7

Coding Problems

Every SDET Gets Asked

(With Solutions You Can Use Tomorrow)

Aston Cook
Senior QA Automation Engineer

01

Reverse a String

Why They Ask This

This is the warm-up question. Interviewers use it to see how you handle basic string manipulation and whether you can write clean code under pressure. It seems simple, but many candidates stumble on edge cases or overcomplicate it.

The Problem

Write a function that takes a string and returns it reversed. Handle empty strings and single characters.

The Solution

```
// JavaScript Solutionfunction reverseString(str) {  return
str.split('').reverse().join('');}// Without built-in
methods:function reverseStringManual(str) {  let reversed = '';
for (let i = str.length - 1; i >= 0; i--) {    reversed +=
str[i];  }  return reversed;}// Test cases to mention://
reverseString("hello") → "olleh"// reverseString("") → ""//
reverseString("a") → "a"
```

Interview Tip: Show both approaches. Mention time complexity is O(n). Ask if they want you to handle special characters or Unicode.

02

Find Duplicates in Array

Why They Ask This

Tests your knowledge of data structures. Interviewers want to see if you reach for a Set or HashMap instead of nested loops. This separates candidates who understand efficiency from those who brute force everything.

The Problem

Given an array of integers, return all elements that appear more than once.

The Solution

```
// JavaScript Solutionfunction findDuplicates(arr) {  const seen = new Set();  const duplicates = new Set();    for (const num of arr) {    if (seen.has(num)) {      duplicates.add(num); } else {      seen.add(num);    }  }  return [...duplicates];}// Test cases:// findDuplicates([1,2,3,2,4,3]) → [2,3]// findDuplicates([1,2,3]) → []// findDuplicates([]) → []
```

Interview Tip: Explain why Set gives O(n) vs O(n²) with nested loops. Ask about handling negative numbers or if order matters.

03

Valid Palindrome

Why They Ask This

Common in SDET interviews because it tests string cleaning, edge case handling, and logical thinking. Interviewers often add constraints mid-question to see how you adapt.

The Problem

Check if a string is a palindrome, considering only alphanumeric characters and ignoring case.

The Solution

```
// JavaScript Solutionfunction isPalindrome(str) {  const
cleaned = str    .toLowerCase()    .replace(/[^a-z0-9]/g, '');
return cleaned === cleaned    .split('')    .reverse()
.join('');}// Two-pointer approach (more efficient):function
isPalindromeTwoPointer(str) {  const cleaned =
str.toLowerCase()    .replace(/[^a-z0-9]/g, '');  let left = 0;
let right = cleaned.length - 1;    while (left < right) {    if
(cleaned[left] !== cleaned[right]) {      return false;    }
left++;    right--;  }  return true;}// Test cases://
isPalindrome("A man, a plan, a canal: Panama") → true//
isPalindrome("race a car") → false
```

Interview Tip: Two-pointer solution uses O(1) extra space. Mention this optimization proactively.

04

FizzBuzz

Why They Ask This

The classic screening question. It filters out candidates who cannot write basic conditional logic. Sounds trivial, but 30-40% of candidates still struggle with the modulo operator or condition ordering.

The Problem

Print numbers 1-100. For multiples of 3 print 'Fizz', multiples of 5 print 'Buzz', multiples of both print 'FizzBuzz'.

The Solution

```
// JavaScript Solutionfunction fizzBuzz(n) {  const result = [];    for (let i = 1; i <= n; i++) {    if (i % 15 === 0) {  result.push('FizzBuzz');    } else if (i % 3 === 0) {  result.push('Fizz');    } else if (i % 5 === 0) {  result.push('Buzz');    } else {  result.push(i.toString());    }  }  return result;}// Cleaner approach:function fizzBuzzClean(n) {  return Array.from({length: n}, (_, i) => {    const num = i + 1;  const fizz = num % 3 === 0 ? 'Fizz' : '';    const buzz = num % 5 === 0 ? 'Buzz' : '';    return fizz + buzz || num.toString();  });}
```

Interview Tip: Check for 15 first (or build the string). Ask if they want it extensible for other divisors.

05

Two Sum

Why They Ask This

The most common interview question in tech. Tests your ability to optimize from brute force O(n²) to HashMap O(n). If you cannot solve this, the interview is likely over.

The Problem

Given an array of numbers and a target, return indices of two numbers that add up to the target.

The Solution

```
// JavaScript Solution - O(n) timefunction twoSum(nums, target) {  const map = new Map();    for (let i = 0; i < nums.length; i++) {    const complement = target - nums[i];        if (map.has(complement)) {      return [map.get(complement), i];    }    map.set(nums[i], i);  }  return []; // No solution found}// Test cases:// twoSum([2,7,11,15], 9) → [0,1]// twoSum([3,2,4], 6) → [1,2]// twoSum([3,3], 6) → [0,1]
```

Interview Tip: Walk through the HashMap approach step by step. Mention you're trading space for time. Ask about duplicate handling.

06

Validate Parentheses

Why They Ask This

Tests stack data structure knowledge. Very relevant to SDET work since you often parse logs, JSON, or code. Shows you understand LIFO principles and can handle nested structures.

The Problem

Given a string containing just '(', ')', '{', '}', '[', ']', determine if the input string is valid.

The Solution

```
// JavaScript Solutionfunction isValid(s) {  const stack = [];
const map = {    ')': '(',    '}': '{',    ']': '['  };    for
(const char of s) {    if (char in map) {      // Closing
bracket      if (stack.pop() !== map[char]) {        return
false;      }    } else {      // Opening bracket
stack.push(char);    }  }  return stack.length === 0;}// Test
cases:// isValid("()[]{}") → true// isValid("([)]") → false//
isValid("{[]}") → true// isValid("(") → false
```

Interview Tip: Draw the stack state as you walk through an example. It shows clear thinking and communication skills.

07

Merge Sorted Arrays

Why They Ask This

Tests understanding of pointers and in-place operations. Common in SDET interviews because merging test results or log files is a real task. Shows you can think about memory efficiency.

The Problem

Merge two sorted arrays into one sorted array. Bonus: do it in-place if the first array has enough space.

The Solution

```
// JavaScript Solution - New arrayfunction mergeSorted(arr1, arr2) {  const result = [];  let i = 0, j = 0;    while (i < arr1.length && j < arr2.length) {    if (arr1[i] <= arr2[j]) { result.push(arr1[i++]);    } else { result.push(arr2[j++]);    }  }    // Add remaining elements while (i < arr1.length) result.push(arr1[i++]);  while (j < arr2.length) result.push(arr2[j++]);    return result;}// Test cases:// mergeSorted([1,3,5], [2,4,6]) → [1,2,3,4,5,6]// mergeSorted([1], [2,3,4]) → [1,2,3,4]// mergeSorted([], [1,2]) → [1,2]
```

Interview Tip: For in-place, start from the end to avoid overwriting. Mention this optimization even if not asked.

Your interview is coming.

Are you ready?

Save this. Practice one problem a day.
By next week, you'll have all 7 memorized.

Follow for more SDET interview prep.

Aston Cook
Senior QA Automation Engineer @ Resilience