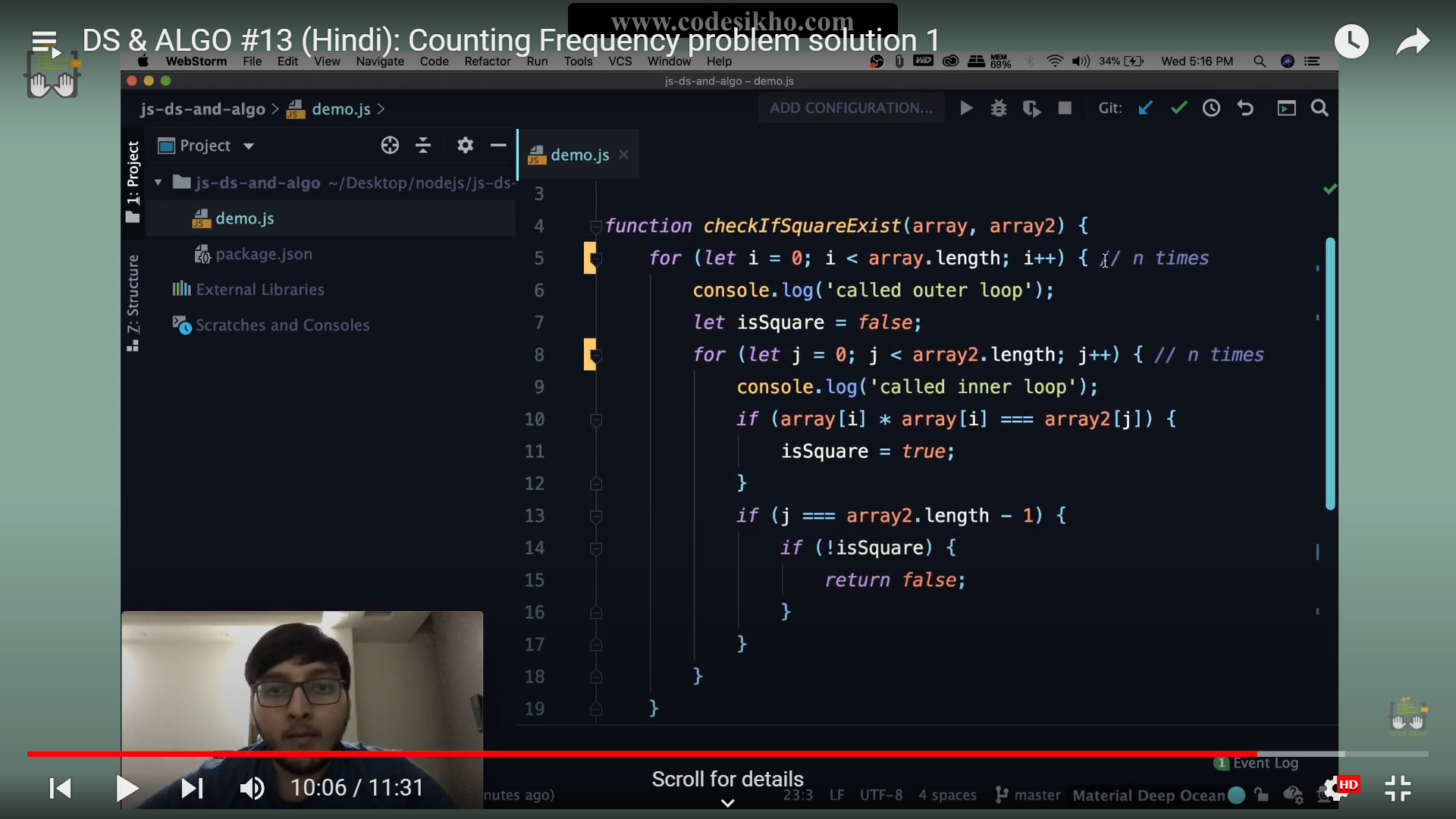
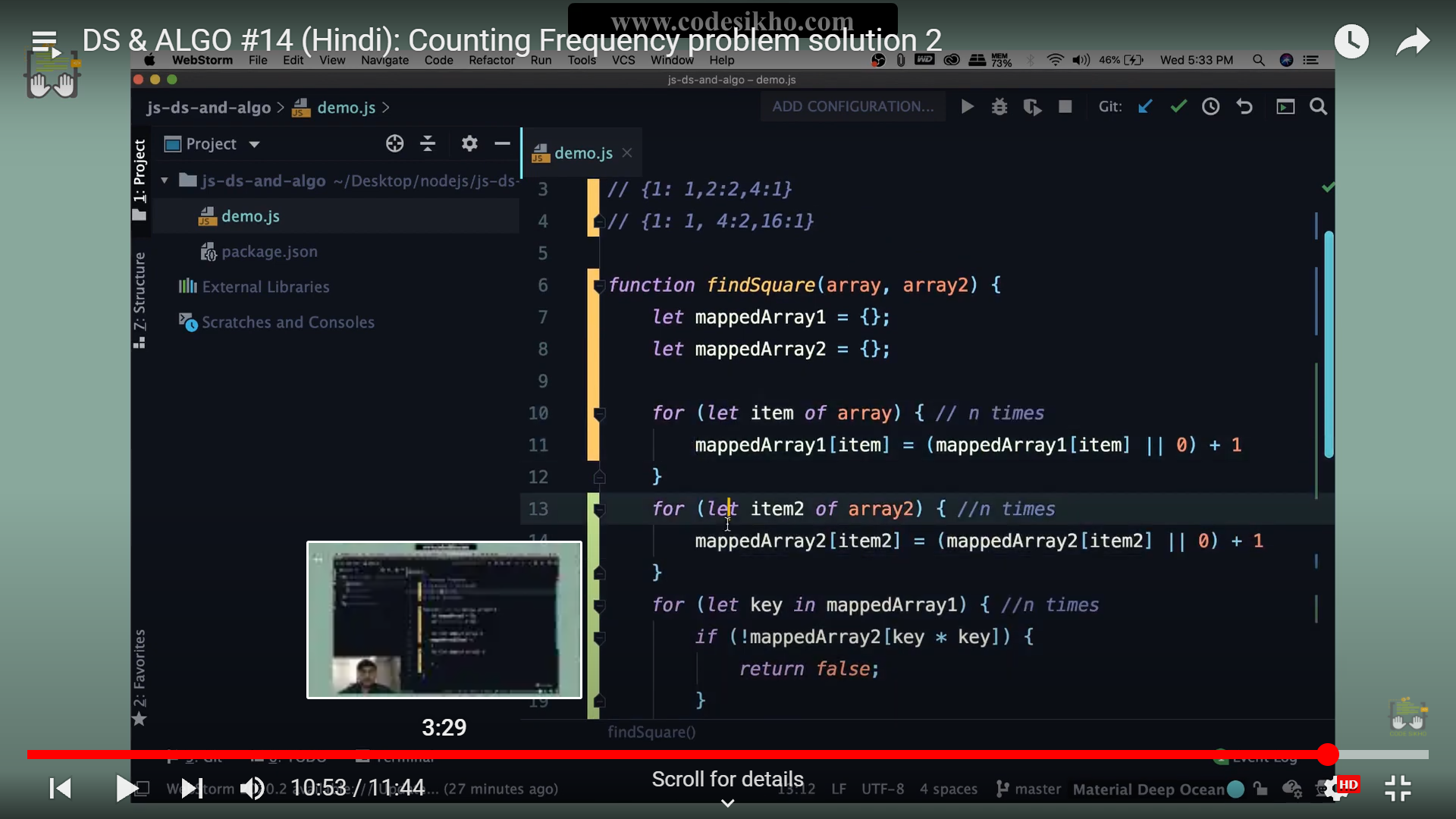
