**Алгоритмы**

[**https://youtu.be/NErrGZ64OdE**](https://youtu.be/NErrGZ64OdE) **- полезное видео по всем алгоритмам**

**Пять столпов асимптотической оценки сложности простых алгоритмов**

Есть базовые правила того, как подходить к оценке сложности используя нотацию О-большое.

Существуют пять основных правил для расчета асимптотической сложности алгоритма:

**1️⃣ Если для некоторой математической функции 𝑓 алгоритму необходимо выполнить 𝑓(𝑁) действий, то это означает, что алгоритму потребуется сделать 𝑂(𝑓(𝑁)) шагов.**

Пример

def search\_max\_item(li: list) -> int:

index: int = 0

max\_item: int = li[index]

size: int = len(li)

while index < size:

if li[index] > max\_item:

max\_item = li[index]

index += 1

return max\_item

Заметка: алгоритм проверяет каждый из 𝑁 элементов набора чисел всего один раз, поэтому потребуется 𝑂(𝑁) шагов.

**2️⃣ Если алгоритм выполняет одно действие, состоящее из 𝑂(𝑓(𝑁)) шагов, а затем вторую, включающую 𝑂(𝑔(𝑁)) шагов, то для функций 𝑓 и 𝑔 потребуется 𝑂(𝑓(𝑁) + 𝑔(𝑁)) шагов.**

Пример

def search\_max\_item(li: list) -> int:

index: int = 0 # 𝑂(1)

max\_item: int = li[index] # 𝑂(1)

n: int = len(li) # 𝑂(1)

while index < n: # 𝑂(N)

if li[index] > max\_item: #

max\_item = li[index] #

index += 1 #

return max\_item # 𝑂(1)

Заметка: алгоритм выполняет три шага перед циклом и ещё один после него. Каждый из них имеет производительность 𝑂(1) (договоримся, считать это однократным действием), поэтому общее количество шагов составит 𝑂(3 + 𝑁 + 1), то есть 𝑂(4 + 𝑁).

**3️⃣ Если алгоритму необходимо сделать 𝑂(𝑓(𝑁) + 𝑔(𝑁)) шагов, и область допустимых значений 𝑁 функции 𝑓(𝑁) больше, чем у функции 𝑔(𝑁), то можно упростить выражение до 𝑂(𝑓(𝑁)).**

Заметка: В функции search\_max\_item мы выяснили, что всего будет выполнено 𝑂(4+𝑁) действий. Если параметр 𝑁 начнёт возрастать, его значение превысит постоянную величину 4, а значит выражение можно упростить до 𝑂(𝑁).

**4️⃣ Если алгоритму внутри каждого действия 𝑂(𝑓(𝑁)) одной операции требуется выполнять ещё 𝑂(𝑔(𝑁)) действий другой операции, то можно утверждать, что в общем алгоритм выполнит 𝑂(𝑓(𝑁) × 𝑔(𝑁)) действий.**

Пример

def repetition\_elements(li: list) -> bool:

i: int = 0

j: int = 0

n: int = len(li)

while i < n:

while j < n:

if i != j and li[i] == li[j]:

return True

j+=1

i += 1

return False

Заметка: в примере есть два цикла зависящих от 𝑁. Один вложен в другой. Внешний цикл перебирает все элементы массива, выполняя 𝑂(𝑁) итераций. На каждой итерации внутренний цикл повторно пересматривает все элементы, совершая 𝑂(𝑁) действий. Следовательно, общая производительность алгоритма составит 𝑂(𝑁×𝑁) = 𝑂(𝑁²).

**5️⃣ Константами можно пренебречь. Если C является константой, то 𝑂(C × 𝑓(𝑁)) или 𝑂(𝑓(C × 𝑁)) можно упростить до выражения 𝑂(𝑓(𝑁)).**

Заметка: В алгоритме выше блок проверка if, на самом-то деле проверяет два условия, а значит и общее количество действий внутреннего цикла получается не О(𝑁), a О(2 × 𝑁), тогда общая производительность 𝑂(2 × 𝑁²). Предположим, что значение 𝑁 увеличится в три раз, т е вместо 𝑁 будет 3 × 𝑁, тогда О(𝑁) и О(2 × 𝑁) превратится в О(3 × 𝑁) и О(2 × 3 × 𝑁), а общее количество действий 𝑂(3 × 𝑁) × О(2 × 3 × 𝑁) = 𝑂((3 × 𝑁) × (2 × 3 × 𝑁)) = 𝑂(18 × 𝑁²), но 18 × 𝑁² = 9 × 2 × 𝑁² = 3² × 2 × 𝑁² = 2 × (3 × 𝑁)². Получается, что при увеличении объёма входных данных, количество действий увеличится в девять раз (обратное тоже верно, при уменьшении выборки алгоритм ускорится). Обратите внимание, увеличение выборки в 3 раза привело к росту общего количества действий 3², таким образом нас волнует не константа, а закон, по которому количество действий зависит от объёма данных — он тут 𝑁².

