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 №10
дисциплины «Алгоритмизация»

Выполнил:
Дзуев Альберт Мухаметович
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., доцент кафедры инфокоммуникаций

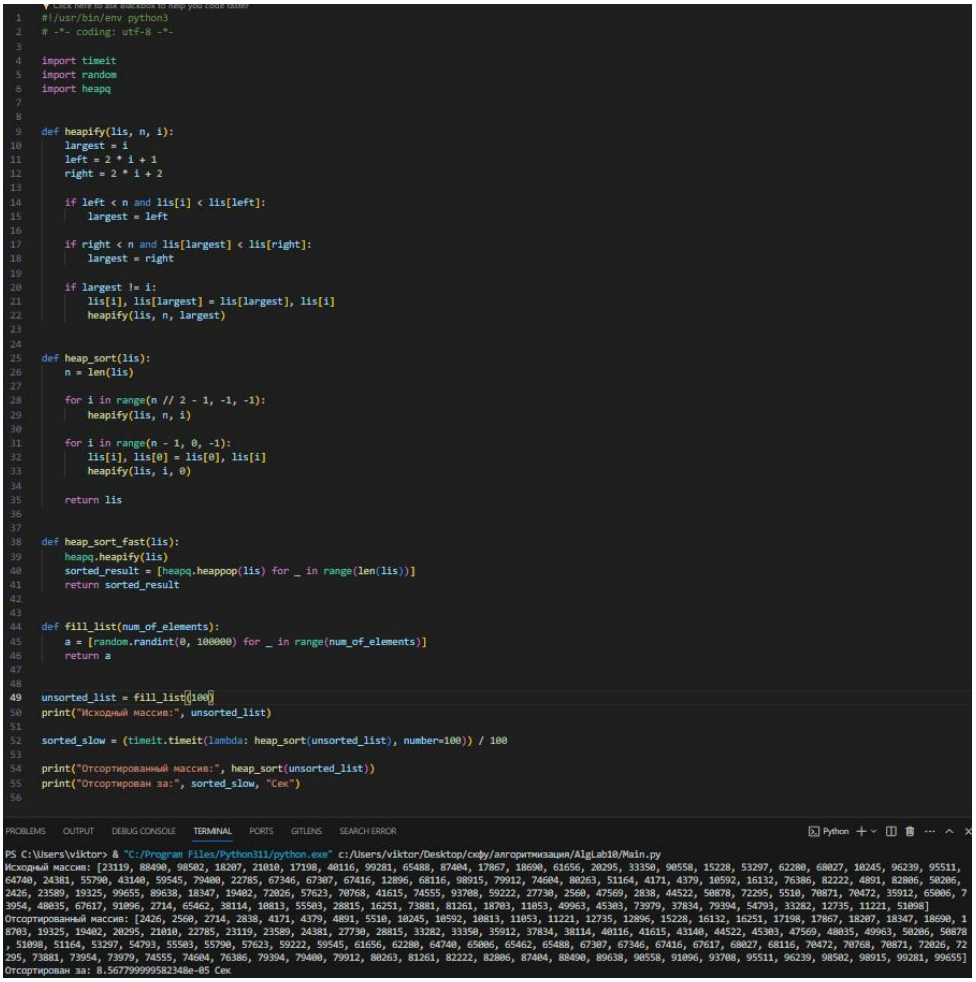
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. Написал программу поиска элемента в массиве, автоматического заполнения массива, расчёта тысячи точек, показывающих время поиска элемента в массиве в худшем и среднем случае, вывода графиков, составленных из этих точек, и подсчета корреляции:



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

import timeit
import random
import heapq


def heapify(lis, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and lis[i] < lis[left]:
        largest = left

    if right < n and lis[largest] < lis[right]:
        largest = right

    if largest != i:
        lis[i], lis[largest] = lis[largest], lis[i]
        heapify(lis, n, largest)


def heap_sort(lis):
    n = len(lis)

    for i in range(n // 2 - 1, -1, -1):
        heapify(lis, n, i)

    for i in range(n - 1, 0, -1):
        lis[i], lis[0] = lis[0], lis[i]
        heapify(lis, i, 0)

    return lis


def heap_sort_fast(lis):
    heapq.heapify(lis)
    sorted_result = [heapq.heappop(lis) for _ in range(len(lis))]
    return sorted_result


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


unsorted_list = fill_list(100)
print("Исходный массив:", unsorted_list)

sorted_slow = (timeit.timeit(lambda: heap_sort(unsorted_list), number=100)) / 100

print("Отсортированный массив:", heap_sort(unsorted_list))
print("Отсортирован за:", sorted_slow, "Сек")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SEARCH ERROR                                Python

