DSA – REVISION SHEET

[Format : -

Problem Np. :- brief desc

Soltuion :-]

- **1071 :- Given two strings . find if any part of second string divides first string . (the substring should replicate n times to give str1 and str2)**

Solution :- the base condition is str1+str2 == str2+str1 , if not its not possible . then u find the GCD of lengths of strings using Euclidian algo
[while len2 : len1 = le2, len2 = len1%len2 . return len1] and the answer would be the substring of str1 upto the GCD

- **1071 :- Reverse vowels of a string**

Solution :- Two pointer approach . swap till l < r. also important to check if l < r in if conditions as well

- **238 :- Product of Array self (product of every other element except itself)**

Solution :- prefix/ postfix product approach . calculate prefix product and postfix product . then multiple both to get answer . u can do it in a single external array .

- **334 :- increasing triplet sequence (non contiguous)**

Solution :- just store 2 variables first and second and update these variables accordingly . if u find some number which is not smaller than first and second return true

- **11 :- container with most water**

Solution :- two pointer . compute min(l, r)*(r-l) and store max. move the smaller pointer

- **1004 :- Longest subsequence of 1s after flipping k 0s**

Solution :- dynamic sliding window . store zeros and whenever 0 is there increment . if zeros go over k, move the left pointer till u reach the first swapped zero to remove it. U can move right pointer using a normal for loop. Store maxLen

- **1657 :-** determine if two strings are close . rule 1: u can swap single chars rule 2 u can swap all chars of a to all chars of b.

Solution :- the primary idea is 1) all unique chars in both strings should be same for rule 1 . 2) when sorted along the values/frequency , they should be same for both strings . **USAGE OF COUNTER function**

- **2352 :-** Number of equal rows and columns

Solution :- know how to get a **column** vals in a single line in python . [row[i] for row in grid]

- **735 :-** Asteroid collision . they collide only if the prev ele was positive and next ele is negative

Solution :- use a stack obvio . for every ele , put it in a **WHILE ELSE** loop w condition if stack not empty and **i < 0 < stack[-1]** (basically -ve and +ve) . then follow the rules they said . else append to stack

- **649 :-** DOTA2 radiant , dire question . an radiant can disable voting right for a dire for all subsequent rounds and vice versa . who will win?

Solution :- We use two queues for each radiant and dire . now add the indices of their occurrences to the q . now while both the queues are not empty pop one from each and compare their indices . the smaller index moves onto to the next round gets appended as (i + n) to simulate next round

- **2130 :-** Max twin sum of linked list (twins are the edge indexes[0,n-1][1,n-2]...)

Solution :- u use all basic linked operations . first u go to middle and end of linked list using slow and fast pointers . then u reverse second half of linked list

from slow to fast . then u compute sum . (reversing : while curr: nxt = curr.next, curr.next = prev, prev = curr, curr = nxt)

- **104 :- max depth of binary tree**

Solution :- two line code . if not null , u return max(bfs(node.left) , bfs(node.right)) + 1 . idea – depth from each node is the depth of the max of its left/right subtree + 1.

- **1448 :- good nodes in a binary tree (if that node.val is the max seen so far while going down)**

Solution :- obvio dfs but remember how to store the counter . send the maxValue as a parameter . if node val > maxVal , then update maxVal and store good as 1 , else 0 (one line statement) now call good += dfs(left, max) dfs(right, max) and then finally return good. Implementation is important(no subtraction or anything)

- **437 :- path sum 3 . path sum should be equal to target sum from any node.**

Solution :- u have to memoization or prefix sum (hash) which basically stores the pathsum till every node along with it's count. Now when u reach a node, u add its value to pathsum and check the hash for the value of [pathsum – target] to check for paths starting from the middle. Store that in count and add the count += for left and right subtree. Also finally u have to remove the pathsum from hash -= 1

- **1372 :- longest zig zag pattern from any node of a binary tree**

Solution :- do a dfs using dir and currLen . at every node u have to choices to take opp dir or same dir . if same currLen = 1 else currLen + 1 . starting store the max currLen .

- **236 :- lowest common ancestor in binary tree of p and q**

Solution :- do a dfs where u return root . store left and right and call dfs on them . base case is if root is p or root is q or null return root . if left and right return

root . or if left then left else right [for cases where p or q itself is the answer] . POSTORDER DFS

- **199 :-** Binary tree right side view (right most element in each level)

Solution :- do a bfs . the concept is every for loop iteration are the nodes in that level . so store each node in a var and append after for loop ends

- **450 :-** deleting a node in BST

Solution :- use the given func itself as a recursion . traverse to the target node . now we have three cases 1) its an leaf node which means just return that node. 2) it has only one child then return the other child . 3) else u swap the val with leftmost child of its right subtree and then call delete on that node with root.right as its root (inorder successor)

- **547 :-** number of provinces . output the no. of diff provinces (connected graphs considered as one ,each indivduals as one)

Solution :- we have to use graph dfs . in order to iterate through every node , we call a for loop on all nodes in visited arr and if false we do a dfs on them . after each dfs res is incremented and finally returned . use UNION FIND

- **1466 :-** Reorder routes to root city . return the cost (if same dir cost = 0 , if opp direction cost = 1. Return cost to visit all cities)

