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»

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических
наук, доцент, доцент кафедры
инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

Зависимость времени выполнения функции бинарного поиска от количества элементов в массиве в худшем случае:

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



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

my_setup = """
from __main__ import a
from __main__ import search
from __main__ import first
from __main__ import last
from __main__ import binary_search
"""


def binary_search(list_num, first_index, last_index, to_search):
    if last_index >= first_index:

        mid_index = (first_index + last_index) // 2
        mid_element = list_num[mid_index]

        if mid_element == to_search:
            return f"{mid_element} находится в позиции {mid_index}"

        elif mid_element > to_search:
            new_position = mid_index - 1
            return binary_search(list_num, first_index, new_position, to_search)

        elif mid_element < to_search:
            new_position = mid_index + 1
            return binary_search(list_num, new_position, last_index, to_search)

    else:
        return "Элемента нет в массиве"


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


N = 5000
x = np.array(range(10, N + 1, 10))
y = []
search = 200
first = 0
for n in range(10, N + 1, 10):
    a = np.array(np.random.randint(-100, 100, n))
    last = n - 1
    a.sort()
    y.append(ti.timeit(setup=my_setup, stmt="binary_search(a, first, last, search)",
number=30))

popt, pcov = curve_fit(log_func, x, y)
a_opt, b_opt = popt

x_fit = np.linspace(min(x), max(x), 100)
y_fit = log_func(x_fit, a_opt, b_opt)
print(f"Уравнение кривой: {a_opt} + {b_opt} * log(x)")

plt.scatter(x, y, s=5, color="b")
```

```
plt.plot(x_fit, y_fit, color="r")
plt.xlabel("Размер массива")
plt.ylabel("Время работы функции")
plt.title("Худший случай")
plt.show()
```

```
Уравнение кривой: 2.2353869817919296e-06 + 1.9035153478862367e-05 * log(x)
```

Рисунок 1. Уравнение кривой в худшем случае

Зависимость времени выполнения функции бинарного поиска от количества элементов в массиве в среднем случае:

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


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

my_setup = """
from __main__ import a
from __main__ import search
from __main__ import first
from __main__ import last
from __main__ import binary_search
"""


def binary_search(list_num, first_index, last_index, to_search):
    if last_index >= first_index:

        mid_index = (first_index + last_index) // 2
        mid_element = list_num[mid_index]

        if mid_element == to_search:
            return f"{mid_element} находится в позиции {mid_index}"

        elif mid_element > to_search:
            new_position = mid_index - 1
            return binary_search(list_num, first_index, new_position, to_search)

        elif mid_element < to_search:
            new_position = mid_index + 1
            return binary_search(list_num, new_position, last_index, to_search)

    else:
        return "Элемента нет в массиве"


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


N = 5000
x = np.array(range(10, N + 1, 10))
y = []
first = 0
for n in range(10, N + 1, 10):
    a = np.array(np.random.randint(-100, 100, n))
    last = n - 1
    a.sort()
    search = a[5]
```

```
    y.append(ti.timeit(setup=my_setup, stmt="binary_search(a, first, last, search)",
number=30))

popt, pcov = curve_fit(log_func, x, y)
a_opt, b_opt = popt
x_fit = np.linspace(min(x), max(x), 100)
y_fit = log_func(x_fit, a_opt, b_opt)
print(f"Уравнение кривой: {a_opt} + {b_opt} * log(x)")

plt.scatter(x, y, s=5, color="b")
plt.plot(x_fit, y_fit, color="r")
plt.xlabel("Размер массива")
plt.ylabel("Время работы функции")
plt.title("Средний случай")
plt.show()
```

```
Уравнение кривой: 2.733332382932980e-05 + 8.444943369120142e-06 * log(x)
```

