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 №7 по дисциплине «Анализ алгоритмов»

Тема Алгоритмы поиска

Студент Шматко К. М.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

В данной лабораторной работе будут рассмотрены алгоритмы поиска.

Поиск необходимой информации в списке — одна из фундаментальных задач теоретического программирования [1]. Информация содержится в записях, составляющих некоторый список, который представляет собой массив данных в программе. Записи, или элементы списка, идут в массиве последовательно и между ними нет промежутков. Списки могут быть неотсортированными или отсортированными по значению ключевого поля. С поиском конкретного значения связана задача выборки, в которой требуется найти элемент, удовлетворяющий некоторым условиям.

Целью данной лабораторной работы является исследование алгоритмов поиска.

Для достижения поставленной цели необходимо выполнить следующие задачи.

1) Описать алгоритмы поиска.
2) Создать программное обеспечение, реализующее следующие алгоритмы поиска:
   - метод полного перебора;
   - бинарный поиск.
3) Замерить время реализации.
4) Провести анализ затрат работы программы по времени, выяснить влияющие на них характеристики.

1 Аналитическая часть

В этом разделе будет представлена информация об алгоритмах поиска.

1.1 Алгоритм полного перебора

Алгоритм последовательного поиска последовательно просматривает по одному элементу списка, начиная с первого, до тех пор, пока не найдет целевой элемент [1]. Очевидно, что чем дальше в списке находится конкретное значение ключа, тем больше времени уйдет на его поиск.

Следовательно, в лучшем случае искомый элемент будет на первом месте в списке, и сложность будет равняться $O(1)$. В худшем случае искомый элемент будет на последнем месте или отсутствовать в списке, и сложность станет $O(N)$.

1.2 Бинарный поиск

Бинарный поиск работает при отсортированном списке. При сравнении целевого значения со средним элементом отсортированного списка возможен один из трех результатов: значения равны, целевое значение меньше элемента списка, либо целевое значение больше элемента списка [1]. В первом, и наилучшем, случае поиск завершен. В остальных двух случаях мы можем отбросить половину списка. Когда целевое значение меньше среднего элемента, мы знаем, что если оно имеется в списке, то находится перед этим средним элементом. Когда же оно больше среднего элемента, мы знаем, что если оно имеется в списке, то находится после этого среднего элемента. Этого достаточно, чтобы мы могли одним сравнением отбросить половину списка. При повторении этой процедуры мы сможем отбросить половину оставшейся части списка.

Таким образом, в лучшем случае искомый элемент будет найден при первом проходе цикла, то есть в середине списка, а в худшем - на первом

или последнем месте или вовсе отсутствовать в списке. В обоих случаях сложность алгоритма составляет $O(log_2(N))$.

Вывод

В данном разделе были рассмотрены алгоритмы поиска — алгоритм полного перебора и бинарный поиск.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритма полного перебора и алгоритма бинарного поиска.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора. На рисунке 2.2 представлена схема алгоритма бинарного поиска.

