Index

1. Find One duplicate number from the array(1 to n-1)
2. Find more than one duplicates from array(0-n-1)
3. Sort array of 0, 1, 2
4. Repeating and Missing Element (1-n)
5. Merge two sorted Arrays without extra space
6. Largest Contiguous subarray
7. Merge Intervals
8. Set Matrix Zeros
9. Pascal Triangle
10. Next Permutation
11. Stock Buy and Sell
12. Rotate image clockwise
13. Excel sheet column number from String
14. Excel sheet String from the column number
15. Power of a Number
16. Count trailing zeros in factorial of a number
17. LCM and GCD of Two numbers
18. Grid Unique Paths
19. Two Sum Problem
20. Longest Consecutive Sequence
21. Longest subarray with 0 sum
22. Number of subarrays with sum as k
23. Reverse Linked List Recursively and Iteratively
24. Find middle of LinkedList
25. Merge two Sorted LinkedList
26. Remove Nth Node From End of List
27. Delete a given Node when a node is given. (0(1) solution)
28. Add two numbers as LinkedList
29. Find intersection point of Y LinkedList.
30. Check if the LinkedList is palindrome or not
31. Reverse a LinkedList in groups.
32. Check for Cycle in the Linked List
33. Detect and remove the cycle in the Linked List
34. Flattening of a LinkedList
35. Rotate Linked List
36. Clone a Linked List with random and next pointer

1. Find One duplicate number from the array(0 to n-1)

i. Use Brute Force t.c = n^2
ii. Use extra space t.c = n, s.c = n

iii.

```
public int findDuplicate(int[] nums) {

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

        do{
            slow = nums[slow];
            fast = nums[nums[fast]];
        }while(slow!=fast);

        slow = nums[0];
        while(slow!=fast){
            slow = nums[slow];
            fast = nums[fast];
        }
        return fast;
    }
```

t.c = n, s.c = 1.

2. Find more than one duplicates from array (0-n-1)

i. Use Brute force n^2, or use extra space s.c = n
ii. Since the elements are from 0 to n-1 change the a[a[i]] index to negative. If it is already negative then a[i] is repeating.

```
 public static ArrayList<Integer> duplicates(int a[], int n) {

          for(int i=0; i<n; i++)
          {
                    int x = Math.abs(a[i]);
                    if(a[x]>=0)
                              a[x] = -a[x];
                    else
                              ans.add(x);
          }
          if(ans.size()==0)
                    ans.add(-1);
          return ans;
}
```

t.c = n, s.c = 1
Problem is when 0 is present

iii. Instead of changing the number to negative add n to the a[a[i]]. Now we can say that is element is not repeated then we only add n once.

```
public static ArrayList<Integer> duplicates(int a[], int n) {
        ArrayList<Integer> ans = new ArrayList<>();

        for(int i=0;i<n;i++)
            a[a[i]%n] = a[a[i]%n]+n;

        for(int i=0;i<n;i++)
            if(a[i]>=n*2)
                ans.add(i);

        if(ans.size()==0)
            ans.add(-1);

        return ans;
    }
```

t.c = n, s.c = 1

3. Sort array of 0, 1, 2

i. Use Arrays.sort t.c = nlogn
ii. Use 3 count and 2 iterations. t.c =n
iii. Single iteration

```
  public void sortColors(int[] nums) {

        int l = 0, m = 0, h = nums.length-1;

        while(m<=h){
            if(nums[m]==0){
                nums[m] = nums[l];
                nums[l] = 0;
                l++;
                m++;
            }
            else if(nums[m]==1)
                m++;
            else{
                nums[m] = nums[h];
                nums[h] = 2;
                h--;
            }
        }
    }
```

t.c = n

4. Repeating and Missing Element (1-n)

i. Use sorting t.c = nlogn, s.c = 1
ii. Use Hash map t.c = n, s.c = n
iii. Use summation logic as

n(n+1)/2 - sum of array elements = A
==> Missing - Repeating = A ———————————>1

n(n+1)(2n+1)/6 - sum of Squares of array elements = B
==> Missing^2 - Repeating^2 = B
(Missing - Repeating)(Missing + Repeating) = B
Missing+ Repeating = B/A ———————————->2

On solving 1 and 2 we get answer.
t.c = n, s.c = 1
Problem is using squares so overflow may occur

iv. Use XOR logic

Find xor of all the elements of array and xor with 1-n
we get,
missing ^ repeating = A
here find the right most set bit and left shit 1 by that many times
(
int pos = 0;
int m = 1;
while((xor&m)==0)
{ m = m<<1; pos++;
}

number is 1<<pos;

