**Hash Tables**

Searching is fast with hash tables making it great for databases.

insert O(1) – this makes hash tables better than arrays. Arrays insert = O(n)

lookup O(1)

delete O(1)

search O(1) - Searching is fast with hash tables making it great for databases.

    // Looping through keys will result in BigO = O(n) because of the for loop that loops over every key.

    keys() {

        // create an empty array to store each key in our hashmap

        const keysArray = [];

        console.log(this.data.length);

        // loop through the hashmap (ALL 50 indexes in the array) causing O(n)

        for (let i = 0; i < this.data.length; i++) {

            if (this.data[i]) {

                // i = the index of the array.  [0][0] = the first element in each array which is the KEY

                keysArray.push(this.data[i][0][0])

            }

        }

        return keysArray;

    }

}

note\* because we must loop through every item in a hash table to get the keys, using objects is not a very good practice because it causes the bigO = O(n)

**Advantages**

* Fast lookups – Good collision resolution needed
* Fast inserts
* Flexible Keys – use key/value pairs in hast tables

**Disadvantages**

* Unordered – not ordered unlike arrays
* Slow key iteration – getting all keys, we need to iterate through the entire hash table which results in O(n)

\*Hash tables are fast at accessing data but require more memory to code. By using hash tables, we can decrease our time complexity to O(n) however we increase our space complexity because we need to create a hash table first then add the items to it.