Arrays

1. **Set Matrix Zero**

* Was able to do this while revising. Just maintain row flag and col flag
* Also keep a note of second for loop I and j
* Separate loops for row flag and col flag

1. **Pascal Triangle**

* Normal one is easy.
* What I learnt was how to calculate nCr. That is good.

1. **Next Permutation**

* Good Question. Requires repeated practice to be able to solve.
* Reverse from I onwards. Set j to i-1. And then check correct index for j, where it can be placed

1. **Kadane’s Algorithm**

* Good algo. I have now mastered it.

1. **Max Product Subarray**

* Based on prev algo + one more variable!

1. **Sort an array of 0’s 1’s 2’s**

* Algo takes time to be remembered, but is easy!

1. **Stock Buy and Sell**

* Is very simple. Do not confuse it with Stock Span which is slightly tougher.
* Just maintain 2 variables, maxProfit and minPrice.

1. **Rotate Matrix**

* Transpose + Swap columns from left to right

1. **Merge Overlapping Subintervals**

* My favourite Question.

1. **Merge 2 arrays without extra space (Leetcode)**

* Easy! Just start from end index

1. **Merge 2 arrays without extra space (GFG)**

* Similar to insert sort algo
* Tell gap method only if told to further optimize

1. **Find the duplicate in an array of N+1 integers.**

* Using Rabbit and Tortoise Algo
* Index tells ki kaunsa element kiss se linked hai

1. **Repeat and Missing Number**

* Use hashmap or count array
* Use concept of sum of naturanl numbers and sum of square of natural numbers

1. **Inversion of Array (Pre-req: Merge Sort)**

* Normal Merge sort. Only change when a[i] > a[j]. That is where we wil count!!

1. **Search in a 2d Matrix (Leetcode)**

* This one is pretty simple. Just think of the 2D matrix as a single 1D matrix of size NxM.
* Then after calculating middle, we just need to see how that will translate to row index or col index.

1. **Search in a 2d Matrix (GFG)**

* This is a particularly good question. Aditya Verma Covered this. We need to start from top right.
* Then we need to do i—or j++ depending on what is the value of cell and what is the value of the key.

1. **Pow(X,n)**

* Good good question.
* Had to know the approach once before coding. Take care of 3 edge cases we discussed.

1. **Majority Element (>N/2 times)**

* Moores Voting Algorithm

1. **Majority Element (>N/3 times)**

* Modified Mooores Voting Algoritm! (Needs to be practiced again and again to get intuition and code)

1. **Grid Unique Paths**

* Brute force is recursive, then dp
* Optimal soultingis using Combinatorics: (n+M-2)C(n-1 or m-1)

1. **Reverse Pairs (Leetcode)**

* Similar to inversion count
* But need to do again and again for clarity

1. **2-Sum-Problem**

* Is easy
* O(N^2), HashMap, sorting

1. **4-Sum-Problem**

* Sort the array
* Take two loops of I and J
* On remaining elements, take left and right pointer method
* Handle duplicates

1. **3-Sum-Problem**

* Pre Req is the above problem.

1. **Longest Consecutive Sequence**

* Good question. Need to know algo once to solve.
* Make use of unordered set and always start the check from the min element of the potential longest consecutive subsequence

1. **Check if subarray with 0 sum exists or not**

* This is just a subset of the next question

1. **Largest Subarray with 0 sum**

* Make use of an unordered map
* Maintain sum. If sum == 0 or a sum repeats, calculate and return the length.

1. **Count number of subarrays with given XOR K**

* Not at all easy
* But XOR properties once known, becomes easy to understand after regular revisions!

1. **Longest Substring without repeat**

* Good question
* Use two pointers left and right and use a set
* Using an unorderedmap of form char:lastIndex , further reduces the time complexity.

1. **Trapping rainwater**

* Using O(N) space I know. Its easy
* Using O(1) space and solving using left right pointer is something new and logic will become clearer with time and solving more and more questions.

1. **Remove Duplicate from Sorted array**

* Easy. Just make use of two pointers. One for iterating and one other for maintain the index where the next unique element needs to be inserted.

1. **Max consecutive ones**

* Very easy. Nothing needs to be done

1. **Spiral Matrix**

* Looks tough, but is fairly simple.
* Just maintain 4 variables, topRow, bottomRow, leftCol, rightCol and then go right,down,left and up or what is the clockwise order.

Linked List

1. **Reverse a LinkedList**  
   - Iteratively(using curr, prev and next pointers): My preferred approach  
   - Recursively (Slightly more involved when approaching it using only one var, i.e. Head). Using two var in function call, it’s easy.
2. **Find the middle of LinkedList**

* Using slow and fast pointer.

