**Algorithm Practice:**

**Q1: What’s the scope of this function?**

var text = 'outside';

function logIt(){

console.log(text);

var text = 'inside';

};

logIt();

A: It’s undefined

Why? Because when I declare with var, declrations are hoisted to the top of the scope but variable assignments are not

So this is how javascript looks at the code:

var text = 'outside';

function logIt(){

var text;

console.log(text);

text = 'inside';

};

logIt();

Solution? Declare var and assign it a value at the beginning of the scope.

**Q 2: /\* Popular Ice Cream Totals Quiz**

\*

\* Using the data array and .reduce():

\* - Return an object where each property is the name of an ice cream flavor

\* and each value is an integer that's the total count of that flavor

\* - Store the returned data in a new iceCreamTotals variable

\*

\* Notes:

\* - Do not delete the data variable

\* - Do not alter any of the data content

\*/

A:

const iceCreamTotals = data.reduce((acc, person) => {

person.favoriteIceCreams.forEach(iceCream => {

if(acc[iceCream]){

acc[iceCream]++

} else{

count = 1;

acc[iceCream] = count;

}

});

return acc

}, {});

**Q3: SAMPLE TEST:**

/\*

\* anagramPalindrome

\*

\* Write a function which accepts an input word and returns true or false if there exists

\* some anagram (permutation) of that input word that is a palindrome.

\*

\* "cat" => "tac", "tca"

\*/

A:

//HIGH LEVEL:

BASICALLY PUT EACH LETTER OF THE WORD INTO A HASH TABLE.

HASH TABLE: ALPHABETICAL ORDER: VALUE

SO HASH TABLE IS BASICALLY AN {}

SO YOU STORE HOW MANY TIMES EACH LETTER APPEARS AS A KEY

//SO SINCE A PALINDROME IS EITHER IN THIS FORM: ABA or AA. But remember this is a permutation so it’s not in that order

//SO STRATEGY IS, IF THERE’S ONLY ONE LETTER, AND EVERYTHING ELSE’S VALUE IS DIVIDED BY 2, THEN THAT’S A PALINDROM! (ABA case)

//ALSO, OF COURSE, IF EVERTHING’S VALUE IS DIVIDED BY 2 (AA case) then it’s a palindrom

// else, nope

var anagramPalindrome = function(word){

let frequencyOfLetters={};

for(var i =0; i<word.length; i++){

let count = 0;

//So that you can reset count to 0 every time

//and why you need to? Because you want to increment the value count, not the count itself by more than 1

if(frequencyOfLetters[word[i]]){

// don ‘t do frequencyOfLetters.word because it’ll look for the literal string ‘word’

frequencyOfLetters[word[i]] ++;

}

else{

count =1;

frequencyOfLetters[word[i]]= count

}

}

let check = true;

let frequencyCount = 0;

//outside so that value of count is tallied up each time and not set to 0

for(let letter in frequencyOfLetters){

//if values of all keys is divided by 2, return true

//+) if only one key is false, also return true

//if values of all but one key is divided by 2, return true

//otherwise, false

if ((frequencyOfLetters[letter] %2) ===1 ){

frequencyCount ++;

}

if ((frequencyOfLetters[letter] %2) ===0 || frequencyCount ===1) {

check;

}

else{

check= false;

}

}

return check;

};

console.log(anagramPalindrome("carrace")); // true

console.log(anagramPalindrome("cutoo")); // false

console.log(anagramPalindrome("an")); // false

console.log(anagramPalindrome("dddaaaayyyy")); // true

console.log(anagramPalindrome("anna")); //true

console.log(anagramPalindrome("bbd")); //true

**Question 4: MULTIPLE POINTERS**

// Example 1 : Sort a Bit Array

// Given a bit array, return it sorted in-place (a bit array

//is simply an array that contains only bits, either a 1 or a 0).

//in place means it doesn't require any extra space

// See if you can solve this in O(N) time and O(1) auxiliary space.

