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»

Лабораторная работа №3
по дисциплине «Анализ Алгоритмов»

Тема Алгоритмы поиска элемента в массиве

Студент Куликов Е. А.

Группа ИУ7-56Б

Преподаватель Волкова Л. Л.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной рабораторной работе изучаются алгоритмы поиска в массиве. Поиск — одна из базовых операций, производимых с массивами. Под поиском в массиве обычно понимают задачу нахождения индекса, по которому в массиве располагается некоторый элемент.

Целью данной лабораторной работы является изучение и реализация различных алгоритмов поиска, таких как поиск полным перебором и бинарный поиск. Для достижения поставленной цели необходимо решить следующие задачи:

— изучить алгоритм поиска полным перебором и алгоритм бинарного поиска в массиве;

— реализовать указанные алгоритмы;

— провести сравнение эффективности данных алгоритмов по количеству проверок элементов;

— описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы поиска в массиве.

1.1 Алгоритм поиска полным перебором

Если нет никакой дополнительной информации о способе хранения данных, то вероятнее всего данные расположены в произвольном порядке. В этом случае очевидный подход – простой последовательный просмотр массива или алгоритм полного перебора [1]. Для данной задачи алгоритм полного перебора заключается в проходе по массиву до тех пор, пока не будет найден указанный элемент или не закончится массив, что будет означать отсутствие указанного элемента в массиве.

1.2 Алгоритм бинарного поиска

Если поиск выполняется многократно и/или массив состоит из большого числа записей, то эффективней предварительно упорядочить этот массив (т.е. необходимо некоторым образом организовать данные) [1]. Для этого можно использовать один из алгоритмов сортировки.

Идея алгоритма:

— Вводятся две переменные, обозначающие индексы левой и правой границ поиска в массиве;

— Определяется значение в середине заданной зоны поиска;

— Если это значение больше, чем искомое, поиск продолжается в первой половине зоны поиска (правая граница смещается к центральному элементу), иначе поиск продолжается во второй половине зоны поиска (левая граница смещается к центральному элементу);

— Процесс продолжается до тех пор, пока не будет найден искомый элемент или не станет пустой зона поиска.

1.3 Оценки трудоемкости алгоритмов

Дадим оценку трудоемкости алгоритмов в терминах числа сравнений, которые понадобятся для нахождения ответа.

1.3.1 Алгоритм поиска полным перебором

Для алгоритма полного перебора число сравнений равно:

$$N = \begin{cases} i + 1, & \text{i - индекс эл-та в массиве} \\ N, & \text{если эл-та нет в массиве} \end{cases} \tag{1.1}$$

Поэтому лучший случай — 1 сравнение, когда искомый элемен является первым в массиве. Худший случай — N сравнений, когда искомый элемент является последним в массиве или когда искомый элемент в массиве отсутствует.

Алгоритмическая сложность данного алгоритма зависит от положения элемента в массиве, чем больше индекс элемента, тем больше проверок придется совершить. В общем случае алгоритм имеет сложность $O(n)$.

1.3.2 Алгоритм бинарного поиска

Для алгоритма бинарного поиска количество сравнений растет в зависимости от «нечетности» положения элемента. Например, для центрального элемента массива (расположенного по индексу $N//2$) будет проведена 2 проверки (считаем и проверку на равенство, и проверку на больше/меньше). В два раза больше проверок будет проведено для элементов, расположенных по индексам кратным $N//4$ — $N//4$, $3N//4$. И так далее. Примерное правило: для элемента будет проведено $2 \cdot k$ проверок, если индекс этого элемента равен $N//(2^k)$.

Поэтому лучший случай для этого алгоритма — 2 проверки, когда искомый элемент находится посередине массива. Худший случай — $2 \cdot log_2(N)$ проверок, когда индекс элемента кратен $log_2(N)$.

Для заранее упорядоченного массива алгоритмическая сложность поиска случайного элемента равна $O(log(n))$, что лучше чем $O(n)$ у алгоритма поиска полным перебором. Однако если исходные данные не были изначально отсортированными, затраты на сортировку могут нивелировать это преимущество.

Вывод

В данном разделе были рассмотрены алгоритмы поиска в массиве.

2 Конструкторская часть

В данном разделе будут разработаны алгоритм бинарного поиска и алгоритм поиска полным перебором .