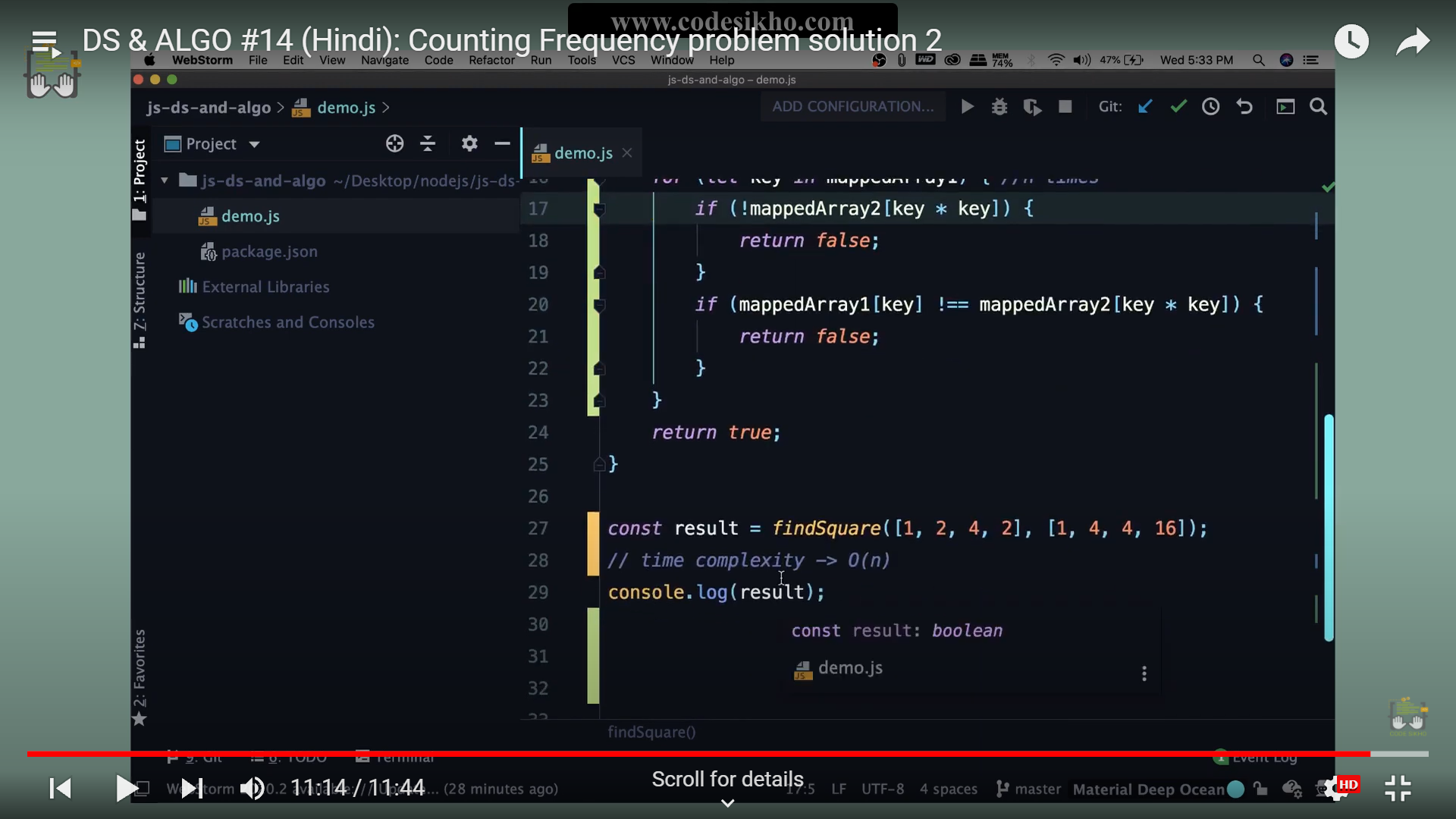