PS C:\Users\viktor> & "C:/Program Files/Python311/python.exe" c:/Users/viktor/Desktop/схфу/алгоритмизация/AlgLab10/Main.py
Исходный массив: [23119, 88490, 98502, 18207, 21010, 17198, 40116, 99281, 65488, 87404, 17867, 18690, 61656, 20295, 33350, 90558, 15228, 53297, 62280, 68027, 10245, 96239, 95511,
64740, 24381, 55790, 43140, 59545, 79400, 22785, 67346, 67307, 67416, 12896, 68116, 98915, 79912, 74604, 80263, 51164, 4171, 4379, 10592, 16132, 76386, 82222, 4891, 82806, 50206,
2426, 23589, 19325, 99655, 89638, 18347, 19402, 72026, 57623, 70768, 41615, 74555, 93708, 59222, 27730, 2560, 47569, 2838, 44522, 50878, 72295, 5510, 70871, 70472, 35912, 65006, 7
3954, 48035, 67617, 91096, 2714, 65462, 38114, 10813, 55503, 28815, 16251, 73881, 81261, 18703, 11053, 49963, 45303, 73979, 37834, 79394, 54793, 33282, 12735, 11221, 51098]
Отсортированный массив: [2426, 2560, 2714, 2838, 4171, 4379, 4891, 5510, 10245, 10592, 10813, 11053, 11221, 12735, 12896, 15228, 16132, 16251, 17198, 17867, 18207, 18347, 18690, 1
8703, 19325, 19402, 20295, 21010, 22785, 23119, 23589, 24381, 27730, 28815, 33282, 33350, 35912, 37834, 38114, 40116, 41615, 43140, 44522, 45303, 47569, 48035, 49963, 50206, 50878
, 51098, 51164, 53297, 54793, 55503, 55790, 57623, 59222, 59545, 61656, 62280, 64740, 65006, 65462, 65488, 67307, 67346, 67416, 67617, 68027, 68116, 70472, 70768, 70871, 72026, 72
295, 73881, 73954, 73979, 74555, 74604, 76386, 79394, 79400, 79912, 80263, 81261, 82222, 82806, 87404, 88490, 89638, 90558, 91096, 93708, 95511, 96239, 98502, 98915, 99281, 99655]
Отсортирован за: 8.567799999582348e-05 Сек
```

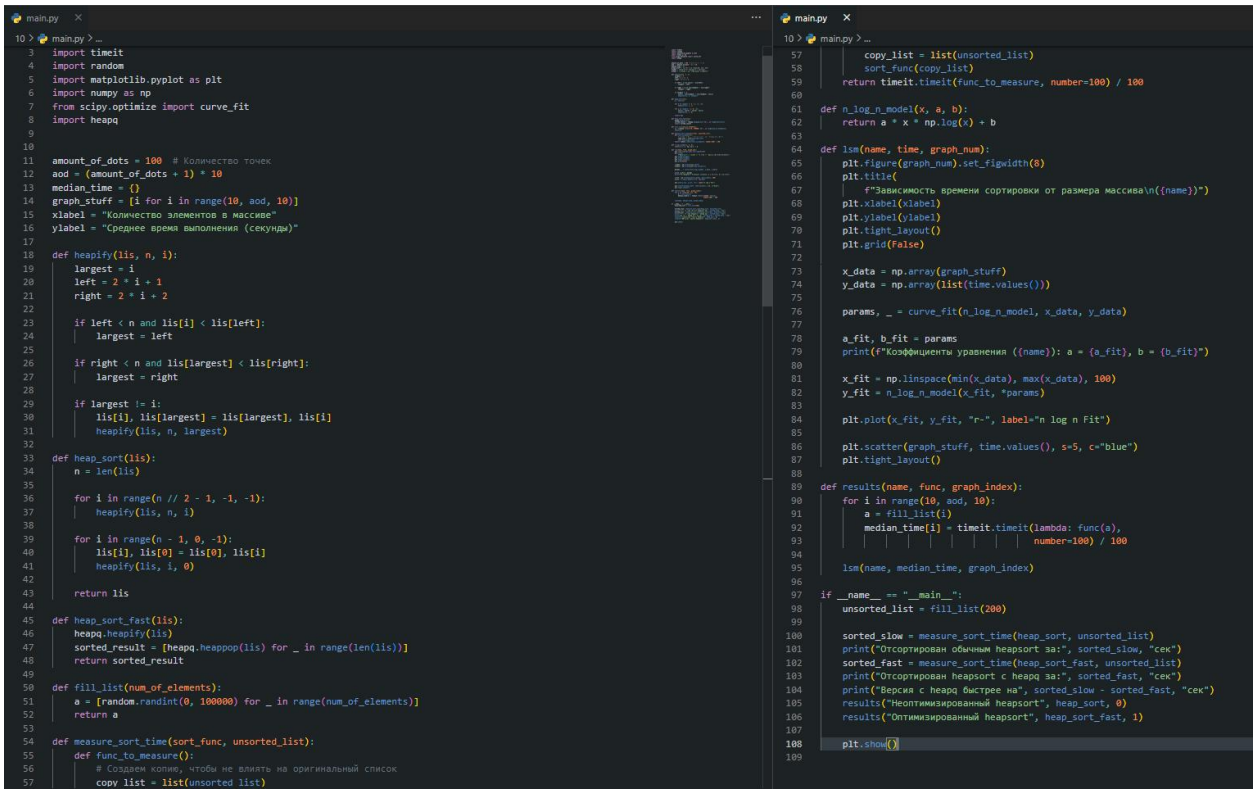
Рисунок 1. Код и результат неоптимизированного алгоритма heapsort

Таблица 1. Сравнение алгоритма Heap Sort с Quick Sort и Merge Sort

Характеристика	**Heap Sort**	**Quick Sort**	**Merge Sort**
Сложность времени	O(n log n)	O(n^2) в худшем случае, O(n log n) в среднем	O(n log n)
Сложность по памяти	O(1) или O(log n)	O(log n) в среднем	O(n)
Лучший случай	O(n log n)	O(n log n)	O(n log n)
Худший случай	O(n log n)	O(n^2)	O(n log n)
Средний случай	O(n log n)	O(n log n)	O(n log n)

Heapsort не требует доп. память, занимает меньше всех места, но считается нестабильным. Quick Sort медленный в худшем случае, занимает больше места, требует доп. память и является нестабильным. Merge Sort стабилен, быстр, не требует доп памяти, но имеет наибольшую сложность по памяти.

2. Произвел оптимизацию алгоритма при помощи встроенной библиотеки heapq:



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


