



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 курсу «Анализ алгоритмов»  
на тему: «Алгоритмы поиска»

Студент ИУ7-54Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Разин А.  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Волкова Л. Л.  
(И. О. Фамилия)

2023 г.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Линейный поиск . . . . .	4
1.2 Бинарный поиск . . . . .	4
<b>2 Конструкторская часть</b>	<b>5</b>
2.1 Псевдокоды рассматриваемых алгоритмов . . . . .	5
<b>3 Технологический раздел</b>	<b>7</b>
3.1 Средства реализации . . . . .	7
3.2 Реализация алгоритмов . . . . .	7
3.3 Тестирование . . . . .	7
<b>4 Исследовательская часть</b>	<b>8</b>
4.1 Технические характеристики . . . . .	8
4.2 Демонстрация работы программы . . . . .	8
4.3 Число сравнений . . . . .	9
<b>ЗАКЛЮЧЕНИЕ</b>	<b>12</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>13</b>
<b>ПРИЛОЖЕНИЕ А</b>	<b>14</b>

# ВВЕДЕНИЕ

Тема поиска всегда остается в центре внимания в информационных технологиях, задача поиска информации в структурах данных является одной из базовых задач программирования. Оптимизация поиска — одна из важнейших задач, ее решение позволяет оптимизировать работу различных баз и структур данных [1].

Целью данной работы является описание метода бинарного и классического поиска и их сопоставления по числу сравнений элементов.

Для поставленной цели необходимо выполнить следующие задачи.

1. Описать алгоритмы поиска.
2. Создать программное обеспечение, реализующее линейный и бинарный алгоритмы поиска.
3. Замерить число сравнений различных алгоритмов.
4. Провести анализ полученных результатов.

# 1 Аналитическая часть

В данной части работы будут рассмотрены бинарный и линейный алгоритмы поиска.

## 1.1 Линейный поиск

При использовании линейного поиска рассматриваются все элементы массива до встречи элемента, совпадающего с требуемым. Таким образом в худшем случае (отсутствия элемента в массиве) необходимо рассмотреть все элементы массива. Алгоритм имеет асимптотическую оценку  $O(n)$  [1].

## 1.2 Бинарный поиск

При использовании данного алгоритма предполагается упорядоченность данных, т. е. элементы массива упорядочены либо по не возрастанию либо по не убыванию. Изначально необходимо сравнить искомый элемент  $k$  с элементом массива, находящимся в середине, результат сравнения поможет определить в какой половине массива продолжить поиск, далее возможно применение той же процедуры к выбранному подмассиву и т. д. После не более чем  $\log_2 N$  сравнений либо ключ будет найден, либо будет установлено его отсутствие. Такая процедура иногда называется «логарифмическим поиском» или «методом деления пополам», но наиболее употребительный термин — бинарный поиск [1].

## Вывод

В данной части были рассмотрены идеи линейного и бинарного алгоритмов поиска.

## 2 Конструкторская часть

В данной части работы будет рассмотрен псевдокод линейного и бинарного алгоритмов поиска.

### 2.1 Псевдокоды рассматриваемых алгоритмов

В листингах 1–2 рассмотрены псевдокоды алгоритмов поиска, входными данными для них являются:

1. *array* — массив элементов;
2. *N* — число элементов в массиве;
3. *target* — число для поиска в массиве.

В случае отсутствия элемента в массиве происходит возврат  $-1$  — не валидного индекса массива. Оператор  $\leftarrow$  обозначает присваивание значение переменной, оператор  $[i]$  обозначает получение элемента из массива с индексом  $i$ , рассматривается только целочисленное деление, иные операторы подобны математическим.

---

**Алгоритм 1** Псевдокод алгоритма бинарного поиска.

---

**Условия:** Массив упорядочен по не убыванию.

*left*  $\leftarrow 0$

*right*  $\leftarrow N - 1$

**До тех пока** *left*  $\leq$  *right* **выполнять**

*middle*  $\leftarrow (left + right)/2$

**Если** *array*[*middle*]  $\leftarrow$  *target* **тогда**

**Возвратить** *middle*

**Конец условия**

**Если** *array*[*middle*]  $<$  *target* **тогда**

*left*  $\leftarrow middle + 1$

**иначе**

*right*  $\leftarrow middle - 1$

**Конец условия**

**Конец цикла**

**Возвратить**  $-1$

---

---

**Алгоритм 2** Псевдокод алгоритма линейного поиска.

---

Цикл  $i$  от 1 до  $N - 1$  выполнять

    Если  $array[i] = target$  тогда

        Возвратить  $i$

    Конец условия

Конец цикла

Возвратить  $-1$

---

## Вывод

В данной части работы был написан псевдокод для алгоритмов линейного и бинарного поиска.

