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. 3373 Журухин Н.А.

Преподаватель Глущенко А. Г.

Санкт-Петербург
2023

Цель работы.

Научиться работать с массивами, изучить сортировки, библиотеку chrono, бинарный поиск. Разработать алгоритм и написать программу с помощью полученных знаний.

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.

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

Стандартные потоки ввода и вывода не “умеют” работать с массивами, поэтому ввод и вывод массивов необходимо реализовывать самостоятельно, обрабатывая массивы поэлементно. Большинство алгоритмов по обработке массивов реализуются с помощью циклов. Ввод и вывод массивов не являются исключением.

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.

Этот алгоритм является улучшенной версией алгоритма пузырьковой сортировки. Он более эффективен в ситуациях, когда большая часть элементов массива уже отсортирована.

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 расческой работает намного быстрее, чем bubble или shaker sort, в некоторых ситуациях comb sort работает быстрее quick sort. Но данная сортировка обладает одним очевидным минусом – неустойчивость.

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. 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.

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. На практике массив обычно делят на две части: «меньше опорного» и «равные и большие» или «меньше опорного или равные» и «большие». Такой поход в общем случае эффективнее, ведь упрощается алгоритм разделения. При том, что это один из самых быстродействующих из алгоритмов, данный алгоритм сортировки неустойчив, а прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стека.

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.

Бинарный поиск работает только в том случае, если массив отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1) Определение значения в середине массива (или иной структуры данных). 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.

Постановка задачи.

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 массив сортировками (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

3) Найти максимальный и минимальный элемент массива. Подсчитать время поиска этих элементов в отсортированном массиве и не отсортированном, используя библиотеку chrono.

4) Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и не отсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитать 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. Реализовать алгоритм бинарного поиска. Сравнить скорость его работы с обычным перебором.

8) Меняет 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 А.

Блок описания кода и использованных алгоритмов

Программа выполняет 8 задач:

1. Создается целочисленный массив размерности $N = 100$. Элементы массивы принимают случайное значение в диапазоне от -99 до 99.
2. Заданный в пункте 1 массив сортируется (от меньшего к большему). Используются сортировки: “Bubble sort”, “Shaker sort”, “Comb sort”,

“Insert sort”, “Quick sort”. Определяется время, затраченное на сортировку, с помощью библиотеки chrono.

3. Находится максимальный и минимальный элементы массива. Считается время поиска этих элементов в отсортированном и не отсортированном массивах с помощью библиотеки chrono.

4. Выводится среднее значение максимального и минимального значения в отсортированном и не отсортированном массивах. Выводятся индексы всех элементов, которые равны этому значению, и их количество. Считается время поиска.

5. Выводится количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.
6. Выводится количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.
7. Выводится информация о том, есть ли введенное пользователем число в отсортированном массиве. Реализован алгоритм бинарного поиска. Сравнивается скорость его работы с обычным перебором.

8. Меняются местами элементы массива, индексы которых вводит пользователь. Выводится скорость обмена с помощью библиотеки chrono.

