САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3
по курсу «Алгоритмы и структуры данных»
Тема: Quick sort

Выполнил:
Лазарев Марк Олегович
К3241

Санкт-Петербург
2025 г.

Задачи по варианту

Задача №1. Улучшение Quick sort
Задача №2. Анти-quick sort
Задача №3. Цифровая сортировка

Задачи по варианту

1 задача. Улучшение Quick sort

1. Используя *псевдокод* процедуры `Randomized - QuickSort`, а так же `Partition` из презентации к Лекции 3 (страницы 8 и 12), напишите программу быстрой сортировки на Python и проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

   - **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^4$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, ***по модулю*** не превосходящих 10^9.
   - **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
   - Ограничение по времени. 2 сек.
   - Ограничение по памяти. 256 мб.
   - Для проверки можно выбрать наихудший случай, когда сортируется массив рамера $10^3, 10^4, 10^5$ чисел порядка 10^9, отсортированных в обратном порядке; наилучший, когда массив уже отсортирван, и средний - случайный. Сравните на данных сетах `Randomized-QuickSort` и простой `QuickSort`. (А также есть `Median-QuickSort`, см. задание 10.2; и `Tail-Recursive-QuickSort`, см. Кормен. 2013, стр. 217)

2. **Основное задание.** Цель задачи - переделать данную реализацию рандомизированного алгоритма быстрой сортировки, чтобы она работала быстро даже с последовательностями, содержащими много одинаковых элементов. Чтобы заставить алгоритм быстрой сортировки эффективно обрабатывать последовательности с несколькими уникальными элементами, нужно заменить двухстороннее разделение на трехстороннее (смотри в Лекции 3 слайд 17). То есть ваша новая процедура разделения должна разбить массив на три части:

   - $A[k] < x$ для всех $\ell + 1 \leq k \leq m_1 - 1$
   - $A[k] = x$ для всех $m_1 \leq k \leq m_2$
   - $A[k] > x$ для всех $m_2 + 1 \leq k \leq r$

   - Формат входного и выходного файла аналогичен п.1.
   - Аналогично п.1 этого задания сравните `Randomized-QuickSort` +с `Partition` и ее с `Partition3` на сетах случайных данных, в которых содержатся всего несколько уникальных элементов при $n = 10^3, 10^4, 10^5$. Что быстрее, `Randomized-QuickSort` +с `Partition3` или `Merge-Sort`?

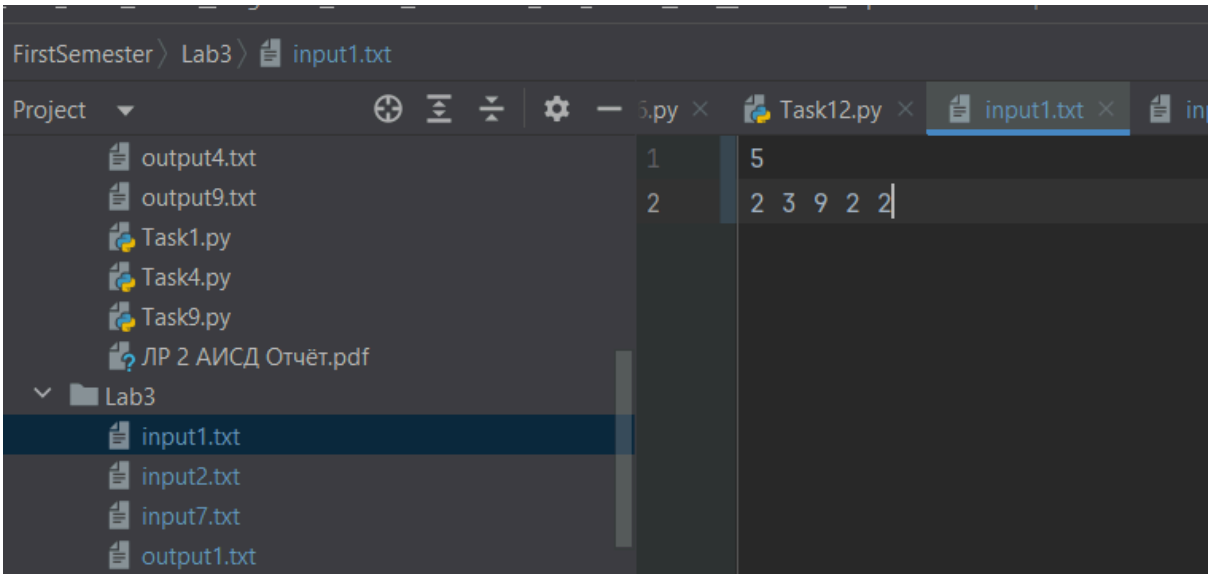
Код программы:

