Задача №3 из урока “Массивы”:

В массиве случайных целых чисел поменять местами минимальный и максимальный элементы.

Python 3.7.4

OS: Linux, 64-bit

В коде решений создается массив (список) случайных чисел.

Разработаны варианты решений: с использованием списка, словаря и рекурсии.

В предложенных решениях создается рандомный список случайных чисел. Поиск выполняется по словарю, другому списку из двух элементов (сравниваются 2 значения, каждое из которых сохраняется как минимальное или максимальное, в зависимости от выполняемой в коде функции) или рекурсией, в которой по-одному отбрасываются значения, пока не останется всего 2, из которых сохраняется минимальное/максимальное – и функция возвращается к предыдущей итерации, чтобы закончить ее.
Итак, имеется:

- один изначальный список, содержащий один набор значений;
- переменные, которые ссылаются на значения из того же списка;
- переменные, содержащие индексы, которые необходимы для обмена значений.
- копии списков, уменьшаемые на 1 элемент при каждом “заходе” рекурсивной функции с теми же самыми значениями (т.е. память на эти значения уже затрачена и повторно их подсчитывать не нужно).

Я исключил первые числа от 0 до 255, которые статически хранятся в памяти компьютера.
На основе результатов (см. в таблице) выбирать стоит список, хотя разницу со случаем, когда используется словарь еще нужно подтвердить (статистически): количество затрачиваемой памяти у них растет линейно, в отличие от варианта с использованием рекурсии, в котором рост использования памяти носит геометрический характер.

Реализация обмена мест максимума и минимума	Рекурсия Lesson4_taskvar1.py			Цикл + словарь Lesson4_taskvar2.py			Цикл for + список Lesson4_taskvar3.py		
Значение n	timeit	cProfile	Memory, bytes	timeit	cProfile	Memory bytes	timeit	cProfile	Memory bytes
10	$15{,}4*10^{-6}$	$9*2^{*}$	1748	$12{,}4*10^{-6}$	1	932	$14{,}3*10^{-6}$	1	684
100	$181*10^{-6}$	$99*2^{*}$	51068	$110*10^{-6}$	1	4528	$120*10^{-6}$	1	4276
500	$1{,}36*10^{-3}$	$499*2^{*}$	1054412	$553*10^{-6}$	1	20688	$615*10^{-6}$	1	20444
1000	$4{,}33*10^{-3}$	$999*2^{*}$	4109180	$1{,}1*10^{-3}$	1	41444	$1{,}2*10^{-3}$	1	41188
10000	$497*10^{-3}$	$9999*2^{*}$	401087724	$11{,}2*10^{-3}$	1	408008	$12.1*10^{-3}$	1	407764
100000	-	-	-	$112*10^{-3}$	1	4024460	$121*10^{-3}$	1	4024252
1000000	-	-	-	1,12	1	40693560	1,22	1	40693520

* множитель указывает количество функций указанным количеством вызовов

Обзор результатов из прошлой работы (время работы алгоритмов):
Из результатов видно, что работа рекурсии занимает значительно больше времени, функции выполняются n-1 раз, а время работы можно выразить в виде $O(n^2)$: время выполнения кода растет на 1 порядок при увеличении числа элементов в 10 раз, а затем резко возрастает (10000 элементов) и достигает неразумных пределов уже на попытке выполнить операцию со 100 тысячами значений.

Цикл с использованием словаря или списка показали себя почти одинаково хорошо (словарь – чуть лучше, но имеющейся разницей можно пренебречь, пока не доказана ее статистическая достоверность). О-большое этих алгоритмов составляет O(n) – имеется почти линейная зависимость от количества элементов: при росте количества чисел в массиве в 10 раз, скорость работы снижается на 1 порядок.

Задача о поиске простого числа.

