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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 3

по курсу «Анализ алгоритмов»

на тему: «Алгоритмы поиска в массиве»

Студент ИУ7-55Б _____ И. Д. Половинкин
(Подпись, дата)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата)

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной лабораторной работы: исследовать различные алгоритмы поиска элемента в массиве.

Для её достижения были поставлены следующие задачи:

- рассмотреть алгоритмы поиска в массиве: линейный и бинарный;
- реализовать алгоритмы поиска в массиве;
- провести сравнительный анализ этих алгоритмов по числу сравнений до нахождения искомого элемента на практике;

1 Аналитический раздел

1.1 Линейный алгоритм поиска

Линейный (последовательный) поиск — это метод поиска элемента в массиве или списке, при котором проверка осуществляется последовательно, начиная с первого элемента и продолжаясь до тех пор, пока не будет найден искомый элемент или не будет пройден весь набор данных [1].

Линейный поиск имеет временную сложность $O(N)$, где N —– размер массива. В лучшем случае элемент находится сразу, что даёт сложность $O(1)$. В худшем случае требуется перебрать весь массив, либо элемент отсутствует, тогда сложность составляет $O(N)$. Алгоритм использует $O(1)$ дополнительной памяти, так как хранит лишь одну переменную для перебора элементов.

1.2 Бинарный алгоритм поиска

Бинарный (двоичный) поиск — это алгоритм поиска элемента в отсортированном массиве, основанный на принципе деления диапазона пополам. На каждом шаге алгоритм сравнивает искомый элемент с серединным значением массива и продолжает поиск в левой или правой половине, в зависимости от результата сравнения. Этот процесс повторяется, пока не будет найден нужный элемент или диапазон поиска не станет пустым [2].

Бинарный поиск имеет временную сложность $O(\log N)$. В лучшем случае, если искомый элемент находится сразу в середине алгоритм выполняется за $O(1)$. В худшем случаях количество проверок сокращается вдвое на каждом шаге, что приводит к временной сложности $O(\log N)$. Алгоритм требует $O(1)$ дополнительной памяти, но при рекурсивной реализации расходуется $O(\log N)$ памяти на хранение вызовов стека.

2 Конструкторский раздел

2.1 Алгоритмы поиска в массиве

2.1.1 Линейный алгоритм поиска

На рисунке 2.1 представлен линейный алгоритм поиска элемента в массиве.