1. **Merge two sorted Linked List (use method used in mergeSort)**

* Do it in place and use head and tail pointers!

1. **Remove N-th node from back of LinkedList**

* Two pass approach is simple and easy.
* Single pass approach using fast and slow pointer.

1. **Add two numbers as LinkedList**  
   - Approach is easy. Practice striver code again and again!
2. **Delete a given Node when a node is given. (0(1) solution)**

* Stupid Q. Don’t give it much thought.

1. **Find intersection point of Y LinkedList**

* Hashing solution I know
* The 2 optimal solutions based on diff in length.

1. **Detect a cycle in Linked List**
2. **Find the starting point of the Loop of LinkedList**

* Both Q8 and Q9 use same algo. Its just at Q8 uses a part of rabbit and tortoise algo
* Algo is simple. Intuition behind it is important!  
  (Picture of algo intuition will be present as a separate pdf)

1. **Reverse a LinkedList in groups of size k.**

* Took 5 pointers. But my approach was same as striver. Is easy. Do not think ki tough hai.

1. **Check if a LinkedList is palindrome or not.**

* Find middle using fast and slow pointer. Then reversing 2nd half. Then checking first half and second half.

1. **Flattening of a LinkedList**

* Is simple. Just take 2 LL at a time, and apply merge function on it, which we have done earlier.
* Can be easily done iteratively.

1. **Rotate a LinkedList**

* Is simple. Can be done in O(N). Remember k = k%N and maintain tail and curr pointer

1. **Clone a Linked List with random and next pointer**

* Maintain an unordered map. Knowing this, pb is simple.

Greedy

1. **N meeting in one room + Activity Selection (it is the same as N meeting in one room)**
2. **Minimum number of platforms required for a railway.**
3. **Job sequencing Problem**
4. **Fractional Knapsack Problem**
5. **Greedy algorithm to find minimum number of coins. A edge case q of bigger DP Problem!**
6. **Dijkstra’s Algorithm. (Part of graphs, but it makes me realize that Dijkstra’s is easy)**

Recursion & Backtracking

Binary Search

1. **The N-th root of an integer**

* The concept is vey simple. Implementation can be slightly tricky the first time.
* Learnt that we could use e in C++.

1. **Matrix Median**

* Slightly more tough to be asked for an interview.

1. **Find the element that appears once in a sorted array, and the rest element appears twice (Binary search)**

* Learnt two great properties of XOR function.
* The answer lies in the indices

1. **Search element in a sorted and rotated array/ find pivot where it is rotated**

* Good Question. Solve using Aditya’s Approach

1. **Median of 2 sorted arrays**
2. **K-th element of two sorted arrays**
3. **Allocate Minimum Number of Pages**

* Looks tough. But once you do it
* It becomes simple!!

1. **Aggressive Cows**

* Once above is done, this becomes simple
* Is opposite to 7 in terms of what is to be returned.

Tries

1. **Implement Tries (Prefix Tree)**
2. **Implement Tries – 2 (Prefix Tree)**
3. **Longest String with All Prefixes**
4. **Number of Distinct Substrings in a String**
5. **Power Set (this is very important)**
6. **Maximum XOR of two numbers in an array**
7. **Maximum XOR With an Element from Array**

Stack & Queue

String

1. Reverse Words in a String
2. Longest Palindrome in a string
3. Roman Number to Integer and vice versa
4. Implement ATOI/STRSTR
5. Longest Common Prefix
6. Rabin Karp
7. Z-Function
8. KMP algo / LPS(pi) array
9. Minimum characters needed to be inserted in the beginning to make it palindromic
10. Check for Anagrams
11. Count and Say
12. Compare version numbers

Binary Tree & BST

1. **Inorder Traversal (Iteratively)**

* Recursive we know.
* Iterative using stack. Is easy
* Initially traverse from root to left most node and store in stack
* After that while stack is not empty, pop element one by one
* If popped element has right, push right and then go all the way left again.

1. **Preorder Traversal (iteratively)**

* Again pretty simple. Just use stack again.
* Initially push root.
* Then first push root->right (if exists), then push root->left(if exists)

1. **Postorder Traversal (iteratively)**

* Take 2 stacks
* Push initially root in s1
* Then pop from s1 and push it to s2
* Push left and then right of popped node in s1
* Repeat while s1 is not emtpy. S2 will contain the post order

1. **Preorder inorder postorder in a single traversal**

* Take num as 1,2,3
* Then there is some logic you do need to apply and remember

1. **Level order Traversal**
2. **Level order traversal in spiral form** or **Zig Zag Traversal of Binary Tree**

* My first approach is kind of brute
* Optimized approach is to use a bool var leftToright
* Here for first time I used vector<int> v(size). So that I could use index of vector to fill values
* This is the magic equation:
* // find position to fill node's value
* int index = (leftToRight) ? i : (size - 1 - i);

1. **Height of a Binary Tree**

* Practiced so much that it is very simple now.

