Algorithms and Databases

Only worst case is considered

```
function countUpAndDown(n) {
console.log("Going up!"); (1 time)
for(let i = 0;i < n; i++) (N time for whole for loop) {
console.log(i)
}
for(let j = n - 1; j>0; j==) (N time) (N time) {
console.log(j}
}
console.log("Going down"); (1)
}
2n+2 n time simplified to O(n) time

function printAllPairs(n) {
for (var i = 0; i < n; i++) {
for (var j = 0; j < n; j++) {
console.log(i, j);
}
}
}
(square as N is within N)
If I is 2, I is run twice, and J is run 4 times
```

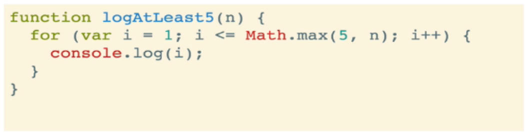
If runtime triples, N time is also going to roughly triple.
N time is proportional to input

Rules of thumb
<u>Constants don't matter.</u>
O(2n) is O(n)
O(500) is O(1)
O(13n^2) is O(n^2)
<u>Smaller terms dont matter</u>
O(1000n + 10) is O(n)
O(n^2 +5n +8) is O(n^2)

This is because in the longterm 5n doesn't matter.
We think for worst case scenario, if n is 1000 then n square is 1000000 while 5n is simply 5000.
We disregard 5000 and 8 as they are small numbers and take away from the bigger runtime

Big O shorthands

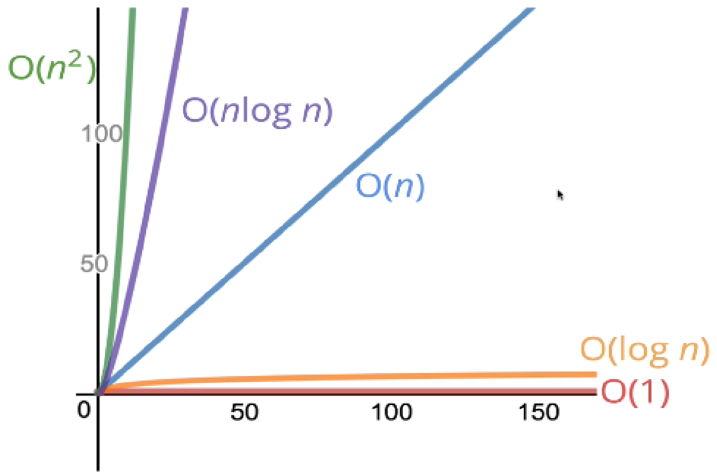
1. **Arithmetic operations are constant**
2. **Variable assignment is constant**
3. **Accssing elements in an array (by index) or object (by key) is constant**
4. **In a loop, the complexity is the length of of the loop times the complexity of whatever happens inside the loop**



```
function logAtLeast5(n) {
  for (var i = 1; i <= Math.max(5, n); i++) {
    console.log(i);
  }
}
```

O(*n*)

As in worst case loop runs at n time if n is bigger than 5



<u>Space Complexity:</u>

Primitives like numbers undefined null and booleans are constant space
Strings are o(n) space
Reference types are generally o(n) were n is the length (for arrays) or number of keys for objects