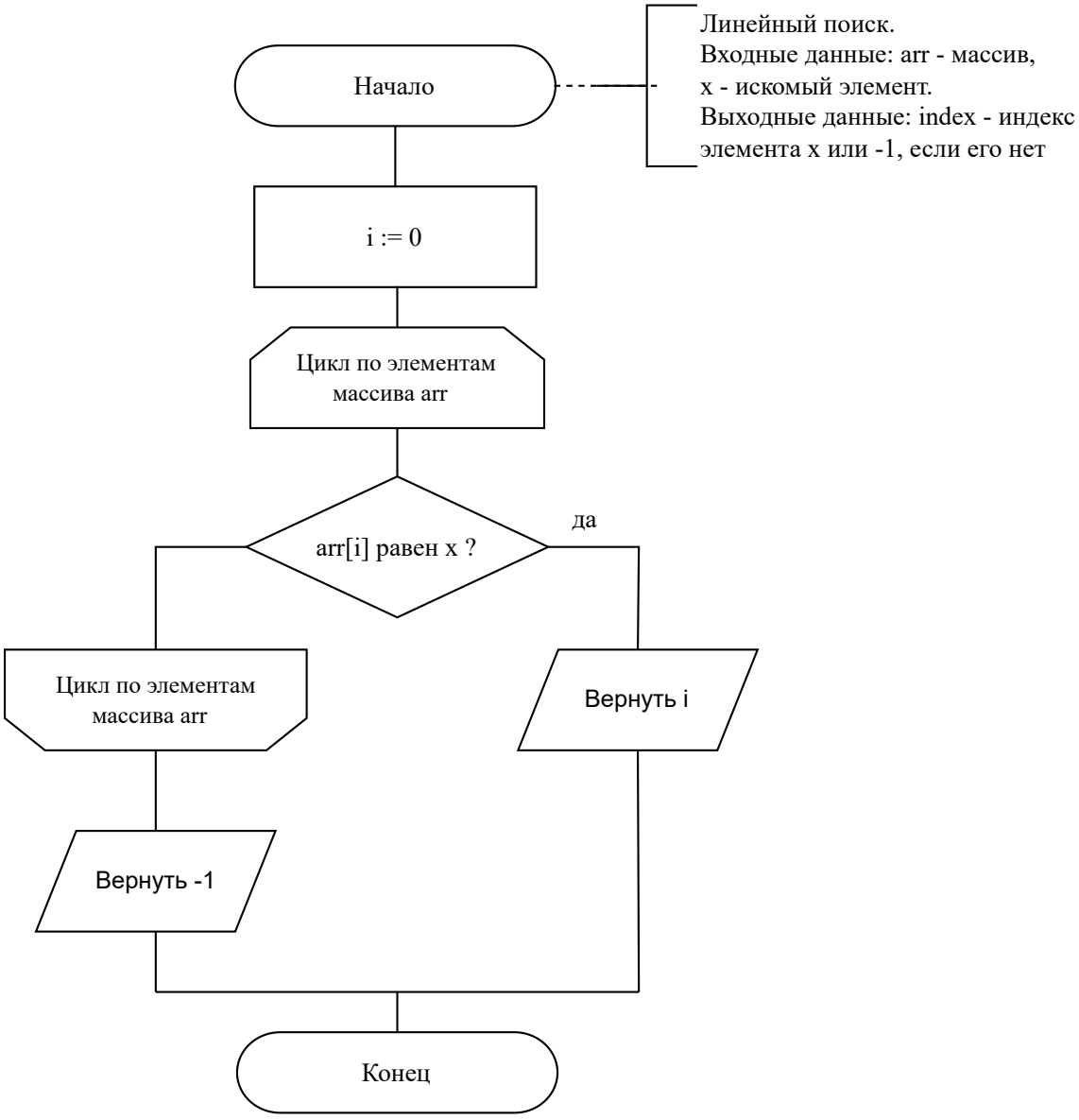

Рисунок 2.1 – Линейный алгоритм поиска

2.1.2 Бинарный алгоритм поиска

На рисунке 2.2 представлен бинарный алгоритм поиска элемента в массиве.

