LeetCode Solutions

Sankalp Sangle

Last updated on May 16, 2020

Contents

I LeetCode Top Interview Questions **1**

1 Easy **3**

1.1 Arrays . . . 3

1.1.1 26. Remove Duplicates from Sorted Array . . . 3

1.1.2 122. Best Time to Buy and Sell Stock II . . . 4

1.1.3 189. Rotate Array . . . 4

1.1.4 217. Contains Duplicate . . . 5

1.1.5 136. Single Number . . . 5

1.1.6 350. Intersection of Two Arrays II . . . 6

1.1.7 66. Plus One . . . 6

1.1.8 283. Move Zeroes . . . 7

1.1.9 1. Two Sum . . . 7

1.1.10 36. Valid Sudoku . . . 8

1.1.11 48. Rotate Image . . . 9

1.2 Strings . . . 10

1.2.1 344. Reverse String . . . 10

1.2.2 7. Reverse Integer 10
1.2.3 387. First Unique Character in a String 11
1.2.4 242. Valid Anagram 11
1.2.5 125. Valid Palindrome 11
1.2.6 28. Implement strStr() 12
1.2.7 38. Count and Say 13
1.2.8 14. Longest Common Prefix 13
1.3 Linked Lists 14
1.3.1 237. Delete Node in a Linked List 14
1.3.2 19. Remove Nth Node From End of List 14
1.3.3 206. Reverse Linked List 14
1.3.4 21. Merge Two Sorted Lists 15
1.3.5 234. Palindrome Linked List 16
1.3.6 141. Linked List Cycle 16
1.4 Trees 17
1.4.1 104. Maximum Depth of Binary Tree 17
1.4.2 98. Validate Binary Search Tree 17
1.4.3 101. Symmetric Tree 18
1.4.4 102. Binary Tree Level Order Traversal 18
1.4.5 108. Convert Sorted Array to Binary Search Tree . . . 19
1.5 Sorting and Searching 20
1.5.1 88. Merge Sorted Array 20
1.5.2 278. First Bad Version 20

1.6 Dynamic Programming 22
1.6.1 70. Climbing Stairs 22
1.6.2 121. Best Time to Buy and Sell Stock 22
1.6.3 53. Maximum Subarray 22
1.6.4 198. House Robber 23
1.7 Design 24
1.7.1 384. Shuffle an Array 24
1.7.2 155. Min Stack 24
1.8 Math 26
1.8.1 412. Fizz Buzz 26
1.8.2 204. Count Primes 26
1.8.3 326. Power of Three 27
1.8.4 13. Roman to Integer 28
1.9 Others 29
1.9.1 191. Number of 1 Bits 29
1.9.2 461. Hamming Distance 29
1.9.3 190. Reverse Bits 30
1.9.4 118. Pascal's Triangle 30
1.9.5 20. Valid Parentheses 30
1.9.6 268. Missing Number 31

2 Medium **33**

2.1 Array and Strings 33
2.1.1 15. 3Sum 33

2.1.2 73. Set Matrix Zeroes 34
2.1.3 49. Group Anagrams 34
2.1.4 3. Longest Substring Without Repeating Characters . 35
2.1.5 5. Longest Palindromic Substring 35
2.1.6 334. Increasing Triplet Subsequence 36
2.1.7 163. Missing Ranges 37
2.2 Linked List 37
2.2.1 2. Add Two Numbers 37
2.2.2 328. Odd Even Linked List 38
2.2.3 160. Intersection of Two Linked Lists 38
2.3 Trees and Graphs 39
2.3.1 94. Binary Tree Inorder Traversal 39
2.3.2 103. Binary Tree Zigzag Level Order Traversal 39
2.3.3 105. Construct Binary Tree from Preorder and Inorder Traversal 40
2.3.4 116. Populating Next Right Pointers in Each Node . . 41
2.3.5 230. Kth Smallest Element in a BST 42
2.3.6 285. Inorder Successor in BST 42
2.3.7 200. Number of Islands 43
2.4 Backtracking 43
2.4.1 17. Letter Combinations of a Phone Number 43
2.4.2 22. Generate Parentheses 44
2.4.3 46. Permutations 44

2.4.4 78. Subsets 44
2.4.5 79. Word Search 45
2.5 Sorting and Searching 45
2.5.1 75. Sort Colors 45
2.5.2 347. Top K Frequent Elements 46
2.5.3 215. Kth Largest Element in an Array 47
2.5.4 162. Find Peak Element 47

Part I

LeetCode Top Interview Questions

Chapter 1

Easy

Link: LeetCode Top Interview Questions: Easy section.

