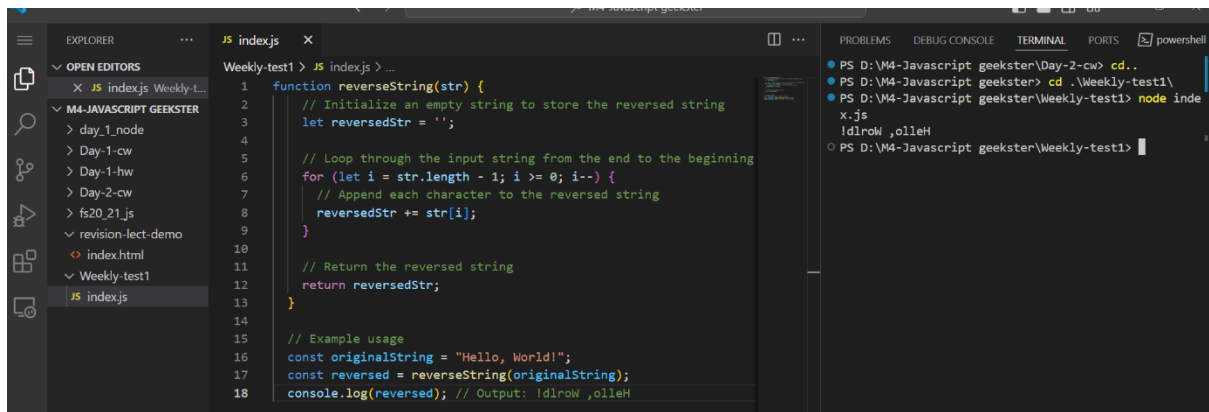


Q1-) String Reversal: Write a function to reverse a given string in JavaScript without using built-in reverse functions.



The screenshot shows a VS Code editor with a file named `index.js` open. The code defines a `reverseString` function that takes a string `str` and returns its reverse. It uses a `for` loop to iterate from the end of the string to the beginning, building the reversed string character by character. An example usage is provided at the bottom, reversing the string "Hello, World!". The terminal on the right shows the command `node index.js` being executed, resulting in the output `!dlrow ,olleH`.

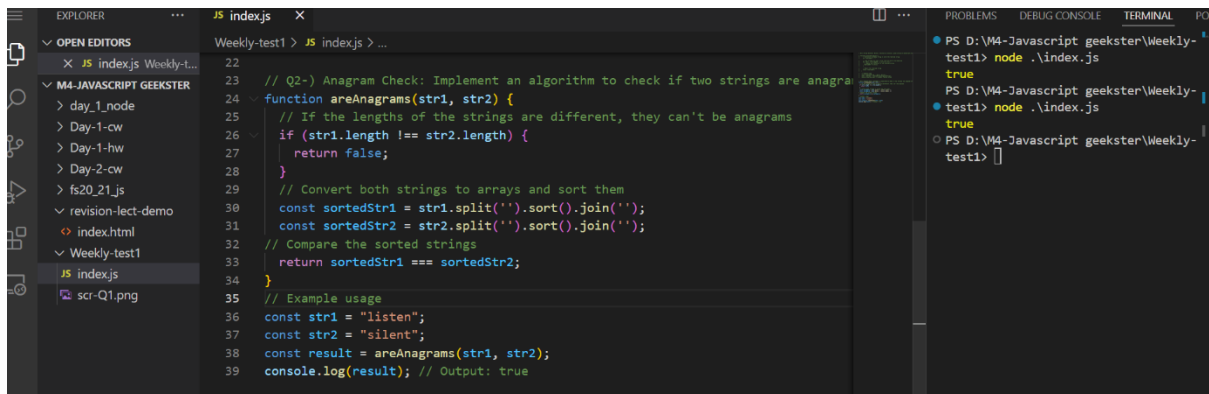
```
function reverseString(str) {
  // Initialize an empty string to store the reversed string
  let reversedStr = '';

  // Loop through the input string from the end to the beginning
  for (let i = str.length - 1; i >= 0; i--) {
    // Append each character to the reversed string
    reversedStr += str[i];
  }

  // Return the reversed string
  return reversedStr;
}

// Example usage
const originalString = "Hello, World!";
const reversed = reverseString(originalString);
console.log(reversed); // Output: !dlrow ,olleH
```

Q2-) Anagram Check: Implement an algorithm to check if two strings are anagrams of each other (contain the same characters with the same frequency)



The screenshot shows a VS Code editor with a file named `index.js` open. The code defines a `areAnagrams` function that takes two strings `str1` and `str2` and returns a boolean indicating if they are anagrams. The algorithm checks if the lengths are equal, then splits the strings into arrays, sorts them, and compares the sorted arrays. An example usage is provided, checking if "listen" and "silent" are anagrams, which returns `true`. The terminal on the right shows the command `node .\index.js` being executed, resulting in the output `true`.

