1. Palindrome Check

Problem Statement: Write a TypeScript function `isPalindrome(input: string): boolean` that checks if a given string is a palindrome (reads the same backward as forward). Ignore case and non-alphanumeric characters.

function isPalindrome(input: string): boolean {

const sanitizedInput = input.replace(/[^a-zA-Z0-9]/g, '').toLowerCase();

const reversedInput = sanitizedInput.split('').reverse().join('');

return sanitizedInput === reversedInput;

}

// Example usage:

console.log(isPalindrome("A man, a plan, a canal, Panama")); // true

console.log(isPalindrome("hello")); // false

```

2. Fibonacci Sequence

Problem Statement: Implement a TypeScript function `fibonacci(n: number): number` that returns the `n-th` Fibonacci number. Use an iterative approach to avoid excessive recursion.

function fibonacci(n: number): number {

if (n <= 1) return n;

let a = 0, b = 1;

for (let i = 2; i <= n; i++) {

[a, b] = [b, a + b];

}

return b;

}

// Example usage:

console.log(fibonacci(6)); // 8

console.log(fibonacci(10)); // 55

```

3. Prime Number Check

Problem Statement: Create a TypeScript function `isPrime(n: number): boolean` that determines if a given number `n` is a prime number.

function isPrime(n: number): boolean {

if (n <= 1) return false;

if (n <= 3) return true;

if (n % 2 === 0 || n % 3 === 0) return false;

for (let i = 5; i \* i <= n; i += 6) {

if (n % i === 0 || n % (i + 2) === 0) return false;

}

return true;

}

// Example usage:

console.log(isPrime(7)); // true

console.log(isPrime(10)); // false

```

4. Array Flattening

Problem Statement: Write a TypeScript function `flattenArray(arr: any[]): any[]` that flattens a nested array of any depth.

function flattenArray(arr: any[]): any[] {

return arr.reduce((acc, val) => Array.isArray(val) ? acc.concat(flattenArray(val)) : acc.concat(val), []);

}

// Example usage:

console.log(flattenArray([1, [2, [3, 4], 5], 6])); // [1, 2, 3, 4, 5, 6]

```

5. String Anagram

Problem Statement: Implement a TypeScript function `areAnagrams(str1: string, str2: string): boolean` that checks if two strings are anagrams of each other.

function areAnagrams(str1: string, str2: string): boolean {

const sanitizeString = (str: string): string => str.replace(/[^\w]/g, '').toLowerCase().split('').sort().join('');

return sanitizeString(str1) === sanitizeString(str2);

}

// Example usage:

console.log(areAnagrams("listen", "silent")); // true

console.log(areAnagrams("hello", "world")); // false

```

6. Binary Search

Problem Statement: Create a TypeScript function `binarySearch(arr: number[], target: number): number` that performs a binary search on a sorted array and returns the index of the target element, or -1 if it is not found.

function binarySearch(arr: number[], target: number): number {

let left = 0, right = arr.length - 1;

while (left <= right) {

const mid = Math.floor((left + right) / 2);

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

if (arr[mid] < target) left = mid + 1;

else right = mid - 1;

}

return -1;

}

// Example usage:

console.log(binarySearch([1, 2, 3, 4, 5, 6], 4)); // 3

console.log(binarySearch([1, 2, 3, 4, 5, 6], 7)); // -1

```

7. Array Chunking

Problem Statement: Write a TypeScript function `chunkArray(arr: any[], size: number): any[][]` that splits an array into chunks of a specified size.

function chunkArray(arr: any[], size: number): any[][] {

const result = [];

for (let i = 0; i < arr.length; i += size) {

result.push(arr.slice(i, i + size));

}

return result;

}

// Example usage:

console.log(chunkArray([1, 2, 3, 4, 5, 6], 2)); // [[1, 2], [3, 4], [5, 6]]

console.log(chunkArray([1, 2, 3, 4, 5, 6], 3)); // [[1, 2, 3], [4, 5, 6]]

```

8. Reverse Words in a String

Problem Statement: Implement a TypeScript function `reverseWords(sentence: string): string` that reverses the order of words in a given sentence.

function reverseWords(sentence: string): string {

return sentence.split(' ').reverse().join(' ');

}

// Example usage:

console.log(reverseWords("Hello world this is TypeScript")); // "TypeScript is this world Hello"

```

9. Array Intersection

Problem Statement: Create a TypeScript function `arrayIntersection(arr1: any[], arr2: any[]): any[]` that returns a new array containing only the elements present in both input arrays.

function arrayIntersection(arr1: any[], arr2: any[]): any[] {

const set2 = new Set(arr2);

return arr1.filter(item => set2.has(item));

}

// Example usage:

console.log(arrayIntersection([1, 2, 3], [2, 3, 4])); // [2, 3]

```

10. Count Vowels

Problem Statement: Write a TypeScript function `countVowels(input: string): number` that counts the number of vowels in a given string.

function countVowels(input: string): number {

const vowels = 'aeiouAEIOU';

return input.split('').filter(char => vowels.includes(char)).length;

}

// Example usage:

console.log(countVowels("Hello world")); // 3

```

11. Sum of Digits

Problem Statement: Implement a TypeScript function `sumOfDigits(n: number): number` that calculates the sum of the digits of a given number.

function sumOfDigits(n: number): number {

return n.toString().split('').reduce((acc, digit) => acc + parseInt(digit, 10), 0);