One step to find the number if xor & ~(xor-1) (This gives 1<<pos)
)
Then separate the array elements into 2 buckets and find the missing and repeating elements.

```
int[] findTwoElement(int a[], int n) {

    int xor = a[0];

    for(int i=1;i<n;i++)
       xor = xor^a[i];

    for(int i=1;i<=n;i++)
       xor = xor^i;

    // int num = xor & ~(xor-1);
```

```
        int pos = setbit(xor);
        int num = 1<<pos;

        int ans[] = new int[2];

        int x=0,y=0;

        for(int i=0;i<n;i++){
            if((a[i] & num)!=0){
                x = x^a[i];
            }
            else{
                y = y^a[i];
            }
        }
        for(int i=1;i<=n;i++){
            if((i & num)!=0)
                x = x^i;
            else
                y = y^i;
        }
        for(int i=0;i<n;i++){
            if(a[i]==x){
                ans[0] = x;
                ans[1] = y;
                break;
            }
            else if(a[i]==y){
                ans[0] = y;
                ans[1] = x;
                break;
            }
        }
        return ans;
    }
    int setbit(int x){
        int pos = 0;
        int m=1;
        while((x&m)==0){
            m = m<<1;
            pos++;
        }
        return pos;
    }
```

5. Merge two sorted Arrays without extra space

i. Use extra space and It is simple
ii. Use insertion sort technique
iii. Use gap algorithm

```
int void nextGap(int gap)
{
	if(gap<=1)
		return 0;
	return gap/2+gap%2;
}

int merge(int a[], int b[], int n, int m)
{
	int gap = n+m;
	int i, j;

	for(gap = nextGap(gap); gap>0; gap = nextGap(gap))
	{
		for(i=0;i+gap<n; i++)
			if(a[i]>a[i+gap]){
				int t = a[i];
				a[i] = a[i+gap];
				a[i+gap] = t;
			}

		for(j = gap>n?gap-n:0 ; j<m && i<n; i++,j++)
			if(a[i]>b[j]){
				int t = a[i];
				a[i] = b[j];
				b[j] = t;
			}
		if(j<m)
		{
		    for(j=0;j+gap<m; j++)
			if(b[j]>b[j+gap]){
				int t = b[j];
				b[j] = b[j+gap];
				b[j+gap] = t;
			}
		}
	}
}
```

iv. Some sorting technique

```
public static void merge(int a[], int b[], int n, int m)
{
	int i=n-1,j=0;

	while (i>=0 && j<m) {
		if(a[i]>b[j] )
		{
			int temp=a[i];
			a[i]=b[j];
			b[j]=temp;
```

```
                        i--;
                }
                else {
                        j++;
                }
        }
        Arrays.sort(a);
        Arrays.sort(b);
        }
}
```

t.c = nlogn+ mlogm

6) Largest Contiguous subarray

```
int maxSubarraySum(int a[], int n){

        int cursum = 0;
        int max = Integer.MIN_VALUE;

        for(int i=0;i<n;i++){
            cursum+=a[i];
            if(max<cursum)
                max = cursum;
            if(cursum<0)
                cursum = 0;
        }
        return max;
    }
```

t.c = n, s.c = 1

7) Merge intervals

```
  class c implements Comparator<int[]>{
        public int compare(int o1[],int o2[]){
            if(o1[0]>o2[0])
                return 1;
            else if(o1[0]<o2[0])
                return -1;
            else
                return 0;
        }
    }
    public int[][] merge(int[][] intervals) {
        if(intervals.length<1)
            return intervals;

        Arrays.sort(intervals, new c());
        int i=0;
```

```
        for(int j=1;j<intervals.length ;j++){
            if(intervals[i][1]>=intervals[j][0])
                intervals[i][1] = Math.max(intervals[i][1],intervals[j][1]);
            else{
                i++;
                intervals[i][0] = intervals[j][0];
                intervals[i][1] = intervals[j][1];
            }
        }
        int answer[][] = new int[i+1][2];

        for(int k=0;k<i+1;k++){
            answer[k][0] = intervals[k][0];
            answer[k][1] = intervals[k][1];
        }
        return answer;
    }
```

t.c = nlogn
s.c = logn

8) Set Matrix Zeros

i. Use Extra space
ii. Use this approach

```
void booleanMatrix(int matrix[][])
    {
        boolean isCol = false;

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

            if(matrix[i][0]==0)
                isCol = true;

            for(int j=1;j<matrix[0].length ;j++){
                if(matrix[i][j]==0){
                    matrix[i][0] = 0;
                    matrix[0][j] = 0;
                }
            }
        }

        for(int i=1;i<matrix.length; i++)
            for(int j=1;j<matrix[0].length; j++)
                if(matrix[i][0]==0 || matrix[0][j]==0)
                    matrix[i][j] = 0;

        if(matrix[0][0]==0)
            for(int i=0;i<matrix[0].length; i++)
```

```
            matrix[0][i] = 0;

        if(isCol)
            for(int i=0;i<matrix.length; i++)
                matrix[i][0] = 0;
    }
```

