Seth Donohue

① Problem: Write a function that returns an array with the Intersections of 2 input arrays
- may have duplicates
- only number
- may not have intersections
- input arrays may be null/empty

② Examples: input [1,2,3,3] [1,4,5] → Output [1]

[0,13,1,5,6] [0,1,13,5] → [0,13,1,5]

[0,0,13,13,7] [0,0,13,7,2] → [0,0,13]

③ Pseudo Code: [ ], [ ]

[0,0,1,5,6] [0,0,2,5]

f(arr1, arr2) ⇒ {

arr1.filter(f(elem, index, array) {

O(n+n)

O(arr1 length + arr2 length)

O(2n) → O(n)

tests:	return	if elem === arr2[index];
1.)	0 === 0	✓ → [0]
2.)	0 === 0	✓ → [0,0]
3.)	1 === 2	X → [0,0]
4.)	5 === 5	✓ → [0,0,5]
5.)	6 === null	X → [0,0,5]
6.)	returns [ ]	

④ Code:

```
const intersection = (arr1, arr2) ⇒ {
  return arr1.filter((element, index) {
    return element === arr2[index];
  });
```

5 null = false

tests	~~input~~ arr1	arr2	result
	[1,1,5,6,7]	[1,1]	[1,1] ✓
	[ ]	[1,2,5]	[ ] ✓
	[5,5,5]	[5]	[5] ✓
	[5]	[5,5]	[5] ✓
	[2,1]	[1,2]	[ ]

need to sort both arrays first

WRITE A FUNCTION THAT INTERSECTS TWO ARRAY

EXAMPLE A[1,2,3] , B[0,3,5]

FIND THE SAME VALUE IN ARRAY & PUSH THEM TO NEW ARRAY IN THIS EXAMPLE THE NEW ARRAY VALUE WILL BE

C[3]

FREDDRIC

```
function intersect = (a, b) {
  VAR aI = 0;
  VAR bI = 0;

Var result = [ ];
while (aI < aI.length && bI < bI.length);
} if aI[ ] < bI[ ], aI ++
  else if aI[ ] > bI[ ]
```

Nicholas Langran

```
function(arr1, arr2) {
  let result = []
  for (X of arr1)
      if arr2.includes(x){
          if(! result.includes(x)
              result.push(x)
      }
return result
```

function (n, m) =

O(n.length * n.length)

or O(n)

Where n is the length of the longer array

f([1,2,3], [2,2,3,4])

returns [2, 3]

f([1,2,3], [4,5,6])

returns [ ]

Kerry Nordstrom 12/5/17

Restate: Create function that takes in two arrays as parameters then return array with unique values.

Pseudo:

Create function w/ two arrays as parameters
Declare variable that is empty array
Push contents of both arrays to new array
Filter third array to remove duplicates
Return third array
Return empty array if both are empty

```
function arrayIntersection((arr1, arr2) => {
    let resultsArray = [];
    if (array1.length=0 && array2.length=0) {
     return [];
    };
    for(let i=0; i<arr1.length; i++){
        if (arr1[i] !== arr2[i]){
          resultsArray.push(arr1[i]);
        };
    };
     for (let i=0; i<arr2.length; i++){
        if (arr2[i] !== arr1[i]){
          resultsArray.push(array2[i]);
        };
     };
};
```

O(n²)

Jacob Evans

take 2 arrays & create a new array that has values that both arrays share

Big O = O(N)

```
let arr1=[1,2,3,4,5];
let arr2=[3,4,9,22];

Const intersectArr =(arr1, arr2) =>{

return arr1.map(ele => (ele === arr2[ele]) ? ele : null);

}
```

intersectArr = [3,4]

Create a function that takes two arrays, and returns the intersection of their values in array form.

Phelan

```
(a, b) => {
  let result = [];
  for (value of a) {
    match = b.indexOf(value)
    if (match === null)
      {continue}
    result.push(value)
    b.removeAt(match)
  }
  return result;
}
```

indexOf
- Works with arrays?
- Returns null?

removeAt
- Does it even exist?

If IndexOf is NOT an O(n) function, my solution is O(n).
Otherwise, my solution is $O(n)^2$.

