WEEK 1 :-

DAY 1:-

Q1. class Solution {

    public void setZeroes(int[][] matrix) {

        int ar[]=new int[matrix.length];

        int b[]=new int[matrix[0].length];

        Arrays.fill(ar,0);

        Arrays.fill(b,0);

        for(int i=0;i<ar.length;i++)

        {

            for(int j=0;j<b.length;j++)

            {

                if(matrix[i][j]==0)

                {

                    ar[i]=-1;

                    b[j]=-1;

                }

            }

        }

        for(int i=0;i<ar.length;i++)

        {

            for(int j=0;j<b.length;j++)

            {

                if(ar[i]==-1||b[j]==-1)

                {

                    matrix[i][j]=0;

                }

            }

        }

    }

}

Q2. class Solution {

public List<List<Integer>> generate(int numRows) {

List<List<Integer>> res=new ArrayList<>();

List<Integer> prev=new ArrayList<>();

prev.add(1);

res.add(prev);

while(--numRows>0){

List<Integer> list=new ArrayList<>();

for(int i=0;i<prev.size()-1;i++){

list.add(prev.get(i)+prev.get(i+1));

}

list.add(1);

list.add(0,1);

res.add(new ArrayList<>(list));

prev=list;

}

return res;

}

}

Q3. class Solution {

public void nextPermutation(int[] nums) {

//find the firest decreasing seq from end 3 5 2 1 so 3 is 1st decreasing num

int i=nums.length-2;

while(i>=0 && nums[i]>=nums[i+1]){

i--;

}

if(i>=0){

//decreasing seq exists

//finding 1st greater number than nums[i] from end

int j=nums.length-1;

while(j>i && nums[i]>=nums[j]){

j--;

}

//swapping

int temp=nums[i];

nums[i]=nums[j];

nums[j]=temp;

}

//reversing after i

reverse(nums,i+1);

}

public void reverse(int nums[],int start){

int end=nums.length-1;

while(start<=end){

int temp=nums[start];

nums[start]=nums[end];

nums[end]=temp;

start++;

end--;

}

}

}

Q4. class Solution {

public int maxProfit(int[] prices) {

int curr=prices[0],maxProfit=0;

for(int i=1;i<prices.length;i++){

if(prices[i]>curr){

maxProfit=Math.max(maxProfit,prices[i]-curr);

}else{

curr=prices[i];

}

}

return maxProfit;

}

}

Q5. class Solution {

public void sortColors(int[] nums) {

int start=0,end=nums.length-1;

for(int i=0;i<=end;){

if(nums[i]==0){

nums[i++]=nums[start];

nums[start++]=0;

}else if(nums[i]==2){

nums[i]=nums[end];

nums[end--]=2;

}else{

i++;

}

}

}

}

Q6.

DAY 2.

1. /\*

Brute : Sort the array and check for i and (i+1)th idx if both are same then that is ans

Better : Use hashing

Best : Use linklist cycle method. Use a fast and slow pointer and keep moving slow by 1 step

and fast by 2 steps till they collide. Then move fast to 1st element. Now again start moving fast and slow by 1 step till they meet and point of collision is ans

\*/

class Solution {

public int findDuplicate(int[] nums) {

int slow=nums[0],fast=nums[0];

do{

slow=nums[slow];

fast=nums[nums[fast]];

}while(slow!=fast);

fast=nums[0];

while(slow!=fast){

slow=nums[slow];

fast=nums[fast];

}

return slow;

}

}

2.

class Solution {

public void rotate(int[][] matrix) {

//Rotate = Transpose + Reverse

//1. Transpose

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

for(int j=i; j<matrix[0].length; ++j){

int temp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = temp;

}

}

//2. Reverse

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

int start = 0, end = matrix.length - 1;

while(start < end){

int temp = matrix[i][start];

matrix[i][start] = matrix[i][end];

matrix[i][end] = temp;

start++;

--end;