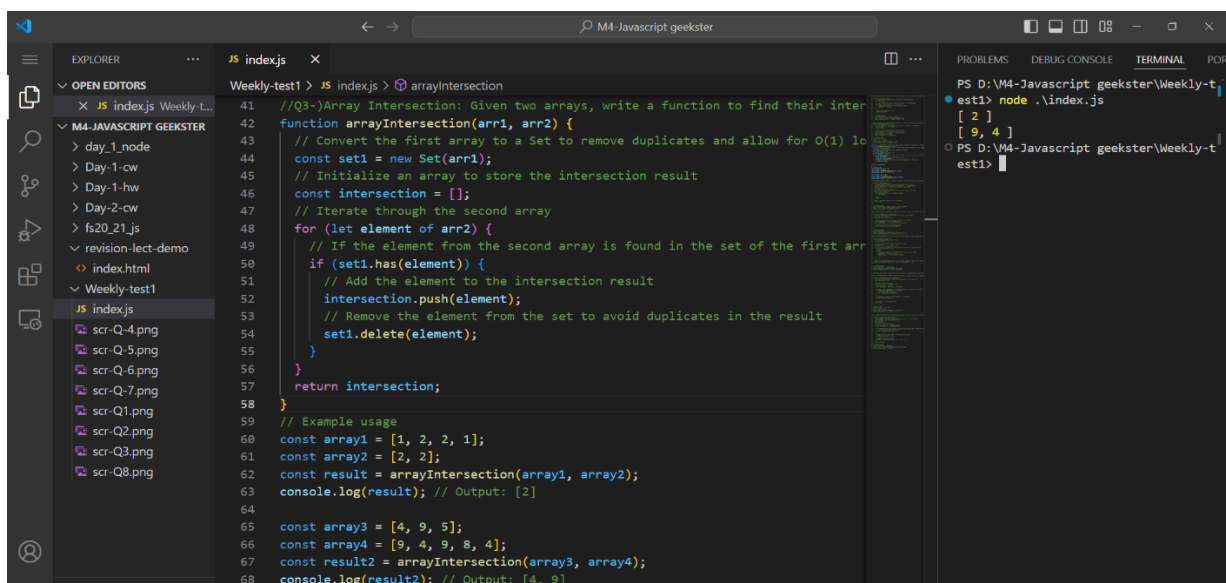
```
function areAnagrams(str1, str2) {
  // If the lengths of the strings are different, they can't be anagrams
  if (str1.length !== str2.length) {
    return false;
  }

  // Convert both strings to arrays and sort them
  const sortedStr1 = str1.split('').sort().join('');
  const sortedStr2 = str2.split('').sort().join('');

  // Compare the sorted strings
  return sortedStr1 === sortedStr2;
}

// Example usage
const str1 = "listen";
const str2 = "silent";
const result = areAnagrams(str1, str2);
console.log(result); // Output: true
```

Q3-) Array Intersection: Given two arrays, write a function to find their intersection (common elements).



