Queues

Definition 0.1

A **queue** is a first in, first out abstract data type. Thus, queue and dequeue operations occur at *opposite* ends of the structure

The main operations for queues include:
- **enqueue(data)** - adds data to the "back" of the queue
- **dequeue()** - removes the data from the front of the queue
- **peek** - returns the data at the front without removing it

SLL Backed Queue

Note 0.1

The SLL-backed queue requires a *tail pointer* in order to get O(1) operations.

The "front" of the queue is the front of the list where data is dequeued from, while the "back" of the queue is the back of the list where data is enqueued

enqueue(data) -> **addToBack(data)**, and **dequeue()** -> **removeFromFront()**

Array Backed Queue

Note 0.2

Array backed queues require a size variable but also a front variable, because *the array behaves circularly.*

arr[front] is the front of the queue, and **arr[(front + size) % arr.length]** is the first empty index at the "back"

For **enqueue**
- Put the element at arr[(front+size) % arr.length] then size++

For **dequeue**
- Remove the element at arr[front], increment front and decrement size
- In this case, front = (front + 1) % arr.length when you increment so that *front never goes out of bounds*

Dequeue

Note 0.3

In Deques (double ended queues), we can add and remove from either side of the deque

The main operations include:
- **addFirst(data)**
- **addLast(data)**
- **removeFirst()**
- **removeLast()**

DLL Backed Queue

Note 0.4

The DLL backed queue requires a tail.

addFirst(data) -> **addToFront(data)**: O(1)

addLast(data) -> **addToBack(data)**: O(1)

removeFirst() -> **removeFromFront()**: O(1)

removeLast() -> **removeFromBack()**: O(1)

Array Backed Deque

Note 0.5

Uses a front variable and a size variably (*circular again*)

Important Indices

- arr[(front - 1) % capacity] = addFirst()
- arr[front] = removeFirst()
- arr[(front + size) % capacity] = addLast()
- arr[(front + size - 1) % capacity] = removeLast()