2.1 Разработка алгоритмов

На вход каждому алгоритму подается массив целых чисел, его размер и искомый элемент.

На рис. 2.1 — 2.2 приведены алгоритмы поиска полным перебором и бинарного поиска.

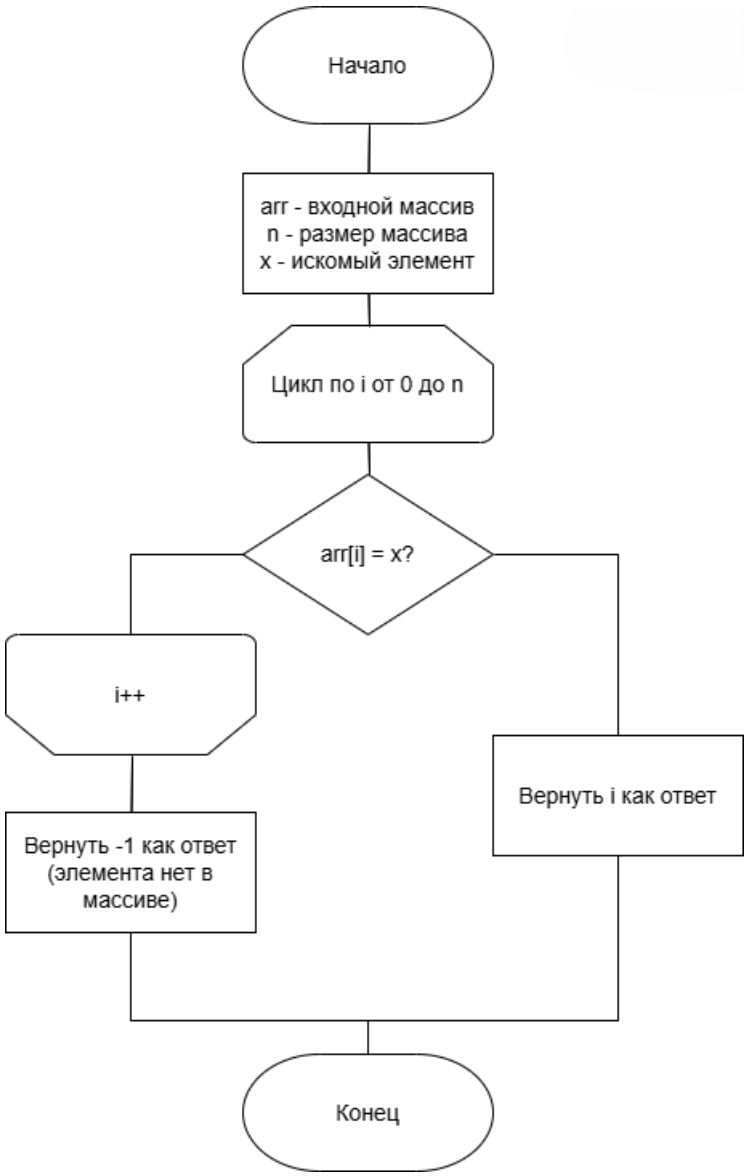

Рисунок 2.1 – Алгоритм поиска полным перебором

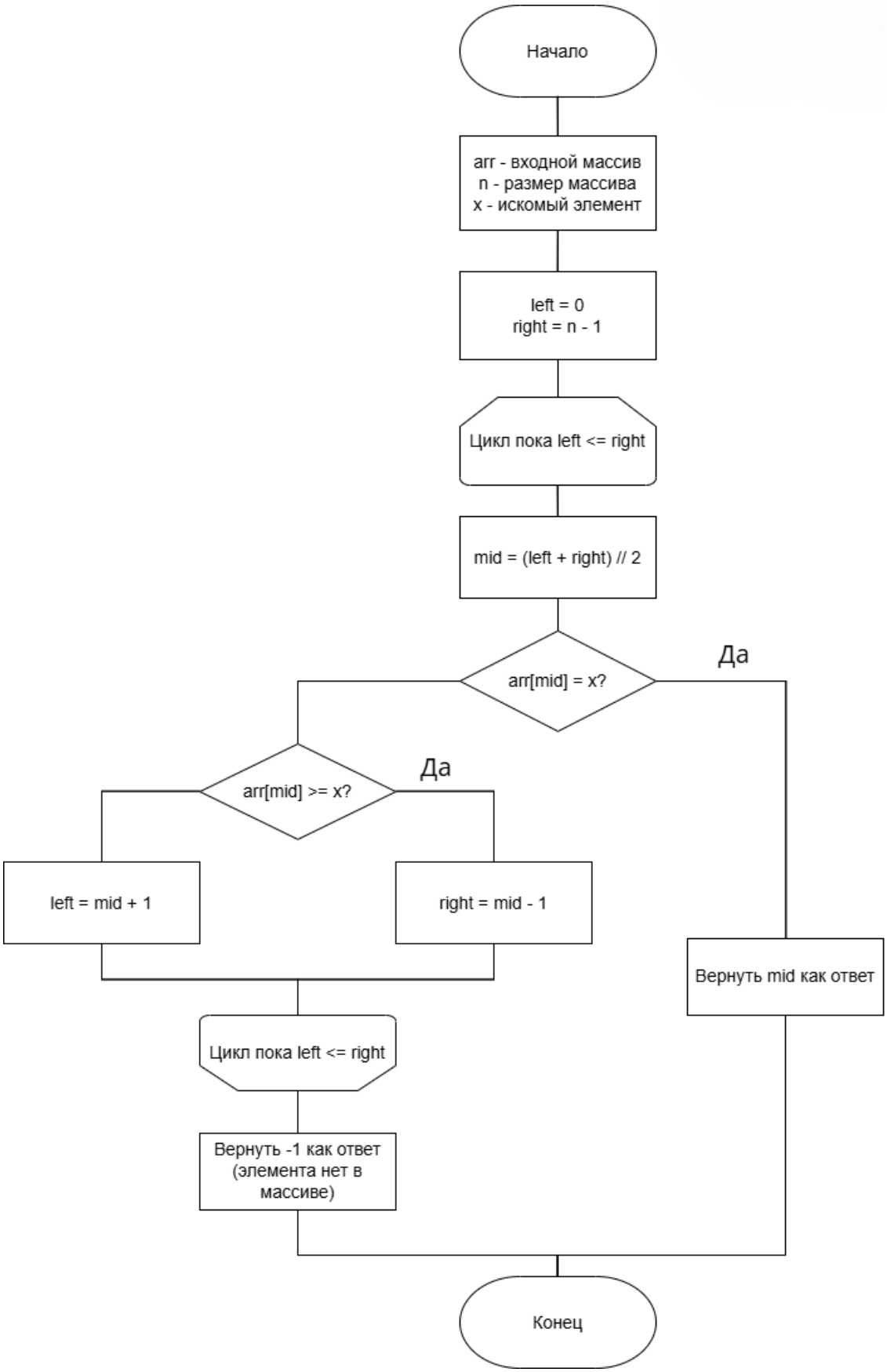

Рисунок 2.2 – Алгоритм бинарного поиска

Вывод

В данном разделе были разработаны алгоритмы поиска полным перебором и бинарного поиска.

3 Технологическая часть

В данном разделе будут приведены данные о выбранном языке программирования, коды алгоритмов и тесты для каждого алгоритма.

3.1 Требования к ПО

Реализуемое ПО будет давать возможность ввести размер массива n, по которому будет сгенерирован массив из чисел от 0 до n, распределенных случайно, ввести искомый элемент. На выходе пользователь получает индекс найденного элемента или -1, если элемент не найден, а также количество проверок, проведенных во время работы каждого алгоритма.

3.2 Язык программирования

В данной работе для реализации алгоритмов был выбран язык программирования *Python* [2]. Язык предоставляет возможность работы с массивами различной длины, а также возможность создания пользовательского интерфейса.

3.3 Реализация алгоритмов

В листингах 1 — 2 представлены реализации алгоритма поиска полным перебором и алгоритма бинарного поиска.

Листинг 1 – Реализация алгоритма поиска полным перебором

```
def full_search(array: list[int], elem: int) -> (int, int):
    checks = 0
    for i in range(len(array)):
        checks += 1
        if array[i] == elem:
            return i, checks
    return -1, checks
```

Листинг 2 – Реализация алгоритма бинарного поиска в массиве

```
def binary_search(array: list[int], elem: int) -> (int, int):
    array.sort()
    left_border = 0
    right_border = len(array) - 1
    checks = 0
    while left_border <= right_border:
        checks += 2
        mid = (left_border + right_border) // 2
        if array[mid] == elem:
            return mid, checks
        elif array[mid] >= elem:
            right_border = mid - 1
        else:
            left_border = mid + 1
    return -1, checks
```

3.4 Функциональные тесты

В листинге 3 описан вывод программы для различных функциональных тестов. Все тесты программа прошла успешно.

