Курс “Алгоритмы на python”

Занятие #1
Введение в алгоритмы

Сентябрь 2025

Avito

Виктория Мунькина

Senior LLMOps

- Alma mater: ВМК МГУ
- LLM платформа Авито

- по всем вопросам: @v_stepanischeva

Единая точка входа – степик

- https://stepik.org/course/251189/

Вопросы и обсуждения – чат (когда создадут)

Посещаемость

Структура курса «Алгоритмы на питоне»

1 модуль

- Введение в алгоритмы
- Базовые структуры данных
- Рекурсия
- Сортировки
- Кучи
- Бинарные деревья поиска
- Хеш-таблицы

2 модуль

- Графы
- Динамическое программирование
- Алгоритмы в строках
- Алгоритмы в ML и LLM
- Итоговый контест

Как устроен курс?

0. Важно ли ходить на занятия? И что делать, если пропустил
   да
1. Когда лучше задавать вопросы по ходу или после?
   не принципиально
2. Как аттестуем по курсу – что делать, чтобы успешно его завершить? Что ждет в конце?
   0.5 домашки + 0.4 итоговая работа + 0.1 вовлеченность
3. Как чередуем лекции и практику?
   не чередуем
4. Где будут храниться материалы?
   степик
5. Сколько будет домашки?
   3 задачи после каждой лекции
6. Как сдавать домашку?
   папка в репозитории гитхаба

Чуть больше про домашнее задание

- 3 задачи после каждой встречи
- решения складываем в репозиторий
- по готовности – на степике отправляем ссылку на соответствующую папку (там есть форма)
- дата отправки ссылки – дата сдачи дз

Чуть больше про домашнее задание

- максимум 2 балла за одну задачу, 6 баллов за одну домашнюю работу
- для каждой задачи пишем тесты
- качество кода и качество тестов влияет на балл по задаче

Чуть больше про домашнее задание

- при сдаче в первые две недели после получения штраф 0 баллов;
- при сдаче в течение 3-4 недель после получения штраф 1 балл
- при сдаче позже, чем через 4 недели задание не принимается
- на академическую честность – смотрим

Правила курса

no fighting

Правила курса

Задаем вопросы

Не боимся ошибок

План занятия

- Часть I. Знакомство
- Часть II. Что такое алгоритм
- Часть III. Анализ алгоритмов
- Часть IV. Бенчмарки

1 Введение в алгоритмы

Какие алгоритмы знаете?

Какие структуры данных знаете?

Ожидания от курса

Зачем нужны алгоритмы?

Что такое алгоритм?

Пошаговая инструкция решения некоторой задачи, реализованная в виде программы

Под алгоритмом в математике понимают точное предписание, задающее вычислительный процесс, ведущий от начальных данных, которые могут варьироваться, к искомому результату.

Свойства:
конечность
детерминированность
массовость

Как оценить эффективность алгоритма?

Подсчитать количество вычислительных операций?

Как это сделать в Python?

