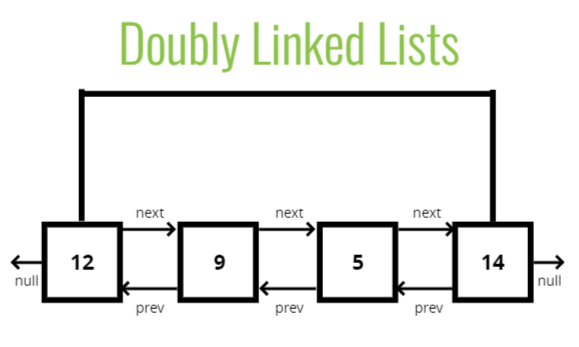
**Doubly Linked List**

All that we do is add a pointer to the prvious node and the next node. So each node points in two directions.

Almost identical to Singly Linked Lists, except every node has another pointer, to the previous node.

No indexing



It’s a tiny change but has a large impact on some of the code we erite and the efficiency of certain operations.

Eg, when we’re popping off the last item in a singly linked list, we had to go through all of the nodes because we had no other way of finding out the 2nd last node.

In doubly linked list, we can just go to the tail and get the 2nd last node from it by moving backward.

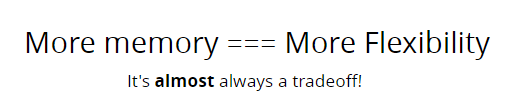
Imagine that you want to print out the singly lined list in reverse. You start at the beginning, go all the way to the end and because you can’t go backward, you start over to go to the 2nd last item then do the same for 3rd last and so on. It’s horribly inefficient.

Singly Linked List:



Doubly Linked List:





Drawback of doubly linked list is that it takes up more memory.

**Node**

* val
* next
* prev

**Doubly Linked List**

* head
* tail
* length

class Node {

  constructor(val) {

    this.val = val;

    this.next = null;

    this.prev = null;

  }

}

class DoublyLinkedList {

  constructor() {

    this.head = null;

    this.tail = null;

    this.length = 0;

  }

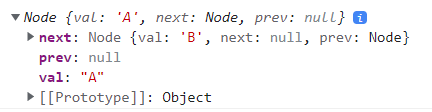
}

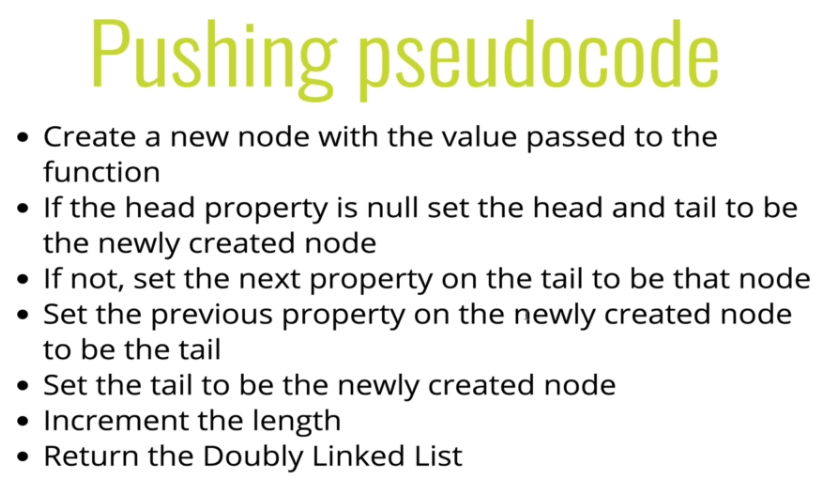
first = new Node("A")

first.next = new Node("B")

first.next.prev = first

console.log(first);





  push(val) {

    let node = new Node(val);

    if (this.length == 0) {

      this.head = node;

      this.tail = node;

    } else {

      let tail = this.tail;

      tail.next = node;

      node.prev = tail;

