Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 3 по курсу "Анализ алгоритмов"

Тема Поиск в массиве

Студент Шавиш Тарек

Группа ИУ7И-54Б

Оценка (баллы)

Преподаватели Волкова Л. Л., Строганов Ю. В.

Москва — 2025 г.

Содержание

ВВЕДЕНИЕ

Целью данной лабораторной работы является разработка и анализ алгоритмов поиска элемента в массиве: методом полного перебора и бинарным поиском. Алгоритм полного перебора заключается в последовательном сравнении искомого элемента с каждым элементом массива до нахождения совпадения или окончания массива. Бинарный поисктребует предварительной сортировки массива и работает путём деления диапазона поиска на половины пока элемент не будет найден.

1 Аналитический раздел

1.1 Цель и задачи

Цель - исследование методов поиска элемента в массиве и их реализация для анализа эффективности.

Требуется решить следующие задачи.

1) Разработка и реализация следующих алгоритмов:

   — алгоритм поиска полным перебором;

   — алгоритм поиска бинарным поиском.

2) Тестирование и анализ алгоритмов. Произвести тестирование реализованных алгоритмов.

1.2 Алгоритм поиска полным перебором

Алгоритм поиска полным перебором проверяет каждый элемент массива последовательно до тех пор, пока не будет найден искомый элемент или до окончания массива. Его сложность составляет $O(N)$, где N - количество элементов в массиве.

1.3 Алгоритм поиска бинарным поиском

Алгоритм поиска элемента бинарным поиском работает на принципе деления массива пополам. Этот метод требует, чтобы массив был отсортирован. Если элемент в середине проверяемого диапазона совпадает с искомым, поиск завершается. Если элемент в середине меньше искомого, поиск продолжается в правой половине массива, иначе — в левой. Это уменьшает количество сравнений в $\log_2(N)$ раз, где N — количество элементов. Таким образом, время выполнения бинарного поиска составляет $O(\log N)$.

1.4 Вывод

Были описаны теоретические основы исследуемых алгоритмов. Алгоритм полного перебора, показывает линейную зависимость времени выполнения от размера массива, что делает его неэффективным для больших данных. Алгоритм бинарного поиска, требующий предварительной сортировки массива, увеличивает скорость поиска за счет логарифмической сложности.

2 Конструкторский раздел

В данной главе представлены основные конструкторские решения, использованные в процессе разработки программного обеспечения. Раздел включает требования к программному обеспечению, определение типов данных, и структуру проекта.

2.1 Требования к реализации программного обеспечения

Для успешной реализации и функционирования разработанного программного обеспечения были установлены следующие требования:

1) ввод исходного массива и задание искомого элемента;
2) возможность выбора алгоритма для поиска выбранного элемента;
3) поддержка различных режимов работы программы, управляемых через интерфейс командной строки;
4) возможность вывода результатов тестирования.

