code decoded

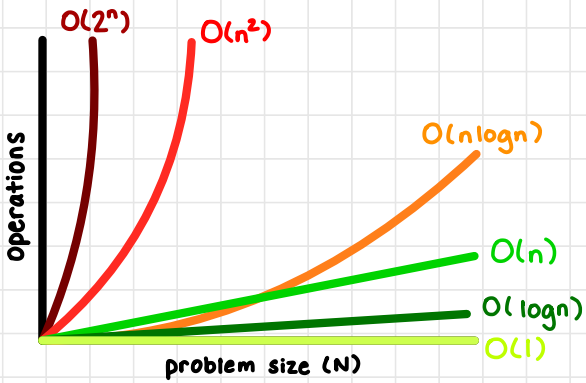
time complexity

We evaluate an algorithm's performance by counting the number of operations relative to the problem size.

Sorting a list with 10,000 elements will take more time than sorting a list with 10 elements.

The **Problem Size** would be the length of the list.

We usually use the variable N to denote problem size.



big-O notation

"Big-O" refers to the upper bound of the relationship between the number of operations and the problem size. We are concerned with how large the function grows as N increases.

We only reference the dominant term of the function. For example—

$5n^2$ becomes $O(n^2)$

$5^n + 7n$ becomes $O(5^n)$

O(1) Constant Time

The algorithm completes within some constant, regardless of size.

e.g. getters / setters
type predicates
simple arithmetic

O(n) Linear Time

The time for the algorithm to complete scales with the problem size.

e.g. traversing a list

O(logn) Logarithmic Time

The problem size is halved with each iteration.

e.g. binary search

O(2^n, 3^n) Exponential Time

Increasing the problem size by 1 doubles the time.

O(nlogn)

e.g. QuickSort
Merge Sort

O(n^2), O(n^3)... Polynomial Time

Usually an O(n) algorithm that performs an O(n) operation for N elements.