Алгоритмы и структуры данных на Python

Урок 4

Эмпирическая оценка алгоритмов на Python

Измерения времени работы с использованием timeit. Профайлер.

План

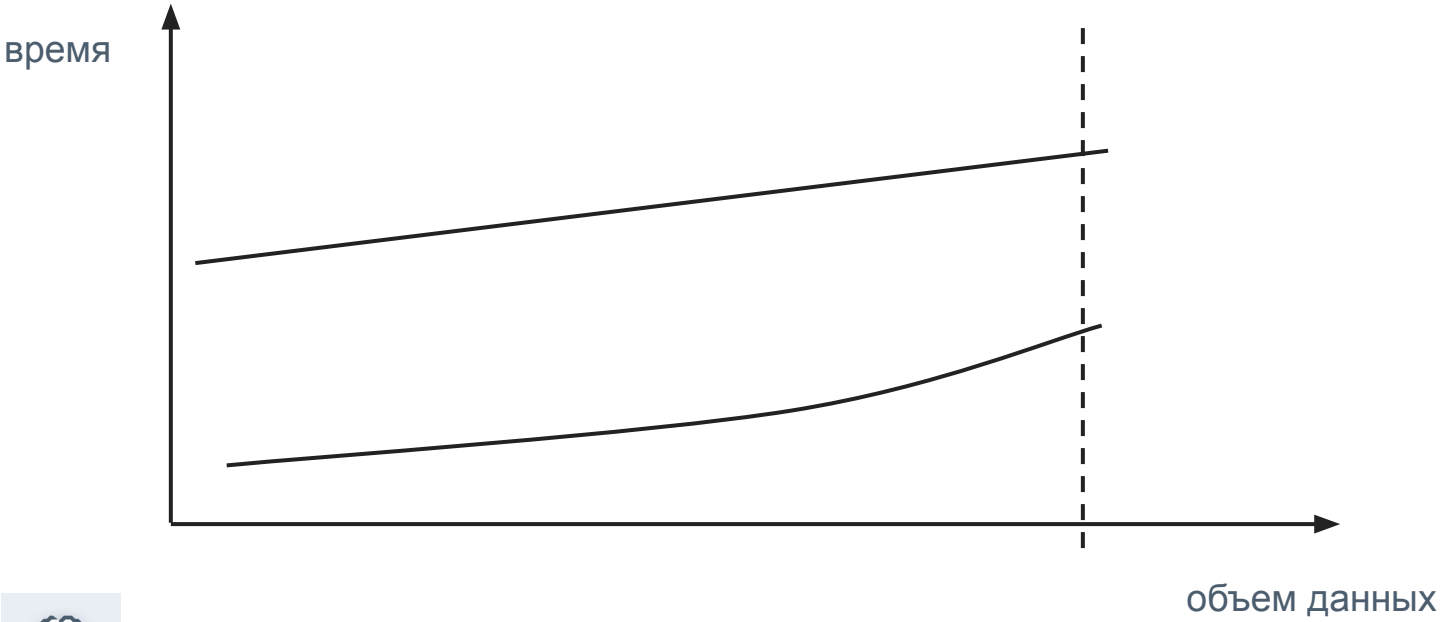
- Что такое сложность алгоритма?
- Классификация алгоритмов по сложности
- Измерения времени работы с использованием **timeit**
- Измерения времени работы с использованием **cProfile**

Что такое сложность алгоритма?