The screenshot shows a VS Code editor with a file named `index.js` open. The code defines an `arrayIntersection` function that takes two arrays `arr1` and `arr2` and returns their intersection. It uses a `Set` to store the elements of the first array and then iterates through the second array to find common elements. An example usage is provided, finding the intersection of `[1, 2, 2, 1]` and `[2, 2]`, which returns `[2]`. The terminal on the right shows the command `node .\index.js` being executed, resulting in the output `[ 2 ]`.

```
function arrayIntersection(arr1, arr2) {
  // Convert the first array to a Set to remove duplicates and allow for O(1) lookups
  const set1 = new Set(arr1);

  // Initialize an array to store the intersection result
  const intersection = [];

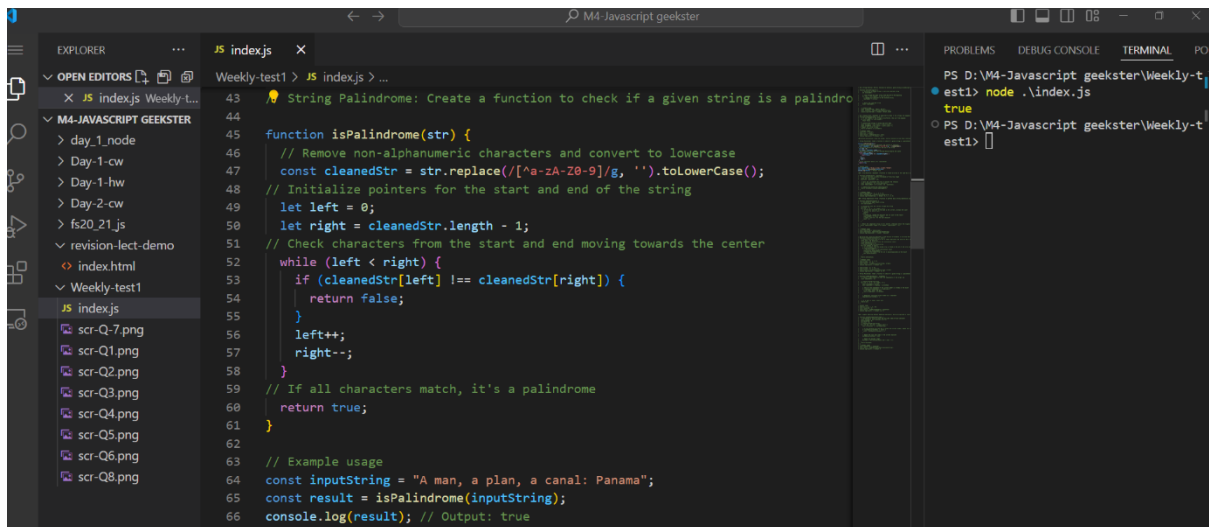
  // Iterate through the second array
  for (let element of arr2) {
    // If the element from the second array is found in the set of the first array
    if (set1.has(element)) {
      // Add the element to the intersection result
      intersection.push(element);
      // Remove the element from the set to avoid duplicates in the result
      set1.delete(element);
    }
  }

  return intersection;
}

// Example usage
const array1 = [1, 2, 2, 1];
const array2 = [2, 2];
const result = arrayIntersection(array1, array2);
console.log(result); // Output: [ 2 ]

const array3 = [4, 9, 5];
const array4 = [9, 4, 9, 8, 4];
const result2 = arrayIntersection(array3, array4);
console.log(result2); // Output: [ 4, 9 ]
```

Q4-) String Palindrome: Create a function to check if a given string is a palindrome (reads the same forwards and backwards) while ignoring non-alphanumeric characters.

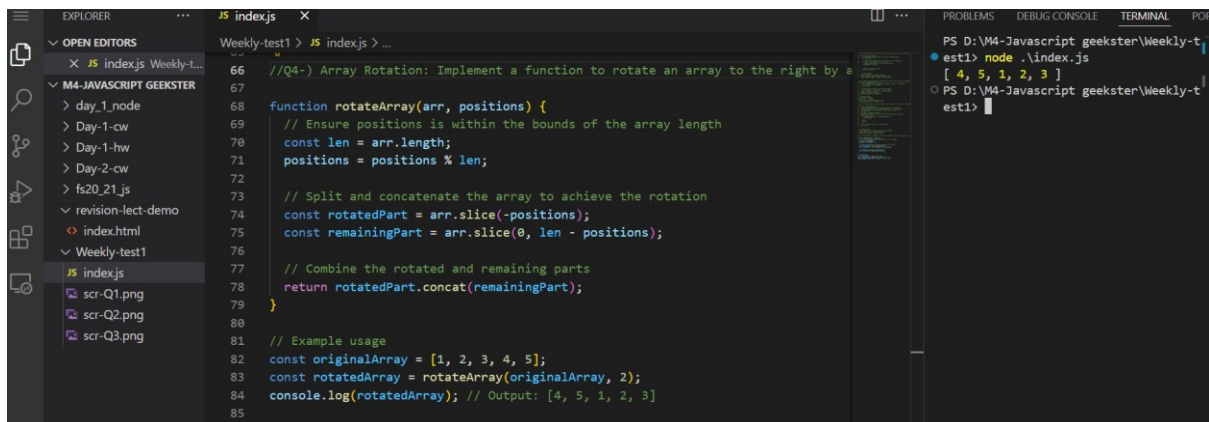


```
43 // String Palindrome: Create a function to check if a given string is a palindro
44
45 function isPalindrome(str) {
46   // Remove non-alphanumeric characters and convert to lowercase
47   const cleanedStr = str.replace(/[^a-zA-Z0-9]/g, '').toLowerCase();
48   // Initialize pointers for the start and end of the string
49   let left = 0;
50   let right = cleanedStr.length - 1;
51   // Check characters from the start and end moving towards the center
52   while (left < right) {
53     if (cleanedStr[left] !== cleanedStr[right]) {
54       return false;
55     }
56     left++;
57     right--;
58   }
59   // If all characters match, it's a palindrome
60   return true;
61 }
62
63 // Example usage
64 const inputString = "A man, a plan, a canal: Panama";
65 const result = isPalindrome(inputString);
66 console.log(result); // Output: true
```

