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 29

Выполнил:
Саенко Андрей Максимович
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 Р.А., канд. технических наук, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Алгоритм бинарного поиска

Порядок выполнения работы:

Написана программа, содержащая функции, реализующие алгоритмы линейного и бинарного поиска, определяющая время их работы для массивов разного размера и выводящая графики зависимости времени работы от размера массива.

Код программы:

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


import bisect
import random
import numpy as np
import timeit
import matplotlib.pyplot as plt
from math import sqrt, log
from scipy.optimize import curve_fit


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 correlation(array_of_values_x,array_of_values_y):
    sigma_first = 0
    sigma_second = 0
    sigma_third = 0
    average_value_x = 0
    average_value_y = 0
    sum_x = 0
    sum_y = 0

    for i in range(len(array_of_values_x)):
        sum_x += array_of_values_x[i]
        sum_y += array_of_values_y[i]

    average_value_x = sum_x/len(array_of_values_x)
    average_value_y = sum_y/len(array_of_values_y)
```

```
    for i in range(len(array_of_values_x)):
        sigma_first += ((array_of_values_x[i] - average_value_x)*
                    (array_of_values_y[i] - average_value_y))
        sigma_second += (array_of_values_x[i]-average_value_x)**2
        sigma_third += (array_of_values_y[i]-average_value_y)**2

    pair_correlation_coefficient = (sigma_first/(sqrt(sigma_second)*
                                        sqrt(sigma_third)))
    return pair_correlation_coefficient


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

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == k:
            return mid
        elif arr[mid] < k:
            low = mid + 1
        else:
            high = mid - 1

    return -1


def linear_search(array,element):
    for i in range(0,len(array)):
        if array[i] == element:
            return 1
    return 0


if __name__ == "__main__":
    x = []

    linear_search_bad = []
    linear_search_avr = []

    bin_search_bad = []
    bin_search_avr = []

    bisect_search_bad = []
    bisect_search_avr = []

    for i in range(1,1001):
        arr = [0 for i in range(0,i)]
        x.append(i)

        for j in range(0,len(arr)):
            arr[j] = random.randint(500,1000)
```

```
    search_time = (timeit.timeit(lambda: linear_search(arr,1),
                                 number=50))/50
    print("Время поиска в массиве из ",i,
        " элементов в худшем случае: ", search_time)
    linear_search_bad.append(search_time)

    search_time = (timeit.timeit(lambda: linear_search(
        arr,arr[round(len(arr)/2)]), number=50))/50
    print("Время поиска в массиве из ",i,
        " элементов в среднем случае: ",search_time,"\n")
    linear_search_avr.append(search_time)

    search_time = (timeit.timeit(lambda: binary_search(arr,1),
                                 number=50))/50
    print("Время бинарного поиска в массиве из ",i,
        " элементов в худшем случае: ", search_time)
    bin_search_bad.append(search_time)

    search_time = (timeit.timeit(lambda: binary_search(
        arr,arr[round(len(arr)/2)]), number=50))/50
    print("Время бинарного поиска в массиве из ",i,
        " элементов в среднем случае: ",search_time,"\n")
    bin_search_avr.append(search_time)

    search_time = (timeit.timeit(lambda: bisect.bisect_left(arr,1),
                                 number=50))/50
    print("Время бинарного поиска с помощью bisect в массиве из ",i,
        " элементов в худшем случае: ", search_time)
    bisect_search_bad.append(search_time)

    search_time = (timeit.timeit(lambda: bisect.bisect_left(
        arr,arr[round(len(arr)/2)]), number=50))/50
    print("Время поиска с помощью bisect в массиве из ",i,
        " элементов в среднем случае: ",search_time,"\n")
    bisect_search_avr.append(search_time)

bin_first_coef_bad, bin_second_coef_bad = (
    find_coeffs_bin(x, bin_search_bad))
bin_first_coef_avr, bin_second_coef_avr = (
    find_coeffs_bin(x, bin_search_avr))

bisect_first_coef_bad, bisect_second_coef_bad = (
    find_coeffs_bin(x, bisect_search_bad))
bisect_first_coef_avr, bisect_second_coef_avr = (
    find_coeffs_bin(x, bisect_search_avr))

func_bin_bad = []
for i in range(0,1000):
    func_bin_bad.append((bin_first_coef_bad*log(x[i]) +
                         bin_second_coef_bad))
```

