Write a function to intersect two singly linked lists based on their value property. Function must be O(n). Arrays can be assumed to be sorted

Examples:

{val: "foo", next: {val: 'bar', next: {val: 0, next: null}}}, {val: 'bar', next: {val: '0', next: null}}

→ {val: bar, next: {val: 0, next: null}}

Pseudocode:

Function = (list A, list B) => {}

Step through List A: extract all values (assumed to be unique) ✓

Step through List B: every time a b.value matches an A.value, add a new node (B.value) to the output LL.

Return return LL.

Set up a recursive ~~loop~~ function

```
while a.next !== null ||
  if (a.val === b.val)
    ~~return~~ out.add

var AUniques = [];
while (a.next !== null) {
  AUniques += a.val;
  a = a.next;
}
```

```
while !(a.next === null || b.next === null)

llOut =
var ~~[illegible]~~ = null;
~~var Bsub = b.~~
~~var LLNew = [illegible];~~
 while (b.next !== null) {
   if (AUniques [b.val]) {
     if (LLOut === null) {
       LLOut = new Node (b.val) → pointer = LLOut
     } else {
       LLOut.next = new Node(b.val)
       pointer = LLOut.next
     }
   } else {
   b = b.next
 }
return LLOut;
```

1 ⇒ New Node

2+ Add, join

Intersect two linked list objects
based on value

Matthew LeBlanc
12/12/17

```
listOne = [4| ][7|][9|][12|]
listTwo = [2|][4|][8|][12|]
   => [4|][12|]

            make     (list)
Function addToIntersect(node) {
   intersect = {};
intersect[node.value] = true
                               while(node.next) {
                                  node = node.next
                                  intersect[node.value] = true;
   If(node.next)               }
     return a2i(node.next)     return intersect;
   else return intersect
                               if intersect[node2] => newinters[node2.value]
}                              while(node.next) {
checkintersect (intersect, list2)  node = node.next;
  newIntersect = {}                if intersect[n2] =>    "     "
  IF (intersect[node2.value])  }
     newintersect[node2.value] return newintersect
```

```
function intersectTwo(listOne, listTwo) {
   let FirstIntersect = (listOne) => {
         let int = {};
         int[listOne.value]
         while(listOne.next) {
            listOne = listOne.next
            int[listOne.value]
         }
         return int;

   }
   let finalList={}

      if(firstIntersect[listTwo.value])
            finalList.value =  listTwo.value;
            finalList = finalList.next;
      while(listTwo.next) {
         listTwo = listTwo.next;
         if(firstIntersect[listTwo.value]){
            finalList.value = listTwo.value;
            finalList = finalList.next
            }
      }
      return finalList;

   }
```

ugh B
value = B.value
tersection. append (A.value)

A
A = {A.value = true}
B
; A[B value])
swer . append (B.value)
x = x.next

est

obj A = { 1 : true,
2 : true,
3 : true,
}

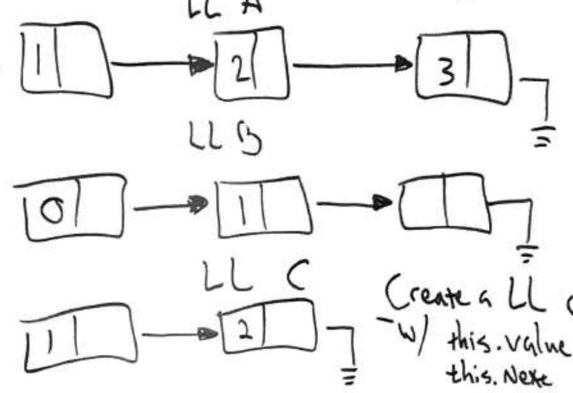
O(length of A + length of B)

O(2n)
O(n)

Kerry Nordstrom 12/12/17

Restate: Traverse two singly linked lists to find common values and return another singly linked list

Example/Pseudo:



```
ListNode {
  next;
  value;
}
```

- Create a LL constructor
- w/ this.value
  this.Next
- Instantiate two new LL with initial values
- Loop through LLA and set all found values to true
- Loop through LLB and if these values are true, add them to LLC