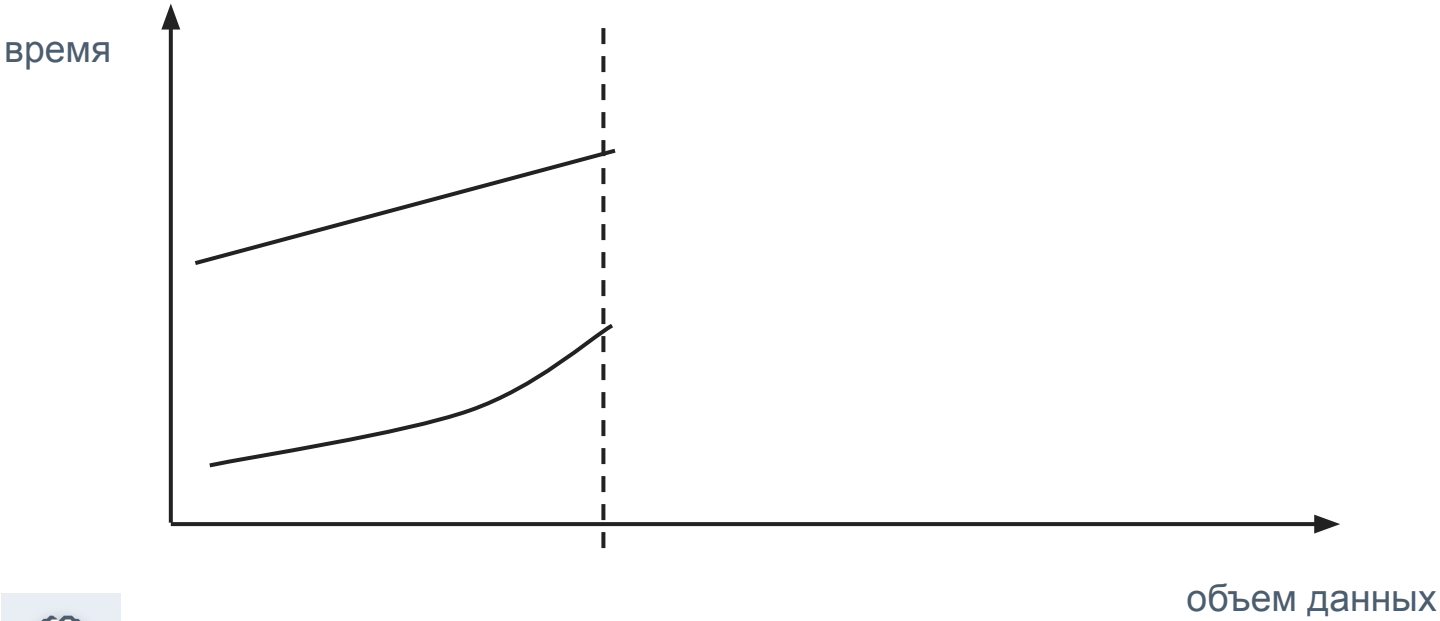
Временная сложность алгоритма определяет время работы, используемое алгоритмом, как функции от длины строки, представляющей входные данные.