1.1 Arrays

Link: Arrays

1.1.1 26. Remove Duplicates from Sorted Array

Link to question, Link to submission

Concepts Two pointer

Algorithm description

- Maintain a read pointer and a write pointer, both starting from zero.
- Advance the write pointer until you see a new value or reach end of array.
- Write value at write location into read location.

- Return read.

1.1.2 122. Best Time to Buy and Sell Stock II

Link to question, Link to submission

Concepts Greedy

Algorithm description

- Construct a consecutive elements difference array
- Return sum of all positive elements in difference array

1.1.3 189. Rotate Array

Link to question, Link to submission approach 1, Link to submission approach 2

Concepts Cyclic replacements, Implementation

Approach 1 description

- Maintain a visited array and a pointer initialized to 0
- while pointer + k is not visited, replace arr[pointer + k] with arr[pointer]. Update pointer to pointer + k. Set pointer + k to visited, increment a numberOfChanges variable.
- Increment pointer by 1
- Keep doing this while numberOfChanges less than size of array.

Approach 2 description

- Reverse the entire array
- Reverse from start to start + k
- Reverse from start + k to end

1.1.4 217. Contains Duplicate

Link to question, Link to submission

Concepts Hash Table, Set

Algorithm description

- Initialize a Set
- For an element in array, if element in Set, return true
- else add element to Set
- If out of loop, return False

1.1.5 136. Single Number

Link to question, Link to submission

Concepts Bit Manipulation, XOR

Algorithm description

- Initialize an answer variable to 0
- For every element, XOR it to answer. Elements appearing twice get XOR'd out to zero

- Return answer

1.1.6 350. Intersection of Two Arrays II

Link to question, Link to submission approach 1, Link to submission approach 2

Concepts Hash Table, Two Pointers

Approach 1 description

- Form an element:frequency mapping using map for smaller array (to save space)
- Traverse bigger array
- If frequency of element less than 0, add to answer. Decrement frequency

Approach 2 description

- If arrays are sorted, use two pointers p1 and p2
- If nums1[p1] == nums2[p2], add to answer and increment both
- Else if nums1[p1] is smaller, increment p1. Else increment p2
- Keep doing until reach end of either array

1.1.7 66. Plus One

Link to question, Link to submission

Concepts Array

Algorithm description

- Initialize a carry variable to 1
- Traverse array from the end.
  digit[i] = carry + digit mod 10, carry = carry + digit div 10
- Finally, if carry is not zero, insert carry at start of array

1.1.8 283. Move Zeroes

Link to question, Link to submission

Concepts Two Pointers

Algorithm description

- Maintain a read and a write pointer, both initialized to 0
- if read end has zero, increment read end
- else, copy read end to write end and increment both
- After read end reaches end, set all numbers from write end to end as 0

1.1.9 1. Two Sum

Link to question, Link to submission approach 1, Link to submission approach 2

Concepts Hash Table, Two Pointer

Approach 1 description

- Create an element:indices mapping
- Sort the array
- Use two pointers to search for a particular sum
- Once you find the sum, pop index from left pointer, and pop index from right pointer
- Return indices

Approach 2 description

- Create a hashmap of int, int
- Iterate the array with i as looping variable
- If element in hashmap, return (hashmap[element], i)
- Else insert hashmap[target - element] = i

1.1.10 36. Valid Sudoku

Link to question, Link to submission

Concepts Hash Table, Set

Algorithm description

- Create sets to hold numbers for each row, col and square.
- Traverse the sudoku
- If a number is already in the row, col, square, return False
- Else, come out of loop and return true

1.1.11 48. Rotate Image

Link to question, Link to submission

Concepts Array, Circular Permutation, Implementation Heavy

Algorithm description

- Do a counterclockwise circular permutation as mentioned in solution
- Pure implementation problem. No algorithmic skill.