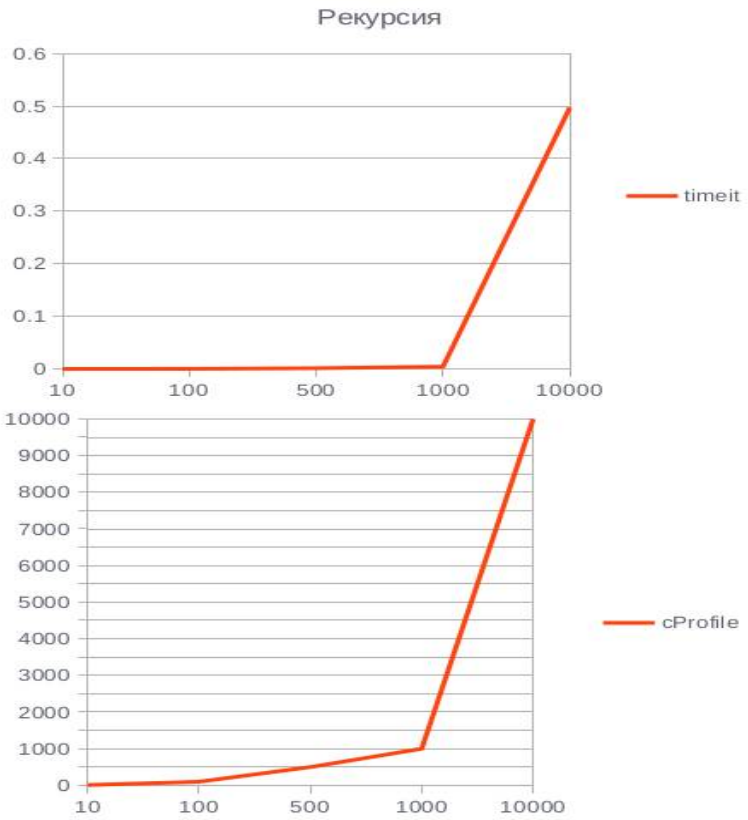
Решения: использование списков в двух вариантах (замена нулями ненужных значений и перебор всех чисел с определением возможности деления без остатка).

Реализация поиска простого числа	Список (заполнение нулями) файл simple_num2		Список (метод перебора) файл simple_num1	
Значение n	timeit	cProfile	timeit	cProfile
10	$1,9*10^{-6}$	3	$45,8*10^{-6}$	1
100	$17,5*10^{-6}$	4	$4,45*10^{-3}$	1
500	$101*10^{-6}$	4	$32,3*10^{-3}$	1
1000	$215*10^{-6}$	4	$415*10^{-3}$	1
10000	$2,48*10^{-3}$	5	97,6	1
100000	$27,7*10^{-3}$	5	-	-
1000000	$337*10^{-3}$	5	-	-

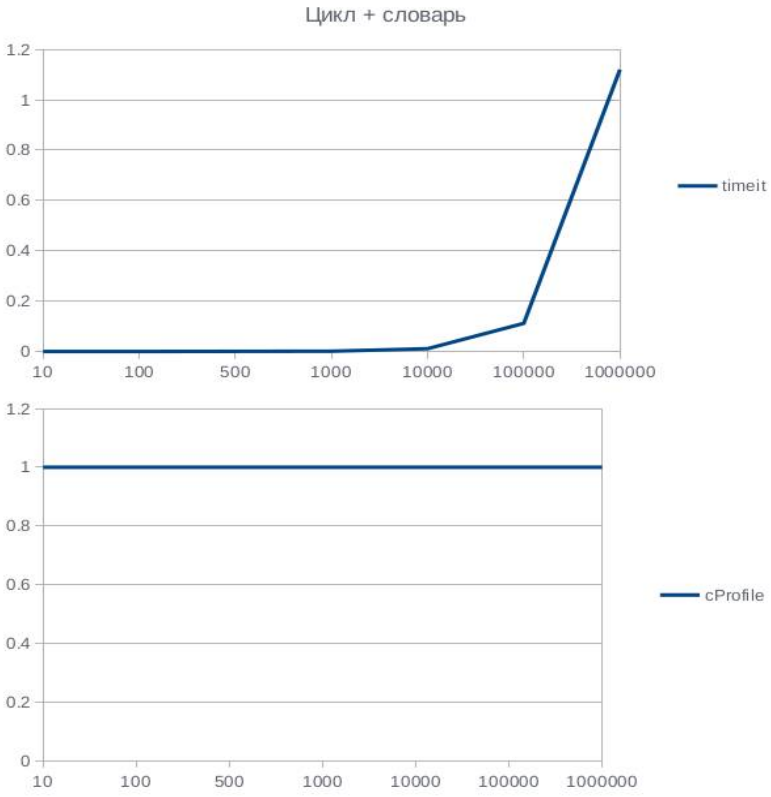
В задачах с поиском простого числа по номеру этого числа алгоритм, разработанный на уроке проявил себя лучше, чем второй, который выполняется методом перебора. Время выполнения задачи первого кода растет примерно на 1 порядок при увеличении номера искомого числа в 10 раз. А второй код растет примерно в 100 раз при увеличении номера элемента в 10 раз. Таким образом, время работы первого алгоритма выражается O(n), а второго – $O(n^2)$. При этом, количество вызовов функций в первом случае растет незначительно , а во втором – не растет вообще.

