# Analyzing Performance of Arrays and Objects

#### 1. Introduction

#### A. Objectives

- Understand how arrays and objects work through the lens of Big 0
- Explain why adding elements to the beginning of an array is costly
- Compare and contrast the runtime for arrays and objects, as well as built-in methods

#### 2. The Big O of Objects



## Unordered, key value pairs!

```
let instructor = {
    firstName: "Kelly",
    isInstructor: true,
    favoriteNumbers: [1,2,3,4]
}
```

#### When to use objects

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

#### A. Big O of Objects

# Big O of Objects

Insertion - O(1)

Removal - **O(1)** 

Searching - O(N)

Access - O(1)

When you don't need any ordering, objects are an excellent choice!

- Objects are very fast.
- Hash maps explain how objects work behind the scenes
- Searching -0 (n), does NOT refer to searching for a KEY; rather, it refers to searching for information within the value.

#### B. Big O of Object Methods

Object.keys - **O(N)** 

Object.values - O(N)

Object.entries - O(N)

hasOwnProperty - **O(1)** 

### 3. When are Arrays Slow?

# ARRAYS Ordered lists!

```
let names = ["Michael", "Melissa", "Andrea"];
let values = [true, {}, [], 2, "awesome"];
```

#### A. When to use Arrays

- · When you need order
- When you need fast access/inserting and removal(sort of...)

Arrays aren't the only ordered data structure. Sometimes, singlely/doubley linked list perform better than arrays

#### B. Big O of Arrays

# Big O of Arrays

Insertion - It depends....

Removal - It depends....

Searching - O(N)

Access - O(1)

Let's see what we mean by that!

- When you access an element in an array using the index, JavaScript doesn't access the
  element by counting up to the n<sup>th</sup> index value, and then returning the element; instead it
  jumps to the index and returns the value, which is why Big O of array access is 0(1).
- Insertion and Removal depends on where you insert/remove:

- insert at end is 0 (1)
- insert at beginning is 0 (N) (requires re-indexing)
- o remove from beginning is O(N) (requires re-indexing)
- remove from end is 0 (1)
- push/pop (end) <u>FASTER</u> than shift/unshift (beginning)
- Searching is 0 (N)

#### C. Big O of Array Operations

- push **O(1)**
- pop **O(1)**
- shift O(N) You don't need to know all this...
- unshift O(N)
- concat O(N)
- slice **O(N)**
- splice **O(N)**
- sort O(N \* log N)
- forEach/map/filter/reduce/etc. O(N)

## Recap

- Object are fast, but there's no order.
- Arrays are great when you need order and plan on inserting and removing only at the end of an array.
- Inserting/removing from the beginning and the middle of an array will cause re-indexing of the array.