**Data Structures and Algorithms: JavaScript**

# Big O Notation

## Purpose

Determine what is the best implementation of an algorithm (i.e. function).

* If there were multiple implementations of the same function, what is the best one?
* Helps identify inefficiencies and pain points in code/applications

The efficiency and accuracy of algorithms must be analysed to compare them and choose a specific algorithm for certain scenarios. The process of making this analysis is called Asymptotic analysis. It refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as f(n) and may be for another operation it is computed as g(n2). This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

Usually, the time required by an algorithm falls under three types −

* Best Case − Minimum time required for program execution.
* Average Case − Average time required for program execution.
* Worst Case − Maximum time required for program execution.

**Asymptotic Notations**

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

* Ο Notation
* Ω Notation
* θ Notation

**Big Oh Notation, Ο**

The notation Ο(n) is the formal way to express the upper bound of an algorithm's running time. It measures the *worst-case* time complexity or the longest amount of time an algorithm can possibly take to complete.

**Analysis Metrics**

How is the algorithm assessed?

* Speed?
* Memory usage?
* Readability? (not always as important)
* Minimizing length of code?

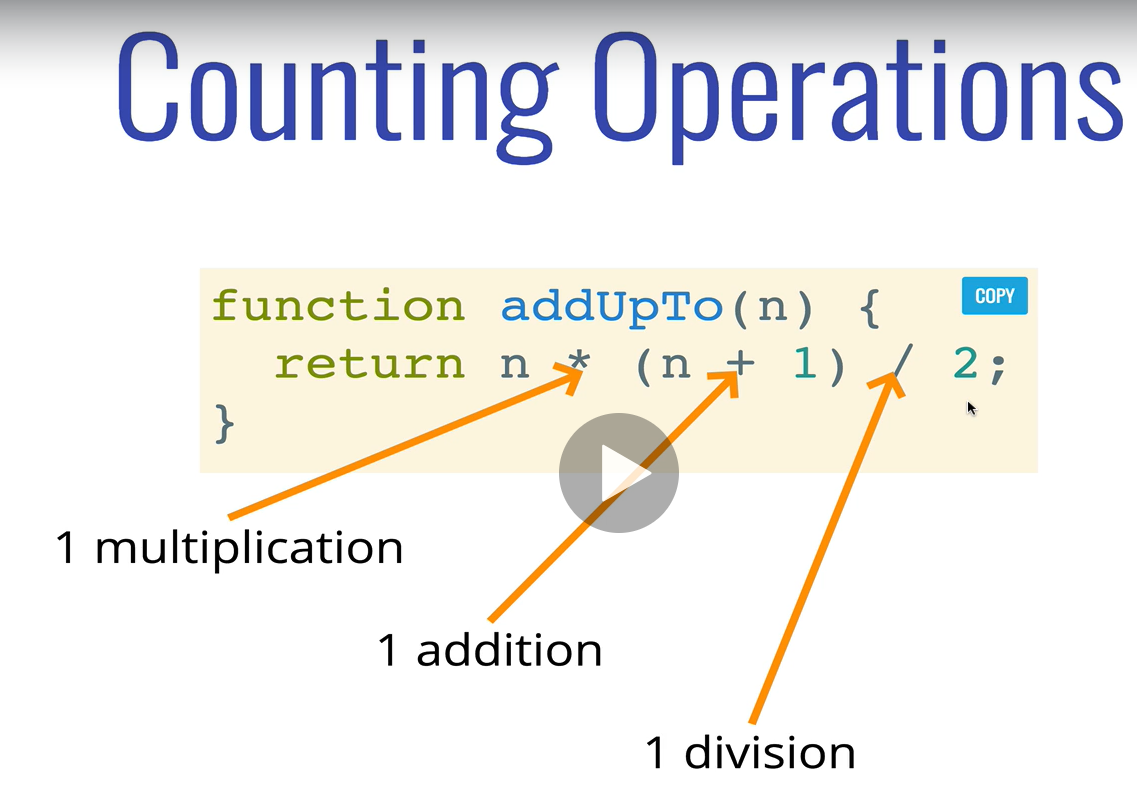
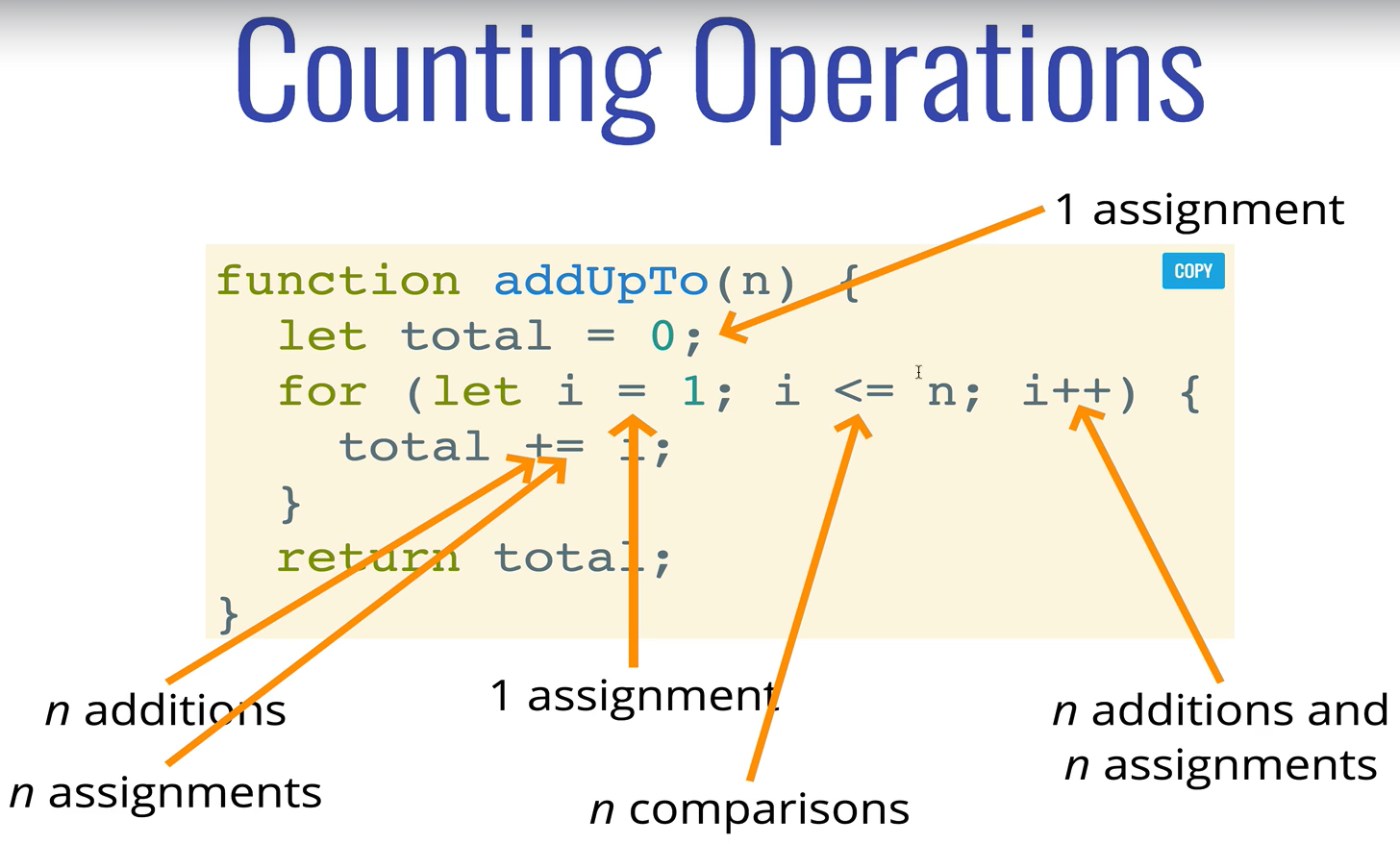
The most important on that list are Speed (how long does it take to run) and Memory Usage (how much memory is used during the run-time of the algorithm). *A lot of time there are trade-offs between the two that must be considered.*

**Speed (Runtime)**

* Different machines will record different times. This does not mean one solution will suddenly run fast/slow, but the time margins may be different (ex: min/max runtimes)
  + The same machine can record different times
  + For fast algorithms, the speed measurements may not be precise enough
* For code that may take hours to run, we do not want to repetitively time it to determine which one is faster
  + Big O Notation can be used to help address this issue