Code:

```
let C         = { };

f intersectLinks => (A, B) {
   let A      = {1, 2, 3}
   let B      = {0, 1, 2}
let current = A ;
let otherCurrent = B
   while (! current.next)
      if (current.value === otherCurrent.value)

   C.value = current.value
   C.next = new ListNode ();

 current = current.next;
```

cA
1 2 3
0 1 5
cB

Seth Donohue

① Write a function that returns the intersection of 2 SLL

② Examples: L1 = 1,2,8,4 → returns 2,8
L2 = 2,8

- no empty LL
- return SLL

③ Psudo:

```
~~f(L1, L2) => {~~
~~- traverse L1 to find L2 values?~~
~~- have counter track node number~~
~~- while (L1.value) {~~
~~x Counter ++;~~
~~  if(L1.value===L2.value) return L1.value;~~
~~    L1.value=L1.next.value;~~
~~}~~
```

③ Psudo:

```
f(L1,L2) => {
  let valuesFound = { }
  let SLL (value) => {
     this.value = value
     this.next = null;
  }
  while(L1.value){
     if(L1value=== L2.value)
        valuesFound [L1.value] = true;   L1.value = L1.next.value;
  while (L2.value){
     if(L2.value===L1.value)
        if(valuesFound [L2.value === true)
        result.append (L2.value);
        L2.value=L2.next.value;
```

④ Const intersection = function (listOne, listTwo) => {
let valuesFound = { };
Class LinkedList (value) => {
this.value = value;
this.next = null;

Shannon

- traverse l1 ✓
- add all node values as object properties ✓
- traverse l2 & see which node values are props. in object1 ✓
- create new linked list (or array) ✓ w/ properties from obj1

★ assuming both linked lists have at least 1 node; if not get an empty array

_Vinicio

```
let l=null;

if(l)
   l.append
else
   l=newLinkd(l.)
```

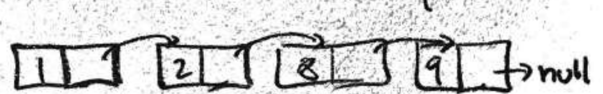
```
let intersection = (l1,l2) => {                      Goal: O(n)
   let obj1 = {};                                          (time)
   let matches = [];
   while (l1 !== null) {                             O(l1)
      if (!obj1[l1.value]) {
      obj1[l1.value] = true;
   };
   };
   while(l2 !== null) {                              O(l2)
      if (obj1[l2.value]) {
      matches.push (obj[l2.value]);                 O(1)
      };
   };

   return matches;                                   O(1)
                                                 ___________
};                                                O(l1+l2)
                                                     ↓
                                                    O(n)
```

Pedra

Intersect 2 LL
by value prop.

$\{1, 2, 8, 9\}$

$\{0, 2, 8\} \Longrightarrow$ Return $\{$value: 2, next: $\{$value: 8, next: null$\}\}$



```
let intor = ( A , B ) => {
    let obj = {}
    while( A.next) {
```

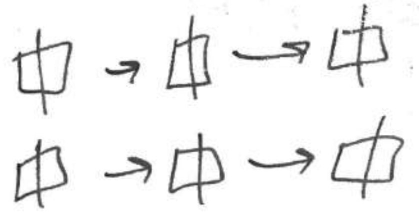
```
loop thru 1st SLL
  and set value to true
      object prop.
loop thru 2nd SLL
    compare if
      object prop value w/ sec. SLL is TRUE
               and matching
    RETURN SLL
```

F
r
e
d
r
i
c

DAVID
LINDAHL
12-12-17

PROBLEM DOMAIN

① function to intersect
2 singly linked lists
based on value prop



while (node)

Setup object {}
loop through LL values
+ traverse list → while !node.next = null
add to object
compare object
to 2L

L1 - 1, 2, 4, 8
L2 - 2, 8
return 2+8
new LinkedList
w/ all intersection pts

```
{ '1 : true
   2 : true
   4 : true
   8 : true
   2 : true
   8 : true
}
```

PSUEDO CODE

```
function takes L1 + L2,
        eObj = {};
    while L1.node
            eObj [node.val ] = true

    while L2.node
            eObj [node.val ] = true

    Magically grab all values
      (and over 1
```

