Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №9
дисциплины «Алгоритмизация»

Выполнил:
Кожуховский Виктор Андреевич
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

Написал программу бинарного и линейного поиска элемента в массиве, автоматического заполнения массива, расчёта тысячи точек, показывающих время поиска элемента в массиве в худшем и среднем случае, вывода графиков сравнения бинарного, линейного и встроенного бинарного поиска, составленных из этих точек:

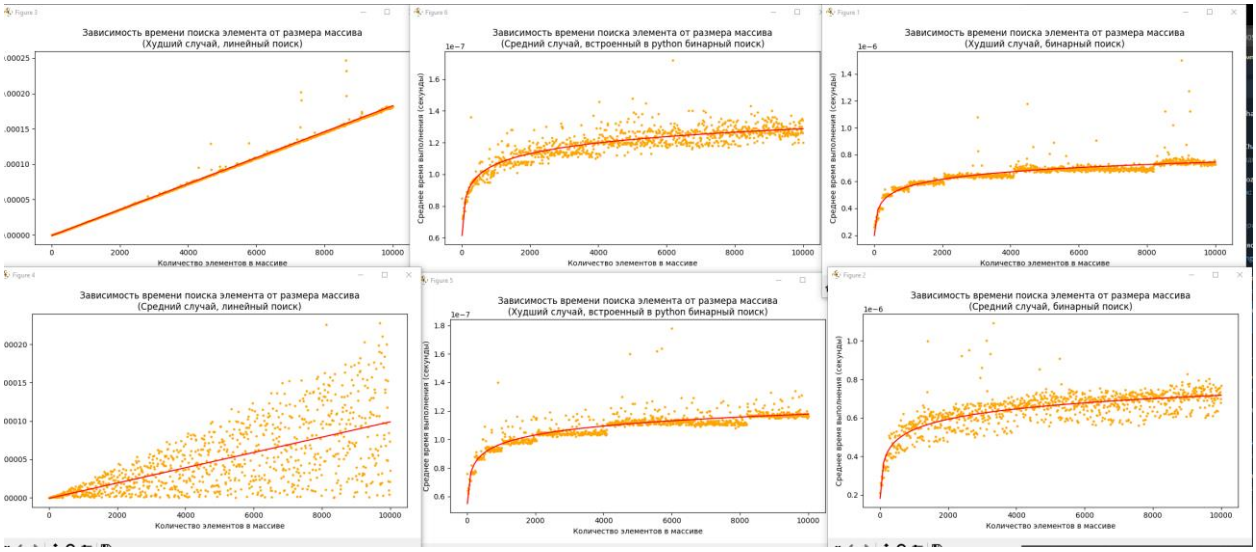

Рисунок 1. Графики времени поиска элемента в массиве в худшем и среднем случае

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import timeit
import random
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
import bisect


amount_of_dots = 100  # Количество точек
aod = (amount_of_dots + 1) * 10
worst_time = {}
median_time = {}
graph_stuff = [i for i in range(10, aod, 10)]
xlabel = "Количество элементов в массиве"
ylabel = "Среднее время выполнения (секунды)"


def bin_search(search_list, value):
    left = -1
    right = len(search_list)
    while left < right - 1:
        mid = (left + right) // 2
        if search_list[mid] < value:
            left = mid
        else:
            right = mid
    return right


def search(search_list, value):      You, 6 days ago • Need to fix graphs
    for index, item in enumerate(search_list):
        if item == value:
            return index
    return -1


def fill_list(num_of_elements):
    a = [random.randint(0, 100000) for _ in range(num_of_elements)]
    return a


def logarithmic_model(x, a, b):
    return a * np.log(x) + b


def lsm(name, time, graph_num, log):
    plt.figure(graph_num).set_figwidth(8)
    plt.title(
        f"Зависимость времени поиска элемента от размера массива\n({name})")
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.tight_layout()
    plt.grid(False)

    x_data = np.array(graph_stuff)
    y_data = np.array(list(time.values()))

    if log:
        params, _ = curve_fit(logarithmic_model, x_data, y_data)

        a_fit, b_fit = params
        print(f"Коэффициенты уравнения ({name}): a = {a_fit}, b = {b_fit}")
        # print(f"Корреляция {name}:", np.corrcoef(
        #     graph_stuff, list(time.values()))[0, 1]) Не знаю, надо ли корреляцию, так что оставлю тут.

        x_fit = np.linspace(min(x_data), max(x_data), 100)
        y_fit = logarithmic_model(x_fit, *params)

        plt.plot(x_fit, y_fit, "r-", label="Quadratic Fit")
    else:
        A = np.vstack([graph_stuff, np.ones(len(graph_stuff))]).T
        alpha = np.dot(
            (np.dot(np.linalg.inv(np.dot(A.T, A)), A.T)
             ), np.array(list(time.values()))
        )
        plt.plot(graph_stuff, alpha[0] *
                 np.array(list(graph_stuff)) + alpha[1], "r")

        formatted_alpha = [format(a, ".10f") for a in alpha]
        print(
            f"Коэффициенты прямой {name}: a =",
            formatted_alpha[0],
            "b =",
            formatted_alpha[1],
        )
        # print(f"Корреляция {name}:", np.corrcoef(
        #     graph_stuff, list(time.values()))[0, 1])

    plt.scatter(graph_stuff, time.values(), s=5, c="orange")


def results(name, func, graph_index, log):
    for i in range(10, aod, 10):
        a = fill_list(i)
        worst_time[i] = (timeit.timeit(
            lambda: func(a, 10000000), number=100)) / 100
        t = int(random.randint(1, i - 1))
        median_time[i] = timeit.timeit(lambda: func(a, a[t]),
                                       number=100) / 100

    lsm("Худший случай, " + name, worst_time, graph_index, log)
    lsm("Средний случай, " + name, median_time, graph_index + 1, log)


if __name__ == "__main__":
    results("бинарный поиск", bin_search, 0, True)
    results("линейный поиск", search, 2, False)
    results("встроенный в python бинарный поиск", bisect.bisect_left, 4, True)

    plt.show()
```

Рисунок 2. Код программы

Вывод: в результате выполнения лабораторной работы был изучен алгоритм бинарного поиска и проведено исследование зависимости времени поиска от количества элементов в массиве, показавшее что зависимость время поиска линейно увеличивается с добавлением элементов в массив.