LEARN HASMAPS, OBJECTS AND MAPS TO SOLVE THESE INTERNSHIP PROBLEMS

Big O of Arrays and Objects

Big O of Objects

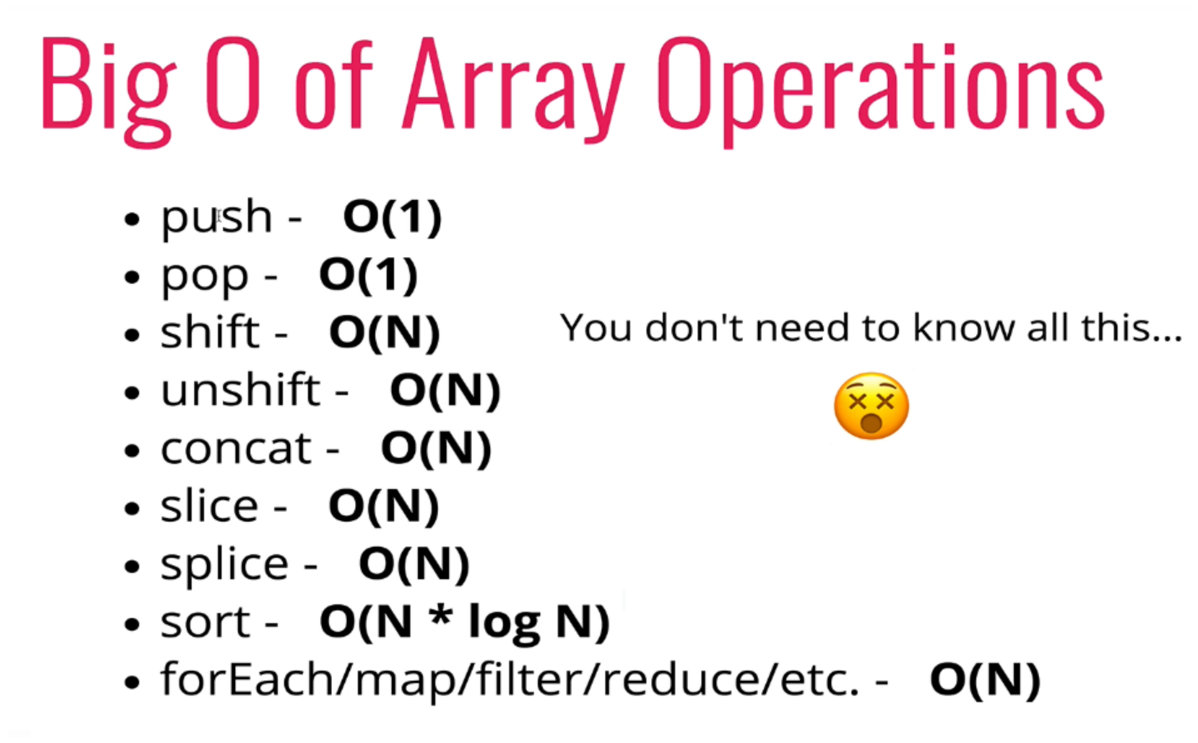
- There is no order in an object

- Insertion - O(1)
- Removal - O(1)
- Searching - O(n)
- Accessing - O(1)

Big O of Objects

- Object.keys -O(N)
- Object.values - O(N)
- Object.entries - O(N)
- hasOwnProperty - O(1)

Big O of Arrays

- Insertion - It depends. Pushing to end is O(1) as only one operation. If pushed to beginning its O(N) as indices have to be rearranged
- Removal- It depends
- Searching - O(N)
- Access - O(1). We access by going to index and JavaScript just jumps to that index
- 

Big O of Array Operations

- push - **O(1)**
- pop - **O(1)**
- shift - **O(N)**
- unshift - **O(N)**
- concat - **O(N)**
- slice - **O(N)**
- splice - **O(N)**
- sort - **O(N * log N)**
- forEach/map/filter/reduce/etc. - **O(N)**

You don't need to know all this... 😵

- array.from is used to split an array's characters

```
Some Array Methods

const arr = [1,2,3,4,5,6]
for(let key of arr){
console.log(key, arr)
} //console.log of key will result in 1,2,3,4 aka array values by index
```

```
const arr = [1,2,3,4,5,6]
for(let key in arr){
console.log(key, arr)
} //console.log of key will result in 0,1,2,3,4 aka index

array.from is used to split an array's characters
- let str = "Ali"
console.log(array.from(str))
//output = ["A","l","i"]
- for(i of array){
console.log(array[i])
}

is equivalant to

- for(let i = 0; array.length; i++){
console.log(array[i])
}

Array.filter
//filters and returns an array of filtered results from array of objects
const items = [
{Name:'Apple',Price:200},
{Name:'Cat',Price:100}
]
const filteredArray = items.filter((item) => {
return item.price <= 100
})

Array.map
//returns array of desired property from array of objects
const items = [
{Name:'Apple',Price:200},
{Name:'Cat',Price:100}
]
const mappedArray = items.map((item) => {
return item.price
})

Array.find
```

```
//returns first instance of desired object property from array of objects
const items = [
{Name:'Apple',Price:200},
{Name:'Cat',Price:100}
]
const mappedArray = items.find((item) => {
return item.name = "Cat"
})
//output: {name:'cat', price:'100'}

Array.foreach
//basically loops through array of objects
const items = [
{Name:'Apple',Price:200},
{Name:'Cat',Price:100}
]
mappedArray = items.map((item) => {
console.log(item.price)
})

Array.some
//returns true when first instance of desired request is found
const items = [
{Name:'Apple',Price:200},
{Name:'Cat',Price:100}
]
const someArray = items.some((item) => {
return item.price >=100
}) //result true

Array.every
//returns true if every instance of desired request is found
const items = [
{Name:'Apple',Price:200},
{Name:'Cat',Price:100}
]
const everyArray = items.every((item) => {
return item.price >=100
}) //result true

Array.reduce
```

```
//combines array elements into a single value using a reducer function
const items = [
{Name:'Apple',Price:200},
{Name:'Cat',Price:100}
]
const total = items.reduce((currentTotal, item) => {
return item.price + currentTotal
}, 0) //result true

Array.includes
//returns true or false on if an array contains a value
const array = ["A","B",3]
connsole.log(array.includes("A")) /returns true
```

Hash Table

- Key value look up
- Gives you a way of associating a value to a key for very quick look ups
- O(1) for good hash tables and O(N) for bad hash table. It depends
- Key and value can be any type of data structure
-