Solution :- we first form a adjaceny list of undirected graph this way . (for a u<-v(needs swapping) we add u[v] = 1, v[u] = 0) . now we do a dfs from root . we pass parameter of parent to make sure we don't get stuck in a cycle . now we iterate through the undirected graph we made . we add change_count += dir and change_count += dfs(neighbor, currNode) and then finally return change_count iterative dfs . DFS PARENT VARIENT i.e undirected DFS

- **399 :-** Evaluate division . give answers for a/b , b/c etc we need to find and return a/c , d/c etc (queries)

Solution :- the idea is that we create a graph of adjacency list and we append graph[a][b] = val and graph[b][a] = 1/val . basic principle to find a/c is a/b*b/c so chain multiply . now in dfs we need to keep track of target , multiplication val and visited set . now we iterate from first letter to last letter DFS and keep updating multiplication . if curr = target return res . and then if res is not equal to -1 we return res after the dfs in the for loop. else we will return res = -1 which Is put outside for loop by default (this format is followed when we need to find the answer which is at the end or return false if u cant, used in sudoku Q as well). This is returned when the current path is invalid

- **1926 :-** nearest exit from entrance of maze

Solution :- we do a bfs . we have two functions . neighbor and isvalid . we iterate through through the neighbors and if they are an edge node we return . isValid generally contains (0<= r<m , 0<=l<n and problem condition) . edges means row = 0 or col =0 or row = m-1 or col =n-1 . neighbors are (r+1, c)(r-1,c)(r,c+1)(r,c-1)

- **994 :-** Rotten oranges question . there could multiple rotten oranges . return how long it takes for all oranges to be rotten . rotten oranges can rot other oranges in the normal perpendicular directions

Solution :- here we do multi source dfs where we add all rotten oranges to the q . we also keep track of fresh count . if fresh = 0 in the starting only then return . else u add time after every iteration of bfs . once done return time + 1

- **215 :-** Kth largest distinct element in an array (without sorting)

Solution :- we form a min heap and add the first k numbers from array . then if we encounter a smaller number in the rest we add it to heap . finally the heap[0] will have the answer

- **2336 :-** smallest infinity set question . u need to give smallest integer . the set has infinite values . and u can addBack popped integers

Solution :- maintain a heap, visited set and curr . when a number is added and is less than curr and not already added to set, add to heap and set . when popping smallest return heap till its empty then return curr-1

- **2542 :-** maximum subsequence score . (given two arrays . u choose three indexes and first arr u sum and multiply that with minimum of the 2nd array)

Solution :- this has a greedy heap approach . u first sort the zipped num2,num1 in descending order based on 2nd array . now u add to heap . once len of heap > k , we pop the min ele, reduce sum . once the len == k take the max of the sum

- **2462 :-** ur given two parameters k rounds and number of candidates . u have hire k workers (1 each round) and get min cost . candidates lets say n is elgibile workers for that round and it arr[:n] and arr[n:].

Solution :- we take 2 pointers l and r . and we maintain a global heap of all candidates . first we add candidates one from and one right updating pointers and l <= r . and then we go pop from heap . if index is lesser than l we push(l) and do l +=1 else r -= 1 (again both only if l <= r) . keep track of cost

- **2300 :-** spells[] and portions[] question . they gave a target where each spell*portion[] should be >= target . how many such pairs[for spell1, for spell2 ...etc]?

Solution :- this is a binary search approach we use python library bisect_left to find the pivot point . we use the concept math.ciel() . in python its achieved by lets say a/ b = a +b – 1 // b . we find the divided target/spell as
target + spell – 1 // spell . then we use bisect_left to get breaking point and append len(n) – point

- **162 :-** find the peak of array (there could be multiple peaks). We need to return any one

Solution :- so this is a stupid solution but remember how to find the pivot point in binary search . the format is while l < r : if mid > mid + 1: right = mid else left = mid + 1 . for this question I don't understand I remember it as if mid > mid + 1 i.e

down slope , the peak is in left . else peak is right (up slope) . return l or r (either)

- **875 :-** koko eating bananas question where piles array has number of bananas. We are given how many hours(h) is there and we need to find minimum speed of her eating bananas to finish piles n

Solution :- idea is the max would max(piles) and min would be 1 . we do binary search on this and compute to see if its working (find the total += for ciel division by mid for each pile and check if it is more or less than h) . again we find the pivot point (similar format) . return l or r (either)

- **17 :-** phone keypad question . give all possible combinations

Solution :- key in backtracking questions is to draw the decision tree . in our backtrack we send index for which digit and the sol. Not using global sol arr int his case . base case is if len(so) == n then we append to res . we use a for loop to loop through each char in that digit and call backtrack for them

- **216 :-** all K numbers that sum up to N.(numbers cant be repeated and 1-9)

Solution :- here we have a global sol []. We keep track of remaining sum to eliminate a few cases . base case is remaining = 0 and len(sol) == k then we append(sol[:])//important . elif len(sol)==k or remaining < 0 : then just return else for loop from start to 9 --- if l > remaining break else backtrack(start, remaining – i). the idea is cases even the ones before start would be covered by previous permutations

- **DP IDEA (FROM VIDEOS AND EASY PROBLEMS)**
  - **First step is to visualize based on a directed acyclic graph**
  - **Second divide it into subproblems**
  - **Now connect the subproblems . we generally use a for loop from base cases upto the solution . bottom up approach .**
  - **Dp is just said recursion + memoization**

- **746 :-** Climbing stairs question . u can either take 1 or 2 steps . u can start from 0 or 1.calc min cost to reach end of array

Solution :- u have the choice of taking the next step or the next to next step . u use dp directly . where base cases are memo[0]=0, memo[1]=0 . in for loop do memo[i] = min(mem[i-1] + cost[i-1] , memo[i-2]+cost[i-2])

- **198 :-** house robber . u cant rob adjacent houses . highest loot?

