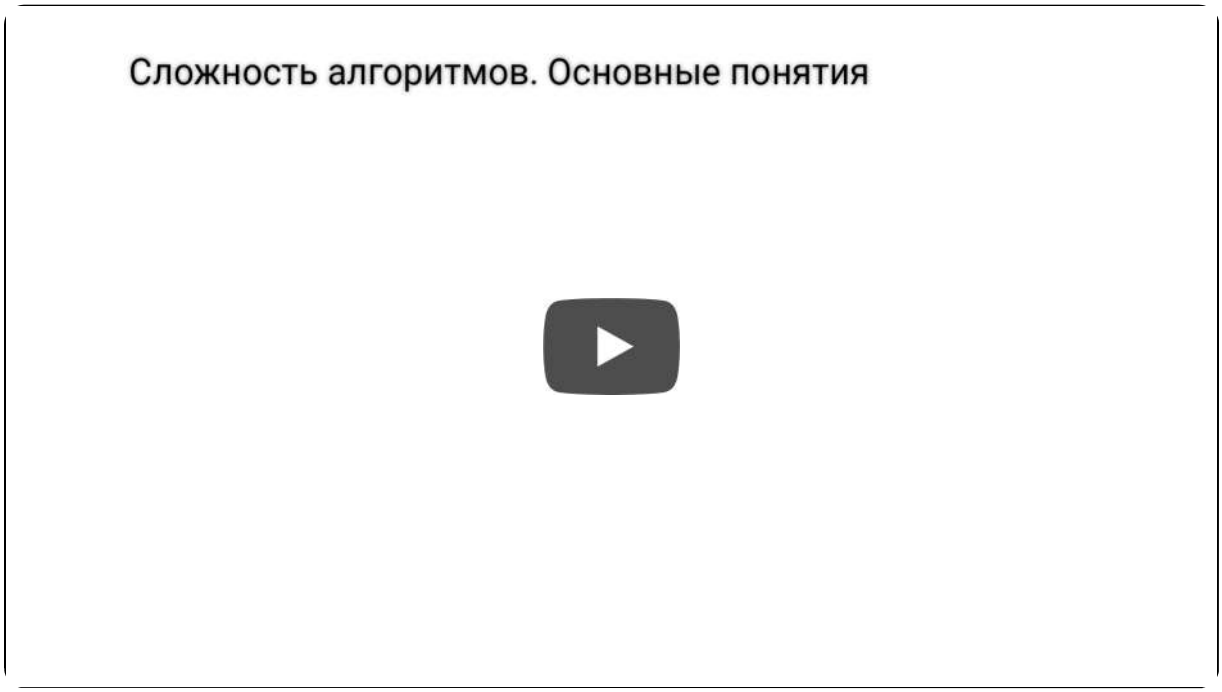
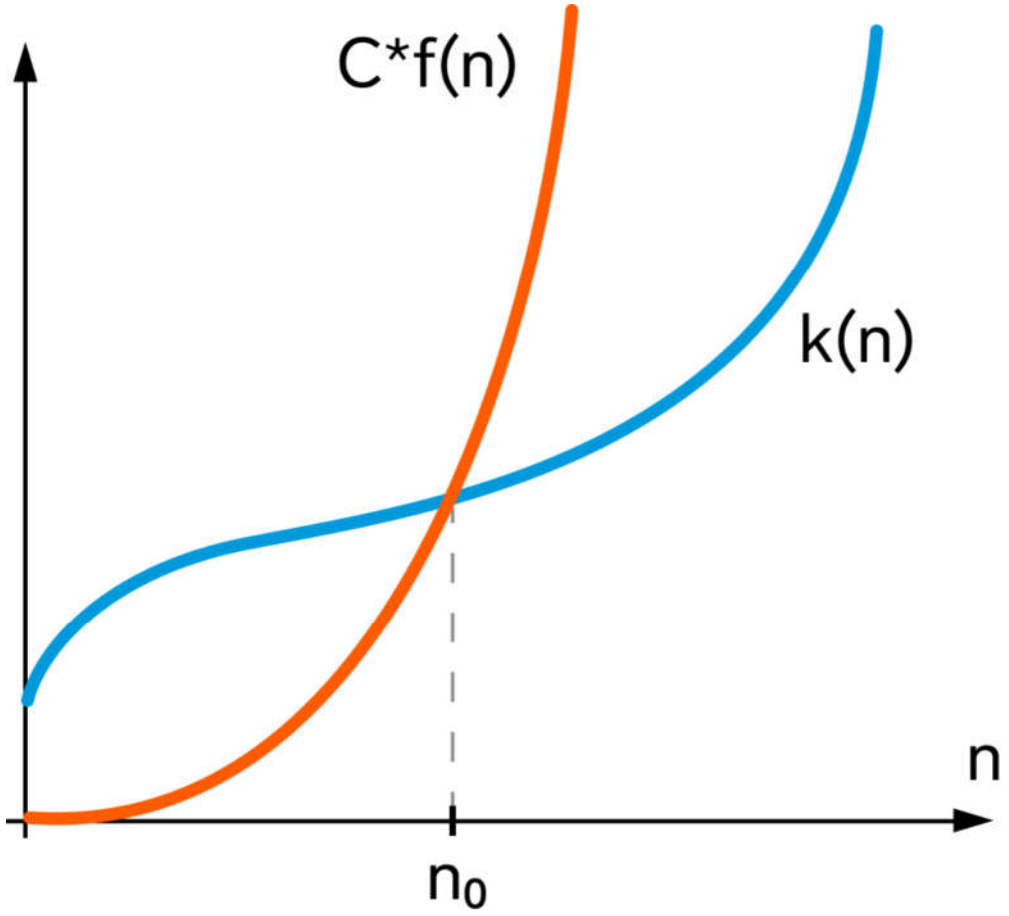
Сложность алгоритмов



