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 Поиск элемента в массиве

Студент Тузов Даниил Александрович

Группа ИУ7-52Б

Преподаватель Кормановский Михаил Владимирович

Москва, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В третьей лабораторной работе по анализу алгоритмов рассматриваются алгоритмы поиска в массиве.

Целью лабораторной работы является изучение алгоритмов поиска элемента в массиве. Для достижения поставленной цели небходимо решить следующие задачи:

- изучить теоретические аспекты алгоритмов поиска в массиве;
- разработать алгоритмы поиска в массиве:
  - алгоритм линейного поиска;
  - алгоритм бинарного поиска;
- провести для каждого алгоритма сравнительный анализ количества операций сравнения относительно расположения искомого элемента в массиве;
- обосновать полученные результаты.

1 Аналитическая часть

В этой части рассматриваются теоретические аспекты алгоритмов поиска в массиве.

1.1 Линейный поиск

Линейный поиск (также известный как последовательный поиск) – это алгоритм поиска элемента в списке или массиве, который последовательно проверяет каждый элемент до тех пор, пока не найдёт совпадение или не проверит все элементы.

Сложность такого алгоритма $O(N)$[1], где N – это размер массива, в котором ищется элемент.

1.2 Бинарный поиск

Бинарный поиск – это алгоритм поиска элемента в отсортированном массиве (векторе). Также известен как метод деления пополам или дихотомия.

Принцип работы бинарного поиска:

1. определяется значение элемента в середине структуры данных;
2. полученное значение сравнивается с ключом;
3. если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе – во второй;
4. поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом;
5. процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

Сложность такого алгоритма $O(logN)$[1], где N – это размер массива, в котором ищется элемент.

1.3 Вывод

В этой части были рассмотрены теоретические аспекты алгоритмов поиска в массиве: алгоритма линейного поиска и алгоритмов бинарного поиска. Заявленная сложность алгоритма бинарного поиска лучше, чем у алгоритма линейного поиска.

2 Конструкторская часть

В этой части представлены описания алгоритмов.

2.1 Описание алгоритмов

На вход алгоритмам подаются массив `arr` (для бинарного поиска массив отсортирован) и искомый элемент `el`, на выходе единственное число – индекс элемента в массиве или значение -1, если элемент не найден.

На рисунках 1 - 2 приведены описания алгоритмов.