## 3 Технологический раздел

В данной части работы будут описаны средства реализации программы, а также листинги, модульные и функциональные тесты.

### 3.1 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке C++, при использовании компилятора gcc версии 10.5.0, так как в стандартной библиотеке приведенного языка присутствует структура данных — вектор, которая позволяет хранить данные и которую возможно упорядочить с помощью средств стандартной библиотеки [2].

### 3.2 Реализация алгоритмов

Листинги исходных кодов программ А.1–А.2 приведены в приложении.

### 3.3 Тестирование

В таблице 3.1 приведены модульные тесты для разработанных алгоритмов поиска, в столбце «Цель» находятся искомые значения. Все тесты пройдены успешно.

Таблица 3.1 – Модульные тесты

Массив	Цель	Ожидаемый р-т	Фактический результат	
			Бинарный поиск	Линейный поиск
1 2 3 4	1	0	0	0
1 2 3 4	4	3	3	3
-9 -3 -2 -1 1 2 3	-1	3	3	3
-9 -3 -2 -1 1 2 3	1	4	4	4
-9 -3 -2 -1 1 2 3	-3	1	1	1

### Вывод

В данной части работы были представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

## **4 Исследовательская часть**

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

### **4.1 Технические характеристики**

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее.

1. Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12.
2. Оперативная память: 16 ГБайт.
3. Операционная система: Майкрософт Windows 10 Pro [3].
4. Используемая подсистема: WSL2 [4].

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

### **4.2 Демонстрация работы программы**

На рисунке 4.1 представлена демонстрация работы разработанного приложения для алгоритмов поиска.



```
Меню:
0)Выход
1)Поиск элемента с помощью бинарного поиска
2)Поиск элемента с помощью линейного поиска
3)Расчет числа сравнений

Введите пункт из меню: 1

Введите размер массива для поиска значения:
5

Введите массив для поиска значения:
1 2 3 4 5

Введите значение для поиска:
2

Индекс искомого значения:1
```

Рисунок 4.1 – Пример работы программы

### 4.3 Число сравнений

В результате проведения замеров по числу сравнений были получены столбчатые диаграммы 4.2–4.4, при выполнении замеров элементы массива были отсортированы по возрастанию и принимали значения от 0 до 1000. Результат числа сравнений под индексом  $-1$  является результатом поиска элемента, отсутствующего в рассматриваемом массиве.