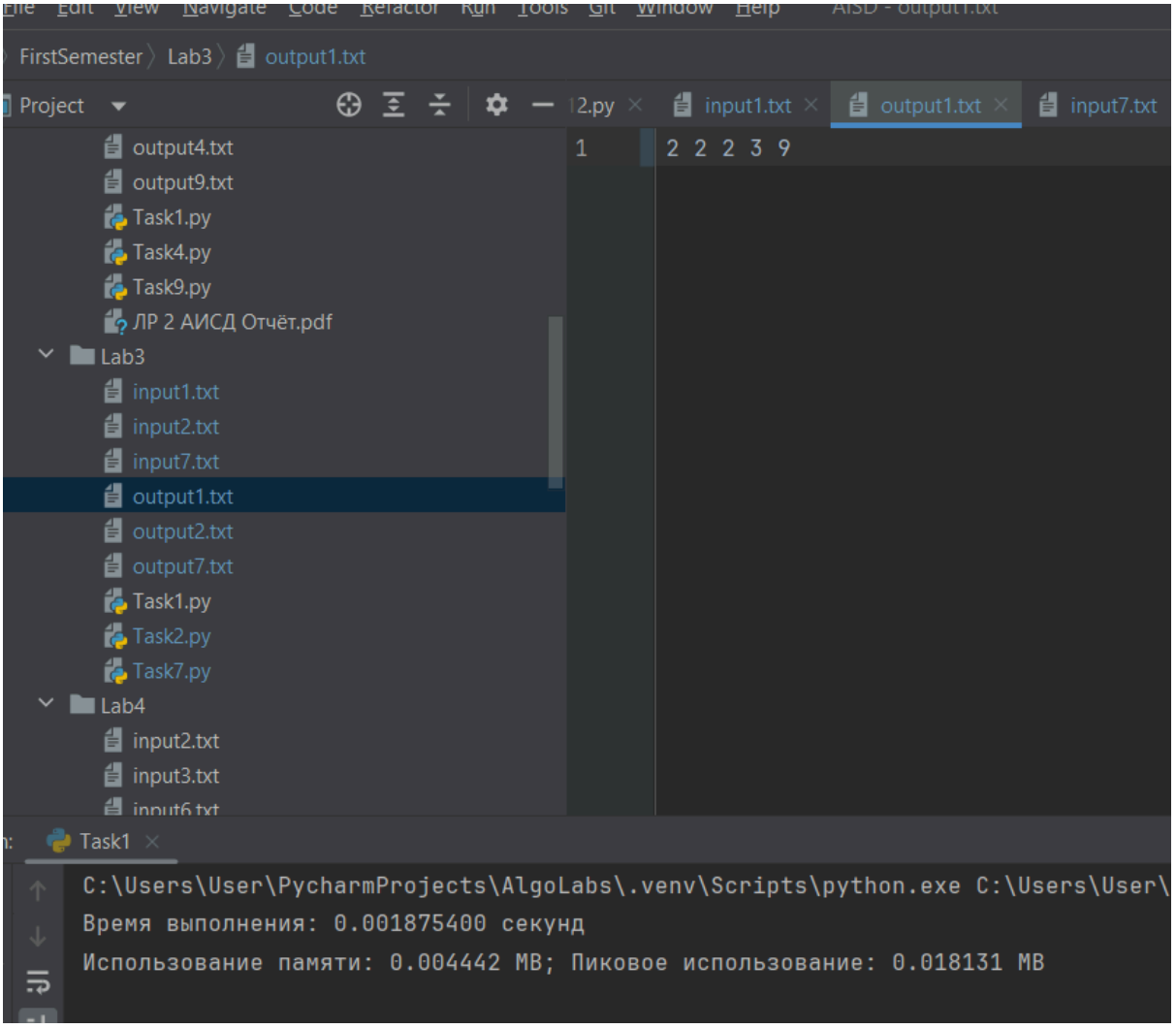
```
import random
import time
import tracemalloc

def partition(arr, low, high):
    pivot = arr[high]
```

```
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def randomized_partition(arr, low, high):
    rand_pivot = random.randint(low, high)
       arr[high],  arr[rand_pivot]  =  arr[rand_pivot],
arr[high]
    return partition(arr, low, high)

def randomized_quick_sort(arr, low, high):
    if low < high:
        pi = randomized_partition(arr, low, high)
        randomized_quick_sort(arr, low, pi - 1)
        randomized_quick_sort(arr, pi + 1, high)

def main():
    # Начало отслеживания времени и памяти
    start_time = time.perf_counter()
    tracemalloc.start()

    with open('input1.txt', 'r') as file:
        n = int(file.readline().strip())
                              array    =    list(map(int,
file.readline().strip().split()))

    randomized_quick_sort(array, 0, n - 1)

    with open('output1.txt', 'w') as file:
        file.write(' '.join(map(str, array)))

    # Подсчет времени и памяти
    end_time = time.perf_counter()
    current, peak = tracemalloc.get_traced_memory()
```

```
    tracemalloc.stop()

        print(f"Время выполнения: {end_time - start_time:.9f} секунд")
        print(f"Использование памяти: {current / 10**6:.6f} MB; Пиковое использование: {peak / 10**6:.6f} MB")

if __name__ == "__main__":
    main()
```