```
func_bin_avr = []
for i in range(0,1000):
    func_bin_avr.append((bin_first_coef_avr*log(x[i]) +
                         bin_second_coef_avr))

func_bisect_bad = []
for i in range(0,1000):
    func_bisect_bad.append((bisect_first_coef_bad*log(x[i]) +
                            bisect_second_coef_bad))

func_bisect_avr = []
for i in range(0,1000):
    func_bisect_avr.append((bisect_first_coef_avr*log(x[i]) +
                            bisect_second_coef_avr))

sum_search_time_bad = sum(linear_search_bad)
sum_amt_of_el = sum(x)
sum_of_sqr_amt_of_el = 0
sum_mltp_amt_time = 0
bn = len(x)

for i in x:
    sum_of_sqr_amt_of_el += i*i

for i in range(0,len(linear_search_bad)):
    sum_mltp_amt_time += x[i]*linear_search_bad[i]

matrix_bad = np.array([[sum_of_sqr_amt_of_el, sum_amt_of_el],
                       [sum_amt_of_el, bn]])

det_bad = np.linalg.det(matrix_bad)

first_mt_bad = np.array([[sum_mltp_amt_time, sum_amt_of_el],
                         [sum_search_time_bad, bn]])
first_det_bad=np.linalg.det(first_mt_bad)

second_mt_bad = np.array([[sum_of_sqr_amt_of_el, sum_mltp_amt_time],
                          [sum_amt_of_el, sum_search_time_bad]])
second_det_bad = np.linalg.det(second_mt_bad)
first_coefficient_bad=first_det_bad/det_bad
second_coefficient_bad = second_det_bad/det_bad

func_bad = []
for i in range(1,1001):
    func_bad.append(first_coefficient_bad*(i)+second_coefficient_bad)

sum_search_time_avr = sum(linear_search_avr)
sum_mltp_amt_time_avr = 0

for i in range(0,len(linear_search_bad)):
    sum_mltp_amt_time_avr += x[i]*linear_search_avr[i]
matrix_avr = np.array([[sum_of_sqr_amt_of_el, sum_amt_of_el],
```

```
                                [sum_amt_of_el, bn]])

det_avr = np.linalg.det(matrix_avr)
first_mt_avr = np.array([[sum_mltp_amt_time_avr, sum_amt_of_el],
                                    [sum_search_time_avr, bn]])
first_det_avr = np.linalg.det(first_mt_avr)
second_mt_avr = np.array([[sum_of_sqr_amt_of_el, sum_mltp_amt_time_avr],
                                      [sum_amt_of_el, sum_search_time_avr]])
second_det_avr = np.linalg.det(second_mt_avr)
first_coefficient_avr = first_det_avr/det_avr
second_coefficient_avr = second_det_avr/det_avr

func_average = []
for i in range(1,1001):
    func_average.append(first_coefficient_avr*i +
                        second_coefficient_avr)

plt.figure(1)
plt.title("Время поиска элемента в худшем случае")
plt.plot(x,func_bad,color='red',linewidth=4)
plt.scatter(x, linear_search_bad,s=3)
plt.xlabel('Размер массива\n Коэффициент парной корреляции равен:'+
        str(correlation(x,linear_search_bad)))
plt.legend(['y='+str(first_coefficient_bad)+"x+("+
            str(second_coefficient_bad)+")"])
plt.ylabel("Время поиска элемента в массиве")

plt.figure(2)
plt.title("Время поиска элемента в среднем случае")
plt.plot(x,func_average,color='red',linewidth=4)
plt.scatter(x,linear_search_avr,s=3)
plt.xlabel('Размер массива\n Коэффициент парной корреляции равен:'+
        str(correlation(x,linear_search_avr)))
plt.legend(['y='+str(first_coefficient_avr)+"x+("+
            str(second_coefficient_avr)+")"])
plt.ylabel("Время поиска элемента в массиве")

plt.figure(3)
plt.title("Время бинарного поиска\n элемента в худшем случае")
plt.plot(x,func_bin_bad,color='red',linewidth=4)
plt.scatter(x,bin_search_bad,s=3)
plt.xlabel('Размер массива')
plt.legend([f"y = {bin_first_coef_bad}*log(x)" +
            f" + ({bin_second_coef_bad})"])
plt.ylabel("Время поиска элемента в массиве")

plt.figure(4)
plt.title("Время бинарного поиска\n элемента в среднем случае")
plt.plot(x,func_bin_avr, color='red',linewidth=4)
plt.scatter(x,bin_search_avr,s=3)
plt.xlabel('Размер массива')
plt.legend([f"y = {bin_first_coef_avr}*log(x)" +
```