Solution :- recursive is simple . for dp u choose to rob this house(this + prev or prev) or skip it and rob prev house .curr house is nums[i] but stores in dp of i+1 to factor in base case. u iterate from the i=1 update memo[i+1] = max(memo[i-1]+val[i],memo[i]) . base case -> memo[0]=0,mem[1]=nums[1].

- **790 :-** given n , and a 2*n matrix . how many ways can u fill it with dominos and trominos with no empty tiles.

Solution :- write down the answers till n = 5 and see the pattern . u can do easily arrive with it after that . it was I = 2*i-1 + i-3 . the idea was there is always 2 diff i-1 matrixes in i and i-3 is the extra part

- **62 :-** find the end of the maze (bottom right) from top left . how many ways? You can only move down or right

Solution :- the idea is to do bottom up from bottom right cell = 0 , last row and col = 1 and then slowly solve till u reach the top

- **1143 :-** longest common subsequence question . very common DP format

Solution :- 2D DP w i,j .now we have 2 options they both can be same or they are diff . if they are diff(atleast wont be in sol) dp[i][j] = max(dp[i-1][j],dp[i],[j-1]) . if they are equal(both are in sol) then dp[i][j] = dp[i-1][j-1] + 1 . we initialize DP with 0s[important for the arr to be [l+1][r+1]]. Base case is anything with a index as 0 I.e [i,0] or [0,j].

- **122 :-** best time to buy sell stock 2 . u can buy and sell in the same day.

Solution :- DP approach –

The idea is that u can either buy or sell at each stock(at each option u can either buy or not buy and go to next similarly for sell).u take the max of both choices and store it . u start dp from the end to make it simpler. Base case is memo[(n,0)] = 0, memo[(n,1)] = 0 (where 0 and 1 represents buy variable). Now u iterate through all the elements and second loop with buy as 1 and 0. Return memo(0,0)

Greedy –

U simply add the profits if the price is higher than for what u bought the previous day . apparently u don't consider anything else [doubt]

- **714 :-** best time to buy and sell stock with transaction fee

Solution :-

DP :- it's the same idea but add a transaction fee while selling as question suggests.

SPACE OPTIMISED :- in dp we can see we just store two more variables(currBuy, currNotBuy, aheadBuy, aheadNotBuy) to see the previous max of each case buy and sell . so we store the max in currBuy, currNotbuy.and then update aheadBuy and aheadNotBuy after that. Return aheadBuy . same formula but instead of memo these variables

- **72 :-** Edit distance question . ur given two string and u can perform 3 operations - > insert (ur appending to the left) , delete,replace . ur goal is to edit str1 to make it match str2

Solution :- once u find recursive its easy to find top down and bottom up dp. First draw the tree and use 3 options and notice . notice it's a 2d DP

Recurisive approach :- we can see we have 3 options which correspond to delete i.e backtrack(l+1,r) , replace(l+1,r+1) and insert turns out to be (l,r-1) . we use standard recursing approach where we return 1 + min(all 3 cases) . main base case is words[l] == words[r] then just return backtrack(l+1,r+1). 2 other cases are where every letter is matching but there are more letters left to delete in either string . we put 2 if cases to l == len(str1) then return len(word2) – r and vice versa .

Memo is just storing result or the 1 + min() part (top down DP)

DP :- the subproblem is min operations till (l,r) index . we can use memo as a dict and use -1 indexes as base case for better understanding but its pretty slow. Using a DP matrix(m+1,n+1) . fill it with 0 and and the first row and column should have l/r present (i.e that many delete operations) now iterate through (1,m) and (1,n) . now base case is again if its same char (check prev index since we are starting from 1,len(n)+1) then just memo[i][j] = memo[i-1][j-1] else the 1 + min() thing. Try doing it directly in dp when ur free

- 338 :- counting '1' bit in each number from 0 to n and returning the array

Solution :- we could use pattern recognition or use the idea that every right shift divides by 2 . even numbers have the same number of 1s as the number/2 does while odd ones have +1 . // does integer division .