1.2 Strings

Link: Strings

1.2.1 344. Reverse String

Link to question, Link to submission

Concepts Two Pointers

Algorithm description

- Set a left pointer to start of string, right pointer to end
- Swap left and right. Increment left, decrement right
- Do while l less than r

1.2.2 7. Reverse Integer

Link to question, Link to submission

Concepts Two Pointers

Algorithm description

- Reverse the integer by converting to a string
- Store result in long
- If stored result is outside integer limits, return 0
- Else return the reversed number

1.2.3 387. First Unique Character in a String

Link to question, Link to submission

Concepts Hash Map

Algorithm description

- Construct element frequency mapping
- Traverse the string from the start, if frequency of a char is 1, return index
- If reach end of string, return -1

1.2.4 242. Valid Anagram

Link to question, Link to submission

Concepts Hash Map, Counting Sort

Algorithm description

- Traverse through s1, incrementing frequency counts
- Traverse through s2, decrementing frequency counts
- If all counts are zero, return true. Else false.

1.2.5 125. Valid Palindrome

Link to question, Link to submission

Concepts Two Pointers

Algorithm description

- Maintain a left and a right pointer
- Before comparing the two, ensure left and right both are pointing to an alphanumeric character

1.2.6 28. Implement strStr()

Link to question, Link to Approach 1, Link to Approach 2

Concepts Two Pointers, Rabin-Karp Algorithm, Rolling Hash

Approach 1 description

- Traverse haystack until you find a character matching with first character of needle
- Once match is found, keep checking for further characters until either there's a mismatch or you reach end of arrays
- Return index accordingly

Approach 2 description - Rabin-Karp

- Hash the needle using a hash function that is easy to be "rolled", that is it is easy to compute hash for next window if hash for previous window is known
- Traverse the haystack using window of length needle.length(). Hash the window and compare with needle hash. If matched, return the index of start of window
- See implementation carefully, very interesting. Also see LeetCode solution article.

1.2.7 38. Count and Say

Link to question, Link to submission

Concepts Recursion, Two Pointers

Algorithm description

- Base case: n = 1, return "1"
- Get the answer for n-1
- Traverse through answer of n-1
- For each consecutive list of same elements, add the count, followed by the element
- Return answer

1.2.8 14. Longest Common Prefix

Link to question, Link to submission

Concepts Implementation

Algorithm description

- Initialize answer string to ""
- Find length of smallest string
- For i from 0 to min length - 1
- Traverse through all the characters at ith positions
- If different, return answer
- If same, add character to answer

1.3 Linked Lists

Link: Linked Lists

1.3.1 237. Delete Node in a Linked List

Link to question, Link to submission

Concepts Trick

Algorithm description

- Copy value of next node into current node
- Set next ptr of current node to next ptr of next node

1.3.2 19. Remove Nth Node From End of List

Link to question, Link to submission

Concepts Two Pointer

Algorithm description

- To do it in one pass, let a forward pointer advance n steps
- Then, start forwarding a slow pointer as well as the forward pointer one at a time until forward reaches the end
- delete the slow pointer node

1.3.3 206. Reverse Linked List

Link to question, Link to iterative approach, Link to recursive approach

Concepts Implementation

Approach 1 description

- Initialize a prev = NULL, and a curr = head
- While head is not NULL, do a cyclic swap between curr.next, prev, and curr.
- Return prev

Approach 2 description

- If head is NULL or head.next is NULL return head
- l = reversed list for head.next
- head.next.next = head, head.next = NULL. Return l

1.3.4 21. Merge Two Sorted Lists

Link to question, Link to iterative submission, Link to recursive submission

Concepts Two Pointers

Algorithm description Iterative

- Make a dummy node, and let tmp = dummynode
- Keep appending the smaller of the two lists to the dummy node and advance the pointers accordingly
- If one of the lists becomes NULL, append the other list to dummy node
- Return next of tmp

Algorithm description Recursive

- If either of lists is NULL, return the other
- if l1 is smaller, get answer to (l1.next, l2) and set it as l1.next. Return l1
- Else get answer to (l1, l2.next) and set it as l2.next. Return l2

