Санкт-Петербургский Государственный Технический Университет (Технологический институт)

Кафедра системного анализа и информационных технологий

**Лабораторная работа №3**

Выполнил:

Андреев С.А

Проверил Мусаев А.А.

Санкт-Петербург,

2022

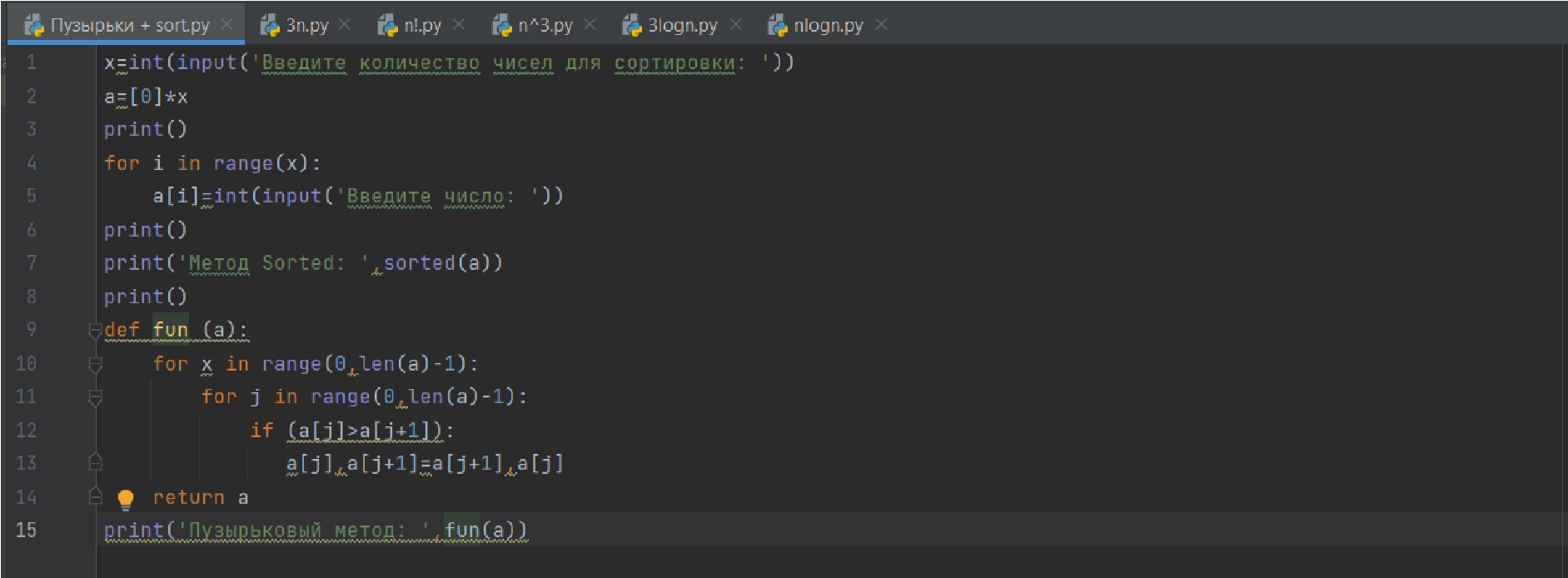
# Ход работы

**Задание 1**

**“Написать программу для пузырьковой сортировки. Оценить сложность данного метода. Сравнить с методом sort().”**

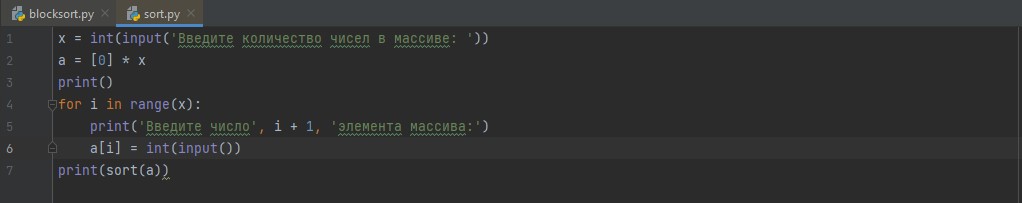
**Выполнение:**

Я написал программу для сортировки методом “пузырька”. Код программы представлен на скриншоте



Этот код позволяет сортировать заданный массив методом «пузырька».

