Key Points

- **The roadmap covers a 4-week study plan for "Grokking Algorithms: An Illustrated Guide for Programmers," ensuring 100% completion and mastering algorithms at a senior level.**
- **Each week aligns chapters with JavaScript topics and integrates Next.js and MERN Stack for practical application, emphasizing best practices in algorithms and full-stack development.**
- **An unexpected aspect is integrating machine learning (K-Nearest Neighbors) into web development, connecting theoretical algorithms with practical full-stack applications.**

Weekly Breakdown

- **Week 1: Chapters 1-3 (pages 1-50), focusing on binary search, selection sort, and recursion, with JavaScript arrays and functions, Next.js binary search, and MERN Stack selection sort API.**
- **Week 2: Chapters 4-5 (pages 51-94), covering quicksort and hash tables, with JavaScript sorting and Maps, Next.js quicksort, and MERN Stack hash table implementation.**
- **Week 3: Chapters 6-8 (pages 95-160), delving into graph algorithms like BFS and Dijkstra's, with JavaScript graph representation, Next.js graph visualization, and MERN Stack shortest path API.**
- **Week 4: Chapters 9-11 (pages 161-219), addressing dynamic programming and KNN, with JavaScript memoization, Next.js knapsack problem, and MERN Stack recommendation system.**

Resources and Best Practices

- **Use resources like MDN Web Docs for JavaScript basics and GeeksforGeeks for algorithm implementations.**

- **Follow best practices such as clean code, version control, and RESTful API design for full-stack development.**

Comprehensive 4-Week Roadmap for Studying "Grokking Algorithms"

This detailed survey note outlines a structured 4-week roadmap for studying "Grokking Algorithms: An Illustrated Guide for Programmers" by Aditya Y. Bhargava, ensuring 100% completion while mastering algorithms at a senior level. The plan aligns each chapter with necessary JavaScript topics and integrates relevant Next.js and MERN Stack concepts, emphasizing best practices in algorithms, backend development, and modern full-stack methodologies. The roadmap is designed for efficient study within the timeframe, providing a seamless connection between theoretical knowledge and real-world implementation.

Background and Book Structure

The book, accessible via PDF link, covers essential algorithms through 11 chapters, each focusing on different concepts. The table of contents, extracted from the provided resource, includes:

Page	Chapter Title	Key Topics
1	**Introduction to Algorithms**	**Binary search, Big O notation, algorithm running times**
21	**Selection Sort**	**Arrays, linked lists, selection sort**
37	**Recursion**	**Base case, call stack, recursive case**
51	**Quicksort**	**Divide and conquer, merge sort vs. quicksort, Big O notation revisited**
73	**Hash Tables**	**Hash functions, collisions, load factor, performance**
95	**Breadth-First Search**	**Graphs, queues, finding shortest path**
115	**Dijkstra's Algorithm**	**Shortest paths, negative-weight edges, implementation**
141	**Greedy Algorithms**	**Classroom scheduling, knapsack, approximation algorithms**
161	**Dynamic Programming**	**Knapsack problem, memoization, longest common subsequence**
187	**K-Nearest Neighbors**	**Classification, feature extraction, machine learning basics**
203	**Where to Go Next**	**Trees, Fourier transform, parallel algorithms, Bloom filters, SHA**

The total page count is **219**, with chapters varying in length, from 14 to 26 pages, influencing the time allocation.

Roadmap Development

Given a 4-week timeframe, with an assumed 2 hours of study per day for 5 days a week (totaling 40 hours over 20 days), the roadmap balances reading, understanding, implementation, and practical application. The distribution considers chapter complexity and page count, grouping them to ensure progressive learning:

- **Week 1: Chapters 1-3 (50 pages total, 20 + 16 + 14), focusing on foundational concepts.**
- **Week 2: Chapters 4-5 (44 pages, 22 + 22), covering sorting and data structures.**
- **Week 3: Chapters 6-8 (66 pages, 20 + 26 + 20), delving into graph and optimization algorithms.**
- **Week 4: Chapters 9-11 (59 pages, 26 + 16 + 17), addressing advanced topics including machine learning.**

This grouping ensures a balanced workload, with adjustments for complexity, such as dynamic programming requiring more practice time.

