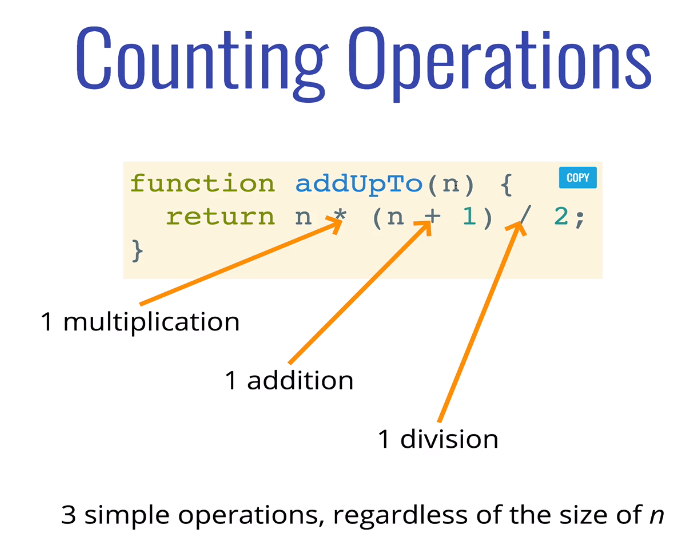
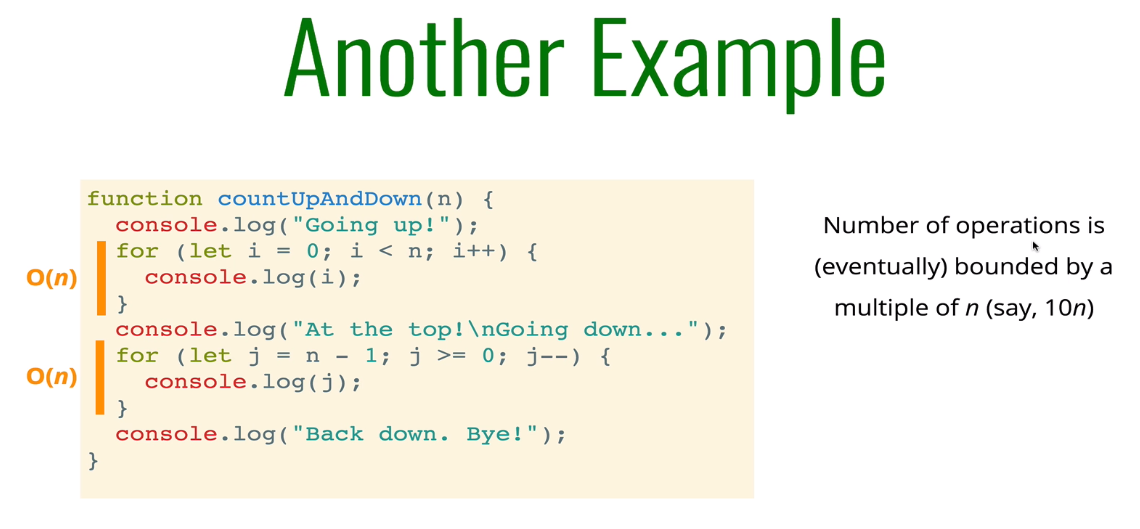
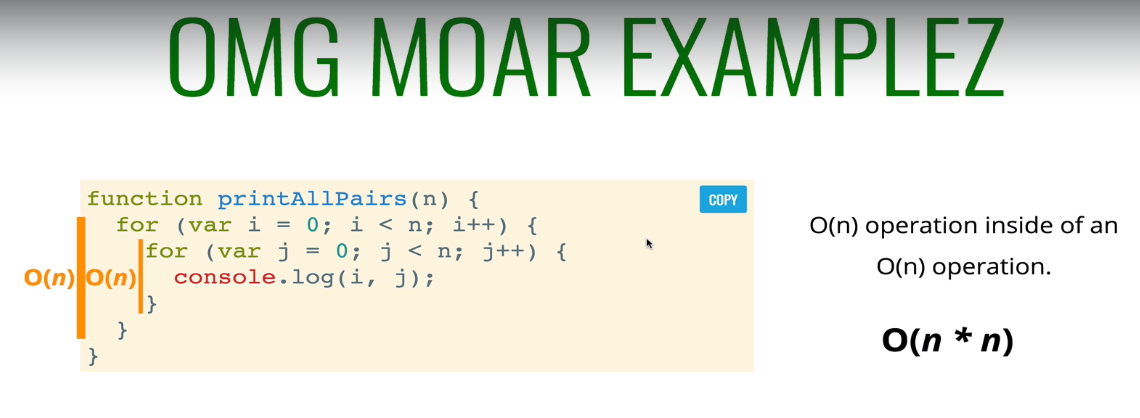
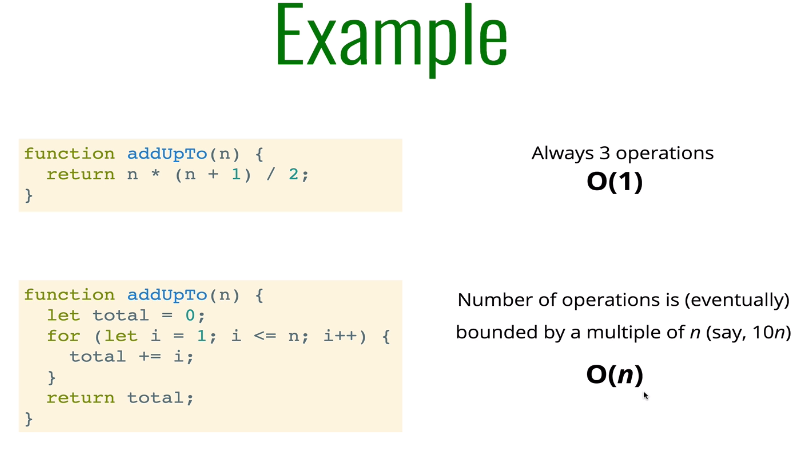
JavaScript Algorithms and Data Structures Masterclass

