DSA CheatSheet

HashSet

HashMap

Binary Search

Two Pointers or Iterators

Sliding Window

Prefix Sum

Bit Manipulation

Intervals

Matrix

ArrayList

Stack

Heaps (Level 3)

BFS/DFS (Level 3)
    Trees

Graphs (Level 3)

Sorting

Dynamic Programming (Level 3)

Recursion

Topological Sort (Level 3)

Greedy (Level 3)

Backtracking (Level 3)

References

HashSet

- When you have to find a **unique element** or check if the set of elements have **duplicates**
    - Contains Duplicates

- When you have to keep track of a **unique set of elements**
    - Happy Number
- To **achieve faster lookups or O(1) time complexity**.

HashMap

- When you have to find the **frequency** or **count of occurrences**
    - First Unique Integer, Majority Element, Word Frequencies
- When you have to **store the mapping of keys to their value**
    - Two Sum In Unsorted Array, Three Sum
- To **achieve faster lookups or O(1) time complexity**.
- **Anagram/Permutation/Palindrome** - which you can solve by keeping track of the number of occurrences of characters
    - Valid Anagram, Find All Anagrams In a String, Permutation Palindrome

Binary Search

- When you are given a **sorted array**
    - Search in a rotated sorted array, Find Minimum in Rotated Sorted Array
- To perform **search** operation in **O(log N)** time complexity
    - Find Peak Element, Find First and Last Position of Element in Sorted Array
- Problems related to “Maximize the minimum” or “Minimize the maximum” something is usually solved using **Binary Search(Important).**
    - Koko Eating Bananas, Capacity to ship packages within D Days, Find Smallest divisor given a threshold

Two Pointers or Iterators

- In Two Pointers - the set of pointers move in either of the pattern given below
    - One pointer starts from the **beginning** while the other pointer starts from the **end**
        - Container with Most Water, Squares of Sorted Array, Rotate Array
    - One will be a **slow-runner** and the other **fast-runner** also known as the **Hare & Tortoise algorithm**. This approach is quite useful when dealing with **cyclic linked lists** or arrays.
        - Detect LinkedList Cycle, Remove Nth Node from the end of the list, Middle of the LinkedList

- In scenarios when you have to track two variables or nodes in a linear data structure such as **LinkedList, Arrays and ArrayList** it is more commonly used. It is often useful when searching for pairs.
    - Remove Duplicates from Sorted Array, Move Zeroes
- Two Pointers is especially helpful to solve problems **Inplace or O(1) Space complexity**.
    - Reverse a Linked List, Dutch National Flag Problem or Sort Colours

Sliding Window

- **The Sliding Window** technique is used when we're supposed to consider all **subarray/substrings** with some particular constraint such that it is possible to increment/decrement the window size in each iteration.
    - Longest Substring without repeating characters, Longest Substring with at most K Distinct Character, Length of the Longest substring without repeating characters
- It's also implemented by using **TwoPointers** which act as a **window** that is constant in size or grows or shrinks with respect to the problem you are solving.
    - Minimum Window Substring, Fruits into Basket

Prefix Sum

- Whenever the **value at a particular position of an array** depends on all **previous elements** or all the next elements, we can either use prefix sum or suffix sum.
    - Product of Array, Find Pivot Index, Minimum size subarray sum
- As values are precomputed it reduces the time complexity to **O(n).**
- When the **subarray sum** is to be **equal to a given value**
    - Zero Sum Subarrays, Prefix Sum Array applications

Bit Manipulation

- XOR of **A ^ A = 0** and **A ^ 0 = A.** Thus used to find **unique elements**.
    - Single Number, Missing Numbers, Single Number - 2
- To remove the last set bit **- A&(A-1)**
    - Number of 1 bits or Hamming Weight, Counting Bits

- To find Union of A and B - or operation **A | B** and to find the intersection of A and B - and operation **A & B**
    - Sum of two Numbers
- Bit Masking is the act of applying a **mask** to a value defining which **bits** you want to keep, and which **bits** you want to clear. **Masking**. **<<** to shift bits to **left** and **>> or >>>** to shift bits to **right**.
    - Reverse Bits

Intervals

- If you hear the term **overlapping intervals** or are asked to produce a list with mutually exclusive intervals**/non-overlapping intervals**.
    - Insert Interval, Non-overlapping intervals
- **Sorting** by **start time** or end **time makes** it easier to solve
    - Merge Intervals, Meeting Rooms
- **Heap** may be applicable to solve intervals
    - Minimum Meeting Rooms

Matrix

- Matrix problems are about understanding **boundary conditions** and the finding **pattern or logic** by trying out the **sample inputs**.
    - Set Matrix Zeroes, Spiral Matrix, Rotate Image, Diagonal Traverse

ArrayList

- When we need to **dynamically insert elements** (not fixed size). Otherwise same characteristics as an array
    - Find all numbers disappeared in an array, Intersection of two arrays, Minimum Index sum of two lists

Stack

- When you are asked to find the **next greater or next smaller** element then it can be done using a **stack** in O(n)
    - Largest Rectangle in Histogram, Daily Temperatures

- Last In First Out (**LIFO**) principle helps to solve **mathematical** expressions
    - Evaluate reverse polish notation, Valid Parentheses
- Any problem solved by **recursion** can be converted to an **iterative solution** using a **stack**. This helps in avoiding **stack overflow** problems as recursion uses an internal stack
    - Preorder Traversal, Inorder Traversal

