**Rule Book**

**BigO – Time complexity of a function**

**Rule 1:** **Worst Case**

const everyone = ['dory', 'bruce', 'marlin', 'gill', 'nemo', 'bloat', 'nigel', 'squirt', 'darla', 'hank'];

If we wanted to find **‘hank’**, we need to iterate through the ‘everyone’ array till the end. However, if ‘hank’ was in the beginning of the array, we only need to iterate through it once because it will find ‘hank’ in the first iteration. This is O(n) because we do not know how many times to iterate through the array to find an element.

**Rule 2:** **Remove Constants**

O(1 + n/2 + 100) -> O(n/2 + 100) -> O(n/2) -> O(n)

Remove the constants. ‘n’ is not a constant however, the numbers ‘1’, ‘/2’, and ‘100’ are constants therefore we can remove them and simplify it to O(n).

function compressBox(boxes) {

    boxes.forEach(function(boxes) {

        console.log(boxes);

    });

    boxes.forEach(function(boxes) {

        console.log(boxes);

    });

}

This function has 2 for loops.

O(2n) -> O(n)

We drop the constant and simplify it to O(n)

**Rule 3:** **Different terms for inputs**

function compressBox(boxes, boxes2) {

    boxes.forEach(function(boxes) {

        console.log(boxes);

    });

    boxes2.forEach(function(boxes) {

        console.log(boxes);

    });

}

There are 2 inputs (boxes and boxes2) here and they are not nested. The first for loop loops over ‘boxes’ and the second for loop loops over ‘boxes2’.

**O(a + b)**

function compressBox(boxes, boxes2) {

    boxes.forEach(function(boxes) {

        console.log(boxes);

        boxes2.forEach(function(boxes) {

            console.log(boxes);

        });

    });

}

This is a nested for loop. When we have a nested for loop, we multiply. Since we have two inputs (boxes and boxes2) and they are nested, the BigO = O(a \* b)

**O(a \* b)**

//Log all pairs of array

const boxes = ['a', 'b', 'c', 'd', 'e'];

function logAllPairsOfArray(array) {

    for (let i = 0; i < array.length; i++) {

        for (let j = 0; j < array.length; j++) {

            console.log(array[i], array[j])

        }

    }

}

logAllPairsOfArray(boxes)

This is a nested for loop. Here there are only one input ‘array’.

O(n \* n) -> O(n^2)

**O(n^2) – known as quadratic time**

**Rule 4:** **Drop Non-Dominants**

function printAllNumbersThenAllPairSums(numbers) {

    console.log('these are the numbers:');

    numbers.forEach(function (number) {

        console.log(number);

    });

    console.log('and these are their sums:');

    numbers.forEach(function (firstNumber) {

        numbers.forEach(function (secondNumber) {

            console.log(firstNumber + secondNumber);

        });

    });

}

printAllNumbersThenAllPairSums([1, 2, 3, 4, 5])

This function has a for loop as well as a nested for loop.

O(n + n^2) -> O(n^2)

We drop the non-dominants meaning we only look at the highest BigO which is n^2 here.

**O(n^2)**

\*note: if we had 2 nested loops, BigO would be O(n^3)

if we had 3 nested loops, BigO would be O(n^4) and so on…

***O(n!)*** *– We are adding a nested loop for every input we have. The worst in scalability.*

**3 Pillars of Programming**

1. **Readable**
2. **Memory** – Space Complexity
3. **Speed** – Time Complexity

**Space Complexity**

**These items take up space:**

* Variables
* Data Structures
* Function Call
* Allocations