

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.

```
1 import timeit
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy.optimize import curve_fit
5
6 def binary_search(arr, target):
7     low, high = 0, len(arr) - 1
8
9     while low <= high:
10         mid = (low + high) // 2
11         mid_value = arr[mid]
12
13         if mid_value == target:
14             return mid
15         elif mid_value < target:
16             low = mid + 1
17         else:
18             high = mid - 1
19
20     return -1
21
22 def log_n(x, a, b):
23     return a * np.log(x) + b
24
25 def find_coeffs_bin(x, time):
26     params, covariance = curve_fit(log_n, np.array(x), np.array(time))
27     a, b = params
28     return a, b
29
30 def generate_data(size):
31     arr = list(range(size))
32     target_avg = arr[size // 2] # Искомый элемент для среднего случая
33     target_worst = size # Значение, которого нет в массиве для худшего случая
34     return arr, target_avg, target_worst
35
36 def measure_time(func, *args):
37     return timeit.timeit(lambda: func(*args), number=100) / 100
38
39 def analyze_binary_search(start, end, step):
40     sizes = list(range(start, end + 1, step))
41     times_avg = []
42     times_worst = []
43
44     for size in sizes:
45         arr, target_avg, target_worst = generate_data(size)
46         time_avg = measure_time(binary_search, arr, target_avg)
47         time_worst = measure_time(binary_search, arr, target_worst)
48         times_avg.append(time_avg)
49         times_worst.append(time_worst)
50
51     return sizes, times_avg, times_worst
52
53 def plot_results(sizes, times_avg, times_worst):
54     plt.scatter(sizes, times_avg, label='Средний случай', s=10)
55     plt.scatter(sizes, times_worst, label='Худший случай', s=10)
56
57     a_avg, b_avg = find_coeffs_bin(sizes, times_avg)
58     a_worst, b_worst = find_coeffs_bin(sizes, times_worst)
59
60     y_fit_avg = log_n(np.array(sizes), a_avg, b_avg)
61     y_fit_worst = log_n(np.array(sizes), a_worst, b_worst)
62
63     plt.plot(sizes, y_fit_avg, label=f'Для среднего: {a_avg:.9f} * log(x) + \n (b_avg:.9f)', color='red')
64     plt.plot(sizes, y_fit_worst, label=f'Для худшего: {a_worst:.9f} * log(x) + \n (b_worst:.9f)', color='blue')
65
66     plt.xlabel('Размер массива')
67     plt.ylabel('Время выполнения (секунды)')
68     plt.title('Анализ времени выполнения бинарного поиска')
69     plt.legend()
70     plt.show()
71
72 if __name__ == '__main__':
73     start = 100
74     end = 10000
75     step = 30
76
77     sizes, times_avg, times_worst = analyze_binary_search(start, end, step)
78     plot_results(sizes, times_avg, times_worst)
```

