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

```javascript
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
```

```
PS D:\M4-Javascript geekster\Day-2-cw> cd..
PS D:\M4-Javascript geekster> cd .\Weekly-test1\
PS D:\M4-Javascript geekster\Weekly-test1> node index.js
!dlroW ,olleH
PS D:\M4-Javascript geekster\Weekly-test1>
```

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

```javascript
// Q2-) Anagram Check: Implement an algorithm to check if two strings are anagra
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
```

```
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
true
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
true
PS D:\M4-Javascript geekster\Weekly-test1>
```

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

```javascript
//Q3-)Array Intersection: Given two arrays, write a function to find their inter
function arrayIntersection(arr1, arr2) {
  // Convert the first array to a Set to remove duplicates and allow for O(1) lo
  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 arr
    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]
```

```
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
[ 2 ]
[ 9, 4 ]
PS D:\M4-Javascript geekster\Weekly-test1>
```

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.

```javascript
// String Palindrome: Create a function to check if a given string is a palindro

function isPalindrome(str) {
  // Remove non-alphanumeric characters and convert to lowercase
  const cleanedStr = str.replace(/[^a-zA-Z0-9]/g, '').toLowerCase();
// Initialize pointers for the start and end of the string
  let left = 0;
  let right = cleanedStr.length - 1;
// Check characters from the start and end moving towards the center
  while (left < right) {
    if (cleanedStr[left] !== cleanedStr[right]) {
      return false;
    }
    left++;
    right--;
  }
// If all characters match, it's a palindrome
  return true;
}

// Example usage
const inputString = "A man, a plan, a canal: Panama";
const result = isPalindrome(inputString);
console.log(result); // Output: true
```

```
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
true
PS D:\M4-Javascript geekster\Weekly-test1>
```

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

```javascript
//Q4-) Array Rotation: Implement a function to rotate an array to the right by a

function rotateArray(arr, positions) {
  // Ensure positions is within the bounds of the array length
  const len = arr.length;
  positions = positions % len;

  // Split and concatenate the array to achieve the rotation
  const rotatedPart = arr.slice(-positions);
  const remainingPart = arr.slice(0, len - positions);

  // Combine the rotated and remaining parts
  return rotatedPart.concat(remainingPart);
}

// Example usage
const originalArray = [1, 2, 3, 4, 5];
const rotatedArray = rotateArray(originalArray, 2);
console.log(rotatedArray); // Output: [4, 5, 1, 2, 3]
```

```
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
[ 4, 5, 1, 2, 3 ]
PS D:\M4-Javascript geekster\Weekly-test1>
```

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

```javascript
//Q5-) String Compression: Write a function to perform basic string compression

function compressString(str) {
  // Initialize an empty result string
  let compressedStr = '';

  // Initialize count and iterate through the string
  let count = 1;
  for (let i = 0; i < str.length; i++) {
    // If the next character is the same as the current, increase the count
    if (str[i] === str[i + 1]) {
      count++;
    } else {
      // Otherwise, append the character and its count to the result
      compressedStr += str[i] + count;
      // Reset count to 1 for the next character
      count = 1;
    }
  }

 // Return the compressed string if it's shorter, otherwise return the original
 return compressedStr.length < str.length ? compressedStr : str;
}

// Example usage
const inputString = "aabcccccaaa";
const compressed = compressString(inputString);
console.log(compressed); // Output: "a2b1c5a3"
```

```
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
a2b1c5a3
PS D:\M4-Javascript geekster\Weekly-test1>
```

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

```javascript
// Q7-)Array Sum: Write an algorithm to find the pair of elements in an array th
function findPairWithSum(arr, targetSum) {
  // Create an empty object to store the complements of the target sum
  const complements = {};

  // Iterate through the array
  for (let i = 0; i < arr.length; i++) {
    const currentNum = arr[i];
    const complement = targetSum - currentNum;

    // Check if the complement of the current number is already in the object
    if (complements[complement] !== undefined) {
      // If found, return the pair
      return [complement, currentNum];
    }

    // Otherwise, store the current number as a complement
    complements[currentNum] = i;
  }

  // If no pair is found, return null
  return null;
}

// Example usage
const arr = [2, 7, 11, 15];
const targetSum = 9;
const result = findPairWithSum(arr, targetSum);
console.log(result); // Output: [2, 7]
```

```
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
[ 2, 7 ]
PS D:\M4-Javascript geekster\Weekly-test1> node .\index.js
```

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