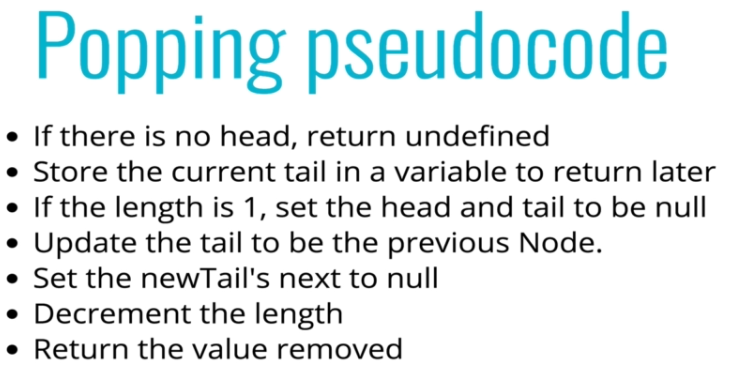
      this.tail = node;

    }

    this.length++;

    return this;

  }



pop() {

    if (this.length === 0) {

      return undefined;

    }

    let oldTail = this.tail;

    if (this.length === 1) {

      this.head = null;

      this.tail = null;

    } else {

      let newTail = oldTail.prev;

      this.tail = newTail;

      newTail.next = null;

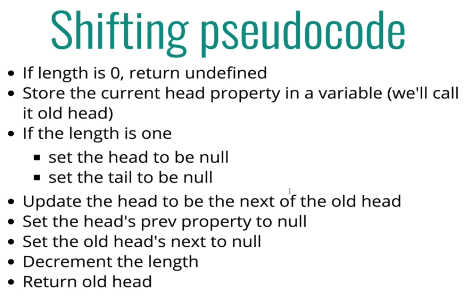
      oldTail.prev = null;

    }

    this.length--;

    return oldTail;

  }



shift() {

    if (this.length === 0) {

      return undefined;

    }

    let oldHead = this.head;

    if (this.length === 1) {

      this.head = null;

      this.tail = null;

    } else {

      let newHead = oldHead.next;

      this.head = newHead;

      newHead.prev = null;

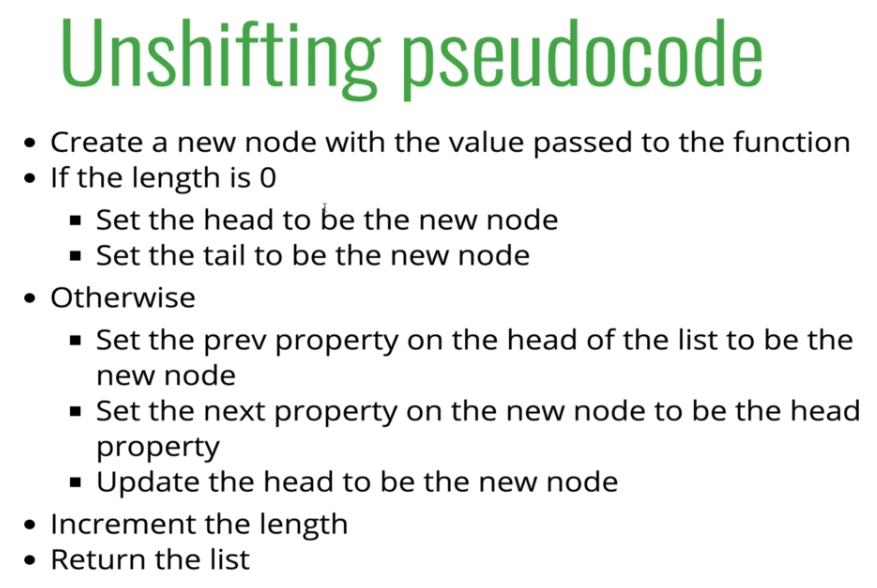
      oldHead.next = null;

    }

    this.length--;

    return oldHead;

  }



unshift(value) {

    let newNode = new Node(value);

    if (this.length === 0) {

      this.head = newNode;

      this.tail = newNode;

    } else {

      let oldHead = this.head;

      oldHead.prev = newNode;

      newNode.next = oldHead;