# Section 2: Big O Notation

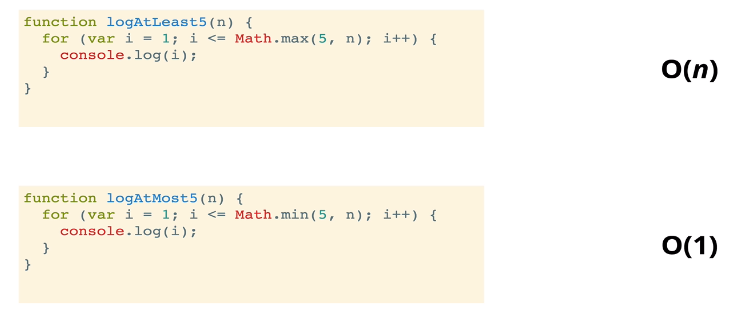
## Lessons

* Ways to record:
  + Time (but can be unreliable)
  + Complexity of code
* We could count number of simple operations the computer performs
  + 

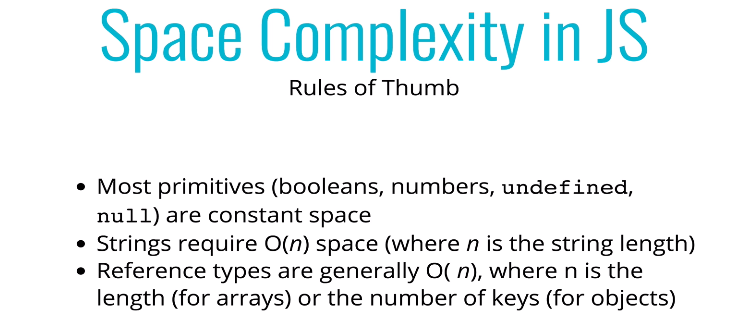
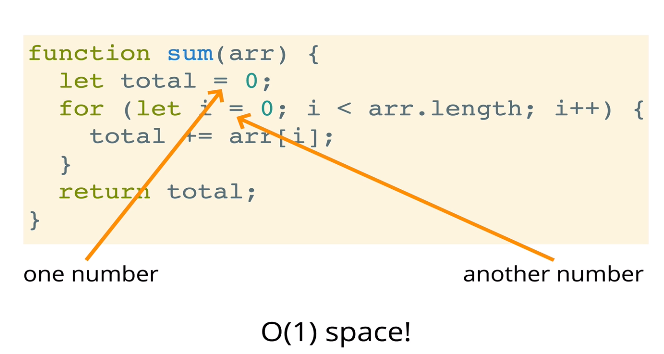
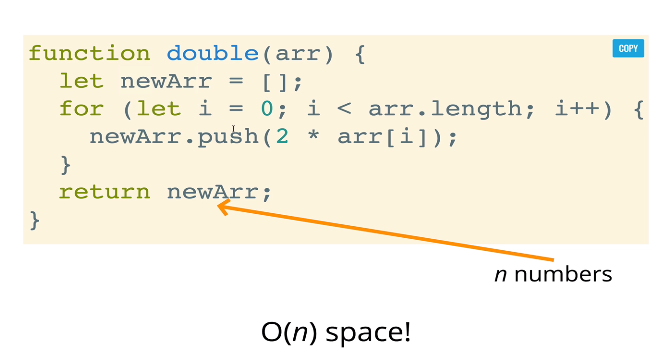
### Intro to Big O

