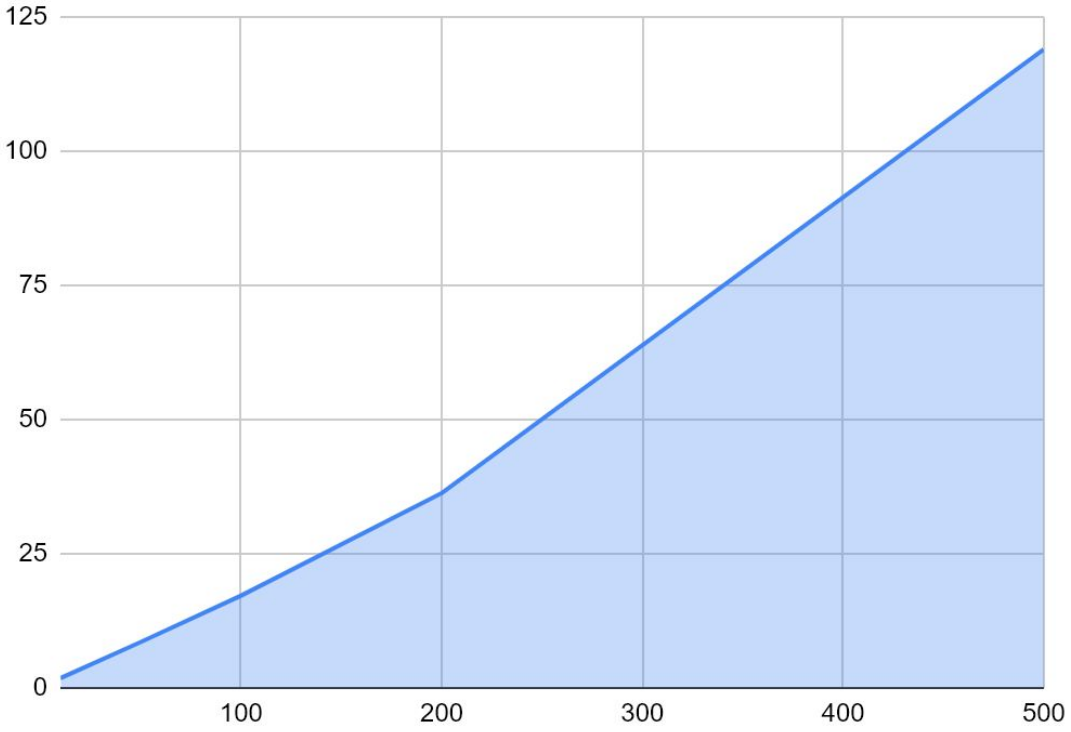
Результаты timeit и cProfile

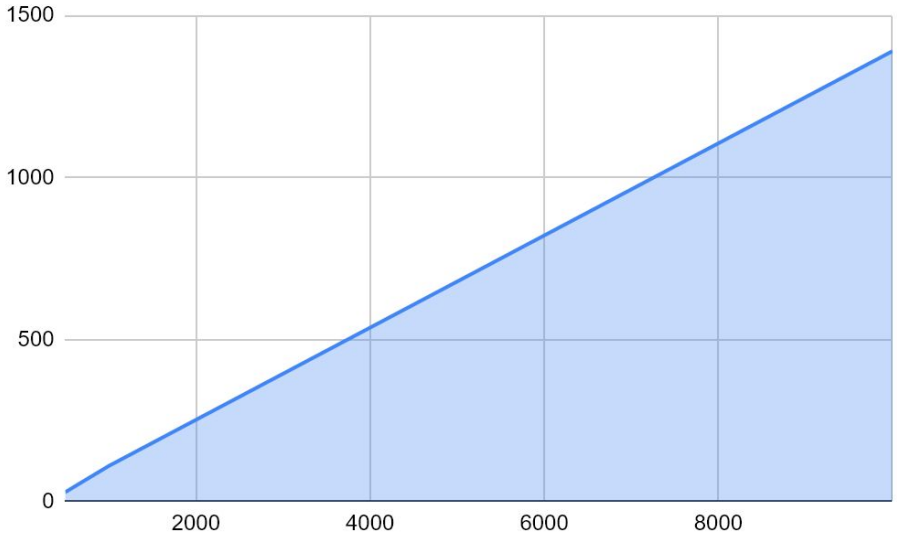
Реализация	Классическая рекурсия var1		проходим циклом по срезу [::-1] var2		функция reversed() var3		срез [::-1] var4		Рекурсия + мемоизация “из коробки” var5	
кол-во символов	timeit	cProfile	timeit	cProfile	timeit	cProfile	timeit	cProfile	timeit	cProfile
10	1,83 * 10-6	11	0.68 * 10-6	1	0.46 * 10-6	1	0.148 * 10-6	1	0.06 * 10-6	11
50	8,55* 10-6	51	2,69 * 10-6	1	1.17 * 10-6	1	0.18 * 10-6	1	0.063 * 10-6	51
100	17,2 * 10-6	101	5,05 * 10-6	1	1.95 * 10-6	-	0.23 * 10-6	1	0.062 * 10-6	101
200	36,3 * 10-6	201	-	-	-	-	-	-	0.061 * 10-6 (0.27 * 10-6 max)	201
400	-	-	-	-	-	-	-	-	0.061 * 10-6 (0.57 * 10-6 max)	401
500	119 * 10-6	501	29 * 10-6	1	7.82 * 10-6	-	0.63 * 10-6	1	-	496 ошибка
1000	-	-	110 * 10-6	1	15.2 * 10-6	-	1.06 * 10-6	1	-	-
10000	-	-	1390* 10-6	1	119 * 10-6	1	7.07 * 10-6	1	-	-