Рассмотрим понятие вычислительной сложности алгоритма. Время работы программы логично измерять с помощью формулы $T = k \cdot t$, где k — количество элементарных операций, которые выполняет алгоритм, а t — время выполнения каждой их этих операций. Тем не менее в теории сложности алгоритмов временем t принято принебрегать, потому что оно отличается для различных вычислительных устройств, на которых будет исполняться наш алгоритм. По сути измеряют только количество характерных операций в зависимости от основной размерности задачи n. Обозначим количество характерных операций функцией $k(n)$.

Говорят, что количество типичных операций алгоритма в зависимости от размера задачи имеет сложность $O(f(n))$ (O большое от $f(n)$) и пишут $k(n) = O(f(n))$, если $k(n)$, начиная с какого-то n_0 растёт не быстрее, чем функция $C \cdot f(n)$, где C — некоторая константа, не зависящая от n.



Более строго: алгоритм имеет сложность $O(f(n))$, если существуют константы C и n_0 такие, что $k(n) \leq C \cdot f(n)$ при любом $n \geq n_0$.

Рассмотрим различные примеры функций, которые выражают вычислительную сложность алгоритмов. Такие функции называют асимптотиками.

- $O(1)$ – данная асимптотика обозначает, что количество действий, которые выполняет алгоритм, не превышает некоторой константы. При этом количество действий может зависеть от n, например быть различным для чётных и нечётных значений, но в любом случае оно не превосходит фиксированной константы.
- $O(n)$. Алгоритмы с такой асимптотикой называют линейными.
- $O(n^2)$. Алгоритмы с такой асимптотикой называют квадратичными.
- $O(2^n)$. Алгоритмы с такой асимптотикой относятся к классу экспоненциальных алгоритмов.

Часто сложность алгоритма выражается в виде полинома степени k:

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0, a_k > 0$$

Асимптотика такого алгоритма составляет $O(n^k)$.

Рассмотрим более подробно понятие элементарной (характерной) операции при подсчёте вычислительной сложности алгоритма. Обычно это или самая часто встречающаяся операция при выполнении алгоритма, или наиболее трудоемкая из операций.

Например, если известно, что в алгоритме выполняются операции умножения и сложения, и их примерно поровну, а также что операция умножения выполняется дольше, чем операция сложения, то можно подсчитывать вычислительную сложность алгоритма в количестве операций умножения. Если же количество операций сложения существенно больше, чем операций умножения, то вычислительная сложность такого алгоритма будет считаться в количестве операций сложения.

Рассмотрим более сложный пример. Пусть у нас есть некоторый цикл, зависящий от значения n, и в этом цикле выполняется несколько различных операций. Такими операциями могут быть умножение, сложение, сравнение и так далее. Тогда все действия внутри цикла можно считать за одну обобщённую типичную операцию данного алгоритма и подсчитывать вычислительную сложность алгоритма в количествах выполнения данного цикла.

Важно понимать, почему при подсчёте вычислительной сложности не разделяются алгоритмы, количество операций в которых отличается в несколько раз. Пусть у нас не хватает времени для вычисления $2n$ операций, однако хватает времени на вычисление n операций. Тогда мы можем использовать в 2 раза более мощный компьютер или использовать несколько вычислительных устройств для решения задачи. К сожалению, в случае, если асимптотика алгоритма есть $O(n^2)$, а мы успеваем сделать только n действий, то решить задачу, увеличив в несколько раз вычислительные мощности, не выйдет. Если же асимптотика относится к классу экспоненциальных, то увеличение вычислительной мощности даже в 100 раз приводит к увеличению максимального n, для которого мы можем выполнить алгоритм, всего на несколько единиц.