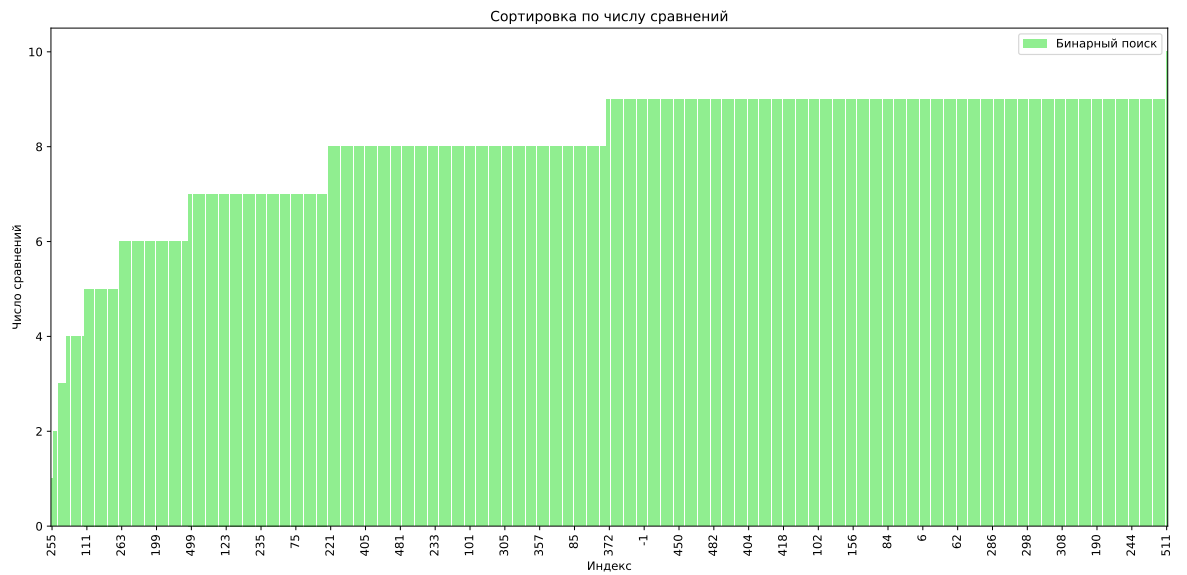


Рисунок 4.2 – Диаграмма зависимости числа сравнений от индекса массива при использовании бинарного поиска

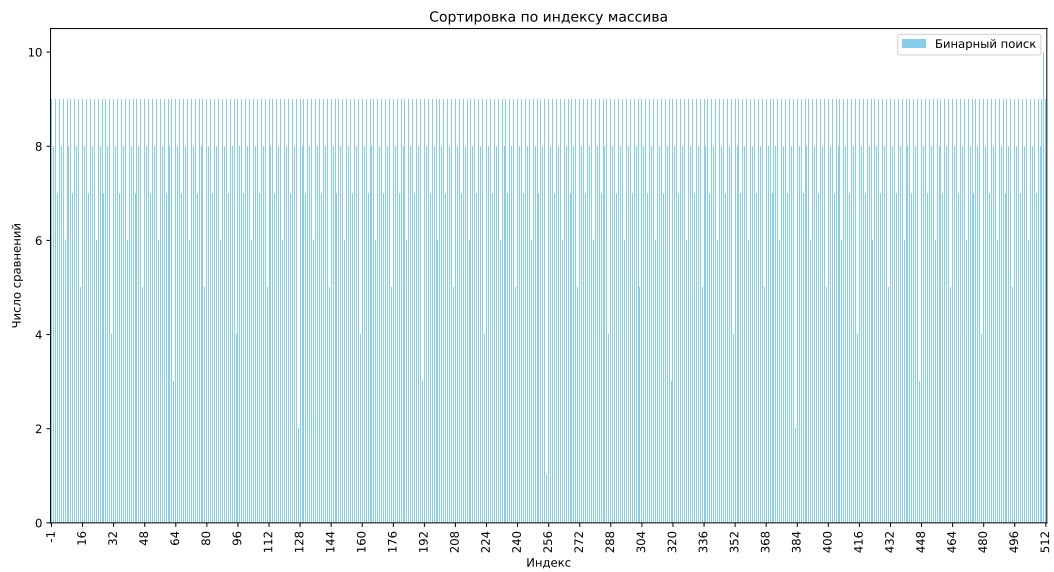


Рисунок 4.3 – Диаграмма зависимости числа сравнений от индекса массива при использовании бинарного поиска