- 1318 :- minimum flips to make A or B equal to C

Solution :- the idea is to right shift i times and get the bits first check if a | b =c if not now check if c is 1 or 0 . if 1 , then u just need to toggle a or b which means op+1 , if 0 then we have to change both a and b to 0 and we can see how many ops it takes by op1 += a_bit + b_bit (gives number of 1s)

- 208 :- Implementing a trie . inserting , search word , searching prefix

Solution :- the implementation of trie is through dict of dicts . all the node connected to root are added to the dict with its on dict and it keeps going on. The end of a word is marked '.' : '.' . now inserting is for every letter u check if its there on trie , if not u add it to dict by d[ch]={} now after every letter u go into its dict and keep going like dfs that's how u iterate . searching word and prefix is self explanatory if u know how to iterate

- **1268 :-** Search suggestions . u should give 3 in lexico order . u have all the words and the search string . u should return list of lists including suggestions as the user types the string

Solution :-

Trie implementation :- here we create a new class with constructor having a dict and list of suggestions . now adding to the trie we do normally and outside we check if len(suggestions) < 3 then we append the word itself . now while appending finally also check if node exists and ch in node.children then append it . else put node as None(to fill the list with []) and append [] . check the implementation as its complicated

Two pointer :- the idea is to sort the array and use two pointers l and r. for each char in search word , we update our and l and r to get the valid words . this is done through a while loop with conditions l <=r and (if the word has len lesser than i (ith char in search word) or product[l][i] != ch [failing cases]) then l+=1 similarly for r-=1 then we append an empty [] to res (for cases where no right answer) then loop min(3,r-l+1) amount of times and append to the last empty res using product[l+j]. look at the implementation as its tricky

- **435 :-** non overlapping intervals . how many intervals do u remove to make everything non overlapping

Solution :- Sorting + greedy approach . we sort based on the ending points of intervals . now we have a prev pointer and initialize counter to 0. when its overlapping just increment else update prev counter

- **452 :-** min number of arrows to burst balloons

Solution :- 1st approach :- similar to last one but we sort based on starting and the idea is merge all the overlapping intervals to where is overlapping. Prev here is an interval and we compare that . remember if overlapping prev = curr[0],min([prev[1],curr[1]) to handle both cases . count intialised to 1 and incremented when its not overlapping
2nd approach :- here we sort based on end intervals and store the arrow shooting point as the end of the intervals . if starting point > arroAt then increment and update arrowAt to its end. Counter initialized to 1 again.

- **739 :-** daily temperatures . how many days before the temperature gets warmer ? if no warmer day append 0

Solution :- We use Monotonic Stack (decreasing) . we stores indices in the stack and pop till the temp lower than stack . if stack is not empty we add the stack[-1] – i . we initalise the array w 0s

- **62 :-** online stock span . span is the number days including itself where the stock price was same or lower (previous days)

Solution :- again montonic stack decreasing but we store the span as well in the stack . when we encounter a higher price , we pop till its lower and add the spans and store it .

- **73 :-** set matrix zeros in place . if 0 change the row and col to 0

Solution :- for o(1) space u keep the first row and col as flag and if any 0 is encountered u flag it . before this u see if any 0 is there in first and last row

- **118:-** pascal's triangle . return in the form[[1][1,1][1,2,1]]..

Solution :- don't overcomplicate have an arr = [1]*(i+1). Then j = the formula from the res arr directly

- **31:-** next permutation of a list in its lexicographically sorted order

Solution :- find the breaking point from right where a[i]<a[i+1]. Then swap it with the smallest element bigger than itself . then reverse the array after the breaking point(making it sorted)

- **53:-** max sum in contingous array

Solution :- kadanes algo . store maxSum every iteration . if currSum goes negative reset to 0

- **75:-** sort the 3 values 0,1,2 in place

Solution :- use two pointers l=-1 and r =len(n) . if 0 swap and increment curr but when swapping 2 to r don't increment curr as the swapped need not be 1

- **121:-** best time to buy and sell 1. U can buy and sell only once

Solution :- kadanes algo . have a lowest and store max each time . if u get a new lower update lower.

- **48:-** rotate the array by 90 deg

Solution :- transpose + reverse = rotate by 90 . transpose(i,j = j,i) can be optimized by pattern tracking where i goes from 0,n-2 and j from i+1,n ignoring diagnol ele

- **56:-** merge intervals

Solution :- add first ele to res array and check with that directly . merge them directly . sort based on starting . edge case is handled by taking max of endpoint while storing

- **88:-** merge sorted array . nums1 array as empty space which should be filled(in the end)

Solution :- 3pointers we use to track empty , actual num1 and num2 all from end. We use conditions and fill resp

- **287:-** find the only duplicate number in the array and all numbers are within 1..n

Solution :- we use floyd's tortoise hare algo. we look at arr as a linked list with the value as the index of next ele . we use slow and fast pointers as nums[slow] and nums[nums[fast]] . when they become equal we break and then another slow2 pointer starting from nums[0] . when they become equal return slow

- **74:-** searching a sorted 2d matrix

Solution :- binary search w l = 0 r = (m*n) -1 . property is row = mid//n(col) and col = mid % n .

- **50:-** pow[x,n]

Solution :- the idea is 2^10 = (2*2)^5 and 2^9=2*2^8 . when even x = x *x n=n/2 and when odd we store it in res by res = res * x , n = n -1 . loop till n is > 0

- **169 :-** majority ele that comes more than n/2 times

Solution :- moore voting algo . keep an count and majority . update majority if count = 0 . if num = majority increase count else decrease

- **229:-** majority element > n/3 . all the elements

Solution :- there can be atmost 2 ele > n/2 . do voting algo for 2 ele . in the end count and return only if the num1 and num2 appear more than > n/2.edge case also check if num2 not in res alr(repeating)

- **18:-** 4sum/ksum

Solution :- the idea is to do 2sum with two pointers and anything above we simply add another for loop. We use recursion and each for loop we do start till n-k+1 as we need atleast k elements after the current loop to form the array. The idea is once we sort the array we can eliminate duplicates by just skipping when nums[i]==nums[i-1] and when i > start (i is not start) . base case of recursion is when k ==2 then we implement 2sum . once we get a result we append it it to main results array and simply continue the loop again ignoring duplicates (change only left pointer)

- **128:-** longest consecutive seq in o(N)

Solution :- we use a hash set and we find the starting of the sequence by checking if n-1 exists . if it doesn't we calc length through a while loop and store max

- **73 :-** subarrays starting w sum k.

