**Striver Sheet Leetcode and Geeks for Geeks :**

**\*\* Striver Sheet \*\***

**Rules to be followed**

**https://www.youtube.com/watch?v=MDomi07XR8Q**

**Leetcode people :**

**https://docs.google.com/spreadsheets/d/1WXK7Mdg-EQpmLP0KT-A74WCKeO-rTj\_hmwOly6nBxXs/edit#gid=0**

**gfg\_old = 7; // for 1 March feb 2022**

**gfg\_curr\_counter = 11; // 31march 2022**

**gfg\_total =22**

**Pro level :**

**Easy : 5 min**

**Medium : 20 min**

**Hard : ?**

**Current Level :**

**Easy : 15 min**

**Medium : 30 min**

**Hard : ?**

Day 1 :

**1.Set matrix Zeroes :**

(first Inner 2d then check the outer

first row and first col)

**2.Pascal Triangle**

(Array ArrayList : arr[i-1] + arr[i] logic )

**3.Next Permutation :**

(find i from the increasing subarray from the last , find j from the last ele

where arr[i] < arr[j] , swap arr[i] and arr[j], now reverse arr[i+1] till arr[n-1])

**4.Kadane's algo Maximum Subarray :**

(Initially sum =0 , max = arr[0] ,

if Sum<0, then make sum = 0 , now from i=1 to n ,sum+= arr[i] then compete the max ,sum )

**5.Sort array of 0's 1's 2's :**

(i=0 j =0 k =n-1,

while(i<=k) if arr[i] ==1 i++ , arr[i]==0 swap(arr[i],arr[j]) i++,j++

,arr[i]==2 swap(arr[i],arr[k])k-- simple do this )

**6.Buy and Sell :**

(It uses the beautiful application of kadane's logic )

**2.Day 2 :**

**7.Rotate Image :**

(1.Transpose 2. reverse by row)

**8.Merge Intervals :**

(start and end use and a condition where s <= end then update else add the range)

**9.Merge two sorted array :**

(tail approach simply better start from the last ele to first)

**10.Duplicate ele :**

(Use 2 pointers approach , tortoise and hare algo)

**11.Missing number :**

(here we use XOR operator approach bcz a^a = 0)

**12.Global and local Inversion :**

(This is the question where the smartness comes into the play... :)

Just find the 1 counter e.g where it can't return false )

**13.Search in 2d matrix :**

(Start from the top right corner and start traversal according to the

constraints given )

**14.Pow(x,n)**

(Beware of the condition of the negative n :

if (n<0) return 1/x \* myPow(1/x, -(n+1)))

**15. Majority ele n>2 :**

(We have use the Boyre Moore majority algo)

**16.Majority ele n>3 :**

(We have used the Boyre Moore majority algo for finding the atmost

2 number if possible)

**17.Unique paths :**

(Learned new way of swapping and how not to use

factorial function instead use mul and div with a for loop)

**18.2 Sum problem :**

(We have used the hashmap concept , remaning part and all)

**19. 4 sum problem :**

(O(n3 solution)

2 for loop of i and j , and a 2 pointer approach from j+1 to n-1)

**20.Longest consecuitve sequence :**

(He have to use the hashset and consecutive logic

set.remove(num)

int val = num

int sum =1

while(set.remove(val-1) ) val--

sum += num -val

val = num

while(set.remove(val+1)) val++

sum += val - num

if(sum>max) update max

**21.Longest subarray with 0 sum :**

(Logic is to store the prfix sum in hashmap key as sum and val as index

if you get the same prefix sum, j - i is a subarray with sum = 0 )

**22.Count number of subarrays with the given xor** :

(We have use the hashmap + the startegy of counting the

xor ^ B count and adding it

)

**23.Longest Substring without repeat :**

(Pepcoding way using the hashmap and i j while loop with ==2 breaks

condition)

