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 ПРАКТИЧЕСКОЙ РАБОТЕ №10
дисциплины «Алгоритмизация»

Выполнил:
Мотовилов Вадим Борисович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Информатика и вычислительная
техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович
_____
(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:

proalg1.py > ...

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

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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\1\algoritm10\prog> & "C:/Program Files/Python311/python.exe" c:/Users/1/algoritm10/prog/proalg1
Отсортирован обычным heapsort за: 0.0003450909999992291 сек
Отсортирован heapsort с heapq за: 3.8548000011360275e-05 сек
Версия с heapq быстрее на 0.0003065429999878688 сек
Коэффициенты уравнения (Неоптимизированный heapsort): a = 3.633415320134366e-07, b = -2.8011694025368e-05
Коэффициенты уравнения (Оптимизированный heapsort): a = 3.563089876689966e-10, b = 4.982005875182478e-07
```

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:

proalg1.py > ...

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

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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\1\algoritm10\prog> & "C:/Program Files/Python311/python.exe" c:/Users/1/algoritm10/prog/proalg1
Отсортирован обычным heapsort за: 0.0003450909999992291 сек
Отсортирован heapsort с heapq за: 3.8548000011360275e-05 сек
Версия с heapq быстрее на 0.0003065429999878688 сек
Коэффициенты уравнения (Неоптимизированный heapsort): a = 3.633415320134366e-07, b = -2.8011694025368e-05
Коэффициенты уравнения (Оптимизированный heapsort): a = 3.563089876689966e-10, b = 4.982005875182478e-07
```

Рисунок 2. Оптимизированный алгоритм heapsort

Применение в реальной жизни: Системы с ограниченной памятью: Эффективен там, где важно минимизировать использование дополнительной памяти.

Приоритетные очереди: Используется для эффективного управления данными с приоритетом, например в обработке событий.

Потоковая обработка: Подходит для сортировки данных в реальном времени, когда данные постоянно поступают.

Базы данных: Помогает в сортировке больших объемов данных вне основной памяти, через внешнюю сортировку.

Вычислительные задачи с таймингом: Хорош для задач в реальном времени, где важна предсказуемость времени выполнения операций.

Heap Sort выбирают из-за надёжности и предсказуемости, когда стоит избегать риска существенного замедления из-за худшего случая выполнения, как, например, у Quick Sort. Также он полезен, если требуется сортировать данные без дополнительного пространства, например, при выполнении сортировки прямо на физических носителях или в ситуациях с ограниченной доступной памятью.

Анализ сложности:

Добавил рисование графиков зависимости времени сортировки с помощью оптимизированного и не оптимизированного heapsort от количества элементов в массиве.

Поскольку Heap Sort может выполняться in-place, для его работы теоретически не требуется дополнительное пространство кроме самого массива, который сортируется. При этом методе сортировки:

- Не требуется выделять дополнительный массив для разделенияданных.
- Манипуляции с элементами осуществляются в пределах того жемассива.
- Требуется ограниченное количество переменных для хранения индексов и временных значений в процессе выполнения алгоритма.

Heap Sort возможно реализовать без рекурсии, при этом алгоритм получается с чистой пространственной сложностью O(1), то есть он будет занимать константное дополнительное пространство, не зависимо от размеров ных данных. Эта модификация использует только циклы и не требует дополнительной памяти для рекурсивного стека, что делает пространственную сложность чисто константной.