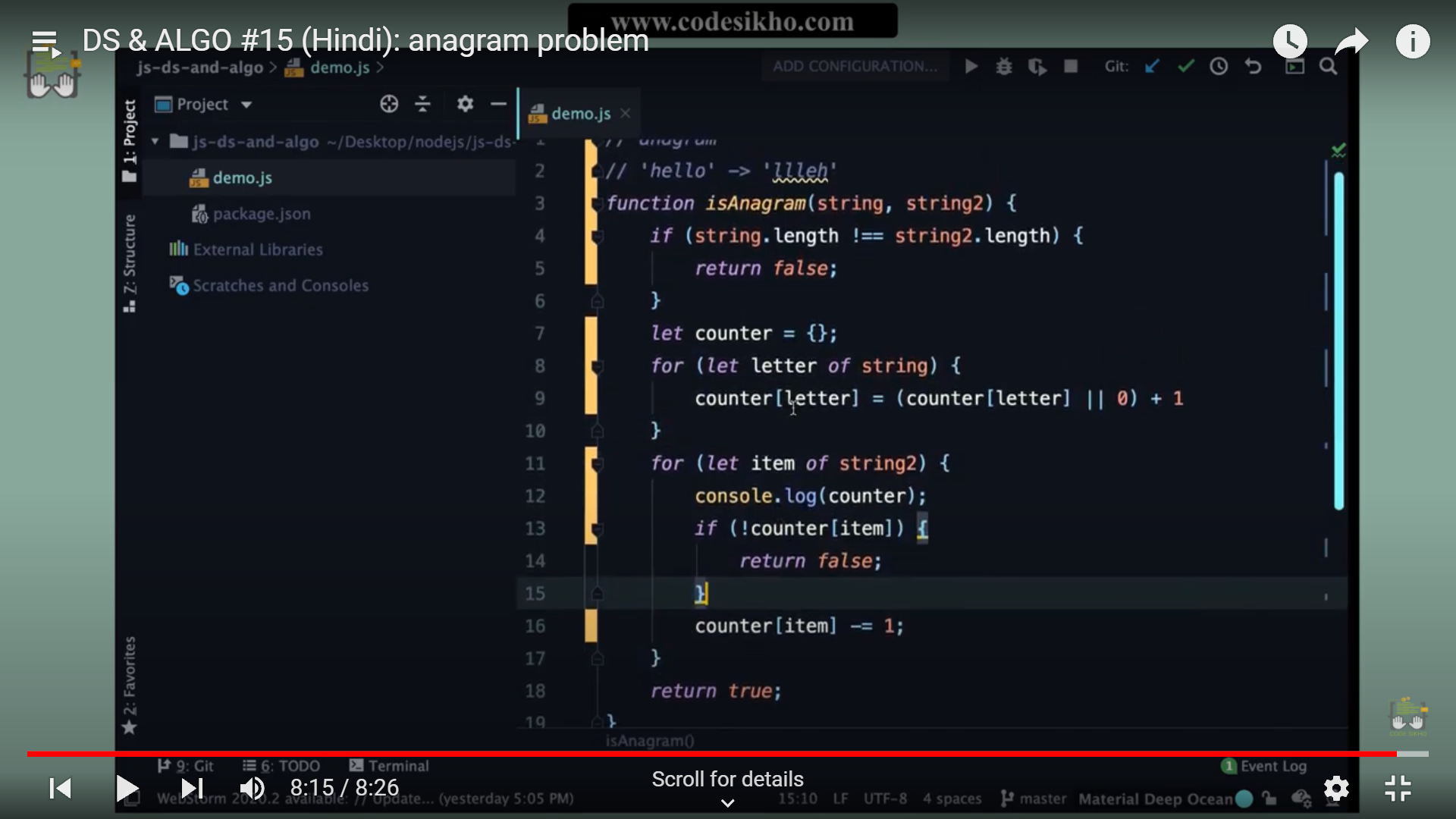
=== for unique array value in algorithm : <https://www.youtube.com/watch?v=dvPybpgk5Y4>