Solution :- similar to tree q. we calculate prefix sum and side by side add to res and check if pathsum – target exists . then we add to hash[prefixsum]+=1 . important to initalise defaultdict(int) to not get key error.

- **3583:-** count special triplets . num[i]=num[k]=num[j]*2

Solution :- we use two hashmaps for prefix and suffix(this filled withs 1). For each j , find the number of ele == nums[j]*2 , to the left and right of j . for each j add prefix[target]*suffix[target] . fill suffix in one pass, then do the main pass.

- **237:-** delete the node given only the node not even the head (o(1) to delete)

Solution :- take next node's val and delete the next node

- **160:-** finding intersection of two Linked lists

Solution :- the idea is that each head will go till the end of its linked list and will be swapped to start from the other list after null . now when the heads intersect we break and return l1 or l2

- **25:-** reverse K nodes in linked list

Solution :- calculate length first and get len//n and reverse those many groups . we need 2 pointers
grp_prev.next = prev
grp_start.next = curr
grp_prev = grp_start
group start is curr before reversing prev is None

- **138:-** deep copy of list with random pointer.

Solution :- o(n) space – we use hashmap to store all the nodes and value alone without pointers first[hash[node] = Node(curr.val)] . then we get the node and connect it to random by getting from hash like hash[curr].next = hash.get(curr.next) and hash[curr].random = hash.get(curr.random).

O(1) space – we take a copy of each node and put it in between resp nodes instead of putting in hash . then we assign and connect random variables in 2nd pass . then we disconnect and connect through a dummy variable as the new head. 3 passes

- **455:-** two arrays cookie size and child's greed . how many children can be satisfied

Solution :- have a left pointer . for each cookie check if the smallest child can be satisfied , if he can go to the next child . sort both the arrays

- **78:-** power set . give every subset

Solution :- decision tree involves either picking each index or not in each level (i.e index) . also its backtracking so remove once u append .

- **90:-** power set . but vals in array could repeat.

Solution :- again its backtracking but not traditional . we sort the array and in the function we first append temp[:] and then iterate from that element till the end of the array and append and call the fn . then we also pop for backtracking.in the loop if the duplicate we ignore

- **39:-** combination sum 1 . all combinations which sum to target . each number can be repeated how many ever times

Solution :- here the decision tree is at each level go through every index till end including that index (not i+1 while calling backtrack in loop) we track of remaining sum and base case is either index == n or target < 0 or target ==0

- **131:-** palindrome partitioning

Solution :- the decision tree is we take an we take the string starting at each index and ending at every forward index .

```
for end in range(start+1, n+1):
        if isPalin(s[start:end]):
          temp.append(s[start:end])
         backtrack(end, temp)
```

- **60:- Kth permutation sequence of n numbers(from 1 till n)**

Solution :- recursive needs practice as well but greedy is tricky .first k-=1 as 0 indexing . if we notice the permutations for n numbers if we fix 1 number we have n-1! Choices we calc k/n-1! And choose that index of the n-1 numbers and then k %= n-1! . repeat it till n = 0

- **51:- N queens problem . in nxn matrix where can u place n queens return all**

Solution :- check the if queen can be placed in which col in each row (it has to be in diff rows for sure) . 3 sets . posDiag has constant i+j , neg has i-j . store these two sums in their sets and also a col set . now backtrack through all 4 options.

- **37:- Sudokku solver . give the unique solution**

Solution :- have 3 defaultdict(set) for row, col and boxes . access box index by row//3 * 3 + col//3 . now backtrack() go through every cell . and in each cell try all 9 numbers . if any one gives True then return true , else return false . outside the 3rd (number for loop) loop return false . no parameters no nothing . add and remove from hashes as well like a normal backtracking

- **540:- find the single element in a sorted array of pairs o(logn)**

Solution :- binary search (finding pivot way) . every pair to the left will start at even index and end at odd vice versa to the right . to implement we decrement mid if odd and then if num[mid]==num[mid+1] we move left = mid + 2 else right = mid .