Блок скриншотов работы программы

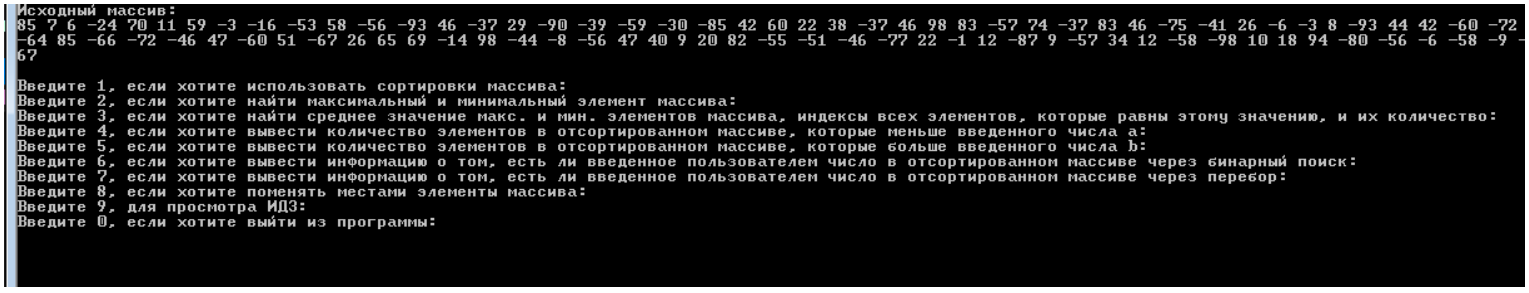

Рис. 1 Меню

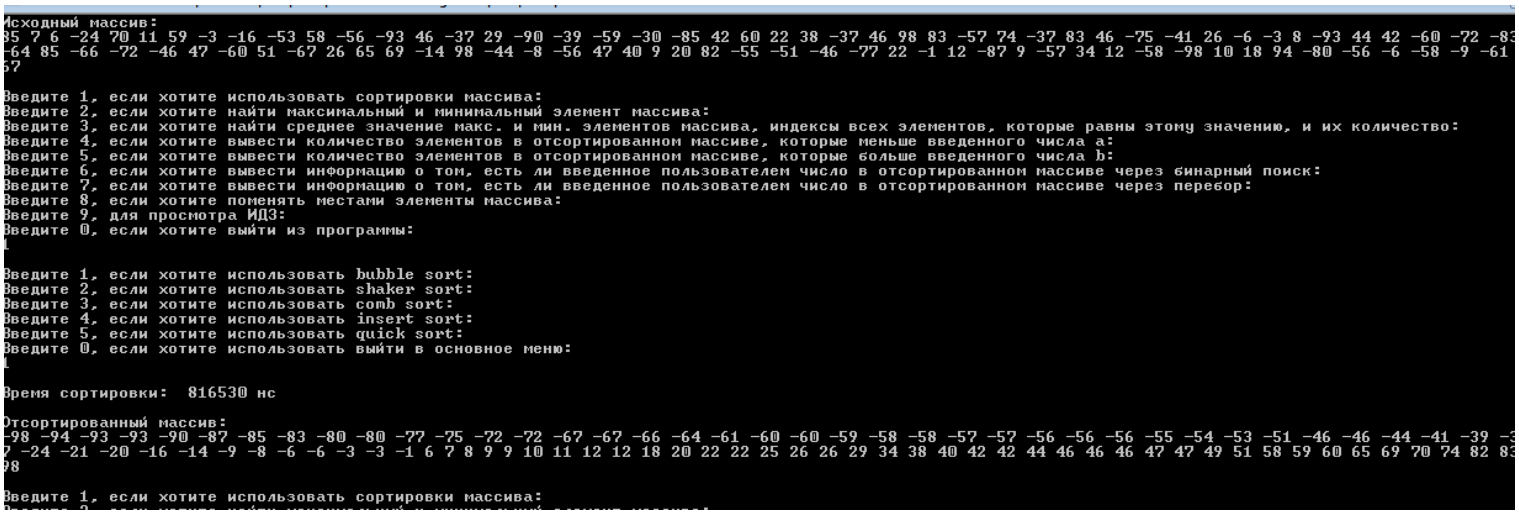

Рис. 2 Пример сортировки

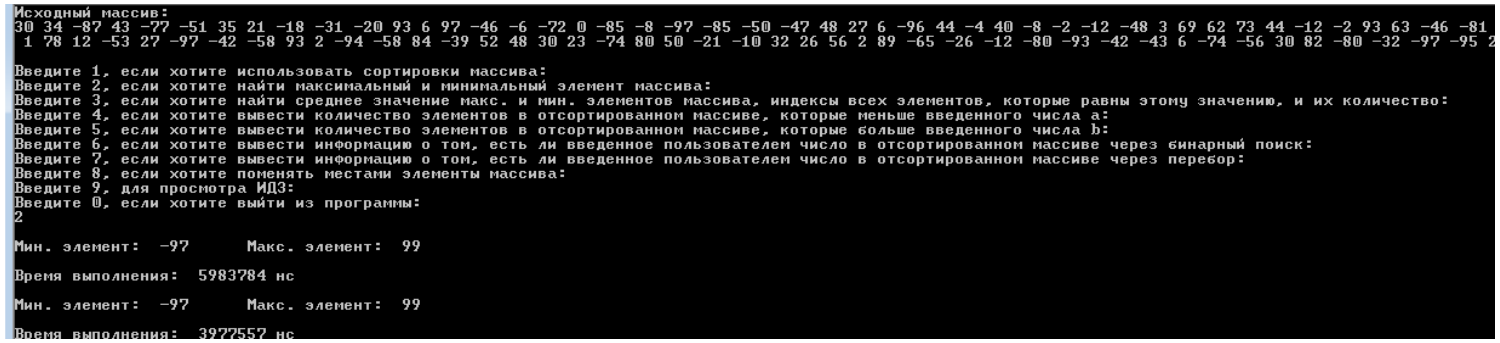

Рис. 3 Максимальный и минимальны элементы массива