[1, 'foo', null, 0] ∩ [0, undefined, 'test', 'poo', 5, 4, 'bar', null]

[null, 0]

Jeff Kusowski

Inputs = [ ] [ ]
only integers

Output = [ ]

```
let ans=[];
for i in arr1
  if arr2.includes(arr1[i])
```

```
function (arr1, arr2) => {
    let ans = [];
    for (i=0; i < arr1.length; i++) {
        if arr2.includes(arr1[i]){
            ans.push(arr1[i]);
            let remove = arr2.indexOf(arr1[i]);
            arr2.splice(remove);
        }
    }
    return ans;
```

O(n) — O(n)

O
n_1
n_2
|
|
|
O(n)

[1,2,3] [2,1] ↓

test	arr 1	arr 2	ans
	[1,2,3]	[3,4,5]	[3] ✓
	[4,4,4]	[4,4,4]	[4,4,4] ✓
	[4,4,4]	[4]	~~[4,4,4]~~ X
	[4]		[4] ✓
		[4,4,4]	[4] ✓
	[4,4,4]	[4,4]	[4,4] ✓

① Problem
Given two arrays, write a function that returns an array containing their intersection.

② Example:
Assume each array has simple elements (Numbers, strings, boolean).

ⓐ [1, 3, 4] [2, 3] >> [3] ⓑ [1, 2] [3, 4] >> [] ⓒ [1, 2, 2] [1, 2] >> [1, 2]

Rob Reed 12/6/17
40ld19

③ Pseudo Code

a) if either array is empty return []
1) create a new array to store intersection
2) iterate through the shorter array (ar1)
   1) get indexOf element ar1[i] in ar2
      a) if index > -1
         i) pop off ar2[index]
         ii) push pop ar1[i] into intersection
3) return intersection

④ Code

```
const intersect = (ar1, ar2) => {
  if (ar1.length === 0 || ar2.length === 0)
    return [];
  let intersected = [];
  let shorter = ar1, longer = ar2;
  if (shorter.length > longer.length)
    [shorter, longer] = [longer, shorter];
  while (shorter.length > 0) {
    let value = shorter.shift();
    let index = longer.indexOf(value);
    if (index.value > -1) {
      longer = longer.splice(0, index).concat(longer.splice(index+1));
      intersected.push(value);
    }
  }
  return intersected;
};
```

⑤ Test let ar1 = [1, 2, 2, 3], ar2 = [2, 3, 4];

intersect(ar1, ar2)
↳ ar1.length === 4 > 0, ar2.length === 3 > 0, do nothing
↳ intersected = []
↳ shorter = [1, 2, 2, 3], longer = [2, 3, 4]
↳ shorter.length === 4 > longer.length === 3
  ↳ swap, shorter = [2, 3, 4], longer = [1, 2, 2, 3]
↳ shorter.length === 3 > 0, enter loop
  ↳ value = 2, shorter = [3, 4]
    index = 1
    index > -1, so enter if
    ↳ longer = [1] + [2, 3] = [1, 2, 3]
      intersected = [2], loop
    ↳ value = 3, shorter = [4]
      index = 2, > -1 so enter if
      ↳ longer = [2] + [1, 2] = [

Big O of O(a·b) where a is the length of array 1 & b is the length of array 2.

Andrew

Write a function that takes two arrays and returns the intersection of the two arrays as a new array

define a function
define a newArray
filter each array for duplicates
loop through the first array and for each value, check if that value is included in the second array
if it is, push that value to the new array
return the new array

```
"use strict";
const ArrayIntersection = (array1, array2) => {
  let newArr = [];                                          O(1)
  if (array1.length > 0 && array2.length > 0) {
    let filteredArray1 = array1.filter((v,i,a) => {
        return i === a.indexOf(v);                          O(array1 length)
        });
    let filteredArray2 = array2.filter((v,i,a) => {
        return i === a.indexOf(v);                          O(array2 length)
        });
    newArr = filteredArray1.filter(value => {
        return filteredArray2.includes(value);
        });                                                 O(filteredArray1 length times filteredArray2 length)
    }
  return newArr;                                            O(1)
};
```

Get 2 arrays
return new array
w/ all shared pts
-only #s given

[][] → []✓
[][1] → []✓
[3,2,3][3,1,0] → [3]
[1,2,3][1,2,3] → [1,2,3]
[3,0,3,0][0,3,1] → [3,0]

*if either [] empty
-return []

*for every element
in arr1: see if arr2
also has that value
-yes? Put it in the
new array
-no? Go to the next
element in arr1
& check that

```
let matchingNums = (arr1,arr2) => {
  if (!arr1.length || !arr2.length) {
      return [];
  }
  let matches = [];
  for (let i=0; i < arr1.length; i++) {
    if (arr2.indexOf[arr1[i]] >-1){
        matches.push(arr1[i]);
    }
  }
  return matches;
}
```

Shannon

storage: O(n)
time: O(n)

arr1 arr2
[3,3,1], [3,4,5]

1) is index of '3' in arr2 >-1 ✓
matches = [3] * not a returned value
2) is index of '2' in arr2 >-1 ✗
matches = [3] * not returned yet
3) is index of '1' in arr2 >-1 ✗
matches = [3]

return [3]