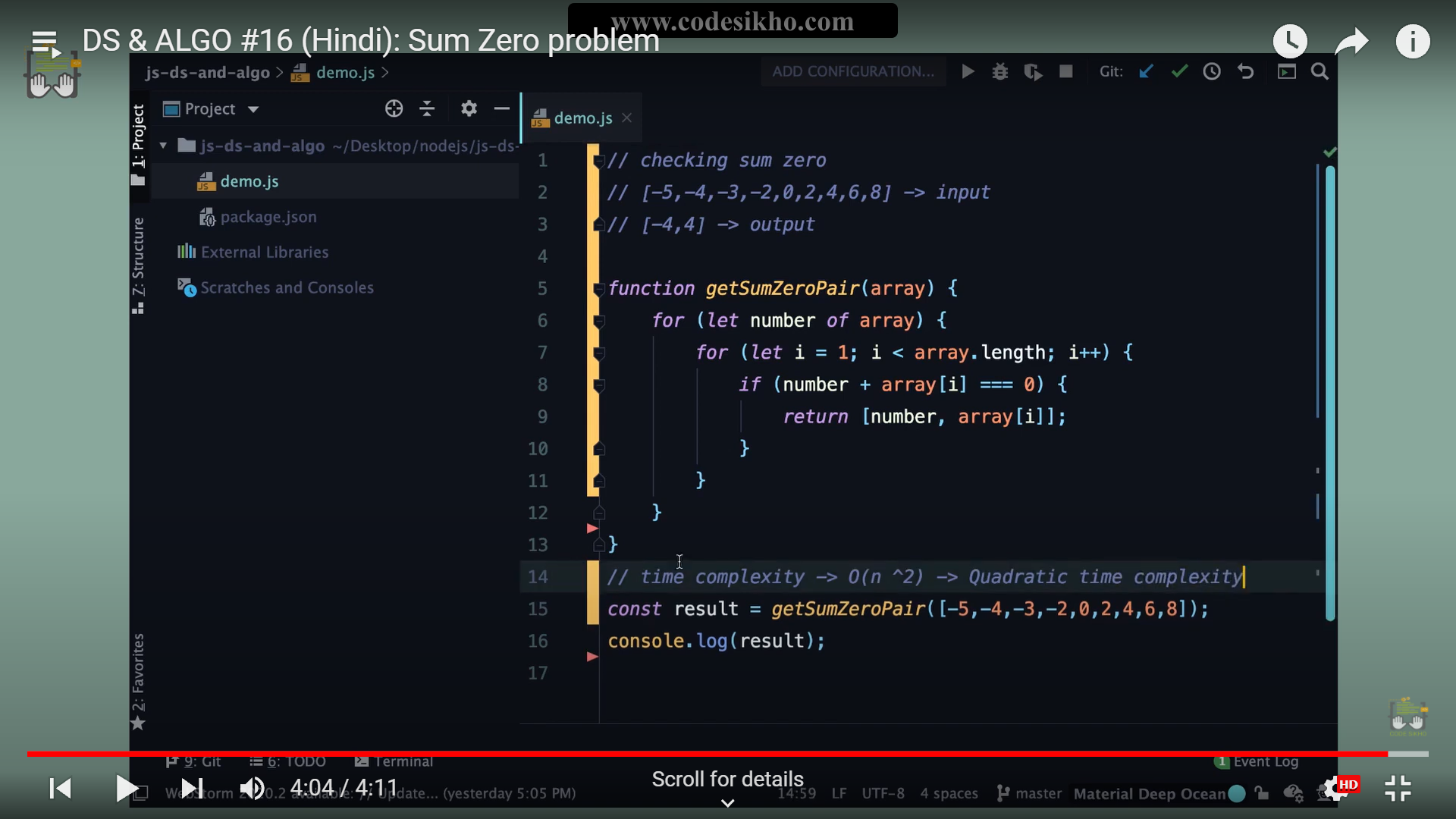
=== find one array element square is available in 2nd array or not

