Preface

Before we get started, we need to know the difference between Performance & Complexity and why we focus on complexity rather than performance in most of the time.

In short, Performance is a measure of how resources are being used when a program is run. While, Complexity is a measure of how such resources scale. (i.e. What happens when the problem being solved gets bigger?) Whilst, Performance relies on the testing machine while complexity relies on the algorithm itself. It's also notable that complexity affects performance not the other way around.

In addition to that, Complexity describes how efficient our algorithm to a specific problem is. It is also important to note that efficiency covers more than just the CPU (time) needed. It also covers memory usage, disk usage & network usage. But we'll just focus on the CPU resources.

Big O Notation

Complexity is often emphasized by the Big O Notation, which is sometimes called Landau's Symbol, after its inventor; the German mathematician Edmund Landau. The letter O is also used since the rate of growth of a function is also called the Order of such function.

Determining Complexity of an algorithm

Here's the important rule

```
Processing time ∝ Number of basic operations
```

Thus, You just determine the number of basic operations that occur. However, there's some other extra rules that you should follow.

- Ignore constants, constants are usually affected by the machine itself. Thus, it's considered as a part of performance not complexity.

- We are usually interested in the worst case of the algorithm. Worst case refers to the case in which your algorithm runs the maximum number of operations needed to accomplish a certain task. Whereas Best case refers to the case in which your algorithm runs the minimum number of operations needed to accomplish such task.

So, lets analyze some algorithms....

```
for (int i = 0; i < n; i++) {
    console.log(`I've said 'Hello World' for ${i} time(s)`);
}
```

The logging statement is considered a basic operation, thus the number of basic operations is 'n'. So, complexity can be determined as O(n). Notice however, that this example doesn't have a worst case nor a best case.

Here's another similar example

```
for (int i = 0; i < n; i+=2) {
    console.log(`I've said 'Hello World' for ${i/2} time(s)`);
}
```

The logging statement will run n/2 times & Since we ignore constants, it will have the same complexity as the previous one. O(n).

Wanna see some logarithmic complexities?

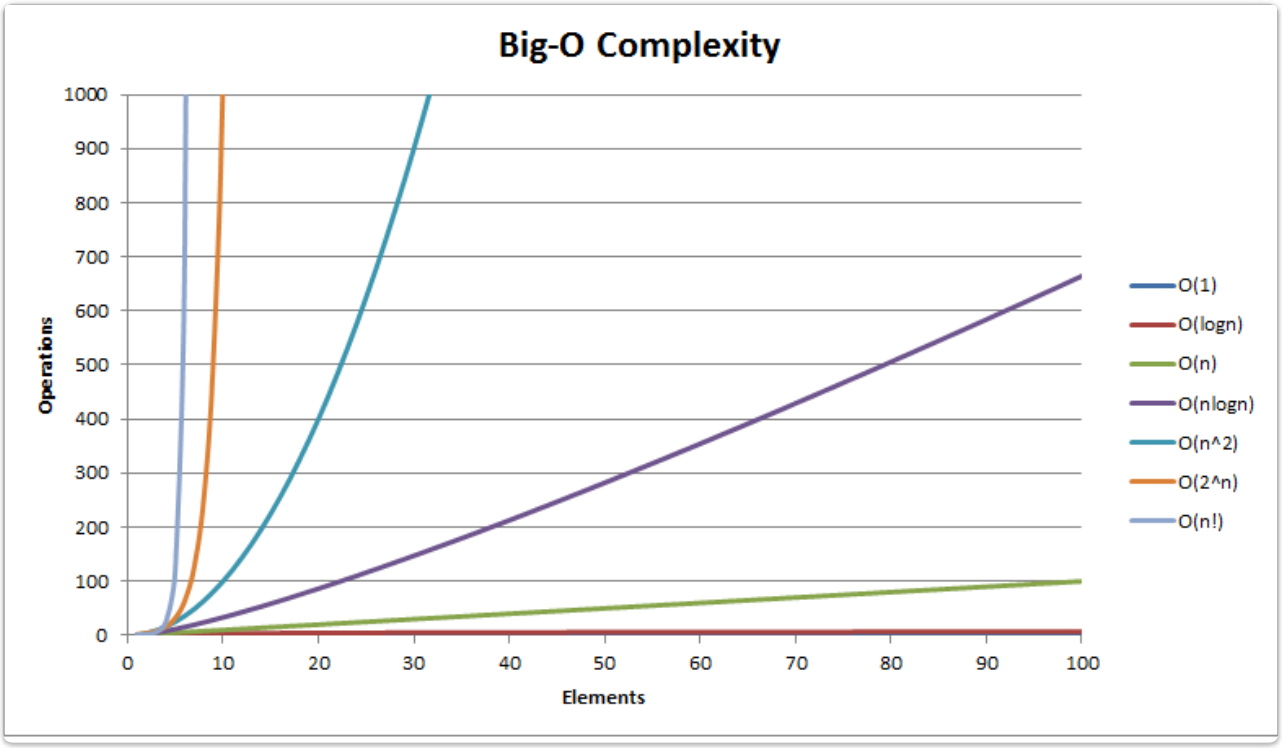
```
for (int i = 0; i < n; i*=2) {
    console.log(`I've said 'Hello World' for ${Math.log2(i)}
time(s)`);
}
```

Such snippet, runs $\log_2(n)$ times. Thus, its complexity is $O(\log(n))$

Notice that we didn't include the base in the notation since we can convert between log. basis by dividing by a constant.