Результат работы кода на примерах:


FirstSemester 〉 Lab3 〉 input1.txt
Project
Task12.py
input1.txt
output4.txt
output9.txt
Task1.py
Task4.py
Task9.py
ЛР 2 АИСД Отчёт.pdf
Lab3
input1.txt
input2.txt
input7.txt
output1.txt
1 5
2 2 3 9 2 2


FirstSemester 〉 Lab3 〉 output1.txt
Project
input1.txt
output1.txt
input7.txt
output4.txt
output9.txt
Task1.py
Task4.py
Task9.py
ЛР 2 АИСД Отчёт.pdf
Lab3
input1.txt
input2.txt
input7.txt
output1.txt
output2.txt
output7.txt
Task1.py
Task2.py
Task7.py
Lab4
input2.txt
input3.txt
1 2 2 2 3 9
Task1
C:\Users\User\PycharmProjects\AlgoLabs\.venv\Scripts\python.exe C:\Users\User\
Время выполнения: 0.001875400 секунд
Использование памяти: 0.004442 MB; Пиковое использование: 0.018131 MB

2 задача. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка - `QuickSort`. Далее приведена программа на языке ~~Pascal~~ Python, которая сортирует массив a, используя этот алгоритм.

```
def qsort (left, right):
        key = a [(left + right) // 2]
        i = left
        j = right
        while i <= j:
                while a[i] < key: # first while
                        i += 1
                while a[j] > key : # second while
                        j -= 1
                if i <= j :
                        a[i], a[j] = a[j], a[i]
                        i += 1
                        j -= 1
        if left < j:
                qsort(left, j)
        if i < right:
                qsort(i, right)

qsort(0, n - 1)
```

Хотя `QuickSort` является очень быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Оценивать время работы алгоритма будем числом сравнений с элементами массива (то есть, суммарным числом сравнений в первом и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений. Задача на acmp.