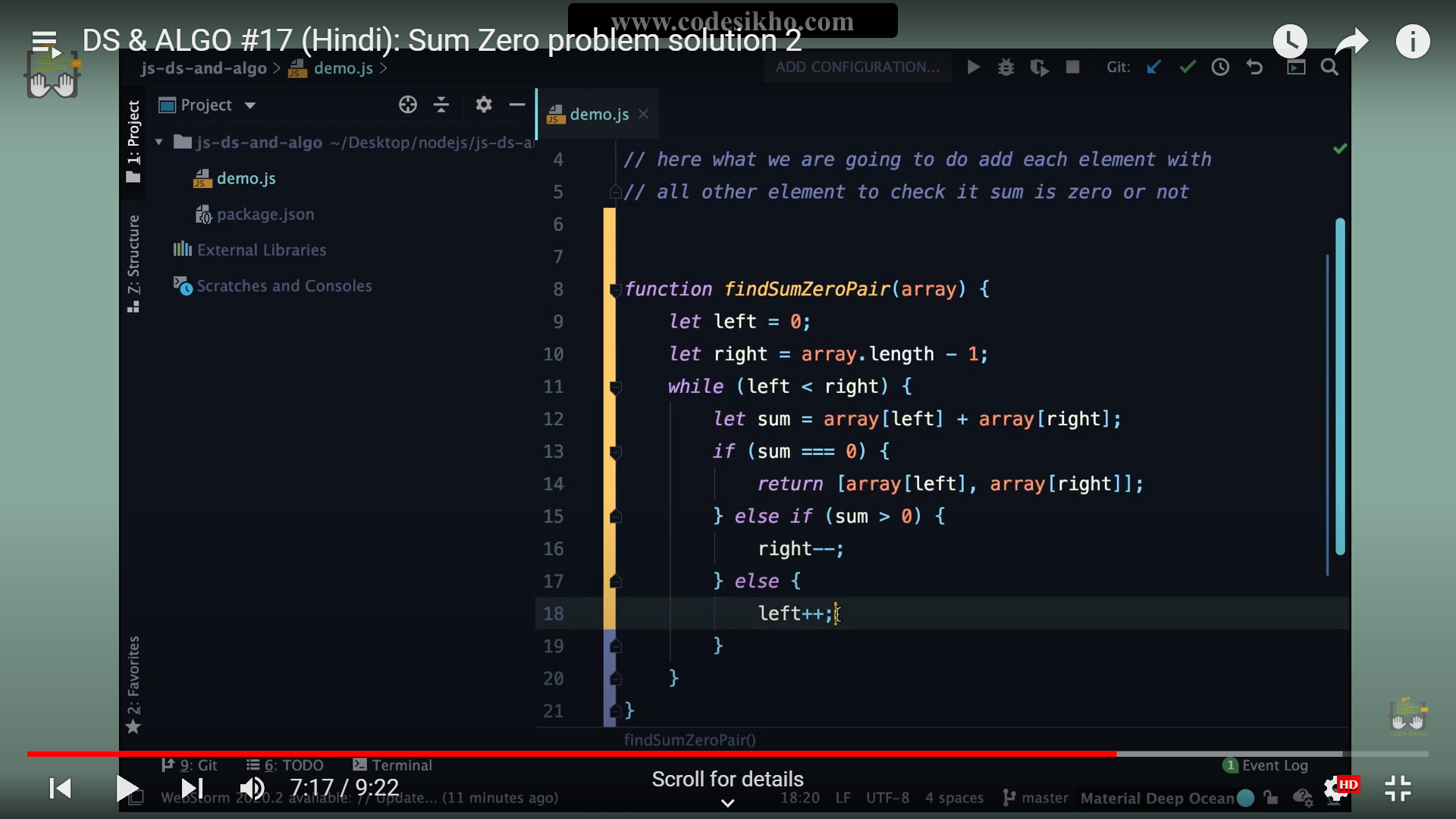
