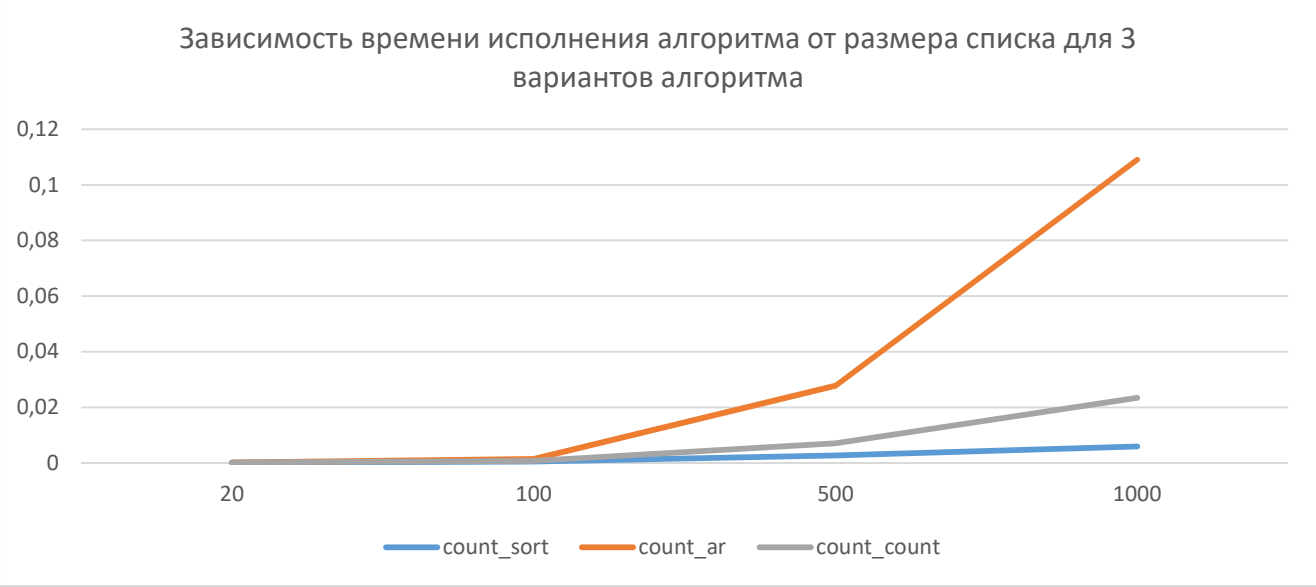
Домашнее задание. Урок 4.
Задание 1:

Проанализировать скорость и сложность одного любого алгоритма из разработанных в рамках домашнего задания первых трех уроков.
Примечание. Идеальным решением будет:
a. выбрать хорошую задачу, которую имеет смысл оценивать,
b. написать 3 варианта кода (один у вас уже есть),
c. проанализировать 3 варианта и выбрать оптимальный,
d. результаты анализа вставить в виде комментариев в файл с кодом (не забудьте указать, для каких N вы проводили замеры),
e. написать общий вывод: какой из трёх вариантов лучше и почему.

Я выбрала **задачу 4 из урока 3**: *Определить, какое число в массиве встречается чаще всего.*
3 варианта решения я разнесла по разным файлам (GB_1_1_1, GB_1_1_2, GB_1_1_3)
Код переделала:
- Обернула в функцию
- проверяла на массиве разного размера
- убрала все выводы (print())

Общие выводы о сложности и скорости алгоритмов представлены ниже.

Реализация	Отсортированный массив + цикл (count_sort)		Массив + цикл (count_ar)		Отсортированный массив + метод count	
Значение size	timeit	cProfile	timeit	cProfile	timeit	cProfile
20	$119 * 10^{-6}$	123	$162 * 10^{-6}$	111	$131 * 10^{-6}$	123
100	$554 * 10^{-6}$	540	$1{,}43 * 10^{-3}$	534	$723 * 10^{-6}$	566
500	$2{,}75 * 10^{-3}$	2515	$27{,}8 * 10^{-3}$	2515	$7{,}06 * 10^{-3}$	2734
1000	$5{,}91 * 10^{-3}$	5028	$5{,}91 * 10^{-3}$	5033	$23{,}4 * 10^{-3}$	5461
Сложность	линейная		степенная		степенная	