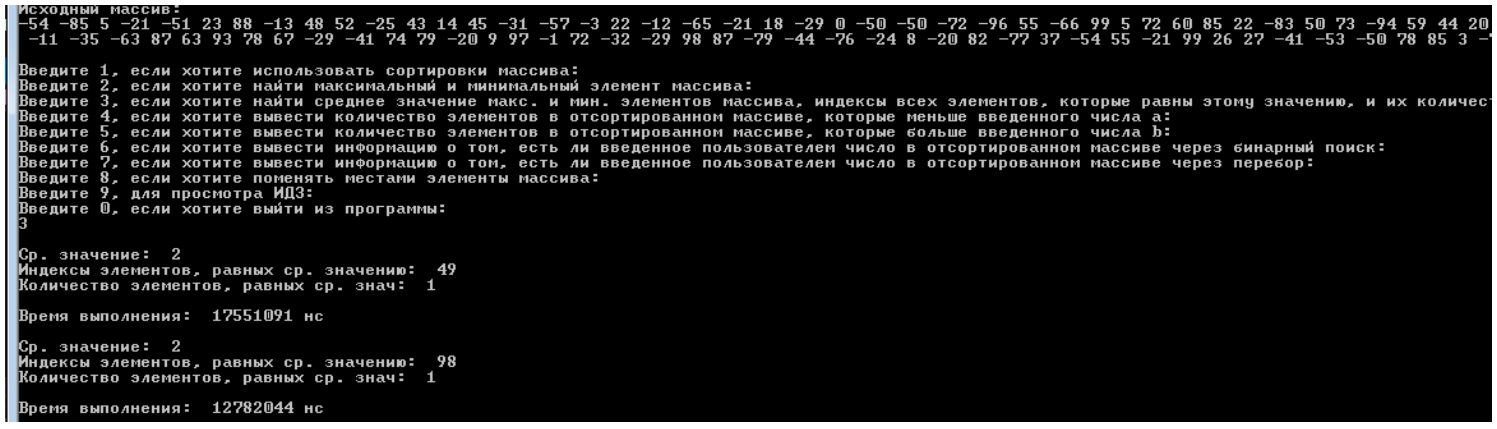

Рис. 4 Среднее значение

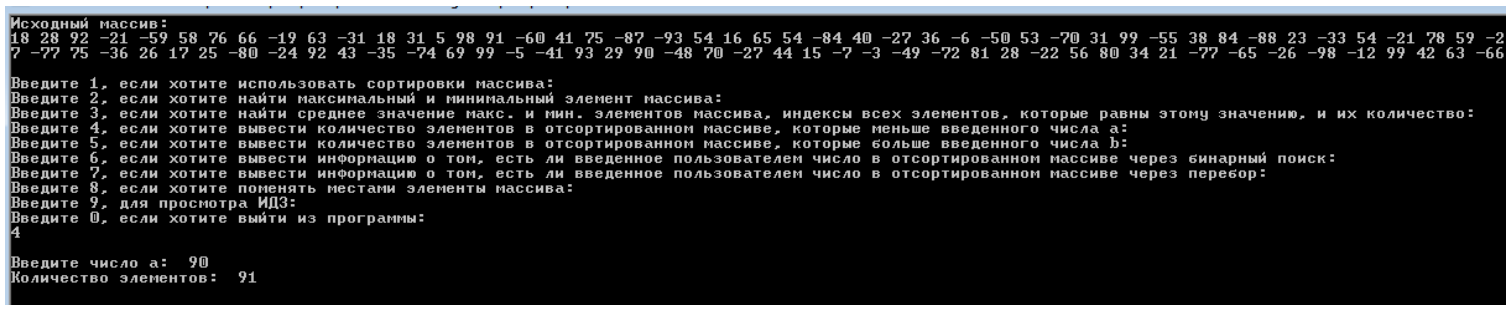

Рис. 5 Элементы, которые меньше числа a

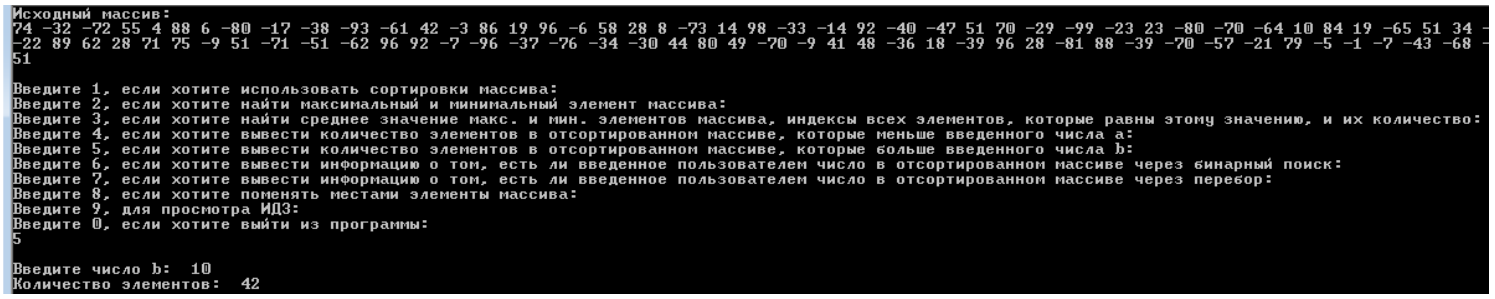

Рис. 6 Элементы, которые больше числа b