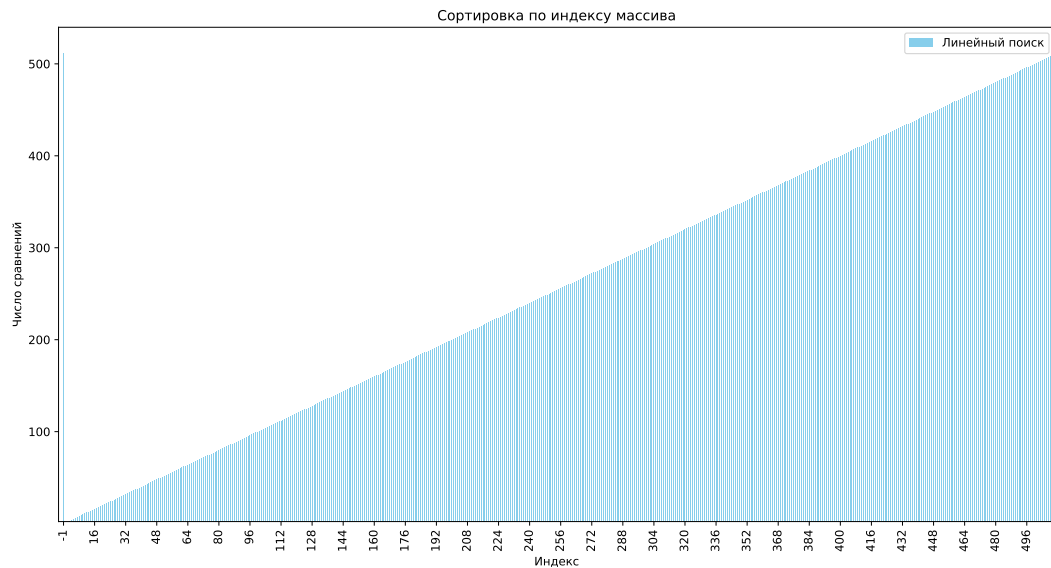


Рисунок 4.4 – Диаграмма зависимости числа сравнений от индекса массива при использовании линейного поиска

## Вывод

Из диаграмм 4.2–4.4, можно сделать вывод, что максимальное число сравнений при использовании реализации бинарного поиска равно 10, при использовании реализации линейного поиска — 512, таким образом бинарный поиск при поиске в массиве длиной 512 в худшем случае для обоих алгоритмов (искомый элемент — последний элемент массива) требует в 51.2 раза меньше сравнений, ввиду возможности рассмотрения подмассивов. Однако, при использовании бинарного поиска стоит учитывать условие упорядоченности массива.

## ЗАКЛЮЧЕНИЕ

В результате исследования что максимальное число сравнений при использовании реализации бинарного поиска равно 10, при использовании реализации линейного поиска — 512, таким образом бинарный поиск при поиске в массиве длиной 512 в худшем случае для обоих алгоритмов (искомый элемент — последний элемент массива) требует в 51.2 раза меньше сравнений, ввиду возможности деления массива пополам и рассмотрения значений только из подмассивов. Однако стоит учитывать условие упорядоченности массива, при использовании бинарного поиска.

Поставленная цель: описание метода бинарного и классического поиска и их сравнения по числу сравнений элементов, была достигнута. Для поставленной цели были выполнены все задачи.

1. Описать алгоритмы поиска.
2. Создать программное обеспечение, реализующее линейный и бинарный алгоритмы поиска.
3. Замерить число сравнений различных алгоритмов.
4. Провести анализ полученных результатов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Д. К. Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. // — М.: ООО «И.Д. Вильямс», 2014. — С. 824.
2. `std::vector` [Электронный ресурс]. — Режим доступа: <https://en.cppreference.com/w/cpp/container/vector> (дата обращения: 28.09.2023).
3. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — Режим доступа: <https://www.microsoft.com/ru-ru/software-download/windows10> (дата обращения: 28.09.2023).
4. Что такое WSL [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/ru-ru/windows/wsl/about> (дата обращения: 28.09.2023).

## ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация алгоритма бинарного поиска

```
1  int binSearch(std::vector<int>& v, int x,  
    std::vector<std::pair<int,int>>& res)  
2  {  
3      int low = 0;  
4      int high = (int)v.size() - 1;  
5      int cmpCount = 0;  
6      while (low <= high)  
7      {  
8          int mid = (high + low) / 2;  
9          res.emplace_back(mid, cmpCount);  
10         ++cmpCount;  
11  
12         if (v[mid] == x)  
13             return mid;  
14  
15         if (v[mid] < x)  
16             low = mid + 1;  
17  
18         else  
19             high = mid - 1;  
20     }  
21     ++cmpCount;  
22  
23     return -1;  
24  
25 }
```

Листинг А.2 – Реализация алгоритма линейного

```
1  int search(std::vector<int>& v, int x,  
    std::vector<std::pair<int,int>>& res)  
2  {  
3  
4      int cmpCount = 0;  
5      for (int i = 0; i < v.size(); i++)  
6      {  
7          ++cmpCount;  
8          res.emplace_back(i, cmpCount);
```

```
9      if (v[i] == x)
10         return i;
11     }
12     return -1;
13 }
```