- **33:- search in a rotated sorted array rotated at a pivot point o(logn)**

Solution :- binary search (normal way) . at every ele either left is sorted or right is sorted (2 outer if cases) . (now these are 2 inner if cases)first if nums[l] <= nums[mid] then we see if target comes in between the l and mid and move pointer accordingly . similary

- **4:- median of two sorted arrays in o(log(m+n))**

Solution :- first copy the smaller array into A and the other to B . the we binary search 0 to len(A) – 1 . we calc partition as the half of total sum of lengths and we take all the lengths possible from A (lets say half is 5 one case is first 2 ele from A as next 3 from B then then next 2 and next 3 from B). If Aleft <= Bright and Aright >= Bleft then we have the right partition return min(bleft,bright) if total is odd else avg of max(aleft,aright) and min(bleft,bright) . other if case is Aleft >= Bright move r = i - 1 else l= i + 1 . i is the mid value while j is half – i – 2. Aleft and aright are I and i+1 , similary j and j + 1

- **295 :- median of Data stream . a function to add num and find median**

Solution :- 2 heaps to store small and large . small is max heap and large is min heap with diff with their diff in len always within 1 . by default we add to small heap . then we see if the max of small is > min of large then we swap and then we check the length condition and again swap . answer is in root of the larger heap or the average of two same len heaps

- **347:- K most freq elements in array . return all the unique k most fre elements**

Solution :- nlogn – use counter and add to maxheap and pop k times
Nlogk – maintain a k length min heap

- **232:- Queue using stack in armotized o(1) . average o(1) basically**

Solution :- instead of copying s2 back to s1 we just let it be there. When first check if s2 is empty then pop till s1 is empty . then return s2.pop(). By default always push to s1 . use similar in finding top too

- **496:- given two arrays num1 and nums2 . fill the next greater element of each element in nums1 based on nums2**

Solution :- we use a hash to store the index to store the res in from nums1 . now we use decreasing monotonic stack in nums . if stack is there and num > stack[-1] i.e greater then we keep popping . outside this while we put append . while stack and num > stack[-1] . (pop only till its greater not fully)

- **146:-** implement a LRU cache . need a put function , get function and a cap in constructor. If it goes above capacity discard least recently used key.

Solution :- we use a hash map with values pointing to a node . there is a doubly linked list with two dummy nodes left(least rec used) and right(recently) . two helper functions insert(inserting at right) and remove(from anywhere) . whenever u use a node use remove and then insert . node is a separate class . in the contructor connect left and right (they have next and prev properties , key and value as well)

- **460:- LFU cache . least frequently used .**

Solution :- use 3 dictionarys , one variable lfuCount to track lowest freq . dicts key to count(to handle freq) , key to val(to retrieve val in o(1)) and count to list(to handle and use LRU for multiple keys with same freq) . each count points to a DLL which has its own popleft, insertright , length functions . its contructor also has a map which maps all the keys to the node just LRU quedtion .

- **84:- max area rectangle based on bar heights.**

Solution :- add each bar to stack (after popping). it's a montonic increasing stack . if the curr ele is smaller than top , then that is the max height of top and start popping while this is true . inside change the width alone . visualize this question that's the only way to understand . we also append 0 to heights at the start to make sure all the elements are popped even at the end

- **239:- res.append the max ele in each window of size k sliding through array**

Solution :-
O(nlogk) solution :- we can maintain a max heap with index and nums . we go through each and first pop while the top ele is out of bounds . then we append the max ele to res

O(n) solution :- we use a montonically decreasing queue of indices . again pop till the top is decreasing . then append to queue, check and pop if out of bounds, then leftmost is appended to res.

- 155:- implement a min stack in o(1) . .pop() should give normal value but .getMin gives the min val.

Solution :- we store another stack (not montonic) and we compare and see if it smaller than min and add that to min stack . when popping pop both .

- 14:- longest common substring in all words

Solution :- sort the array and compare only the last 2 words .

- 686:- how many times do u repeat string a for string b to be a substring of a . if not return -1

Solution :- normal implementation is count = len(b) / len(a) + 2 . and the loop till count and keep adding a and checking if it's a substring

Rabin karp algo :- HAVE NOT IMPLEMENTED . the idea is to have a rolling hash function which check every (window of size of smaller string) and check if hash of the desired string is matching with window . if match has occurred then return . while rolling we subtract hash of first letter and add that of hash . hashed using h=0 and for ch in s : h = (h * base + ord(char)) % mod .

- 94:- tree traversal inorder

Solution :- Iterative approach :- take a stack and curr = root . while curr or stack . another while loop keep adding the left to stack. and then curr = pop , append and go right.

Morris inorder traversal :- threaded binary tree . 2 cases if no left node , its considered root and appended . else , use a new pointer and go to the rightmost node or till the right is not root itself. point it to the root if its none else remove the connection , append root and go right of root

Morris preorder traversal :- same code but instead of adding after going back from tread , we add when we connect the thread and directly go right when we remove . one line change

- 987:- vertical order of BT (from min col number to max col number all elemtents. Left is x+1,y-1 and right is x+1,y+1 . x is row , y is col)

Solution :- used a hash of lists and append for each column the (row, val) . then for sorted(hash.keys()) go through sorted(hash[col], key=lambda x: (x[0],x[1])) i.e sort based on row first then val . finally add only val to result through res.append([val for x, val in sorted_cols]) . col is calculates by row+1,col-1 and row+1, col+1 . this relative col spacing method is used when we don't care abt null spacing and stuff eg:-bottom view, top view, vertical order etc

- 662:- Maximum Width of Binary Tree based on index

Solution :- do bfs and in each while iteration calc len of q and iterate len times and adding to q normally . append ind to q , index of left Is 2*ind and right is 2*ind + 1 mimicking array representation . store last = index in for loop and first = q[0]'s index before for . store max = max(sum, last – first + 1) . this index simulates a complete binary tree and works with null spacing and stuff.