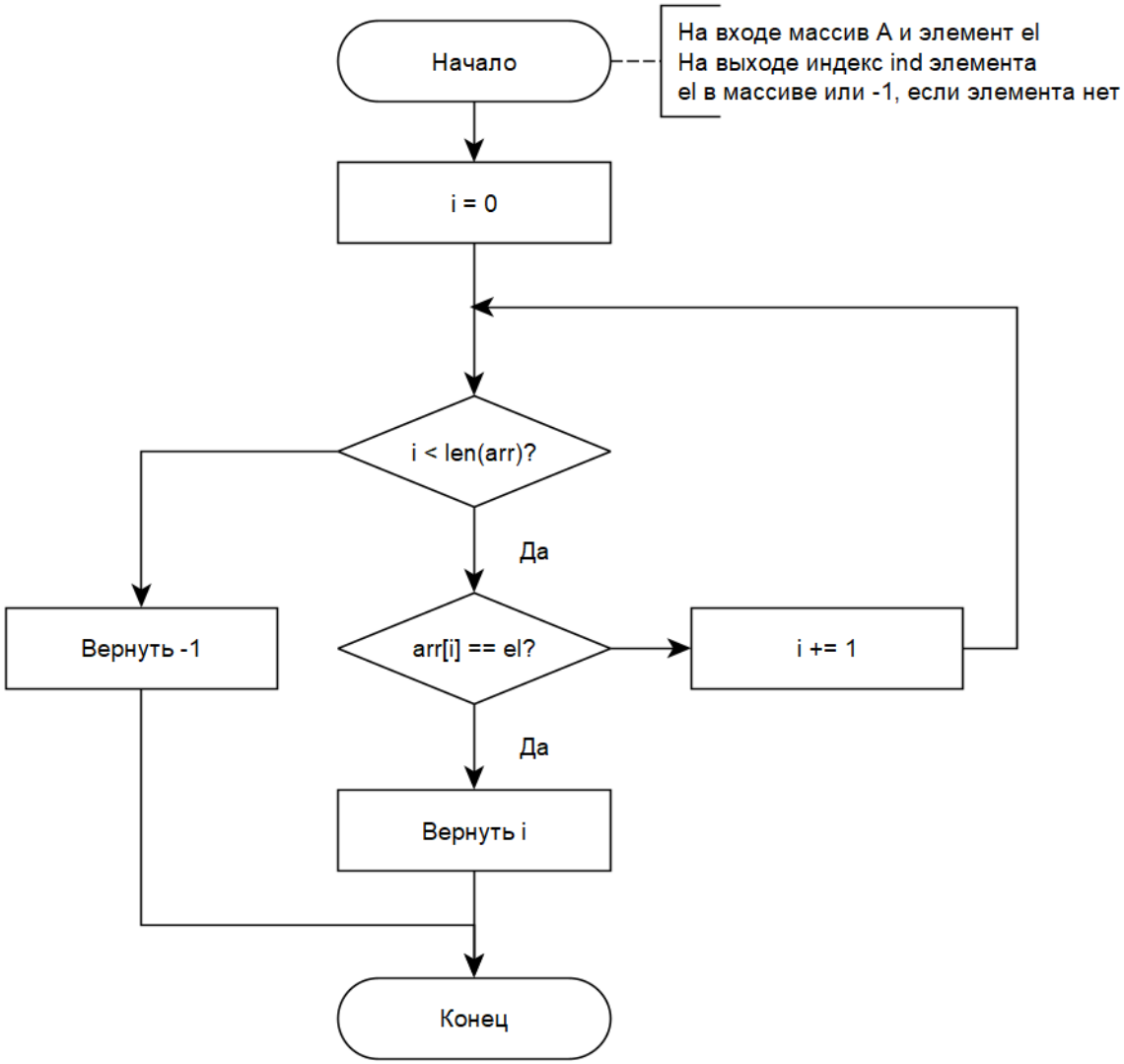

Рисунок 1 – Описание алгоритма линейного поиска в массиве

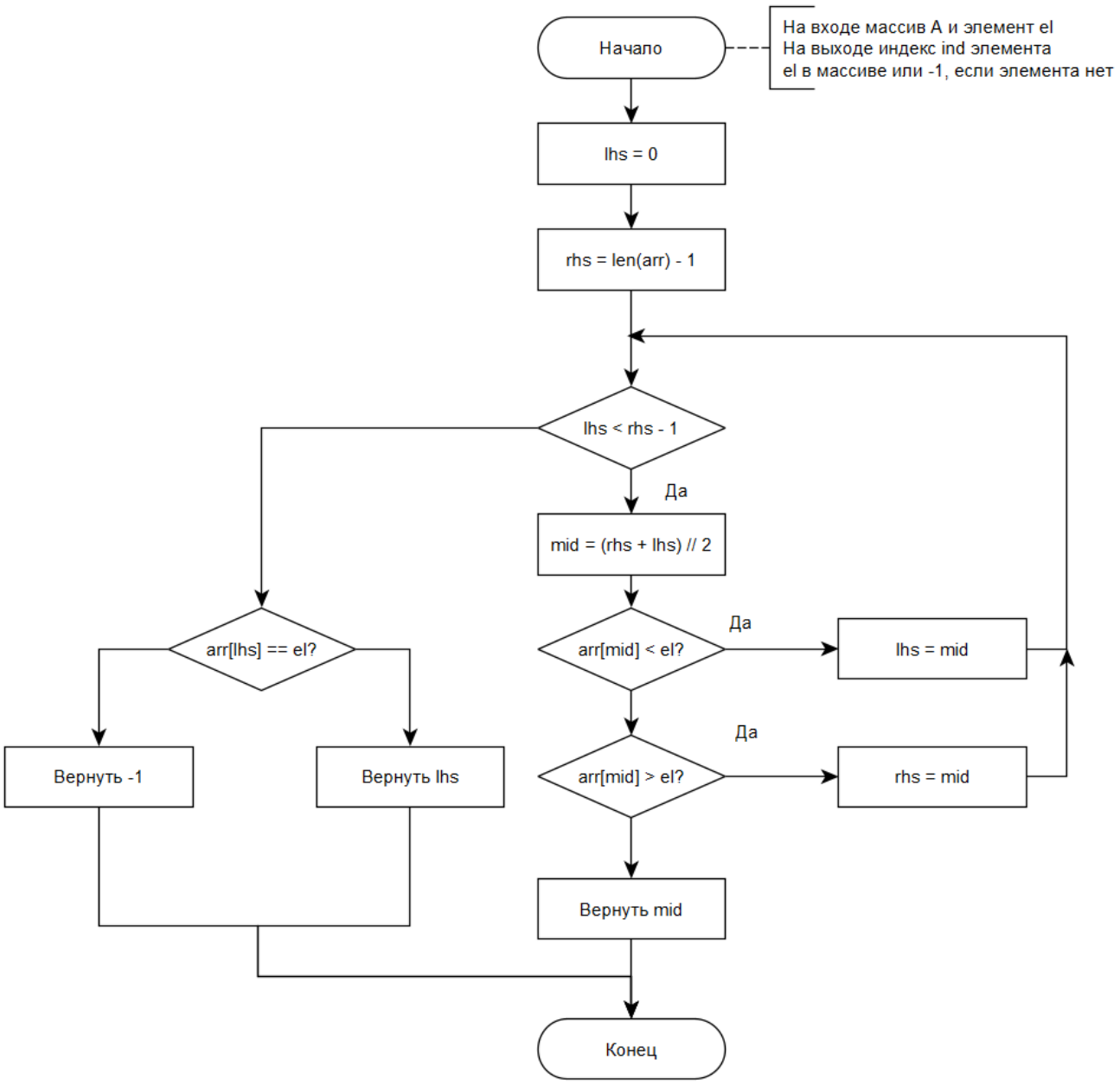

Рисунок 2 – Описание алгоритма бинарного поиска в массиве

2.2 Вывод

В этом разделе на основе теоретических аспектов были представлены описания алгоритма линейного поиска в массиве и алгоритма бинарного поиска в массиве.

3 Технологическая часть

В этом разделе обоснованы средства реализации, а так же представлены реализации алгоритмов и функциональные тесты.

3.1 Средства реализации

Для реализации алгоритмов в этой лабораторной работы был выбран язык *Python*[2], потому что в нем нет автоматического сборщика мусора.

3.2 Реализация алгоритмов

В листингах 1 - 2 представлены коды написанных алгоритмов.

Листинг 1 – Алгоритм линейного поиска в массиве

```
def linear_search(arr, x):
    if len(arr) == 0:
        return -1, 0

    compares = 0
    for ind, i in enumerate(arr):
        compares += 1
        if i == x:
            return ind + 1, compares

    return -1, compares
```

Листинг 2 – Алгоритм бинарного поиска в массиве

```
def binary_search(arr, x):
    if len(arr) == 0:
        return -1, 0

    compares = 0
    lhs = 0
    rhs = len(arr)
    while lhs < rhs - 1:
        compares += 1
        mid = (rhs + lhs) // 2
        if arr[mid] == x:
            return mid + 1, compares
        elif arr[mid] < x:
            lhs = mid
        else:
            rhs = mid

    compares += 1
    if arr[lhs] != x:
        return -1, compares

    return lhs + 1, compares
```

3.3 Функциональные тесты

В таблице 1 приведены функциональные тесты, на которых тестировалась программа.