DA REAL CODE

```
function (L1, L2) => {
        eObj = {};
        while (L1.node) {
            eObj [node.value] = true;

        };
        while (L2.node) {
            eObj [node.value] = true;

        };
```

BigO (L1+L2]
BigO (n)

ext: null}}

ll

F
r
e
d
r
i
c

Write a function that intersects two singly linked lists based on VALUE property

L1: 1, 2, 3   intesect value would be (3)
L2: 3, 5, 7

```
const intersect (l1, l2) {
      var val = { }
      var results = ( );
   for(var i=0; i < l2.next; i ++) {

    val (l2:(i)) = true;

    }
   for (var j=0; j < l1.next; j ++) {
      if (val(l1:(j))
        results.next (L1(j));

   }
    return results;

   };
```

f

(

2 (3)

Approach: Brute Force; Consider Map

Cameron

```
const findLLIntersection = (list1, list2) => {
  if (!list1 instanceof LinkedList ||!list2 instanceof LinkedList) {
      return null;
  }
  if (!list1 ||!list2) {
      return null;
  }
  const intersection = new LinkedList();
  const Dictionary = new Set();
  let current1 = list1;
  while(current1.next) {
      Dictionary.add(current.value);
      current1 = current1.next;
  }
  let current2 = list2;
  while (current2.next) {
     if (Dictionary.has(current2 value)) {
          intersection.append(current2.value);
     }
     current2 = current2.next;
  }
  return intersection;
};
```

Big O:

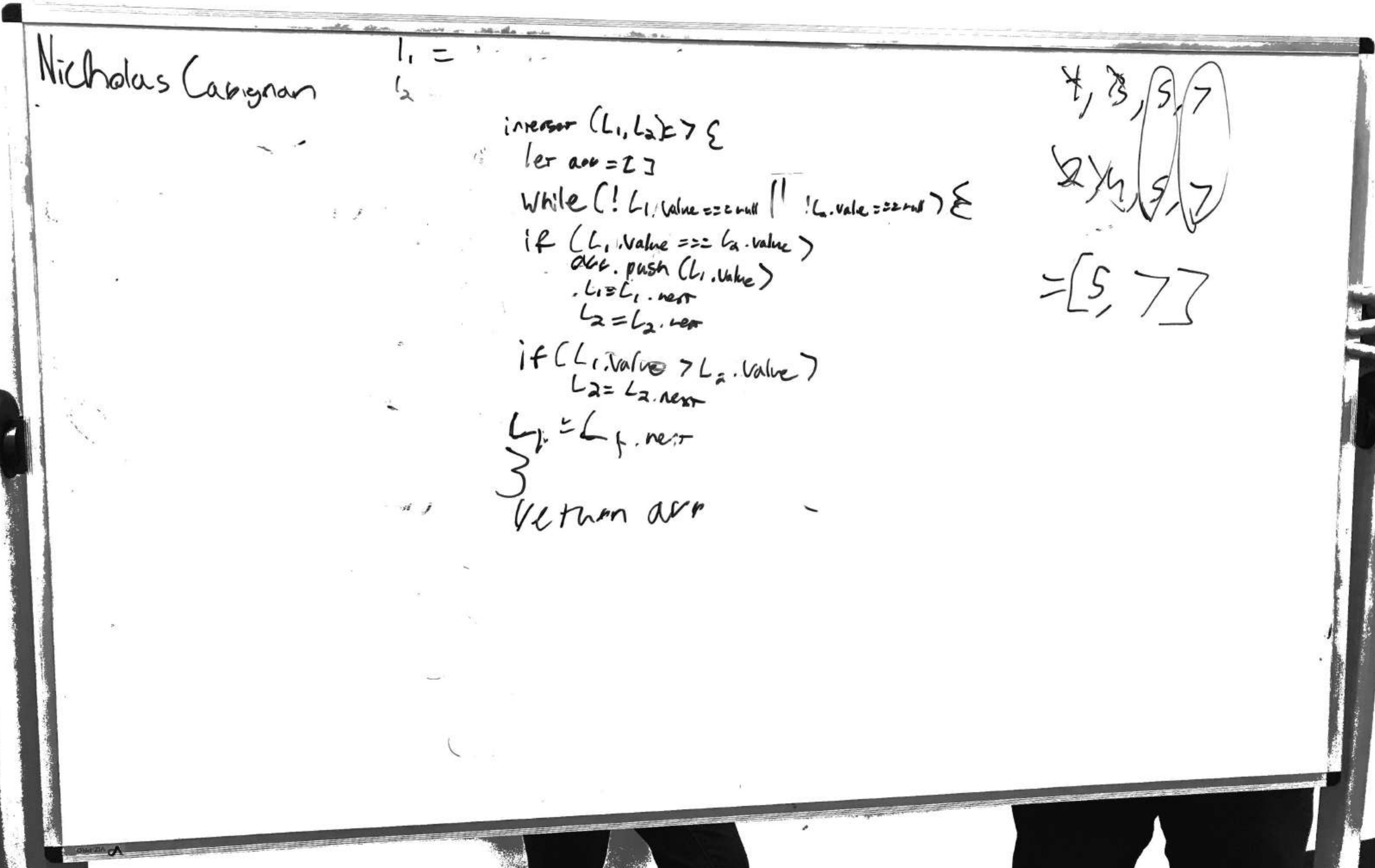
Space: O(n) where n is equal to # of nodes in list1

