# Отчёт №3. Анализ сложности алгоритмов

Существует несколько способов измерения сложности алгоритма.

Программисты обычно сосредотачивают внимание на скорости алгоритма, но не менее важны и другие показатели – требования к объёму памяти, свободному месте на диске. Использование быстрого алгоритма не приведёт к ожидаемым результатам, если для его работы понадобится больше

памяти, чем есть у компьютера.

При анализе алгоритмов не берутся во внимание размер кэша процессора или тип многозадачности операционной системы. Анализ алгоритмов

проводят на модели абстрактного вычисления, называемого **машиной с произвольным доступом к памяти**.

Эта модель состоит из памяти и процессора:

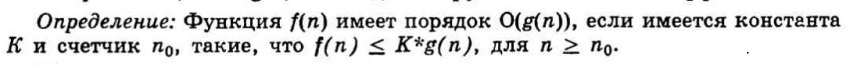
* Память состоит из ячеек (адрес и один элемент данных).
* Каждое обращение к памяти занимает одну единицу времени.
* Количество памяти достаточно для выполнения любого алгоритма.
* Процессор выполняет любую операцию за один временный шаг.
* Циклы и функции не считаются элементарными операциями.

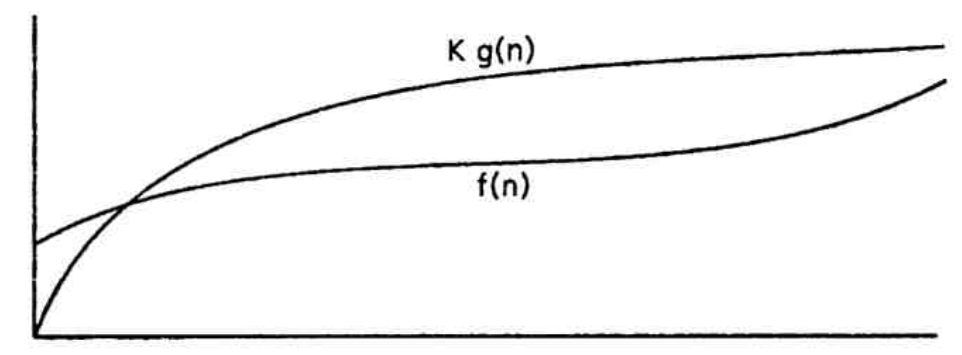
Несмотря на то, что эта модель далека от реального компьютера, она замечательно подходит для анализа алгоритмов.

**Big O нотация** нужна для описания сложности алгоритмов.

**Big O** – это мера эффективности «в худшем случае», верхняя граница того, сколько времени потребуется для выполнения задачи, или сколько памяти для этого необходимо. Например, поиск элемента в несортированном списке имеет значение O(n). Для получения результата, возможно, вам придется

перебрать весь список.

****



По категории сложности нотации подразделяются:

* O(1) – при увеличении числа входных параметров сложность алгоритма не меняется.
* O(n) – сложность алгоритма пропорциональна числу входных параметров.
* O(n^2) – сложность алгоритма пропорциональна квадрату числа входных параметров.
* O(log n) – сложность алгоритма пропорциональна логарифму числа входных параметров.

# Общие функции оценки сложности

Сейчас мы перечислим некоторые функции, которые чаще всего

используются для вычисления сложности. Функции перечислены в порядке возрастания сложности. Чем выше в этом списке находится функция, тем быстрее будет выполняться алгоритм с такой оценкой.