2.2 Описание алгоритмов

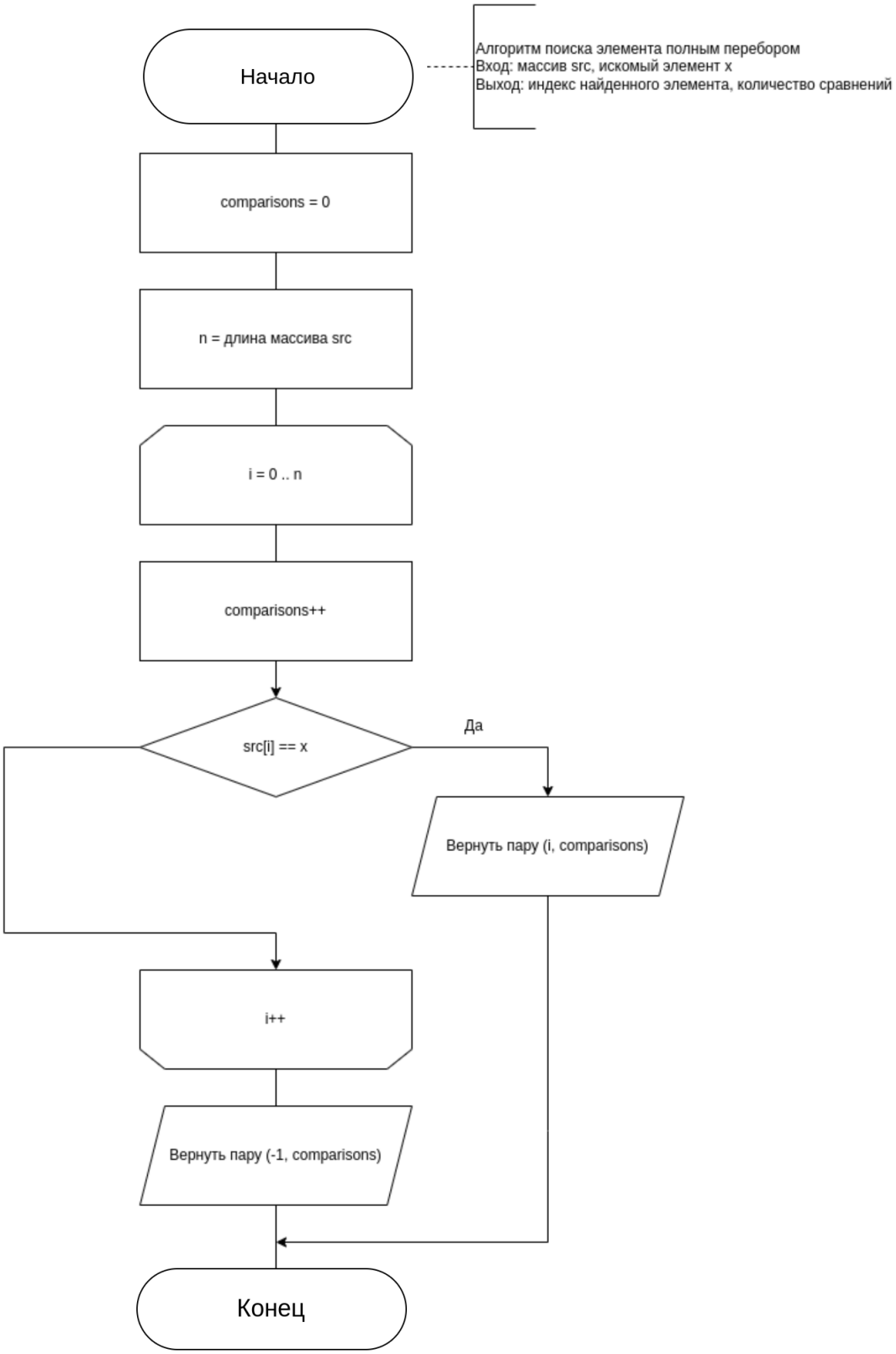

Рисунок 2.1 – Алгоритм полного перебора

Рисунок 2.2 – Алгоритм бинарного поиска

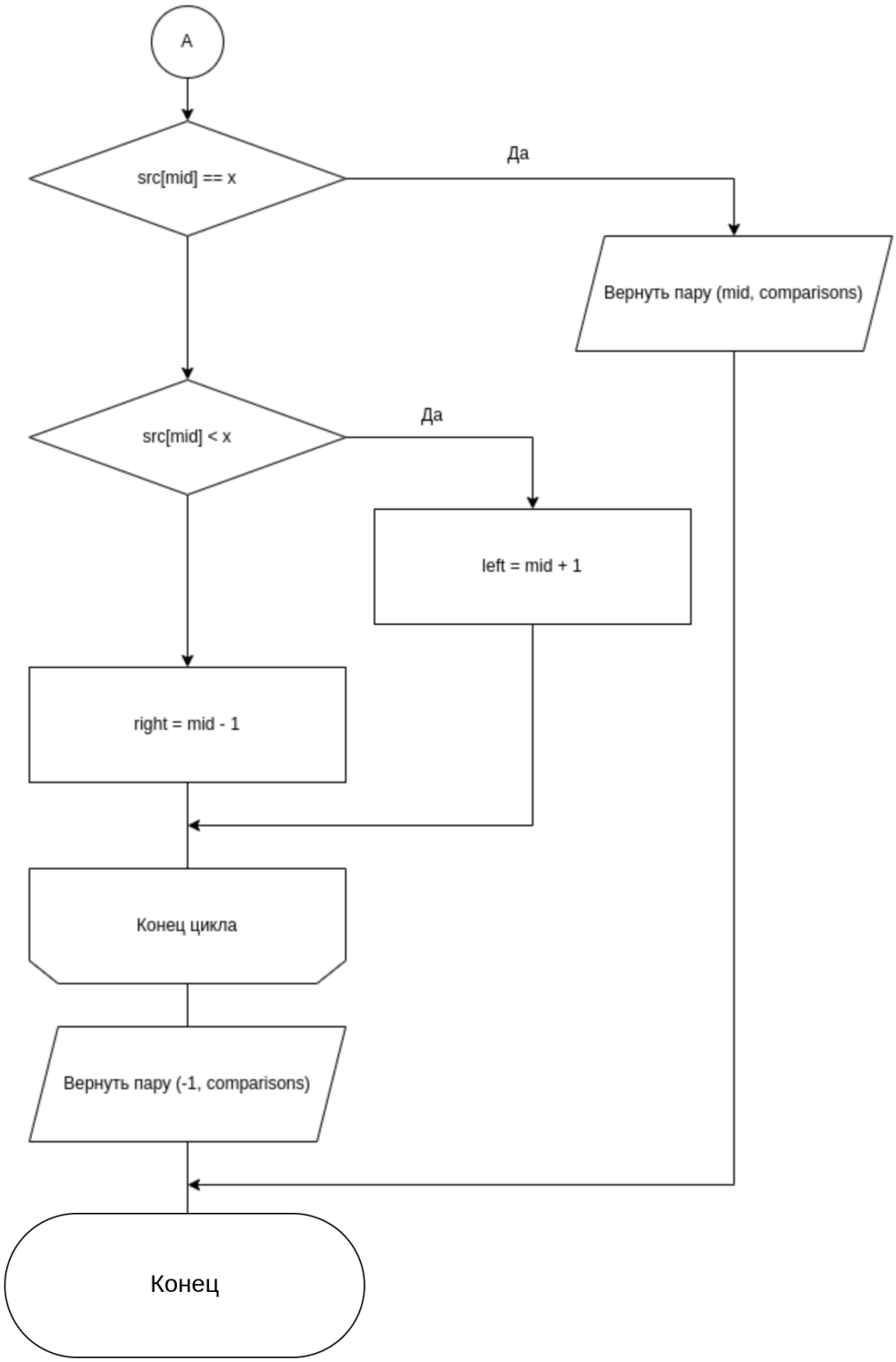

Рисунок 2.3 – Алгоритм бинарного поиска (продолжение)

2.3 Описание типов данных и классов

Выделены следующие классы:

1) `Finder` — сущность, содержащая алгоримты поиска расстояний;

2) `TaskHandler` — сущность, содержащая логику управление задачами и режимом выполнения программы;

3) `EfficiencyAnalyzer` — сущность, содержащая логику анализа эффективности алгоритмов.

Используются следующие типы данных: ADD ENUMERATE

1) `целочисленный тип` — для описания элементов массива в котором проиводится поиск элемента;

2) `одномерный целочисленный массив` — для хранения набора элементов.

2.4 Структура проекта

Программа разделена на заголовочные файлы (.h), исходные файлы (.cpp) и файл сборки (Makefile). Ниже представлена структура каталогов и файлов проекта

Каталог **include** — каталог для заголовочных файлов:

1) `auxiliar.h` — декларации вспомогательных функций;

2) `finder.h` — интерфейс класса *Finder*;

3) `taskhandler.h` — интерфейс класса *TaskHandler*;

4) `efficiencyanalyzer.h` — интерфейс класса *EfficiencyAnalyzer*.

Каталог **source** - каталог с исходным кодом и реализацией методов классов:

1) `main.cpp` — главный файл программы, сожержащего точку входа программы;

2) `auxiliar.cpp` — реализация вспомогательных функций;

3) `finder.cpp` — реализация методов класса *Finder*;

4) `taskhandler.cpp` — реализация методов класса *TaskHandler*;

5) `efficiencyanalyzer.cpp` — реализация методов класса *EfficiencyAnalyzer*.