Листинг 3 – Функциональные тесты

```
Enter the size of the array: 10
Your array:  [1, 5, 4, 0, 3, 6, 7, 9, 2, 8]
Enter element: 4
FULL SEARCH index found: 2, checks: 3
BINARY SEARCH index found: 2, checks: 2

Enter the size of the array: 10
Your array:  [5, 0, 6, 9, 4, 2, 8, 7, 1, 3]
Enter element: 5
FULL SEARCH index found: 0, checks: 1
BINARY SEARCH index found: 0, checks: 6

Enter the size of the array: 10
Your array:  [3, 5, 6, 8, 7, 1, 2, 0, 4, 9]
Enter element: -1
FULL SEARCH index not found, checks: 10
BINARY SEARCH index not found, checks: 6
```

Вывод

В данном разделе были приведены коды реализаций алгоритмов поиска, а также функциональные тесты.

4 Исследовательская часть

В данном разделе будут представлены результаты исследования эффективности работы алгоритмов по количеству проведенных проверок. При этом не учитывается проводимая предварительная сортировка входного массива для алгоритма бинарного поиска.

4.1 Замеры количества проверок

Результаты замеров количества произведенных проверок от позиции искомого элемента массива представлены на графиках 4.1 — 4.3. Замеры производились на массиве длины 1086 в соответствии с вариантом задания.

На рисунке 4.1 приведены замеры количества проведенных проверок от индекса искомого элемента для алгоритма поиска полным перебором, в том числе и для элемента, которого нет в массиве (ему соответствует левый высокий столбик графика)

На рисунке 4.2 приведены аналогичные замеры для алгоритма бинарного поиска

На рисунке 4.3 приведен отсортированный по количеству проверок график для алгоритма бинарного поиска для визуально более простого сравнения его с графиком поиска полным перебором.