Python – интерпретируемый язык программирования

Python –
интерпретируемый язык
программирования

код программы -> байткод

Python –
интерпретируемый язык
программирования

код программы -> байткод

программа-интерпретатор
разбирает и выполняет байткод

Python –
интерпретируемый язык
программирования

код программы -> байткод

программа-интерпретатор
разбирает и выполняет байткод

программа-интерпретатор
написана на C и скомпилирована

Python – интерпретируемый язык программирования

код программы -> байткод

программа-интерпретатор разбирает и выполняет байткод

программа-интерпретатор написана на C и скомпилирована

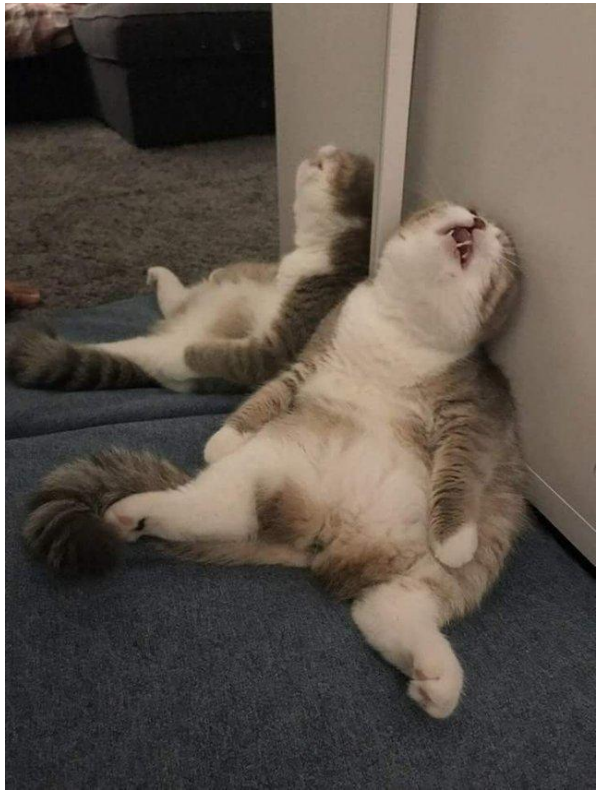
есть встроенные функции
написаны на C,
скомпилированы
в машинные
инструкции
min()
max()

```
>>>
>>> def custom_sum(values: list):
...     total = 0
...     for elem in values:
...         total += elem
...     return total
...
>>>
>>> dis.dis(custom_sum)
  1           RESUME                   0

  2           LOAD_CONST               1 (0)
              STORE_FAST               1 (total)

  3           LOAD_FAST                0 (values)
              GET_ITER
        L1:   FOR_ITER                 7 (to L2)
              STORE_FAST               2 (elem)

  4           LOAD_FAST_LOAD_FAST     18 (total, elem)
              BINARY_OP               13 (+=)
              STORE_FAST               1 (total)
              JUMP_BACKWARD            9 (to L1)

  3   L2:     END_FOR
              POP_TOP

  5           LOAD_FAST                1 (total)
              RETURN_VALUE
>>>
```

```
>>> def custom_sum_2(values: list):
...     return sum(values)
...
>>> dis.dis(custom_sum_2)
  1           RESUME                   0

  2           LOAD_GLOBAL              1 (sum + NULL)
              LOAD_FAST                0 (values)
              CALL                     1
              RETURN_VALUE
>>>
```

К чему это все?

Реальное количество операций, выполняемое процессором – **сложно и зачем?**



Просто – исследовать зависимость количества базовых операций от входных данных

Настало время формализовать

Сложность по времени

Алгоритм
`A: input -> output`

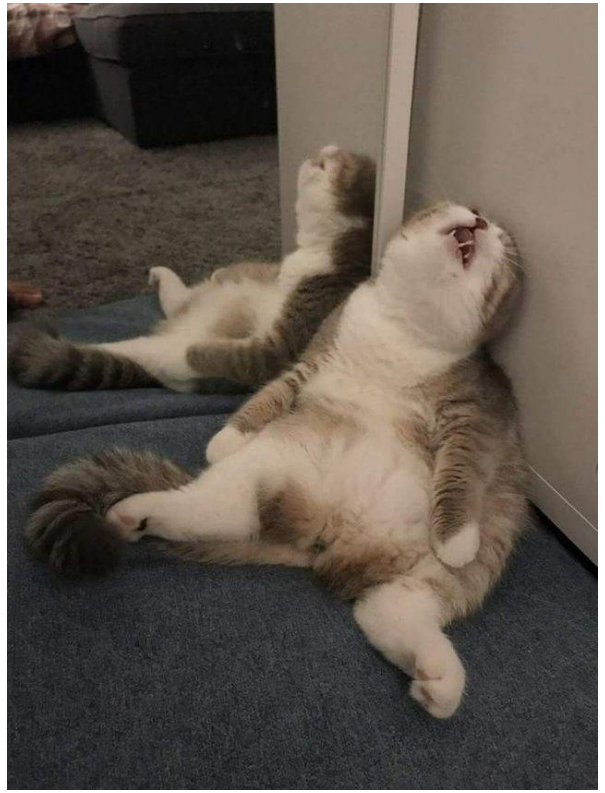
Задача оценки сложности по времени – получить зависимость количества базовых инструкций от размера набора входных данных

Количество базовых операций на заданных входных данных – `T(N)`

Сложность по памяти

Сколько дополнительной памяти потребуется для обработки входных данных размером N