- 543:- diameter of a BT . the longest path between 2 nodes not necessarily through root node

Solution :- the idea is to calc height of left and right subtrees and then updating maxSum . for every node . have to do it in one pass o(n) . for that we do DFS first[left = dfs(root.left), right = dfs(root.right)] , update[max(sum, left + right)] and then return max(left, right) + 1 for height calc . THIS SINGLE PASS POSTORDER DFS TRAVERSAL IMPLEMENTATION IS IMPORTANT WHEN U WANT TO CONSIDER HEIGHT OF EACH SUBTREE FOR EVERY ROOT . even used for AVL check, LCA question

- 100:- same tree

Solution :- we do inorder/ any traversal and check each node. Same function . base case is if not p and if not q , return True , the other case is if not p or not q or not equal return False . now we use our False/true format . if not dfs(root.left): return False and sam for root.right and finally return True if it passes all above

- **124:- maxSum of any path not necessarily passing through root**

Solution :- we use one pass postorder DFS traversal method . but when storing left and right , we ignore negative sums by max(0,recur) .

- **105:- construct binary tree from inorder and preorder lists**

Solution :- idea :- first node of preorder is the root . for the next nodes , we check the index as mid in inorder list and then 1:mid+1 is the left subtree and mid+1:end is the right subtree . we can call this directly in main function passing preorder[1:mid+1], inorder[:mid] for left and opposite for right . finally return root

Non direct recursive o(n) :- we store index of each value in inorder in hash . we pass start and end in a helper . if start > end return None else calc mid , left = helper(start, mid+1) right helper(mid+1, end) . return root

- **106:- construct binary tree from inorder and postorder lists**

Solution :- idea :- last node of preorder is the root . for the next nodes , we check the index as mid in inorder list and then mid+1: is the right subtree and :mid is the left subtree . we pass postorder as it is . we compute right subtree first

Non direct recursive o(n) :- similar just use l,r . a builder function and a hash . right = mid+1,r , left = l, mid-1.

- **114:- flatten a binary tree into a linked list in place**

Solution :- similar to morris traversal (recollect) . if left child exists , find the preorder successor(right most of left subtree) and connect it to the curr node's right susbtree . then curr.right = curr.left , curr.left = None
and finally repeat outside if condition by curr = curr.right

- **116:-** each node has a next pointer which should point to the node right of it . populate these in o(1)

  Solution :- we maintain 2 pointers curr pointing to root and nxt pointing to curr.left . while curr and nxt , we link left child and right child. If curr.next is alr connected , we connect the curr's right child to nxt's leftchild . now move curr = curr.nxt and if not curr we move to the next level updating curr and nxt

- **108:-** convert sorted array to a height balanced BST

  Solution :- the idea is have 2 l,r . calc mid and make that the root .for left consider the left range i.e helper(l,mid-1) and right is helper(mid+1,r). return root. Base case is l > r return None i.e either r has gone too below(for left) or l has gone too above(for right)

- **98:-** validate a BST

  Solution :- we send low and high range . if not root return true , if not in range false , now return left subtree and right subtree

- **235:-** LCA of a BST in o(h)

  Solution :- here we just use a while loop and see if both p and q are lesser or bigger than root moving left and right resp . when both fails we return root

- **230:-** find kth smallest ele in BST . how to optimize if u do a lot of insertions and deletions?

  Solution :- normal is kth ele in inorder as its sorted . to optimize we use the idea that if the left subtree contains k-1 elemnts , then curr node is the answer . we add a new prop to the node leftcount and store there . while searching if leftCount = k+1 , return node , if k is smaller than leftcount go left else go right with (k – left_count – 1) . recursion this is . DON'T UNDERSTAND WHY THIS HELPS

- **653:-** two sum in BST

Solution :- see how to do in one pass . preorder dfs traversal , check if target – root.val exists if it does return true , else add to hash and return dfs.left or dfs.right for left and right subtrees

- **1373:- Max sum of a BST in a BT**

Solution :- similar to bottom up dp done through postorder dfs . but we leftIsBst, leftMin, leftMax, leftSum and similar for right for every every node . check if both BST are valid and root.val comes between left max and right min. if it does return True, min(leftMin, val), max(rightMax, val), newSum) . else return False,0,0,0 .

- **297:- Serialize and deserialize BT**

Solution :- do a normal postorder traversal and store nulls as N and serialize . when deserializing we use self.index to do postorder again . if N , we return null increment index . else we create a root , increment index and root.left = postorder()and root.right=postorder() . no arguments no nothing . we keep building left till null and then we build right . POSTRODER DFS again.

- **133:- clone a graph .**

Solution :- we store an oldToNew hash . we do dfs . if clone alr exists return , else create a clone and for each neighbor append the recursive call . basically if clone does exist we simply connect it , else we make the clone and connect the new clones nei , finally that will also get connected eventually apparently . finally return copy

- **207:- DFS cycle detection Directed**

Solution :- we have a visited array with 0s . 1 means visited in current dfs , 2 is visited alr in another iteration without cycle . do normal dfs , 1 means cycle is detected 2 means no cycle directly return true . now do dfs for every node in visited array.