**Day 5 :**

**24.Reverse a LL**

**25.Middle of LL**

**26.Merge two sorted LL**

**27.Remove Nth node from back of LL**

**28.Add two numbers \*\* LL**

**29.Delete given node from LL**

**## Day 6 :**

**30.Find Intersection pt. of linked list :**

(first cal the len of both the linked list , move the pointer of the bigger one till

the len become same ,

once it is of same length, move both pointer at the same time if pointer is same at any tiem

we have got the intersection point)

**31.Detect a cycle :**

(Take a slow and fast,

run the fast by 2 and slow by 1 , if any time slow == fast

return true we have found the cycle)

**32.Check if linked list is a palindrome or not :**

(go to the mid+1 address with slow, by moving fast ptr by 2,

once you reached mid+1 , fast = head, and reverse the right part of LL,

now start a while loop till slow!= null , if slow.val != fast.val

then not a pali else pali slow=slow.ptr and fast=fast.val )

**33.Reverse a Linked List k groups:**

(Recursive solution , curr =head , count = 0 globally

then a while loop while(curr != null && count!= k)curr=curr.next count++

now if(count == k){

curr = Recursion(curr,k)

while(count -- >0){

ListNode tmp = head.next

head.next =curr

curr = head

head = tmp

}

head = curr

}

return head )

**34.meeting node of a cycle in a linked list :**

We will use Floyd Cycle Algorithm :

https://leetcode.com/problems/linked-list-cycle-ii/discuss/1701055/JavaC%2B%2BPython-best-explanation-ever-happen's-for-this-problem

(move slow by 1and fast by 2, if they meet at any point

fast = head , slow ko wahi rhko aab , start a new while loop and just

move slow and fast by 1 , now the point where they we will meet is the

meeting node of the lined list

)

**35.\*\* Flatening the Linked List :**

( 24\*7 Innovation Labs Amazon Drishti-Soft

Flipkart Goldman Sachs Microsoft Paytm

Payu Qualcomm Snapdeal Visa )companies :

(It will be solve by the recursion, a very beautiful recursion )

(think to root.next = flatten(root.next)

and have to merge root = merge\_LL(root,root.next)

return root

think merge\_LL as the simple 2 Linked list now just have to merge it int a

single linked list )

**36. \*\* Rotate List :**

(make a dummy node ,

fast = dummy , slow = dummy

calculate the length ,len of LL

fast ko last node prr le jaoo , while cal len

then for j=len-k%len ; j>0 ; j--

slow = slow.next

// logic

//last node ko head se jodo

fast.next = head

// dummy.next ko slow.next krdo

dummy.next = slow.next

// slow ke next ko end bna do

slow.next = null

return dummy.next

)

**## Day 7 :**

**37.Random LL :**

(Have seen from the dicussion )

**38.3 sum : (Medium)**

(Striver you tube solution

take care of the duplicate and maintain i=0 to i<arr.length-2

in the inner loop

just run the 2 pointer approach by maintaining the sum = -arr[i]

if (arr[lo] + ar[hi ] == sum) then res.add(arr[i],arr[lo],arr[hi])

}

**39. Trapping running water : (Hard\*\*)**

(logic is water is added to the curr left building if the right building is

bigger than the current and also the max\_left building is also big

so these are the sufficient conditions to take a decision for the curr building

water += max\_li - height[li] ;

li++

and if the height[li] > height[ri ]

and height[ri] < max\_ ri

then only

water += max\_ri - height[ri]

else updation max\_ri = height[ri]

ri-- ;

**40.Remove duplicates :**

(2 pointer approach , maintain the curr and skip the index that have

duplicate, if not arr[count] = arr[i], count++ )

**41.Count consecutive ones :**

(initialize 2 things max =0 , curr = 0

// if u see a 0 make curr= 0 , else if u keep seeing 1,increase curr by 1

for (int n : arr){

max = Math.max(max, curr = n == 0 ? 0 : curr +1 ) ;

}

return max ; )

## Day 8 :

**42.#DC :**

4 sum II (454):

