Big O – Measures the worst case scenario.

O(1) – Constant Time

* push() is O(1) operations because we are pushing(adding) to the end of the array therefore the rest of the array does not require re-indexing.

O(n) – Linear or Proportional. One for loop.

* pop() is O(n) because we need to iterate from HEAD to the node that is before the node that we want to remove.
* we use the shift() and unshift() functions to remove or add an element therefore we need to re-index the array.
* shift() – removing the first index of an array
* unshift() – adding element to the first index of the array
* splice(1, 0, ‘Hi’) – adding or removing an element IN the array (not the 1st or last). We need to re-index the elements in the array that are after the element we inserted.

O(n^2) – Loop within a Loop. Nested for loops.

O(log n) – Divide and Conquer. Remove half of the array with each iteration until the item is found.

* 2^n
* 2^3 = 8
* O(log 8) 🡪 8 / 2 = 4 🡪 4 / 2 = 2 🡪 2 / 2 = 1 --- This means that O(log 8) = 3 because we needed to do 3 operations to get 1.

O(a **+** b)

function logItems (a, b) {

    for (let i = 0; i < a; i++) {

        console.log(i);

    }

    for (let i = 0; i < b; i++) {

        console.log(i);

    }

}

This cannot be simplified to O(n) because there are 2 input terms (a, b).

O(a **\*** b)

function logItems (a, b) {

    for (let i = 0; i < a; i++) {

        for (let j = 0; j < b; j++) {

            console.log(i, j);

        }

    }

}

This cannot be simplified to O(n^2) because there are 2 input terms (a, b).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Linked List**

**Remove node from the end of list.**

Start from HEAD and iterate through the entire list to set the tail equal to the last node causing and O(n).

**Adding a node to front of list.**

Set pointer of node to be inserted in the front to the HEAD of the linked list.

Then set the HEAD equal to the new node that was inserted into the front of the linked list resulting in O(1).

**Removing a node from the beginning of list.**

Set HEAD equal to the first nodes node.next value resulting in O(1).

**Inserting a node inside of the linked list.**

Iterate through the linked list to the node that you want to place the new node after.

Set the new node’s pointer to the node.next of that node.

Set the node’s node.next to point to the new node inserted.

Iterating through the linked list results in O(n).

**Removing a node from inside the linked list.**

Iterate through the linked list to the node that you want to remove.

Have the previous node of the node you want to remove point it’s node.next to the node.next of the node you are removing.

Iterating through the linked list results in O(n).

**Finding a node in the linked list by value or by index.**

Iterate through the linked list until you find the node resulting in O(n).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Arrays vs Linked List**

A screenshot of a computer

Description automatically generated

Pop and Lookup by Index are **better** for Arrays.

Shift and Unshift are **better** for Linked Lists.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class Node {

    constructor(value) {

        this.value = value;

        this.next = null;

    }

}

class LinkedList {

    constructor(value) {

        const newNode = new Node(value);

        this.head = newNode;

        this.tail = this.head;

        this.length = 1;

    }

    printList() {

        let temp = this.head;

        while (temp !== null) {

            console.log(temp.value);

            temp = temp.next;

        }

    }

    getHead() {

        if (this.head === null) {

            console.log("Head: null");

        } else {

            console.log("Head: " + this.head.value);

        }

    }

    getTail() {

        if (this.tail === null) {

            console.log("Tail: null");

        } else {

            console.log("Tail: " + this.tail.value);

        }

    }

    getLength() {

        console.log("Length: " + this.length);

    }

    makeEmpty() {

        this.head = null;

        this.tail = null;

        this.length = 0;

    }

    push(value) {

        const newNode = new Node(value);

        if (!this.head) {

            this.head = newNode;

            this.tail = newNode;

        } else {

            this.tail.next = newNode;

            this.tail = newNode;

        }

        this.length++;

        return this;

    }

    pop() {

        if (this.length === 0) return undefined;

        let temp = this.head;

        let pre = this.head;

        while (temp.next) {

            pre = temp;

            temp = temp.next;

        }

        this.tail = pre;

        this.tail.next = null;

        this.length--;

        if (this.length === 0) {

            this.head = null;

            this.tail = null;

        }

        return temp;

    }

    unshift(value) {

        const newNode = new Node(value);

        if (!this.head) {

            this.head = newNode;

            this.tail = newNode;

        } else {

            newNode.next = this.head;

            this.head = newNode;

        }

        this.length++;

        return this;

    }

    shift() {

        if (this.length === 0) return undefined;

        let temp = this.head;

        this.head = this.head.next;

        temp.next = null

        this.length--;

        if (this.length === 0) {

            this.tail = null;

        }

        return temp;

    }

    get(index) {

        if (index < 0 || index >= this.length) return undefined;

        let temp = this.head;

        for (let i = 0; i < index; i++) {

            temp = temp.next;

        }

        return temp;

    }

    set(index, value) {

        let temp = this.get(index);

        if (temp) {

            temp.value = value;

            return true;

        }

        return false;

    }

    insert(index, value) {

        if (index < 0 || index > this.length) return false;

        if (index === this.length) return this.push(value);

        if (index === 0) return this.unshift(value);

        const newNode = new Node(value);

        const temp = this.get(index - 1);

        newNode.next = temp.next;

        temp.next = newNode;

        this.length++;

        return true;

    }

    remove(index) {

        if (index < 0 || index >= this.length) return undefined;

        if (index === 0) return this.shift();

        if (index === this.length - 1) return this.pop();

        const before = this.get(index - 1);

        const temp = before.next;

        before.next = temp.next;

        temp.next = null;

        this.length--;

        return temp;

    }

    reverse() {

        let temp = this.head;

        this.head = this.tail;

        this.tail = temp;

        let next = temp.next;

        let prev = null;

        for (let i = 0; i < this.length; i++) {

            next = temp.next;

            temp.next = prev;

            prev = temp;

            temp = next;

        }

        return this;

    }

}