2. Implement Memoization in JavaScript for Optimized Recursive Functions

Why: Recursive functions like Fibonacci can be computationally expensive due to repeated calculations. Memoization caches results, drastically reducing runtime in algorithms, which is essential for performance in web apps handling complex computations without lagging user experience.

How to Implement: Use a cache object (e.g., a Map or object) to store computed values. Wrap your recursive function in a higher-order function that checks the cache before recursing. Apply this in scenarios like dynamic programming or tree traversals.

Javascript

```
// Without memoization
function fib(n) {
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}

// Hack: With memoization
function memoizedFib() {
  const cache = {};
  return function fib(n) {
    if (n in cache) return cache[n];
    if (n <= 1) return n;
    cache[n] = fib(n - 1) + fib(n - 2);
    return cache[n];
  };
}
const fibFast = memoizedFib();
console.log(fibFast(10));  // Output: 55 (much faster for larger n)
```

Analysis: For fib(40), non-memoized takes exponential time (billions of operations), while memoized is linear (O(n) time, O(n) space). This hack prevents stack overflows and improves app responsiveness, proven in benchmarks where it cuts execution time by orders of magnitude.