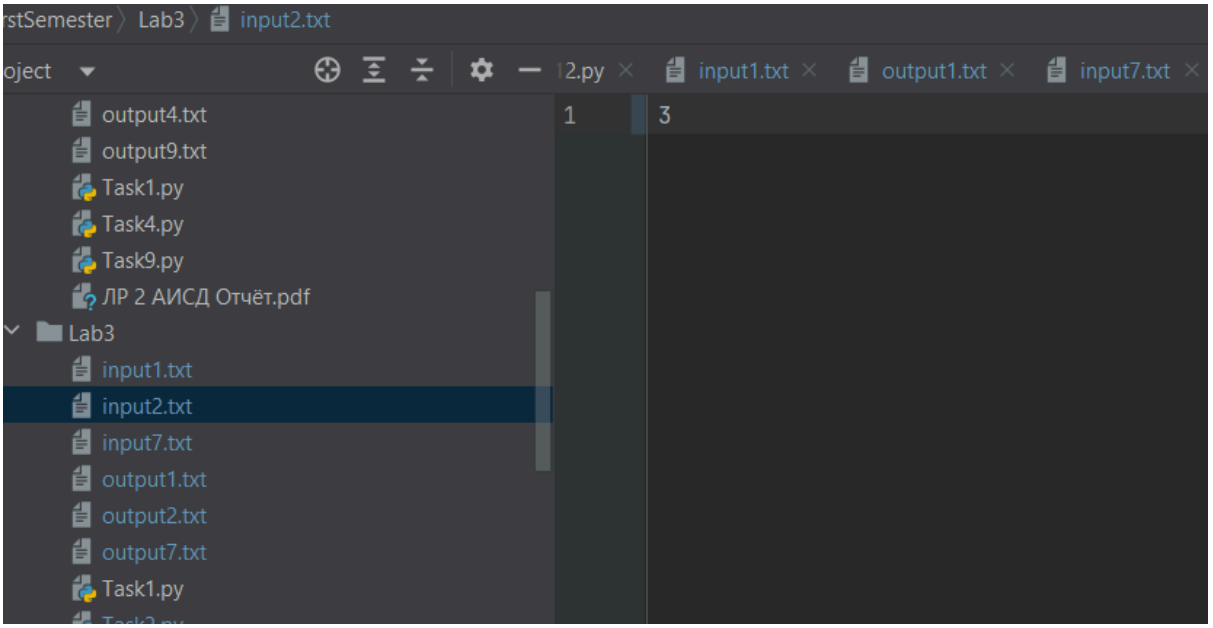
- **Формат входного файла (input.txt).** В первой строке находится единственное число n ($1 \le n \le 10^6$).
- **Формат выходного файла (output.txt).** Вывести перестановку чисел от 1 до n, на которой быстрая сортировка выполнит максимальное число сравнений. Если таких перестановок несколько, вывести любую из них.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