1. C – константа
2. log(log(N))
3. log(N)

4. N^C, 0<C<1

1. N
2. N\*log(N)

7. N^C, C>1

8. C^N, C>1

1. N!

Если мы хотим оценить сложность алгоритма, уравнение сложности которого

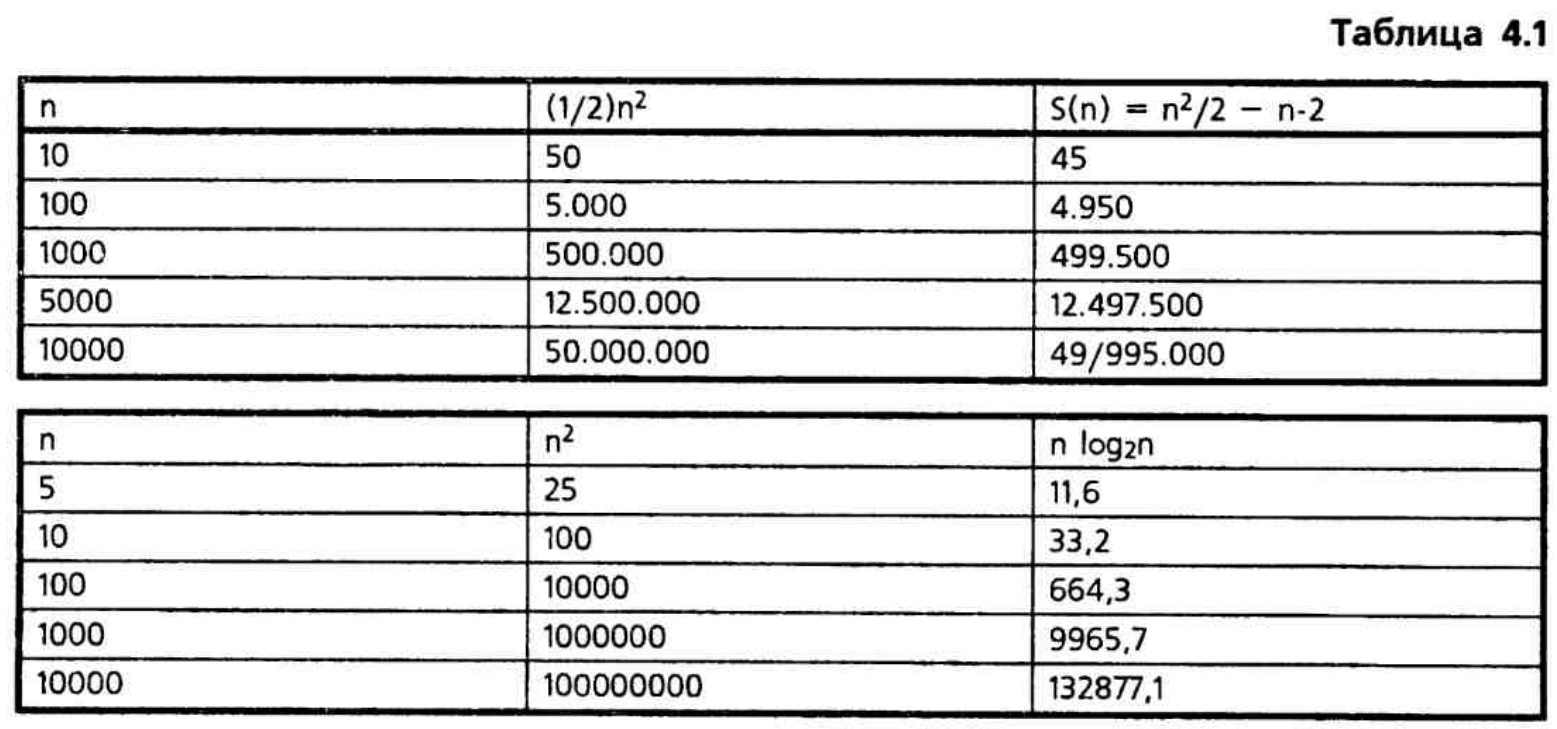
содержит несколько этих функций, то уравнение можно сократить до

функции, расположенной ниже в таблице. Например, O(log(N)+N!)=O(N!).

Если алгоритм вызывается редко и для небольших объёмов данных, то

приемлемой можно считать сложность O(N^2), если же алгоритм работает в реальном времени, то не всегда достаточно производительности O(N).

Обычно алгоритмы со сложностью N\*log(N) работают с хорошей скоростью. Алгоритмы со сложностью N^C можно использовать только при небольших значениях C. Вычислительная сложность алгоритмов, порядок которых определяется функциями C^N и N! очень велика, поэтому такие алгоритмы могут использоваться только для обработки небольшого объёма данных.



Источники:

* 1. <https://habr.com/ru/post/104219/>
  2. <https://habr.com/ru/post/444594/>
  3. https://proglib.io/p/big-o-notaciya-chto-eto-takoe-i-pochemu-ee- obyazatelno-nuzhno-znat-kazhdomu-programmistu-2022-02-17