[Data Structure Basics 1](#_Toc53904866)

[Array 1](#_Toc53904867)

[Linear Arrays 1](#_Toc53904868)

[Dynamic Arrays 2](#_Toc53904869)

[Two Dimensional Arrays 2](#_Toc53904870)

[Big O efficiency 2](#_Toc53904871)

[Linked List 2](#_Toc53904872)

[Doubly Linked List 2](#_Toc53904873)

[Circular Linked List 3](#_Toc53904874)

[Stack 3](#_Toc53904875)

[Queues 3](#_Toc53904876)

[Big O efficiency 3](#_Toc53904877)

[Hash Table or Hash Map 3](#_Toc53904878)

[Hash functions: 3](#_Toc53904879)

[Hash Collisions 4](#_Toc53904880)

[Big O Efficiency 4](#_Toc53904881)

[Binary Tree 4](#_Toc53904882)

# Data Structure Basics

## Array

1. Stores data elements based on a sequential, most commonly 0 based index.
2. Based on tuples from set theory
3. They are one of the oldest and most commonly used data structures.
4. Optimal for indexing
5. Bad at searching, inserting and deleting (except at the end)

### Linear Arrays

1. One dimensional arrays are the most basic
2. Static in size, meaning that they are declared with a fixed size.

### Dynamic Arrays

1. One dimensional arrays but have reserved space for additional elements.
2. If a dynamic array is full, it copies it’s content to a larger array.

### Two Dimensional Arrays

1. Have x and y indices like grid or nested arrays.

### Big O efficiency

1. Indexing:

Linear array: O(1)

Dynamic array: O(1)

1. Search:

Linear array: O(n)

Dynamic array: O(n)

1. Optimized Search

Linear array: O(logn)

Dynamic array: O(logn)

1. Insertion

Linear array: n/a

Dynamic array: O(n)

## Linked List

1. Stores data with nodes that point to other nodes.
2. Nodes at its most basic it has data and one reference to another node.
3. A linked list chains node together by pointing one node’s reference towards another node.
4. Designed to optimize insertion and deletion, slow at indexing and searching.

### Doubly Linked List

1. Has nodes that reference the previous node.

### Circular Linked List

1. Simple linked list whose tail means the last node references head that is first node.

### Stack

1. Commonly implemented with linked lists but can be made from arrays too.
2. Stacks are Last in First out(LIFO) data structures.
3. Made with a linked list by having head be the only place for insertion and removal.

### Queues

1. Can be implemented using linked list or an array.
2. Queues are First in, First out (FIFO) data structure.
3. Made with doubly linked list that only removes from head and adds to tail.

### Big O efficiency

1. Indexing:

LinkedList: O(n)

1. Search:

LinkedList: O(n)

1. Optimized Search:

Linked Lists: O(n)

1. Insertion:

Linked Lists: O(1)

## Hash Table or Hash Map

1. Stores data with key value pairs.

## Hash functions:

1. Accept a key and return an output unique only to that specific key.
2. This is known as hashing, which is the concept that an input and an output have a one-to-one correspondence to map information.
3. Hash functions return a unique address in memory for that data.
4. Designed to optimize searching, insertion and deletion.

### Hash Collisions

1. Hash collisions happen when a hash function returns the same output for two distinct inputs.
2. All hash function has this problem.
3. This is often accommodated by having hash tables to be very large.
4. Hashes are important for associative arrays and database indexing.

### Big O Efficiency

1. Indexing:

Hash Tables: O(1)

1. Search:

Hash Tables: O(1)

1. Insertion:

Hash Tables: O(1)

## Binary Tree