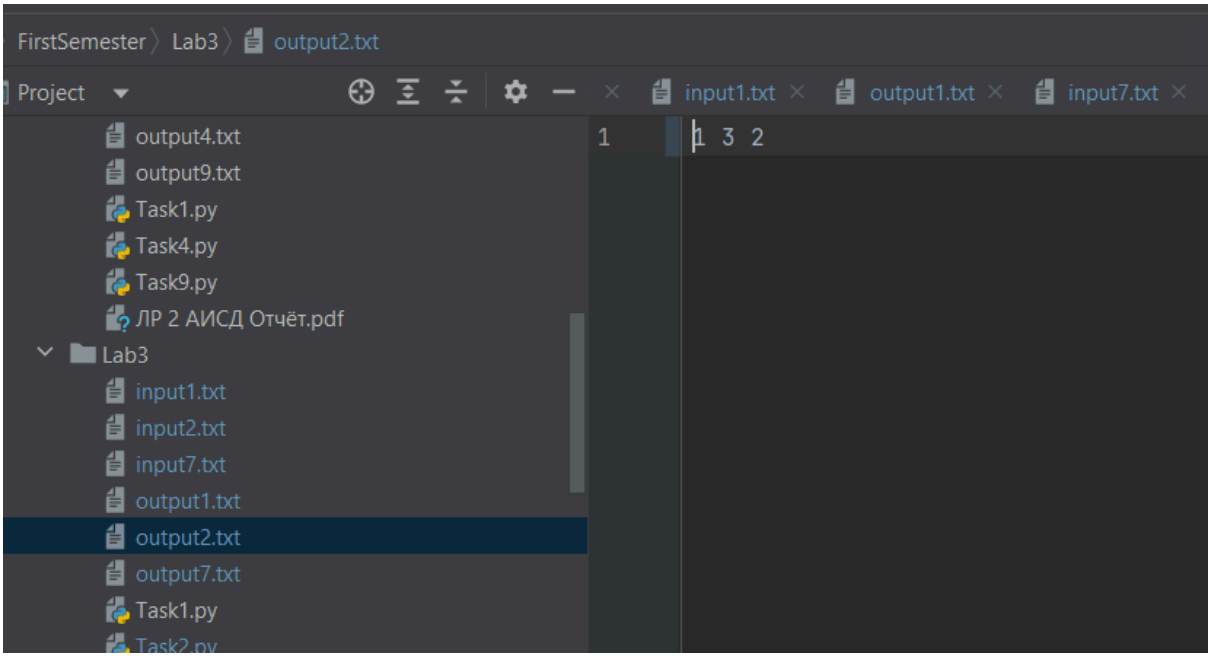
Код программы:

```
def generate_worst_case(n):
    if n == 1:
        return [1]
    elif n == 2:
```

```
        return [1, 2]
    else:
        return [1] + list(range(n, 1, -1))


with open('input2.txt', 'r') as file:
    n = int(file.readline().strip())

# Генерация наихудшей перестановки
worst_case = generate_worst_case(n)

with open('output2.txt', 'w') as file:
    file.write(' '.join(map(str, worst_case)))
```

Результат работы кода на примерах:




FirstSemester
Lab3
output2.txt
Project
input1.txt
output1.txt
input7.txt
output4.txt
output9.txt
Task1.py
Task4.py
Task9.py
ЛР 2 АИСД Отчёт.pdf
Lab3
input1.txt
input2.txt
input7.txt
output1.txt
output2.txt
output7.txt
Task1.py
1
1 3 2

7 задача. Цифровая сортировка

Дано n строк, выведите их порядок после k фаз цифровой сортировки.

- **Формат входного файла (input.txt).** В первой строке входного файла содержатся числа n - число строк, m - их длина и k - число фаз цифровой сортировки ($1 \leq n \leq 10^6$, $1 \leq k \leq m \leq 10^6$, $n \cdot m \leq 5 \cdot 10^7$). Далее находится описание строк, но **в нетривиальном формате**. Так, i-ая строка ($1 \leq i \leq n$) записана в i-ых символах второй, ..., $(m+1)$-ой строк входного файла. Иными словами, строки написаны по вертикали. **Это сделано специально, чтобы сортировка занимала меньше времени.**

Строки состоят из строчных латинских букв: от символа "a"до символа "z"включительно. В таблице символов ASCII все эти буквы располагаются подряд и в алфавитном порядке, код буквы "a"равен 97, код буквы "z"равен 122.

