Big O -

Оценка сложности алгоритма определяется по количеству тактов(то есть проходов) а не по времени.

———————————————————————————————————————————————

O (N) - это когда зависимость прямо пропорциональна, чем больше n тем больше тактов

* например, рекурсивная функция

const sum = number => {

if(number === 1) return 1;

else {

return number + sum(number - 1);

}

};

const result = sum(5);

**O(n) — линейная сложность**

Такой сложностью обладает, например, алгоритм поиска наибольшего элемента в не отсортированном массиве. Нам придётся пройтись по всем n элементам массива, чтобы понять, какой из них максимальный.

————————————————————————————————————————————————

O (1) - это постоянная зависимость один

О (0) - это если ничего не происходит

————————————————————————————————————————————————

Big O описывает только скорость роста, поэтому мы описываем сложность O(2N) как O(N)

————————————————————————————————————————————————

Неважная сложность O(n2 + n) -> O(n2) - если сложность н2 больше в два раза поесть значительно то мы отбрасываем незначительную сложность

————————————————————————

**O(log n) — логарифмическая сложность**

Простейший пример — бинарный поиск. Если массив отсортирован, мы можем проверить, есть ли в нём какое-то конкретное значение, методом деления пополам. Проверим средний элемент, если он больше искомого, то отбросим вторую половину массива — там его точно нет. Если же меньше, то наоборот — отбросим начальную половину. И так будем продолжать делить пополам, в итоге проверим log n элементов.