**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе № 5 на тему:   
**«Методы поиска элемента в массиве.»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-312, Кандиков М.В.

«24» Сентября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/М.В. Кандиков /

Принял: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«24» Сентября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ А.М. Белов /

**Содержание**

**1. Титульный лист 1**

**2. Содержание 2**

**3. Основная часть 3**

3.1. Цель работы 3

3.2. Теоретическая часть 3

3.3. Практическая часть 4

3.4. Результаты выполнения программы 6

3.5. Выводы 7

**4. Приложение 9**

**Основная часть**

**Цель работы:**

Целью данной лабораторной работы является изучение и реализация алгоритмов поиска элементов в массиве с использованием методов линейного и бинарного поиска. Необходимо оценить их трудоемкость, провести сравнение по количеству операций, требуемых для нахождения элемента. Также требуется разработать алгоритм сортировки для подготовки массива к бинарному поиску. Исходные данные считываются из файла, а результат работы программы записывается в новый файл. Работа включает также анализ и описание полученных результатов, вывод о целесообразности использования того или иного метода поиска в различных условиях.

**Теоретическая часть:**

Линейный поиск — это один из простейших методов поиска элемента в массиве. Поиск начинается с первого элемента массива и продолжается до тех пор, пока не будет найден искомый элемент или не будут проверены все элементы массива. Если элемент найден, программа возвращает его индекс, иначе сообщает о неудачном поиске.

Основные свойства линейного поиска:

1. Подходит для неотсортированных массивов.
2. Трудоемкость в худшем случае составляет O(n)O(n)O(n), где nnn — количество элементов в массиве.
3. Преимущество линейного поиска заключается в его простоте, однако его недостаток — низкая производительность на больших объемах данных, так как он требует проверки каждого элемента.

Бинарный поиск — это более эффективный метод, который работает только на отсортированных массивах. Алгоритм бинарного поиска делит массив пополам на каждом этапе, что позволяет значительно сократить количество проверок.

Этапы бинарного поиска:

1. Выбор элемента, находящегося в середине массива.
2. Сравнение этого элемента с искомым значением.
3. Если искомый элемент меньше середины, поиск продолжается в левой части массива, если больше — в правой.
4. Процесс повторяется до тех пор, пока не будет найден элемент или не останется элементов для поиска.

Основные свойства бинарного поиска:

1. Подходит только для отсортированных массивов.
2. Трудоемкость алгоритма в худшем случае составляет O(logn), что значительно быстрее по сравнению с линейным поиском при больших массивах.
3. Требует предварительной сортировки массива, что может добавить дополнительных затрат на вычисления.

Для того чтобы применить бинарный поиск, необходимо предварительно отсортировать массив. В данной работе для сортировки был использован алгоритм сортировки пузырьком. Суть алгоритма заключается в многократном прохождении по массиву и сравнении соседних элементов с их последующей перестановкой, если они расположены не по порядку.

**Практическая часть:**

Для решения задачи по поиску элементов в массиве с использованием методов линейного и бинарного поиска были выполнены несколько этапов. Сначала исходный массив чисел был считан из текстового файла inp.txt, который содержал 30 положительных, отрицательных и нулевых элементов. Затем был реализован алгоритм линейного поиска, перебирающий элементы массива. При нахождении искомого элемента программа выводила его индекс и количество сравнений, необходимых для поиска. Если элемент не находился, пользователю сообщалось об этом.

Далее, для выполнения бинарного поиска массив был отсортирован с помощью алгоритма сортировки пузырьком, который многократно сравнивал соседние элементы и переставлял их, если они находились в неправильном порядке. На уже отсортированном массиве был применён бинарный поиск, который делил массив пополам на каждом шаге, что позволяло значительно сократить количество сравнений.

В конце результаты поиска, включая отсортированный массив, были записаны в новый файл out.txt, а также выведены на экран, что включало информацию о найденных элементах, их индексах и количестве сравнений для каждого метода. Эти шаги позволили провести полное исследование алгоритмов поиска и оценить их производительность.

