Ассимптотическая сложность для тех кто ничего не понял или прошляпил ту лекцию по матану

Начнем с математической стороны вопроса, которая представляет собой О-символику. Все понятия оттуда нам не нужны, нас интересуют понятия О-большое и функции одного порядка

О-большое

$$f(n) = O(g(n)) \Leftrightarrow$$

$$\exists k > 0\ \exists n_0\ \forall n > n_0 :\ f(n) < k \cdot g(n) \Leftarrow$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

$$f(n) \preceq g(n)$$

Или переводя на русский: функция g ведет себя на бесконечности ± точно также как и функция f с поправкой на умножение на какой-то конечный коэффициент.
Также для лучше понимания нам помогут следующие св-ва

> $O(cf(x)) = O(f(x))$, где $c \neq 0$ – постоянная;
>
> $O(f(x)) \pm O(f(x)) = O(f(x))$;

$$O(f(x)) \cdot O(g(x)) = O(f(x) \cdot g(x));$$

Сумма двух О равна О, произвдение двух О равно О.
В информатике в общем случае употребление этого понятия сводится к написанному выше вольному определению.
Эту запись используют для оценки временной (количество операций) и пространственной (количество памяти) сложностей. Т.к. Большинство алгоритмов взаимодействуют с коллекциями (массив, список, стек, дерево и тд.) то за n в условной записи O(n) берется количество элементов в коллекции.
Например разберем пузырьковую сортировку

```
/* Отсортируем массив по убыванию */
for(int i = 1; i < n; ++i)
{
        for(int r = 0; r < n-i; r++)
        {
                if(mass[r] < mass[r+1])
                {
                        // Обмен местами
                        int temp = mass[r];
                        mass[r] = mass[r+1];
                        mass[r+1] = temp;
                }
        }
}
```

Для оценки асимптотической сложности нам нужно максимально условно получить функцию количества операций от элементов в массиве в худшем случае. Для сортировки это будет отсортированный в обратном порядке массив.
Пройдемся по алгоритму. Мы видим что цикл перебирает n-1 элементов массива следовательно сложность только этого цикла O(n) (т. к. предел отношения n-1 и n на бесконечности равен 1 т. е. Конечному числу). Будь в цикле, к примеру, 4 фиксированных операции (арифметические операции, операции вывода и тд), то спокойно можно было утверждать что сложность такого алгоритма составила бы 4(n-1), т. к. мы n-1 раз делаем 4 операции, следовательно алгоритм имел бы асимптотикой O(n) (т. к. предел отношения 4(n-1) и n на бесконечности равен 1/4 т. е. Конечному числу). Но в алгоритме из примера все сложнее. Рассмотрим тело первого цикла. Там встречается еще один цикл, количество итераций в котором зависит от внешнего i, т.о. образом количество итераций в худшем случае равна n-i, где i константа что про n стремящемуся к бесконечности почти-почти равно n. Следовательно можно говорить что сложность этой части O(n). В теле второго цикла мы делаем фиксированное количество операций в том смысле, что они не зависят от нашего n – количества элементов в массиве, следовательно асимптотика этой части равна O(1). Пользуясь свойством произведения получаем что наша сложность приблизительна равна O(n)*O(n)*O(1) = O(n^2) (предел отношения будет стремиться к конечному числу).

Так же стоит рассмотреть алгоритмы с логарифмической сложностью. Возьмем к примеру бинарный поиск. В отсортированном массиве берется центральный элемент. Если искомый элемент меньше центрального, то аналогичным образом проводим поиск левой части, иначе в правой. Те проводится деление массива на 2 до тех пор пока у нас не останется один элемент в промежутке. 1 = n/2^k , где к- количество делений. Т.о образом будет произведено приблизительно log[2](n) делений, т. е. Операций. Следовательно асимптотика бинарного поиска O(log[2](n)).