Файл **makefile** — Файл для сборки проекта с использованием утилиты make.
Файл **plot.py** — Логика получения аналитических графиков времени выполнения алгоритмов.

Вывод

Описаны схемы алгоритмов, выделенные сущности, используемые типы данных и структура проекта.

3 Технологический раздел

В данном разделе описывается выбор инструментов для реализации программы, включая язык программирования. Также представлены реализации ключевых алгоритмов исследования, описания методов тестирования программы и анализ полученных результатов.

3.1 Выбор языка программирования

Для реализации алгоритмов вычисления редакционного расстояния был выбран язык программирования C++. Выбор обусловлен следующими факторами:

— производительность;

— объектно-ориентированный подход;

— наличие стандартной библиотеки шаблонов.

3.2 Реализация алгоритмов

Листинг 3.1 – Реализация алгоритма поиска полным перебором

```
1  std::pair<int, int> Finder::bruteForce(const std::vector<int> &src,
       int x)
2  {
3      int comparisions = 0;
4      for (unsigned long int i = 0; i < src.size(); i++)
5      {
6          ++comparisions;
7          if (src[i] == x)
8          {
9              return {i, comparisions};
10         }
11     }
12     return {-1, comparisions};
13 }
```

Листинг 3.2 – Реализация алгоритма бинарного поиска

```
1  std::pair<int, int> Finder::binarySearch(const std::vector<int>
       &src, int x)
2  {
3      int left = 0;
4      int right = src.size() - 1;
5      int comparisions = 0;
6
7      while (left <= right)
8      {
9          int mid = left + (right - left) / 2;
10         ++comparisions;
11
12         if (src[mid] == x)
13         {
14             return {mid, comparisions};
15         }
16         else if (src[mid] < x)
17         {
18             left = mid + 1;
19         }
20         else
21         {
22             right = mid - 1;
23         }
24     }
25     return {-1, comparisions};
26 }
```

3.3 Тестирование

Тестирование реализаций алгоритмов проводилось на основе следующих классов эквивалентности:

1) искомого элемента нет в массиве;

2) искомый элемент находится на первой позиции массива;

3) искомый элемент находится в середине массива;

4) искомый элемент находится на последней позиции массива;

5) искомый элемент находится в произвольной позиции массива.

Результаты тестирования представлены в таблице:

Таблица 3.1 – Результаты тестирования алгоритмов вычисления расстояний

№	**Входные данные**		**Результат**	
	Массив	**Элемент**	**Полный перебор**	**Бин. поиск**
1	[1, 3, 5, 7]	9	-1	-1
2	[2, 4, 6, 8]	2	0	0
3	[11, 22, 33, 44, 55]	33	2	2
4	[9, 8, 7, 6, 5]	5	4	4
5	[1, 4, 8, 2, 3]	2	3	3

Вывод

В результате технологического анализа были выбраны инструменты и методы, наиболее подходящие для реализации задачи вычисления редакционных расстояний и были выделены класси эквивалентности для тестирования алгоритмов. Все тесты были пройдены.

4 Исследовательский раздел

В данном разделе представлены результаты исследования по анализу эффективности алгоритмов изучаемых алгоритмов поиска элемента в массиве. Также приводятся пример работы программы и описываются технические характеристики устройства, на котором проводились тесты.

4.1 Примеры работы программы

Ниже приведен пример работы программы:

