**DSA course note!!**

1. Big O notation

* It allows us to talk formally about how the runtime of an algorithm grows as the inputs grow
* Only the broad trends!

**Big 0**

**Time compelxity(general rules )**

1 . Arithmetic operations are constant

2. Variable assignment is constant

3. Accessing elements in an array (by index) or object (by key) is constant

4. In a loop, the the complexity is the length of the loop times the complexity of whatever happens inside of the loop

**Space complexity(general rules)**

**Rules of Thumb**

1. Most primitives (boolean, numbers, undefined,null) are constant space
2. Strings require O(n) space (where n is the string length
3. Reference types are generally 0( n), where n is the length (for arrays) or the number of keys (for objects)

**When to use objects**

1. When you don't need order
2. When you need fast access / insertion and removal

**….………………….………………….………………….………………………………………**

1. **General understanding!!**

**When to use objects**

1.When you don't need order

2. When you need fast access / insertion and removal!

**Big 0 of Objects**

1. Insertion- 0(1)
2. Removal- 0(1)
3. Searching- O(N)
4. Access- 0(1)

**When you don't need any ordering,**

**objects are an excellent choice!**

**Big 0 of Object Methods**

Object.keys- O(N)

Object.values- O(N)

Object.entries- O(N)

hasOwnProperty- 0(1)

**Big 0 of an ARRAY**

* Adding and removing from the beginning of an Array is best avoided(as its time complexity becomes 0(n) because we are re-indexing every element but if we are pushing and popping from the end no re-indexing so time complexity is O(1))

**NB PUSH AND POP IS ALWAYS FASTER THAN SHIFT AND UNSHIFT**

**Time complexity of array operations**

Insertion (it depends, we talking pushing? of unshifting?)

Removing(it depends, we talking popping or shifting??)

Searching O(n)

Accesing O(1)

push- 0(1)

pop- 0(1)

shift- O(N)

unshift- O(N)

concat- O(N)

slice- O(N)

splice- O(N)

sort- O(N \* log N)

for Each/map/filter/reduce-O(N)