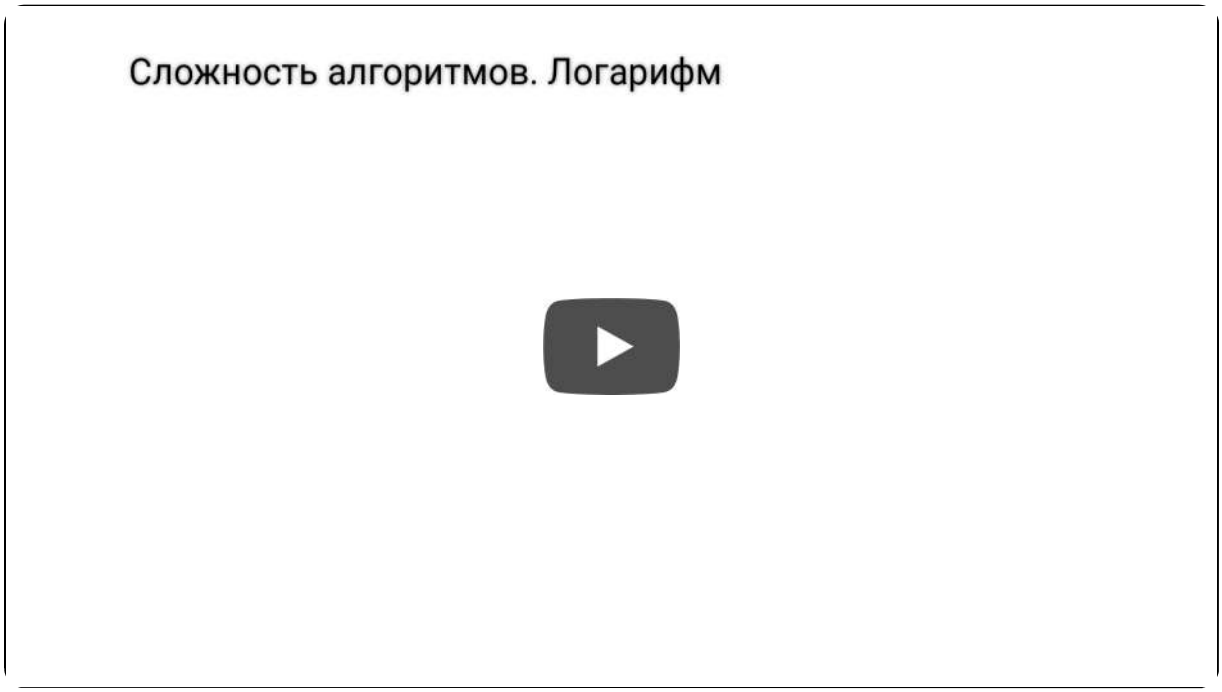
Логарифм



При вычислении сложности алгоритмов очень часто возникает такая функция, как двоичный логарифм. Давайте разберемся, что же это за функция. Обозначать нашу функцию мы будем через $\log_2$. Например, двоичный логарифм от числа n будем записывать как $\log_2 n$.

Давайте представим, что нам дали натуральное число n, а мы хотим разделить его на 2 столько раз, чтобы оно стало равно числу 1. Например, если $n = 8$, то после трёх делений на 2 мы получим в результате число 1. А если $n = 1$, то надо сделать нулевое количество делений. Необходимое число делений и будет логарифмом от заданного числа. Таким образом, можно вычислить, что $\log_2 8 = 3$, а $\log_2 1 = 0$.

Данное определение подходит для всех натуральных чисел, которые являются степенью числа 2. В общем виде можно записать: $\log_2 2^k = k$. Или k является решением уравнения $2^k = n$. Но что делать с числами, которые не являются степенью числа 2?

Например, если мы разделим число 5 два раза на 2, то получим число, большее 1, а если разделим три раза на 2, то получим число, меньшее 1. Заметим, что для степеней двоек наша функция является возрастающей — чем больше число n, тем больше значение $\log_2 n$. Логично, чтобы это свойство

распространялось для всех натуральных чисел, а не только для степеней числа 2. Тогда мы сможем сделать заключение о том, что значение $\log_2 5$ — это число между 2 и 3, потому что $4 < 5 < 8$, а логарифмы чисел 4 и 8 равны 2 и 3 соответственно.

В более общем виде можно записать, что если $2^{k-1} < n \leq 2^k$, то $k - 1 < \log_2 n \leq k$.

Таким образом, мы научились находить не само значение нашей функции, а некоторый отрезок длины 1, в котором это значение находится.

На самом деле этого уже достаточно для того, чтобы использовать данную функцию при подсчёте сложности алгоритмов. Действительно, ведь когда мы записываем функцию внутри нотации О-большое, то прибавление числа 1 к значению функции не играет никакой роли. Поэтому тот факт, что мы определили нашу функцию с точностью до единицы, тоже не играет никакой роли.

Мы выбрали для функции двоичный логарифм обозначение $\log_2$ не случайно. Так как в определении этой функции участвует число 2, то это же число мы отобразили и в названии нашей функции. Аналогичным образом можно дать определение функции $\log_3$ или $\log_{10}$, которое будет основано на степенях числа 3 или 10.

Но так как в программировании ключевую роль играет двоичная система счисления, а алгоритмы зачастую основаны на делении на число 2, то и функцию логарифм логично определить именно с числом 2. Далее мы всё время будем пользоваться именно функцией $\log_2$, а для простоты записи двойку в названии функции будем опускать и писать просто $\log$.

Рассмотрим пример алгоритма, в асимптотике которого естественным образом возникает функция логарифм. А именно алгоритм быстрого возведения числа в степень.

Напомним, в чём заключается суть этого алгоритма. Для того чтобы найти значение степени a^n, где n — некоторое натуральное число, будем использовать следующие формулы:

- $a^n = (a^{n/2})^2$, если n — чётно
- $a^n = a^{n-1} \cdot a$, если n — нечётно

Таким образом, если n чётно, то для решения задачи для заданного числа n мы можем сначала решить задачу для вдвое меньшего числа, а затем результат возвести в квадрат. Если же n нечётно, то мы можем сначала перейти к числу $n - 1$, а после этого уже для числа $n - 1$, которое является чётным, свести задачу к числу $(n - 1)/2$, которое более чем в два раза меньше исходного числа n.

Получается, что мы за одно или за два действия (в зависимости от чётности текущего числа) сводим задачу к числу, которое как минимум в два раза меньше текущего. Мы определили функцию $\log n$ как количество делений на 2, которые необходимо выполнить, чтобы из числа n получить число 1. Следовательно, так как за одну или за две операции мы будем наше число уменьшать как минимум в два раза, то количество действий нашего алгоритма будет равно $O(\log n)$.