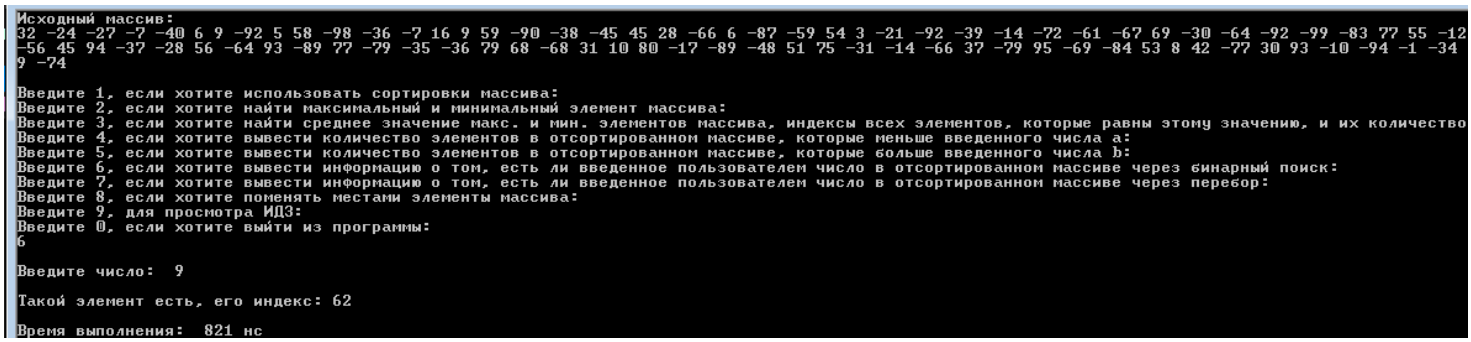

Рис. 7 Поиск через бинарный поиск

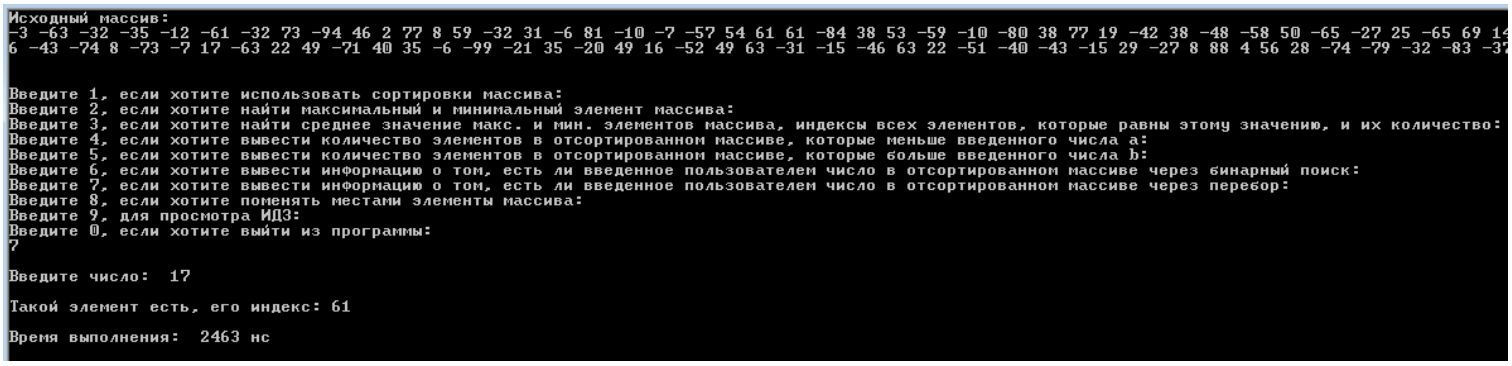

Рис. 8 Поиск через перебор

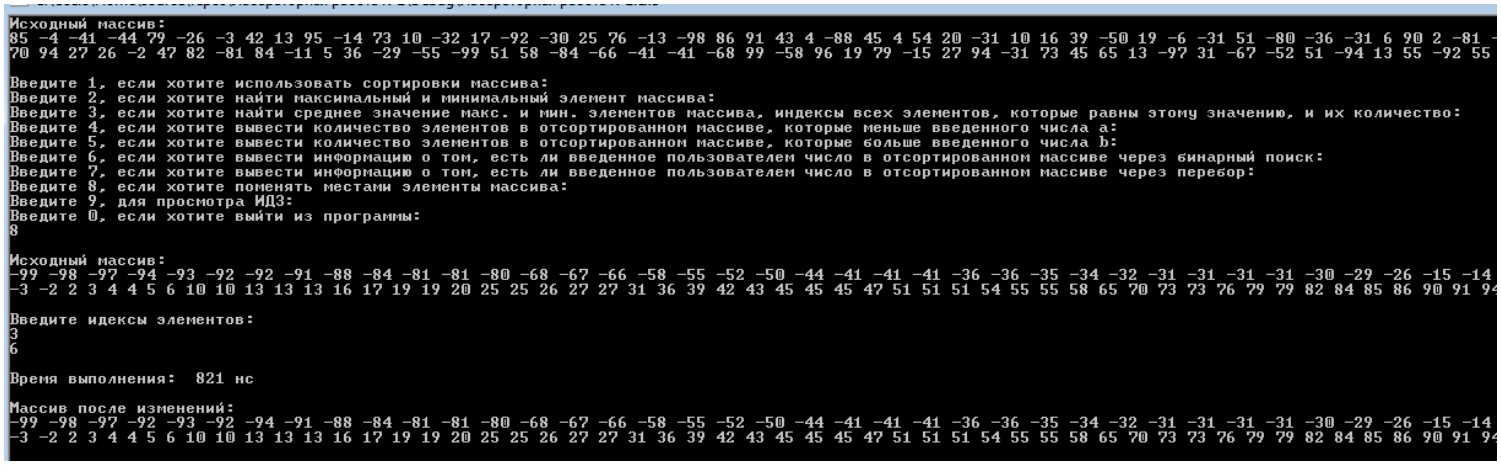

Рис. 9 Смена элементов массива

Выводы.

