Top 50 Intermediate JavaScript Technical Interview Questions

Array Manipulation & Logic

1. **Find the second largest number in an array without sorting**
   - Test: `[3, 1, 4, 1, 5, 9, 2, 6]` → Expected: `6`
2. **Remove duplicates from an array and return unique elements**
   - Test: `[1, 2, 2, 3, 4, 4, 5]` → Expected: `[1, 2, 3, 4, 5]`
3. **Find the intersection of two arrays**
   - Test: `[1, 2, 3, 4]` and `[3, 4, 5, 6]` → Expected: `[3, 4]`
4. **Rotate an array to the right by k positions**
   - Test: `[1, 2, 3, 4, 5]`, k=2 → Expected: `[4, 5, 1, 2, 3]`
5. **Find all pairs in an array that sum to a target value**
   - Test: `[2, 7, 11, 15]`, target=9 → Expected: `[[2, 7]]`

String Processing

6. **Check if a string is a palindrome (ignore spaces and case)**
   - Test: `"A man a plan a canal Panama"` → Expected: `true`
7. **Find the longest substring without repeating characters**
   - Test: `"abcabcbb"` → Expected: `"abc"` (length 3)
8. **Implement a function to reverse words in a sentence**
   - Test: `"hello world javascript"` → Expected: `"javascript world hello"`
9. **Count the frequency of each character in a string**
   - Test: `"hello"` → Expected: `{h: 1, e: 1, l: 2, o: 1}`
10. **Check if two strings are anagrams**
    - Test: `"listen"` and `"silent"` → Expected: `true`

Object Manipulation

11. **Deep clone an object (handle nested objects and arrays)**
    - Test: `{a: 1, b: {c: 2, d: [3, 4]}}` → Expected: independent copy
12. **Merge two objects recursively**
    - Test: `{a: 1, b: {c: 2}}` and `{b: {d: 3}, e: 4}` → Expected: `{a: 1, b: {c: 2, d: 3}, e: 4}`
13. **Flatten a nested object**
    - Test: `{a: {b: {c: 1}}}` → Expected: `{"a.b.c": 1}`
14. **Group array of objects by a property**

- Test: `[{name: "John", age: 25}, {name: "Jane", age: 25}]` → Group by age

15. **Find the path to a value in a nested object**
    - Test: Find path to value `3` in `{a: {b: {c: 3}}}` → Expected: `"a.b.c"`

Closures & Scope

16. **Create a counter function using closures**
    - Should return a function that increments and returns count each time called
17. **Implement a function that creates multiple counters**
    - Each counter should maintain its own state independently
18. **Fix the classic loop closure problem**
    - Make `for(var i=0; i<5; i++) setTimeout(() => console.log(i), 100)` print 0,1,2,3,4
19. **Create a memoization function using closures**
    - Cache function results to avoid redundant calculations
20. **Implement a once function (function that runs only once)**
    - Should return the result of first call for subsequent calls

Asynchronous JavaScript

21. **Implement Promise.all from scratch**
    - Should resolve when all promises resolve, reject when any rejects
22. **Create a delay function using Promises**
    - `delay(1000).then(() => console.log("After 1 second"))`
23. **Implement a retry mechanism for failed async operations**
    - Retry a function n times with delay between attempts
24. **Convert callback-based function to Promise-based**
    - Transform `setTimeout` callback style to Promise style
25. **Implement Promise.race from scratch**
    - Should resolve/reject with the first settled promise

Function Programming

26. **Implement map, filter, and reduce from scratch**
    - Don't use built-in array methods
27. **Create a pipe function for function composition**
    - `pipe(add1, multiply2, subtract3)(5)` should work left to right
28. **Implement a curry function**
    - Transform `f(a, b, c)` into `f(a)(b)(c)`

29. **Create a debounce function**
    - Delay function execution until after specified delay since last call
30. **Implement a throttle function**
    - Limit function execution to once per specified time period

Advanced Logic Problems

31. **Find missing number in array of consecutive integers**
    - Test: `[1, 2, 4, 5, 6]` → Expected: `3`
32. **Check if parentheses are balanced in a string**
    - Test: `"((()))"` → `true`, `"(()"` → `false`
33. **Implement binary search on a sorted array**
    - Return index of target element or -1 if not found
34. **Find the majority element in an array**
    - Element that appears more than n/2 times
35. **Implement a LRU (Least Recently Used) cache**
    - Fixed size cache that removes least recently used items

DOM & Browser Concepts

36. **Implement event delegation**
    - Handle clicks on dynamically added elements using single parent listener
37. **Create a function to get all ancestors of a DOM element**
    - Return array of all parent elements up to document
38. **Implement a simple virtual DOM diffing algorithm**
    - Compare two virtual DOM trees and return differences
39. **Create a function to serialize DOM to JSON**
    - Convert DOM tree to JSON representation
40. **Implement infinite scroll with performance optimization**
    - Load more content when user scrolls near bottom

Data Structures

41. **Implement a Stack class with push, pop, peek, and isEmpty methods**
    - Use array or linked list as underlying structure
42. **Create a Queue class with enqueue, dequeue, and front methods**
    - Implement using arrays or objects
43. **Implement a simple Hash Table/Map**

- Handle collisions and provide get/set/delete methods

44. **Create a Binary Tree class with insert, search, and traversal methods**
    - Implement in-order, pre-order, and post-order traversals
45. **Implement a Graph class with addVertex, addEdge, and BFS/DFS**
    - Support both directed and undirected graphs

ES6+ Features & Patterns

46. **Create a class hierarchy using ES6 classes and inheritance**
    - Implement proper constructor chaining and method overriding
47. **Use Proxy to create a reactive object**
    - Log property access and modifications
48. **Implement async/await for sequential and parallel execution**
    - Show difference between sequential and parallel async operations
49. **Create a generator function for infinite sequences**
    - Generate Fibonacci sequence using generators
50. **Use WeakMap and WeakSet for memory-efficient solutions**
    - Implement a solution that benefits from weak references

---

Tips for Approaching These Questions:

Problem-Solving Strategy:

1. **Understand the problem** - Read carefully and ask clarifying questions
2. **Think out loud** - Explain your approach before coding
3. **Start with a brute force solution** - Get something working first
4. **Optimize if needed** - Consider time and space complexity
5. **Test your solution** - Walk through with examples

Key Areas to Focus On:

- **Time & Space Complexity** - Analyze and optimize your solutions
- **Edge Cases** - Consider empty inputs, null values, boundary conditions
- **Code Quality** - Write clean, readable, and maintainable code
- **Multiple Approaches** - Know different ways to solve the same problem

Common Patterns:

- Two pointers technique

- Sliding window
- Hash maps for O(1) lookups
- Recursion with memoization
- Divide and conquer
- Dynamic programming basics

Remember: The goal isn't just to solve these problems, but to demonstrate your thinking process, coding style, and ability to communicate technical concepts clearly.