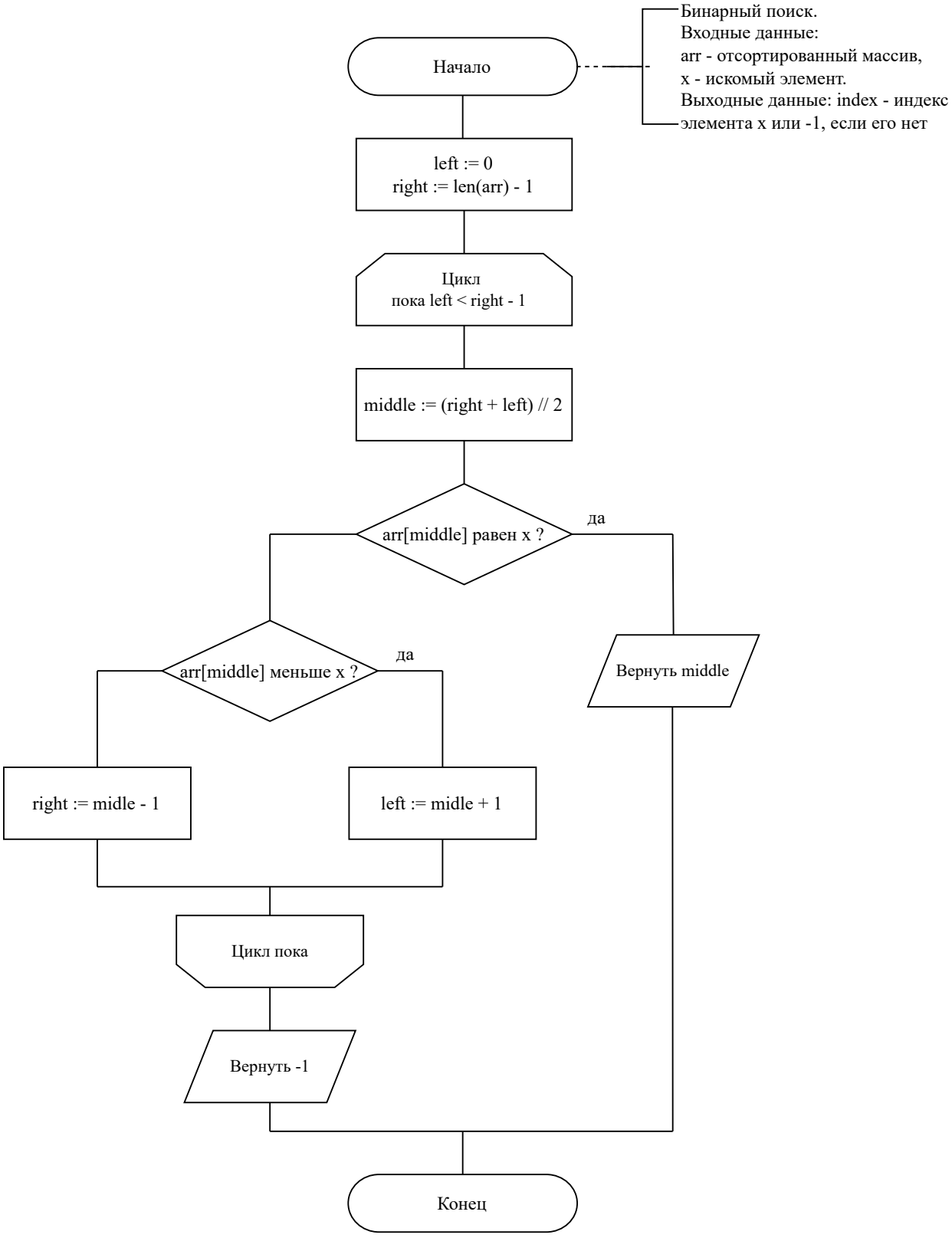

Рисунок 2.2 – Бинарный алгоритм поиска

3 Технологический раздел

3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

– на вход программе подаются массив целых чисел и целое число, которое будет искаться в нём;

– осуществляется выбор алгоритма на исследование;

– на вход программе подаются только корректные данные;

– в зависимости от выбранного пункта меню на выходе программа выдает индекс найденного в массиве элемента или результат исследования.

3.2 Выбор средств реализации

Для реализации алгоритмов был выбран язык программирования Python [3].

3.3 Реализация алгоритмов

3.3.1 Алгоритм линейного поиска

Реализация алгоритма линейного поиска элементов в массиве приведена на листинге 3.1.

Листинг 3.1 – Алгоритм линейного поиска

```
def linear_search(arr, x):
    if len(arr) == 0:
        return -1, 0
    comparisons = 0
    for index, i in enumerate(arr):
        comparisons += 1
        if i == x:
            return index + 1, comparisons
    return -1, comparisons
```

3.3.2 Алгоритм бинарного поиска

Реализация алгоритма бинарного поиска элементов в массиве приведена на листинге 3.2.

Листинг 3.2 – Алгоритм бинарного поиска

```
def binary_search(arr, x):
    if len(arr) == 0:
        return -1, 0
    comparisons = 0
    left = 0
    right = len(arr) - 1
    while left <= right:
        comparisons += 1
        middle = (left + right) // 2
        if arr[middle] == x:
            return middle, comparisons
        elif arr[middle] < x:
            left = middle + 1
        else:
            right = middle - 1
    return -1, comparisons
```

3.4 Тестирование

Для тестирования линейного и бинарного алгоритмов поиска элементов в массиве были составлены таблицы с входными данными (массив и искомый элемент), ожидаемым результатом (индексом) и полученным результатом от обоих способов.

Таблица 3.1 – Таблица тестов для алгоритмов поиска элементов в массиве

Массив	Искомый элемент	Ожидание		Результат	
		Линейный	Бинарный	Линейный	Бинарный
[]	1	-1	-1	-1	-1
[1, 2, 3]	4	-1	-1	-1	-1
[1, 2, 3]	1	1	1	1	1
[1, 2, 3]	2	1	1	1	1
[1, 2, 3]	3	2	2	2	2
[3, 2, 1]	1	2	0	2	0

Для линейного и бинарного алгоритмов поиска элемента в массиве данные тесты были пройдены успешно.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

– операционная система: Windows 10 Pro;

– память: 8 Гб;

– процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60 ГГц 1.80 ГГц.

Тестирование проводилось на ноутбуке, который был подключен к сети питания. Во время проведения тестирования ноутбук был нагружен только встроенными приложениями окружения, самим окружением и системой тестирования.