Heaps (Level 3)

- When you have to find **Top K or K smallest/Largest elements**
    - Top k frequent elements, Reorganize String, Ugly Number - 2
- When to choose MinHeap vs MaxHeap (counter-intuitive)
    - When you want to keep track of **K smallest** elements, use **MaxHeap** (the parent node is always larger than the child node value).
        - Last stone weight
    - When you want to keep track of **K largest** elements, use **MinHeap** (the parent node always has a smaller value than the child nodes).
    - Kth largest element in an array, Kth largest element in a stream
- If you're asked to **sort an array** to find an exact element or **minimum number**
    - Sort Characters by Frequency, Minimum Meeting Rooms, Task Scheduler
- Whenever you're given '**K' sorted arrays**, you can use a Heap to efficiently perform a **sorted traversal of all the elements** of all array
    - Merge K Sorted Lists, K Pairs with Largest Sums
- **Two Heaps** - In some problems, where we are given a set of elements such that we can divide them into two parts. We are interested in knowing the **smallest element** in one part and the **biggest element** in the other part. As the name suggests, this technique uses a **Min-Heap** to find the smallest element and a **Max-Heap** to find the biggest element.
    - Find Median from data stream

BFS/DFS (Level 3)

Trees

- In **Trees,** When you have to traverse **level by level** - **BFS**

    - Binary Tree Level Order Traversal, Binary Tree ZigZag Level order Traversal, Minimum Depths of Binary tree, Path sum
- **In Trees,** When you have to traverse **Depth wise - DFS**
    - **In Order, Postorder and PreOrder** traversal
        - **The last element** of the array in **Post Order** traversal is the **root node**
            - Postorder Traversal, Construct Binary Tree from postorder and inorder
        - **The middle element** of the array in **In Order traversa**l is the **root node**. **In order traversal of a BST** gives values in **ascending order**
            - Validate Binary Tree, Inorder successor
            - Inorder Traversal,
        - **The first element** in **preorder** traversal is the **root node**
            - Preorder Traversal, Lowest Common Ancestor

Graphs (Level 3)

- In **Graphs/Grid**
    - Problems related to **min number of steps or shortest path** can sometimes be done by creating a **graph** and doing a **BFS** on it
        - Word Ladder or Minimum Knight Moves or Shortest path to get keys
    - When you need to **start BFS for multiple vertices** as well as calculate the **travel depth**
        - Rotten oranges, Walls and gates
    - When **Root to leaf** traversal is needed - **DFS**
        - Number of islands, Flood Fill**,** Clone graph
    - In both cases, we need to keep **track** of **visited cells**
        - Graph Valid Tree
- **DFS/BFS is not just limited to Graph/Grid** problems. They can be used in any problem where it is possible to **connect two entities** and some path is formed.
    - String Transformation, Word Ladder

Sorting

- When elements in an **orderly fashion** are required to solve the problem such as Binary Search or Intervals.

    - Chocolate Distribution, Minimum Meeting Rooms, Sort List

Dynamic Programming (Level 3)

- **DP** problems usually have a pattern of count number of distinct ways, max/min value. Maximum/minimum subarray/subset
    - Climbing Stairs, Maximum Product Subarray
- **DP** is used in most of the problems where some value is to be minimized/maximized.
    - Perfect Squares, Coin Change
- **Overlapping Subproblem** is the main characteristic of **Dynamic Programming**
    - Perfect Squares, House Robber, Coin Change
- Finding the **state variable**, **recurrence** relation and **base cases** are key steps in DP
    - Longest increasing subsequence, Longest common subsequence

Recursion

- Any problems which have **repetitive functions or subproblems** can be solved by recursion
    - Reverse String, All paths from source to target, Same tree
- If it is possible to draw a **recursion tree** that forms a **DAG**, then only that problem can be solved by **recursion**. Visualization
    - When you need to explore all combinations recursion is used
        - Perfect Squares or Permutations or All subarrays

Topological Sort (Level 3)

- “Topological sorting” using BFS or DFS only works with graphs that are **directed and acyclic**.
- It provides a linear sorting based on the required ordering between vertices in directed acyclic graphs. ie If you’re asked to update all **objects in sorted order** or If you have a class of objects that follow a particular order
    - Task Scheduling
- To be specific, given vertices `u` and `v`, to reach vertex `v`, we must have reached vertex `u` first. In “topological sorting”, `u` has to appear before `v` in the ordering. The most popular algorithm for “topological sorting” is **Kahn’s algorithm.**

    - Alien Dictionary
    - Minimum Height Trees

Greedy (Level 3)

- **Greedy** Approach - when we can make a choice of the maximum or minimum at any given point
    - Integer to Roman or Jump game or Best time to sell stock - maximize profit

Backtracking (Level 3)

- All Permutations/Subsets - **Backtracking**
    - Letter Combinations of Phone Number or Permutations or Subsets
- Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, which incrementally builds candidates to the solution and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot lead to a valid solution.
    - Generate Parentheses, Combination Sum and Word Search.

References

- Smarter way to prepare for coding interviews
- Sliding Window for beginners
- Binary Search for beginners
- Graph for beginners
- DP for beginners