**Theory**

Array

+ Can retrieve items at specific index

- Non dynamic (cannot alter size)

Array List

+ Can retrieve items at specific index

- Pseudo-dynamic (Waste space by reserving max\_size first)

LinkedList

+ True dynamic (Takes memory and removes memory according to add/ remove node)

- Not that good for traversal - O(N), so preferably used for something that doesn’t require constant traversal

LinkedList with back node

+ True dynamic (Takes memory and removes memory according to add/ remove node)

+ Less limited in intended capability / Adaptable

+ Inclusion of back node allows appending new nodes at O(1) time complexity.

- Not that good for traversal - O(N), so preferably used for something that doesn’t require constant traversal

Stack Array

+ Follows first in last out methodology (niche use + limitation on functions for protection)

- Pseudo-dynamic (Waste space by reserving max\_size first)

Stack Ptr

+ True dynamic (Takes memory and removes memory according to add/ remove node)

+ Follows first in last out methodology (niche use + limitation on functions for protection)

- Not intended for traversal / searching

Queue Array

+ Follows first in first out methodology (niche use + limitation on functions for protection)

- Pseudo-dynamic (Waste space by reserving max\_size first)

- Incredibly inefficient through the need to shift every item forward in the queue hence a O(N) for removing from queue.

Queue Circular Array

+ Follows first in first out methodology (niche use + limitation on functions for protection)

+ Accounts for inefficiency in shifting items forward by shifting index accordingly instead

- Pseudo-dynamic (Waste space by reserving max\_size first)

- Wastes 1 memory space, so use it for simple datatypes

Queue Circular Array with counter

+ Follows first in first out methodology (niche use + limitation on functions for protection)

+ Accounts for inefficiency in shifting items forward by shifting index accordingly instead

- Pseudo-dynamic (Waste space by reserving max\_size first)

- Uses a counter hence affecting time and performance slightly, so use it for complex datatypes

Queue Ptr

+ Follows first in first out methodology (niche use + limitation on functions for protection)

+ No inefficiency in shifting items forward

+ True dynamic (Takes memory and removes memory according to add/ remove node)

- Not intended for traversal / searching

Hash Table Perfect

+ O(1) time complexity for retrieving

+ Does not need to store keys hence saving storage

- Not viable for large max\_size/ large amt of data

- Pseudo-dynamic for the array (Waste space by reserving max\_size first)

- Requires unique keys and unique hashing results

Hash Table Imperfect

+ O(1) time complexity + O(N) from linked list, however linked list size is reduced

+ True dynamic for the linked list part (Takes memory and removes memory according to add/ remove node)

- Pseudo-dynamic for the array (Waste space by reserving max\_size first)

- Stores additional data known as keys and overall uses more space to store pointers etc.

- Requires unique keys

Do note that if any changes are meant to queues and stacks, i.e removing from middle of queue, the code for it is relatively simple as queues are simply more limited linked lists, so just get from linked list and adapt it slightly.

**Conversion**

(int) character

string.push\_back(char)

string += char

string.append(times,char)

string.insert(index,times,char)

int integer = atoi(string)

**Building Blocks**

In the unfortunate event that we need to draw diagrams into Microsoft word.

Pointers and Nodes

FirstNode

Array