What about nested loops? You do the same, just figure out how many times does it run. And in many cases, you just multiply 'n' of the nested loop by the 'n' of the parent loop. And that's when you get quadratic, cubic, and other horrible nightmares...

Here's a graph that demonstrates different complexities, **Lower is better**.



A graphical representation of the growth of different functions.
Credits to Varun N R

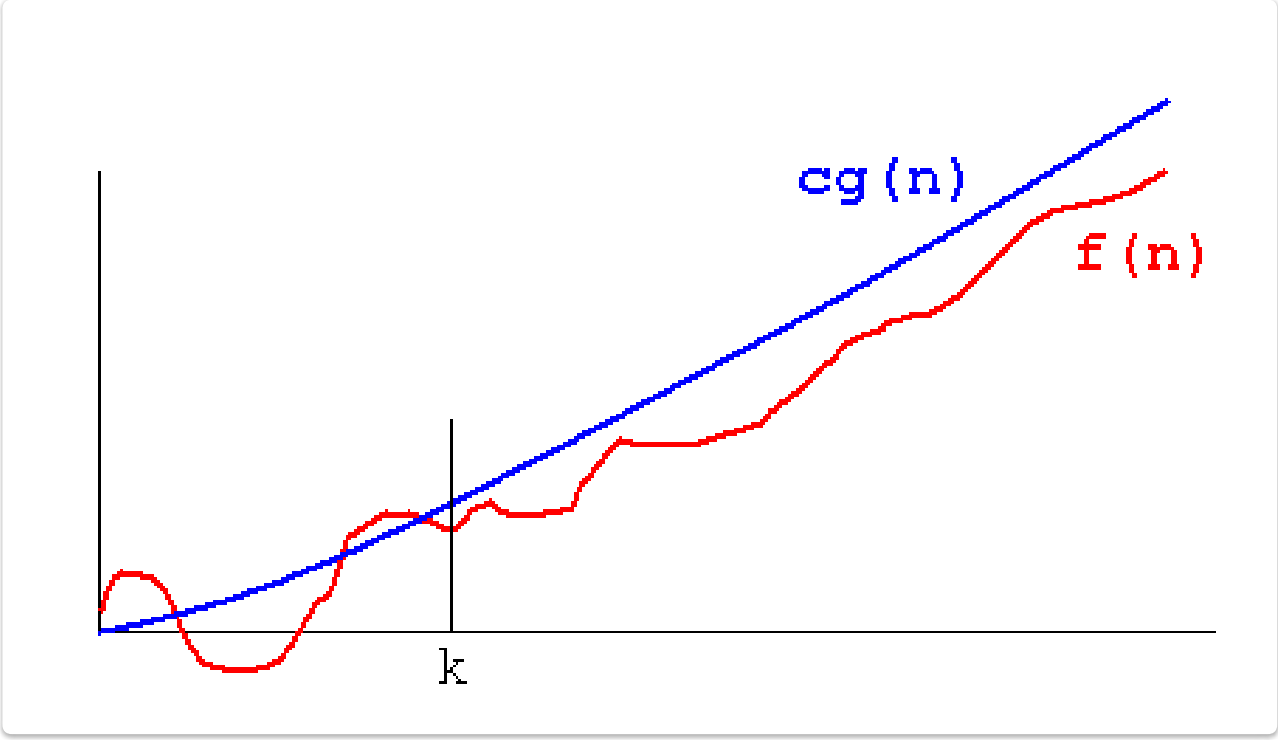
The Mathematics behind it

The Big O Notation generally expresses the upper bound of a function. Meaning that it descripes a value that the function cannot exceed. For example, a function $f(x) = 2$. has an upper bound of 2. because, well; $f(x)$ can never exceed the value of 2.

Here's the formal definition

We say that $f(x) = O(g(x))$. Iff, there exists constants c & κ. Such that $\forall\ x \geq \kappa$, $|f(x)| \leq |c.g(x)|$

Here's a graphical representation of the definition



A graphical representation of a function acting as an upper bound for another function.
Credits to **Donald E. Knuth**, Big Omicron and Big Omega and Big Theta, SIGACT News, 8(2):18-24, April-June 1976.

So, lets practice a little

Prove that the function $f(x) = x^5 + 9x^2$ is $O(x^5)$

$$\because x^5 + 9x^2 \leq C.x^5$$

$$\text{Let } \kappa = 1$$

$$\therefore 10 \leq C$$

$$\therefore \kappa \ \& \ C \text{ exists}$$

$$\therefore f(x) = O(x^5)$$

Here's some common useful Identities

- $\text{Given a polynomial } f(x) = a_n x^n + ... \ f(x) = O(x^n)$
- $n! = O(n^n)$
- $\log(n!) = O(n.\log n)$

Other notations' examples

- Big Omega (Ω), A complementary of Big O, as it describes the lower bound of a function.
- Big Theta (Θ) Describes both the lower and upper bounds of a function.

References

- MIT Lecture notes