Introduction to Time and Space Complexity

We saw algorithms and data structures but how do we estimate which data structure is faster and which is the best algorithm for that given data structure. There enters the concept of time complexity and space complexity.

When we are developing an algorithm there are two different things which we have to check: -

1. **Time Complexity**
2. **Space Complexity**

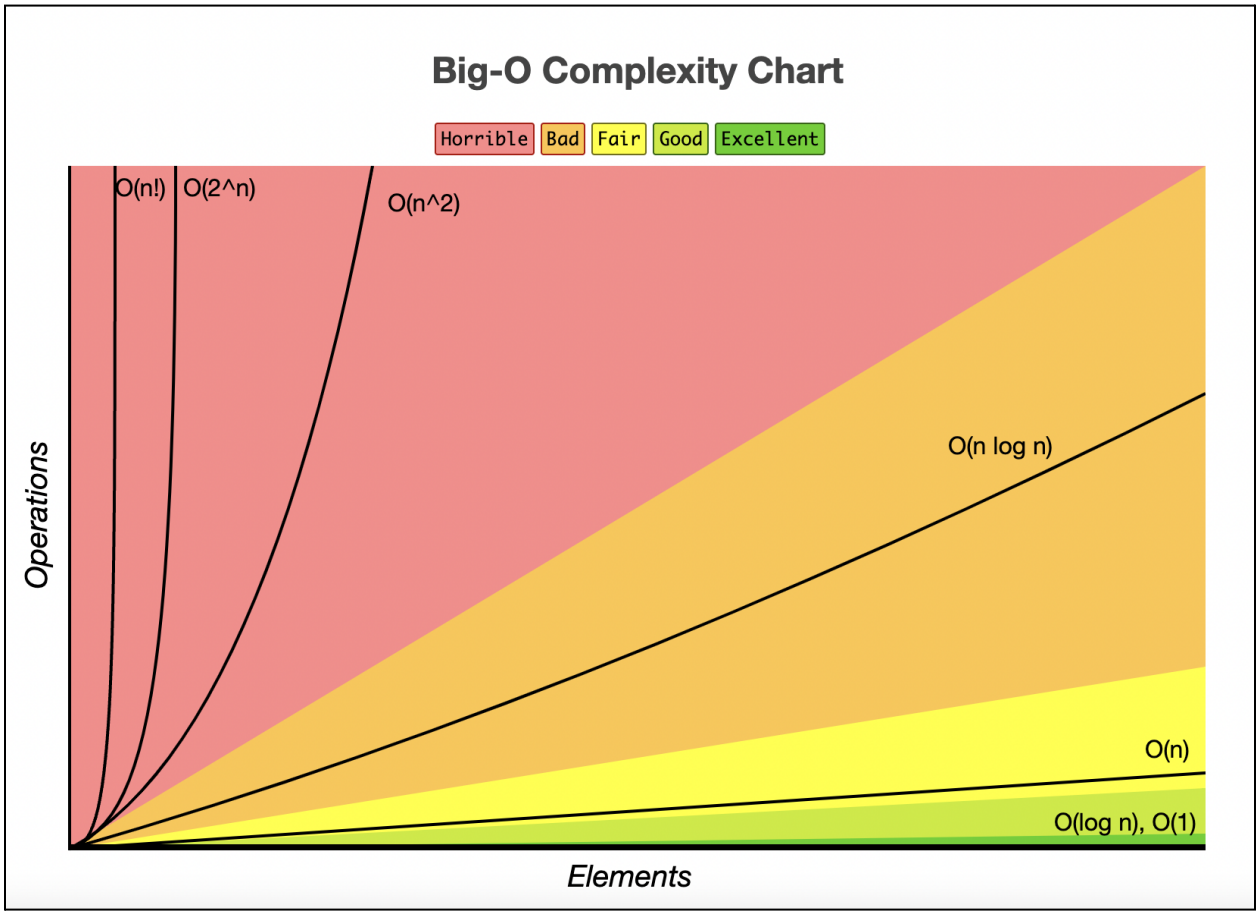
Time of Complexity: The time complexity of an algorithm is how much time it requires to solve a particular problem. The time complexity is measured using a notation called big-o notation, it shows the effectiveness of the algorithm. It shows the amount of time taken relative to the number of data elements given as an input. This is good because it allows you to predict the amount of time it takes for an algorithm to finish given the number of data elements.