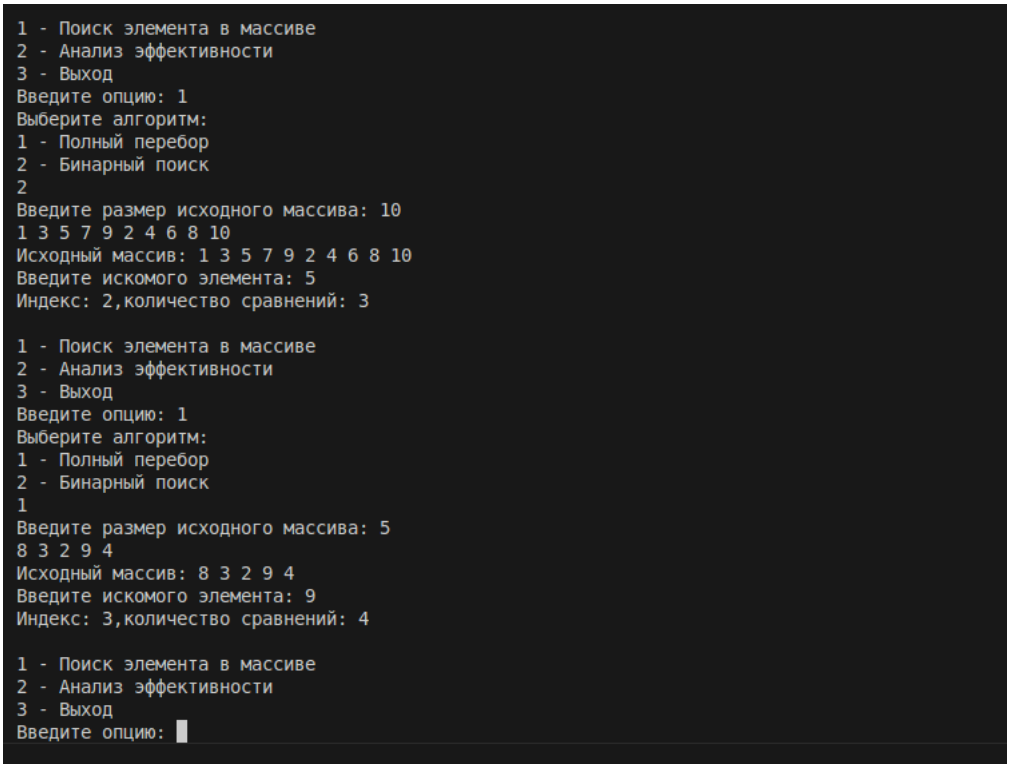

Рисунок 4.1 – Пример работы программы

4.2 Исследование работы алгоритмов

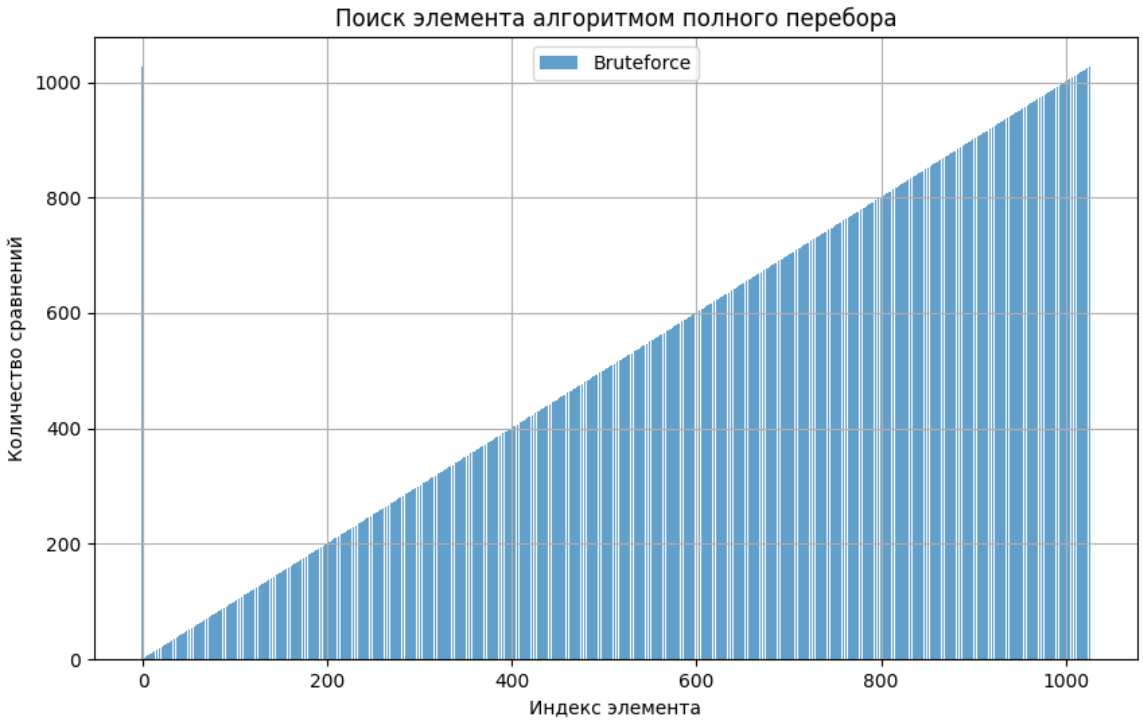

Рисунок 4.2 – Зависимость количества сравнений от индекс искомого