## How It Works

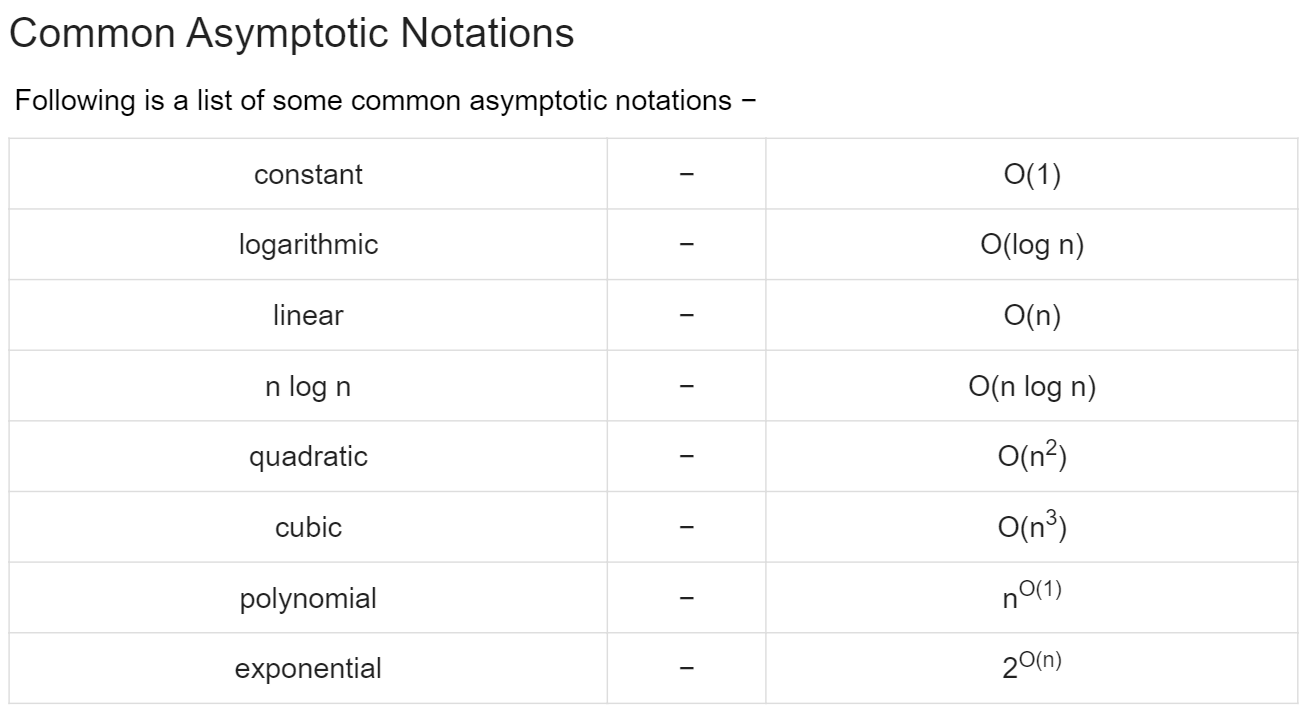
Rather than counting seconds (that can fluctuate), count the number of **simple operations** the computer has to perform!

* **Simple Operations:** Mather operations, assignments, comparisons
* Left Algorithm
  + n is irrelevant -> O(1)
  + Constant; always 3 operations
* Right Algorithm
  + If n = 10, then 52 operations (5N+2)🡪 O(n)
  + Regardless of the exact number, the number of operations grows roughly proportionally with n
* Comparison between the two algorithms quickly shows that the *left* algorithm consists of significantly fewer simple operations that the algorithm on the *right*.

Big O is the analysis of how the runtime of an algorithm grows as the input grows.

* When we are discussing Big O, we are talking about the worst-case scenario (i.e. The upper-bound for runtime)
* **An algorithm is O(f(n)) if the number of simple operations the computer has to do is eventually less than a constant times f(n), as n increases**
  + **f(n) could be linear -> (f(n) = n)**
  + **f(n) could be quadratic (f(n) = n2)**
  + **f(n) could be constant (f(n) = 1)**
  + **f(n) could be completely different**



# Analyzing Performance of Arrays & Objects

# Problem Solving

## PS – Approach

## PS – Patterns

# Recursion

# Searching Algorithms

# Sources:

1. Big O Notation - <https://www.tutorialspoint.com/python_data_structure/python_big_o_notation.htm>