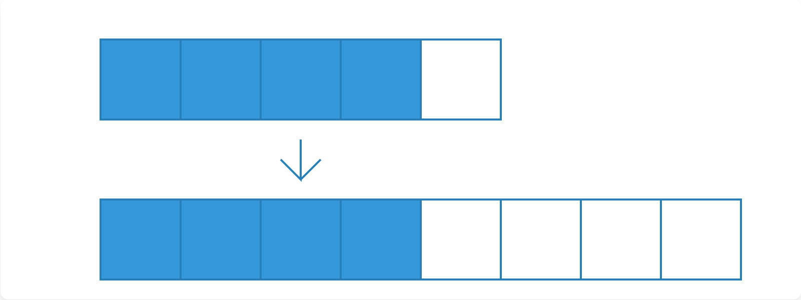
CSE 212 Data Structure Cheat Sheet

By Matheus Felber

Dynamic Array



Purpose and Example:

A Dynamic Array is an array that can resize itself automatically when more space is needed. Unlike a fixed-size array, it grows by allocating a new, larger array and copying elements over.

Example: Used in text editors to store an expanding document or game inventories where items can be added dynamically.

Time Complexity:

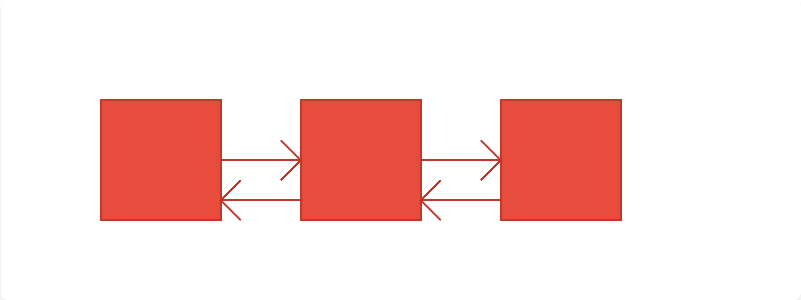
Insert at beginning: O(n) (shifting elements)

Insert at end: Amortized O(1) (resizing occasionally)

Find by value: O(n)

Find by index: O(1)

Linked List (Doubly-Linked)



Purpose and Example:

A Doubly Linked List is a collection of nodes where each node has two pointers—one pointing to the next node and one pointing to the previous node.

Example: Used in browser history navigation or music playlists, where you need to move back and forth easily.

Time Complexity:

Insert at beginning: O(1)

Insert at end: O(1)

Find by value: O(n)

Find by index: O(n) (sequential traversal)

Stack

Purpose and Example:

A Stack follows the LIFO (Last In, First Out) principle, meaning the last item added is the first to be removed.

Example: Used in undo/redo features in word processors or function call stacks in programming.

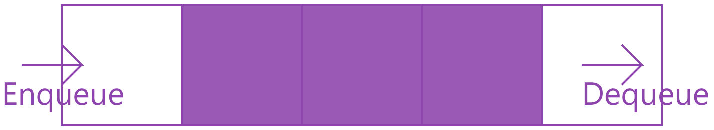
Time Complexity:

Push (Insert at top): O(1)

Pop (Remove from top): O(1)

Access middle elements: O(n) (must pop elements to reach it)

Queue



Purpose and Example:

A Queue follows the FIFO (First In, First Out) principle, meaning the first item added is the first to be removed.

Example: Used in task scheduling, such as a printer queue or customer service lines.

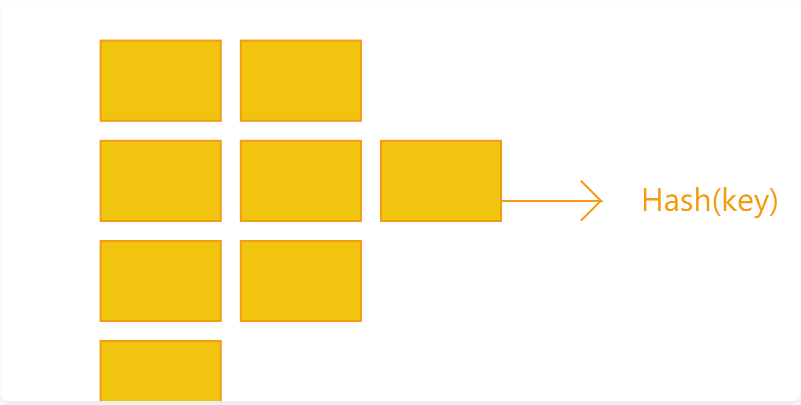
Time Complexity:

Enqueue (Insert at back): O(1)

Dequeue (Remove from front): O(1)

Access middle elements: O(n) (must traverse)

Map (Hash Table)



Purpose and Example:

A Map (Hash Table) is a data structure that stores key-value pairs and provides fast access to values based on their keys.

Example: Used in databases for indexing or storing user login information.

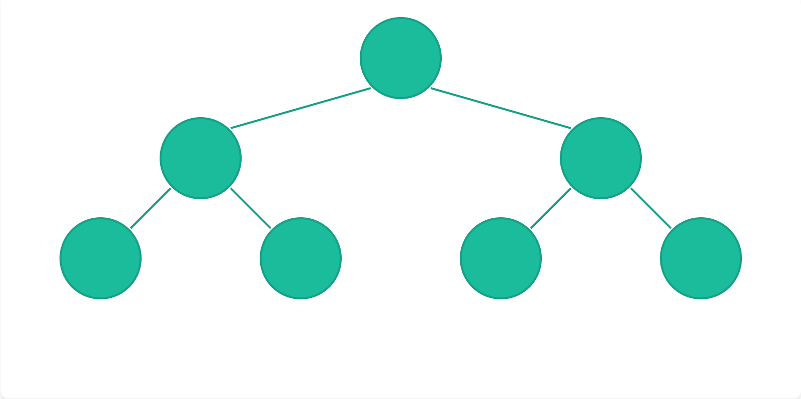
Time Complexity:

Insert: O(1) (in most cases, unless there are collisions)

Find by key: O(1) (direct lookup)

Find by value: O(n) (must check all values)

Balanced BST (Binary Search Tree)



Purpose and Example:

A Balanced BST maintains its height to ensure efficient searching, insertion, and deletion. Common types include AVL trees and Red-Black trees.

Example: Used in database indexes or auto-suggestion features in search engines.

Time Complexity:

Insert: O(log n)

Find by value: O(log n)

Find by index: O(log n) (if implemented with an augmented tree)