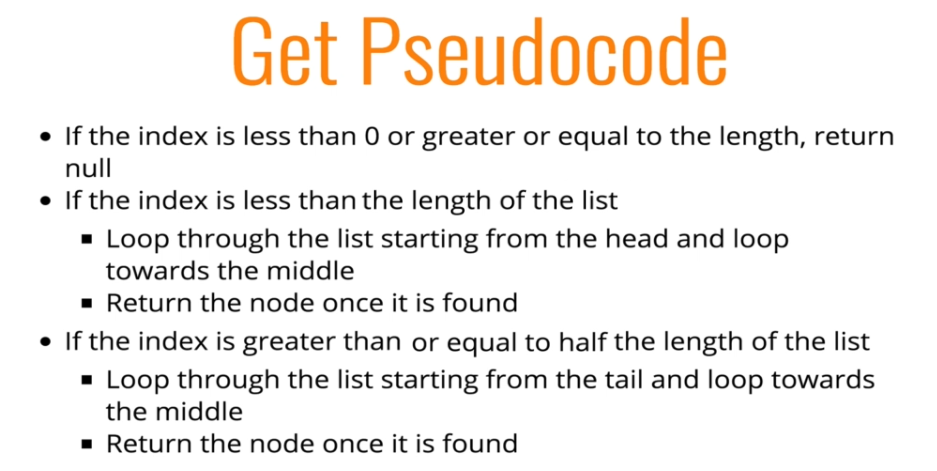
      this.head = newNode;

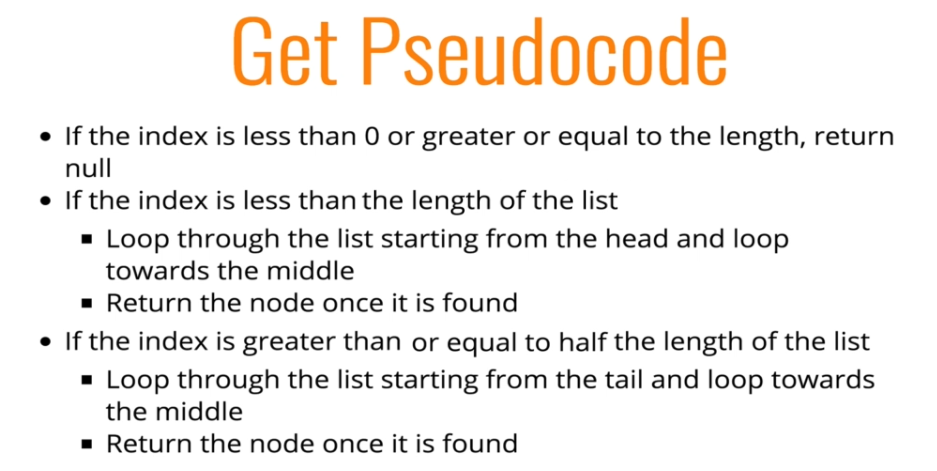
    }

    this.length++;

    return this;

  }





  get(i) {

    if (i < 0 || i >= this.length) {

      return undefined;

    }

    let node = null;

    if (i < this.length / 2) {

      console.log("IF");

      let count = 0;

      node = this.head;

      while (count < i) {

        node = node.next;

        count++;

      }

    } else {

      console.log("ELSE");

      let count = this.length - 1;

      node = this.tail;

      while (count > i) {

        node = node.prev;

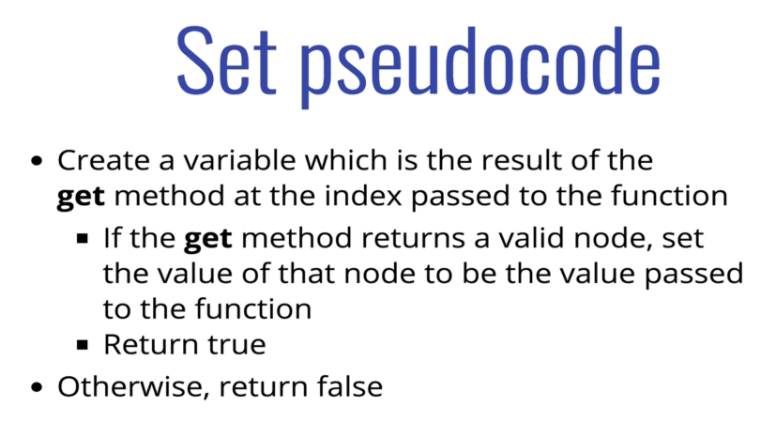
        count--;

