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»
Вариант 5.

Выполнила:
Михеева Елена Александровна
2 курс, группа ИВТ-б-з-20-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Изучение алгоритмов бинарного и линейного поисков.

Порядок выполнения работы.

1. Была написана программа для изучения времени работы бинарного поиска в зависимости от количества элементов в массиве.

```python
import timeit
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit


def binary_search(arr, target):
    low, high = 0, len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        mid_value = arr[mid]

        if mid_value == target:
            return mid
        elif mid_value < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1


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


def find_coeffs_bin(x, time):
    params, covariance = curve_fit(log_n, np.array(x), np.array(time))
    a, b = params
    return a, b


def generate_data(size):
    arr = list(range(size))
    target_avg = arr[size // 2]  # Искомый элемент для среднего случая
    target_worst = size  # Значение, которого нет в массиве для худшего случая
    return arr, target_avg, target_worst


def measure_time(func, *args):
    return timeit.timeit(lambda: func(*args), number=100) / 100


def analyze_binary_search(start, end, step):
    sizes = list(range(start, end + 1, step))
    times_avg = []
    times_worst = []

    for size in sizes:
        arr, target_avg, target_worst = generate_data(size)

        time_avg = measure_time(binary_search, arr, target_avg)
        time_worst = measure_time(binary_search, arr, target_worst)

        times_avg.append(time_avg)
        times_worst.append(time_worst)

    return sizes, times_avg, times_worst


def plot_results(sizes, times_avg, times_worst):
    plt.scatter(sizes, times_avg, label='Средний случай', s=10)
    plt.scatter(sizes, times_worst, label='Худший случай', s=10)

    a_avg, b_avg = find_coeffs_bin(sizes, times_avg)
    a_worst, b_worst = find_coeffs_bin(sizes, times_worst)

    y_fit_avg = log_n(np.array(sizes), a_avg, b_avg)
    y_fit_worst = log_n(np.array(sizes), a_worst, b_worst)

    plt.plot(sizes, y_fit_avg, label=f'Для среднего: {a_avg:.9f} * log(x) + \
             {b_avg:.9f}', color='red')
    plt.plot(sizes, y_fit_worst, label=f'Для худшего: {a_worst:.9f} * log(x) +\
             {b_worst:.9f}', color='blue')

    plt.xlabel('Размер массива')
    plt.ylabel('Время выполнения (секунды)')
    plt.title('Анализ времени выполнения бинарного поиска')
    plt.legend()
    plt.show()


if __name__ == '__main__':
    start = 100
    end = 10000
    step = 30

    sizes, times_avg, times_worst = analyze_binary_search(start, end, step)
    plot_results(sizes, times_avg, times_worst)
```