1.3.5 234. Palindrome Linked List

Link to question, Link to submission

Concepts Reverse a linked list, Two Pointers

Algorithm description

- Reverse the second half of the linked list
- Compare nodewise the head of linked list and the head of reversed list to check for palindrome

1.3.6 141. Linked List Cycle

Link to question, Link to submission

Concepts Hare and Tortoise, Two Pointers

Algorithm description

- Initialize a slow and a fast pointer
- Advance slow by 1, fast by 2
- If slow and fast meet, there's a cycle. Else if fast reaches end, there's no cycle.

1.4 Trees

Link: Trees

1.4.1 104. Maximum Depth of Binary Tree

Link to question, Link to recursive submission, Link to iterative submission

Concepts Recursion, Stack

Algorithm description Recursive

- If root is null, return 0
- Else return 1 + max(maxDepth(left), maxDepth(right))

Algorithm description Iterative

- If root is null, return 0
- Initialize stack holding pair of TreeNode and depth
- Push {root, 1}
- While stack is not empty, get top of stack
- If top is leaf, compare with maxDepth
- Push children if any with depth = 1 + parent depth

1.4.2 98. Validate Binary Search Tree

Link to question, Link to iterative submission, Link to recursive submission

Concepts Top-Down

Algorithm description (for recursive/iterative)

- Approach is a top-down one
- At every node, check if node.val is between a range of [small, large]
- If not, return False
- else check left subtree for range[small, node.val] and check right subtree for range[node.val, large]
- Return the AND of the above two

1.4.3 101. Symmetric Tree

Link to question, Link to recursive submission, Link to iterative submission

Concepts Top-Down

Algorithm description (for recursive/iterative)

- Top down approach
- Check if leftTree.val == rightTree.val
- If true, check for leftTree.left, rightTree.right and leftTree.right, rightTree.left
- Else, return False

1.4.4 102. Binary Tree Level Order Traversal

Link to question, Link to submission

Concepts Top-Down, BFS

Algorithm description

- Push root into a queue
- At beginning of an iteration, take size of queue
- Pop out #size items from queue, while adding their children to queue
- Add to level
- Add level to final answer

1.4.5 108. Convert Sorted Array to Binary Search Tree

Link to question, Link to submission

Concepts Recursion, Preorder

Algorithm description

- call procedure with left = 0, right = arr.size() - 1
- if left greater than right, return NULL
- construct node for middle element
- node.left = procedure(left, middle-1), node.right = procedure(middle+1, right)
- return node

1.5 Sorting and Searching

Link: Sorting and Searching

1.5.1 88. Merge Sorted Array

Link to question, Link to submission

Concepts Two Pointers

Algorithm description

- Create a copy array for nums1
- Maintain write pointer for nums1, p1 for nums1copy, p2 for nums2
- Write smaller of p1, p2 into nums1. Advance smaller and write head.
- Once out of the loop, see which array still has elements remaining. Add them to nums1

1.5.2 278. First Bad Version

Link to question, Link to submission

Concepts Binary Search

Algorithm description

- set left as 0, right as n - 1
- while l less than equal to r
- if mid is bad, right = middle - 1

- else left = middle + 1
- Once you come out of loop, return l

1.6 Dynamic Programming

Link: Dynamic Programming

1.6.1 70. Climbing Stairs

Link to question, Link to submission

Concepts Dynamic Programming

Algorithm description

- Ways to reach ith step = ways to reach i-1 th step plus ways to reach i-2 th step

1.6.2 121. Best Time to Buy and Sell Stock

Link to question, Link to submission

Concepts Dynamic Programming

Algorithm description

- Maintain a smallest stock price seen yet variable
- Update maxProfit = max(maxProfit, current price - maxProfit)

1.6.3 53. Maximum Subarray

Link to question, Link to submission

Concepts Dynamic Programming

Algorithm description

- Maintain a current sum variable, denoting the highest sum possible that contains the element at the index
- Maintain a highest sum variable, denoting the highest sum encountered among the current sums

1.6.4 198. House Robber

Link to question, Link to submission, Link to submission (space optimized)