4.2 Сравнение алгоритмов

Трудоемкость алгоритма оценивается по количеству сравнений, необходимых для поиска элемента. Размер массива, который был предварительно заполнен целыми числами случайным образом, составляет 1091. На рисунке 4.1 представлена гистограмма, отражающая зависимость количества сравнений от индекса искомого элемента при линейном поиске.

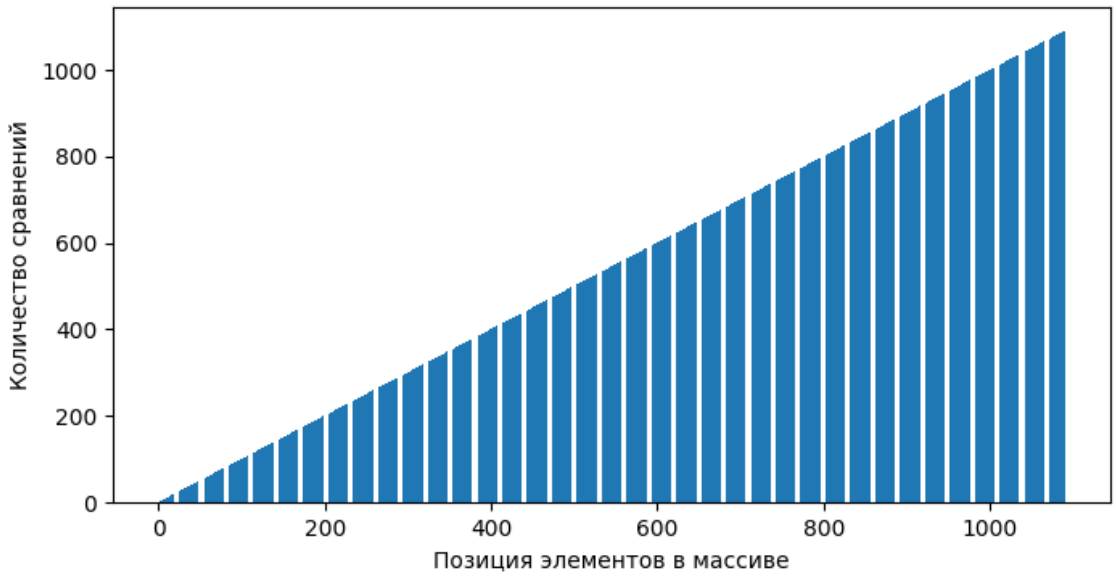

Рисунок 4.1 – Гистограмма зависимости числа сравнений от индекса искомого элемента при линейном поиске.

На рисунках 4.2 и 4.3 представлены гистограммы работы алгоритма бинарного поиска. Гистограмма на рисунке 4.3 показывает количество сравнений, отсортированных по возрастанию.

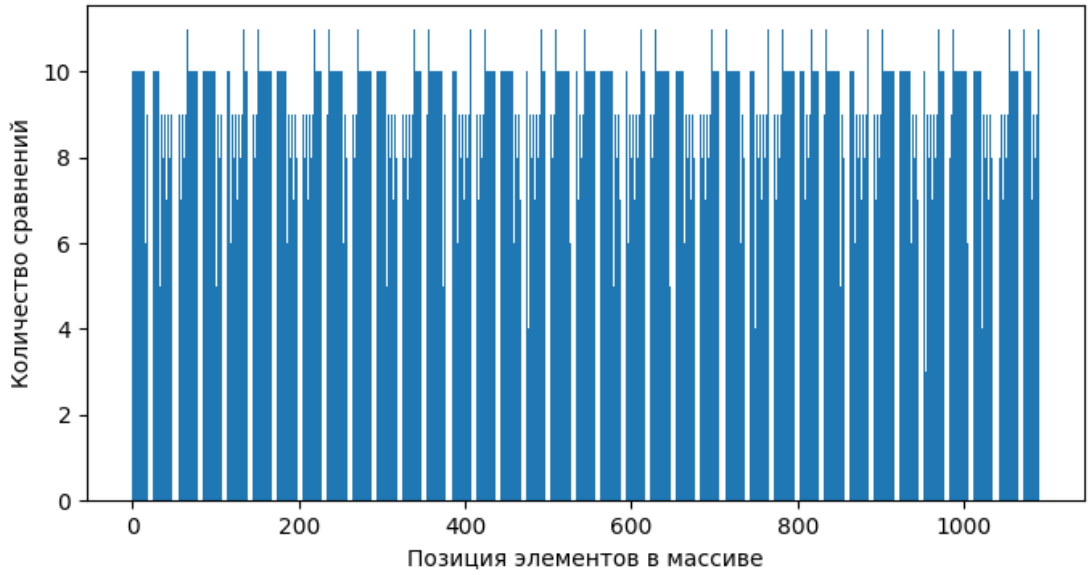

Рисунок 4.2 – Гистограмма зависимости числа сравнений от индекса искомого элемента при бинарном поиске.