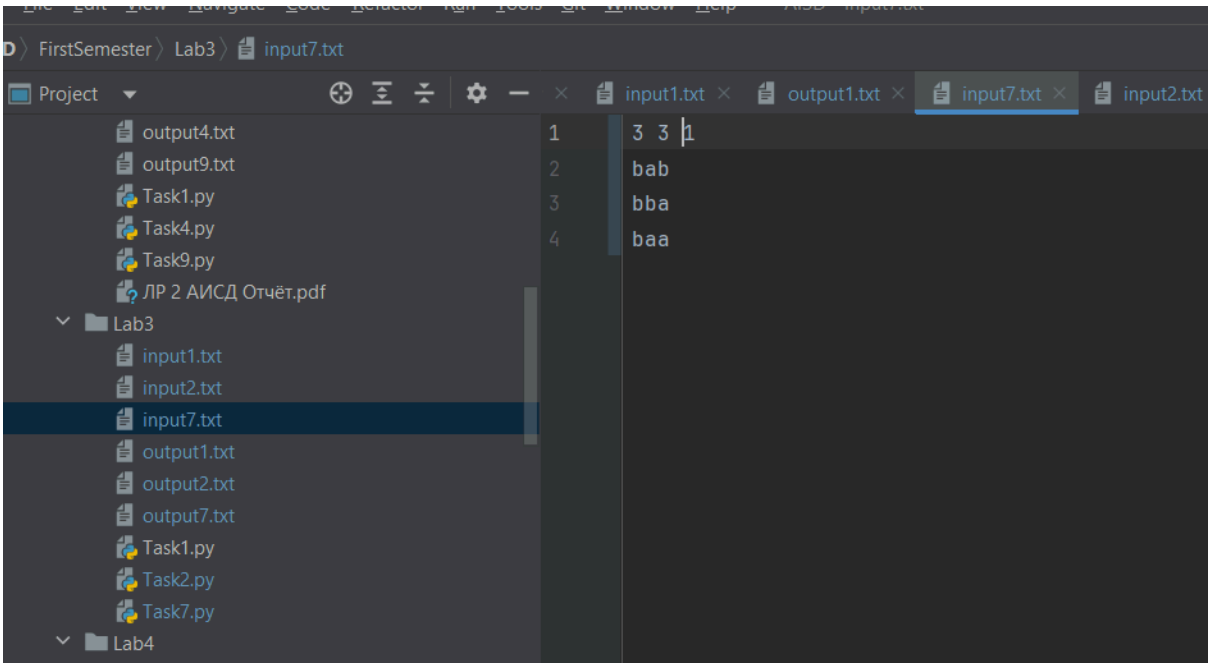
- **Формат выходного файла (output.txt).** Выведите номера строк в том порядке, в котором они будут после k фаз цифровой сортировки.
- Ограничение по времени. 3 сек.
- Ограничение по памяти. 256 мб.