Cameron

```
const findCommonElements = (arr1, arr2) => {
  if (!Array.isArray(arr1) || !Array.isArray(arr2)) {
    return null;
  }
  if (arr1.length === 0 || arr2.length === 0) {
    return [];
  }
  let returnedArr = [];
  for (let i=0; i < arr1.length; i++) {
    if (arr2.indexOf(arr1[i]) > -1) {
      returnedArray.push(arr1[i]);
    }
  }
return returnedArray;
}
```

Test:

[], [1,2] ✓

[1,2], [] ✓

'not an array', [1,2] ✓

[1,2], [3,4]

[1,2,3], [2,4,6]

indexOf as constant time

Big O:

Space: O(n)

Time: O(n)

Where n is the # of elements in the largest array

indexOf as linear time

Big O:

Space: O(n)

Time: O(n×m)

Where n is the # of elements in arr1 and m is the # of elements in arr2

Andrew

Write a function that takes two arrays and returns the intersection of the two arrays as a new array

define a function
define a newArray
filter each array for duplicates
loop through the first array and for each value, check if that value is included in the second array
if it is, push that value to the new array
return the new array

```
"use strict";
const ArrayIntersection = (array1, array2) => {
  let newArr = [];    O(1)
  if (array1.length > 0 && array2.length > 0) {
    let filteredArray1 = array1.filter((v,i,a) => {
      return i === a.indexOf(v);
    });                                          O(array1 length)
    let filteredArray2 = array2.filter((v,i,a) => {
      return i === a.indexOf(v);
    });                                          O(array2 length)
    newArr = filteredArray1.filter(value => {
      return filteredArray2.includes(value);
    });
                                  O(filteredArray1 length times filteredArray2 length)
  }
  return newArr;    O(1)
};
```

DAVID
LINDAHL

- FUNCTION THAT INTERSECTS 2 ARRAYS
- WILL RETURN A NEW ARRAY W VALUES FROM BOTH ARRAYS
- IF NONE → BLANK ARRAY
- NOTE BIG O

IDEAS - LOOPS, MAP

PSUEDO CODE-

loop over ArrayA (i)
  loop over ArrayB (x)
    if i = x,
    pop into new Array (x)
return new Array

if (ArrayB.length > 0 && ArrayB.length > 0)

else {
  return newArray [ ];

};

REAL CODE

```
arrayA = [ ];
arrayB = [ ];
newArray = [ ];
function intersectArrays () {
  if (arrayA.length > 0 && arrayB.length > 0) {
    for (i=0, i < arrayA.length, i++) {
      for (b=0, b < arrayB.length, b++) {
        if (i=b) {
          newArray.pop(b)
        }
      }
    return newArray;
  };
```

$O(n^2)$

test

```
arrayA = [1,2,3,4,5];
arrayB = [1,3,6,2];
newArray = [1,2,3];

arrayA = [2,1]
arrayB = [ ],
newArray = [ ];
```

$O(array)^2$

Catherine Looper

Problem Domain:
Write a function that joins two arrays at an intersecting value.

$O(n^2)$ ~ where n is the arrays

```
let newArray = [ ];                    O(1)
const array1 = [0,1,2,3,4,5]           O(1)
const array2 = [ 4,5,6,7 ]             O(1)

function joinArrays = (array1, array2) {
  for (let i = 0; i < array1.length; i++) {          O(n)
    for (let j = 0; i < array2.length; i++) {        O(n)
      if (array1[i] === array2[j]) {                 O(1)
        newArray.push(array1[i]);                    O(1)
      }
    }
  }
  return newArray;
}
```

Pedja Josifovic

function
intersects 2 Arrays

EDGE CASES
1. If one of the arrays is empty RETURN {}

```
let interArray = (arrayOne, arrayTwo) {

    if(arrayOne.length ===0 || arrayTwo.length ===0)   O(n)
        return {};

    for(let value in arrayOne){                        O(n²)
        newArray=[];
        if(arrayTwo.includes(value))
            newArray.push(value);

    };

}
interArray([0,1,2],[1,2,3]);
```

Loop thru every item in arrayOne and see if that item is included in arrayTwo.
If so → assign that item to newArray

Matthew LeBlanc

12/5/17

① [1, 2, 3, 4, 5, 6]

② [3, 6, 7, 2]

array①.filter(each, i, arrOne) => [2, 3, 6]

return array②.includes(each);

}

pseudo

function (arrOne, arrTwo) {

let newArr = arrOne.filter(each => {

return arrTwo.includes(each),

}

return newArr;

}

$O(n^2)$

Method 1

Method 2

function (arrOne, arrTwo) {

let newArr = [];

arrOne.forEach(each => {

if(arrTwo.includes(each))

newArr.push(each);

}

return newArr;

}

$O(n^2)$