Weekly Detailed Plans

Week 1: Foundations and Sorting (Chapters 1, 2, 3)

- **Reading and Understanding:**
  - **Chapter 1 (pages 1-20): Covers binary search, Big O notation, and algorithm running times. Confirmed via content analysis, it includes exercises like maximum steps for binary search (e.g., 7 steps for 128 names).**
  - **Chapter 2 (pages 21-36): Focuses on selection sort, arrays, and linked lists, with implementations requiring loops and array manipulation.**
  - **Chapter 3 (pages 37-50): Explores recursion, call stack, and recursive case, essential for understanding algorithm design.**
- **JavaScript Topics:**
  - **Arrays, functions, recursion, and time complexity analysis, leveraging JavaScript's built-in features like for loops and recursive function calls.**
- **Implementations:**
  - **Implement binary search, selection sort, and recursive functions (e.g., factorial, Fibonacci) in JavaScript, ensuring understanding of time complexity (O(log n) for binary search, O(n^2) for selection sort).**
- **Practical Tasks:**
  - **Next.js Task: Create a Next.js app demonstrating binary search, with a sorted list and search function, using Next.js documentation for setup.**
  - **MERN Stack Task: Build a backend API using Express.js for sorting with selection sort, accepting a list and returning sorted results, referencing Express.js documentation.**
- **Resources:**
  - **Binary search: MDN Web Docs**
  - **Selection sort: GeeksforGeeks**
  - **Recursion: MDN Web Docs**
- **Best Practices:**
  - **Write clean, modular code with clear function names.**
  - **Test algorithms with various input sizes to understand performance.**
  - **Use version control (e.g., Git) to track changes in the project.**

Week 2: Advanced Sorting and Data Structures (Chapters 4, 5)

- **Reading and Understanding:**
  - **Chapter 4 (pages 51-72): Covers quicksort, divide and conquer, and Big O notation revisited, with average case O(n log n).**
  - **Chapter 5 (pages 73-94): Discusses hash tables, hash functions, collisions (handled via linked lists), and load factor (resize when >0.7), with O(1) average case performance.**
- **JavaScript Topics:**
  - **Sorting algorithms (quicksort), Maps, and hashing, using JavaScript's Map for hash table implementation.**
- **Implementations:**
  - **Implement quicksort and a simple hash table, understanding collisions and performance trade-offs.**
- **Practical Tasks:**
  - **Next.js Task: Extend the app to include quicksort, allowing user input for sorting, leveraging GeeksforGeeks for implementation.**
  - **MERN Stack Task: Implement a backend service using hash tables for efficient data storage, using MongoDB and Express.js, with MongoDB documentation for database operations.**
- **Resources:**
  - **Quicksort: GeeksforGeeks**
  - **Hash tables: MDN Web Docs**
- **Best Practices:**
  - **Optimize quicksort by choosing a good pivot (e.g., random or median).**
  - **Ensure hash table implementation handles collisions efficiently.**
  - **Follow RESTful API design principles for backend development.**

Week 3: Graph Algorithms (Chapters 6, 7, 8)