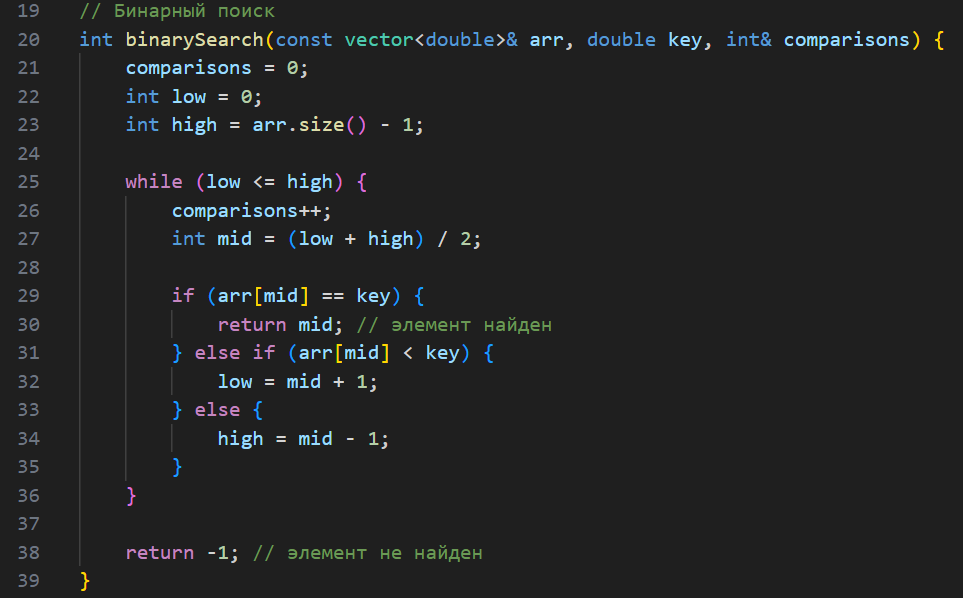


Рисунок 1. Код для бинарного поиска.

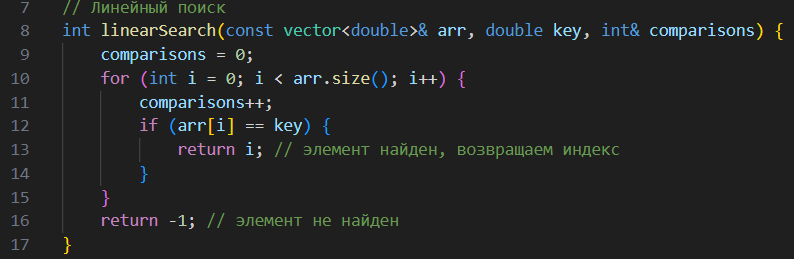


Рисунок 2. Код для линейного поиска.

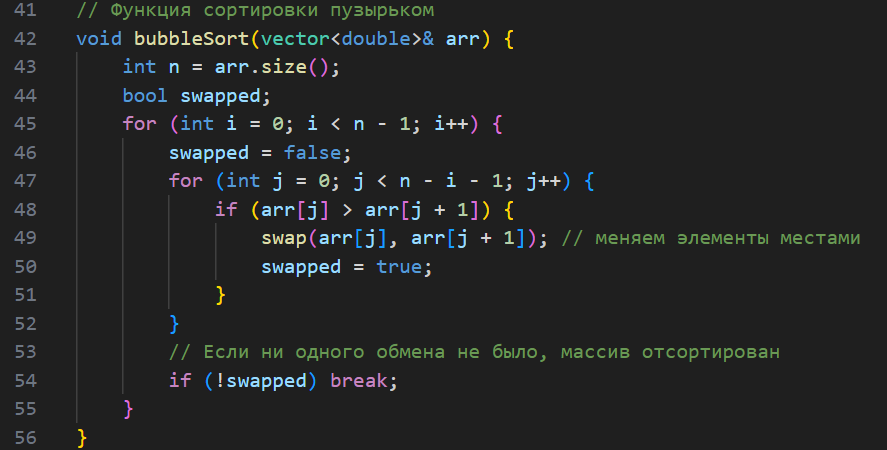


Рисунок 3. Код для пузырьковой сортировки.