}

}

}

}

Q3.

public class Solution {

public static long getInversions(long arr[], int n) {

// Write your code here.

return mergeSort(arr,new long[n],0,n-1);

}

public static long mergeSort(long arr[],long temp[],int left,int right){

int mid;

long inv\_cnt=0;

if(right>left){

mid=left+(right-left)/2;

//breaking the array

inv\_cnt+=mergeSort(arr,temp,left,mid);

inv\_cnt+=mergeSort(arr,temp,mid+1,right);

//merging

inv\_cnt+=merge(arr,temp,left,mid+1,right);

}

return inv\_cnt;

}

public static long merge(long arr[],long temp[],int left,int mid,int right){

//i=idx of left subarray

//j=idx of right subarray

//k=idx of merged array

int i=left,j=mid,k=left;

long inv\_cnt=0;

while((i<=mid-1) && (j<=right)){

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

//if left is smaller than right

temp[k++]=arr[i++];

}else{

temp[k++]=arr[j++];

//number of elements right of i upto mid in arr

inv\_cnt+=(mid-i);

}

}

while(i<=mid-1) temp[k++]=arr[i++];

while(j<=right) temp[k++]=arr[j++];

//copying back sorted val to original array

for(i=left;i<=right;i++){

arr[i]=temp[i];

}

return inv\_cnt;

}

}

Day 3.

Q1. class Solution {

public double myPow(double x, int n) {

double ans=1;

long nn=n;

if(nn<0) nn\*=-1;

while(nn>0){

if(nn%2==1){

ans\*=x;

nn--;

}else{

x\*=x;

nn/=2;

}

}

if(n<0) return (double)(1.0)/(double)(ans);

return ans;

}

}

Q2. class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

int i=0,j=matrix[0].length-1;

while(i<matrix.length && j>=0){

if(target==matrix[i][j]) return true;

if(target>matrix[i][j]){

i++;

}else{

j--;

}

}

return false;

}

}

Q3.

/\*

Brute: For each i check the condition

Best: Use merge sort approach.While merging i will point to 1st array and j to second array.

For each i move j till arr1[i]<2\*arr2[j] and add j+1 to answer as all indexed less than

j in arr2 will make pair with arr1[i]

\*/

class Solution {

public int reversePairs(int[] nums) {

return mergeSort(nums,0,nums.length-1);

}

public int mergeSort(int[] nums,int lo,int hi){

if(lo>=hi) return 0; //if there is only single element then no answer

int mid=lo+(hi-lo)/2;

int inv=mergeSort(nums,lo,mid);

inv+=mergeSort(nums,mid+1,hi);

inv+=merge(nums,lo,mid,hi);

return inv;

}

public int merge(int[] nums,int lo,int mid,int hi){

int cnt=0,j=mid+1;

//counting reverse pairs

//as numbers are sorted dont move j back to mid for each i

for(int i=lo;i<=mid;i++){

//checking the condition nums[i] > 2 \* nums[j] for each i

while(j<=hi && nums[i]>(2\*(long)nums[j])){

j++;

}

cnt+=(j-(mid+1));

}

//merging the arrays

ArrayList<Integer> temp=new ArrayList<>();

int left=lo,right=mid+1;

while(left<=mid && right<=hi){

if(nums[left]<=nums[right]){

temp.add(nums[left++]);

}else{

temp.add(nums[right++]);

}

}

while(left<=mid) temp.add(nums[left++]);

while(right<=hi) temp.add(nums[right++]);

for(int i=lo;i<=hi;i++){

nums[i]=temp.get(i-lo);

}

return cnt;

}

}

Day 4.