Рисунок 1. Программа binarySearch.py

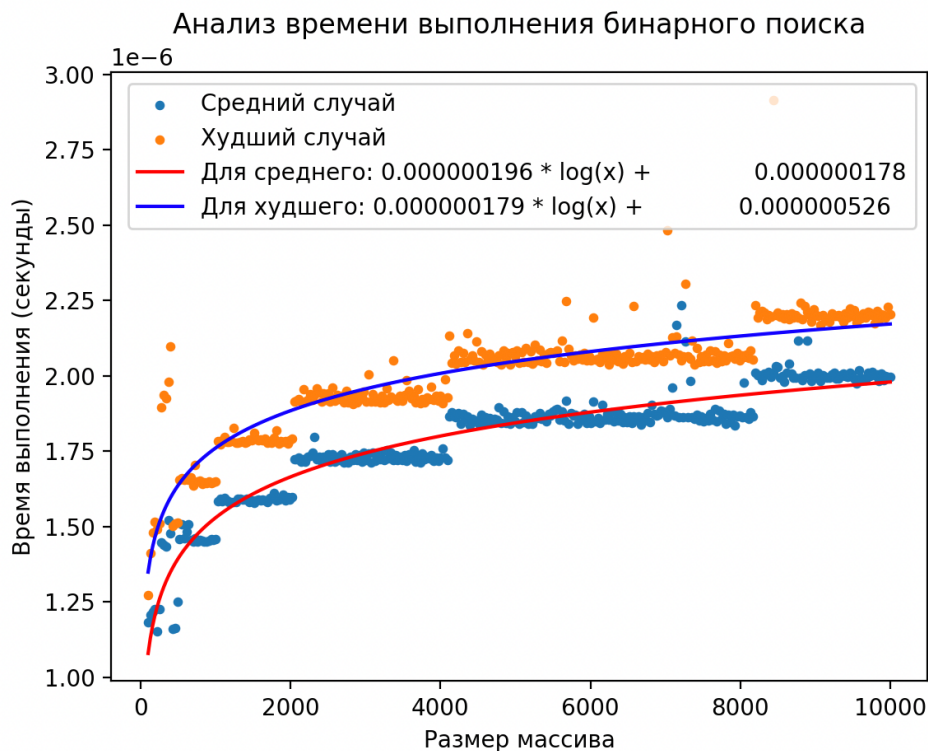


Рисунок 2. Результат выполнения программы

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

```
1 import timeit
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy.optimize import curve_fit
5
6
7 def binary_search(arr, target):
8     low, high = 0, len(arr) - 1
9
10    while low <= high:
11        mid = (low + high) // 2
12        mid_value = arr[mid]
13
14        if mid_value == target:
15            return mid
16        elif mid_value < target:
17            low = mid + 1
18        else:
19            high = mid - 1
20
21    return -1
22
23
24 def log_n(x, a, b):
25     return a * np.log(x) + b
26
27
28 def find_coeffs_bin(x, time):
29     params, covariance = curve_fit(log_n, np.array(x), np.array(time))
30     a, b = params
31     return a, b
32
33
34 def generate_data(size):
35     arr = list(range(size))
36     target_avg = arr[size // 2] # Искомый элемент для среднего случая
37     target_worst = size # Значение, которого нет в массиве для худшего случая
38     return arr, target_avg, target_worst
39
40
41 def measure_time(func, *args):
42     return timeit.timeit(lambda: func(*args), number=100) / 100
43
44
45 def analyze_binary_search(start, end, step):
46     sizes = list(range(start, end + 1, step))
47     times_avg = []
48     times_worst = []
49
50     for size in sizes:
51         arr, target_avg, target_worst = generate_data(size)
52         time_avg = measure_time(binary_search, arr, target_avg)
53         time_worst = measure_time(binary_search, arr, target_worst)
54
55         times_avg.append(time_avg)
56         times_worst.append(time_worst)
57
58     return sizes, times_avg, times_worst
59
60
61 def plot_results(sizes, times_avg, times_worst):
62     plt.scatter(sizes, times_avg, label='Средний случай', s=10)
63     plt.scatter(sizes, times_worst, label='Худший случай', s=10)
64
65     a_avg, b_avg = find_coeffs_bin(sizes, times_avg)
66     a_worst, b_worst = find_coeffs_bin(sizes, times_worst)
67
68     y_fit_avg = a_avg * np.array(sizes) + b_avg
69     y_fit_worst = a_worst * np.array(sizes) + b_worst
70
71     plt.plot(sizes, y_fit_avg, label=f'Для среднего: \
72     {a_avg:.9f} * x + {b_avg:.9f}', color='red')
73     plt.plot(sizes, y_fit_worst, label=f'Для худшего: \
74     {a_worst:.9f} * x + {b_worst:.9f}', color='blue')
75
76     plt.xlabel('Размер массива')
77     plt.ylabel('Время выполнения (секунды)')
78     plt.title('Анализ времени выполнения линейного поиска')
79     plt.legend()
80     plt.show()
81
82
83 if __name__ == '__main__':
84     start = 100
85     end = 10000
86     step = 50
87
88     sizes, times_avg, times_worst = analyze_binary_search(start, end, step)
89     plot_results(sizes, times_avg, times_worst)
90
```

Рисунок 3. Программа linearSearch.py

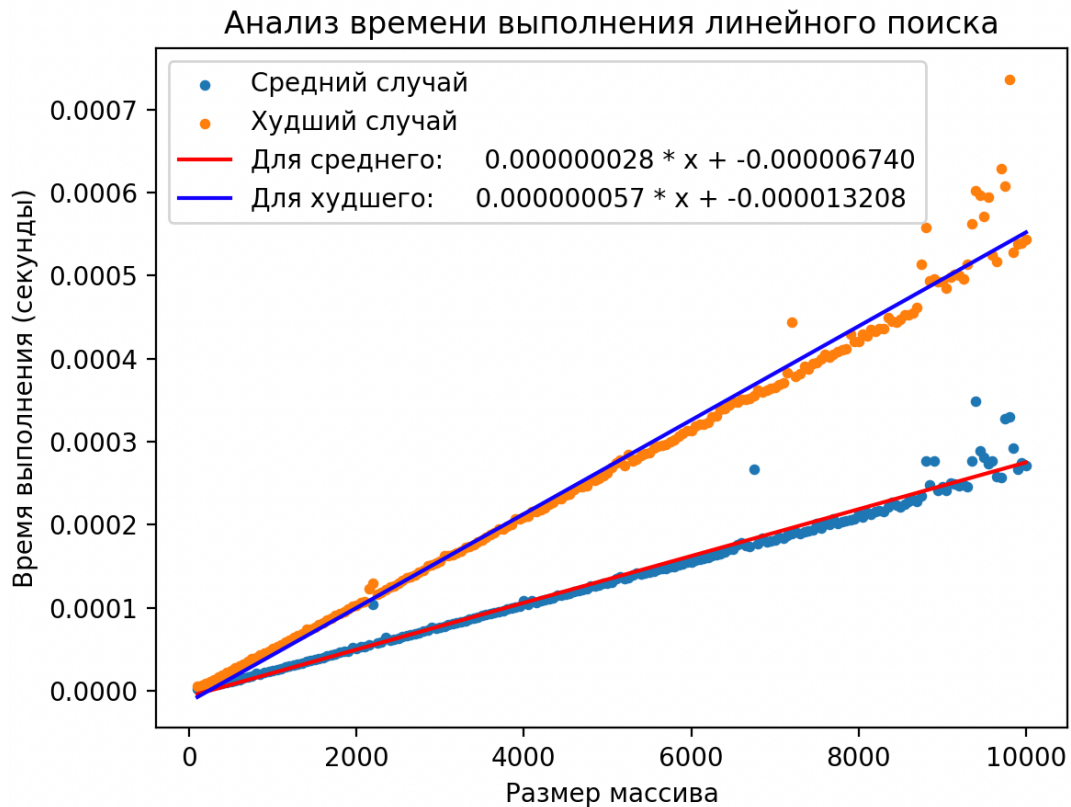


Рисунок 4. Результат выполнения программы.

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