```
        f" + ({bin_second_coef_avr})"])
plt.ylabel("Время поиска элемента в массиве")

plt.figure(5)
plt.title("Время бинарного поиска элемента с\n"+
        " помощью bisect в худшем случае")
plt.plot(x,func_bisect_bad,color='red', linewidth=4)
plt.scatter(x,bisect_search_bad, s=3)
plt.xlabel('Размер массива')
plt.legend([f"y = {bisect_first_coef_bad}*log(x) " +
        f"+ ({bisect_second_coef_bad})"])
plt.ylabel("Время поиска элемента в массиве")

plt.figure(6)
plt.title("Время бинарного поиска элемента с\n"+
        " помощью bisect в среднем случае")
plt.plot(x,func_bisect_avr, color='red',linewidth=4)
plt.scatter(x,bisect_search_avr,s=3)
plt.xlabel('Размер массива')
plt.legend([f"y = {bisect_first_coef_avr}*log(x)" +
        f" + ({bisect_second_coef_avr})"])
plt.ylabel("Время поиска элемента в массиве")

plt.show()
```

Результат работы программы:

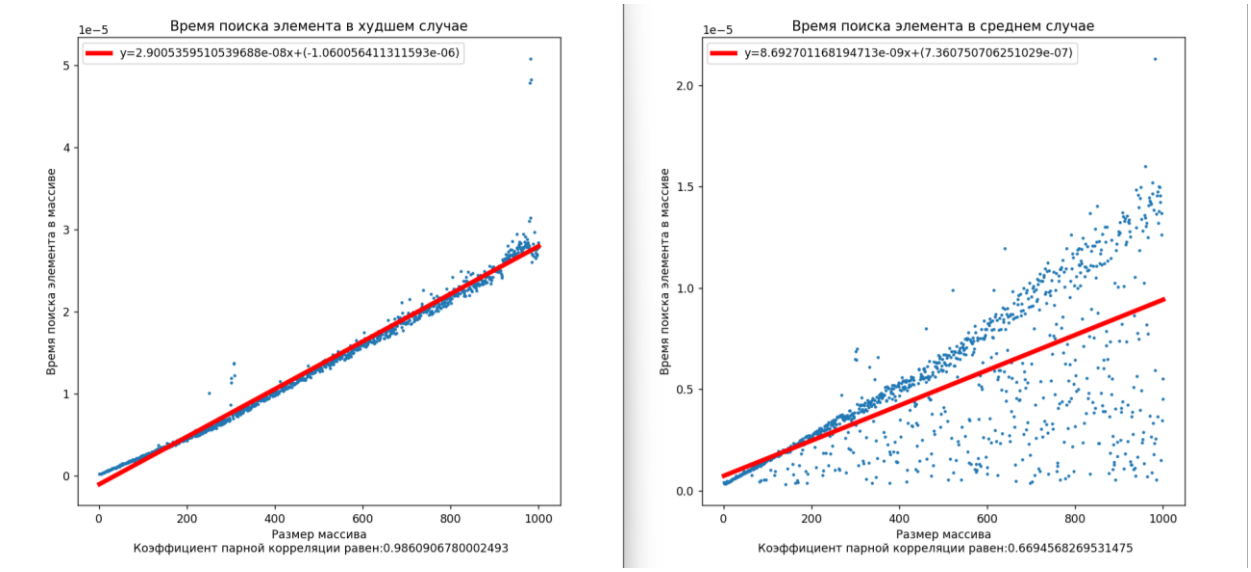


Рисунок 1 – Скорость работы функции, использующей алгоритм линейного поиска

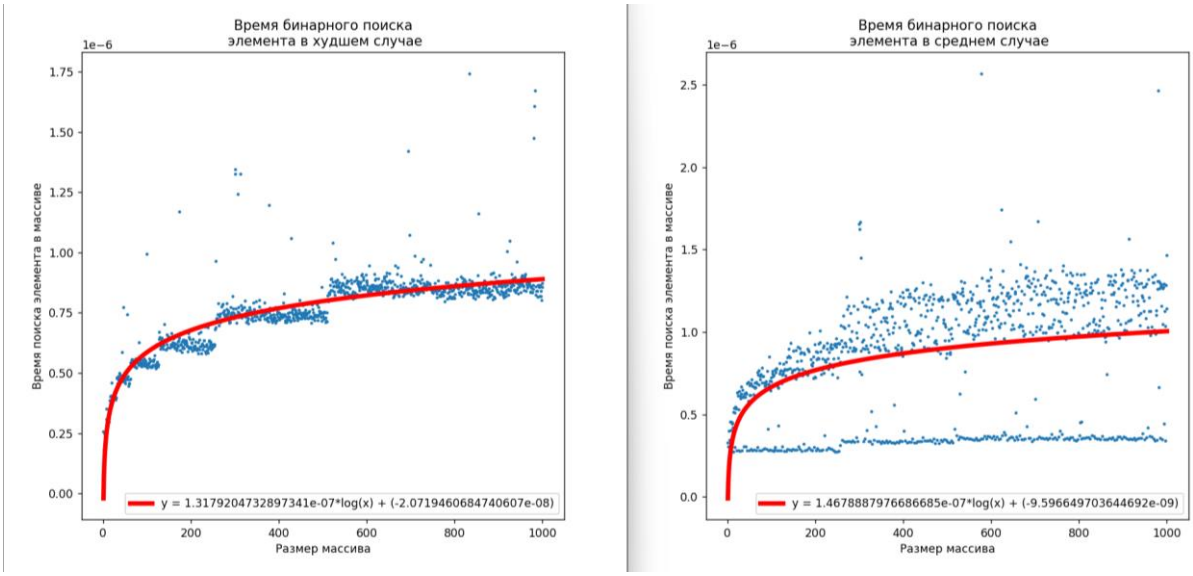


Рисунок 2 – Скорость работы функции, использующей алгоритм бинарного поиска

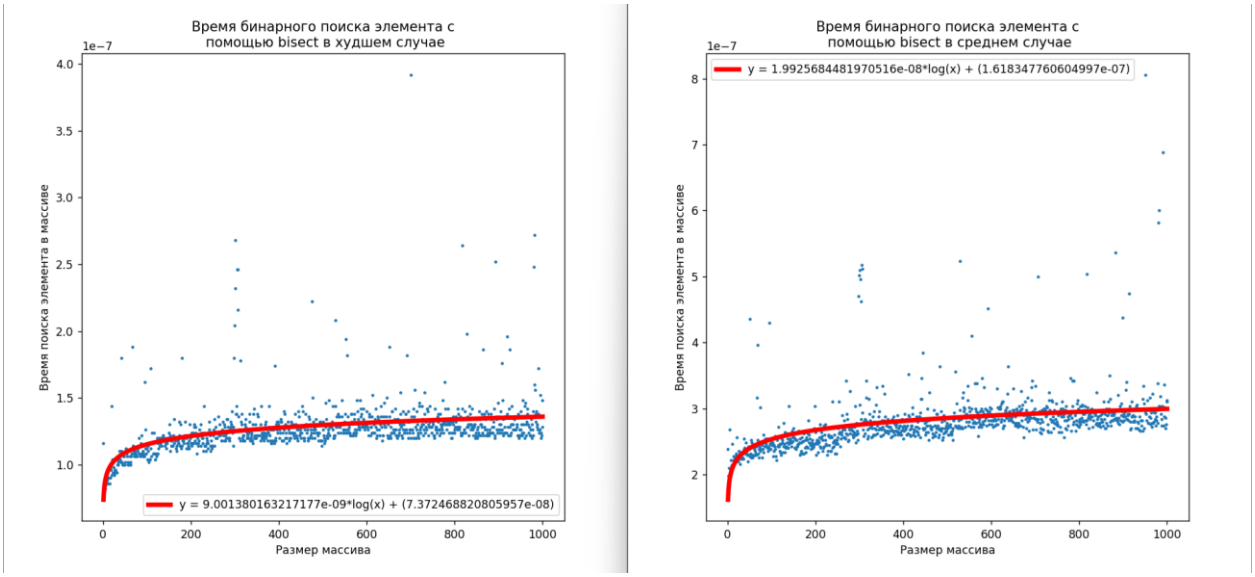


Рисунок 3 – Скорость работы встроенной функции, использующей алгоритм бинарного поиска

## Вывод

Алгоритм бинарного поиска находит элемент значительно быстрее алгоритма линейного поиска, так как ему требуется меньше итераций. Скорость работы данного алгоритма равна O(log(n)). В языке Python есть модуль bisect, позволяющий использовать бинарный поиск.