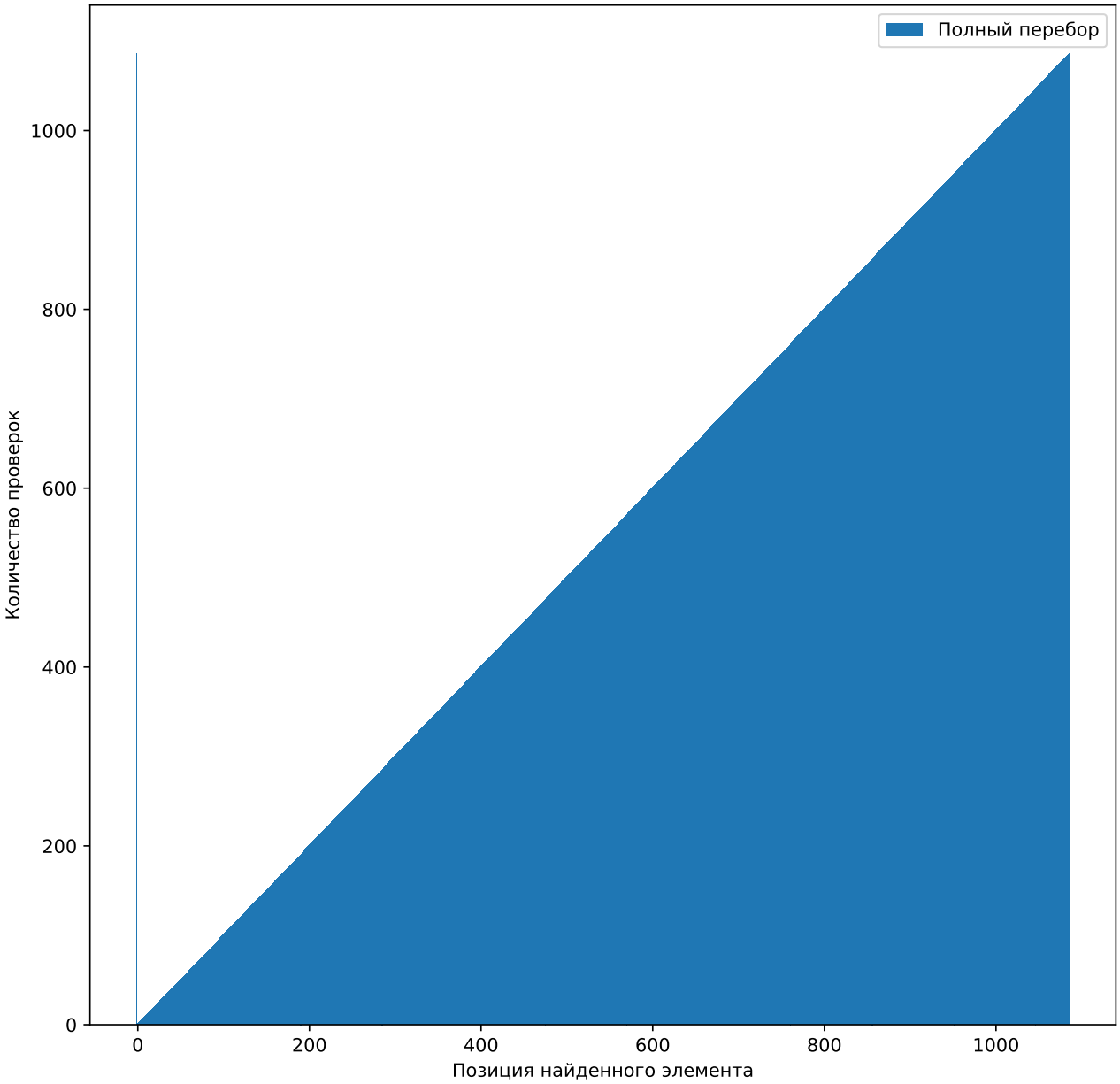

Рисунок 4.1 – График количества проверок для поиска полным перебором i-го элемента от i