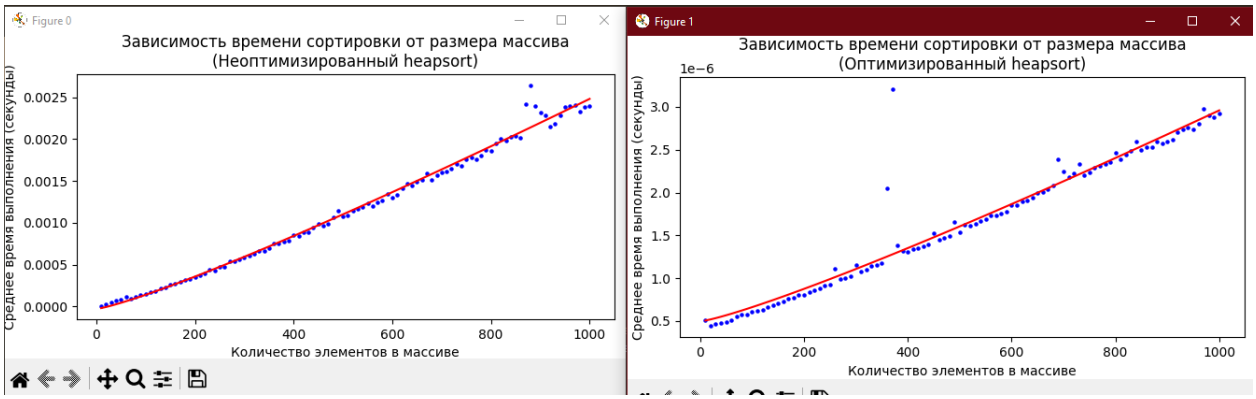
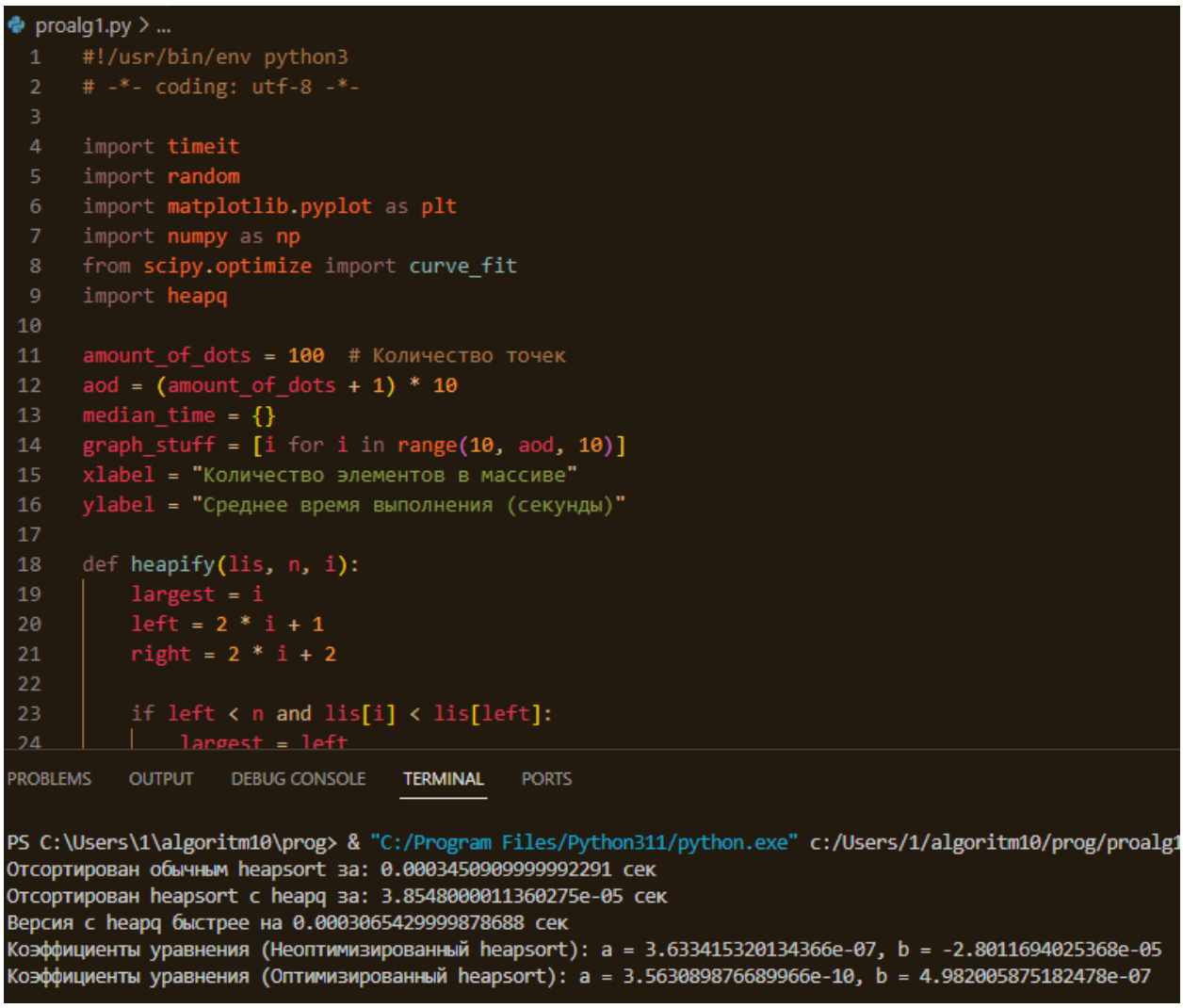


Рисунок 3. Графики зависимости времени сортировки с помощью оптимизированного и не оптимизированного heapsort от количества элементов в массиве



```
proalg1.py > ...
  1   #!/usr/bin/env python3
  2   # -*- coding: utf-8 -*-
  3
  4   import timeit
  5   import random
  6   import matplotlib.pyplot as plt
  7   import numpy as np
  8   from scipy.optimize import curve_fit
  9   import heapq
 10
 11   amount_of_dots = 100  # Количество точек
 12   aod = (amount_of_dots + 1) * 10
 13   median_time = {}
 14   graph_stuff = [i for i in range(10, aod, 10)]
 15   xlabel = "Количество элементов в массиве"
 16   ylabel = "Среднее время выполнения (секунды)"
 17
 18   def heapify(lis, n, i):
 19       largest = i
 20       left = 2 * i + 1
 21       right = 2 * i + 2
 22
 23       if left < n and lis[i] < lis[left]:
 24           largest = left

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\1\algoritm10\prog> & "C:/Program Files/Python311/python.exe" c:/Users/1/algoritm10/prog/proalg1
Отсортирован обычным heapsort за: 0.0003450909999992291 сек
Отсортирован heapsort с heapq за: 3.8548000011360275e-05 сек
Версия с heapq быстрее на 0.0003065429999878688 сек
Коэффициенты уравнения (Неоптимизированный heapsort): a = 3.633415320134366e-07, b = -2.8011694025368e-05
Коэффициенты уравнения (Оптимизированный heapsort): a = 3.563089876689966e-10, b = 4.982005875182478e-07
```

Рисунок 4. Полный код с двумя алгоритмами heapsort и выводом графиков скорости их сортировки

3. Решение задания:

Рисунок 5. Код решения задания 6 и результат выполнения

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.