Big-O notation — это формальный способ описания верхней границы роста времени выполнения (или использования памяти) алгоритма в зависимости от размера входных данных. Она используется для оценки эффективности алгоритмов и их производительности.

**Основные обозначения:**

1. **O(1) - константное время:**
   * Время выполнения (или использование памяти) не зависит от размера входных данных. Пример: доступ к элементу массива по индексу.
2. **O(log n) - логарифмическое время:**
   * Время выполнения (или использование памяти) растет логарифмически по размеру входных данных. Пример: бинарный поиск в отсортированном массиве.
3. **O(n) - линейное время:**
   * Время выполнения (или использование памяти) растет линейно по размеру входных данных. Пример: поиск элемента в несортированном массиве.
4. **O(n log n) - линейно-логарифмическое время:**
   * Часто встречается в эффективных сортировках и некоторых других алгоритмах. Пример: быстрая сортировка.
5. **O(n^2) - квадратичное время:**
   * Время выполнения (или использование памяти) пропорционально квадрату размера входных данных. Пример: сортировка пузырьком.
6. **O(2^n) - экспоненциальное время:**
   * Рост времени выполнения (или использование памяти) экспоненциально зависит от размера входных данных. Пример: решение задачи о коммивояжере перебором всех возможных вариантов.

**Как читать Big-O notation:**

* **O(f(n))** - время выполнения алгоритма растет не быстрее, чем функция **f(n)**.
* **f(n)** - представляет собой математическую функцию, описывающую рост времени выполнения от размера входных данных.

**Примеры:**

1. **O(1)**: Доступ к элементу массива по индексу.
2. **O(log n)**: Бинарный поиск в отсортированном массиве.
3. **O(n)**: Поиск элемента в несортированном массиве.
4. **O(n log n)**: Быстрая сортировка.
5. **O(n^2)**: Сортировка пузырьком.
6. **O(2^n)**: Решение задачи о коммивояжере перебором.

Big-O notation позволяет абстрагироваться от конкретных деталей реализации и фокусироваться на общем понимании эффективности алгоритмов. Она помогает выбирать подходящий алгоритм для конкретной задачи и оценивать, насколько он эффективен при увеличении размера входных данных.