t.c = n*m, s.c = 1

9) Pascal Triangle

Triangle
0c0
1c0 1c1
2c0 2c1 2c2
3c0 3c1 3c2 3c3

For 4th col
1, 1*(4-1)/1, 3*(4-2)/2 ..

```
public List<List<Integer>> generate(int num) {

        List<List<Integer>> answer = new ArrayList<>();
        for(int line=1; line<=num ;line++){
            int c = 1;
            List<Integer> l = new ArrayList<>();
            for(int i=1;i<=line ;i++){
                l.add(c);
                c = c*(line-i)/i;
            }
            answer.add(l);
        }
        return answer;
}
```

10) Next Permutation

```
public void nextPermutation(int[] nums) {
        if(nums.length<1)
            return;
        int i = nums.length-2;

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

        if(i>=0){
            int j = nums.length-1;
            while(nums[i]>=nums[j])
                j--;
            swap(nums ,i ,j);
        }
```

```
        reverse(nums, i+1 ,nums.length-1);
    }
   void swap(int a[],int i ,int j){
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
    void reverse(int a[],int i, int j){
        while(i<j)
            swap(a, i++,j--);
    }
```

t.c = n, s.c = 1

11) Stock Buy and Sell

```
ArrayList<ArrayList<Integer> > stockBuySell(int a[], int n) {
        ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

        int i=0;
        while(i<n){
            while(i<n-1 && a[i]>=a[i+1]){
                i++;
            }
            if(i==n-1)
                break;
            int buy = i;
            i++;
            while(i<n-1 && a[i]<=a[i+1])
                i++;

            int sell = i;
            ArrayList<Integer> t = new ArrayList<>();
            t.add(buy);
            t.add(sell);
            ans.add(t);
            i++;
        }
        return ans;
    }
```

12) Rotate image clockwise

```
public void rotate(int[][] a) {
        if(a.length<1)
            return;
        int n = a.length;
        for(int i=0;i<n;i++){
            for(int j=i+1;j<n; j++){
                int t = a[i][j];
```

```
                a[i][j] = a[j][i];
                a[j][i] = t;
            }
        }

        for(int i=0;i<n;i++){
            for(int j=0;j<n-j-1;j++){
                int t = a[i][j];
                a[i][j] = a[i][n-j-1];
                a[i][n-j-1] = t;
            }
        }
    }
```

t.c = n*m

13) Excel sheet column number from String

```
public int excelColumnNumber(String s) {

        int res = 0,i=0;

        while(i<s.length()){
            res = res*26;
            res = res+s.charAt(i)-'A'+1;
            i++;
        }
        return res;
}
```

t.c = s.length(), s.c = 1

14) Excel sheet String from the column number

```
 public String excelColumn(int n){

        StringBuffer ans = new StringBuffer();

        while(n>0){
            int res = n%26;

            if(res==0){
                ans.append("Z");
                n = n/26 - 1;
            }
            else{
                ans.append((char)((res-1)+'A'));
                n = n/26;
            }
        }
```

```
        return ans.reverse().toString();
    }
```

t.c = log(n)

15) Power of a number

Iterative

```
 long power(int n, int r)
    {
        long ans = 1;
        long mod = 1000000007;
        long m = n;
        while(r>0){

            if(r%2==1){
                ans = ((ans%mod)*(m%mod))%mod;
            }

            m = (m*m)%mod;
            r/=2;
        }
        return ans;
    }
```

Recursive

```
long m = 1000000007;
long raise(long t ,int r){
        if(r==1)
            return t;

        if(r%2!=0)
            return (t*(raise(t, r-1)))%m;
        return (raise(((t%m)*(t%m))%m ,r/2))%m;
    }
```

16) Count trailing zeros in factorial of a number

```
 public int trailingZeroes(int n) {

        int count=0;

        for(int i=5;n/i>=1;i*=5)
            count+=n/i;

        return count;
    }
```

17) LCM and GCD of two numbers

```
static long[] lcmAndGcd(Long a , Long b) {
        long ans[] = new Long[2];
        long hcf = gcd(a, b);
        long lcm = (a*b)/hcf;
        ans[1] = hcf;
        ans[0] = lcm;
        return ans;
    }
    static long gcd(long a, long b){
        if(b==0)
            return a;
        return gcd(b, a%b);
    }
```

t.c = log(Max(a ,b))