}

// Example usage:

console.log(sumOfDigits(12345)); // 15

```

12. Find Duplicates in Array

Problem Statement: Create a TypeScript function `findDuplicates(arr: any[]): any[]` that returns an array of duplicate elements from the input array.

function findDuplicates(arr: any[]): any[] {

const seen = new Set();

const duplicates = new Set();

for (const item of arr) {

if (seen.has(item)) {

duplicates.add(item);

} else {

seen.add(item);

}

}

return Array.from(duplicates);

}

// Example usage:

console.log(findDuplicates([1, 2, 2, 3, 4, 4, 5])); // [2, 4]

```

13. Longest Common Prefix

Problem Statement: Write a TypeScript function `longestCommonPrefix(strs: string[]): string` that finds the longest common prefix string amongst an array of strings.

function longestCommonPrefix(strs: string[]): string {

if (strs.length === 0) return '';

let prefix = strs[0];

for (const str of strs) {

while (str.indexOf(prefix) !== 0) {

prefix = prefix.slice(0, -1);

if (prefix === '') return '';

}

}

return prefix;

}

// Example usage:

console.log(longestCommonPrefix(["flower", "flow", "flight"])); // "fl"

console.log(longestCommonPrefix(["dog", "racecar", "car"])); // ""

```

14. Balanced Parentheses

Problem Statement: Implement a TypeScript function `isBalanced(input: string): boolean` that checks if the parentheses in a given string are balanced.

function isBalanced(input: string): boolean {

const stack = [];

const pairs: { [key: string]: string } = { ')': '(', '}': '{', ']': '[' };

for (const char of input) {

if (['(', '{', '['].includes(char)) {

stack.push(char);

} else if ([')', '}', ']'].includes(char)) {

if (stack.pop() !== pairs[char]) return false;

}

}

return stack.length === 0;

}

// Example usage

:

console.log(isBalanced("()[]{}")); // true

console.log(isBalanced("(]")); // false

```

15. Merge Sorted Arrays

Problem Statement: Create a TypeScript function `mergeSortedArrays(arr1: number[], arr2: number[]): number[]` that merges two sorted arrays into one sorted array.

function mergeSortedArrays(arr1: number[], arr2: number[]): number[] {

let i = 0, j = 0;

const result = [];

while (i < arr1.length && j < arr2.length) {

if (arr1[i] < arr2[j]) {

result.push(arr1[i++]);

} else {

result.push(arr2[j++]);

}

}

return result.concat(arr1.slice(i)).concat(arr2.slice(j));

}

// Example usage:

console.log(mergeSortedArrays([1, 3, 5], [2, 4, 6])); // [1, 2, 3, 4, 5, 6]

```

16. Remove Duplicates from Array

Problem Statement: Write a TypeScript function `removeDuplicates(arr: any[]): any[]` that removes duplicate elements from an array.

function removeDuplicates(arr: any[]): any[] {

return Array.from(new Set(arr));

}

// Example usage:

console.log(removeDuplicates([1, 2, 2, 3, 4, 4, 5])); // [1, 2, 3, 4, 5]

```

17. Caesar Cipher

Problem Statement: Implement a TypeScript function `caesarCipher(str: string, shift: number): string` that performs a Caesar cipher encryption on a given string with a specified shift.

function caesarCipher(str: string, shift: number): string {

return str.split('').map(char => {

if (char.match(/[a-z]/i)) {

const code = char.charCodeAt(0);

const base = char >= 'a' ? 97 : 65;

return String.fromCharCode(((code - base + shift) % 26) + base);

}

return char;

}).join('');

}

// Example usage:

console.log(caesarCipher("Hello World", 3)); // "Khoor Zruog"

```

18. Frequency Counter

Problem Statement: Create a TypeScript function `frequencyCounter(arr: any[]): { [key: string]: number }` that returns an object representing the frequency count of each element in the array.

function frequencyCounter(arr: any[]): { [key: string]: number } {

return arr.reduce((acc, val) => {

acc[val] = (acc[val] || 0) + 1;

return acc;

}, {} as { [key: string]: number });

}

// Example usage:

console.log(frequencyCounter(["apple", "banana", "apple", "orange", "banana", "apple"]));

// { apple: 3, banana: 2, orange: 1 }

```

19. Array Rotation

Problem Statement: Write a TypeScript function `rotateArray(arr: any[], k: number): any[]` that rotates an array to the right by `k` steps.

function rotateArray(arr: any[], k: number): any[] {

const len = arr.length;

k = k % len;

return arr.slice(len - k).concat(arr.slice(0, len - k));

}

// Example usage:

console.log(rotateArray([1, 2, 3, 4, 5], 2)); // [4, 5, 1, 2, 3]

```

20. String Compression

Problem Statement: Implement a TypeScript function `compressString(input: string): string` that compresses a string using the counts of repeated characters (e.g., "aabcccccaaa" -> "a2b1c5a3").

function compressString(input: string): string {

let compressed = '';

let count = 1;

for (let i = 0; i < input.length; i++) {

if (input[i] === input[i + 1]) {

count++;

} else {

compressed += input[i] + count;

count = 1;

}

}

return compressed.length < input.length ? compressed : input;

}

// Example usage:

console.log(compressString("aabcccccaaa")); // "a2b1c5a3"

console.log(compressString("abcd")); // "abcd"

```