Time: O(n+m) where n is equal to # of nodes in list1 and m is equal to # of nodes in list2

Anthony

```
linkedlist1 = { Value: 1
                next: { Value: 2
                        next: { Value: 3
linkedlist2 = { Value: 2        next: null } } }
                next: { value: 4
                        next: null } }

x find intersecting values

        let findIntersect = (linkedlistOne, linkedlisTwo) => {
              match = {};
loop <-       While (linkedlist1.value !== linkedlist2.Value){
                  linkedlist2.value = linkedlist2.next.Value
   need          if (linkedlist1.value === linkedlist2.Value){
                    match.Push(linkedlist2.Value);
loop<-           } else {

                  I need to check if ll2 has no next.
                  I need another loop to cycle
                  through ll1
                  If ll1 does not have a next
                  return match.
```

$$O(1)$$
$$\uparrow$$
match[1] = true;



Nicholas Carignan

l_1 =
l_2

```
intersect (L1,L2)=> {
 let arr = [ ]
 While (! L1.value === null || !L2.value ===null ) {
 if (L1.value === L2.value )
     arr.push (L1.value)
     L1=L1.next
     L2=L2.next
 if (L1.value > L2.value )
     L2= L2.next
 L1 = L1.next
}
 return arr
```

~~1~~, ~~3~~, 5, 7

~~2~~, 4, 5, 7

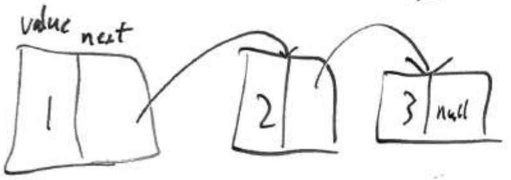
= [5, 7]

Jeff Kusowski

- input → 2 linked lists (A, B)

output → linked list of intersection

assume constructor called LinkedList with append Function



```
let intersection = (A, B) => {
    let answer = new LinkedList
    let objA = {}
    while ( A ){
        objA[A.value]=true


        A=A.next
    }
    while ( B ){
        if (objA[B.value])
            answer.append(B.value)

        B=B.next
    }
    return answer
}
```

```
~~loop through A~~
    ~~loop through B~~
        ~~if A.value = B.value~~
            ~~intersection.append(A.value)~~
```

```
loop through A
    let objA = {A.value = true}
loop through B
    if (objA[B.value])
        answer.append(B.value)

x = x.nex
```

Test

```
objA = {1 : true,
        2 : true,
        3 : true,
       }
```

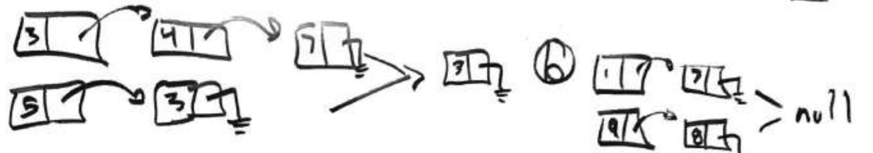
$O($length of A + length of B$)$

$O(2n)$

$O(n)$

① Problem: Write a function intersect that takes two singly linked lists as arguments (Assume both are provided and valid) and returns a new linkedlist with only the values in both.
Assume we have a LL constructor w/ append method.

