Big O

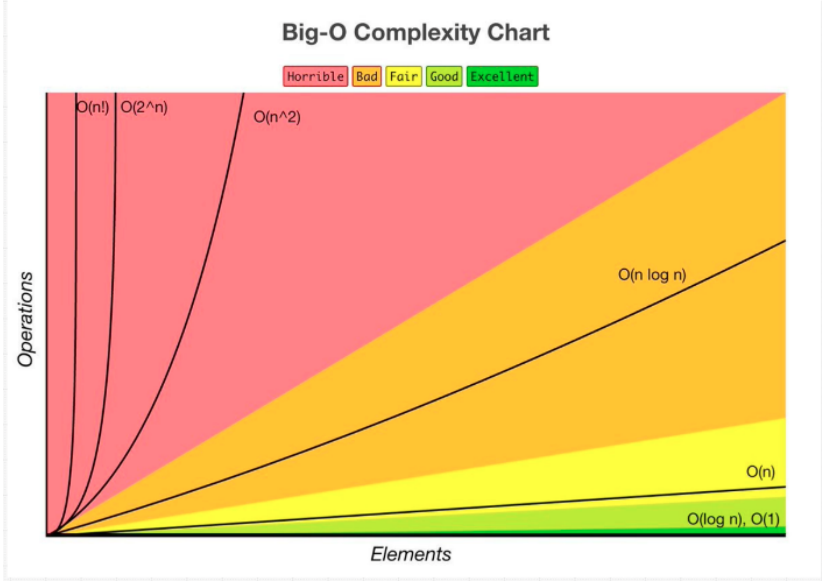
Friday, 11 February 2022 4:59 PM

BigO-cheat-

What is good code?

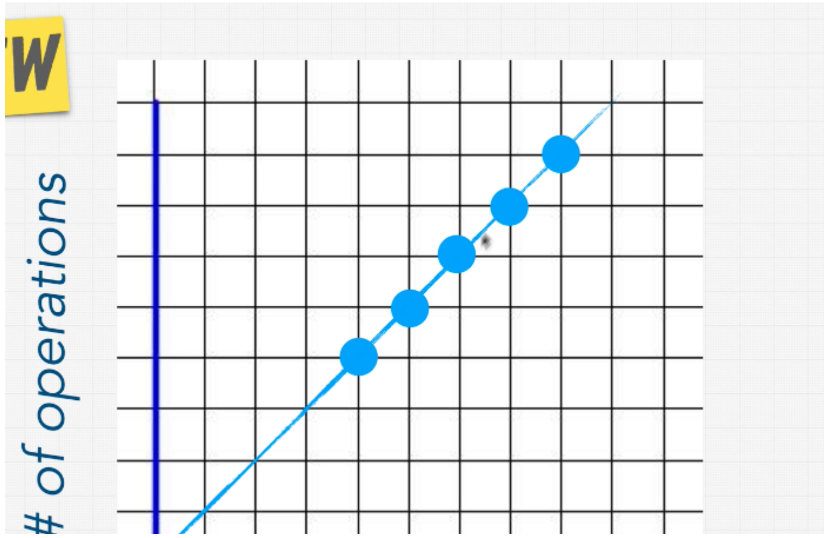
1. Readable
2. Scalable (Big O)

Big O helps to calculate how long an algorithm takes to run.

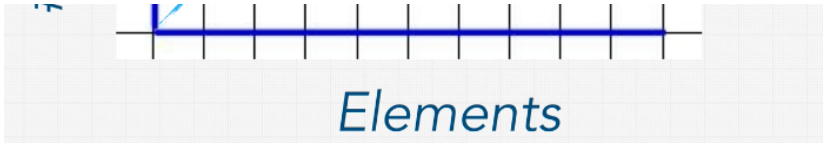


O(n) - Linear Time

```
const { performance } = require('perf_hooks');
const nemo = ['nemo'];
const everyone = ['dory', 'bruce', 'marlin', 'gill', 'bloat', 'nigel',
'squirt', 'darla', 'hank', 'nemo'];
const large = new Array(1000).fill('nemo');
function findNemo(nemo){
  let t0 = performance.now();
  for(let i=0; i< nemo.length; i++) {
    if(nemo[i] === 'nemo') {
      console.log('found nemo at ' + i)
    }
  }
  let t1 = performance.now();
  console.log("Total time in ms: "+ (t1-t0));
}
findNemo(large);
```