Terminal output:

```
PS D:\M4-Javascript geekster\Weekly-t...
est1> node .\index.js
true
est1>
```

Q5-) Array Rotation: Implement a function to rotate an array to the right by a specified number of positions.

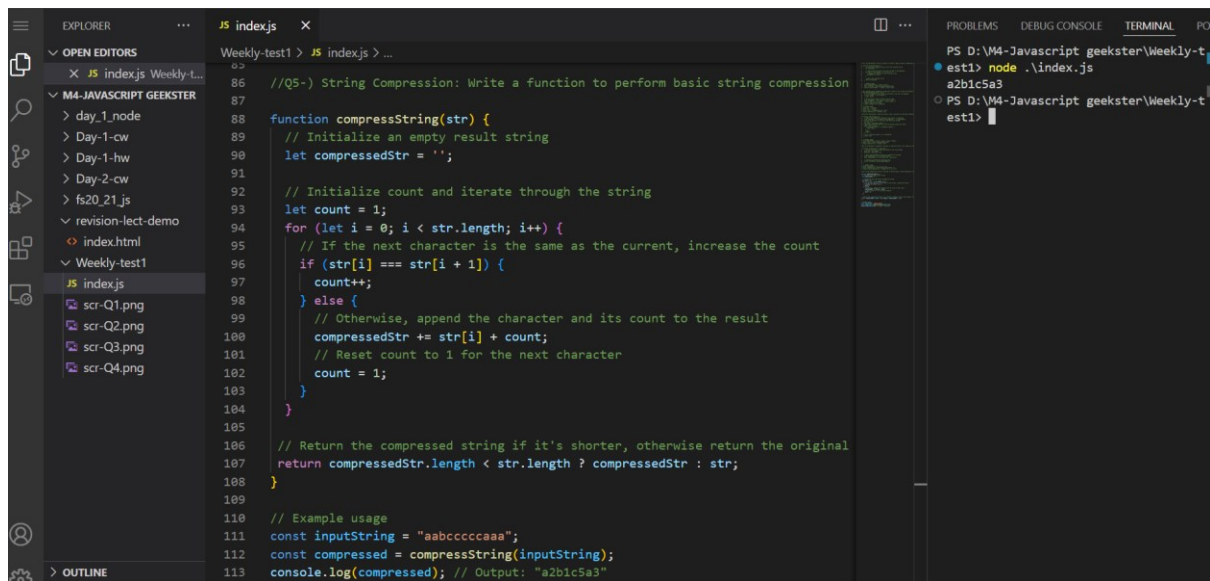


```
66 //Q4-) Array Rotation: Implement a function to rotate an array to the right by a
67
68 function rotateArray(arr, positions) {
69   // Ensure positions is within the bounds of the array length
70   const len = arr.length;
71   positions = positions % len;
72
73   // Split and concatenate the array to achieve the rotation
74   const rotatedPart = arr.slice(-positions);
75   const remainingPart = arr.slice(0, len - positions);
76
77   // Combine the rotated and remaining parts
78   return rotatedPart.concat(remainingPart);
79 }
80
81 // Example usage
82 const originalArray = [1, 2, 3, 4, 5];
83 const rotatedArray = rotateArray(originalArray, 2);
84 console.log(rotatedArray); // Output: [4, 5, 1, 2, 3]
85
```

Terminal output:

```
PS D:\M4-Javascript geekster\Weekly-t...
est1> node .\index.js
[ 4, 5, 1, 2, 3 ]
est1>
```

Q6-) String Compression: Write a function to perform basic string compression using the counts of repeated characters. For example, "aabcccccaaa" would become "a2b1c5a3."

