Data Structures and Algorithms by [**Elshad Karimov**](https://tcsglobal.udemy.com/user/elshad-karimov/)

Time Complexity

To prepare for Data Structures and Algorithms (DSA) interviews for FAANG companies (Facebook, Amazon, Apple, Netflix, Google), you'll need a solid understanding of key concepts and problem-solving techniques. Here's a structured list of topics that will help you prepare:

**1. Mathematics and Number Theory**

* **Time Complexity Analysis (Big O, Omega, Theta)**: Understanding algorithmic complexity is crucial.
* **Recursion**: Recursive problem solving and recursion trees.
* **Mathematical Induction**
* **Prime Numbers**: Sieve of Eratosthenes, prime factorization.
* **Greatest Common Divisor (GCD), Least Common Multiple (LCM)**: Euclidean algorithm.
* **Modular Arithmetic**: Used in many algorithmic problems, especially related to large numbers.
* **Combinatorics**: Permutations, combinations, and basic probability.

**2. Arrays and Strings**

* **Array Manipulation**: Searching, sorting, sliding window, two-pointer technique.
* **Subarrays**: Maximum subarray (Kadane’s algorithm), prefix sums.
* **String Matching Algorithms**: KMP algorithm, Rabin-Karp.
* **Anagram Problems**: String sorting, hashing for anagram detection.
* **Pattern Matching**: Regular expressions, substring search.
* **Substring problems**: Longest common substring, longest palindromic substring, etc.

**3. Linked Lists**

* **Singly and Doubly Linked Lists**: Insertion, deletion, and traversal.
* **Reverse a Linked List**: Iterative and recursive solutions.
* **Detecting Cycles**: Floyd’s Tortoise and Hare algorithm.
* **Intersection of Linked Lists**: Using hashmaps, two-pointer technique.
* **Flattening Linked Lists**: Flattening multi-level linked lists.

**4. Stacks and Queues**

* **Basic Operations**: Push, pop, peek, and size.
* **Stacks**: Use of stacks for balanced parentheses, reversing expressions.
* **Queues**: FIFO implementation, circular queues.
* **Priority Queue**: Min-heap, max-heap, and heap operations.
* **Implementations using Stacks**: Queue using stacks, or vice versa.
* **Deque**: Double-ended queue operations.
* **Sliding Window**: Using queues/stacks for sliding window problems.

**5. Hashing**

* **Hash Tables**: Hash map, hash set, handling collisions (chaining, open addressing).
* **Hash Functions**: Designing good hash functions.
* **Applications**: Anagram detection, counting frequencies, finding duplicates, etc.
* **LRU Cache**: Using hash maps and doubly linked lists.

**6. Trees**

* **Binary Trees**: Traversal techniques (preorder, inorder, postorder).
* **Binary Search Trees (BST)**: Insertion, deletion, search.
* **Balanced Trees**: AVL Trees, Red-Black Trees.
* **Heaps**: Min-heap, max-heap, heapify operations.
* **Trie**: Prefix trees for storing strings, autocomplete, and spell check.
* **Segment Trees & Fenwick Trees (Binary Indexed Trees)**: Range queries, range updates.
* **Lowest Common Ancestor (LCA)**: Binary Lifting, Tarjan’s algorithm.

**7. Graphs**

* **Graph Representation**: Adjacency list, adjacency matrix, edge list.
* **Graph Traversal**: Depth-First Search (DFS), Breadth-First Search (BFS).
* **Topological Sorting**: For Directed Acyclic Graphs (DAGs).
* **Shortest Path Algorithms**: Dijkstra's algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm.
* **Minimum Spanning Tree (MST)**: Kruskal’s algorithm, Prim’s algorithm.
* **Graph Connectivity**: Connected components, Union-Find / Disjoint Set Union (DSU).
* **Cycle Detection**: In directed and undirected graphs.
* **Bipartite Graphs**: BFS/DFS for checking bipartiteness.
* **Strongly Connected Components (SCC)**: Kosaraju's algorithm, Tarjan’s algorithm.

**8. Dynamic Programming**

* **Basic DP**: Fibonacci series, climbing stairs, coin change problem.
* **Optimal Substructure**: Solving problems by breaking them down into overlapping subproblems.
* **Memoization vs Tabulation**: Recursive vs iterative dynamic programming.
* **Knapsack Problem**: 0/1 knapsack, unbounded knapsack.
* **Longest Common Subsequence (LCS)**, **Longest Increasing Subsequence (LIS)**.
* **Matrix Chain Multiplication**.
* **DP on Trees**: Tree DP (e.g., maximum path sum in a binary tree).
* **DP for Subset Problems**: Subset sum, partition problem.

**9. Backtracking**

* **Backtracking Basics**: Solving problems like N-Queens, Sudoku, permutation generation, combination generation.
* **Subset and Combination Generation**: Using recursion to generate subsets/combinations.
* **Pruning**: Efficient backtracking by eliminating unpromising paths early.

**10. Greedy Algorithms**

* **Greedy Strategy**: Local optimal choice leading to a global optimum.
* **Activity Selection Problem**.
* **Fractional Knapsack**: Greedy approach to solving knapsack problem.
* **Huffman Coding**: Greedy for optimal prefix encoding.
* **Job Sequencing Problem**.
* **Interval Scheduling**.