$$t = f(n)$$

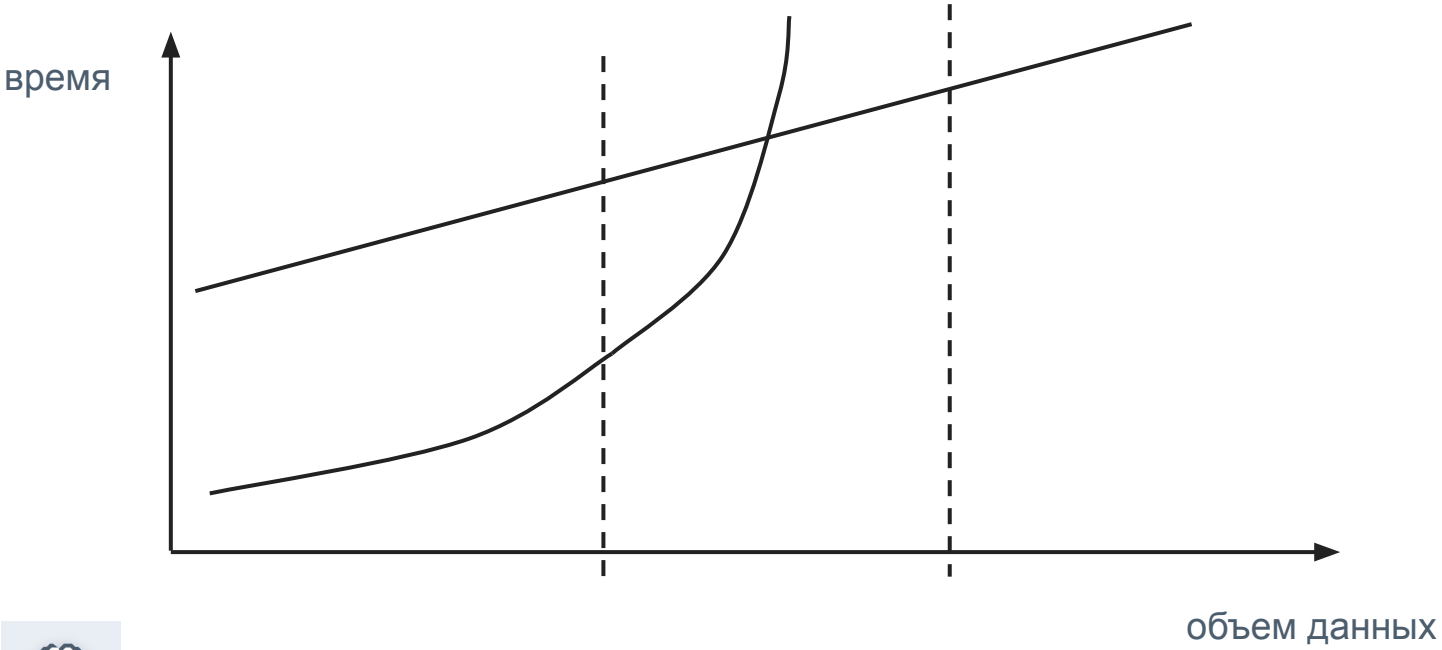
Зачем нужно знать сложность алгоритма



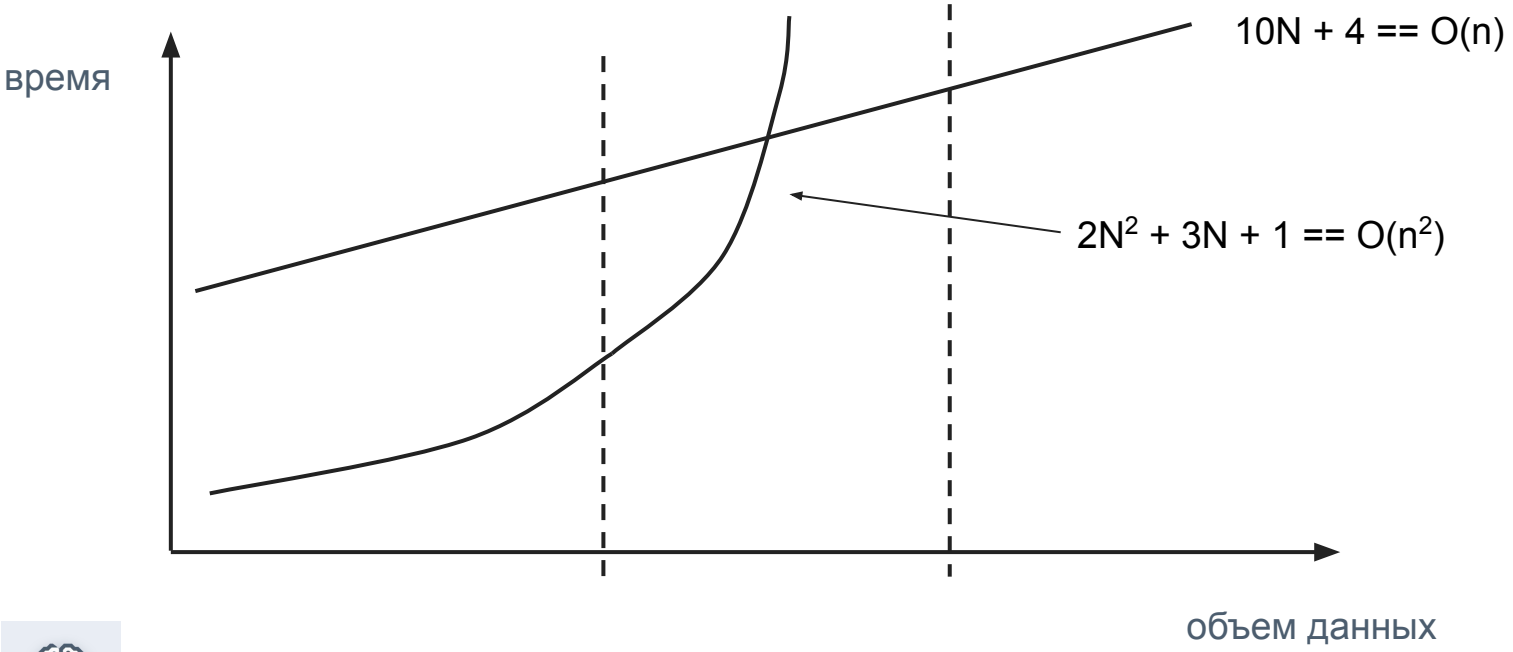
Зачем нужно знать сложность алгоритма



Зачем нужно знать сложность алгоритма



Зачем нужно знать сложность алгоритма



Классификация сложности алгоритмов

Название	Обозначение	Пример
Постоянная сложность	O(1)	Определение чётности числа
Логарифмическая сложность	O(log n)	Двоичный (бинарный) поиск
Линейно-логарифмическая сложность	O(n log n)	Быстрая сортировка Хоара (среднее время)

Классификация сложности алгоритмов

Название	Обозначение	Пример
Линейная сложность	$O(n)$	Поиск наименьшего элемента в неотсортированном массиве
Квадратичная сложность	$O(n^2)$	Сортировка пузырьком
Экспоненциальная сложность	$O(2^n)$	Решение задачи о рюкзаке методом прямого перебора

Эмпирическая оценка алгоритмов

- Измерения времени работы с использованием **timeit**

Эмпирическая оценка алгоритмов

- Измерения времени работы с использованием **cProfile**

Итоги:

Теория

- Что такое сложность алгоритма?
- Классификация алгоритмов по сложности

Практика

- Использование timeit и cProfile для оценки времени выполнения программы

План

- Оптимизация алгоритма на примере чисел Фибоначчи
- Практическое использование **timeit** и **cProfile**

Числа Фибоначчи

индекс	0	1	2	3	4	5	6	7	8	9
значение	0	1	1	2	3	5	8	13	21	34

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n \geq 2, \; n \in Z$$

Итоги:

Практика

- Написание теста для проверки работы функции
- Поиск чисел Фибоначчи при помощи рекурсии
- Использование timeit и cProfile для оценки времени выполнения программы

План

- Продолжение оптимизация алгоритма на примере чисел Фибоначчи
- Использование технологии мемоизации (memoization)
- Практическое использование **timeit** и **cProfile**

Мемоизация

Мемоизация (memoization) - сохранение результатов выполнения функций для предотвращения повторных вычислений.