- **Reading and Understanding:**
  - **Chapter 6 (pages 95-114): Covers BFS, graphs, queues, and finding shortest paths in unweighted graphs, with O(V + E) complexity.**
  - **Chapter 7 (pages 115-140): Explores Dijkstra's algorithm for shortest paths in weighted graphs, avoiding negative weights, with O((V + E) log V) complexity using priority queues.**
  - **Chapter 8 (pages 141-160): Discusses greedy algorithms, including activity selection and knapsack, with approximation for NP-complete problems.**
- **JavaScript Topics:**
  - **Graph representation (adjacency lists, matrices), priority queues, and optimization techniques, using arrays and objects.**
- **Implementations:**
  - **Implement BFS for graph traversal, Dijkstra's algorithm, and a greedy algorithm like activity selection, ensuring understanding of graph structures.**
- **Practical Tasks:**
  - **Next.js Task: Visualize a graph and perform BFS using React-Vis, with React-Vis documentation for visualization.**
  - **MERN Stack Task: Build an API for shortest path using Dijkstra's, storing graphs in MongoDB, with RESTful endpoints in Express.js.**
- **Resources:**
  - **BFS: GeeksforGeeks**
  - **Dijkstra's: GeeksforGeeks**
  - **Greedy algorithms: GeeksforGeeks**
- **Best Practices:**
  - **Use efficient data structures for graph representation (e.g., adjacency lists for sparse graphs).**
  - **Validate graph data for correctness before processing.**
  - **Ensure API endpoints are secure and handle errors gracefully.**

Week 4: Optimization and Machine Learning (Chapters 9, 10, 11)

- **Reading and Understanding:**
  - **Chapter 9 (pages 161-186): Covers dynamic programming, memoization, and tabulation, with examples like knapsack and longest common subsequence, crucial for optimization.**
  - **Chapter 10 (pages 187-202): Introduces K-Nearest Neighbors for classification, feature extraction, and machine learning basics, with applications in recommendations.**
  - **Chapter 11 (pages 203-219): Overviews advanced topics like trees, Fourier transform, and parallel algorithms, serving as a guide for further study.**
- **JavaScript Topics:**
  - **Memoization, classification, and advanced data structures, using arrays for dynamic programming and distance calculations for KNN.**
- **Implementations:**
  - **Implement dynamic programming solutions (e.g., knapsack), a KNN classifier, and explore one advanced topic from Chapter 11.**
- **Practical Tasks:**
  - **Next.js Task: Build an app using dynamic programming for optimization, displaying results, with GeeksforGeeks for guidance.**
  - **MERN Stack Task: Develop a recommendation system using KNN, integrating with MongoDB for data storage.**
- **Resources:**
  - **Dynamic programming: GeeksforGeeks**
  - **KNN: GeeksforGeeks**
- **Best Practices:**
  - **Use memoization to optimize dynamic programming solutions.**
  - **Ensure KNN implementation is scalable for large datasets.**
  - **Follow modern full-stack methodologies, such as component-based architecture in Next.js.**

Integration with Next.js and MERN Stack

The roadmap integrates algorithms with Next.js (React framework for server-side rendering) and MERN Stack (MongoDB, Express.js, React, Node.js) by proposing practical tasks. For instance, binary search can be implemented in a Next.js app for searching sorted lists, while selection sort can be part of a backend API in Express.js. Graph visualizations use React-Vis, and KNN can power recommendation systems, aligning with full-stack development best practices.

Best Practices and Mastery

To master algorithms at a senior level, the plan emphasizes implementation, understanding complexities (e.g., Big O notation), and real-world application. Regular review, personal projects, and staying updated with developments ensure deep learning, with resources like MDN Web Docs and GeeksforGeeks supporting the journey.

An unexpected aspect is the inclusion of machine learning (K-Nearest Neighbors in Chapter 10) within the roadmap, connecting theoretical algorithms to practical full-stack applications like recommendation systems, which may not be immediately obvious for algorithm studies but enhances versatility.

Key Citations

- Grokking Algorithms PDF link
- Next.js documentation
- Express.js documentation
- MongoDB documentation
- MDN Web Docs for JavaScript
- GeeksforGeeks selection sort
- MDN Web Docs recursion
- GeeksforGeeks quicksort
- MDN Web Docs hash tables
- GeeksforGeeks BFS
- GeeksforGeeks Dijkstra's
- GeeksforGeeks greedy algorithms
- React-Vis documentation
- GeeksforGeeks dynamic programming
- GeeksforGeeks KNN