**11. Divide and Conquer**

* **Basic Strategy**: Dividing the problem into subproblems, solving recursively, and combining results.
* **Merge Sort, Quick Sort**.
* **Closest Pair of Points**.
* **Matrix Multiplication**: Strassen's algorithm.
* **Binary Search**: Both iterative and recursive.
* **Majority Element**: Boyer-Moore Voting Algorithm.

**12. Bit Manipulation**

* **Basic Operations**: AND, OR, XOR, NOT, shifting (left, right).
* **Counting Set Bits**: Brian Kernighan’s algorithm.
* **Bit Masking**: Setting, clearing, and toggling bits.
* **Find Unique Element**: Using XOR for detecting unique elements in an array.
* **Power of Two**: Efficient checking using bitwise operations.

**13. String Algorithms**

* **Rabin-Karp Algorithm**: For substring search.
* **KMP (Knuth-Morris-Pratt)**: For efficient substring matching.
* **Z Algorithm**: Pattern matching in linear time.
* **Manacher’s Algorithm**: For finding the longest palindromic substring in linear time.

**14. Advanced Topics (Optional but Helpful)**

* **Suffix Arrays**: Efficient string matching.
* **Bloom Filters**: Space-efficient probabilistic data structure.
* **Data Structure Design**: Custom data structures, like a balanced BST, Trie, etc.
* **Heavy-Light Decomposition**: For path queries on trees.

**Resources for Preparation**

1. **Books**:
   * *Cracking the Coding Interview* by Gayle Laakmann McDowell.
   * *Elements of Programming Interviews* by Adnan Aziz.
   * *Introduction to Algorithms* by Cormen, Leiserson, Rivest, Stein (CLRS).
2. **Online Platforms**:
   * **LeetCode** (especially for FAANG-style questions).
   * **HackerRank**.
   * **CodeSignal**.
   * **Codeforces**.
   * **GeeksforGeeks**.
   * **InterviewBit**.
3. **Mock Interviews**:
   * Try platforms like *Pramp*, *interviewing.io* for mock interviews.
4. **YouTube Channels**:
   * *Tech Dummies*, *Abdul Bari*, *MyCodeSchool*, *Tushar Roy*.

**Tips for Preparation:**

1. **Practice consistently**: Aim for at least 2-3 problems a day.
2. **Master problem-solving techniques**: Learn patterns (like sliding window, two-pointer, dynamic programming on intervals, etc.) and apply them to different problems.
3. **Understand time complexity**: Always aim to optimize solutions and understand trade-offs between time and space complexity.
4. **Simulate interviews**: Solve problems in a timed environment to replicate interview conditions.
5. **Mock interviews**: Get feedback from others, especially in peer-to-peer interviews.

By thoroughly covering these topics, you’ll be well-equipped to tackle DSA interviews at FAANG companies.

How to learn DSA in Java? - <https://youtu.be/p-8YpVkG3aI>

25 Dec, 2024 – Today I will complete BigO Notation. I will complete it later.

BigO Notation

* + It is a metric we use to determine the efficiency of code.

25 Dec, 2024 – Today I will complete Arrays.

Array

* + An array is a DS that can store fixed size collection of similar elements with contiguous memory allocation.

A screenshot of a computer

Description automatically generated

2D Arrays are also stores as 1D i.e. linearly in memory as below:

A screenshot of a computer program

Description automatically generatedA computer screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

Insert operation is O(1) time complexity.

Accessing an element is also O(1) time complexity.

Traversing an Array require O(N) time complexity.

A screenshot of a computer program

Description automatically generated

Searching is also takes O(1) TC.

A screenshot of a computer program

Description automatically generated

Deleting an element from array is, setting that index to Integer.MIN\_VALUE; and it takes O(1) time complexity.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Creation of 2D Array

A screenshot of a computer

Description automatically generated

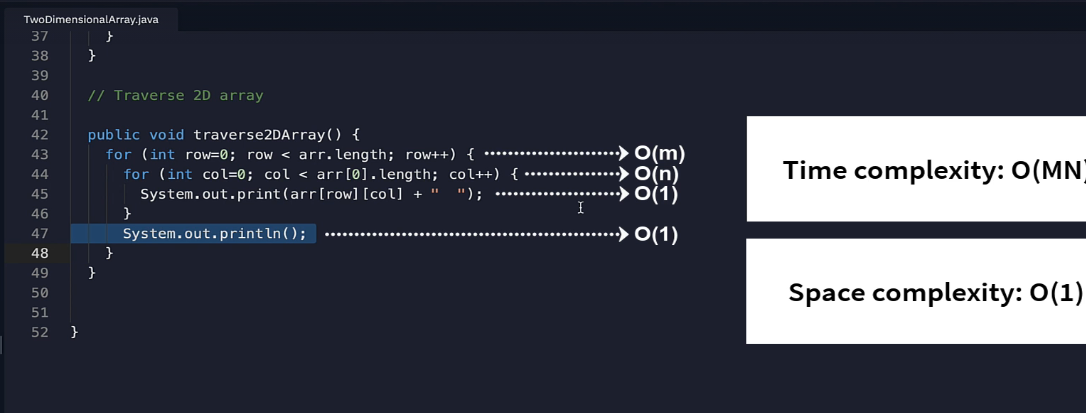
Insert operation is O(1) time complexity.

A screenshot of a computer program

Description automatically generated

Accessing an element takes O(1) time complexity.

Traversing a 2D array has O(mn) time complexity.



Searching in 2D array has O(mn) time complexity.

A screenshot of a computer program

Description automatically generated

Deleting a value has O(1) time complexity.

A blue and white rectangular object with text

Description automatically generated with medium confidence