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

def heapify(lis, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and lis[i] < lis[left]:
        largest = left

    if right < n and lis[largest] < lis[right]:
        largest = right

    if largest != i:
        lis[i], lis[largest] = lis[largest], lis[i]
        heapify(lis, n, largest)

def heap_sort(lis):
    n = len(lis)

    for i in range(n // 2 - 1, -1, -1):
        heapify(lis, n, i)

    for i in range(n - 1, 0, -1):
        lis[i], lis[0] = lis[0], lis[i]
        heapify(lis, i, 0)

    return lis

def heap_sort_fast(lis):
    heapq.heapify(lis)
    sorted_result = [heapq.heappop(lis) for _ in range(len(lis))]
    return sorted_result

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

def measure_sort_time(sort_func, unsorted_list):
    def func_to_measure():
        # Создаем копию, чтобы не влиять на оригинальный список
        copy_list = list(unsorted_list)
        sort_func(copy_list)
    return timeit.timeit(func_to_measure, number=100) / 100

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

def lsm(name, time, graph_num):
    plt.figure(graph_num).set_figwidth(8)
    plt.title(
        f"Зависимость времени сортировки 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()))

    params, _ = curve_fit(n_log_n_model, x_data, y_data)

    a_fit, b_fit = params
    print(f"Коэффициенты уравнения ({name}): a = {a_fit}, b = {b_fit}")

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

    plt.plot(x_fit, y_fit, "r-", label="n log n Fit")

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

def results(name, func, graph_index):
    for i in range(10, aod, 10):
        a = fill_list(i)
        median_time[i] = timeit.timeit(lambda: func(a),
                                       number=100) / 100

    lsm(name, median_time, graph_index)

if __name__ == "__main__":
    unsorted_list = fill_list(200)

    sorted_slow = measure_sort_time(heap_sort, unsorted_list)
    print("Отсортирован обычным heapsort за:", sorted_slow, "сек")
    sorted_fast = measure_sort_time(heap_sort_fast, unsorted_list)
    print("Отсортирован heapsort с heapq за:", sorted_fast, "сек")
    print("Версия с heapq быстрее на", sorted_slow - sorted_fast, "сек")
    results("Неоптимизированный heapsort", heap_sort, 0)
    results("Оптимизированный heapsort", heap_sort_fast, 1)

    plt.show()
```

Рисунок 2. Оптимизированный алгоритм heapsort

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