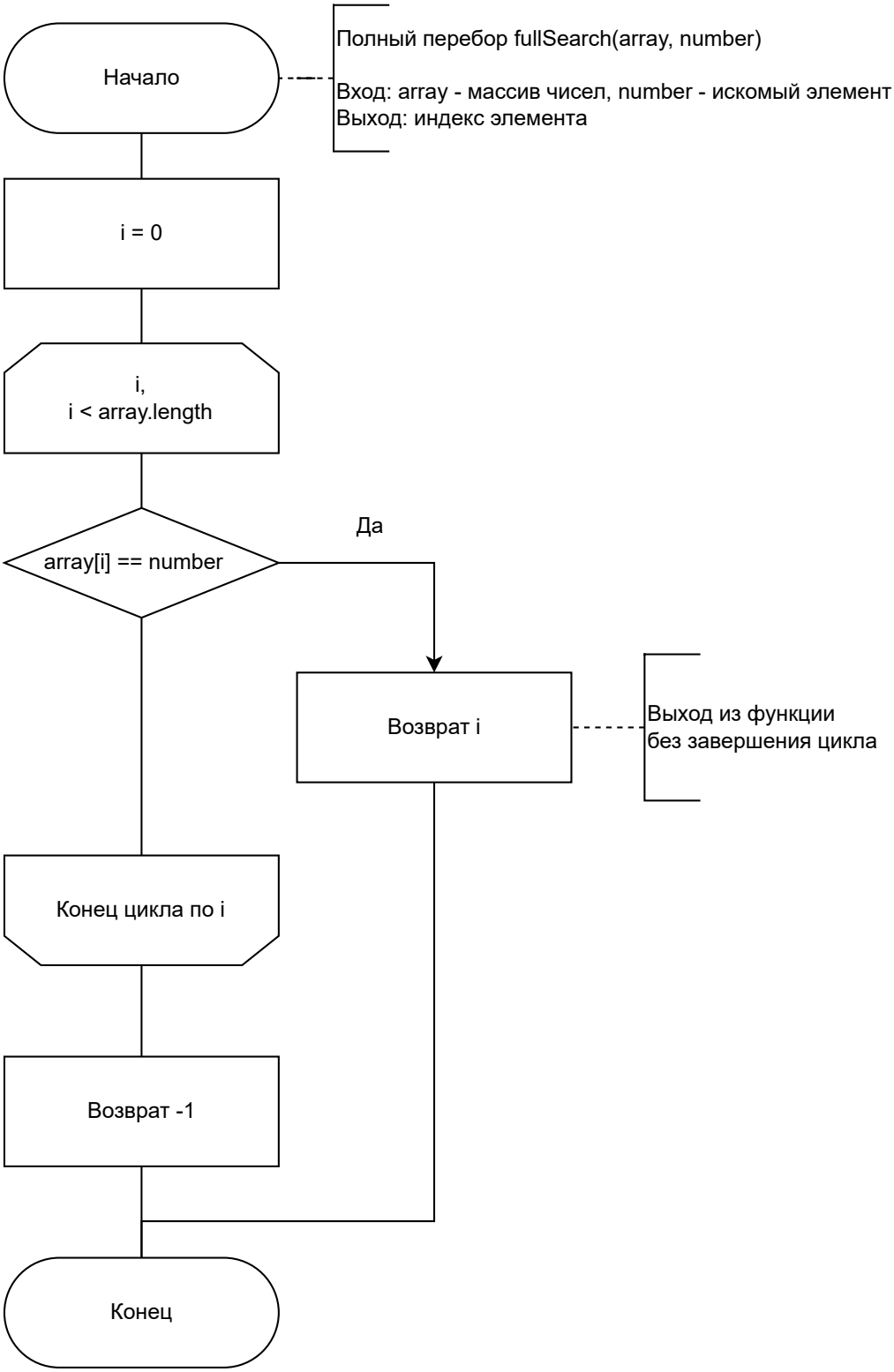

Рисунок 2.1 – Схема алгоритма полного перебора

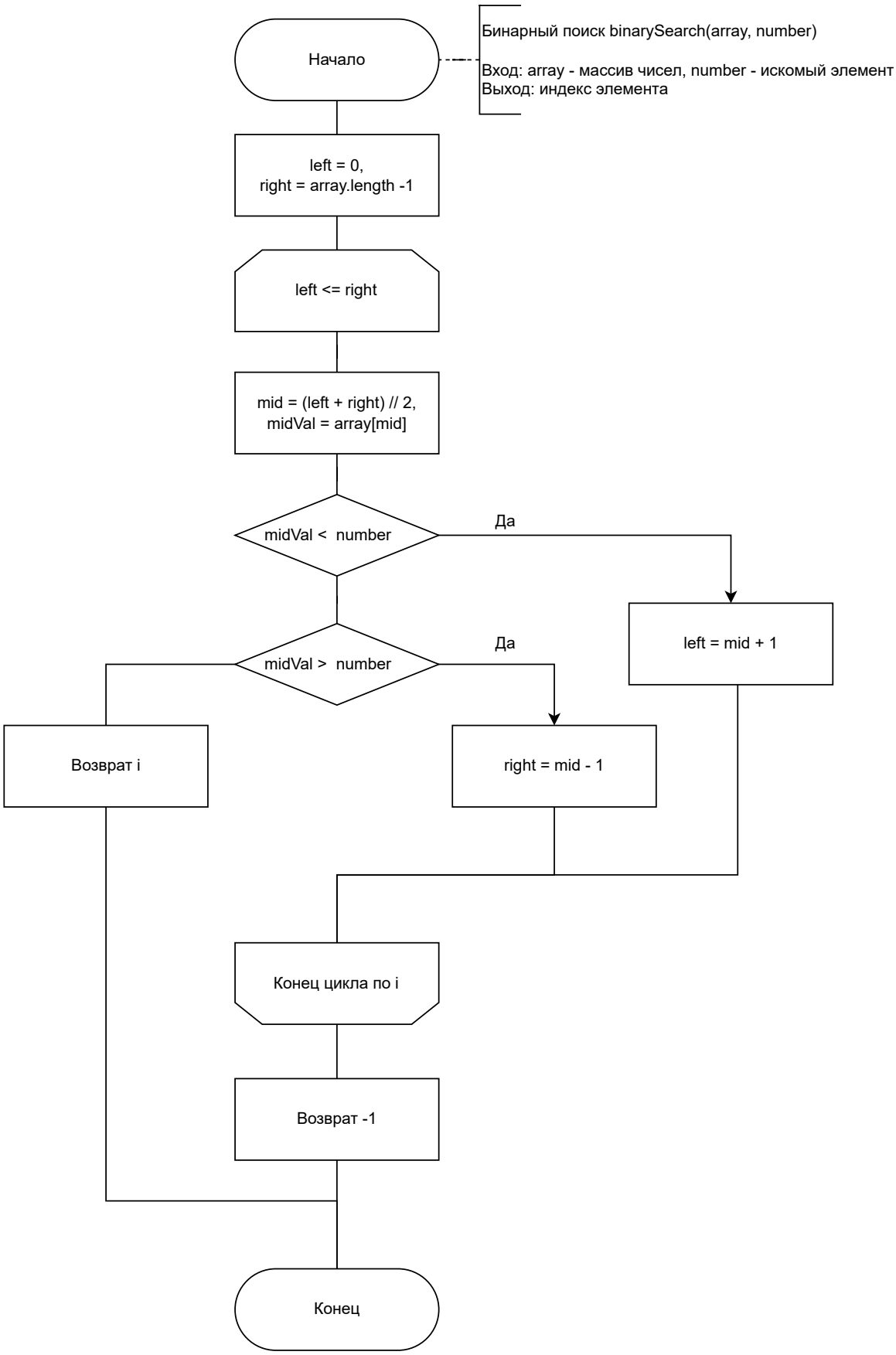

Рисунок 2.2 – Схема алгоритма бинарного поиска

Вывод

В данном разделе были рассмотрены алгоритм полного перебора и алгоритм бинарного поиска.

3 Технологическая часть

В данном разделе будут описаны требования к программному обеспечению, средства реализации, выбранные типы данных, листинг кода и функциональные тесты.

3.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— принимает на вход массив и искомый элемент;

— выдает индекс, являющийся результатом поиска;

— имеет интерфейс для выбора действий;

— имеет функциональность замера процессорного времени работы реализаций алгоритмов.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык JavaScript [2]. Данный выбор обусловлен наличием у языка встроенной функции $process.cpuUsage()$ для измерения процессорного времени. Это позволяет удовлетворить требованиям для выполнения лабораторной работы.

Время выполнения программы было замерено с использованием функции *process.cpuUsage()* из программной платформы Node.js [3].

3.3 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.js` — файл, содержащий точку входа в программу, из которой происходит запуск работы алгоритмов.

— `algorithms.js` — файл, содержащий функции реализации всех алгоритмов.

— `test.js` — файл содержит функции, замеряющие процессорное время алгоритмов.

3.4 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма полного перебора, а в листинге 3.2 представлена реализация алгоритма бинарного поиска.