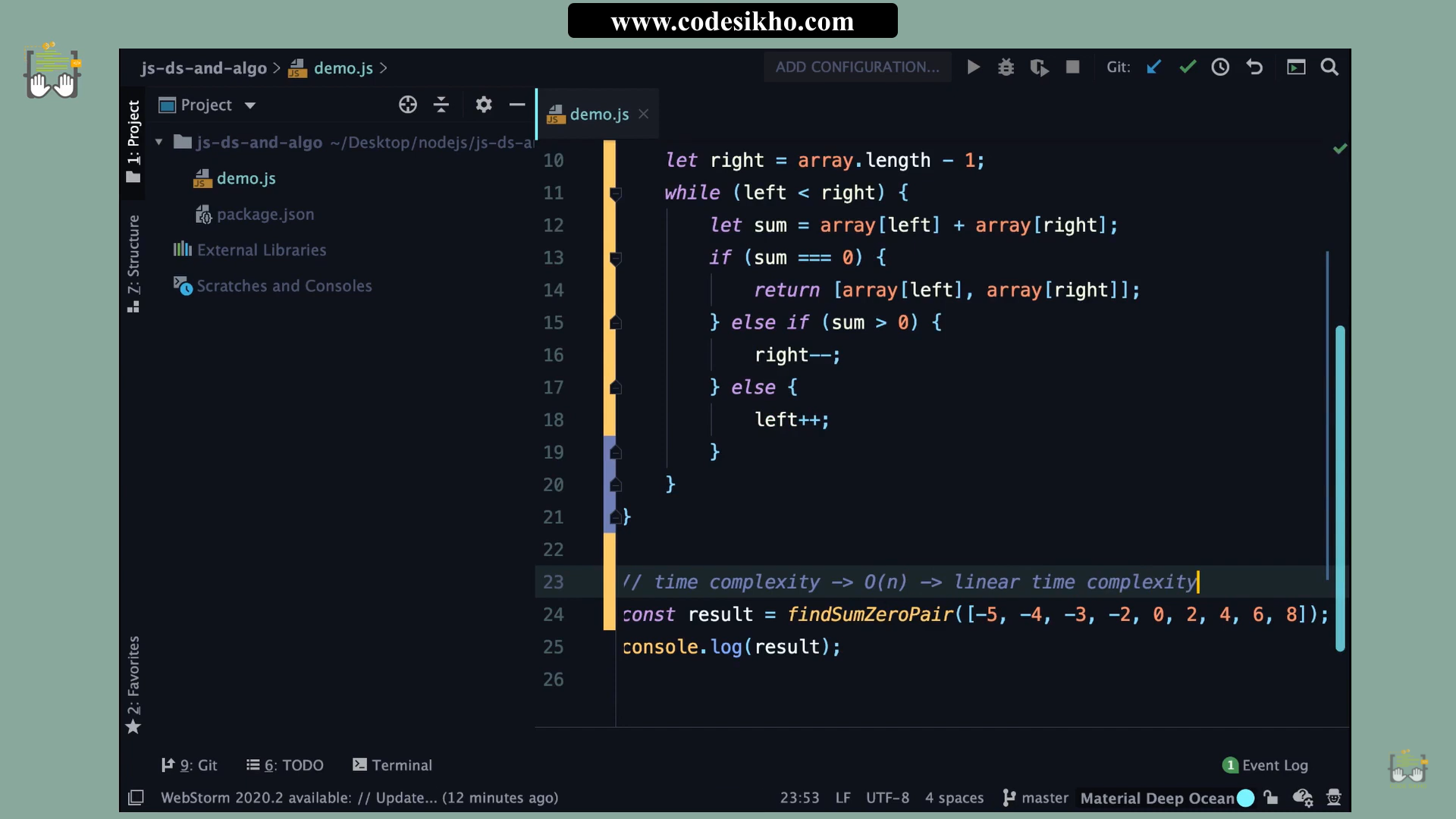


