1. **Asynchronous JavaScript**
   * **Callbacks**: Understanding callback functions.
   * **Promises**: Creating and handling promises (.then(), .catch()).
   * **Async/Await**: Syntactic sugar over promises for cleaner async code.
   * **Event Loop**: Understanding how JavaScript handles asynchronous tasks.
2. **DOM Manipulation**
   * **Selecting Elements**: getElementById(), querySelector(), querySelectorAll().
   * **Event Handling**: Adding event listeners, understanding event propagation (bubbling and capturing).
   * **Creating/Modifying DOM elements**: createElement(), appendChild(), innerHTML, textContent.
3. **Object-Oriented Programming (OOP) in JavaScript**
   * **Classes**: Basic class structure, methods, and constructors.
   * **Inheritance**: extends, super() method, and constructor chaining.
   * **Prototypes**: Understanding prototype-based inheritance.
   * **Encapsulation**: Public and private members.
   * **Polymorphism and Overloading**.
4. **Modules and Namespaces**
   * **Import/Export**: Using ES6 modules to break code into smaller pieces.
   * **Module Bundlers**: Introduction to tools like Webpack (Optional, but useful for modern JS).
5. **Working with APIs**
   * **Fetching Data**: fetch() API, working with JSON data.
   * **Handling Errors**: Proper error handling when dealing with APIs.

**Resources**:

* JavaScript.info (<https://javascript.info/>)
* MDN Web Docs for async concepts (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function>)

**Phase 3: Data Structures & Algorithms (3-4 weeks)**

1. **Basic Data Structures**
   * **Arrays & Strings**: Advanced manipulation, slicing, joining, merging, reversing, etc.
   * **Stacks & Queues**: Implementing stacks/queues and using them in algorithms.
   * **Linked Lists**: Singly and doubly linked lists, reversal, insertion, and deletion.
   * **Hash Tables/Maps**: Inserting, deleting, and retrieving values in constant time.
2. **Sorting & Searching Algorithms**
   * **Sorting Algorithms**:
     + Bubble Sort, Insertion Sort, Merge Sort, Quick Sort.
     + Understand their time complexities.
   * **Searching Algorithms**:
     + Linear Search, Binary Search (for sorted arrays).
3. **Recursion**
   * **Understanding Recursion**: Base case, recursive case.
   * **Solving problems recursively**: Factorial, Fibonacci, Tower of Hanoi.
4. **Advanced Data Structures**
   * **Trees**: Binary Trees, Binary Search Trees, and Tree Traversals (Pre-order, In-order, Post-order).
   * **Graphs**: Introduction to graphs, graph representations (adjacency matrix/list), BFS and DFS.
   * **Heaps**: Basic understanding of Min and Max Heaps.
5. **Dynamic Programming & Greedy Algorithms**
   * **Dynamic Programming**: Memoization, tabulation, and solving problems like Fibonacci, Longest Subsequence.
   * **Greedy Algorithms**: Solving problems like coin change, activity selection.

**Resources**:

* LeetCode (JavaScript practice problems)
* GeeksforGeeks (Data Structures & Algorithms in JavaScript)
* HackerRank (Algorithms section)

**Phase 4: Advanced Topics & Practice (2-3 weeks)**

1. **Bit Manipulation**
   * Common bit tricks: Check if a number is even/odd, swapping values using XOR, counting set bits.
2. **Graph Algorithms**
   * **Graph Traversal**: BFS, DFS.
   * **Shortest Path Algorithms**: Dijkstra, A\*.
   * **Cycle Detection** in graphs.
3. **Practice Competitive Programming**
   * Solve coding challenges and problems from platforms like:
     + LeetCode (JavaScript problems).
     + HackerRank (JavaScript section).
     + Codeforces (Focus on algorithmic problem-solving).
4. **Time and Space Complexity**
   * Understand Big-O notation.
   * Analyze the efficiency of your code.
5. **Mock Tests**
   * Take mock coding tests (Infosys-specific, or general competitive programming tests) to practice within time limits.

**Resources**:

* LeetCode: JavaScript problems.
* Codeforces: Competitive programming practice.

**Phase 5: Mock Tests & Review (1-2 weeks)**

1. **Mock Coding Tests**
   * Focus on coding under time pressure (Infosys Power Programmer-style tests).
   * Practice solving a mix of algorithmic and data structure problems.
2. **Final Review**
   * Go back and review topics you find challenging or confusing.
   * Take short quizzes or tests on specific topics (JavaScript fundamentals, algorithms, data structures).