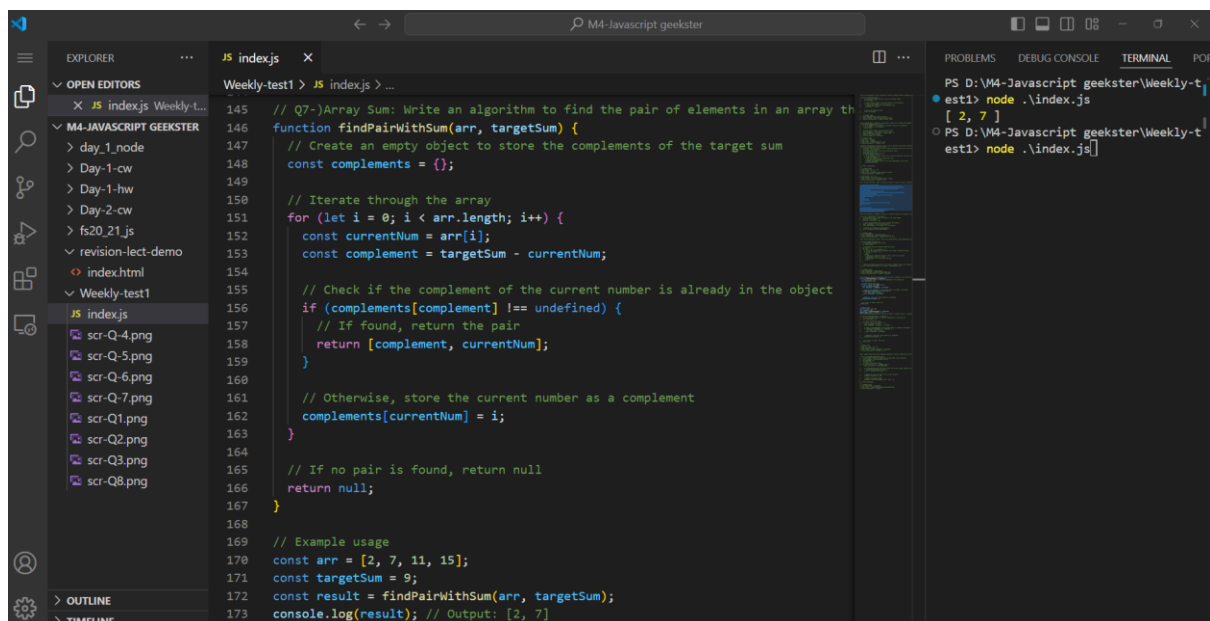


The screenshot shows a VS Code editor with a file explorer on the left, a code editor in the center, and a terminal on the right. The code editor displays a JavaScript file named 'index.js' with the following content:

```
Weekly-test1 > JS indexjs > ...
86 //Q5-) String Compression: Write a function to perform basic string compression
87
88 function compressString(str) {
89     // Initialize an empty result string
90     let compressedStr = '';
91
92     // Initialize count and iterate through the string
93     let count = 1;
94     for (let i = 0; i < str.length; i++) {
95         // If the next character is the same as the current, increase the count
96         if (str[i] === str[i + 1]) {
97             count++;
98         } else {
99             // Otherwise, append the character and its count to the result
100             compressedStr += str[i] + count;
101             // Reset count to 1 for the next character
102             count = 1;
103         }
104     }
105
106     // Return the compressed string if it's shorter, otherwise return the original
107     return compressedStr.length < str.length ? compressedStr : str;
108 }
109
110 // Example usage
111 const inputString = "aabcccccaaa";
112 const compressed = compressString(inputString);
113 console.log(compressed); // Output: "a2b1c5a3"
```

The terminal on the right shows the command 'PS D:\M4-Javascript geekster\Weekly-t... est1> node .\index.js' and the output 'a2b1c5a3'.

Q7-) Array Sum: Write an algorithm to find the pair of elements in an array that adds up to a specific target sum.