Concepts Dynamic Programming

Algorithm description

- Maintain a dp array with dp[0] = nums[0], dp[1] = max(nums[0], nums[1]). dp[i] denotes maximum amount that can be robbed with first i+1 houses
- dp[i] = max(dp[i-1], dp[i-2] + nums[i])
- Finally return dp[n-1]

1.7 Design

Link: Design

1.7.1 384. Shuffle an Array

Link to question, Link to submission

Concepts Fisher-Yates Algorithm, Random Permutation

Algorithm description

- Iterate through the array
- For every iteration, generate an index between [current index, last index]
- Swap elements at current index and generated index
- Return array

1.7.2 155. Min Stack

Link to question, Link to submission

Concepts Two stacks

Algorithm description

- Use one stack to keep track of all elements, and another minstack to keep track of minimums
- Push(x): push x to stack. Push x to minstack only if x less than or equal to top of minstack or if minstack is empty

- Pop(): pop from stack. Pop from minstack if stack.top() == minstack.top()
- getMin(): return minstack.top()

1.8 Math

Link: Math

1.8.1 412. Fizz Buzz

Link to question, Link to submission 1, Link to submission 2 (Easier to maintain)

Concepts Divisibility

Algorithm description

- Instead of following naive approach of check divisibility by 15 first, then by 3 and 5, use an incremental approach
- Add "Fizz" to answer if divisible by 3
- Add "Buzz" to answer if divisible by 5
- This code is much easier to maintain if more conditions like 7:"Jazz" are added. Also note submission 2, which is even easier to maintain

1.8.2 204. Count Primes

Link to question, Link to submission

Concepts Number Theory, Math, Primes

Algorithm description

- Let's start with a isPrime function. To determine if a number is prime, we need to check if it is not divisible by any number less than n. The runtime complexity of isPrime function would be O(n) and hence counting the total prime numbers up to n would be O(n2). Could we do better?

- As we know the number must not be divisible by any number greater than n / 2, we can immediately cut the total iterations half by dividing only up to n / 2. Could we still do better?
- We don't need to go all the way till n / 2. Just stopping at sqrt(n) is enough. Complexity is now O(n to the power 1.5)
- Notice that if we've tested for the number x being prime, we don't need to test for multiples of x being prime anymore. This is the motivation for the Sieve of Eratosthenes. Take a number, if it is not visited, mark all it's multiples excluding itself as visited. Increment number and repeat.
- One optimization is to not start at 2x but to start at x times x, as 2x had already been marked when marking multiples of two.
- Finally, there is no need to go through all numbers till n. We only need to do the sieve for numbers till root of n.
- Answer is count of unvisited elements in visited array
- Definitely look through the final submission for all the optimizations.
- Complexity - O(n log log n)

1.8.3 326. Power of Three

Link to question, Link to submission

Concepts Math

Algorithm description

- Since they're asking for no loops/recursion (which would be the naive approach), the idea is to find the largest power of 3 that fits in 4 byte size

- If (largest number which is power of 3) % num == 0, then number is a power of 3
- Bear in mind that this will only work for powers of x where x is a prime number.
- Do look at editorial for a good discussion on logarithms approach as well.

1.8.4 13. Roman to Integer

Link to question, Link to submission

Concepts Parsing

Algorithm description

- Maintain two pointers, current and next, initialized to 0 and 1
- If value at curr greater than equal value at next, add value at curr to answer. Increment both pointers
- Else, add value at next to answer, subtract value at curr to answer, increment both pointers by two.
- Once out of loop, if curr less than string.length(), add value at curr to answer
- Return answer
- Note: this is a left to right pass solution. Also see the right to left pass submission shown in the editorial

1.9 Others

Link: Others

1.9.1 191. Number of 1 Bits

Link to question, Link to submission

Concepts Bit Manipulation

Algorithm description

- initialize answer to 0
- While number not equal to 0, set number as (number & number-1), increment answer
- return answer

1.9.2 461. Hamming Distance

Link to question, Link to submission

Concepts Bit Manipulation

Algorithm description

- Let A be xor of x and y (xor returns a 1 if the operands are different)
- Count number of set bits in A

1.9.3 190. Reverse Bits

Link to question, Link to submission 1 (naive), Link to submission 2 (constant time)

Concepts Bit Manipulation

Algorithm description

- Naive solution is clear, compare bits at opposite ends. If different, flip them
- For constant time, first, we break the original 32-bit into 2 blocks of 16 bits, and switch them.
- We then break the 16-bits block into 2 blocks of 8 bits. Similarly, we switch the position of the 8-bits blocks
- We then continue to break the blocks into smaller blocks, until we reach the level with the block of 1 bit.