В языке программирования Python существует метод “sort()”. Этот метод также позволяет выполнить сортировку чисел в массиве.



В итоге: две программы, которые выполняют сортировку массива, но выполняют они эту операцию разными способами.Необходимо сравнить их.

Сложность алгоритма сортировки методом “пузырька” можно оценить с помощью

концепции **Big O**. Данный метод сортировки содержит в себе три цикла **for**, один из которых вложен в другой. Итого мы имеем сложность **О(n + n2)**. Однако **n** можно пренебречь, потому что это не что иное, как, так называемая, “ненужная сложность”. Соответственно, окончательно сложность нашей программы можно охарактеризовать как **О(n2)**. Тогда как метод sort имеет сложность O(nlog)

2

3

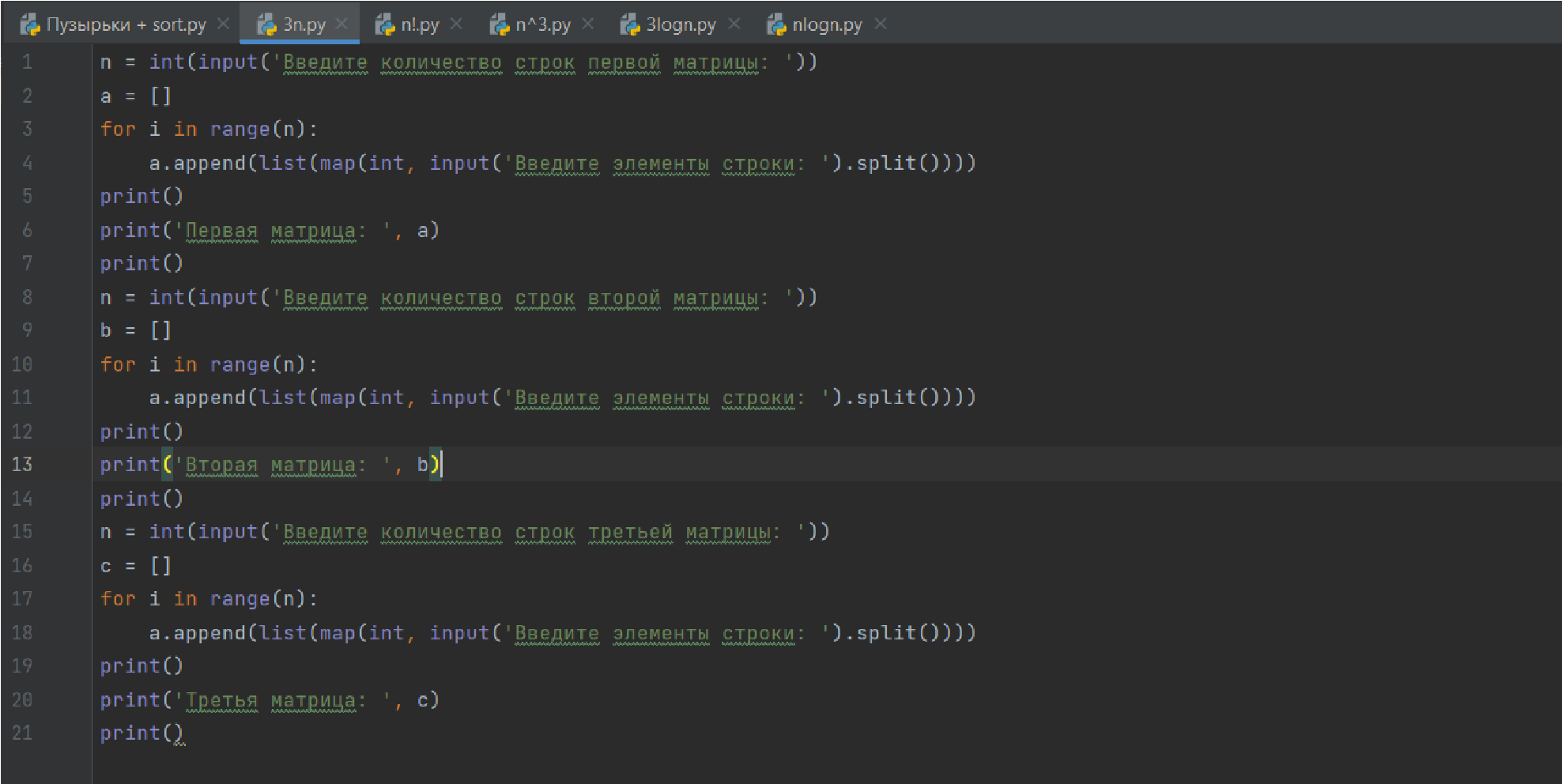
**Задание 2**

**“Придумать и реализовать алгоритмы, имеющие сложность O(3n), O(n log n), O(n!), O(n^3), O(3log(n))”**

**Выполнение:**

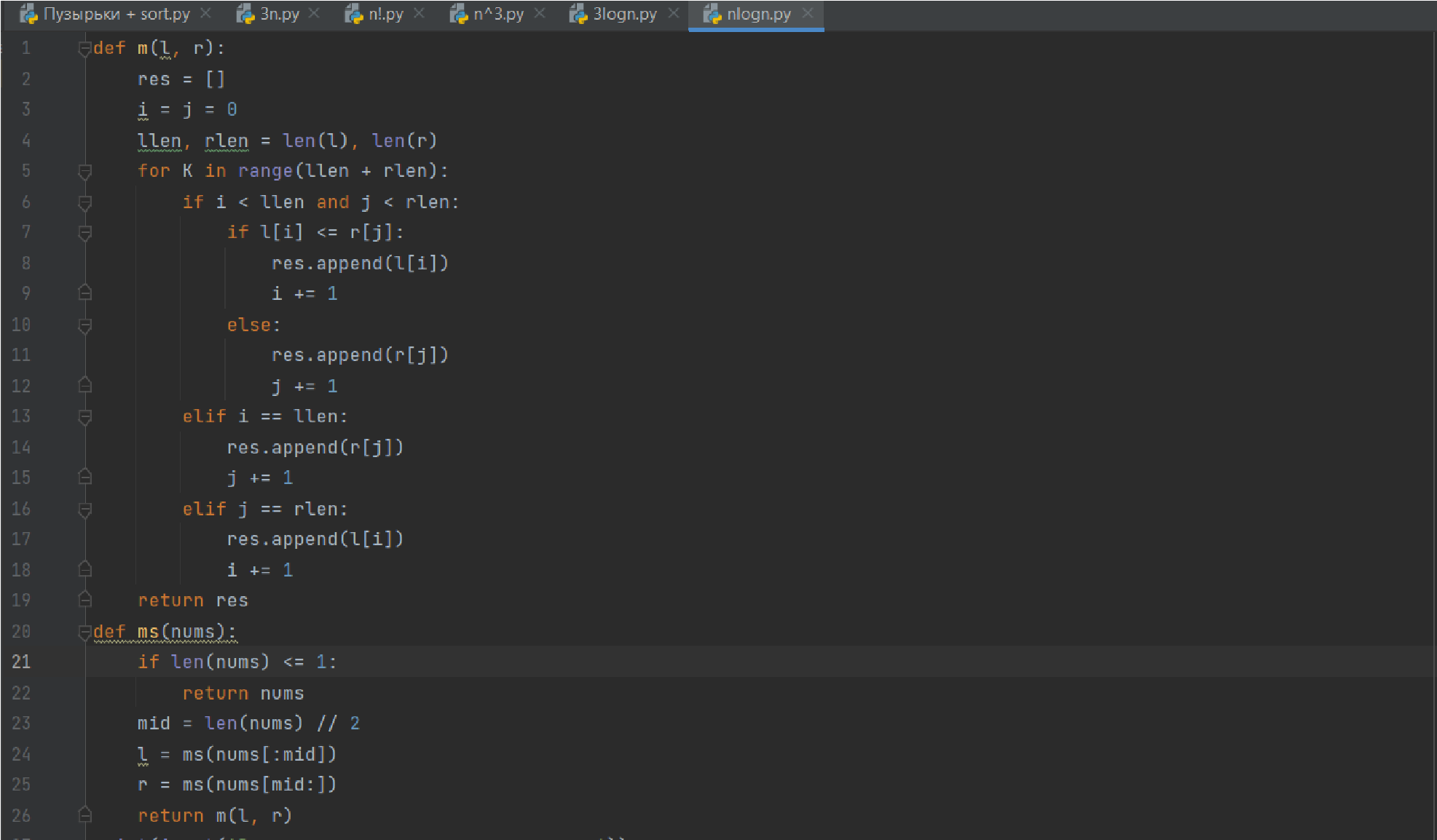
С помощью доп.источников я написал программы (коды), имеющие сложность **O(3n)** (рис.3)**, O(n log n)** (рис.4,5)**,**