Таблица 1 – Функциональные тесты, на которых были протестированы алгоритмы

Массив	Искомый элемент	Линейный поиск	Бинарный поиск
[ ]	1	-1	-1
[1]	2	-1	-1
[1, 2, 3, 4, 5]	0	-1	-1
[1, 2, 3, 4, 5]	1	1	1
[1, 2, 3, 4, 5]	2	2	2
[1, 2, 3, 4, 5]	3	3	3
[1, 2, 3, 4, 5]	4	4	4
[1, 2, 3, 4, 5]	5	5	5
[1, 2, 3, 4, 5]	6	-1	-1
[1, 2, 3, 4]	4	4	4
[1, 2, 3, 4]	1	1	1
[1, 2, 3, 4]	0	-1	-1

3.4 Вывод

В этом разделе на основе описаний алгоритмов были написаны и представлены коды алгоритмов. Помимо кодов представлены функциональные тесты, на которых был протестирован каждый алгоритм. Причем функциональные тесты обеспечивают полное покрытие кода[5].

4 Исследовательская часть

В этом разделе приведен сравнительный анализ алгоритмов по количеству операций сравнения, необходимых для того, чтобы найти искомый элемент в массиве.

4.1 Технические характеристики ЭВМ

Все замеры проводились на ЭВМ, характеристики которой приведены ниже:

- Процессор – 12th Gen Intel(R) Core(TM) i5-12450H 2.00 ГГц
- Оперативная память – 16,0 ГБ
- Тип системы – 64-разрядная операционная система, процессор x64
- Операционная система – Windows 11
- Версия ОС – 23H2

4.2 Сравнение алгоритмов

Размер массива, на котором проводилось исследование равен 1028. Предварительно массив был заполнен случайным образом с помощью библиотеки *random*[3]. Графики построены с помощью библиотеки *matplotlib*[4]. На рисунках 3 - 4 представлена зависимость количества сравнений от расположения искомого элемента для алгоритмов линейного и бинарного поиска. На рисунке 5 приведена отсортированная гистограмма из рисунка 4.

