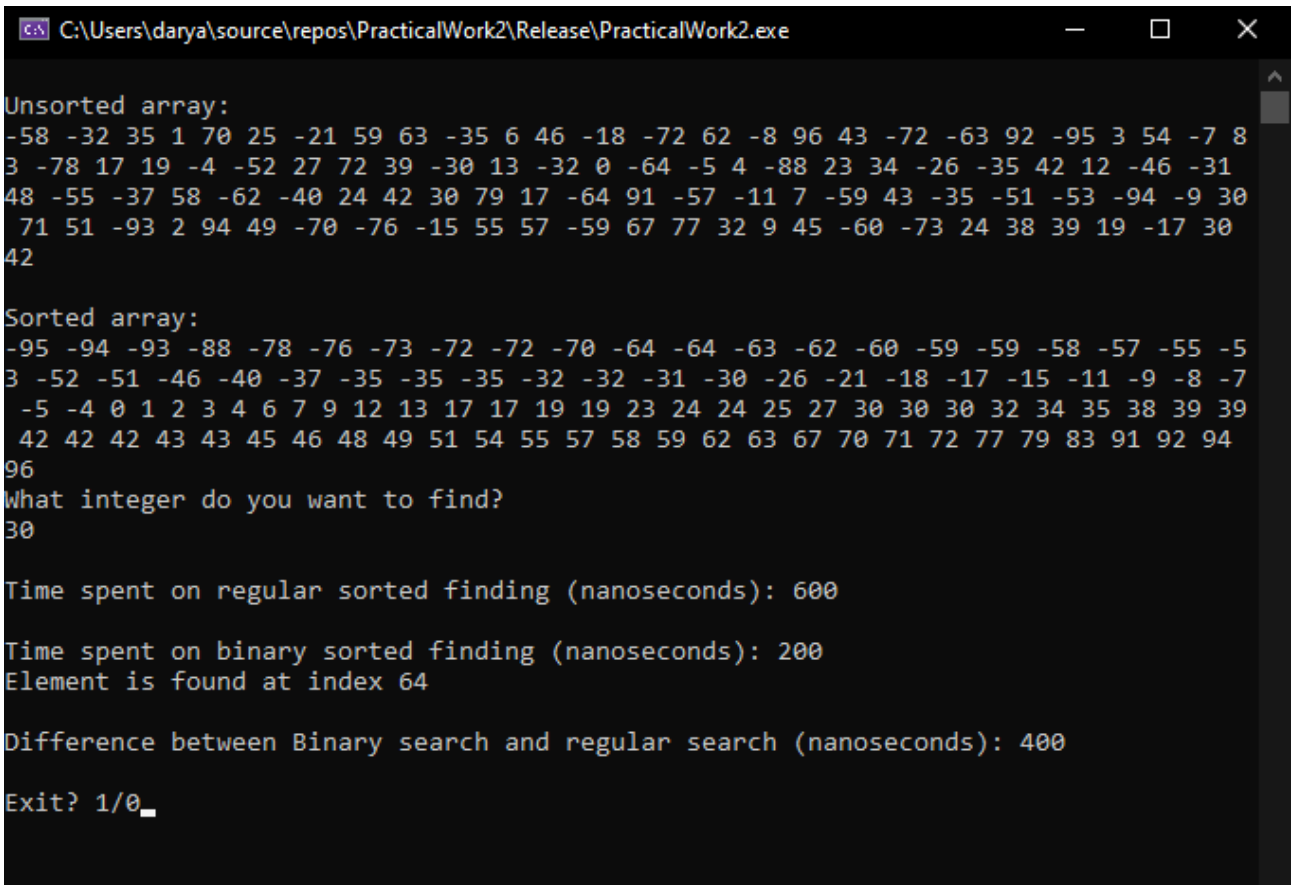

Рисунок 7 - Реализация 7 задания

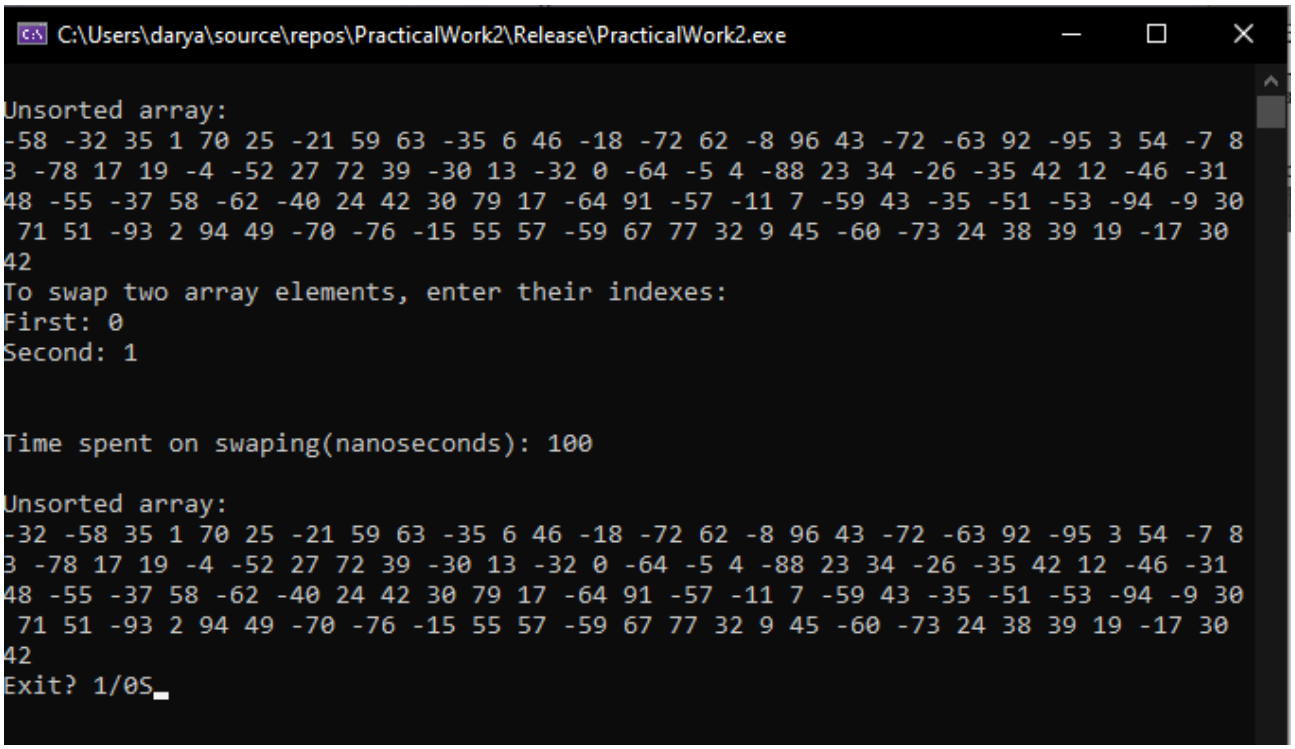

Рисунок 8 - Реализация 8 задания

Вывод.

При разработке программы мною была изучена быстрая сортировка. Также мною был изучен метод бинарного поиска, позволяющий значительно ускорить поиск необходимого элемента в отсортированном массиве с большим количеством элементов. Библиотека chrono позволяет замерять быстродействие отдельных функций и участков кода, что позволяет выбирать более рациональное решение поставленной проблемы.