Выводы: ниже.

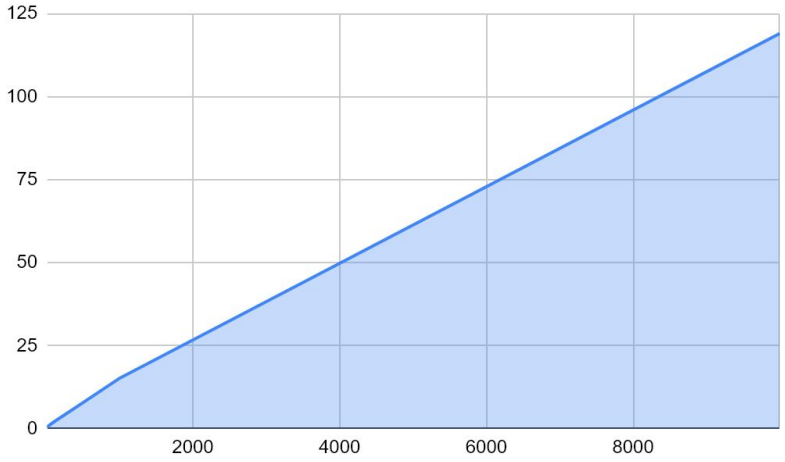
1. Классическая рекурсия var1, сложность алгоритма линейная O(n), но по факту выполняется медленнее чем цикл по срезу в 4 раза (119/29), медленнее в 15 раз медленнее reverse(), медленнее в 188 раз чем срез, так же рекурсия имеет существенный недостаток - ограничение в 1000 вызовов.



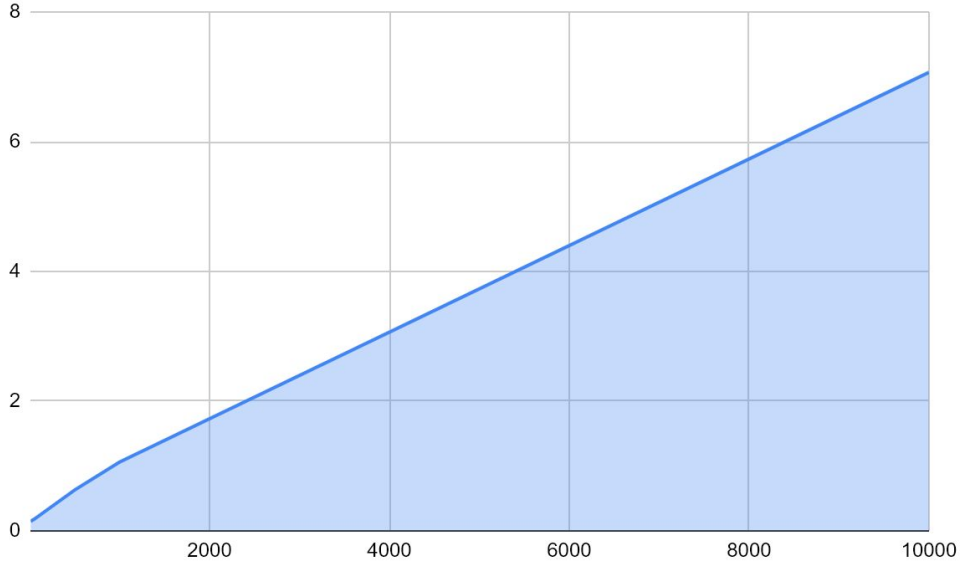
2. Цикл по срезу [::-1] var2, сложность алгоритма линейная O(n), (для 10000 символьной строки) по факту выполняется медленнее в 11,5 раз чем reversed(), и в 196 раз медленнее чем просто срез. (хотя бессмысленный код, так как можно взять просто срез (var4), просто было любопытно как работает)