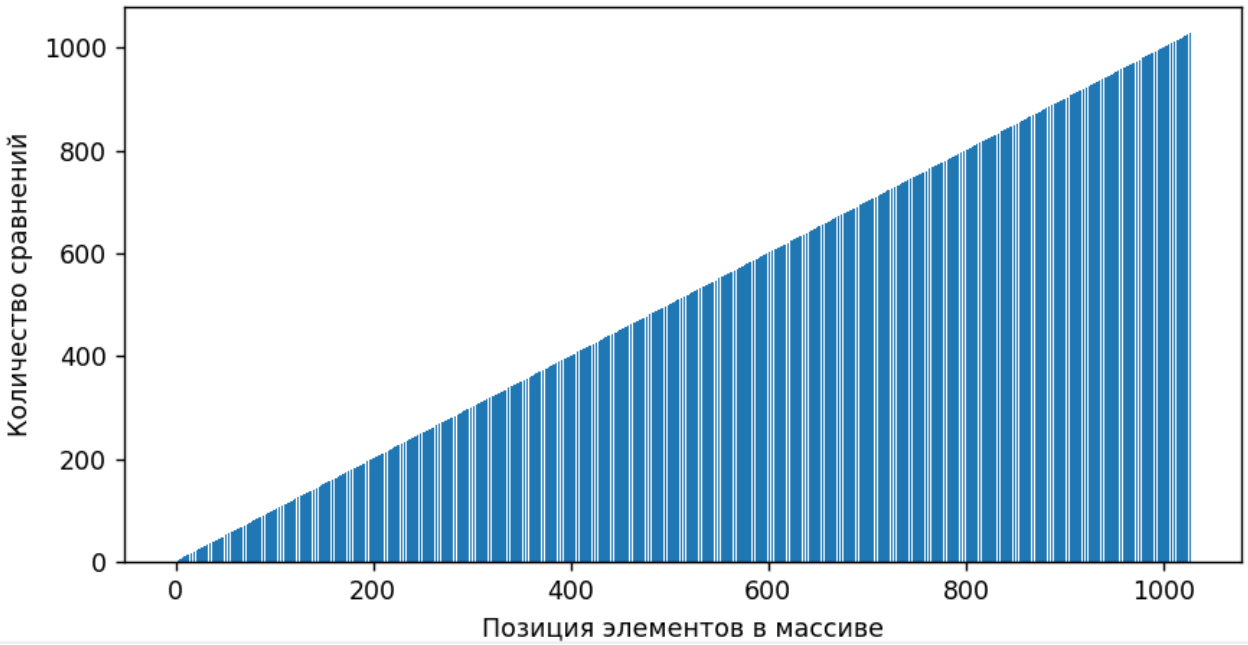

Рисунок 3 – Алгоритм линейного поиска

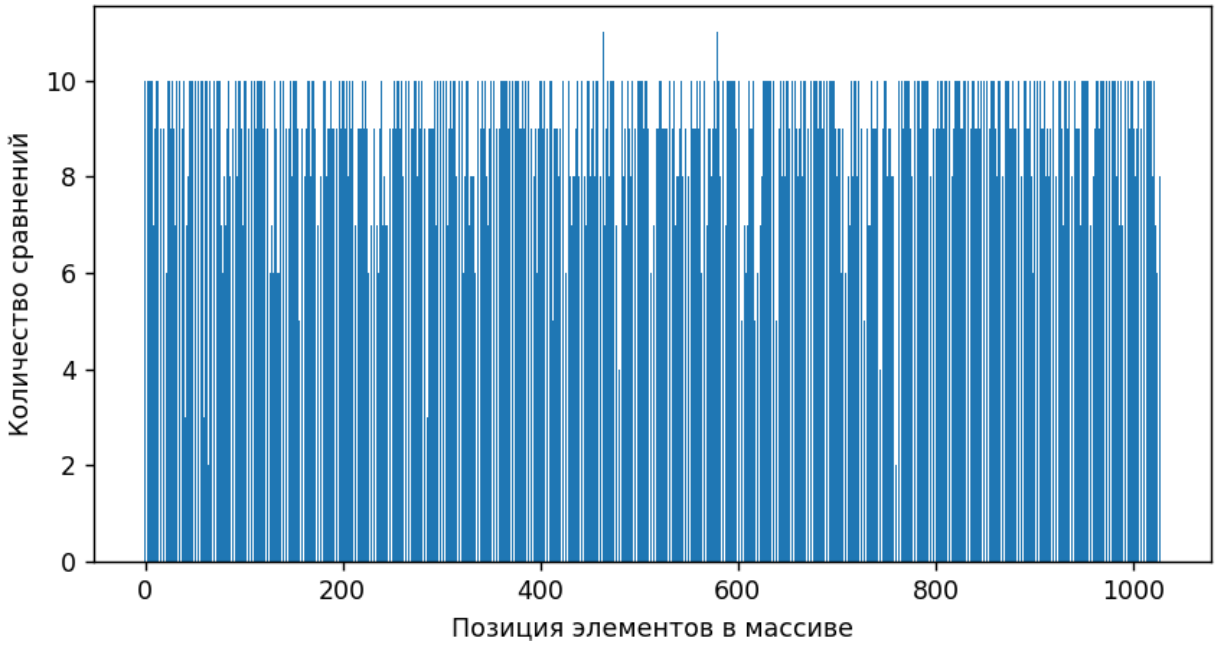

Рисунок 4 – Алгоритм бинарного поиска

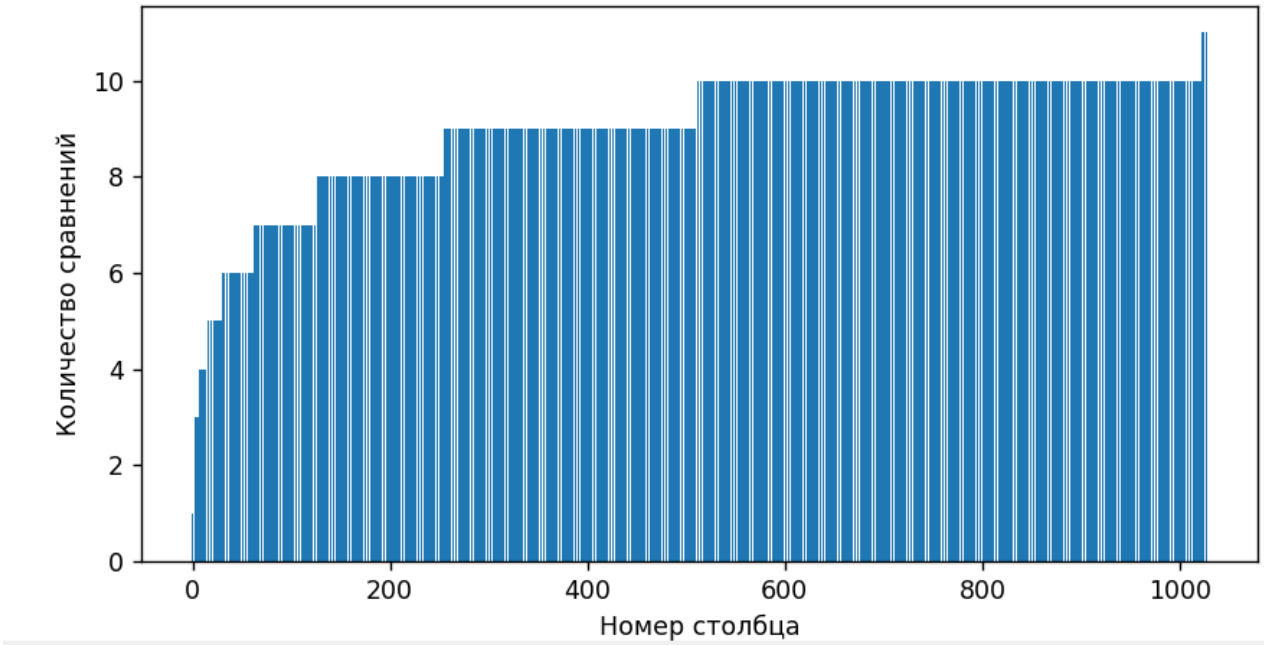

Рисунок 5 – Алгоритм бинарного поиска с сортировкой значений гистограммы

4.3 Вывод

В этом разделе приведен сравнительный анализ алгоритмов. Исследована зависимость количества сравнений, необходимых, чтобы найти элемент в массиве, от расположения элемента в массиве. Оказалось, что в линейном алгоритме количество сравнений изменяется прямопропорционально индексу расположения элемента в массиве. Так же линейный поиск проигрывает бинарному поиску в количестве операций сравнения. Из рисунка 5 видно, что количество операций сравнений не превышает величину равную логарифму от размера массива.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы поставленная цель была достигнута, а также были решены следующие задачи:

- изучены теоретические аспекты алгоритмов поиска в массиве;
- реализованы алгоритмы на языке *Python*;
- проведено сравнение алгоритмов по количеству операций сравнения, необходимых для того, чтобы найти искомый элемент в массиве:
  - в алгоритме с линейным поиском эта величина прямопропорциональна индексу искомого элемента в массиве;
  - количество операций сравнения меньше в бинарном поиске;
  - количество операций сравнения не превышает логарифма от размера массива;
- описаны и обоснованы полученные результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Никлаус Вирт Алгоритмы и структуры данных. Новая версия для Оберона. Никлаус Вирт. — М.: ДМК Пресс, 2016 — 272 с.
2. Welcome to Python – https://www.python.org
3. Библиотека random – https://docs.python.org/3/library/random.html
4. Библиотека matplotlib – https://matplotlib.org/
5. Утилита coverage – https://coverage.readthedocs.io/en/latest/