(To solve this

\*\*Brute force : TC :O(n^4), SC: O(1) have to run the 4 loops it will run all the

possibility that arr[i] + arr[j] + arr[k] + arr[l] == 0

\*\*Optimized : TC: O(n^2 \* logn) SC: O(n^2):

n^2\*logn for using HashMap for storing the sum of every pair arr[k]+arr[l]

and then after storing, we just run the 2 for loop of i and j

and check if, int sum = -(arr[i] + arr[j])

int ans = 0 ;

inside the 2 for loop

ans+=map.getOrDefault(sum,0) ;

return ans ;

)

**43.Merge k sorted linked list :**

(Divide and conquer just like the merge sort

, but here the merge is different

merge (LinkedList l1, LinkedList l2){

if (l1==null) return l2

if (l2==null) return l1

if(l1.val < l2.val){

// if this is small competition is between l1's next and l2

l1.next = merge(l1.next,l2);

return l1;

)

else {

l2.next = merge(l1,l2.next) ;

return l2 ;

}

**44.find the difference** :

(2 strings given s and t , one char is different ,

don't use the xor multiplication technique , instead use the

int conversion with s subtraction and t addition

int ans = (int) t.charAt(t.length()-1))

for (int i=0;i<s.length() ;i++){

ans -= (int)s.charAt(i);

ans -= (int)t.charAt(i);

}

return (char)ans ;

Greedy prob :

**45.Minimum Platforms :**

(Techinque is don't make the seperate class as

Activity question

Instead

Step 1 : Sort the arrays arrival arr, deperature dep

step 2 :We have to check the curr platform with the smallest departure prev

i=1 , j=0 ,n = # platforms, min\_plat =1 , ans = 1 ;

while(i <n && j<n){

if (arr[i] > dep[j]){

min\_plat -- ;

j++ ;

}

else if(arr[i] <= dep[j]){

min\_plat ++ ;

i++ ;

}

ans = Math.max(min\_plat,ans);

} )

}

**46.Job Sequencing :**

(Step 1 : sort the array wrt the profit

step2 : find the max deadline

step3 : make arr called res[max\_dead+1]

fill the res from 1 to max\_Dead = -1

int count = 0, tot\_profit = 0 ;

step4:for (int i =0 ; i<jobs; i++){

// j is for finding the suitable place to complete the job

for(int j =arr[i].deadline ; j>=1 ; j--){

if(res[j] == -1){

res[j] = i ;

count++ ;

tot\_profit += arr[i].profit;

}

}

return tot\_profit ; )

**47.(med)Subarray sum equal k :**

Brute force : O(n2) 2 for loops

Optimized : O(n) (Step1 :use the concept of frequency sum

STEP2 : MAKE A hashmap<int,int> of rem\_sum ,#times it appears before

map.put(0,1) ; // base case // k=target

int res = 0 , sum = 0 ;

for (int i =0 ; i<arr.length ; i++){

sum += arr[i] ;

int rem\_sum = sum - k ;

if(map.containsKey(rem\_sum) // agar rem\_sum hai toh ho bhi contigous hai){

res += map.get(rem\_sum); // previously kitni baar aaya hai ye

}

map.put(sum, map.getOrDefault(sum,0) + 1); // bcz sum is contiguous in iteslf

}

return res ;

)

**48.Minimum number of coins :**

(Step 1 :we have to use the dp

m : len of coins , V: Target value

make V size dp array

dp[0] and from dp[1] t0 dp[V] = Integer.MAX\_VALUE ;

Brute force is Exponential ,

Step 2 : O(Vm) TC

for (int i=1; i<=V;i++){

for(int j = 0 ; j<m ; j++){

if (coins[j] <= i ){

int cur\_res = i-coins[j] ; // check the prev coin and further check

if (cur\_res != Integer.MAX\_VALUE && cur\_res + 1 <dp[i]){

dp[i] = cur\_res +1 ;