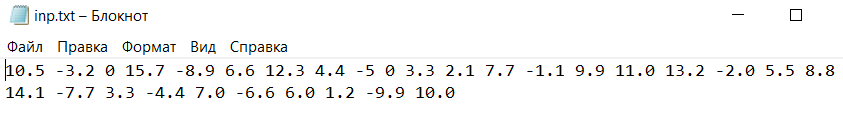


Рисунок 4. Файл inp.txt соддержащий числа для массива.

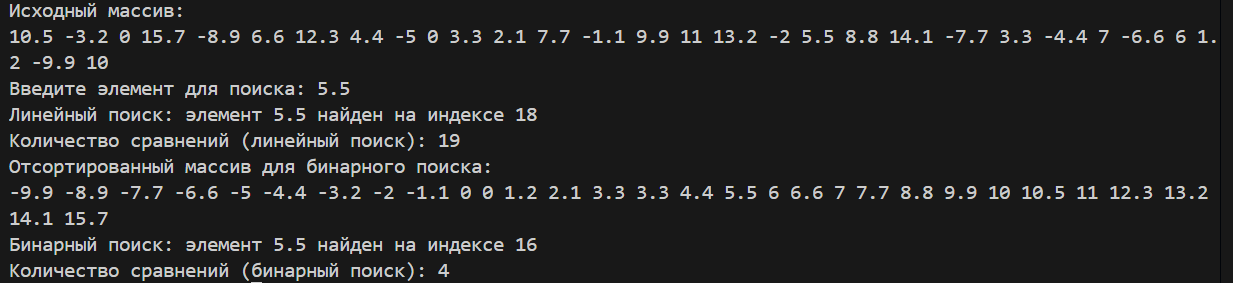
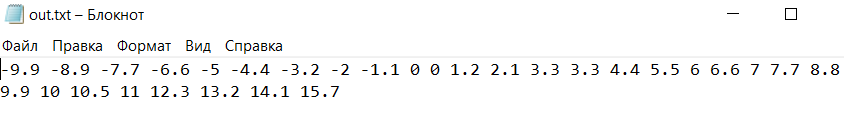


Рисунок 5. Результат выполненной работы в терминале.

Рисунок 6. Отсортированный массив, сохранённый в файл out.txt после выполнения задачи.

**Выводы:**

Линейный поиск является простым методом поиска, который подходит для небольших массивов или случаев, когда данные не отсортированы. Однако его трудоемкость возрастает линейно с увеличением размера массива, что делает его неэффективным для больших объемов данных. В худшем случае, при поиске последнего элемента или при отсутствии элемента в массиве, количество сравнений будет равно числу элементов массива O(n). Такой подход можно использовать только тогда, когда нет требований к высокой производительности, например, в случае небольших массивов или одноразового поиска.

Бинарный поиск является более эффективным методом, так как его трудоемкость растет логарифмически O(log n) с увеличением размера массива. Это делает его значительно более быстрым по сравнению с линейным поиском при больших объемах данных. Однако бинарный поиск требует, чтобы массив был отсортирован. В случаях, когда необходимо выполнить несколько операций поиска на одном и том же наборе данных, предварительная сортировка оправдана, так как она позволяет существенно ускорить последующие поисковые операции. В то же время для одноразовых операций поиск может оказаться менее выгодным из-за затрат на сортировку.

Алгоритм сортировки пузырьком был выбран для данной задачи из-за его простоты и наглядности. Однако он является достаточно медленным при больших объемах данных, так как его трудоемкость составляет O(n^2). В реальных приложениях на больших массивах предпочтительнее использовать более быстрые алгоритмы сортировки, такие как быстрая сортировка или сортировка слиянием.