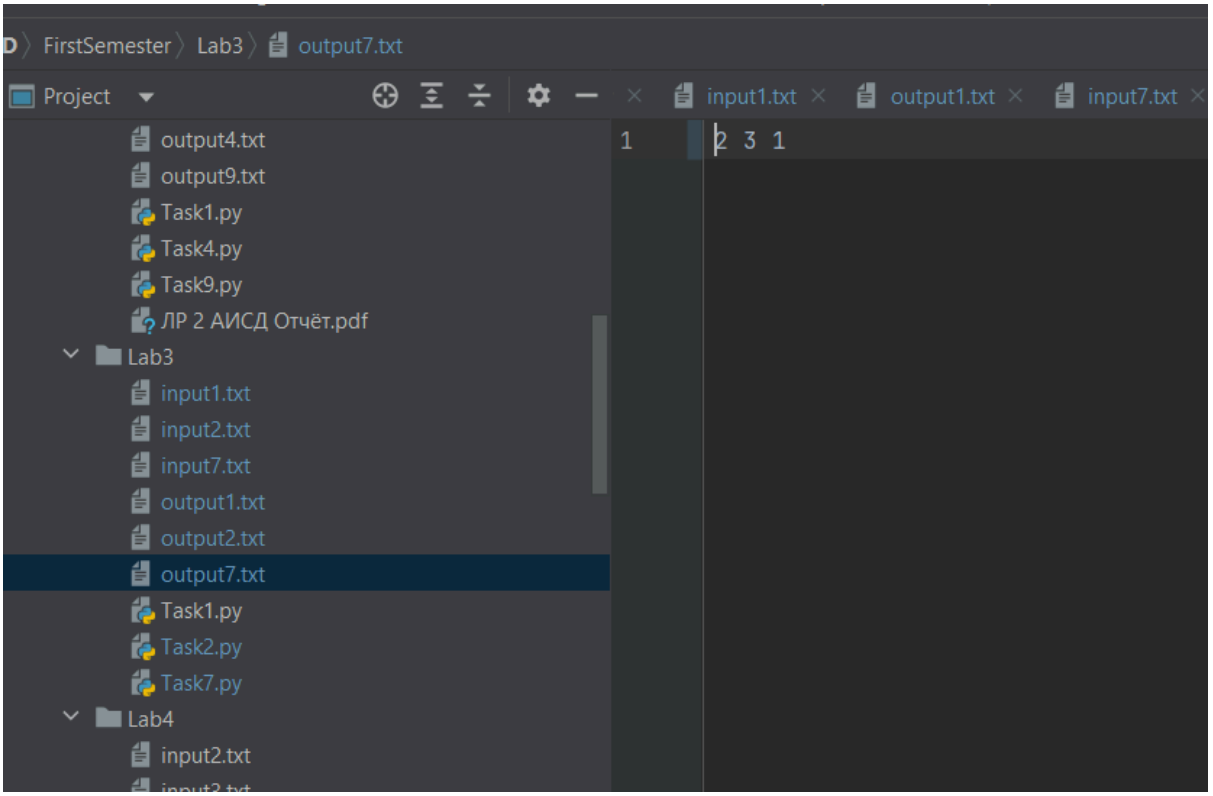
Код программы

```
def radix_sort(strings, k):
    # Создаем список для хранения строк вместе с их индексами
    indexed_strings = [(s, i) for i, s in enumerate(strings)]

    # Проведение k фаз цифровой сортировки
    for phase in range(k):
        # Создаем списки для каждой буквы алфавита
        buckets = [[] for _ in range(26)]

        # Распределяем строки по спискам в зависимости от текущего символа
        for s, idx in indexed_strings:
            char_code = ord(s[-phase - 1]) - ord('a')
            buckets[char_code].append((s, idx))

        # Объединяем списки обратно в один
        indexed_strings = []
        for bucket in buckets:
            indexed_strings.extend(bucket)

    # Возвращаем индексы строк в отсортированном порядке
    return [idx for _, idx in indexed_strings]


def read_input(filename):
    with open(filename, 'r') as file:
        n, m, k = map(int, file.readline().split())

        # Читаем строки по вертикали
        strings = [''] * n
        for i in range(m):
            line = file.readline().strip()
            for j in range(n):
                strings[j] += line[j]
```

```
    return n, m, k, strings


def write_output(filename, indices):
    with open(filename, 'w') as file:
        file.write(' '.join(map(lambda x: str(x + 1),
indices)))


def main():
    n, m, k, strings = read_input('input7.txt')
    sorted_indices = radix_sort(strings, k)
    write_output('output7.txt', sorted_indices)


if __name__ == '__main__':
    main()
```

Результат работы кода на примерах:


FirstSemester
Lab3
input7.txt
Project
input1.txt
output1.txt
input7.txt
input2.txt
output4.txt
output9.txt
Task1.py
Task4.py
Task9.py
ЛР 2 АИСД Отчёт.pdf
Lab3
input1.txt
input2.txt
input7.txt
output1.txt
output2.txt
output7.txt
Task1.py
Task2.py
Task7.py
Lab4
1 3 3 1
2 bab
3 bba
4 baa


FirstSemester
Lab3
output7.txt
Project
input1.txt
output1.txt
input7.txt
output4.txt
output9.txt
Task1.py
Task4.py
Task9.py
ЛР 2 АИСД Отчёт.pdf
Lab3
input1.txt
input2.txt
input7.txt
output1.txt
output2.txt
output7.txt
Task1.py
Task2.py
Task7.py
Lab4
input2.txt
1 2 3 1