}

}

}

if (dp[V] == Integer.MAX\_VALUE ) return -1 ;

else return dp[V]

}

)

**#Day 9 :**

**49.#DC :**

Permutation in a String :

Brute force : Sort both and check s1 length match return true nlogn

Optimized approach : use the freq contribution of s1 as + and s2 as -

(Step 1 : Sliding window and the frequency usage )

int [] count = new int [26] ;

(Step 2 add both s1 size , ele from s1 and s2 starting

then release a letter from starting of s2 as ++ and add one ele from i=len2

as -ve and everytime check if count's sum is 0 or not )

\*Recursion\* :

**50.#DC:**

Subsets :

https://leetcode.com/problems/subsets/discuss/27281/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning)

The Java String class compareTo() method compares the given

string with the current string lexicographically.

It returns a positive number, negative number, or 0.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

if s1 > s2, it returns positive number

if s1 < s2, it returns negative number

if s1 == s2, it returns 0

Step 1 :Sort the array nums

then call add\_recursive(li,new ArrayList<>(), nums, 0)

then in this add\_recursive we have

step2 : add\_rec(li,temp\_li,nums,st){

li.add(new ArrayList<>(temp\_li) ;

for(int i=start; i<nums.length ; i++){

temp\_li.add(nums[i]) ;

add\_rec(li,temp\_li,nums,i+1);

temp\_li.remove(temp\_li.size()-1) ;

})

**51.#DC**

Height of binary tree :

Recursive 2 calls in the left and right

return Math(left,right) + 1 ;

**52.Subset sum :**

Tc (2^N)

Optimized :

we let the recursion play here

\* consider the idx ele and add it to the sum and increase the index by 1

\* don't consider the ele and just increase the index

base case if idx == n

Al.add(sum)

sort(sum)

**53.Subset II (MED) :**

**Brute force :**

**Optimized :**

**#Day 10 :**

**54. Leetcode Q 51 (N Queens) Hard**

Brute force Recursion

**Step 1**

first fill the board array with the “.”

**Step 2**

We are filling the chess board column wise

So the recursion call will have an increase by 1 everytime which represent the column

**Step 3**

Lekr(board, 0, ans) this 0 denotes we are placing the right queen in the 0th column

Everytime in Lekr (board,col,ans) function we are checking for the every row that is this the valid position for the fixed column col, if yes place the queen and search for next col , if any time you are not able to have col == board.length this means we have to backtrack , utha lo queen ko auur new position prr rhko with keep in mind that all the queens are placed with the valid positions.

This means if are queens are placed with the valid\_move and col==board.length that means we have a found a solution in which we can place the queen

So now, we will add this solution to our main ans List of List and hence we will get all the answers in ans .

N queens over ☺

**55. Leetcode Q 111 (Minimum depth of Binary tree ) Easy**

**Brute force : 53 %**

**// using DFS**

public int minDepth(TreeNode root) {

if (root == null) return 0;

int left = minDepth(root.left) ;

int right = minDepth(root.right) ;

return (left ==0 || right == 0) ? left + right + 1 : Math.min(left,right) + 1 ;

**Explanantion :**

// last step is tricky and logical

// if any of the left or right is 0, we can't take minimum beacuse 0 will return everytime from the minimum and 1 will return

// that will be wrong, so we have two make two cases, if any of the left or right is , we say left + right + 1 and second case if not, then take min (left,right) then add 1

**Drawback**

of this approach is that we have to traverse all the nodes, of the binary tree,

To find the min depth can we do something better,

**Yes**

**// Better solution :**

**Link**

https://leetcode.com/problems/minimum-depth-of-binary-tree/discuss/905414/C%2B%2BJavaPython-BFS-DFS-Clean-and-Concise-O(N)

In O(n)

**Using BFS,**

 We iterate the tree level by level, and the first leaf we reach corresponds to the minimum depth. As a result, we do not need to iterate all nodes.