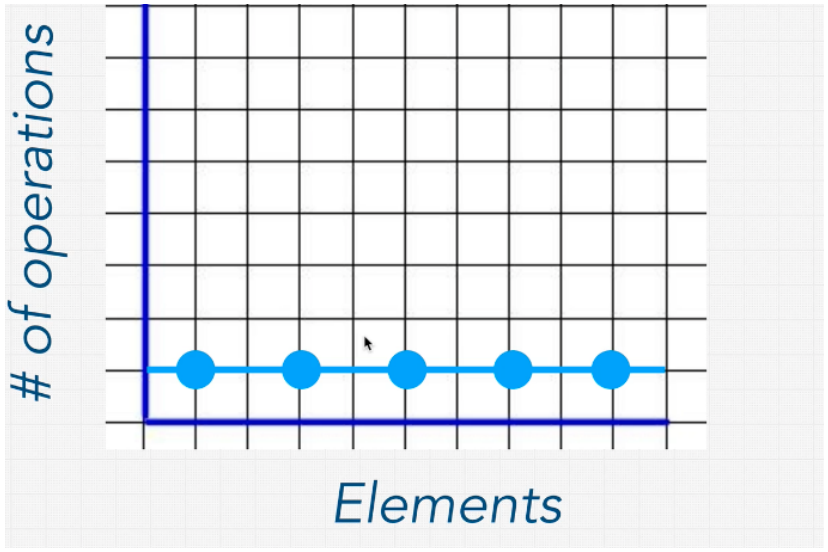


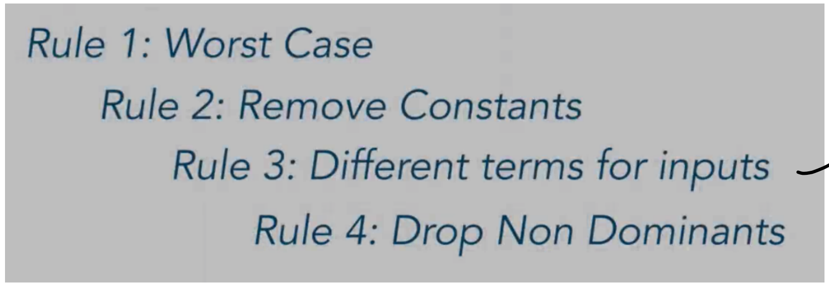
- If number of operations increase as the input increase
- Linear complexity
- Most common.



O(1)

- Constant time





Rule 1: Worst Case

Rule 2: Remove Constants

Rule 3: Different terms for inputs

Rule 4: Drop Non Dominants

If the inputs are diff(ary1, ary2)

→ we have to consider $O(a+b)$ or $O(a \times b)$

$O(n * n)$

O(n^2)

- As we see nested loops, we do multiplication
- $n * n$
- Quadratic time

* Rule 4

$$O(n + n^2) \longrightarrow O(n^2)$$

Big O Cheat Sheet:

-Big Os-

O(1) Constant- no loops

O(log N) Logarithmic- usually searching algorithms have log n if they are sorted (Binary Search)

O(n) Linear- for loops, while loops through n items

O(n log(n)) Log Liniear- usually sorting operations

O(n^2) Quadratic- every element in a collection needs to be compared to ever other element. Two nested loops

O(2^n) Exponential- recursive algorithms that solves a problem of size N

O(n!) Factorial- you are adding a loop for every element

Iterating through half a collection is still O(n)
*Two separate collections: O(a * b)*

-What can cause time in a function?-

Operations (+, -, *, /)
Comparisons (<, >, ==)
Looping (for, while)
Outside Function call (function())*

-Rule Book-

Rule 1: Always worst Case
Rule 2: Remove Constants
Rule 3: Different inputs should have different variables. O(a+b). A and B arrays nested would be O(a*b)
+ for steps in order
* for nested steps
Rule 4: Drop Non-dominant terms

-What causes Space complexity?-

Variables
Data Structures
Function Call
Allocations

https://www.bigocheatsheet.com/

3 pillars of code:

- Readable
- Memory (space complexity)
- Speed (time complexity)

Space Complexity

Heap - area where we store assigned values.

Stack - where we track the funtions calls.