Topological sort /BFS Cycle detection:- calc indegrees and add all nodes w indegree 0 to queue . now iterate through the neigbhors , reduce their indegree . if indegree is 0 append to q . if cycle is there , we compare length nodes and res
DFS undirected :- we pass parent as well . root has a parent of -1 . if the nei is not parent and unvisited , we visited and continue . else if alr visited in the same path we return False . then we backtrack and remove visited

BFS undirected :- we pass parent . here we check don't backtrack but same concept .

Topological sort DFS(without cycle detection) :- the idea is to go to the last node u can reach , add that to stack and prev ones in the opp order. We iterate through a visited array , if visited is 0 call dfs . in dfs append to stack after calling dfs for all its neighbhors . this wont work the same way as bfs for cycle detection with just length check .

- **785:-** is the graph bipartite

Solution :- idea is if we can color the graph w 2 colors and no 2 adjacent nodes have same color its bipartite . also if a graph has odd length cycle it cant be bipartite

BFS :- we have a colors array instead of visited with -1s . do bfs and color every adjacent array with not of parent if not colored . if already colored check if it's the same color then return false . if it passes bfs return true . do it for all unconnected components by iterating trough color array finally

- **743:-** network delay time . time for the network to reach all nodes

Solution :- DIJKSTRA's ALGO :- . we maintain a min Heap , Distance array(inf) and a parent array . first push into heap (0, source) . then pop smallest ele , go to its neighbors and check if dist val is lesser . if its lesser update parent , dist and push to heap . do this till heap get empty . we can backtrack and find path from parent array if needed .

- **684:-** redundant connection . return the connection which makes that tree cyclic/graph

Solution :- UNION FIND ALGO :- we store parent array and rank array(num of elements in parent) . parent initialized to itself . find is while p != parent[p] we keep going . we also compress by parent[p] = parent[parent[p]]. Union is finding parent of both nodes , and the higher ranked node becomes parent , increases its rank

In this q we check when the parent is equal and return false if they are .

- **1584:-** MIN SPANNING TREE PROBLEM(min to reach all nodes without cycle)

Solution :- KRUSKAL's alg : we use DSU but the diff is we sort the array with distance or wtv parameter given and the try to do union . if union returns false we simple skip it and continue .

- **152:-** max product subarray .

Solution :- currMax = max(currMin* num, currMax*num, num)
currMin = min(currMin*num, temp*num, num) . initialize res = max(nums).

- **300 :-** Longest Increasing Subsequence

Solution :- DP approach – 1 D dp LIS starting at each ele is 1. Go from last node and calc the LIS till the end of the array . 2 loops . max(lis[i], lis[j]+1) if num[i]<j

- **332 :-** coin change. U can repeat coins, return min number

Solution :- 1D bottom up dp storing 0 to target initialized to target+1(max number of coins). Memo[0]=0. We for each target from 0, target we check each coin and if target-coin > 0 , we put memo[i] = min(memo[target], 1+memo[target-coin])

- **332 :-** 0/1 knapsack problem . ur given a val array, wt array ,target W .

Solution :- Recursion :- we track index and W . base case is if index =0 and wt[ind] < W . then return val else return 0. Then we have 2 options notpick=ind+1,W or pick=0. If wt< W, then we update pick by recursive call . finally return max(pick and notpick)

Memoization :- just use a dict to store . its top down dp
Bottom up DP:- Steps to follow
1.see the indexing in recursion from top to bottom for both and reverse and put as loops
2. fill base cases in memo[][]
3. copy paste the things u do inside recursion and update calls to memo[][]

O(2*n) Space optimization : each value depends only on the prev row of the memo[][] . so just store prev and curr . base cases filled in prev . store the result in curr , after the inner for loop put prev = curr[:]

O(n) Space optimization : when we fill from right to left in the second loop , it still doesn't make a diff . but now we wouldn't need prev arr as the righ elements depend only on the left elements of prev arr in the last implementation . so just change loop conditions and use only prev.

[for unbounded knapsack we just stay at the index while picking]
Follow leetcode 416 for an idea.

- **139 :-** word break. Ur given a string and list of words . see if the string can be broken perfectly

Solution :- recursion :- we check if every index starting at i and ending at every index j 0...n-1 are in wordSet and call recur(end)i.e starting at that end . if both these conditions return true then true . base case is start == len(s) . if the loop runs out return false
Memoization :- use a memodict to store starting points . if it returns true store true , else after loop store false . check before for loops

Bottom up dp :- 1 d dp . dp[0] = true , we cant break a string ending at 0. Use 2 loops I 1..n+1 and j 0..i . if dp ending at j can be broken and s[j:i] in wordset return true and break

- **421 :-** Max XOR of 2 numbers in a array

Solution :- the idea is to use a trie which include all numbers in the array in its binary format upto the max number of bits i.e max(num).bit_length() . trie class has a normal insert func but doesn't need flag . it has a get_max(num) func

which goes through each bit in num, checks if opposite bit(1-bit) is present if so update max else just move on . no in main add each num to try and then call the func for each num and store max . implement trie from 31st bit to 0th bit.

- **1707 :- max XOR of each query [x,m] max xor of number x with domain of numbers in the array lesser or equal to m**

Solution :- same trie implementation but now we zip the original index to all queries and sort it according to m . nums is also sorted . now at each query add all nums lesser than m and getmax and store .