Trick is simple that we will use the queue and will traverse level order if we get any ele which is a leaf node we will return depth but if we getting the node whose left or right is there we are adding it in the queue after a while loop a level is completed keep doing depth+= 1.

This is the better way because once we get a node which is a leaf node, algo stops and we get the minimum depth automatically.

**56. Leetcode Q 36 (Valid Sudoku ) Medium**

Brute force

Step 1 : We run the for loop , for the board traversal

For every row we made a new 3 hashset, where one is row one is for column one is for 3\*3 cell.

If anyhow we get, same number in a row or in cell oe in 3\*3 cell that means we can’t solve this soduko hence return false .

But if after the whole traversal of the board, if we don’t get any false that means that we have to return true.

**57. Leetcode Q 36 (Valid Sudoku ) Medium**

In valid Sudoku, we are checking 3 things everytime we are facing ‘.’ Cell which means we have to take a decision.

2 loop outer : for rows

Initialize 3 hashset everytime, row ,col, cube

Inner loop on j col

Current board[i][j] != ‘.’ && (!row.add(board[i][j])) return false ;

// do this for col swap iandj pos and col.add

// for cell checking

Int st\_row = 3\*(i/3) ; int st\_col = 3\*(i%3) ;

if(board[st\_row + j/3][st\_col + j%3] != ‘.’ && (!cube.add(board[st\_row + j/3][st\_col = j%3])))

return false ;

and even if after the 2 for loop ends you don’t get the false

then return true that means we can solve the Sudoku.

**58. Leetcode Q 37 ( Sudoku Solver) Hard**

It can’t be put in words you have to see the code and feel it.

**59. Leetcode Q 144 (Preorder Traversal )**

**Brute force**

Root.val store kro

Then recursion call on left and then right and return ans

**Better Approach**

We will use the deque data structure

Because we can have our inbuilt object in this deque interface of java

**60. Allocate Books ( Binary Search Interview Bit)**

**Brute Force**

Check the different different partition

And TC O(n^2)

For each I we have to calculate the sum by traversing the array

**Optimal approach by Binary Search**

For each element I am choosing a valid sum which will divide it in the number of assigned students or not

If yes, we can further reduce our search space

So low = min ele, high = sum of the arr

If (allocation is possible){

Try to search the answer in big search space

High = mid - 1}

Else {

Low = mid + 1

}

**61. Allocate Books**

**Brute force**

It is that for every partition check the sum and take the minimum sum of a partition that is possible.

This method will take O (n2)

**TC O(n^2)**

**SC O(1)**

**Optimized**

Can we do pattern partitioning yes, we have to think like shrinking the search space

The final answer will lie in the range of

Low = arr[0]

And high = sum of all ele of arr

We can think of a binary search shrinkning space

This is how we can do the best partitioning in log(n)

And for every partition, we can check whether this partition is valid or not in O(n)

Hence the outer loop is a binary search and inner loops runs in O(n)

So the

**TC : O(nlogn)**

**SC : O(1)**

**62.Aggressive Cows**

Place the cows in that way minimum distance between any two cows is maximum.

Largest min distance in all cases

**Brute force**

With Recursion, It can be exponential, in nature.

With the smart trick, gap = 1 then try to place the cow with the gap =1,

If possible try doing that with gap =2 … and so on,

Suppose at gap=n+1, you can’t place the cows that mean the answer is n,

**TC O(n2)**

Outer loop n, inner loop that validates that the cows can be placed in O(n)

**Optimized**

Better approach can be making a search space and try to find the gap with the binary search,

But the question is how to set the low, high?

Gap can be minimum 1 b/w 2 cows and maximum is given by

Arr[arr.length-1] – arr[0]

So low = 1, high = arr[arr.length-1] – arr[0]

Now apply the same thing outer loop logn and inner loop O(n) for validation that setting of cows can be possible with the current gap.

**TC O(nlogn)**

**63.Aggressive Cows**