Рисунок 2. Уравнение кривой в среднем случае

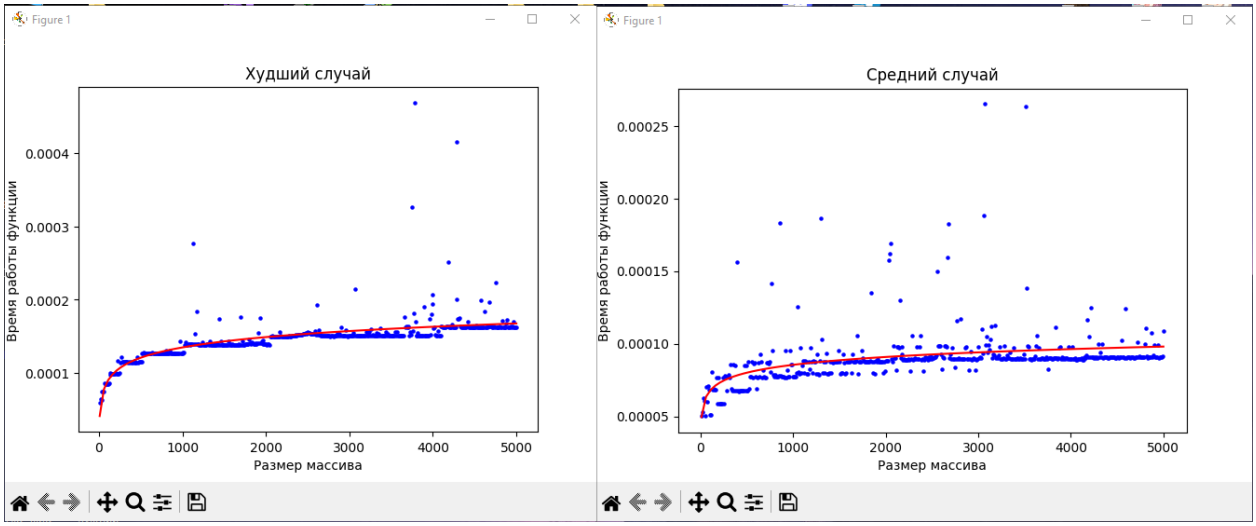

Рисунок 3. Результат выполнения программы для худшего и среднего случая

Определение времени выполнения бинарного поиска средствами python (модуль bisect) в худшем случае:

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


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

my_setup = """
from __main__ import a
from __main__ import search
import bisect
"""


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

```
N = 5000
x = np.array(range(10, N + 1, 10))
y = []

search = 200
for n in range(10, N + 1, 10):
    a = np.array(np.random.randint(-100, 100, n))
    a.sort()
    y.append(ti.timeit(setup=my_setup, stmt="bisect.bisect_left(a, search)",
number=30))

popt, pcov = curve_fit(log_func, x, y)
a_opt, b_opt = popt
x_fit = np.linspace(min(x), max(x), 100)
y_fit = log_func(x_fit, a_opt, b_opt)

plt.scatter(x, y, s=5, color="b")
plt.plot(x_fit, y_fit, color="r")
plt.xlabel("Размер массива")
plt.ylabel("Время работы функции")
plt.title("Худший случай")
plt.show()
```

Определение времени выполнения бинарного поиска средствами python (модуль bisect) в среднем случае:

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


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

my_setup = """
from __main__ import a
from __main__ import search
import bisect
"""


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


N = 5000
x = np.array(range(10, N + 1, 10))
y = []

for n in range(10, N + 1, 10):
    a = np.array(np.random.randint(-100, 100, n))
    a.sort()
    search = a[5]
    y.append(ti.timeit(setup=my_setup, stmt="bisect.bisect_left(a, search)",
number=30))

popt, pcov = curve_fit(log_func, x, y)
a_opt, b_opt = popt
x_fit = np.linspace(min(x), max(x), 100)
y_fit = log_func(x_fit, a_opt, b_opt)

plt.scatter(x, y, s=5, color="b")
plt.plot(x_fit, y_fit, color="r")
plt.xlabel("Размер массива")
plt.ylabel("Время работы функции")
plt.title("Средний случай")
plt.show()
```

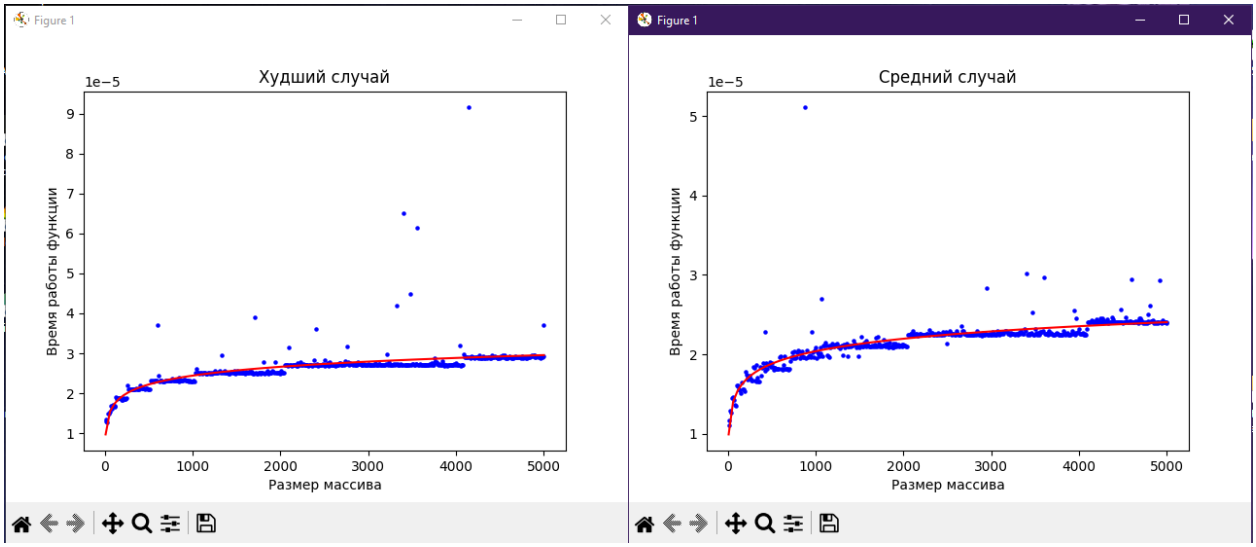

Рисунок 4. Результат выполнения программы для худшего и среднего случая