1. **Left View Of Binary Tree**

* Maintain a level and max Level
* Recur first to left then to right
* Can also be done by level order traversal by just printing the first node at each level.

1. **Right View of Binary Tree**

* Maintain a level and max Level
* Recur first to right then to left

1. **Vertical order traversal**

* This is key for q 11 and q 12
* Use map<int, map<int, multiset<int>>>
* Use x and y co-ord and use BFS traversal

1. **Bottom View of Binary Tree**

* Uses concept of vertical order traversal

1. **Top View of Binary Tree**

* Uses concept of vertical order traversal

1. **Boundary Traversal of Binary Tree**

* Easy, just break problem into 3 parts

1. **Check if the Binary tree is height-balanced or not**

* Use a class or pair to store int height and bool isBalanced and propagate that.
* Has become easy due to doing it large no of times.

1. **Diameter of Binary Tree**

* Has become easy due to doing it large no of times.
* Use a class to store height and diameter and propagate that

1. **Check if two trees are identical or not**

* Is not at all that tough
* Jo main store karne ka sochta hoon, no need to do that
* Just use the two pointers and keep on checking both
* Simple, short and easy recursive code

1. **Symmetric Binary Tree**

* Use of helper function and left and right pointers
* Left.left == right.right and left.right = righ.left
* Good question. I get confused sometimes between 16 and 17

1. **Root to node path in a Binary Tree**

* Easy, just recursion and backtracking

1. **Invert a binary tree**
2. **Construct Binary Tree from inorder and preorder**

* Use recursion
* Need to maintain 4 indices
* Need to maintain a map

1. **Construct Binary Tree from Inorder and Postorder**
2. **Check for Children Sum Property**

* A very elegant solution to thus problem

1. **Max width of a Binary Tree**
2. **Maximum path sum**
3. **LCA in Binary Tree**

* Good question But after enough practice they are easy now. This and next

1. **Find LCA of two nodes in BST**
2. **Morris Inorder Traversal**
3. **Morris Preorder Traversal**
4. **Flatten Binary Tree to LinkedList**
5. **Connect nodes at same level (asked)**
6. **Binary Tree to Double Linked List**
7. **Cousins in Binary Tree**
8. **Sum of leaf nodes at min level**

Graphs

1. **Clone a graph (Not that easy as it looks)**

* Becomes quite Easy for me having done the Clone Linked list with random and next pointer.
* Use a map, a visited set and traverse in BFS fashion

1. **DFS**
2. **BFS**
3. **Detect A cycle in Undirected Graph using BFS**

* Using Parent Node and visited array

1. **Detect A cycle in Undirected Graph using DFS**

* Using Parent Node and visited array

1. **Detect A cycle in a Directed Graph using DFS**

* Using visited and recursive stack

1. **Detect A cycle in a Directed Graph using BFS**

* Looks tough. But is as simple as applying topological sort! Topo sort only works on directed acyclic graphs
* So if we get a valid topo sort, return no cycle else cycle exists

1. **Topological Sort BFS**

* Kahn’s Algorithm.
* No need to maintain visited array

1. **Topological Sort DFS**

* Not doing. Above one is cool

1. **Number of islands(Do in Grid and Graph Both)**

* Is easy!
* Asked in Expedia and was able to do.

1. **Bipartite Check using BFS**

* Nothing but 2 colouring problem.
* Easy

1. **Bipartite Check using DFS**

* Nothing but 2 colouring problem.
* Easy

1. **Dijkstra’s Algorithm**

* Greedy Algo. Use a priority queue
* Used when edges are weighted. Weight should not be less than 0
* Works for both directed and undirected graphs

1. **Bellman-Ford Algo**

* Similar to dijkstras only
* Repeat this(relax the nodes) N-1 times
* Nodes/Edges can be taken in any order

1. **MST using Prim’s Algo**

* Need to take 3 arrays, key,mst,parent
* For optimization, use a PQ

1. **MST using Kruskal’s Algo**
2. **Strongly Connected Component(using KosaRaju’s algo)**
3. **Floyd Warshall Algorithm**
4. **Word Ladder**
5. **M coloring Problem**

DP