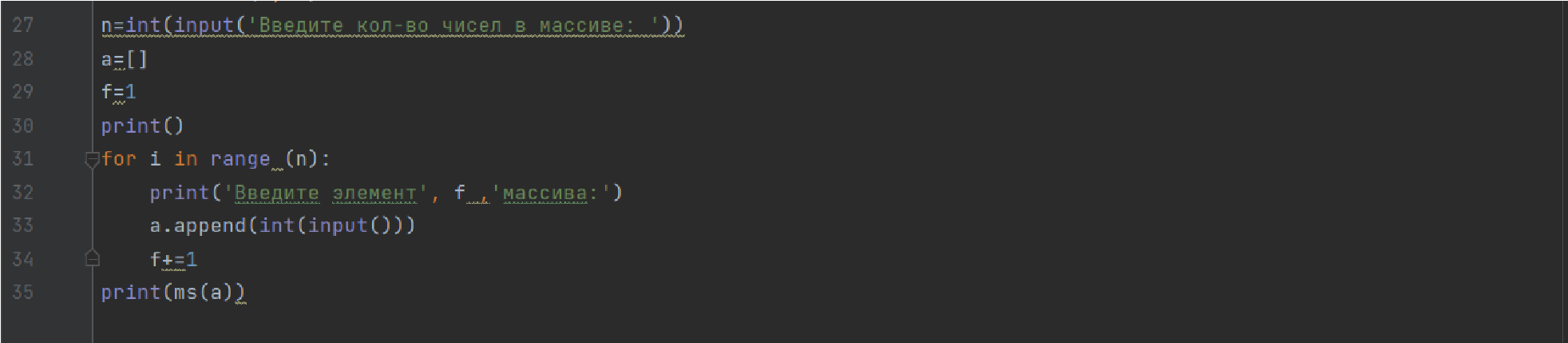
**O(n!)** (рис.6)**, O(n^3)** (рис.7)**, O(3log(n))** (рис.8).