Выполнив данную лабораторную работу, я научился работать с массивами, изучил сортировки, библиотеку chrono, алгоритм бинарного поиска.

ПРИЛОЖЕНИЕ А
РАБОЧИЙ КОД

```
#include <iostream>
#include <iomanip>
#include <time.h>
#include <chrono>
using namespace std;
using namespace chrono;

void menu() {
    cout << "Введите 1, если хотите использовать сортировки массива:\n";
    cout << "Введите 2, если хотите найти максимальный и минимальный элемент
массива:\n";
    cout << "Введите 3, если хотите найти среднее значение макс. и мин.
элементов массива, индексы всех элементов, которые равны этому значению, и их
количество:\n";
    cout << "Введите 4, если хотите вывести количество элементов в
отсортированном массиве, которые меньше введенного числа a:\n";
    cout << "Введите 5, если хотите вывести количество элементов в
отсортированном массиве, которые больше введенного числа b:\n";
    cout << "Введите 6, если хотите вывести информацию о том, есть ли введенное
пользователем число в отсортированном массиве через бинарный поиск:\n";
    cout << "Введите 7, если хотите вывести информацию о том, есть ли введенное
пользователем число в отсортированном массиве через перебор:\n";
    cout << "Введите 8, если хотите поменять местами элементы массива:\n";
    cout << "Введите 9, для просмотра ИДЗ:\n";
    cout << "Введите 0, если хотите выйти из программы:\n";
}
void menu2() {
    cout << "Введите 1, если хотите использовать bubble sort:\n";
    cout << "Введите 2, если хотите использовать shaker sort:\n";
    cout << "Введите 3, если хотите использовать comb sort:\n";
    cout << "Введите 4, если хотите использовать insert sort:\n";
    cout << "Введите 5, если хотите использовать quick sort:\n";
    cout << "Введите 0, если хотите использовать выйти в основное меню:\n";
}

int bubble_sort(int Arr[], int N) {
    bool flag;
    for (int j = 0; j < N - 1; ++j) {
        flag = false;
        for (int i = 0; i < N - 1 - j; ++i) {
            if (Arr[i] > Arr[i + 1]) {
                swap(Arr[i], Arr[i + 1]);
                flag = true;
            }
        }
        if (flag == false) {
            break;
        }
    }
    return 0;
}

int shaker_sort(int Arr[], int N) {
    bool flag;
    for (int j = 0; j < N / 2; ++j) {
        flag = false;
        for (int i = j; i < N - 1 - j; ++i) {
            if (Arr[i] > Arr[i + 1]) {
```

```
                swap(Arr[i], Arr[i + 1]);
                flag = true;
            }
        }
        for (int k = N - j - 2; k >= j; --k) {
            if (Arr[k] < Arr[k - 1]) {
                swap(Arr[k], Arr[k - 1]);
                flag = true;
            }
        }
        if (flag == false) {
            break;
        }
    }
    return 0;
}

int comb_sort(int Arr[], int N) {
    int swap;
    float k = 1.247, S = N - 1;
    int count = 0;
    while (S >= 1) {
        for (int i = 0; i + S < N; i++) {
            if (Arr[i] > Arr[int(i + S)]) {
                swap = Arr[int(i + S)];
                Arr[int(i + S)] = Arr[i];
                Arr[i] = swap;
            }
        }
        S /= k;
    }
    while (true) {
        for (int i = 0; i < N - 1; i++) {
            if (Arr[i] > Arr[i + 1]) {
                swap = Arr[i + 1];
                Arr[i + 1] = Arr[i];
                Arr[i] = swap;
            }
            else count++;
        }
        if (count == N - 1)
            break;
        else
            count = 0;
    }
    return 0;
}

int insert_sort(int Arr[], int N) {
    int perem, elem;
    for (int i = 1; i < N; ++i) {
        perem = Arr[i];
        elem = i;
        for (int j = i - 1; j >= 0 && Arr[j] > perem; --j) {
            Arr[j + 1] = Arr[j];
            elem = j;
        }
        Arr[elem] = perem;
    }
    return 0;
}
```

```
    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);
    }

    void search_elem_sort(int Arr[]) {
        int min_elem, max_elem;
        min_elem = Arr[0];
        max_elem = Arr[99];
        cout << "Мин. элемент:  " << min_elem << '\t' << "Макс. элемент:  " <<
max_elem;
    }

    void search_elem(int Arr[], int N) {
        int min_elem2 = Arr[0], max_elem2 = Arr[0];
        for (int i = 1; i < N; i++) {
            if (Arr[i] < min_elem2) {
                min_elem2 = Arr[i];
            }
            if (Arr[i] > max_elem2) {
                max_elem2 = Arr[i];
            }
        }
        cout << "Мин. элемент:  " << min_elem2 << '\t' << "Макс. элемент:  " <<
max_elem2;
    }

    void average_value_sort(int Arr[], int N) {
        int min_elem, max_elem, average;
        min_elem = Arr[0];
        max_elem = Arr[99];
        if (((min_elem + max_elem) % 2 == 1) || ((min_elem + max_elem) % 2 == -1))
{
            average = (min_elem + max_elem) / 2 + ((min_elem + max_elem) % 2);
        }
        else {
            average = (min_elem + max_elem) / 2;
        }
        cout << "Ср. значение:  " << average << "\n";
        int count = 0;
        cout << "Индексы элементов, равных ср. значению:  ";
        for (int i = 0; i < N; i++) {
            if (Arr[i] == average) {
                cout << i << '\t';
                count++;
            }
        }
```

```
        cout << '\n' << "Количество элементов, равных ср. знач:  " << count;