1.9.4 118. Pascal's Triangle

Link to question, Link to submission

Concepts Implementation

Algorithm description

- Nothing fancy, just construct row by row as mentioned in the description.

1.9.5 20. Valid Parentheses

Link to question, Link to submission

Concepts Stack

Algorithm description

- Iterate through string
- If it's an opening bracket, push onto stack
- Else, if it's not a matching bracket, return False
- If matching bracket, pop from stack
- When you come out of loop, if stack is empty return true
- Return false

1.9.6 268. Missing Number

Link to question, Link to submission

Concepts Bit Manipulation

Algorithm description

- XOR all numbers in the range [0,n] into a variable answer
- Iterate through the array, XORing every element into the answer variable
- Return answer. All elements will have appeared twice, except the missing number which appeared once, and hence is stored in answer variable

Chapter 2

Medium

Link: LeetCode Top Interview Questions: Medium section.

2.1 Array and Strings

Link: Array and Strings

2.1.1 15. 3Sum

Link to question, Link to submission

Concepts Two Pointer, Sorting

Algorithm description

- Sort the array
- Traverse from left. For each iteration, fix target as -1 * nums[i]
- Maintain left ptr as i+1, and right ptr as end of arrays
- Search if sum of values at left and right equals target

- If so, add triplet to answer. Move left and right pointers along accordingly

- Take care to write loops to skip over duplicate values at left, right and i. Avoids TLE.

2.1.2 73. Set Matrix Zeroes

Link to question, Link to submission

Concepts Space Optimization, In Place

Algorithm description

- Set boolean variables to decide if first row and first column need setting to zero

- Traverse matrix (excluding first row and first column)

- Wherever arr[i][j] == 0, set arr[i][0] and arr[0][j] as 0

- Traverse matrix (excluding first row and first column). If arr[i][0] == 0 or arr[0][j] == 0, set arr[i][j] = 0

- Finally, set first row and first column as zero if needed, as decided in first step

2.1.3 49. Group Anagrams

Link to question, Link to submission

Concepts Sorting, Hashtable

Algorithm description

- Set up a map of string, vector¡string¿
- Traverse array
- For a string, sort it, and append original string to vector at hashed value of sorted string
- Finally, append all vectors to an answer array and return the array

2.1.4 3. Longest Substring Without Repeating Characters

Link to question, Link to submission

Concepts Sliding Window, HashMap, Two Pointer

Algorithm description

- Initialize left and right both at 0
- Advance right as you keep getting characters and store their indexes in a map. Keep updating maxLen as max(maxLen, r - l + 1)
- The moment you get a repeated character, delete all entries in the map for characters from left ptr to first occurence of repeated character.
- Then, update position of left to one index after the first occurence of repeated character, as well as update the first occurence of repeated character as the right pointer.
- Return maxLen

2.1.5 5. Longest Palindromic Substring

Link to question, Link to submission, Link to DP submission

Concepts DP, Two Pointer, Implementation Heavy

Approach 1 description

- Start at each of the 2 * len - 1 possible centres of the string.
- Keep expanding outside until palindrome.
- Store longest palindrome in answer and return answer

DP description

- dp[i][i] = true, dp[i][i+1] = true if s[i] == s[i+1]
- dp[i][j] = true if dp[i+1][j-1] == true and s[i] == s[j]
- Finally return s.substr(starting index, maxLength)
- Do look at implementation to see how dp array is filled. Order is not top to bottom, left to right. It is filled in a diamond shaped manner. Remember DAA course? That way.