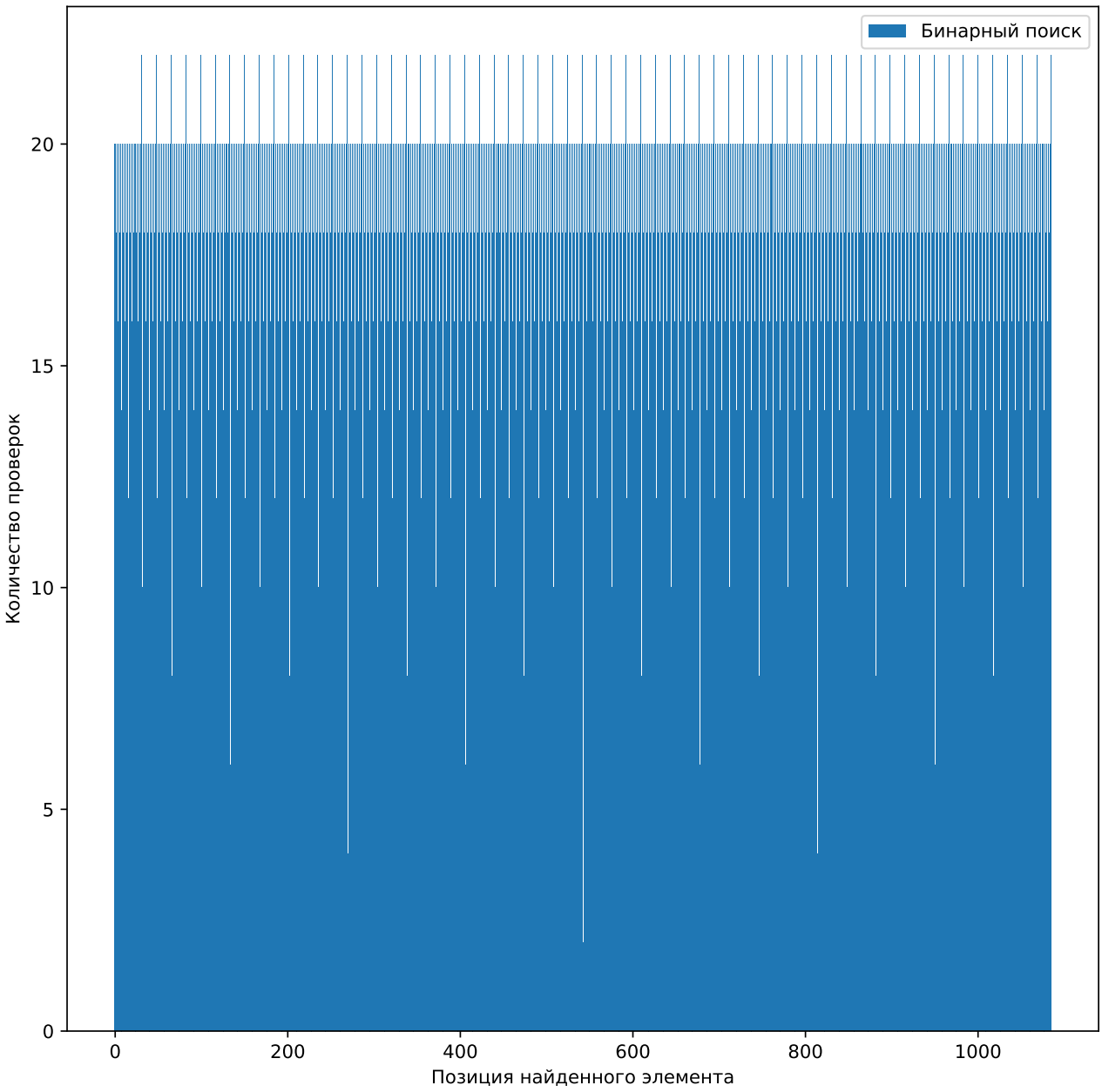

Рисунок 4.2 – График количества проверок для бинарного поиска i-го элемента от i

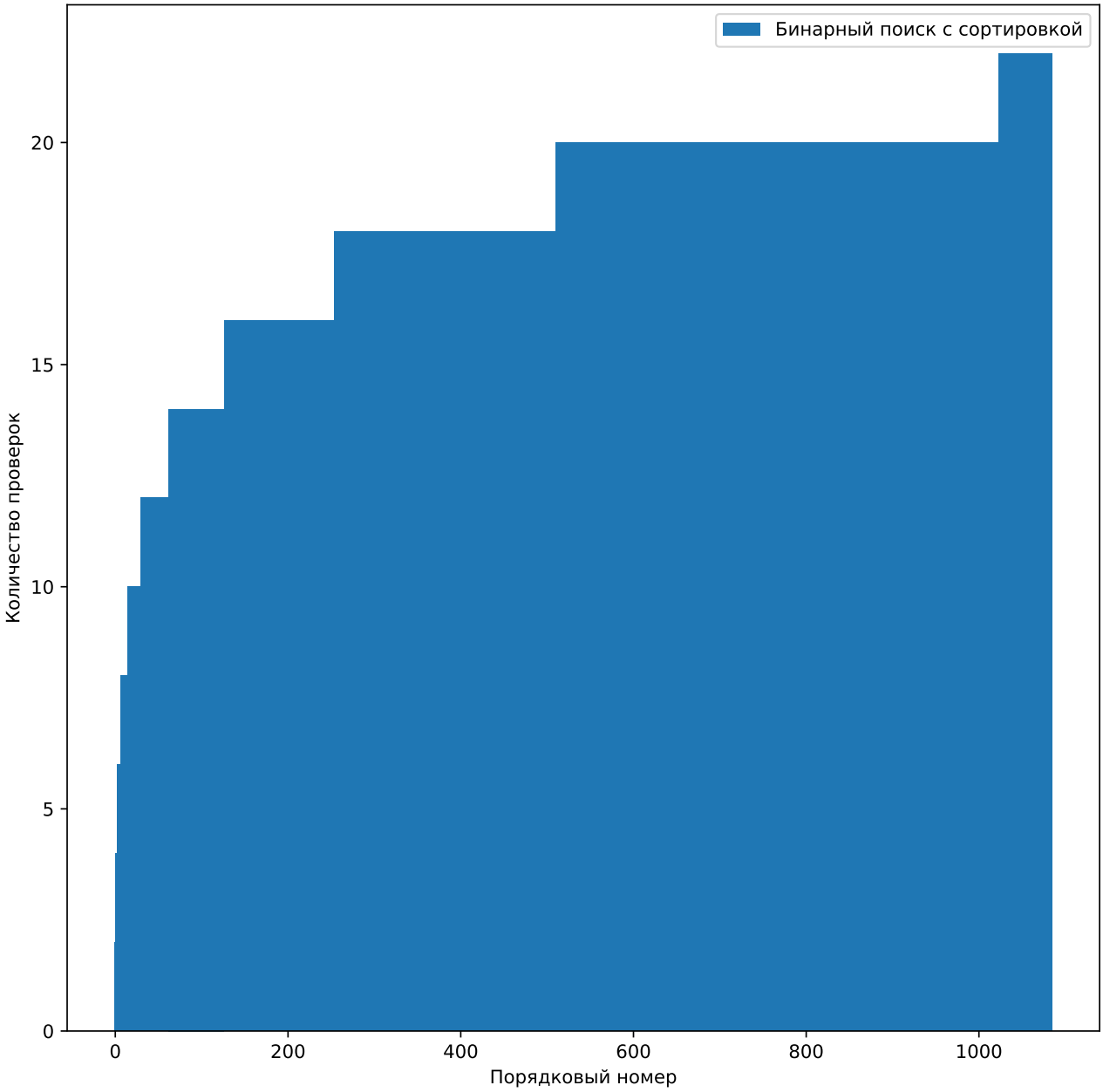

Рисунок 4.3 – Отсортированный по количеству проверок график для поиска полным перебором

Вывод

Алгоритм бинарного поиска производит в среднем существенно меньше проверок, чем алгоритм поиска полным перебором. На графике 4.1 видно, что в среднем для случайного элемента массива было произведено $1086/2 = 543$ проверки, в то время как для бинарного поиска было произведено максимум 22 проверки (см график 4.2). Однако на графике не учтено, что перед бинарным поиском массив был отсортирован, то есть преимущество в количестве проверок могло быть нивелировано необходимостью сортировки массива.

Также, из графиков 4.1 и 4.2 видно, что в алгоритма поиска полным перебором количество проверок растет линейно от позиции искомого элемента, в то время как для алгоритма бинарного поиска количество проверок для каждого элемента зависит от его положения в массиве: чем ближе элемент расположен к индексу, равному степени двойки, и чем меньше эта степень, тем быстрее элемент будет найден. Так, меньше всего проверок нужно для элемента, расположенного посередине, на две проверки больше для элементов, расположенных в четвертях и так далее. Это также объясняет увеличение ширины столбцов одинаковой высоты ровно в 2 раза на графике 4.3 — каждый столбец содержит количество проверок для $n - 1$ элементов, расположенных в $1/2^n$ частях исходного массива.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены алгоритмы бинарного поиска и поиска полным перебором, а также решены следующие задачи:

— изучены алгоритм поиска полным перебором и алгоритм бинарного поиска в массиве;

— реализованы указанные алгоритмы;