Big-O notation is written in the form O(n), where n is the relationship between n: the number of inputted entities, and O(n) is the time relationship.

There are different **Big O notations**:

Big O Notation	**Name**	**What is means**
O(1)	Constant time complexity	The amount of time taken to complete an algorithm is independent from the number of elements inputted.
O(n)	Linear time complexity	The amount of time taken to complete an algorithm is directly proportional to the number of elements inputted.
$O(n^2)$	Polynomial time complexity (example)	The amount of time taken to complete an algorithm is directly proportional to the square of the elements inputted.

$O(n^n)$	Polynomial time complexity	The amount of time taken to complete an algorithm is directly proportional to the elements inputted to the power of
$O(2^n)$	Exponential time complexity	The amount of time taken to complete an algorithm will double with every additional item.
O(logn)	Logarithmic time complexity	The time taken to complete an algorithm will increase at a smaller rate as the number of elements inputted.



Order:- O(1) [Best Case] < O(log(n)) < O(n) < O(nlog(n)) < $O(n^2)$ < $O(2^n)$ < O(n!) [Worst Case]

Introduction to Asymptotic Notations

The first question that comes to our mind is what is **Asymptotic Notation**?
In easy words, it gives us an idea of how good an algorithm is when compared with another algorithm.
We cannot directly compare two algorithms side by side. It heavily depends on the tools and the hardware we use for comparisons, such as the Operating System, CPU model, processor generation, etc. Even if we calculate time and space for two algorithms running on the same system, their time and space complexity may be affected by the subtle changes in the system environment.
Therefore, we use **Asymptotic Analysis** to compare space and time complexity.
It analyzes two algorithms based on changes in their performance concerning the increment or decrement in the input size. **Big-O is important for analyzing and comparing the efficiencies of algorithms.**

Primarily there are three types of Asymptotic notations:

1. **Big-Oh (O) notation.**
2. **Big Omega (Ω) notation.**
3. **Big Theta (Θ) notation.**

Best case, Worst case, and Average case in Asymptotic Analysis:
Best Case: It is defined as the condition that allows an algorithm to complete the execution of statements in the minimum amount of time. In this case, the execution time acts as a lower bound on the algorithm's time complexity. This is denoted by **Big Omega (Ω) notation.**
Average Case: In the average case, we get the sum of running times on every possible input combination and then take the average. In this case, the execution time acts as both lower bound and upper bound on the algorithm's time complexity. This is denoted by **Big Theta (Θ) notation.**
Worst Case: It is defined as the condition that allows an algorithm to complete the execution of statements in the maximum amount of time. In this case, the execution time acts as an upper bound on the algorithm's time complexity. This is denoted by **Big-Oh (O) notation.**

Practice Examples:-

1.) The following code sums the digits in a number.

What is its big O time?

```
function sumDigits( n) {
let sum = 0;
while (n > 0) {
sum += n % 10;
n /= 10;
}
return sum;
}
```

Solution:- O(log(n)) [Logarithmic Time Complexity]

2.) What is the Big O Time for the given code?

```
function a(n){
var count =0;
for (var i=0;i<n*n;i++){
count+=1;}
return count;
}
```

Solution:- $O(n^2)$ [Quadratic Time Complexity]

3.) Find the Big O Time Complexity and Big O Space Complexity for the given code?

```
function a(n){
var count =0;
for (var i=0;i<n;i++){
count+=1;
}
for (var i=0;i<5*n;i++){
count+=1;
}
return count;
}
```

Solution:- O(n) + O(5*n) => Take the bigger term => O(5*n) => O(n) {Remove the Constant 5}

4.) What is the Big O Time Complexity of the given code?

```
function a(n){
var count =0;
for (var i=0;i<n;i++){
count+=1;
}
return count;
}
```

Solution:- O(n) linear time complexity.

5.) What is the Big O Time Complexity of the given code?

```
function someFunction(n) {
while (true){
console.log(n);
}
}
```

Solution:- O(∞) Infinite loop. This function will not end.