Эмпирическая оценка сложности алгоритма

Задача1:

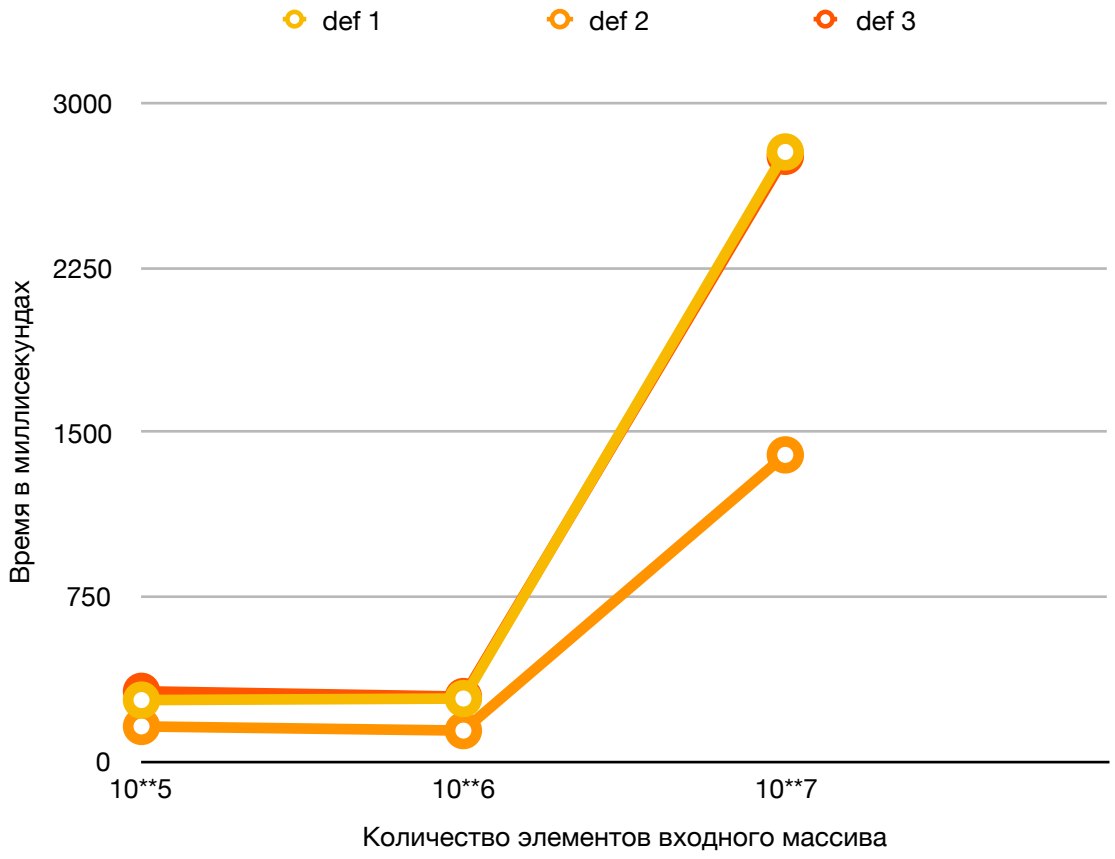
Решаем 5 задачу из третьей домашней работы:

“В массиве найти максимальный отрицательный элемент. Вывести на экран его значение и позицию в массиве.”

Есть три варианта решения задачи.

Воспользовавшись модулем cProfile, видим, что значения по времени получаем видимыми начиная с n ~ 10000, и с небольшой разницей (3мс и 2мс), достигает разницы в секунду с n = 10**7

	n, размер массива			
	10**4	10**5	10**6	10**7
def_1	3	280	287	2777
def_2	2	160	141	1397
def_3	3	320	296	2757
	Время на выполнение, в миллисекундах			



Как видно из графика, определить сложность алгоритма исходя только из времени действия алгоритма было бы недостаточно: сложность нелинейная, но схожая асимптотическая и разница по времени не фееричная: “быстрый” алгоритм не более чем в два раза быстрее, и два решения совпадают по времени. Поэтому посмотрим на количество вызовов функций, там ситуация более динамичная начиная с n = 10.

	Количество элементов входного массива		
Варианты решения Task5, HW_3	**10**	**100**	**1000**
def search_max_negative_1(list_init)	10 function calls in 0.000 seconds	79 function calls in 0.000 seconds	1050 function calls in 0.001 seconds
def search_max_negative_2(list_init)	6 function calls in 0.000 seconds	7 function calls in 0.000 seconds	12 function calls in 0.000 seconds
def search_max_negative_3(list_init)	9 function calls in 0.000 seconds	44 function calls in 0.000 seconds	533 function calls in 0.000 seconds

Видно, что первый алгоритм (**def_1**, def search_max_negative_1(list_init)) имеет самое большое количество вызовов функций из предоставленных решений, увеличивая на порядок количество вводимых данных, увеличивается на порядок так же количество вызовов, из чего следует, что асимптотическая сложность алгоритма линейная — O(n), **~n**

Третий алгоритм (**def_3**, def search_max_negative_3(list_init)), работает в два раза экономнее, но зависимость так же линейная, **~1/2*n**

Вторй алгоритм самый оптимальный (**def_2**, def search_max_negative_2(list_init)), количество вызовов функций линейно зависит от объема данных, но зависит куда экономичнее первых двух: **~ n/10**2**

Затратные места решений:

Положим n = 1000, тогда:

def_1, def search_max_negative_1(list_init)):
built-in method builtins.abs - 504 из 1005 function calls
method 'append' of 'list' objects - 496 из 1005 function calls
То что подлежит доработке в первом варианте решения это операции со взятием модуля и с добавлением элементов в список.

def_2, def search_max_negative_2(list_init)):
index_max_negative = list(i for i, e in enumerate(list_init) if e == max_negative)[0]
Поиск индекса максимального отрицательного числа во втором решении самое слабое место (5 из 9 function calls)

def_3, def search_max_negative_3(list_init)):
method 'append' of 'list' objects - самое слабое место, 496 из 506 function calls