* + How runtime of an algorithm grows as the inputs grow (O(f(n))
  + f(n) can be:
    - Linear
      * f(n) = n
        + 
    - Quadratic (ex. nested loops)
      * f(n) = n2
        + 
    - Constant
      * f(n) = 1
        + 
    - Or Other

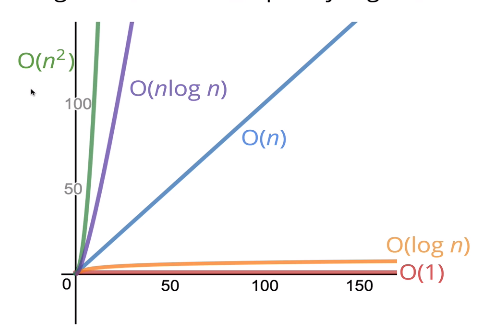
### Simplifying Big O

* Notes:
  + Constants don’t matter (ex. O(2n + 10) --> actually just O(n))
  + Smaller Terms don’t matter (Ex. O(n + 10) --> actually just O(n))
    - ex. O(n2 + 5n + 8) --> actually just O(n2)
* **Big O Shorthand**
  + Arithmetic operations are constant (time for 1+1 same as 100+100)
  + Variable Assignment is constant
  + Access elements in an array (by index) or object (key) is constant
  + Loop’s complexity = length of loop \* complexity of inside loop (ex. Nested loop = n2)
  + Examples
    - 

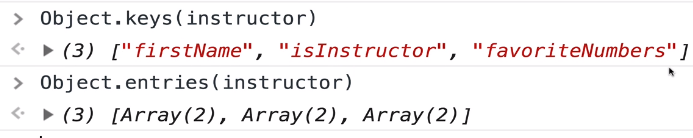
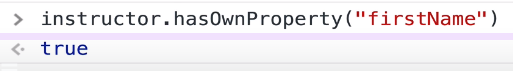
### Space Complexity

* Time complexity = runtime of algo
* Space complexity = additional memory to run algo
  + We are focusing on “auxiliary space complexity” --> space required by algo to run ... not inputs
* Rules of Thumb of Space Complexity
  + 
  + O(1) space:
    - 
  + O(n) space:
    - 

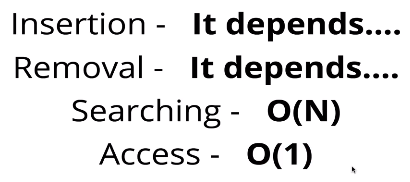
### Logarithms

* Notes, in IT/Comp sci
  + Treat log === log2
    - \*\*\*instead of log10
  + Log of a number roughly measures # of times you can divide a number by 2 until is <= 1
    - ex. 8 can be divided by 2, three times!
* Chart to determine log complexity and speed
  + 
    - O(1) being fastest and O(n2) being slowest

### Big O of Objects

* Time Complexity
  + Insertion = O(1)
  + Removal = O(1)
  + Searching = O(n)
  + Access = O(1)
* Big O of Object Methods
  + Object.keys = O(n)
    - 
  + Object.values = O(n)
  + Object.entries = O(n)AQ
  + Object.hasOwnProperty = O(1)
    - 

### Big O of Arrays (Ordered List)

* 
* Insertion
  + Depends on *WHERE* you are inserting
    - Pushing on **end** of array is “constant/O(1)”
    - Insert at **beginning** of array is “linear/O(n)”
* Big O of Array Methods
  + 