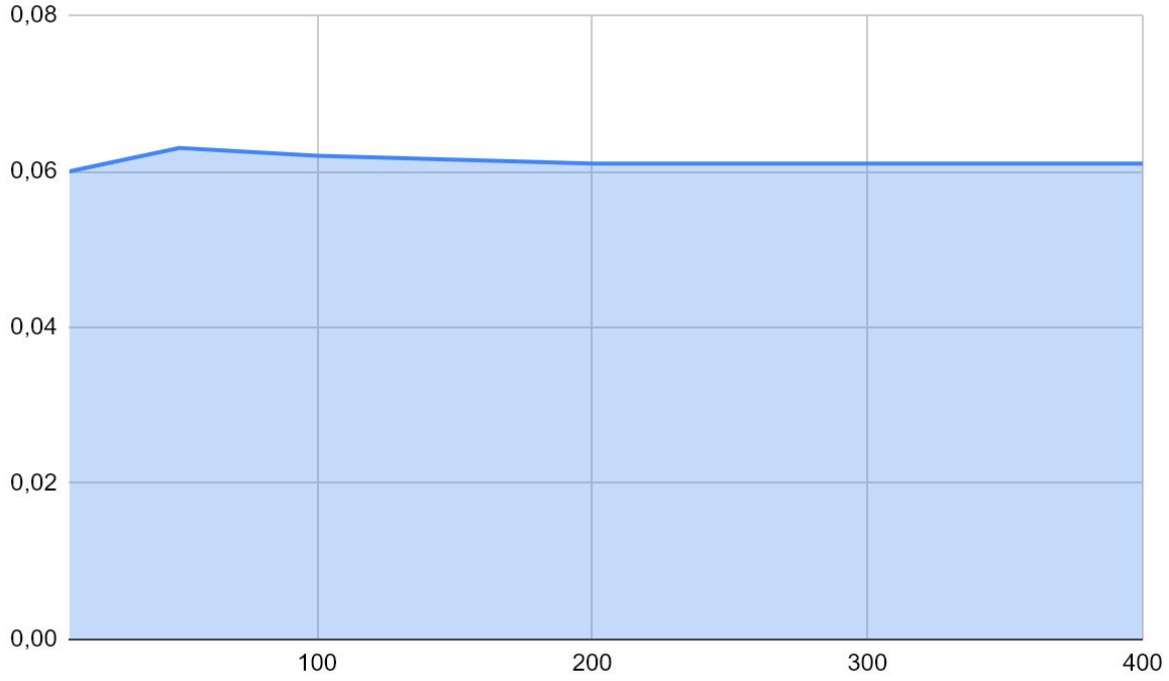
3. функция reversed() var3, сложность алгоритма линейная O(n), (для 10000 символьной строки) выполняется достаточно быстро, по факту медленнее, чем срез в 16,8 раз



4. срез [::-1] var4, сложность алгоритма линейная O(n), (для 10000 символьной строки) выполняется относительно быстро, самый короткий код. По результатам быстрее чем срез работает Рекурсия + мемоизация “из коробки” var5, но к этому последнему варианту много вопросов.



5. Рекурсия + мемоизация “из коробки” var5, хоть и наследует существенный недостаток от обычной рекурсии (ограничение на переполнение стека вызовов 1000), однако сложность алгоритма получается константная O(1) (для пустой строки выдает такое же время как и для максимально возможной, до ограничения рекурсии 1000). Кроме того функция проходит все тесты, поэтому я не сомневаюсь в корректности ее работы. При длине строки более 200 символов при проверке функцией timeit выскакивает предупреждение, что среднее (например для 400 символов 61.2 nsec) и максимальное (для 400 символов 566 nsec) время расходится в разы. Возможно это влияние декоратора @functools.lru_cache(). Однако даже максимальное время выполнения очень неплохое, по сравнению с остальными реализациями задачи.



Выводы: самые оптимальные по времени выполнения и величине кода оказались варианты срез [::-1] var4 и Рекурсия + мемоизация “из коробки” var5. Однако var5 можно использовать, когда точно уверен, что количество вызовов функции будет менее 1000. Если нет в этом уверенности, то для данного условия задачи следует использовать var4 срез [::-1].