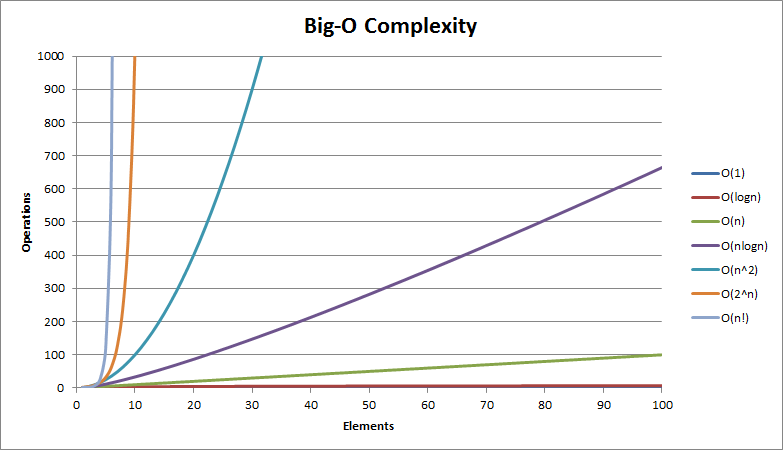
What makes good Code?

1. Readable and Scalable
2. Good code/instructions that make something, similar to baking a cake
3. We can use let t0=performance.now(); before code and t1=performance.now(); after
   1. Take the difference console.log( (t1- t0) + ‘milliseconds) and that will tell us how long the code took to run
      1. Performance is built into the browser

What is Big O?

1. Big O measures how long a algorithm takes to run, regardless of computer differences
   1. When we ***grow with input, how much does the algorithm slow down***
      1. I.e when an array increases, how many operations do we have to do

A runtime is how long something takes to run



O(n) -- Linear Time

1. This is one operation for One element.
   1. Example. If we loop through an array using a for loop, we run one operation for each element to check if it matches the given parameters
   2. That For loop is linear time (find nemo example)
   3. N is inputs, so if we had an array if 100, it would be O(100)
   4. O(n) is FAIR
   5. The amount of operations are equal to the operation
   6. N=input
2. O(1) -- constant time
   1. The same # of operations stays flat.
   2. Flat line
   3. Very scalable

// What is the Big O of the below function? (Hint, you may want to go line by line)

function anotherFunChallenge(input) {

let a = 5; //1

let b = 10; //1

let c = 50; //1

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

let x = i + 1; //n

let y = i + 2; //n

let z = i + 3; //n

}

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

let p = j \* 2; //n

let q = j \* 2; //n

}

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

}

BIG O(4 + 7n)

BIG O rules

Rule 1.

1. Use break; in an if loop to exit after a condition is met (in the if block)
2. Assume the worst case, so if a loop is set to ten loops, assume all ten

Rule 2

1. Remove constants
2. O( 1 + n/2 +100) will result in O(n)
3. O(2n) becomes O(n)

Rule 3

1. Different terms for inputs
2. Two if loops with different inputs, would result in differences in the inputs
3. O( a +b) (a and b can be anything I.e m,n,box 1 etc
4. Rule of thumb, nested are multiplied, one after another are added

O(n^2) --Quadratic time

/Nested loops

1. \*
2. O(n \*n) or O(n^2)
3. I.e. Log all pairs of array
4. Horrible
5. Interview questions ask how to make these better

Rule 4

1. Drop non dominants
   1. We care about the most important parts
   2. Big O( x^2 + 100 + x/2)
      1. X^2 will be most important (if we plug in 25, we’ll see why

Big O doesn’t matter with small scale complexity

Data structures - ways to store data

Alogrithms - ways to write data

Data structures + algorithms = programs

O(n!) --factorial time

1. Probably won’t encounter
2. “Oh no”
3. Worst
4. Nested loop for every input

Scalable breaks into two things

1. Speed - how fast is the runtime.
2. Memory

THREE PILLARS OF CODE

1. Readable
2. Speed. Time complexity. Efficient
3. Memory

There’s usually a trade off between speed and memory

Heap

1. Where we usually store variables

Stack

1. Where we keep track of function calls

Space complexity