    }

    void average_value(int Arr[], int N) {
        int min_elem2 = Arr[0], max_elem2 = Arr[0], average2;
        for (int i = 1; i < N; i++) {
            if (Arr[i] < min_elem2) {
                min_elem2 = Arr[i];
            }
            if (Arr[i] > max_elem2) {
                max_elem2 = Arr[i];
            }
        }
        if ((min_elem2 + max_elem2) % 2 == 1 || ((min_elem2 + max_elem2) % 2 == -
1)) {
            average2 = (min_elem2 + max_elem2) / 2 + ((min_elem2 + max_elem2) % 2);
        }
        else {
            average2 = (min_elem2 + max_elem2) / 2;
        }
        cout << "Ср. значение:  " << average2 << "\n";
        int count2 = 0;
        cout << "Индексы элементов, равных ср. значению:  ";
        for (int i = 0; i < N; i++) {
            if (Arr[i] == average2) {
                cout << i << '\t';
                count2++;
            }
        }
        cout << '\n' << "Количество элементов, равных ср. знач:  " << count2;
    }

    void min_numbers(int Arr[], int N) {
        int a;
        cout << "Введите число a:  ";
        cin >> a;
        int count = 0;
        while (Arr[count] < a && count < N) {
            count++;
        }
        cout << "Количество элементов:  " << count << "\n\n";
    }

    void max_numbers(int Arr[], int N) {
        int b;
        cout << "Введите число b:  ";
        cin >> b;
        int i = N - 1;
        int count = 0;
        while (Arr[i] > b && i >= 0) {
            count++;
            i--;
        }
        cout << "Количество элементов:  " << count << "\n\n";
    }

    int binarySearch(int Arr[], int value, int start, int end) {
        if (end >= start) {
            int mid = start + (end - start) / 2;

            if (Arr[mid] == value) {
                return mid;
```

```
            }

            if (Arr[mid] > value) {
                return binarySearch(Arr, value, start, mid - 1);
            }

            return binarySearch(Arr, value, mid + 1, end);
        }

        return -1;
    }

    void search_value(int Arr[], int value, int N) {
        time_point<steady_clock, duration<__int64, ratio<1, 1000000000>>> start,
end;
        nanoseconds result;
        start = steady_clock::now();
        bool flag = false;
        for (int i = 0; i < N; i++) {
            if (Arr[i] == value) {
                end = steady_clock::now();
                cout << "Такой элемент есть, его индекс: " << i;
                flag = true;
                break;
            }
        }
        if (flag == false) {
            end = steady_clock::now();
            cout << "Такого элемента нет";
        }
        cout << "\n\n";
        result = duration_cast<nanoseconds>(end - start);
        cout << "Время выполнения:  ";
        cout << result.count() << " нс";
        cout << "\n\n";
    }

    void swapElem(int Arr[], int ind1, int ind2) {
        time_point<steady_clock, duration<__int64, ratio<1, 1000000000>>> start,
end;
        nanoseconds result;
        start = steady_clock::now();
        int perem = Arr[ind1];//
        Arr[ind1] = Arr[ind2];
        Arr[ind2] = perem;//
        end = steady_clock::now();
        result = duration_cast<nanoseconds>(end - start);
        cout << "Время выполнения:  ";
        cout << result.count() << " нс";
        cout << "\n\n";
    }



    void element_reduction(int Arr[], int N) {
        int num;
        cout << "Введите число, на которое хотите уменьшить нечетные элементы
массива:  ";
        cin >> num;
        cout << "\n\n";
        for (int i = 1; i < N; i += 2) {
            Arr[i] -= num;
```

```
        }
    }

    void element_multiplication(int Arr[], int N) {
        int num2;
        cout << "Числа, на которые происходит умножение:\n";
        for (int i = 1; i < N; i += 2) {
            num2 = rand() % 9 + 1;
            cout << num2 << " ";
            Arr[i] *= num2;
        }
    }

    void sum_elem(int Arr[], int N) {
        int count = 0, count1 = 0, count2 = 0, count3 = 0, count4 = 0, count5 = 0,
count6 = 0, count7 = 0, count8 = 0, count9 = 0;
        for (int i = 0; i < N; i++) {
            if (Arr[i] % 2 == 0 && Arr[i] % 3 == 0 && Arr[i] % 4 == 0 && Arr[i] % 5
== 0 && Arr[i] % 6 == 0 && Arr[i] % 7 == 0 && Arr[i] % 8 == 0 && Arr[i] % 9 == 0) {
                count++;
                cout << Arr[i] << "\n";
            }
            if (Arr[i] % 1 == 0) {
                count1++;
            }
            if (Arr[i] % 2 == 0) {
                count2++;
            }
            if (Arr[i] % 3 == 0) {
                count3++;
            }
            if (Arr[i] % 4 == 0) {
                count4++;
            }
            if (Arr[i] % 5 == 0) {
                count5++;
            }
            if (Arr[i] % 6 == 0) {
                count6++;
            }
            if (Arr[i] % 7 == 0) {
                count7++;
            }
            if (Arr[i] % 8 == 0) {
                count8++;
            }
            if (Arr[i] % 9 == 0) {
                count9++;
            }
        }
        cout << "Количество чисел, которые нацело делятся на 1, 2, 3, 4, 5, 6, 7,
8, 9: " << count << "\n";
        cout << "Количество чисел, которые нацело делятся на 1: " << count1 <<
"\n";