Листинг 3.1 – Реализаиция алгоритма полного перебора

```
1 function fullSearch(array, number) {
2     for (let i = 0; i < array.length; i++) {
3         if(array[i] == number) {
4             return i;
5         }
6     }
7
8     return -1;
9 }
```

Листинг 3.2 – Реализаиция алгоритма бинарного поиска

```
function binarySearch(array, number) {
    let left = 0;
    let right = array.length -1;
    while(left <= right) {
        let mid = Math.floor((left + right)/2);
        let midVal = array[mid];
        if (midVal < number) {
            left = mid+1;
        }
        else if (midVal > number) {
            right = mid-1;
        }
        else {
            return mid;
        }
    }
    return -1;
}
```

3.5 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов поиска в виде: массив, искомый элемент и индекс элемента. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Массив и элемент	Полный перебор	Бинарный поиск
[1, 2, 3, 4, 5], 1	0	0
[10, 20, 30, 40, 50], 50	4	4
[3, 6, 9, 12, 15], 10	-1	-1

Вывод

Были реализованы алгоритмы поиска — бинарный поиск и алгоритм полного перебора. Проведено тестирование реализаций алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, проведение исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени.

— Процессор: 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц.

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 11 Домашняя 64-разрядная система версии 22H2 [4].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

Программа получает на вход массив и искомый элемент и выдает индекс элемента, найденный с помощью бинарного поиска и алгоритма полного перебора.

На рисунке 4.1 представлена демонстрация работы программы.

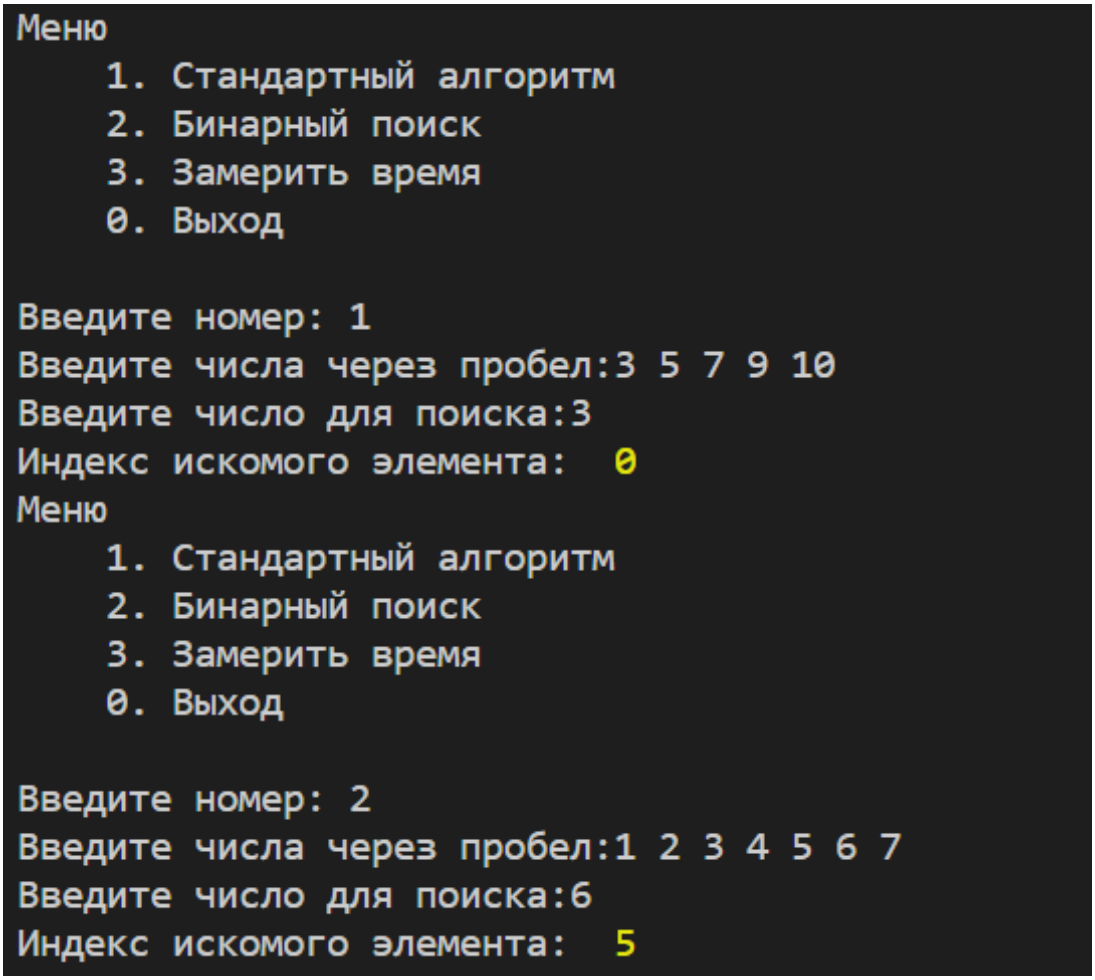

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблице 4.1. Использованы следующие обозначения:

— full_best — лучший случай полного перебора;

— full_bad — худший случай полного перебора;

— bin_best — лучший случай бинарного поиска;

— bin_bad — худший случай бинарного поиска.

Таблица 4.1 – Результаты замеров времени

Размер	**full_best, мс**	**full_bad, мс**	**bin_best, мс**	**bin_bad, мс**
256	1 098.871	7 411.957	1 246.929	1 259.804
1024	546.217	34 413.815	2 008.2	2 129.316
4096	625.134	182 800.531	2 201.08	2 331.495
16384	617.027	666 109.562	3 542.662	3 694.057
65536	903.845	3 083 545.923	4 140.854	3 781.557

По таблице 4.1 был построен график 4.2 для алгоритма полного перебора. Исходя из этих данных можно понять, что время выполнения растет линейно при худшем случае и остается константным при лучшем.

По таблице 4.1 был построен график 4.3 для бинарного поиска. Исходя из этих данных можно понять, что время выполнения растет логарифмически при лучшем и худшем случаях.

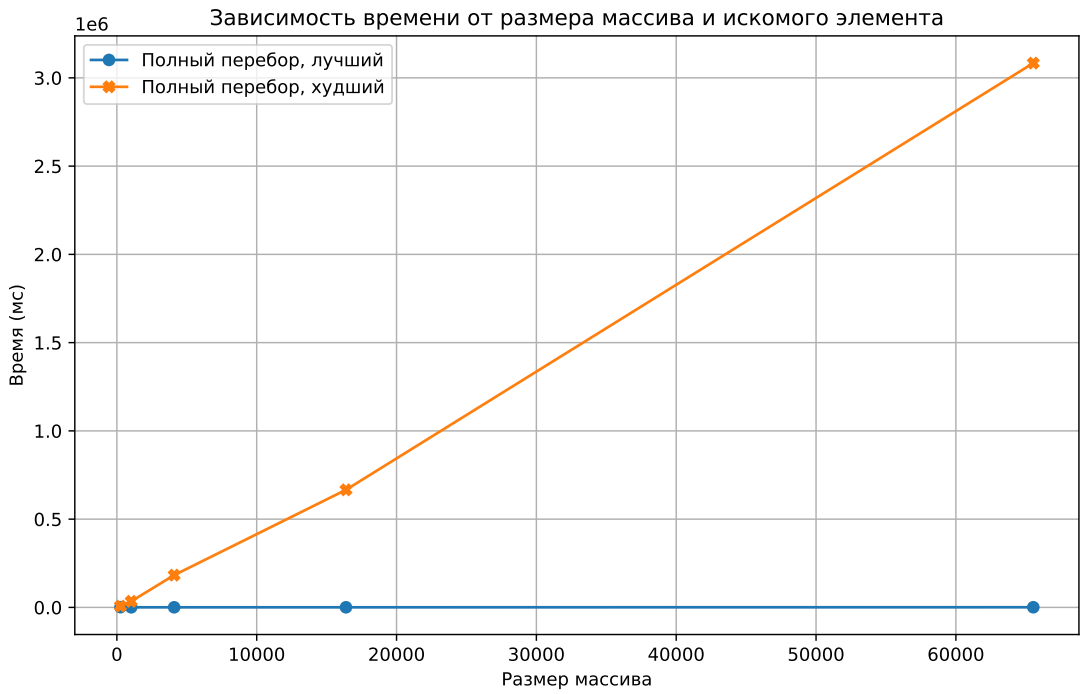

Рисунок 4.2 – Результаты замеров времени работы реализации алгоритма полного перебора