2.1.6 334. Increasing Triplet Subsequence

Link to question, Link to submission

Concepts If-Else

Algorithm description

- Keep a smallest and a second smallest, both initialized at INT MAX
- Traverse the array
- If number less than equal to smallest, update smallest
- Else if number less than equal to second smallest, update second smallest
- Else return true

2.1.7 163. Missing Ranges

Link to question, Link to submission

Concepts Arrays, Implementation

Algorithm description

- Create a new long datatype vector out of integers of nums
- Push lower - 1 and upper + 1 to long vector
- Generate a differences array
- If difference ¡ 2, continue
- If difference equals 2, push back longarray[i]+1 to answer
- Else push back (longarray[i]+1)-¿(longarray[i+1] - 1) to answer
- Return answer

2.2 Linked List

Link: Linked List

2.2.1 2. Add Two Numbers

Link to question, Link to submission

Concepts Linked List

Algorithm description

- Recursive algorithm, construct new node as sum of l1, l2 and carry.
- Let next of new node be answer to recursion call for l1.next, l2.next and new carry.
- Return new node

2.2.2 328. Odd Even Linked List

Link to question, Link to submission

Concepts Linked List Manipulation

Algorithm description

- Maintain a current pointer. Set curr.next as curr.next.next. Advance the current pointer.
- Finally link the end of the odd list to the start of the even list.

2.2.3 160. Intersection of Two Linked Lists

Link to question, Link to approach 1, Link to approach 2

Concepts Two pointers, modulus, Smart

Approach 1 description

- Push the pointer for the larger list forward by x times where x is difference between length of larger and smaller lists.
- Then, while pointer 1 doesn't equal pointer 2, keep advancing both.
- Return pointer 1

Approach 2 description

- Keep advancing both pointers. If either one reaches the end, shift it to start of other's head and save the last node of the list.
- If they ever match, return the match. Else if their last nodes are both not NULL but different, return NULL (Means no intersection at all).

2.3 Trees and Graphs

Link: Trees and Graphs

2.3.1 94. Binary Tree Inorder Traversal

Link to question, Link to iterative submission, Link to recursive submission.

Concepts Stack, Tree, Inorder

Iterative Algorithm description

- Recursive is trivial. Left, Middle, Right.
- For iterative, before starting while loop, keep pushing root into stack, and setting root as root.left
- Now, inside the while loop, pop the top. Push back top to inorder vector. If top.right, exists, keep pushing top.right to stack and setting top.right to it's left subtree.

2.3.2 103. Binary Tree Zigzag Level Order Traversal

Link to question, Link to submission

Concepts Queue, Level Order Traversal

Algorithm description

- Same as a level order traversal, just flip each alternate level before adding to answer array

2.3.3 105. Construct Binary Tree from Preorder and Inorder Traversal

Link to question, Link to submission

Concepts Inorder, Preorder, Recursion, Implementation Heavy

Algorithm description

- Construct a mapping of ele:index in inorder arrays
- Recursively construct the binary tree using helper(inorder, preorder, left index, right index, mapping)
- If left ¿ right, return NULL
- Else, construct node with value as preorder[0]
- Pop front of preorder
- node.left is helper(inorder, preorder, left index, mapping[node.value] - 1, mapping)
- node.right is helper(inorder, preorder, mapping[node.value] + 1, right index, mapping)
- return node

2.3.4 116. Populating Next Right Pointers in Each Node

Link to question, Link to level order submission, Link to recursive submission, Link to iterative submission.

Concepts Level Order Traversal, Recursion

Level Order Algorithm description

- Go via a level order traversal
- For each level, link (i)th node.next = (i+1)th node

Recursive Algorithm description

- Top down approach
- Assume you're at a level i, and the i-1th has next links established. You have the leftmost node at ith level, and it's parent at i-1th level.
- Set current node = leftmost node. If current node is a left node, set next as right node of parent. If current node is a right node, set next as left node of next of parent.
- Advance current, and parent nodes accordingly.
- Finally, call function for (leftmost.left, leftmost)
- Return leftmost

Iterative Algorithm description

- Same algorithm as above, you just form linkages for i+1th level, while having already established linkages for the ith level which you are on.
- Set leftmost node to root. While leftmost.left != NULL, proceed as follows

- Set curr = leftmost. Set curr.left.next as curr.right. Set curr.right.next = curr.next.left. Keep advancing curr.
- Set leftmost = leftmost.left
- Finally, return root

2.3.5 230. Kth Smallest Element in a BST

Link to question, Link to approach 1, Link to approach 2.