Q1. class Solution {

public int[] twoSum(int[] nums, int target) {

Map<Integer,Integer> map=new HashMap<>();

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

if(map.containsKey(target-nums[i])){

return new int[]{i,map.get(target-nums[i])};

}

map.put(nums[i],i);

}

return new int[]{};

}

}

Q2.

class Solution {

public int longestConsecutive(int[] nums) {

if(nums.length == 0) return 0;

TreeSet<Integer> set = new TreeSet<>();

// add all the elements in treeset so that element come in sorted order without duplicate

for (int num : nums) {

set.add(num);

}

int max = 1, prev = 0, count = 1;

// set prev to first element in the set

for(int el: set) {

prev = el;

break;

}

for(int el: set) {

// check the difference between current element and previous element,

//if it is one, increment count and update max value is applicable

if(el - prev == 1) {

count++;

max = Math.max(max, count);

} else {

count = 1;

}

prev = el;

}

return max;

}

}

Q4.

class Solution {

public int lengthOfLongestSubstring(String s) {

int left=0,right=0,maxlen=0;

HashSet<Character> set=new HashSet<>();

while(right<s.length()){

while(set.size()>0 && set.contains(s.charAt(right))){

set.remove(s.charAt(left));

left++;

}

set.add(s.charAt(right));

maxlen=Math.max(maxlen,set.size());

right++;

}

return maxlen;

}

}

Day 5.

/\*

Brute: Calculate size then iterate from i=1 to i=size-n then break the link

Best: Take a dummy that points to head.Take 2 pointers fast and slow and both pointing to head.

Move fast from i=1 to i<=n. Then move both fast and slow till fast points to last node.

Break link at slow and return dummy.next

\*/

class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

ListNode dummy=new ListNode(-1);

dummy.next=head;

ListNode fast=dummy,slow=dummy;

while(n-->0) fast=fast.next;

while(fast.next!=null){

fast=fast.next;

slow=slow.next;

}

slow.next=slow.next.next;

return dummy.next;

}

}

Q2.

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode addTwoNumbers(ListNode list1, ListNode list2) {

int carry=0;

ListNode ans=new ListNode(-1),dummy=ans;

while(list1!=null || list2!=null){

int sum=carry;

if(list1!=null){

sum+=list1.val;

list1=list1.next;

}

if(list2!=null){

sum+=list2.val;

list2=list2.next;

}

ans.next=new ListNode(sum%10);

ans=ans.next;

carry=sum/10;

}

if(carry>0){

ans.next=new ListNode(carry);

}

return dummy.next;

}

}

Q4.

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseList(ListNode head) {

ListNode prev=null;

while(head!=null){

ListNode next\_node=head.next;

head.next=prev;

prev=head;

head=next\_node;

}

return prev;

}

}

Q5.

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

class Solution {

public void deleteNode(ListNode node) {

node.val=node.next.val;

node.next=node.next.next;

}

}

Day 6.

Q1.

/\*\*

\* Definition for singly-linked list.

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public boolean hasCycle(ListNode head) {

ListNode fast=head,slow=head;

while(fast!=null && fast.next!=null){

fast=fast.next.next;

slow=slow.next;

if(fast==slow) return true;

}

return false;

}

}

Q2.

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

ListNode h1=headA,h2=headB;

while(h1!=h2){

if(h1==null){

h1=headB;

}else{

h1=h1.next;

}

if(h2==null){

h2=headA;

}else{

h2=h2.next;

}

}

return h1;

}

}

Q3.

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseKGroup(ListNode head, int k) {

int count=0;

ListNode dummy=new ListNode(-1);

dummy.next=head;

ListNode temp=dummy;

//calculating the size of list

while(temp.next!=null){

temp=temp.next;

count++;

}

temp=dummy;

while(temp.next!=null){

if(count<k) break;

int nodes=k-1;

ListNode tempNext=temp.next;

ListNode first=temp.next;

ListNode second=first.next;

//reversing the sublist of size k

while(nodes-->0){

ListNode next=second.next;

//reversing pointers of first and second then shifting them 1 step ahead

second.next=first;

first=second;

second=next;

}

count-=k;

temp.next=first;

tempNext.next=second;

temp=tempNext;

}

return dummy.next;

}

}