Точные функции зашумлены константами и младшими членами



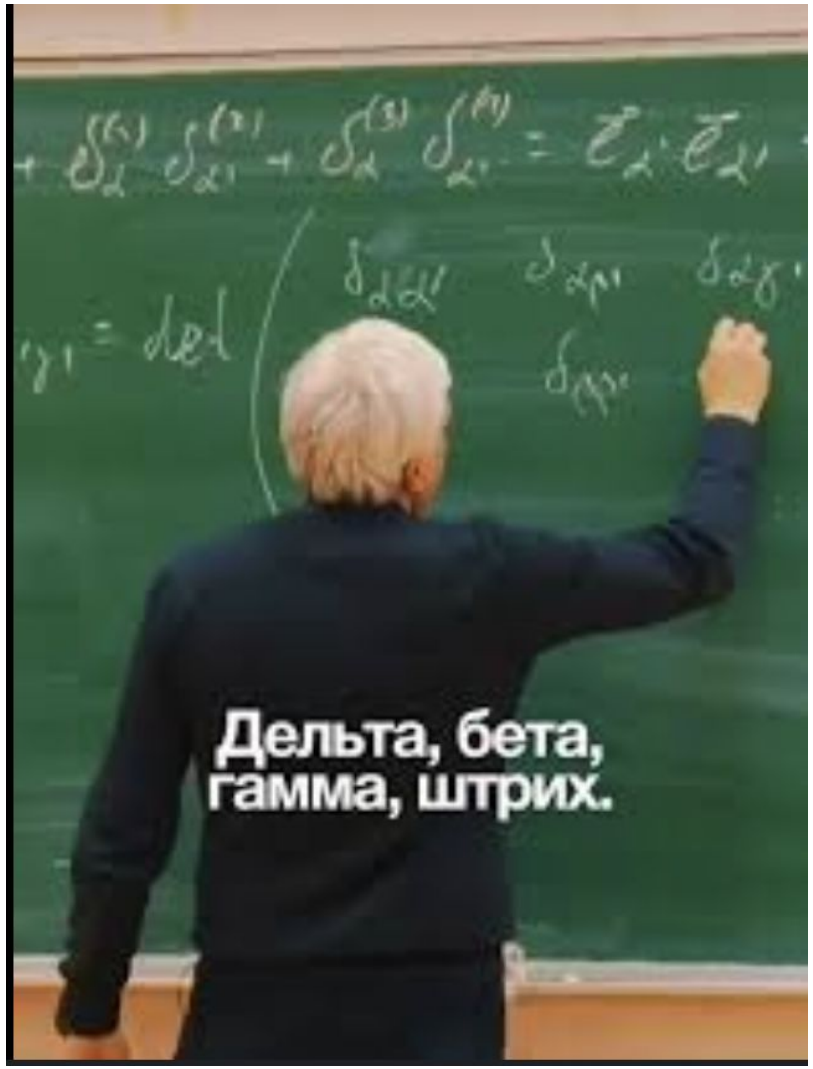
Лучше оценим функцию

Настало время асимптотических оценок

Что важно?

Скорость, с которой функция растет при росте объема входных данных

Чуть матана



Асимптотические оценки

Обозначение	Интуитивное объяснение	Определение
$f(n) \in O(g(n))$	f ограничена сверху функцией g (с точностью до постоянного множителя) асимптотически	$\exists(C > 0), n_0 : \forall(n > n_0)\ \lvert f(n)\rvert \leq \lvert Cg(n)\rvert$ или $\exists(C > 0), n_0 : \forall(n > n_0)\ f(n) \leq Cg(n)$
$f(n) \in \Omega(g(n))$	f ограничена снизу функцией g (с точностью до постоянного множителя) асимптотически	$\exists(C > 0), n_0 : \forall(n > n_0)\ \lvert Cg(n)\rvert \leq \lvert f(n)\rvert$
$f(n) \in \Theta(g(n))$	f ограничена снизу и сверху функцией g асимптотически	$\exists(C, C' > 0), n_0 : \forall(n > n_0)\ \lvert Cg(n)\rvert \leq \lvert f(n)\rvert \leq \lvert C'g(n)\rvert$

Асимптотические оценки худший случай

Строим функцию `g(N)` – оценивает количество выполняемых действий на наборе данных размером N в худшем случае