        cout << "Количество чисел, которые нацело делятся на 2: " << count2 <<
"\n";
        cout << "Количество чисел, которые нацело делятся на 3: " << count3 <<
"\n";
        cout << "Количество чисел, которые нацело делятся на 4: " << count4 <<
"\n";
        cout << "Количество чисел, которые нацело делятся на 5: " << count5 <<
"\n";
```

```
        cout << "Количество чисел, которые нацело делятся на 6: " << count6 <<
"\n";
        cout << "Количество чисел, которые нацело делятся на 7: " << count7 <<
"\n";
        cout << "Количество чисел, которые нацело делятся на 8: " << count8 <<
"\n";
        cout << "Количество чисел, которые нацело делятся на 9: " << count9 <<
"\n\n";
    }


    int main() {
        setlocale(LC_ALL, "Russian");
        time_point<steady_clock, duration<__int64, ratio<1, 1000000000>>> start,
end;
        nanoseconds result;
      srand(time(NULL));
        cout << "Исходный массив:\n";
      const int N = 100;
      int A[N];
      for (int i = 0; i < N; i++) {
            A[i] = rand() % 199 - 99;
            cout << A[i] << " ";
      }
      cout << "\n\n";

        int B[N];
        for (int i = 0; i < N; i++) {
            B[i] = A[i];
        }

        int C[N];
        for (int i = 0; i < N; i++) {
            C[i] = A[i];
        }

        int D[N];
        for (int i = 0; i < N; i++) {
            D[i] = A[i];
        }

        int E[N];
        for (int i = 0; i < N; i++) {
            E[i] = A[i];
        }

        int F[N];
        for (int i = 0; i < N; i++) {
            F[i] = A[i];
        }

        int G[N];
        for (int i = 0; i < N; i++) {
            G[i] = A[i];
        }

        int H[N];
        for (int i = 0; i < N; i++) {
            H[i] = A[i];
        }

        int I[N];
```

```
for (int i = 0; i < N; i++) {
    I[i] = A[i];
}

int ends = N - 1, begin = 0;
menu();
int i2;
cin >> i2;
cout << "\n";
while (i2 != 0) {
    switch (i2) {
        case 0:
            exit(0);
        case 1:
            menu2();
            int i;
            cin >> i;
            switch (i) {
                case 1:
                    start = steady_clock::now();
                    bubble_sort(A, N);
                    end = steady_clock::now();
                    result = duration_cast<nanoseconds>(end - start);
                    cout << "\n";
                    cout << "Время сортировки:  ";
                    cout << result.count() << " нс";
                    cout << "\n\n";
                    cout << "Отсортированный массив:\n";
                    for (int i = 0; i < N; ++i) {
                        cout << A[i] << " ";
                    }
                    cout << "\n\n";
                    menu();
                    break;
                case 2:
                    start = steady_clock::now();
                    shaker_sort(B, N);
                    end = steady_clock::now();
                    result = duration_cast<nanoseconds>(end - start);
                    cout << "\n";
                    cout << "Время сортировки:  ";
                    cout << result.count() << " нс";
                    cout << "\n\n";
                    cout << "\n" << "Отсортированный массив:\n";
                    for (int i = 0; i < N; ++i) {
                        cout << B[i] << " ";
                    }
                    cout << "\n\n";
                    menu();
                    break;
                case 3:
                    start = steady_clock::now();
                    comb_sort(C, N);
                    end = steady_clock::now();
                    result = duration_cast<nanoseconds>(end - start);
                    cout << "\n";
                    cout << "Время сортировки:  ";
                    cout << result.count() << " нс";
                    cout << "\n\n";
                    cout << "\n" << "Отсортированный массив:\n";
                    for (int i = 0; i < N; i++)
                    {
```

```
                    cout << C[i] << " ";
                }
                cout << "\n\n";
                menu();
                break;
            case 4:
                start = steady_clock::now();
                insert_sort(D, N);
                end = steady_clock::now();
                result = duration_cast<nanoseconds>(end - start);
                cout << "\n";
                cout << "Время сортировки:  ";
                cout << result.count() << " нс";
                cout << "\n\n";
                cout << "\n" << "Отсортированный массив:\n";
                for (int i = 0; i < N; ++i) {
                    cout << D[i] << " ";
                }
                cout << "\n\n";
                menu();
                break;
            case 5:
                start = steady_clock::now();
                quicksort(E, ends, begin);
                end = steady_clock::now();
                result = duration_cast<nanoseconds>(end - start);
                cout << "\n";
                cout << "Время сортировки:  ";
                cout << result.count() << " нс";
                cout << "\n";
                cout << "\n" << "Отсортированный массив:\n";
                for (int i = 0; i < N; i++)
                    cout << E[i] << " ";