// Hint 1

// Since we want to sort it in-place we should be

//modifying the values change changing its

//position rather than creating a new array.

//Additionally because there are only two possible values,

//and we know that the 0’s will have to be on the left,

//and the ones have to be on the right.

//How can we identify elements to swap?

// Hint 2

// If we have two pointers:

//one that starts on the very left, and one on the very right,

//then we can iterate inward.

// We need to iterate the left pointer until it hits a 1

// Then decrement the right pointer to the left until

//it reaches a 0

// Once we find them we will do a swap.

A:

function bitArraySort(arr) {

let left = 0;

let right = arr.length - 1;

while(left < right) {

while(arr[left] === 0) { left++; }

while(arr[right] === 1) { right--; }

if(left < right) {

// swap the left and right values

// with []

[arr[left], arr[right]] = [arr[right], arr[left]];

}

}

return arr;

}

var lol= [0,1,0,0,1,1,0]

bitArraySort(lol)

// so assign left and right the 2 numbers which later can be the top and bottom of the array

// if left is smaller than right (meaning the position of the left pointer is smaller than the right pointer, then increase that position, and do the reverse with right)

// also since 0s will be on the left and 1s on the right anyways, so if the two pointers encounter them, they should just leave them where they are

//So basically this is saying that as long as arr[left] is 0 and arr[right] is 1, then swap the place using this syntax: [arr[left], arr[right]] = [arr[right], arr[left]]; it doesn’t care if there’s a 0 or 1 at each of those places

// can’t do if(arr[left] < arr[right]), cuz there might be a point where left = 1, right = 0, so browser is confused

// This is O(n) because you’re basically just going through the array once

// O(1) b/c it’s inline-sorted

**QUESTION 5: FREQUENCY COUNTING**

// Given an array of integers,

//and a target value determine

//if there are two integers that add to the sum.

// Input: [4,2,6,5,7,9,10], 13

// Output: true

//Basically my strategy is pushing (target- number) to a hash table

// give them value true

// so if we find the same number in the array, and in the hash

// that has value True. Then RETURN true. Exit out of the func

var twoSum = (arrayIn,target) => {

let hash ={};

for( const number of arrayIn){

let current = number;

if(hash[number]){

// Basically this line has 2 meanings: if hash has the number// if value of that number at hash === true, then it’s true

return true

}

hash[target-number] = true;

// put the additive of the number into the hash, with value true

}

return false

}

twoSum([4,2,6,5,7,9,10], 13)

//object will sort numbers: ex: hash={6: true, 8: true, 11: true}, not just by order putting in

//return true exist the loop

**BINARY SEARCH:**

Given a sorted array of unique integers, and a target value determine the index of a matching value within the array. If there is not match, return -1.

Input: [1,3,4,5,6,7,8,10,11,13,15,17,20,22], 17

Output: 11

function binarySearch(arr, target) {

let start = 0;

let end = arr.length - 1;

let mid;

while(start <= end) {

mid = Math.floor((start + end) / 2);

if(arr[mid] === target) { return mid; }

if(target < arr[mid]) {

end = mid - 1;

} else {

start = mid + 1;

}

}

return -1;

}

binarySearch([1,3,4,5,6,7,8,10,11,13,15,17,20,22], 10)

// note that the array is SORTED

// the mid is always determined by the start and the end, not just the whole array. If at blue I wrote (arr.length/2), that’s a forever loop because mid will always have that value

// So basically the logic is if target is smaller than the mid point of the array, ignore all the right (cuz the array is sorted). Same with the other side.

//Also: tip: when to put something in the loop? Check if that thing is affected by the loop (reset every time), then put in the loop

// When in doubt, use an example

// Don’t jump to the conclusion I need to do indexOf, can just give a var the index

**Modulo = remainder**

10 % 3 // =1

Strategy:

\_Read the case

\_Pseudo code: run thru a few cases and testing them

\_ And write