Concepts Inorder

Approach 1 description

- Find inorder array
- Return inorder[k-1]

Approach 2 description

- Traverse tree inorder. Once you visit k nodes, return value at current node.

2.3.6 285. Inorder Successor in BST

Link to question, Link to approach 1, Link to approach 2.

Concepts Inorder

Approach 1 description

- Find inorder array
- Return node that is next in line after node to be searched for

Approach 2 description

- Traverse tree inorder. Once you reach node to be searched for, return the very next node you find.

2.3.7 200. Number of Islands

Link to question, Link to DFS submission Link to BFS submission.

Concepts BFS / DFS

Algorithm description

- Simple problem of finding connected components in a graph
- Use either BFS or DFS

2.4 Backtracking

Link: Backtracking

2.4.1 17. Letter Combinations of a Phone Number

Link to question, Link to submission

Concepts Backtracking, Recursion

Algorithm description

- For every letter corresponding to an index of string, add it to the string and keep searching for further possibilites
- Add to answer once you reach end of string

2.4.2 22. Generate Parentheses

Link to question, Link to submission

Concepts Backtracking, Recursion, Stack

Algorithm description

- Maintain an integer variable to act as a counter. It increments if "(" is added and decrements if ")" is added.
- If at any point if counter goes negative, backtrack
- If index reaches end of string and counter is zero as well, add to answer

2.4.3 46. Permutations

Link to question, Link to submission

Concepts Backtracking, Recursion, Permutations

Algorithm description

- For every position i from index to end of string, swap i with index, and make the call on string with index = index + 1
- Remember to swap back before the next i
- When index reaches length of string, add to answer

2.4.4 78. Subsets

Link to question, Link to submission

Concepts Backtracking, Recursion

Algorithm description

- For element at current index, add it to current subset. Call helper for next index.
- Once you return from helper, pop from current subset and call helper again for next index.
- Whenever you reach index == set.length(), add current subset to answer

2.4.5 79. Word Search

Link to question, Link to submission

Concepts DFS

Algorithm description

- Once you see that the first character has matched, start a DFS for the rest of the string
- Algorithm is difficult to implement taking care of all edge cases, definitely implement it once.

2.5 Sorting and Searching

Link: Sorting and Searching

2.5.1 75. Sort Colors

Link to question, Link to submission

Concepts Hashtable, 2 Pointers

Algorithm description

- One approach is to simply keep a count of elements in first pass, and overwrite array in second pass accordingly.
- Question asks for one pass approach.
- Keep a write end for zeroes, at start of array, and a write end for twos , at the end of the array.
- Let a read pointer go from start to right end.
- If it encounters a 0, swap with left end. Increment both left end and read end.
- If it encounters a 1, continue
- If it encounters a 2, swap with right end. Decrement right end but DO NOT increment read end as the element coming from the right end also needs to be put in place.

2.5.2 347. Top K Frequent Elements

Link to question, Link to submission

Concepts Priority Queue/ Sorting

Algorithm description

- Build a hashmap of character frequencies
- Insert pair of (frequency, character) into a priority queue/ vector
- Either sort the vector and return last k elements, or pop k elements from the priority queue

2.5.3 215. Kth Largest Element in an Array

Link to question, Link to submission

Concepts Priority Queue/ Sorting, Quick Select

Algorithm description

- Sort in descending and return the kth element from the beginning
- Or use a priority queue and maintain its max length as k. After insertion of all elements, keep popping and return the last
- OR, use quick select. Very important, see solution for more details. Average running time complexity is O(n), worst is O(n**2)

2.5.4 162. Find Peak Element

Link to question, Link to submission

Concepts Binary Search

Algorithm description

- If element is lesser than the one on its right, search in right subarray (since it's now guaranteed to have a peak)
- Else if element is lesser than the one on its left, search in left subarray (since it's now guaranteed to have a peak)
- Else return index of element. (Since it's a peak as it's larger than both)