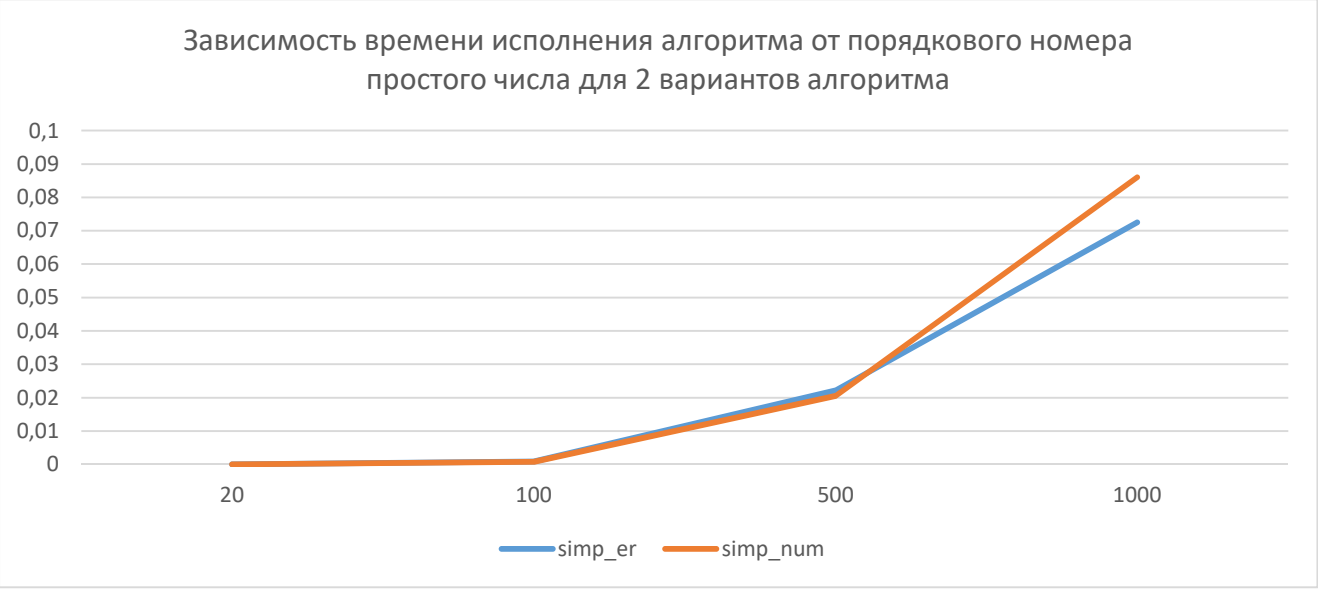
Выводы:
Нетрудно заметить, что первый алгоритм показал лучшие результаты в сравнении с другими. Это произошло потому, что движение по списку происходит только 1 раз – с начала и до конца. В других же случаях, каждый раз, когда считается количество вхождений нового числа, программа проверяет заново весь список. И при больших размерах списка время увеличивается значительно.

Задание 2:

Написать два алгоритма нахождения i-го по счёту простого числа. Функция нахождения простого числа должна принимать на вход натуральное и возвращать соответствующее простое число. Проанализировать скорость и сложность алгоритмов.
Первый — с помощью алгоритма «Решето Эратосфена».
Второй — без использования «Решета Эратосфена»

Я разделила на 2 файла: GB_2_1, GB_2_2

Реализация	Решето Эратосфена (simp_er)		«Не решето» (simp_num)	
Значение size	timeit	cProfile	timeit	cProfile
20	$44{,}1 * 10^{-6}$	56	$44{,}3 * 10^{-6}$	23
100	$843 * 10^{-6}$	353	$811 * 10^{-6}$	103
500	$22{,}2 * 10^{-3}$	2040	$20{,}5 * 10^{-3}$	503
1000	$72{,}5 * 10^{-3}$	4289	$86 * 10^{-3}$	1003
Сложность	степенная		степенная	



Вывод: сложность и время исполнения обоих алгоритмов приблизительно одинаковые. Однако на очень больших числах алгоритм с использованием решета Эратосфена показывает лучшие результаты. Лучше использовать его (не смотря на то, что код получился длиннее).