② example:



③ Psudo:

```
a) establish inA object and newList object w/ null value
1) iterate through list A
    ↳ establish currentNode var set to list A
    ↳ while currentNode
        ↳ set prop on inA w/ key as currentNode.value & value as true
        ↳ set current Node as currentNode.next
2) iterate through list B
    ↳ set current Node as list B
    ↳ while currentNode
        ↳ if inA[currentNode.value]
            ↳ if !newList.value  (none added yet)
                ↳ newList.value = currentNode.value
            ↳ else (already added a match)
                ↳ newList.append (new LL (currentNode.value)
        ↳ currentNode = current Node.next
3) return new list or null if no value
```

Big O time and space of $O(A+B)$ where A is the length of list A & B is the length of list B, which boils down to just $O(n)$

④ Code:

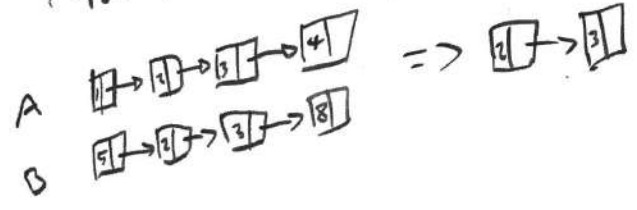
```
const intersect = (A, B) => {
  let inA = {}, newList = new LinkedList(null);
  let currentNode = A;
  while (currentNode) {
    inA[currentNode.value] = true;
    CurrentNode = currentNode.next;
  }
  currentNode = B;
  while (currentNode) {
    if (inA[currentNode.value]) {
      if (newlist.Value)
        newList.append (new LinkedList (currentNode.value);
      else
        newList.value = currentNode.value;
    CurrentNode = currentNode.next;
    }
  }
  return newList.value ? newList : null;
};
```

Robert Reed
12/12/17
401d19

input = 2
output → lin
assum
value next
let intersecti
let
let
whil
whil
}
re
}

Jacob Evans

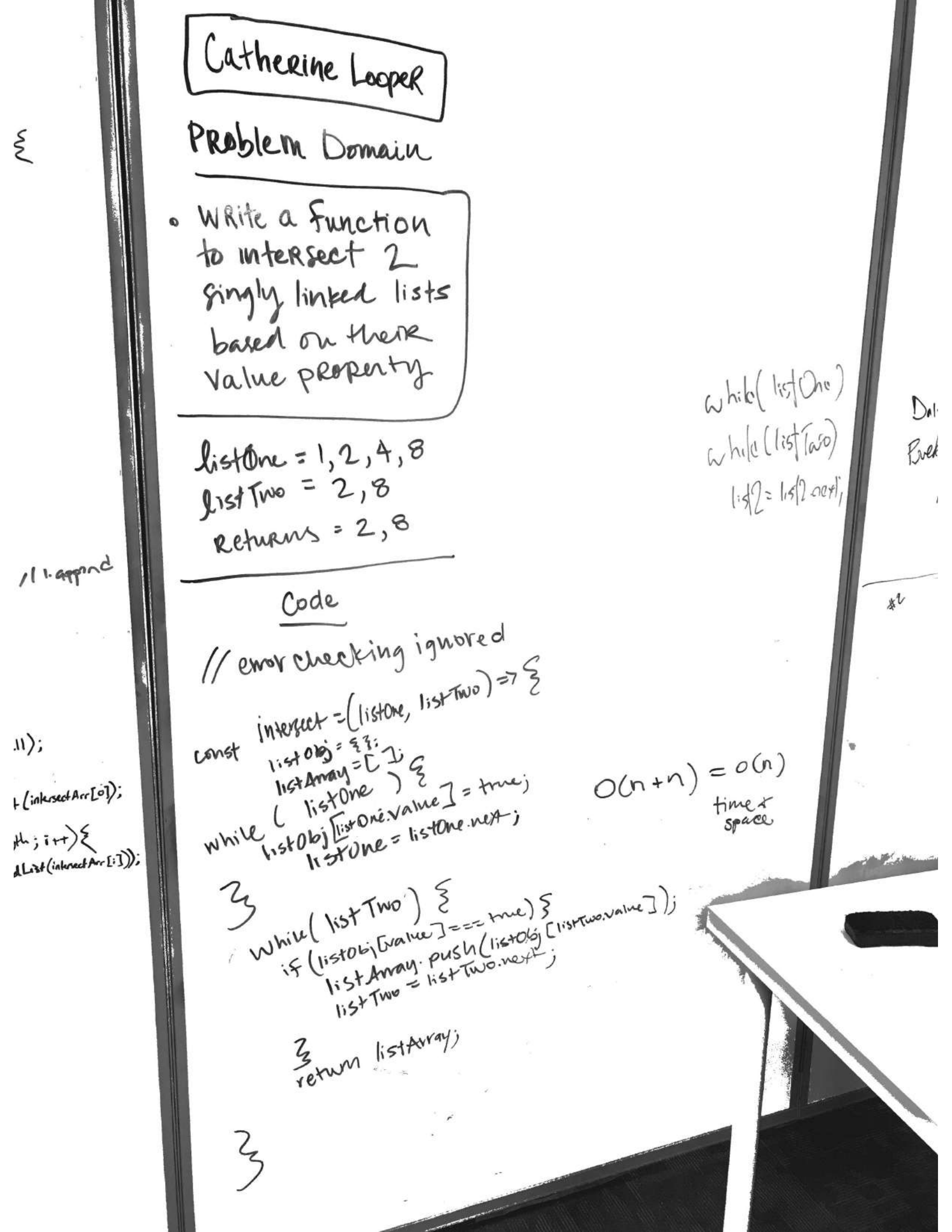
two linked list <u>intersect</u> based on values



