Оценки времени исполнения. Символ O()

Для оценки производительности алгоритмов можно использовать разные подходы. Самый бесхитростный - просто запустить каждый алгоритм на нескольких задачах и сравнить время исполнения. Другой способ - математически оценить время исполнения подсчетом операций.

Рассмотрим алгоритм вычисления значения многочлена степени n в заданной точке x.
$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_i x^i + ... + a_1 x^1 + a_0$

Алгоритм 1 - для каждого слагаемого, кроме a_0 возвести x в заданную степень последовательным умножением и затем домножить на коэффициент. Затем слагаемые сложить.

Вычисление i-го слагаемого($i=1..n$) требует i умножений. Значит, всего $1 + 2 + 3 + ... + n = n(n+1)/2$ умножений. Кроме того, требуется $n+1$ сложение. Всего $n(n+1)/2 + n + 1 = n^2/2 + 3n/2 + 1$ операций.

Алгоритм 2 - вынесем x-ы за скобки и перепишем многочлен в виде $P_n(x) = a_0 + x(a_1 + x(a_2 + ... (a_i + .. x(a_{n-1} + a_n x)))$.

Например,
$P_3(x) = a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0 = a_0 + x(a_1 + x(a_2 + a_3 x))$

Будем вычислять выражение изнутри. Самая внутренняя скобка требует 1 умножение и 1 сложение. Ее значение используется для следующей скобки... И так, 1 умножение и 1 сложение на каждую скобку, которых.. $n-1$ штука. И еще после вычисления самой внешней скобки умножить на x и прибавить a_0. Всего n умножений + n сложений = $2n$ операций.

Зачастую такая подробная оценка не требуется. Вместо нее приводят лишь асимптотическую скорость возрастания количества операций при увеличении n.

Функция f(n) = $n^2/2 + 3n/2 + 1$ возрастает приблизительно как $n^2/2$ (отбрасываем сравнительно медленно растущее слагаемое $3n/2+1$). Константный множитель $1/2$ также убираем и получаем асимптотическую оценку для алгоритма 1, которая обозначается специальным символом $O(n^2)$ [читается как "О большое от эн квадрат"].

Это - верхняя оценка, т.е количество операций(а значит, и время работы) растет не быстрее, чем квадрат количества элементов. Чтобы почувствовать, что это такое, посмотрите на таблицу, где приведены числа, иллюстрирующие скорость роста для нескольких разных функций.

n	**log n**	**n*log n**	**n^2**
1	0	0	1
16	4	64	256
256	8	2,048	65,536
4,096	12	49,152	16,777,216
65,536	16	1,048,565	4,294,967,296
1,048,576	20	20,969,520	1,099,301,922,576
16,775,616	24	402,614,784	281,421,292,179,456

Если считать, что числа в таблице соответствуют микросекундам, то для задачи с *n=1048576* элементами алгоритму с временем работы *O(log n)* потребуется 20 микросекунд, алгоритму со временем *O(n)* - 17 минут, а алгоритму с временем работы *O(n^2)* - более 12 дней... Теперь преимущество алгоритма 2 с оценкой *O(n)* перед алгоритмом 1 достаточно очевидно.

Наилучшей является оценка *O(1)*... В этом случае время вообще не зависит от *n*, т.е постоянно при любом количестве элементов.

Таким образом, *O()* - "урезанная" оценка времени работы алгоритма, которую зачастую гораздо проще получить, чем точную формулу для количества операций.

Итак, сформулируем два правила формирования оценки O().

При оценке за функцию берется количество операций, возрастающее быстрее всего.
То есть, если в программе одна функция, например, умножение, выполняется *O(n)* раз, а сложение - *O(n^2)* раз, то общая сложность программы - *O(n^2)*, так как в конце концов при увеличении *n* более быстрые (в определенное, константное число раз) сложения станут выполнятся настолько часто, что будут влиять на быстродействие куда больше, нежели медленные, но редкие умножения. Символ *O()* показывает *исключительно* асимптотику!

При оценке O() константы не учитываются.
Пусть один алгоритм делает 2500n + 1000 операций, а другой - 2n+1. Оба они имеют оценку *O(n)*, так как их время выполнения растет линейно.

В частности, если оба алгоритма, например, *O(n*log n)*, то это отнюдь не значит, что они одинаково эффективны. Первый может быть, скажем, в 1000 раз эффективнее. O() значит лишь то, что их время возрастает приблизительно как функция *n*log n*.

Другое следствие опускания константы - алгоритм со временем $O(n^2)$ может работать значительно быстрее алгоритма *O(n)* при малых n... За счет того, что реальное количество операций первого алгоритма может быть $n^2 + 10n + 6$, а второго - $1000000n + 5$. Впрочем, второй алгоритм рано или поздно обгонит первый... n^2 растет куда быстрее $1000000n$.

Основание логарифма внутри символа *O()* не пишется.
Причина этого весьма проста. Пусть у нас есть $O(\log_2 n)$. Но $\log_2 n = \log_3 n / \log_3 2$, а $\log_3 2$, как и любую константу, асимптотика - символ *O()* не учитывает. Таким образом, $O(\log_2 n) = O(\log_3 n)$.

К любому основанию мы можем перейти аналогично, а значит и писать его не имеет смысла.

Математическое толкование символа O().

Определение
O(g) - множество функций *f*, для которых существуют такие константы *C* и *N*, что $|f(x)| <= C|g(x)|$ для всех $x>N$.
Запись $f = O(g)$ дословно обозначает, что *f* принадлежит множеству *O(g)*. При этом обратное выражение $O(g) = f$ не имеет смысла.

В частности, можно сказать, что $f(n) = 50n$ принадлежит $O(n^2)$. Здесь мы имеем дело с неточной оценкой. Разумеется, $f(n) <= 50n^2$ при $n>1$, однако более сильным утверждением было бы $f(n) = O(n)$, так как для $C=50$ и $N=1$ верно $f(n) <= Cn, n>N$.

Другие виды оценок.

Наряду с оценкой *O(n)* используется оценка $\Omega(n)$ [читается как "Омега большое от эн"]. Она обозначает нижнюю оценку роста функции. Например, пусть количество операций алгоритма описывает функция $f(n) = \Omega(n^2)$. Это значит, что даже в самом удачном случае будет произведено не менее порядка n^2 действий.
...В то время как оценка $f(n) = O(n^3)$ гарантирует, что в самом худшем случае действий будет порядка n^3, не больше.

Также используется оценка $\Theta(n)$ ["Тэта от эн"], которая является гибридом *O()* и $\Omega()$.
$\Theta(n^2)$ является и верхней и нижней асимптотической оценкой

одновременно - всегда будет выполняться порядка n^2 операций. Оценка $\Theta()$ существует только тогда, когда $O()$ и $\Omega()$ совпадают и равна им.

Для рассмотренных выше алгоритмов вычисления многочлена найденные оценки являются одновременно $O()$, $\Omega()$ и $\Theta()$.
Если добавить к первому алгоритму проверки на $x=0$ в возведении в степень, то на самых удачных исходных данных(когда $x=0$) имеем порядка n проверок, 0 умножений и 1 сложение, что дает новую оценку $\Omega(n)$ вкупе со старой $O(n^2)$.

Как правило, основное внимание все же обращается на верхнюю оценку $O()$, поэтому, несмотря на "улучшение", алгоритм 2 остается предпочтительнее.

Итак, $O()$ - асимптотическая оценка алгоритма на худших входных данных, $\Omega()$ - на лучших входных данных, $\Theta()$ - сокращенная запись одинаковых $O()$ и $\Omega()$.