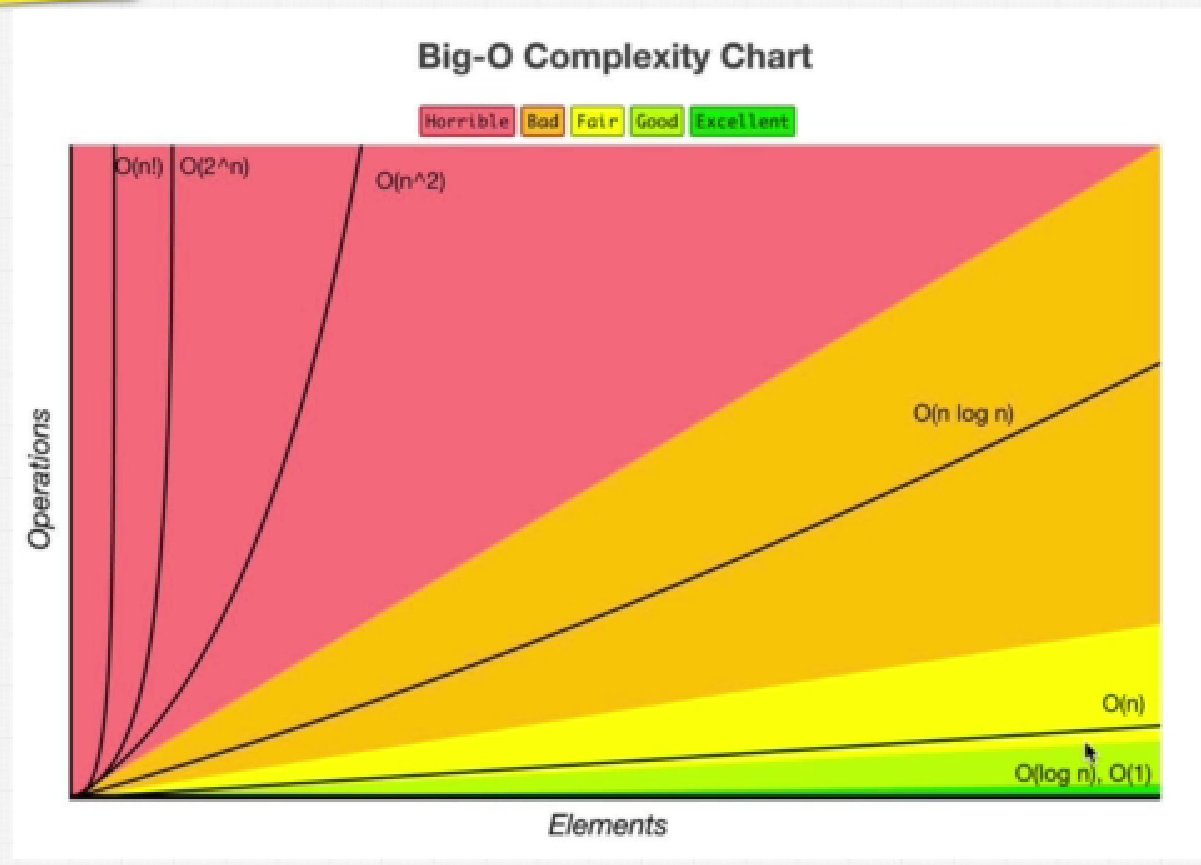
**BIG O**

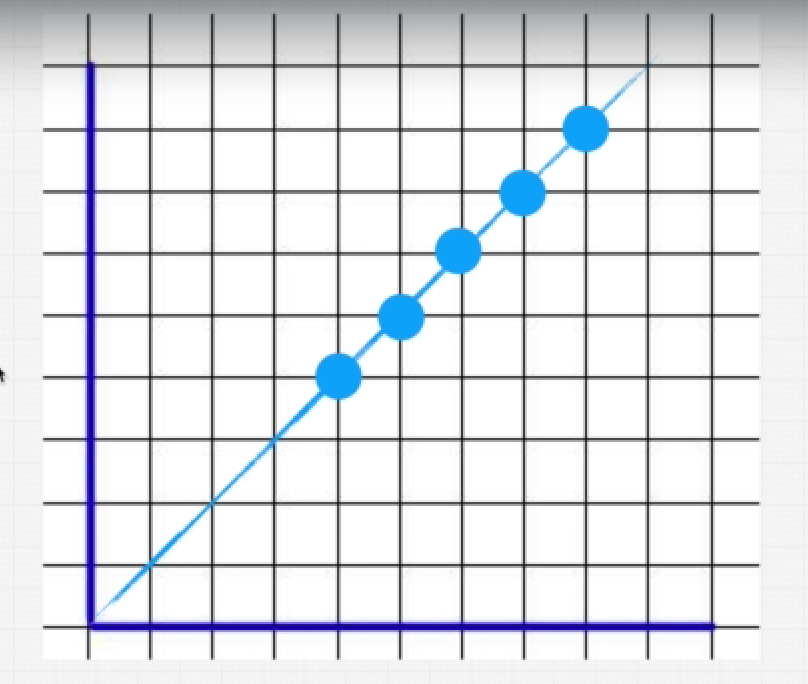
Big O notation is a language we use to measure how much does it takes to run an algorithm.



When we go bigger with input how much does the algorithm and function slow down.

What is a Good Code:

* Readable
* Scalable



1. 0(n): It also called Linear Time which denotes as the number of input(n) increases the time will also increase.
2. O(1) : Known as constant time ,

Example : function findOpener(array) {

    console.log(array[0]) // O(1)

    console.log(array[1]) // O(2)

}

findOpener(names)

Here in the example above array which has been provided has an argument to the function findOpener() has 11 items as names.

When we console.log(array[0]), it doesn’t matters how big the array is, its always going to print the first name. This notation is called O(1) also known as Constant Time. It comes in excellent region in Complexity Chart.

Exercise 1:

function funChallenge(input) {

    let a = 10; // O(1)

    a = 50 + 3; // O(1)

    for (let i = 0; i < input.length; i++) { // O(n)

      anotherFunction(); // O(n)

      let stranger = true; // O(n)

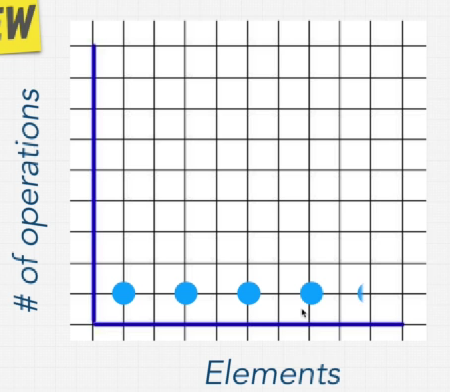
      a++; // O(n)

    }

    return a; // O(1)

  }

funChallenge()

In the function above you can see that there are some functions/methods or operations that runs only once and they are denoted by O(1) , no matter how big the input is it will run only once. Again, there are some functions/Operations or methods that will run every time it iterates or It is directly dependent upon the number of input and this is denoted by O(n).

So, by adding all the notations we get: BIG O(3 + 4n)

Exercise 2:

function anotherFunChallenge(input) {

    let a = 5; // O(1)

    let b = 10; // O(1)

    let c = 50; // O(1)

    for (let i = 0; i < input; i++) { // O(n)

      let x = i + 1; // O(n)

      let y = i + 2; // O(n)

      let z = i + 3; // O(n)

    }

    for (let j = 0; j < input; j++) { // O(n)

      let p = j \* 2; // O(n)

      let q = j \* 2; // O(n)

    }

    let whoAmI = "I don't know"; // O(1)

  }

  // BIG (4 + 7n)

Now we will se How to actually calculate BIG O instead of counting it in each line as I did in last 2 exercises.

There are Four Simple rules to calculate BIG O:

* RULE 1: Worst Case
* RULE 2: Remove Constants
* RULE 3: Different terms for inputs
* RULE 4: Drop Non-Dominants