**Stack**

*Pseudocode for Stack (add to the start, remove from the start):*

push:

* The function should accept a value
* Create a new node with that value
* If there are no nodes in the stack, set the first and last property to be the newly created node
* If there is at least one node, create a variable that stores the current first property on the stack
* Reset the first property to be the newly created node
* Set the next property on the node to be the previously created variable
* Increment the size by 1

pop:

* If there are no nodes in the stack, return null
* Create a temporary variable to store the first property on the stack
* If there is only 1 node, set the first and last property to be null
* If there is more than 1 node, set the first property to be the next property on the current first
* Decrement the size by 1
* Return the value of the node removed

*Full Code:*

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class Stack {

constructor() {

this.first = null;

this.last = null;

this.size = 0;

}

push(val){

var newNode = new Node(val);

if (!this.first) {

this.first = newNode;

this.last = newNode;

} else {

var oldFirst = this.first;

this.first = newNode;

this.first.next = oldFirst;

}

return ++this.size;

}

pop() {

if (!this.first) {

return null;

} else {

var oldFirst = this.first;

if (this.first === this.last) {

this.last = null;

} else {

this.first = oldFirst.next;

oldFirst.next = null;

}

this.size--;

return oldFirst.val;

}

}

}

*Big O:*

Insertion – O(1)

Removal – O(1)

Searching – O(N)

Access – O(N)

**Queue**

*Pseudocode for Queue (add to the end, remove from the start):*

enqueue:

* This function accepts some value
* Create a new node using the value passed to the function
* If there is no node in the queue, set the first and last to be the newly created node
* Otherwise, set the next property on the current last to be that node, and then set the property of the queue to be that node
* Increment the size by 1

dequeue:

* If there is no first property, return null
* Store the first property to the variable
* See if the first is the same as the last (check if there is only 1 node). If so, set the first and last to be null
* If there is more than 1 node , set the first property to be the next property of first
* Decrement the size by 1
* Return the value of the node dequeued

*Full Code:*

class Node{

constructor(val) {

this.val = val;

this.next = null;

}

}

class Queue {

constructor() {

this.first = null;

this.last = null;

this.size = 0;

}

enqueue(val) {

var newNode = new Node(val);

if (!this.first) {

this.first = newNode;

this.last = newNode;

} else {

this.last.next = newNode;

this.last = newNode;

}

return ++this.size;

}

dequeue() {

if (!this.first) {

return null;

} else {

var temp = this.first;

if (this.first === this.last) {

this.last = null;

} else {

this.first = this.first.next;

}

this.size--;

return temp.val;

}

}

}

*Big O:*

Insert – O(1)

Removal – O(1)

Searching – O(N)

Access – O(N)