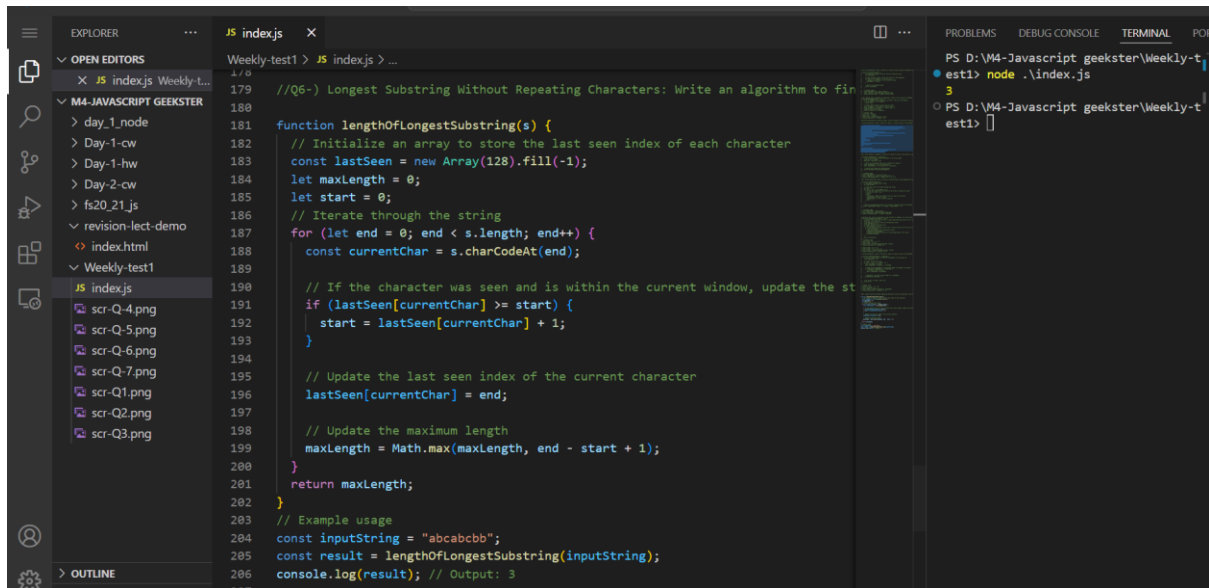


The screenshot shows a VS Code editor with a file explorer on the left, a code editor in the center, and a terminal on the right. The code editor displays a JavaScript file named 'index.js' with the following content:

```
Weekly-test1 > JS indexjs > ...
145 // Q7-)Array Sum: Write an algorithm to find the pair of elements in an array th
146 function findPairWithSum(arr, targetSum) {
147     // Create an empty object to store the complements of the target sum
148     const complements = {};
149
150     // Iterate through the array
151     for (let i = 0; i < arr.length; i++) {
152         const currentNum = arr[i];
153         const complement = targetSum - currentNum;
154
155         // Check if the complement of the current number is already in the object
156         if (complements[complement] !== undefined) {
157             // If found, return the pair
158             return [complement, currentNum];
159         }
160
161         // Otherwise, store the current number as a complement
162         complements[currentNum] = i;
163     }
164
165     // If no pair is found, return null
166     return null;
167 }
168
169 // Example usage
170 const arr = [2, 7, 11, 15];
171 const targetSum = 9;
172 const result = findPairWithSum(arr, targetSum);
173 console.log(result); // Output: [2, 7]
```

The terminal on the right shows the command 'PS D:\M4-Javascript geekster\Weekly-t... est1> node .\index.js' and the output '[ 2, 7 ]'.

Q8-) Longest Substring Without Repeating Characters: Write an algorithm to find the length of the longest substring without repeating characters in a given string.



```
179 //Q6-) Longest Substring Without Repeating Characters: Write an algorithm to find
180
181 function lengthOfLongestSubstring(s) {
182     // Initialize an array to store the last seen index of each character
183     const lastSeen = new Array(128).fill(-1);
184     let maxLength = 0;
185     let start = 0;
186     // Iterate through the string
187     for (let end = 0; end < s.length; end++) {
188         const currentChar = s.charAt(end);
189
190         // If the character was seen and is within the current window, update the start
191         if (lastSeen[currentChar] >= start) {
192             start = lastSeen[currentChar] + 1;
193         }
194
195         // Update the last seen index of the current character
196         lastSeen[currentChar] = end;
197
198         // Update the maximum length
199         maxLength = Math.max(maxLength, end - start + 1);
200     }
201     return maxLength;
202 }
203
204 // Example usage
205 const inputString = "abcabcb";
206 const result = lengthOfLongestSubstring(inputString);
207 console.log(result); // Output: 3
```

PS D:\M4-Javascript geekster\Weekly-t...  
est1> node .\index.js  
3  
est1> []