# Array

* **Static Array**

Push, Lookup = O(1)

Insert, Delete= O(n)

* **Dynamic Array**

Lookup, append (can be O(n) when array double its size) = O(1)

Insert, delete = O(n)

Advantage: Arrays are faster for lookup, push or pop (at the end of the array) and ordered

Disadvantage: Slow insert and delete, can be fixed size if using static array.

# Hash tables

* They are also known as HashMap, Maps (Java), Unordered Map, Dictionary (Python/C++), Objects (JavaScript) in different languages.

It is a **key value** pair; the key is passed into a hash function and it returns an index for that key. This usually takes O (1) time.

* **Insert**, **delete**, **lookup** (can occasionally take O(n) depending on the hash function, but languages takes care of it and usually O(1)), **Search** is O(1)
* It can have Collision given there is enough data, so in case same address for two key there is many ways to deal with this.
  + **Separate Chaining:** The collided data is stored through linked list (one of many implementations of sperate chain).
  + **Linear Probing:** In case of collision, it scans for the next free slot in the bucket.

Due to this issue, lookup can be not efficient.

**Disadvantage**: there is no order for insertion. Slow key iteration (get all keys)

**Advantage**: Fast Lookup (good collision resolution needed), insert, flexible keys.

# Linked List

This data structure has a value and a pointer to next node in the list. And also have a head and tail. Tail points to null. It is a low-level data structure and thus not available in some languages (JavaScript).

* **Singly Linked List:** Append O(1), Lookup, Insert, Delete O(n)
  + Can’t iterate in backward, takes less memory used for mostly fast insertion and deletion
* **Doubly Linked List:** Prepend and Append O(1), Lookup, Insert, Delete O(n)
  + It can be iterated in both ways, have option to prepend easily. Requires more memory and storage.

**Advantage:** Fast insert and delete, ordered, flexible size

**Disadvantage:** slow lookup, more memory.

# Stack and Queue:

* **Stack**: **lookup** O(n) and **push/pop/peek** O(1). LIFO structure
* **Queue**: **lookup** O(n) and **enqueue/dequeue/peek** O(1). FIFO structure