Задача 1. Графики

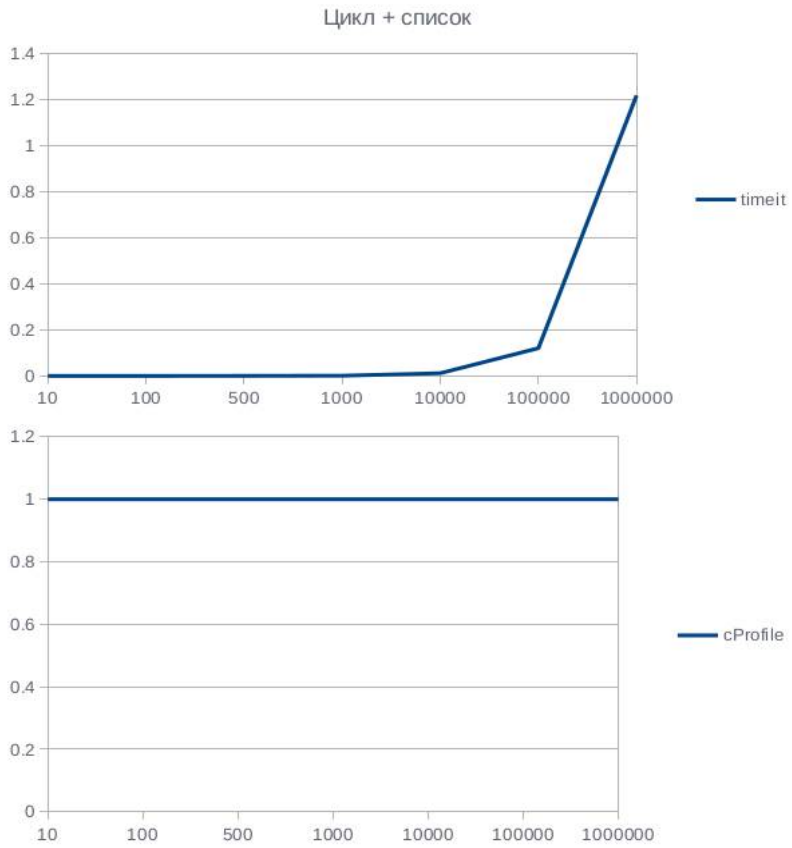
а)



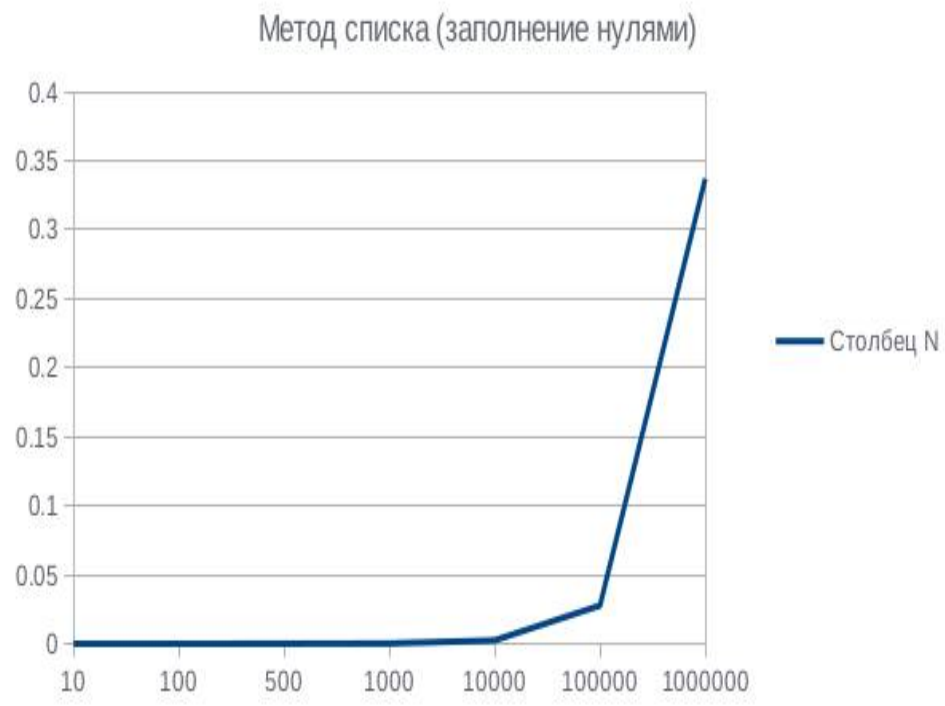
б)



в)


Цикл + список
1.4
1.2
1
0.8
0.6
0.4
0.2
0
10
100
500
1000
10000
100000
1000000
timeit
1.2
1
0.8
0.6
0.4
0.2
0
10
100
500
1000
10000
100000
1000000
cProfile

Задача 2


Метод списка (заполнение нулями)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10
100
500
1000
10000
100000
1000000
Столбец N


Метод списка (перебор значений)
120
100
80
60
40
20
0
10
100
500
1000
10000
Столбец P