check each A list value
to each B list ~~value~~
find matching values
create new link list with
matching values

```
while current Avalue !== ALL Bvalues -> reassign current nodeA
    if values match
node to new list => "append"
```

```
Const intersect = (nodeA, nodeB) => {
```



Catherine Looper

Problem Domain

- Write a function to intersect 2 singly linked lists based on their value property

listOne = 1, 2, 4, 8
listTwo = 2, 8
Returns = 2, 8

Code

```
// error checking ignored
const intersect = (listOne, listTwo) => {
  listObj = {};
  listArray = [ ];
  while ( listOne ) {
    listObj[listOne.value] = true;
    listOne = listOne.next;
  }
  while( listTwo ) {
    if (listObj[value] === true) {
      listArray.push(listObj[listTwo.value]);
    listTwo = listTwo.next;
  }
  return listArray;
}
```

$O(n+n) = O(n)$
time + space

```
while( listOne )
while( listTwo )
  list2 = list2.next;
```

```
{
//l.append
ll);
t(intersectArr[0]);
th; i++){
dList(intersectArr[i]));
```

Dal
Break

#2

Andrew

write a function that takes 2 linked lists and returns a new linked list comprised of the intersection of those 2 linked lists. Assume that the linked list class and methods already exist.

```
declare new function
  perform input validation
    -> is input linked list?
    -> are values numbers

declare new object, newObj
declare new array, intersection

traverse linked list function
        if this.next,
            newObj[this.value] = true
              return traverse(this.next)
        else
            newObj[this.value] = true

   intersect linked list function
          traverse linked list,
        if newObj[this.value]
            then push to array
        declare linked list to return
          create new linked list
        out of the intersect array,
          intersection.forEach( e => {
            linked list.append(e)
          }
```

```
const linkedListIntersection = (list1, list2) => {
      // input validation here
                        = null;
    const hashObj = {};
    const intersectArr = [];
    (const traverse = list => {
O(n)    hashObj[list.value] = true;
        if (list.next){
            return traverse(list.next)
        }

    })(list1);
    ( const intersect = list => {
        if (hashObj[list.value]){
O(n)        intersectArr.push(list.value); // l.append
        }
        if (list.next) {
            return intersect(list.next);
        }
    })(list2);
    if (intersectArr.length < 1){
        return new linkedList(null);
    }
    const returnedList = new linkedList(intersectArr[0]);
    for (let i = 1; i < intersectArr.length; i++){
O(n)    returnedList.append(new linkedList(intersectArr[i]));
    };
    return returnedList;
};
```