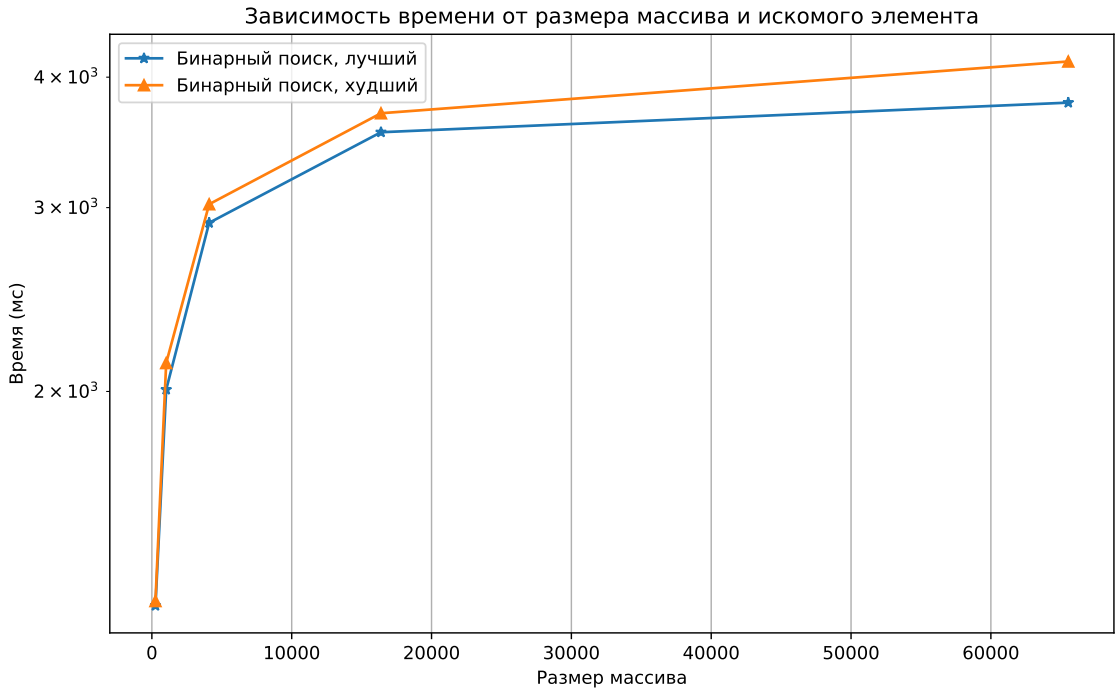

Рисунок 4.3 – Результаты замеров времени работы реализации алгоритма бинарного поиска

4.4 Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов.

Приведенные временные характеристики показывают, что алгоритм бинарного поиска в худшем работает быстрее алгоритма полного перебора независимо от размера массива. В лучшем же случае быстрее работает алгоритм полного перебора. Эти результаты объяснимы сложностью самих алгоритмов.

Заключение

Было экспериментально подтверждено, что алгоритм бинарного поиска в худшем работает быстрее алгоритма полного перебора независимо от размера массива. В лучшем же случае быстрее работает алгоритм полного перебора. Эти результаты объяснимы сложностью самих алгоритмов.

В ходе выполнения лабораторной работы были решены все задачи:

1) Описаны алгоритмы поиска.
2) Создано программное обеспечение, реализующее следующие алгоритмы поиска:
   - метод полного перебора;
   - бинарный поиск.
3) Замерено время реализации.
4) Проведен анализ затрат работы программы по времени, выяснены влияющие на них характеристики.

Список использованных источников

1. Макконнелл. Дж. Основы современных алгоритмов. — Москва: Техносфера, 2004. С. 53–64.

2. Документация по JavaScript [Электронный ресурс]. — Режим доступа: `https://262.ecma-international.org/13.0/` (дата обращения: 10.10.2023).

3. Node.js function process.cpuUsage([previousValue]) [Электронный ресурс]. — Режим доступа: `https://nodejs.org/api/process.html#processcpuusagepreviousvalue` (дата обращения: 10.10.2023).

4. Windows 11 [Электронный ресурс]. — Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows11` (дата обращения: 10.10.2023).