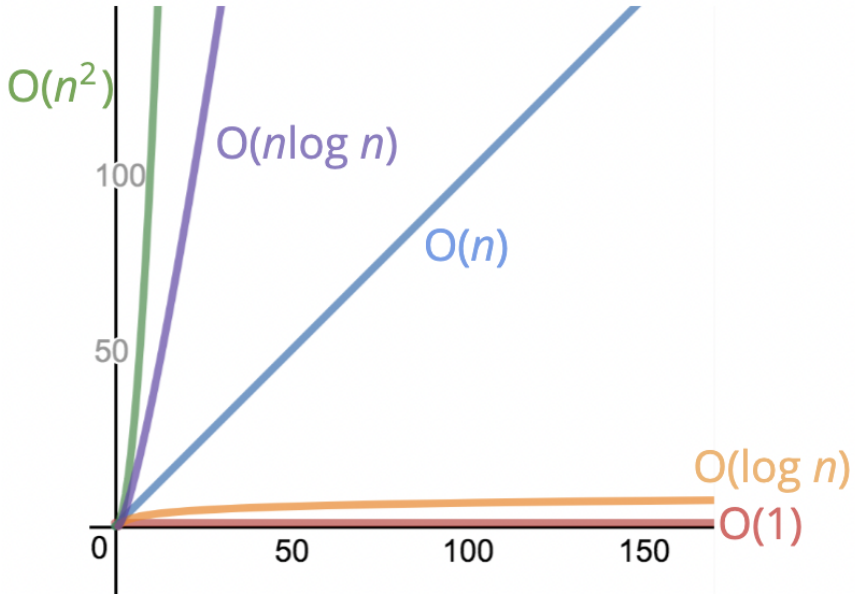
S2.Big O

Logarithms

1. Some big O expressions might not be able to be deduced to 0(1), O(2), 0(n^2)
2. What is a log?
    a. log finds the matching exponent
    b. log_2(8) = 3 → 2^3 = 8
    c. log_y(value) = exponent → y^exponent = value
3. In big O, we care about the big picture, so we ommit the base
    a. log == log_x
4. 0(log n) performance
    a. Run time is very good
    b. It is just slightly higher than O(1)



5. Why use O(log n)
    a. Some algorithms have logarithmic complexity

b. Recursion algorithms

Analyzing Performance of Arrays and Objects

1. Objects (through the lenses of big O)
    a. no order = faster
    b. everything floats in a geletinous mass
    c. Use an object if you don't need order, it is better for performance

    Insertion - O(1) - insert at the end

    Removal - O(1) - remove memory key

    Searching - O(N) [we have to check every key and it's value]

    Access - O(1) - access memory key

    Object.keys - O(N) - it iterates throught each key (more inputs, more operations)

    Object.values - O(N) - it iterates throught each value

    Object.entries - O(N) - it compiles throught each key value pair

    hasOwnProperty - O(1) - it just tells us if it has it or not

2. Arrays
    a. Ordered lists (index)

    Insertion - O(1 | n) - if .push (at at the end), no difficult, but if adding at the begging, we have to change all indexes

    Removal - O(1 | n) - "" vs removing from front vs end

    Searching - O(N) [we have to check every key and it's value]

    Access - O(1) - it only has to access the one index give, it does not have to count/access all the inputs, it jump directly to the index

push - O(1) - it iterates throught each key (more inputs, more operations)

pop - O(1) - it iterates throught each value

Object.entries - O(N) - it compiles throught each key value pair

hasOwnProperty - O(1) - it just tells us if it has it or not

shift - **O(N) -** lead to more operations as array/input grows (index changes)

unshift - **O(N) -** lead to more operations as array/input grows (index changes)

concat - **O(N) -** lead to more operations as array/input grows (index changes)

slice - **O(N)** (index changes)

splice - **O(N)** (index changes)

sort - **O(N * log N)**

forEach/map/filter/reduce/etc. - **O(N)** (iterates, comes through indexes)