Результаты timeit и cProfile

Реализация	Классическая рекурсия		Рекурсия + словарь		Рекурсия + список	
Значение **n**	timeit	cProfile	timeit	cProfile	timeit	cProfile
10	$19{,}3 \times 10^{-6}$	177				
15	206×10^{-6}	1973				
20	$2{,}31 \times 10^{-3}$	21894				
25	$27{,}4 \times 10^{-3}$	242785				

Результаты timeit и cProfile

Реализация	Классическая рекурсия		Рекурсия + словарь		Рекурсия + список	
Значение **n**	timeit	cProfile	timeit	cProfile	timeit	cProfile
10	$19{,}3 \times 10^{-6}$	177	$3{,}5 \times 10^{-6}$	19	$8{,}31 \times 10^{-6}$	19
15	206×10^{-6}	1973	$6{,}21 \times 10^{-6}$	-		-
20	$2{,}31 \times 10^{-3}$	21894	$6{,}66 \times 10^{-6}$	39	$9{,}57 \times 10^{-6}$	39
25	$27{,}4 \times 10^{-3}$	242785	-	-	-	-
100	-	-	$39{,}5 \times 10^{-6}$	199	$37{,}9 \times 10^{-6}$	199
200	-	-	$79{,}5 \times 10^{-6}$	-	$74{,}4 \times 10^{-6}$	-
500	-	-	231×10^{-6}	999	202×10^{-6}	999

Итоги:

Практика

- Поиск чисел Фибоначчи при помощи рекурсии
- Мемоизация в словарь (`dict`) и в список (`list`)
- Использование **timeit** и **cProfile** для оценки времени выполнения программы

План

- Продолжение оптимизация алгоритма на примере чисел Фибоначчи
- Реализация задачи с использованием цикла
- Мемоизация при помощи библиотеки **functools**
- Практическое использование **timeit** и **cProfile**

Результаты timeit и cProfile

Реализация	Классическая рекурсия		Рекурсия + словарь		Рекурсия + список		Цикл		Рекурсия + мемоизация “из коробки”	
Значение **n**	timeit	cProfile	timeit	cProfile	timeit	cProfile	timeit	cProfile	timeit	cProfile
10	$19{,}3 \times 10^{-6}$	177	$3{,}5 \times 10^{-6}$	19	$8{,}31 \times 10^{-6}$	19	$0{,}678 \times 10^{-6}$	1	$0{,}0977 \times 10^{-6}$	11
15	206×10^{-6}	1973	$6{,}21 \times 10^{-6}$	-	-	-	-	-	-	-
20	$2{,}31 \times 10^{-3}$	21894	$6{,}66 \times 10^{-6}$	39	$9{,}57 \times 10^{-6}$	39	-	-	-	-
25	$27{,}4 \times 10^{-3}$	242785	-	-	-	-	-	-	-	-
100	-	-	$39{,}5 \times 10^{-6}$	199	$37{,}9 \times 10^{-6}$	199	$4{,}8 \times 10^{-6}$	1	$0{,}137 \times 10^{-6}$	101
200	-	-	$79{,}5 \times 10^{-6}$	-	$74{,}4 \times 10^{-6}$	-	-	-	$0{,}102 \times 10^{-6}$	201
500	-	-	231×10^{-6}	999	202×10^{-6}	999	$27{,}3 \times 10^{-6}$	1	-	-
50000	-	-	-	-	-	-	$25{,}1 \times 10^{-3}$	1	-	-

Итоги:

Практика

- Поиск чисел Фибоначчи при помощи цикла
- Мемоизация при помощи библиотеки **functools**
- Использование **timeit** и **cProfile** для оценки времени выполнения программы

Домашнее задание

1. Проанализировать скорость и сложность алгоритмов, разработанных в рамках домашнего задания из первых трех уроков.

Домашнее задание

- Выберите любые 3 задачи
- Измерьте время работы вашего кода при помощи **timeit** и **cProfile**
- Результаты измерений сохраните в файл с кодом в виде комментариев
- При необходимости, измените исходной код

Домашнее задание

2. Написать два алгоритма нахождения i-го по счёту простого числа.

- использовать алгоритм решето Эратосфена.
- без использования "решета".

Проанализировать скорость и сложность алгоритмов.

План

- Разбор домашнего задания