      }

    }

    return node;

  }



  set(i,value){

    let foundNode = this.get(i)

    if(foundNode){

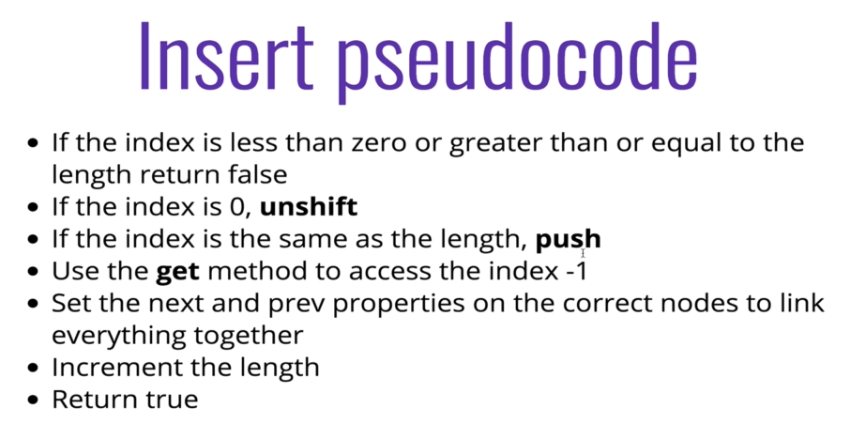
      foundNode.val = value;

      return true

    }

    return false

  }



  insert(i, value) {

    if (i < 0 || i > this.length) {

      return false;

    }

    if (i === 0) return !!this.unshift(value);

    // if (i === 0) {

    //   this.unshift(value);

    //   return true;

    // }

    if (i === this.length) {

      this.push(value);

      return true;

    }

    let prevNode = this.get(i - 1);

    let newNode = new Node(value);

    let nextNode = prevNode.next;

    prevNode.next = newNode;

    newNode.prev = prevNode;

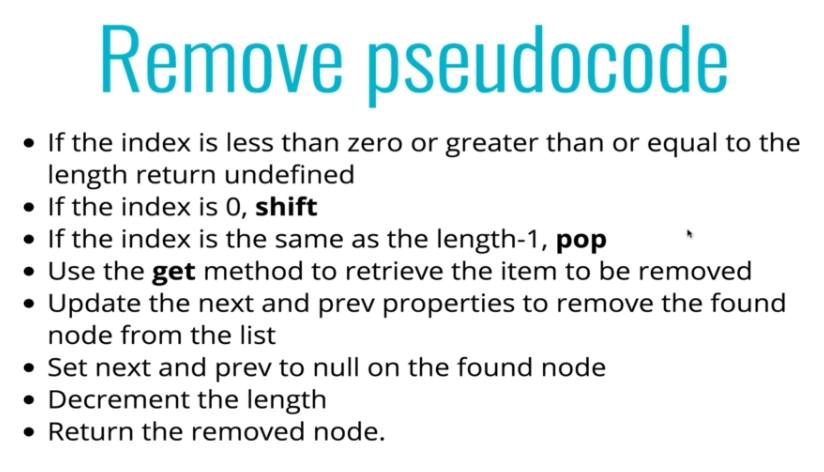
    newNode.next = nextNode;

    nextNode.prev = newNode;

    this.length++;

    return true;

  }



  remove(i) {

    if (i < 0 || i >= this.length) {

      return undefined;

    }

    if (i === 0) {

      return this.shift();

    }

    if (i === this.length - 1) {

      return this.pop();

    }

    let removedNode = this.get(i);

    let prevNode = removedNode.prev;

    let nextNode = removedNode.next;

    removedNode.next = null;

    removedNode.prev = null;

    prevNode.next = nextNode;

    nextNode.prev = prevNode;

    this.length--;

    return removedNode;

  }

Time Complexity

Comparing Singly And Doubly Linked List