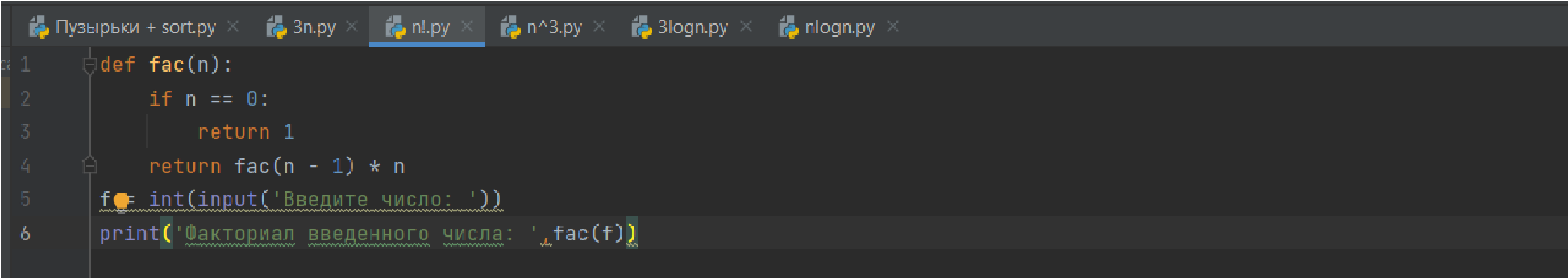
Данная программа создает **n** количество элементов для каждого из трех массивов.

4



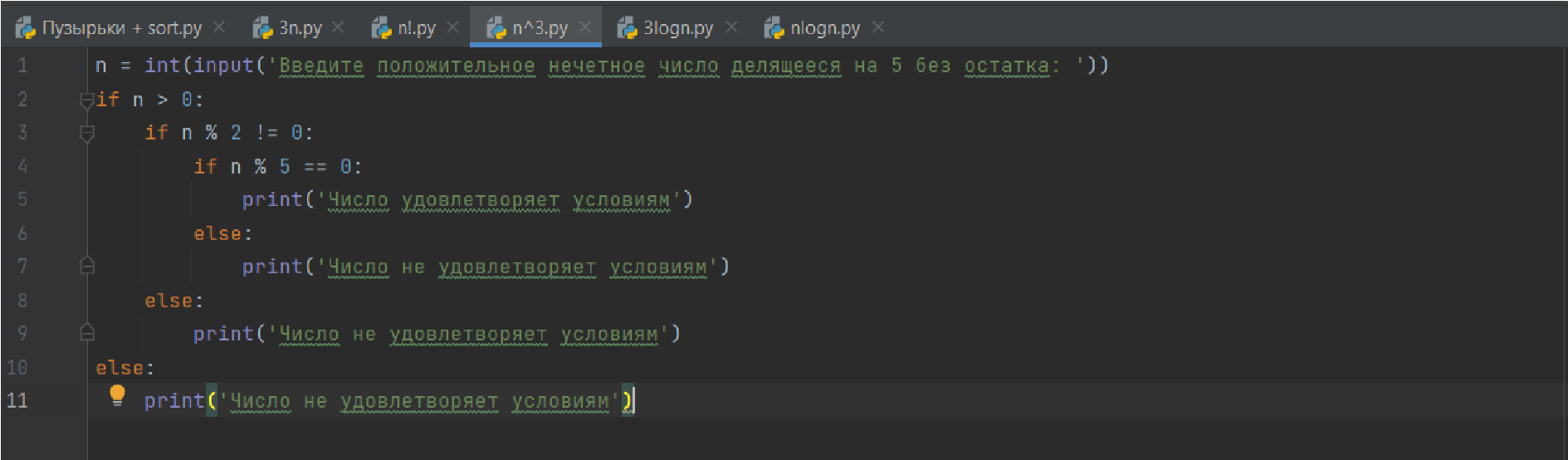


Merge sort (сортировка слиянием)