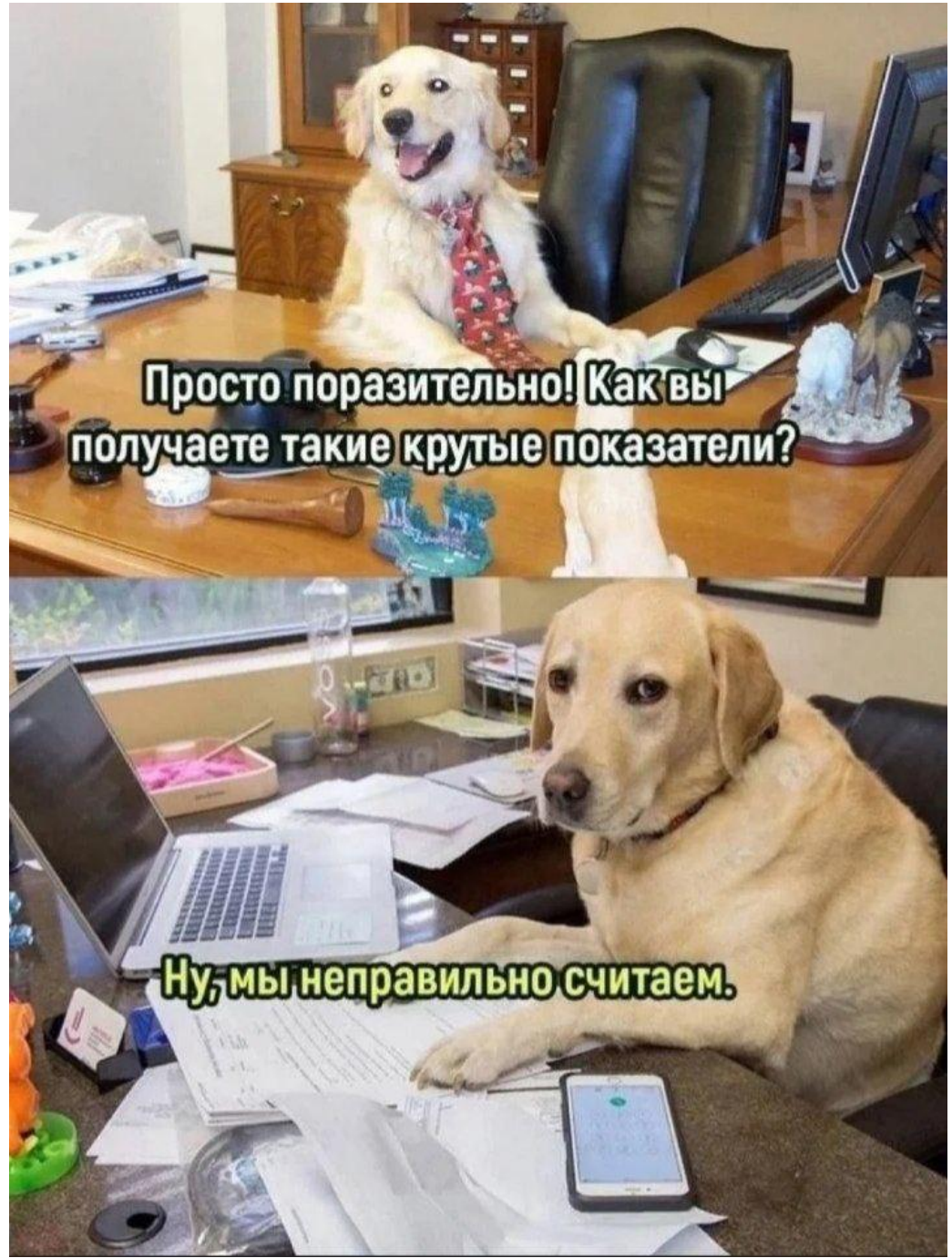
Оцениваем алгоритм как `O(g(N))` – это будет верхняя граница.

То есть растем не быстрее `g(N)` с точностью до константы

Асимптотические оценки лучший случай

Строим функцию **g(N)** – оценивает количество выполняемых действий на наборе данных размером N в лучшем случае

Оцениваем алгоритм как Ω(**g(N)**) – это будет нижняя граница.


Просто поразительно! Как вы получаете такие крутые показатели?
Ну, мы неправильно считаем.

Примеры

Линейный поиск
Задача: найти элемент в массиве длиной n

Худший случай: элемент последний или отсутствует → n сравнений.
Лучший случай: элемент первый → 1 сравнение.

Асимптотика:
Худший случай: $O(n)$.
Лучший случай: $\Omega(1)$.

Примеры

Бинарный поиск
Задача: найти элемент в отсортированном массиве длиной n.

Функция количества операций:
Каждый шаг делим массив пополам.
Кол-во шагов = $\log_2(n)$.

Асимптотика:
Худший случай: $O(\log n)$.
Лучший случай: нашли за первый шаг → $\Omega(1)$.

Примеры

Дано: массив n элементов, N = 1000000
Необходимо сделать N поисков.

Два подхода:
1) N линейных поисков
2) отсортировать массив и N бинарных поисков

Что быстрее?

Примеры

Дано: массив n элементов, N = 1000000
Необходимо сделать N поисков.

Два подхода:
1) N линейных поисков
Время: `n * O(N) = O(N^2)`
Память: `O(1)`

2) отсортировать массив и N бинарных поисков
Время: `O(N log N) + N * O(log N) = O(2N log N) = O(N log N)`
Память: от сортировки зависит

Теория к семинару

Линейный поиск

Перебираем массив по элементам слева направо, пока не найдём нужный элемент или не пройдём все элементы.

Алгоритм:

- Начинаем с первого элемента.
- Сравниваем его с искомым значением x.
- Если совпало → возвращаем индекс.
- Если нет → идём дальше.
- Если дошли до конца и не нашли → возвращаем "нет в массиве".

Теория к семинару

Бинарный поиск

Работает только в отсортированном массиве.
Вместо перебора всех элементов, мы каждый раз делим массив пополам.

Алгоритм:

- Берём середину массива.
- Если середина = x, нашли → вернуть индекс.
- Если x меньше середины → ищем в левой половине.
- Если x больше середины → ищем в правой половине.
- Повторяем, пока диапазон не станет пустым.