                cout << "\n\n";

                menu();
                break;
            case 0:
                cout << "\n";
                menu();
                break;
            default:
                cout << "Введено неправильное число";
                cout << "\n\n";
                menu();
        }
        break;
    case 2:
        quicksort(I, ends, begin);
        start = steady_clock::now();
        search_elem_sort(I);
        end = steady_clock::now();
        result = duration_cast<nanoseconds>(end - start);
        cout << "\n\n";
        cout << "Время выполнения:  ";
        cout << result.count() << " нс";
        cout << "\n\n";

        start = steady_clock::now();
        search_elem(F, N);
        end = steady_clock::now();
        result = duration_cast<nanoseconds>(end - start);
```

```
                    cout << "\n\n";
                    cout << "Время выполнения:  ";
                    cout << result.count() << " нс";
                    cout << "\n\n";
                    menu();
                    break;
                case 3:
                    quicksort(I, ends, begin);
                    start = steady_clock::now();
                    average_value_sort(I, N);
                    end = steady_clock::now();
                    result = duration_cast<nanoseconds>(end - start);
                    cout << "\n\n";
                    cout << "Время выполнения:  ";
                    cout << result.count() << " нс";
                    cout << "\n\n";

                    start = steady_clock::now();
                    average_value(F, N);
                    end = steady_clock::now();
                    result = duration_cast<nanoseconds>(end - start);
                    cout << "\n\n";
                    cout << "Время выполнения:  ";
                    cout << result.count() << " нс";
                    cout << "\n\n";
                    menu();
                    break;
                case 4:
                    quicksort(I, ends, begin);
                    min_numbers(I, N);
                    cout << "\n\n";
                    menu();
                    break;
                case 5:
                    quicksort(I, ends, begin);
                    max_numbers(I, N);
                    cout << "\n\n";
                    menu();
                    break;
                case 6:
                    quicksort(I, ends, begin);
                    int value, res;
                    cout << "Введите число:  ";
                    cin >> value;
                    cout << "\n";
                    start = steady_clock::now();
                    res = binarySearch(I, value, 0, N - 1);
                    end = steady_clock::now();
                    result = duration_cast<nanoseconds>(end - start);
                    if (res == -1) {
                        cout << "Такого элемента нет\n\n";
                    }
                    else {
                        cout << "Такой элемент есть, его индекс: " << res <<
"\n\n";
                    }
                    cout << "Время выполнения:  ";
                    cout << result.count() << " нс";
                    cout << "\n\n";
                    menu();
                    break;
                case 7:
```

```
                quicksort(I, ends, begin);
                int value2;
                cout << "Введите число:  ";
                cin >> value2;
                cout << "\n";
                search_value(I, value2, N);
                cout << "\n\n";
                menu();
                break;
            case 8:
                quicksort(G, ends, begin);
                cout << "Исходный массив:\n";
                for (int i = 0; i < N; i++) {
                    cout << G[i] << " ";
                }
                cout << "\n\n";
                int ind1, ind2;
                cout << "Введите идексы элементов:\n";
                cin >> ind1 >> ind2;
                cout << "\n";
                swapElem(G, ind1, ind2);
                cout << "Массив после изменений:\n";
                for (int i = 0; i < N; i++) {
                    cout << G[i] << " ";
                }
                cout << "\n\n";
                menu();
                break;
            case 9:
                cout << "Исходный массив:\n";
                for (int i = 0; i < N; i++) {
                    cout << H[i] << " ";
                }
                cout << "\n\n";
                element_reduction(H, N);
                cout << "Результаты уменьшения:\n";
                for (int i = 0; i < N; i++) {
                    cout << H[i] << " ";
                }
                cout << "\n\n";
                element_multiplication(H, N);
                cout << "\n\n";
                cout << "Результаты умножения:\n";
                for (int i = 0; i < N; i++) {
                    cout << H[i] << " ";
                }
                cout << "\n\n";
                sum_elem(H, N);
                menu();
                break;
            default:
                cout << "Введено неправильное число";
                cout << "\n\n";
                menu();
        }
        cin >> i2;
        cout << "\n";
    }
    return 0;
}
```