элемента при использовании алгоритма полного перебора

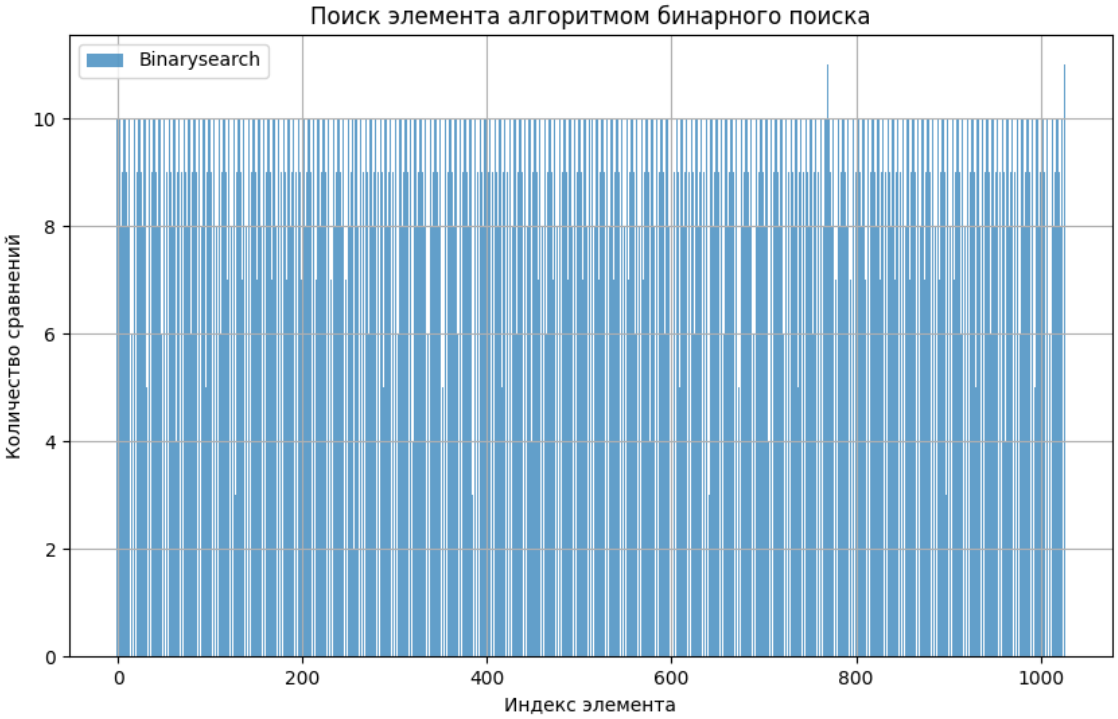

Рисунок 4.3 – Зависимость количества сравнений от индекс искомого элемента при использовании алгоритма бинарного поиска