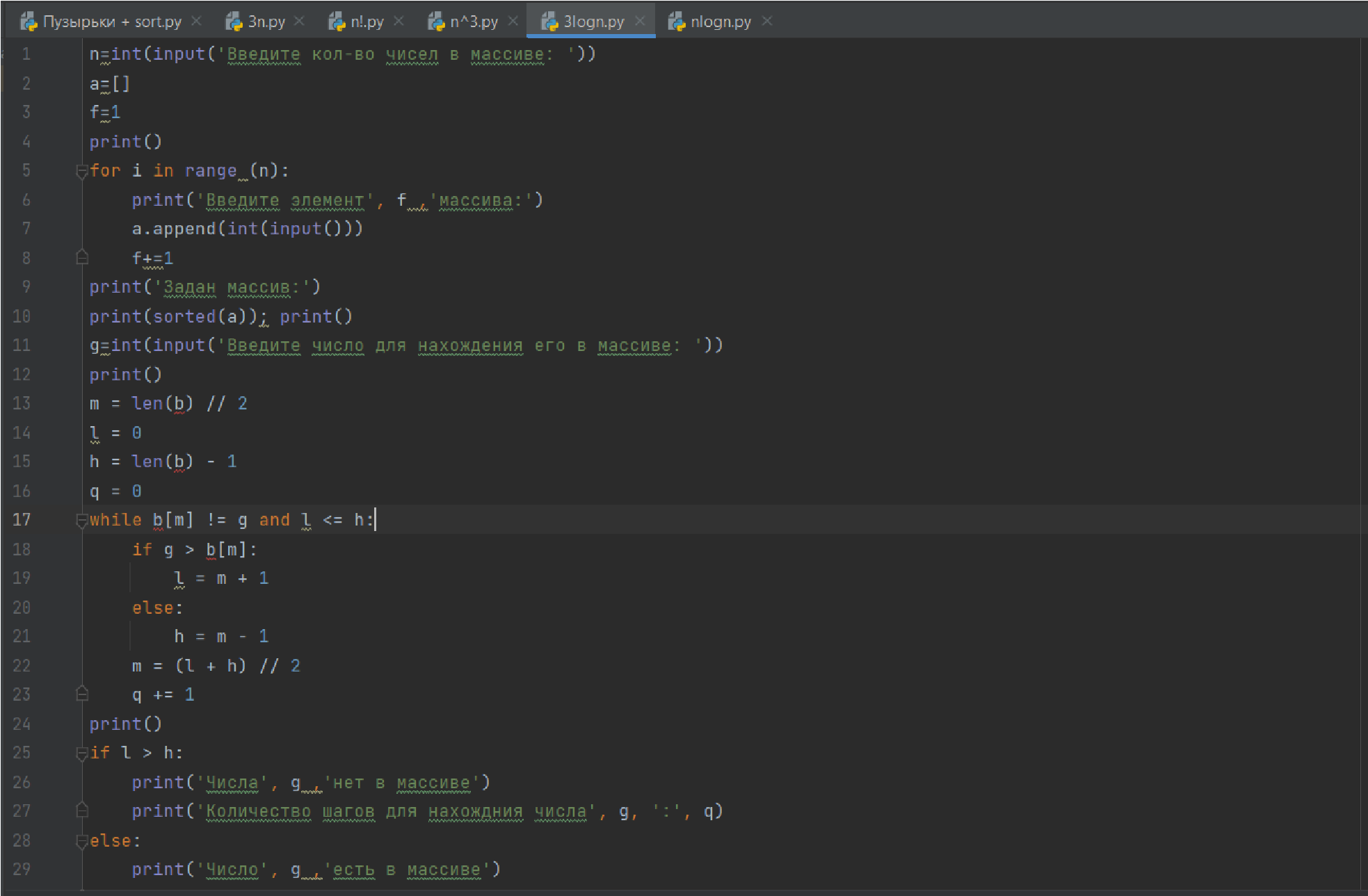


Программа вычисляет факториал для каждого числа в заданном массиве с помощью рекурсии.

5



Эта программа проверяет цифру на удовлетворение заданным условиям.