Рисунок 1. Программа binarySearch.py

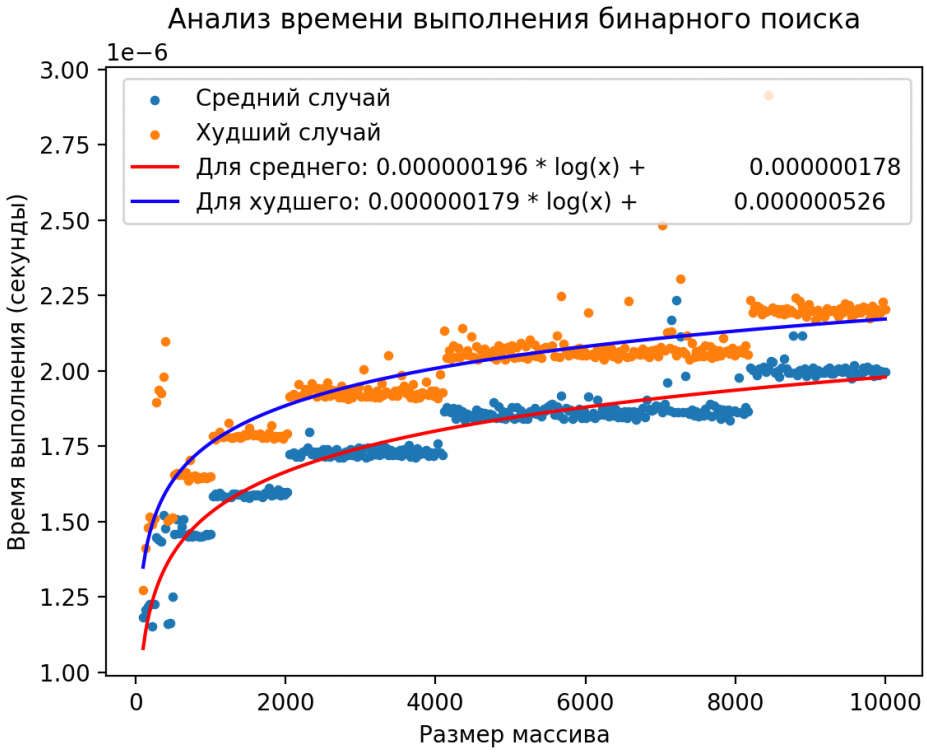

Рисунок 2. Результат выполнения программы

2. Была написана программа для изучения времени работы линейного поиска в зависимости от количества элементов в массиве.

```python
import timeit
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit


def binary_search(arr, target):
    low, high = 0, len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        mid_value = arr[mid]

        if mid_value == target:
            return mid
        elif mid_value < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1


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


def find_coeffs_bin(x, time):
    params, covariance = curve_fit(log_n, np.array(x), np.array(time))
    a, b = params
    return a, b


def generate_data(size):
    arr = list(range(size))
    target_avg = arr[size // 2]  # Искомый элемент для среднего случая
    target_worst = size  # Значение, которого нет в массиве для худшего случая
    return arr, target_avg, target_worst


def measure_time(func, *args):
    return timeit.timeit(lambda: func(*args), number=100) / 100


def analyze_binary_search(start, end, step):
    sizes = list(range(start, end + 1, step))
    times_avg = []
    times_worst = []
```

```python
def analyze_linear_search(start, end, step):
    sizes = list(range(start, end + 1, step))
    times_avg = []
    times_worst = []

    for size in sizes:
        arr, target_avg, target_worst = generate_data(size)

        time_avg = measure_time(linear_search, arr, target_avg)
        time_worst = measure_time(linear_search, arr, target_worst)

        times_avg.append(time_avg)
        times_worst.append(time_worst)

    return sizes, times_avg, times_worst


def plot_results(sizes, times_avg, times_worst):
    plt.scatter(sizes, times_avg, label='Средний случай', s=10)
    plt.scatter(sizes, times_worst, label='Худший случай', s=10)

    a_avg, b_avg = find_coeffs_line(sizes, times_avg)
    a_worst, b_worst = find_coeffs_line(sizes, times_worst)

    y_fit_avg = a_avg * np.array(sizes) + b_avg
    y_fit_worst = a_worst * np.array(sizes) + b_worst

    plt.plot(sizes, y_fit_avg, label=f'Для среднего: \
    {a_avg:.9f} * x + {b_avg:.9f}', color='red')
    plt.plot(sizes, y_fit_worst, label=f'Для худшего: \
    {a_worst:.9f} * x + {b_worst:.9f}', color='blue')

    plt.xlabel('Размер массива')
    plt.ylabel('Время выполнения (секунды)')
    plt.title('Анализ времени выполнения линейного поиска')
    plt.legend()
    plt.show()


if __name__ == '__main__':
    start = 100
    end = 10000
    step = 50

    sizes, times_avg, times_worst = analyze_linear_search(start, end, step)
    plot_results(sizes, times_avg, times_worst)
```

Рисунок 3. Программа linearSearch.py

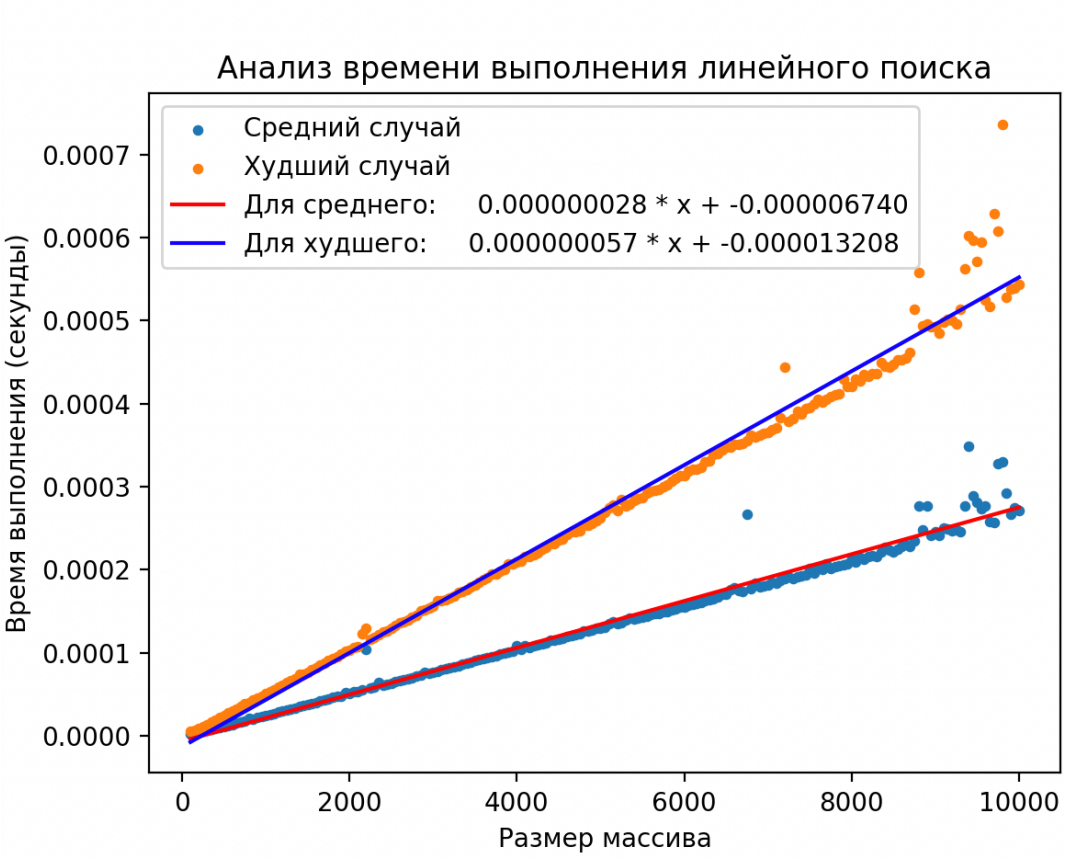

Рисунок 4. Результат выполнения программы.

Вывод: в ходе выполнения лабораторной работы было выяснено, что бинарный поиск O(log n) является более эффективным, чем линейный O(n).