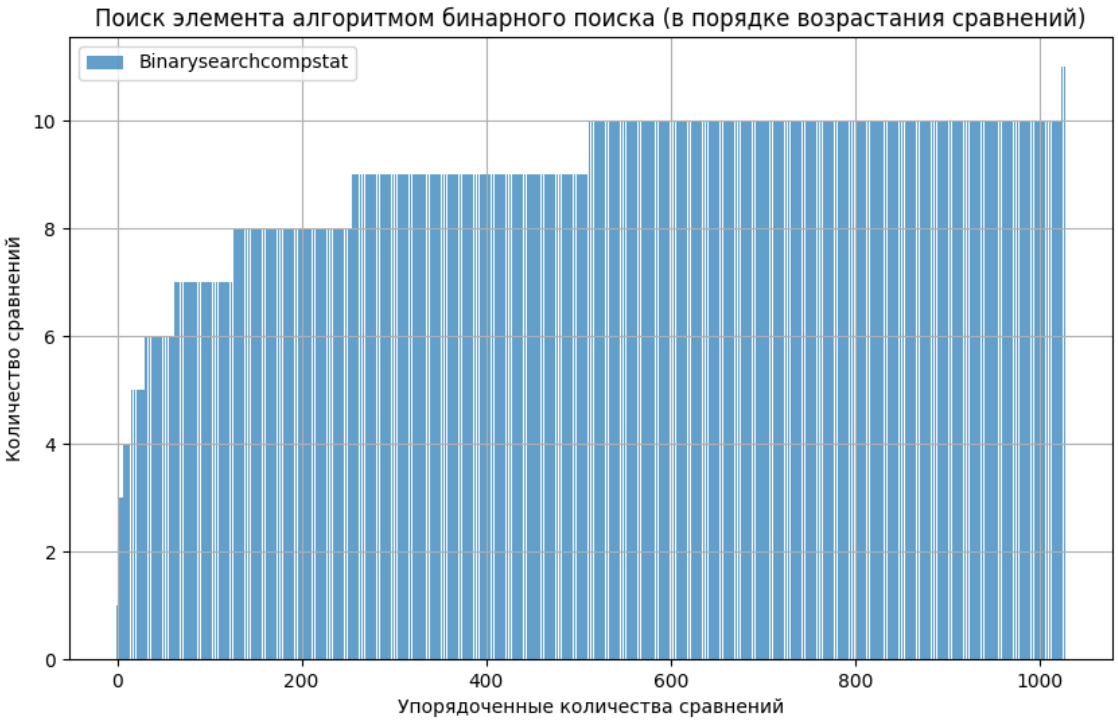

Рисунок 4.4 – Зависимость количества сравнений от индекс искомого элемента при использовании алгоритма бинарного поиска (количество сравнений упорядочены)

4.3 Технические характеристики устройства

Настройки устройства, на котором проводились тесты, разработка программы и анализ работы алгоритмов следующие:

1) операционная система — Ubuntu 22.04 LTS;
2) процессор — Intel(R) Core(TM) i5-1035G1 CPU @ 1.00 ГГц;
3) оперативная память — 16 Гб.

4.4 Вывод

Алгоритм бинарного поиска значительно эффективнее алгоритма полного перебора в терминах количества сравнений, необходимых для нахождения элемента. Полный перебор, имея линейную сложность, становится неэффективным при увеличении размера данных. В свою очередь, бинарный поиск демонстрирует логарифмическую сложность.

ЗАКЛЮЧЕНИЕ

Цель достигнута. Следующие задачи были решены:

1) были разработаны алгоритмы поиска элемента в массиве полным перебором и бинарным поиском;

2) реализации алгоритмов были протестированы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бинарный приск [Электронный ресурс]. URL: https://ocw.mit.edu/courses/6-00-introduction-to-computer-science-and-programming-fall-2008/resources/lecture-9/ (дата обращения: 16.10.2024)

2. C/C++ Development User Guide [Электронный ресурс]. URL: https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Fconcepts%2Fcdt_o_home.html (дата обращения: 16.10.2024)

3. Официальная документация дистрибутива Linux - Ubuntu [Электронный ресурс]. URL: https://assets.ubuntu.com/v1/544d9904-ubuntu-server-guide-2024-01-22.pdf? (дата обращения: 16.10.2024)

4. Официальная документация библиотеки Matplotlib [Электронный ресурс]. - URL: https://matplotlib.org/stable/index.html (дата обращения: 16.10.2024)

5. Справочник ЯП C++ [Электронный ресурс]. URL: https://learn.microsoft.com/en-us/cpp/?view=msvc-170 (дата обращения: 16.10.2024)