— проведено сравнение эффективности данных алгоритмов по количеству проверок элементов;

— описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Алгоритм бинарного поиска производит в среднем существенно меньше проверок, чем алгоритм поиска полным перебором. На графике 4.1 видно, что в среднем для случайного элемента массива было произведено $1086/2 = 543$ проверки, в то время как для бинарного поиска было произведено максимум 22 проверки (см график 4.2). Однако на графике не учтено, что перед бинарным поиском массив был отсортирован, то есть преимущество в количестве проверок могло быть нивелировано необходимостью сортировки массива.

Также, из графиков 4.1 и 4.2 видно, что в алгоритма поиска полным перебором количество проверок растет линейно от позиции искомого элемента, в то время как для алгоритма бинарного поиска количество проверок для каждого элемента зависит от его положения в массиве: чем ближе элемент расположен к индексу, равному степени двойки, и чем меньше эта степень, тем быстрее элемент будет найден. Так, меньше всего проверок нужно для элемента, расположенного посередине, на две проверки больше для элементов, расположенных в четвертях и так далее. Это также объясняет увеличение ширины столбцов одинаковой высоты ровно в 2 раза на графике 4.3 — каждый столбец содержит количество проверок для $n - 1$ элементов, расположенных в $1/2^n$ частях исходного массива.

Поставленная цель достигнута, все задачи решены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Самуйлов, Сергей Владимирович. Структуры и алгоритмы обработки данных : учеб. пособие / С. В. Самуйлов, С. В. Самуйлова, Л. В. Гурьянов. — Пенза : Изд-во ПГУ, 2023. – 80 с.

[2] Документация языка Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 30.09.2024).