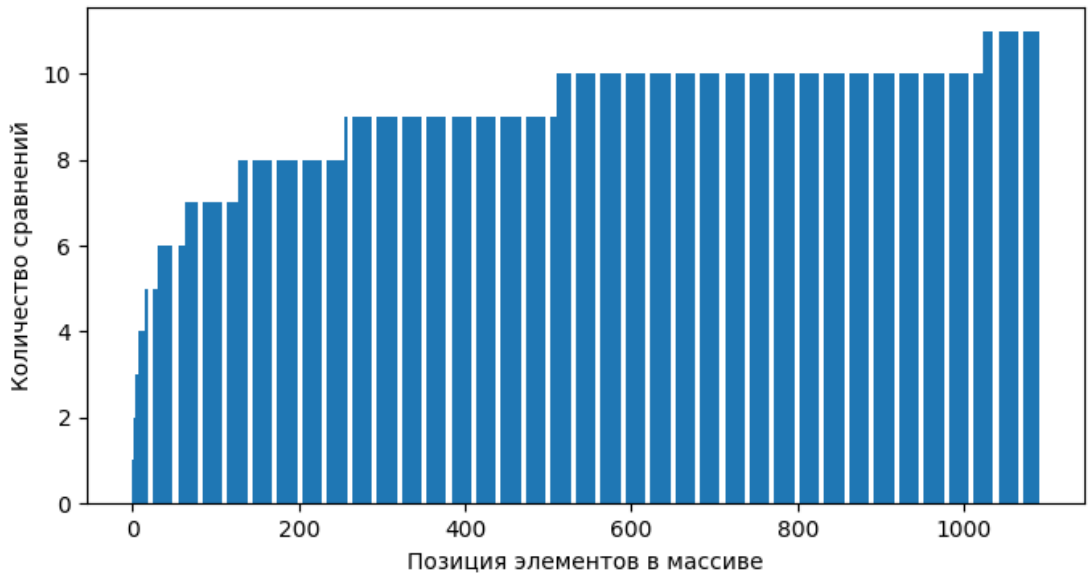

Рисунок 4.3 – Гистограмма зависимости числа сравнений от индекса искомого элемента при бинарном поиске, отсортированная по возрастанию количества сравнений.

Вывод

В ходе исследования были рассмотрены и сравнены линейный и бинарный алгоритмы поиска. Линейный поиск выполняет последовательное сравнение элементов массива с искомым значением, что приводит к числу сравнений, равному позиции элемента плюс один. В худшем случае, если элемента нет в массиве, необходимо выполнить n сравнений, где n — размер массива.

Бинарный поиск использует метод деления массива пополам, сокращая область поиска на каждом шаге. Это позволяет уменьшить количество сравнений: в худшем случае оно составляет примерно $\log_2(n)$. Например, для массива из 50 элементов максимальное количество сравнений составит 6. Основное ограничение бинарного поиска заключается в необходимости предварительной сортировки массива, которая может потребовать дополнительных затрат времени.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были исследованы линейный и бинарный алгоритмы поиска элементов в массиве. Было выполнено описание каждого из этих алгоритмов, сравнена эффективность этих алгоритмов на практике. Цель работы достигнута. Поставленные задачи решены.

При сравнении полученных программ сделан вывод о том, что бинарный алгоритм работает быстрее линейного, так как в большинстве случаев делает меньше сравнений, но, в связи с тем, что для бинарного поиска нужен отсортированный массив, такого выигрыша по времени не удастся добиться на не отсортированных данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритм линейного поиска [Электронный ресурс]. – Режим доступа, URL: https://kvodo.ru/lineynyiy-poisk.html (дата обращения: 20.12.2024)

2. Двоичный (бинарный) поиск [Электронный ресурс]. – Режим доступа, URL: https://kvodo.ru/dvoichnyiy-poisk-2.html (дата обращения: 20.12.2024)

3. Python 3.12.1 documentation [Электронный ресурс]. – Режим доступа, URL: https://docs.python.org/3/ (дата обращения: 20.12.2024)