Программа трижды выполняет бинарный поиск

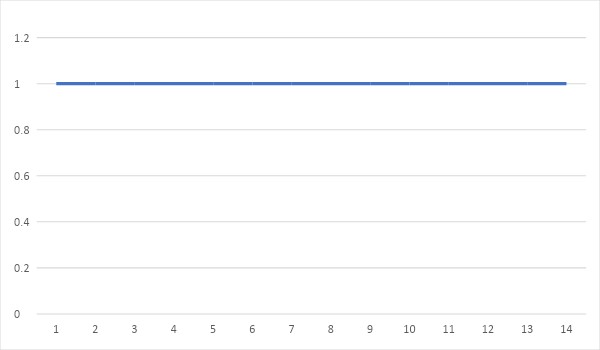
6

**Задание 3**

**“Построить зависимость между количеством элементом и количеством шагов для алгоритмов со сложностью O(1), O(log n), O(n^2), O(2^n). Сравнить сложность данных алгоритмов.”**

**Выполнение:**

Для наглядного понимания зависимости между количеством элементов и количеством шагов для алгоритмов со сложностью **O(1)** (рис.9), **O(log n)** (рис.10), **O(n^2)** (рис.11), **O(2^n)** были построили графики . Все графики приведены ниже.



В сложности **O(1)** количество шагов не зависит от количества переменных, это означает то, что при изменении количества переменных время выполнения программы никак не изменится.

7

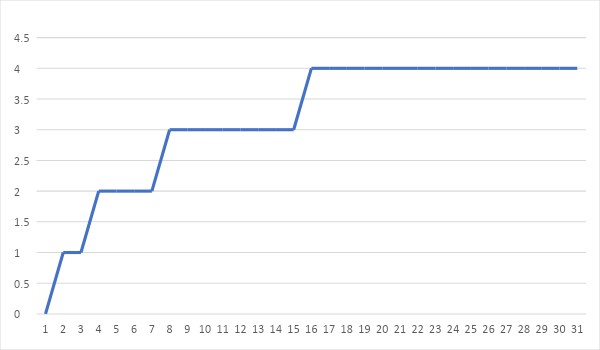


рис.10

В сложности **O(log n)** происходит отрицательное ускорение функции, следовательно данная сложность выполнения алгоритма самая оптимизированная, особенно, когда речь идет о большом количестве n.

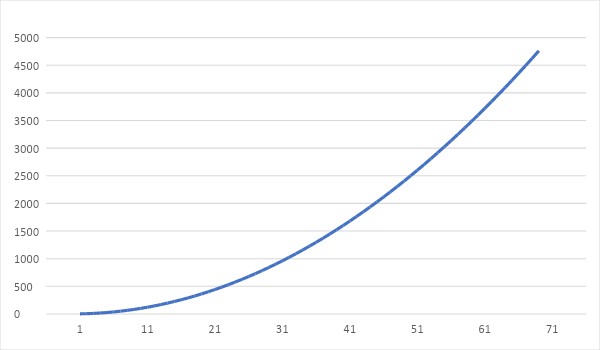


рис.11

В сложности **O(n^2)** количество шагов увеличивается полиномиально.

8

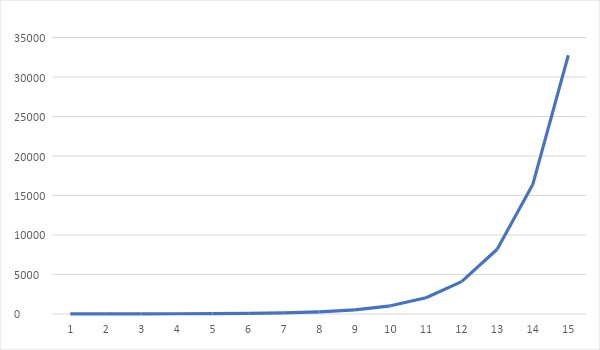


рис.12

В сложности **O(2^n)** количество шагов увеличивается экспоненциально.

**Сравнение:** Отталкиваясь от оценки всех вышеперечисленных сложностей, можно сказать, что **худшей** сложностью из приведенных является **O(2^n)**, а **лучшей** сложностью

**O(log n)**.9