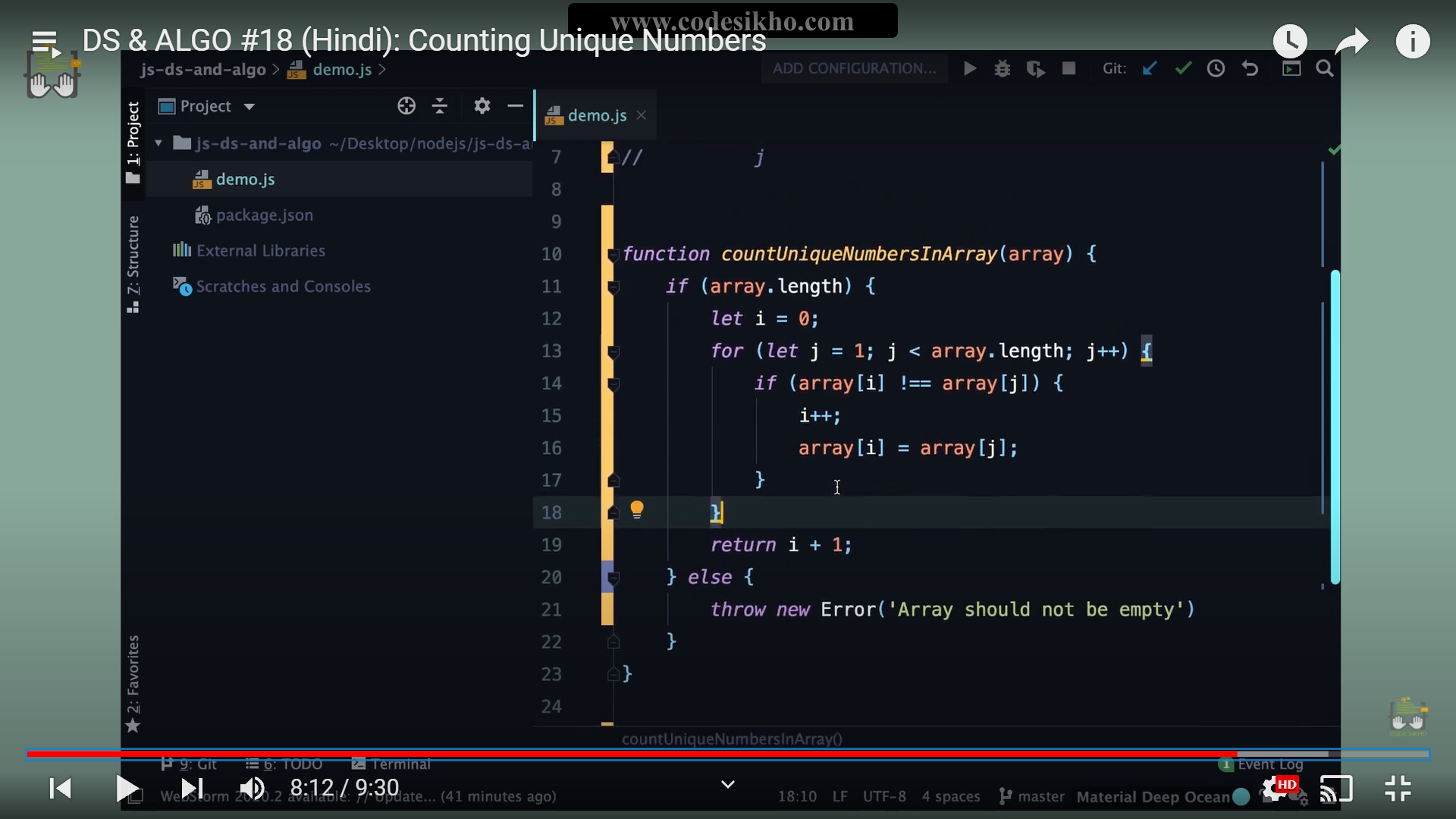
2nd way   
  


=== Anagram data structure

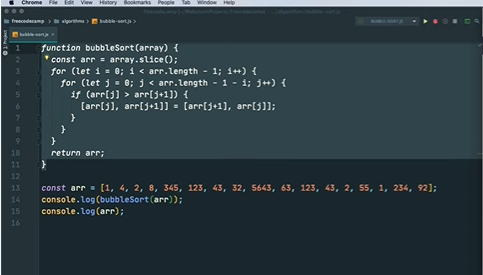


=====checking sum zero  


===Best way for same  
  


=== counting unique number  


=====bubble sorting  
  
===buble sort with pure implementation. Means it will not change it’s original array after sorting

  
==== recursive=== sorting  
const bubbleSort = function (array, pointer = array.length - 1) {

  // Base Case

  if (pointer === 0) {

    return array;

  }

  for (let i = 0; i < pointer; i++) {

    if (array[i] > array[i + 1]) {

      let temp = array[i + 1];

      array[i + 1] = array[i];

      array[i] = temp;

    }

  }

  // Recursive call on smaller portion of the array

  return bubbleSort(array, pointer - 1);

};

console.log(bubbleSort([1,-1,4444,33,444566]));

Binary searych=====

let recursiveFunction = function (arr, x, start, end) {

if (start > end) return false;

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

if (arr[mid]===x) return true;

if(arr[mid] > x)

return recursiveFunction(arr, x, start, mid-1);

else

return recursiveFunction(arr, x, mid+1, end);

}

// Driver code

let arr = [1, 3, 5, 7, 8, 9];

let x = 5;

recursiveFunction(arr, x, 0, arr.length-1)

**Binary searych=====**

let iterativeFunction = function (arr, x) {

let start=0, end=arr.length-1;

while (start<=end){

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

if (arr[mid]===x) return true;

else if (arr[mid] < x)

start = mid + 1;

else

end = mid - 1;

}

return false;

}

========================= **Time complexity** ========================================

Note : **Big O** notation is the most common metric for calculating time complexity. It describes the execution time of a task in relation to the number of steps required to complete it.

Big O notation is written in the form of **O(n)**

1. **Loop inside loop === o(n2)** for (let i = 0; i < pointer; i++) {

{

for (let j = 0; j < pointer; j++) {

}

}

1. Two loop but not inside a loop = ==**o(n)**  
   function test(array,lenght){

{

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

{

}

for (let j = 0; j < lenght; j++) {

}

}

1. When call recursive ===**o(n)**

function sum(arr){

if(arr===null){

return 0

}

return sum()+sum()

}

1. One loop with fixed loop === **o(1)**

for(let i=0;i<=1000;i++){

ddd

ddd

}