Из результатов практической работы видно, что бинарный поиск значительно превосходит линейный по скорости на больших объемах данных, однако требует предварительной сортировки. Линейный поиск может быть использован в случаях, когда массив небольшого размера или поиск выполняется на неотсортированных данных, и затраты на сортировку не оправданы.  
Для больших данных предпочтителен бинарный поиск с предварительной сортировкой массива. Линейный поиск стоит применять в простых случаях, когда не требуется высокая производительность и массив небольшой.

**Приложение**

Листинг программы:

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

// Линейный поиск

int linearSearch(const vector<double>& arr, double key, int& comparisons) {

    comparisons = 0;

    for (int i = 0; i < arr.size(); i++) {

        comparisons++;

        if (arr[i] == key) {

            return i; // элемент найден, возвращаем индекс

        }

    }

    return -1; // элемент не найден

}

// Бинарный поиск

int binarySearch(const vector<double>& arr, double key, int& comparisons) {

    comparisons = 0;

    int low = 0;

    int high = arr.size() - 1;

    while (low <= high) {

        comparisons++;

        int mid = (low + high) / 2;

        if (arr[mid] == key) {

            return mid; // элемент найден

        } else if (arr[mid] < key) {

            low = mid + 1;

        } else {

            high = mid - 1;

        }

    }

    return -1; // элемент не найден

}

// Функция сортировки пузырьком

void bubbleSort(vector<double>& arr) {

    int n = arr.size();

    bool swapped;

    for (int i = 0; i < n - 1; i++) {

        swapped = false;

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

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

                swap(arr[j], arr[j + 1]); // меняем элементы местами

                swapped = true;

            }

        }

        // Если ни одного обмена не было, массив отсортирован

        if (!swapped) break;

    }

}

// Функция для вывода массива

void printArray(const vector<double>& arr) {

    for (double elem : arr) {

        cout << elem << " ";

    }

    cout << endl;

}

int main() {

    // Ввод и вывод файлов

    ifstream inp("inp.txt");

    ofstream out("out.txt");

    if (!inp) {

        cerr << "Не удалось открыть входной файл!" << endl;

        return 1;

    }

    if (!out) {

        cerr << "Не удалось открыть выходной файл!" << endl;

        return 1;

    }

    vector<double> arr;

    double value;

    // Считывание массива из файла

    while (inp >> value) {

        arr.push\_back(value);

    }

    inp.close();

    // Вывод исходного массива

    cout << "Исходный массив:" << endl;

    printArray(arr);

    // Пользователь вводит искомое значение

    double key;

    cout << "Введите элемент для поиска: ";

    cin >> key;

    // Линейный поиск

    int linearComparisons = 0;

    int linearIndex = linearSearch(arr, key, linearComparisons);

    if (linearIndex != -1) {

        cout << "Линейный поиск: элемент " << arr[linearIndex] << " найден на индексе " << linearIndex << endl;

    } else {

        cout << "Линейный поиск: элемент не найден" << endl;

    }

    cout << "Количество сравнений (линейный поиск): " << linearComparisons << endl;

    // Сортировка пузырьком для бинарного поиска

    bubbleSort(arr);

    // Вывод отсортированного массива для бинарного поиска

    cout << "Отсортированный массив для бинарного поиска:" << endl;

    printArray(arr);

    // Бинарный поиск

    int binaryComparisons = 0;

    int binaryIndex = binarySearch(arr, key, binaryComparisons);

    if (binaryIndex != -1) {

        cout << "Бинарный поиск: элемент " << arr[binaryIndex] << " найден на индексе " << binaryIndex << endl;

    } else {

        cout << "Бинарный поиск: элемент не найден" << endl;

    }

    cout << "Количество сравнений (бинарный поиск): " << binaryComparisons << endl;

    // Запись отсортированного массива в файл

    for (double elem : arr) {

        out << elem << " ";

    }

    out.close();

    return 0;

}