18) Grid Unique Paths

Recursive

```
long mod = 1000000007;
long uniquePaths(int m, int n)
{
        long dp[][] = new long[m+1][n+1];
        for(long r[]: dp)
                Arrays.fill(r,-1);
        return paths(m, n, dp);
}
long paths(int m, int n, long d[][])
{
        if(m==1 || n==1)
                return 1;

        if(dp[m][n]!=-1)
                return dp[m][n];

        dp[m][n] = (paths(m-1,n)%mod + paths(m,n-1)%mod)%mod;
        return dp[m][n];
}
```

t.c = n*m
s.c = n*m

19) Two Sum Problem

GFG(Just Check)

```
  boolean hasArrayTwoCandidates(int a[], int n, int x) {
```

```
        Arrays.sort(a);
        int i=0,j=n-1;

        while(i<j){
            if(a[i]+a[j]==x)
                return true;
            else if(a[i]+a[j]<x)
                i++;
            else
                j--;
        }
        return false;
    }
```

t.c = nlogn, s.c = 1

LeetCode(Return Indices)
So here I can't sort

```
 boolean hasArrayTwoCandidates(int a[], int n, int x) {

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

        for(int i=0;i<n;i++)
            map.put(a[i],i);

        for(int i=0;i<n;i++){
            if(map.containsKey(x-a[i]) && map.get(x-a[i])!=i)
                    return new int[]{map.get(c),i};
        }
        return false;
    }
```

t.c = n, s.c = n

20) Longest Consecutive Sequence

```
int LCS(int a[])
{
          if(a.length<1)
                    return 0;

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

          for(int i=0; i<a.length; i++)
                    set.add(a[i]);

          int c = 0;
          for(int i=0; i<a.length; i++)
```

```
        {
                int temp = 0;
                if(!set.contains(a[i]-1))
                {
                        int x = a[i];
                        while(set.contains(x))
                        {
                                temp++;
                                x++;
                        }
                }
                count = Math.max(count, temp);
        }
        return count;
}
```

t.c = ~n, s.c = n

21) Longest subarray with 0 sum

```
int maxLen(int a[], int n)
   {
      if(a.length<1)
         return 0;

      HashMap<Integer,Integer> map = new HashMap<>();
      int sum = 0;
      int ans = 0;
      for(int i=0;i<n;i++){
         sum+=a[i];
         if(sum==0)
            ans = i+1;
         if(map.containsKey(sum))
            ans = Math.max(ans, i-map.get(sum));
         else
            map.put(sum, i);
      }
      return ans;
   }
```

t.c = n, s.c = n

22) Number of subarrays with sum as k

```
public int subarraySum(int[] a, int k) {
      int c = 0,s=0;
      HashMap<Integer,Integer> map = new HashMap<>();
      map.put(0,1);

      for(int i=0;i<a.length ;i++){
         s+=a[i];
```

```
            if(map.containsKey(s-k))
                c = c+map.get(s-k);
            map.put(s, map.getOrDefault(s,0)+1);
        }
        return c;
    }
```

t.c = n, s.c = n

23) Reverse Linked List Recursively and Iteratively

Recursively

```
Node reverseList(Node head)
    {
        if(head==null || head.next==null)
            return head;

        Node temp = reverseList(head.next);
        head.next.next = head;
        head.next = null;
        return temp;
    }
```

t.c = n, s.c = 1

Iteratively

```
Node reverseList(Node head)
{
        if(head==null || head.next==null)
                return head;

        Node cur = head;
        Node prev = null;

        while(cur!=null)
        {
                head = head.next;
                cur.next = prev;
                prev = cur;
                cur = head;
        }
        return prev;
}
```

t.c = n, s.c = 1

24) Find middle of LinkedList

```
int getMiddle(Node head)
```

```
    {
        Node slow = head;
        Node fast = head;

        while(fast!=null && fast.next!=null){
            fast = fast.next.next;
            slow = slow.next;
        }
        return slow.data;
    }
```

t.c = n/2
s.c = 1

25) Merge two Sorted LinkedList

Recursive

```
Node sortedMerge(Node headA, Node headB) {

        if(headA==null)
            return headB;
        if(headB==null)
            return headA;
        if(headA.data<headB.data){
            headA.next = sortedMerge(headA.next, headB);
            return headA;
        }
        else{
            headB.next = sortedMerge(headA, headB.next);
            return headB;
        }
    }
```

t.c = n, s.c = 1

Iterative can be done easily

26) Remove Nth Node From End of List

```
 public ListNode removeNthFromEnd(ListNode head, int n) {
        if(head==null)
            return head;
        ListNode slow = head;
        ListNode fast = head;
        while(n-->0)
            fast = fast.next;
        if(fast==null)
            return head.next;

        while(fast.next!=null){
            slow = slow.next;
```

```
            fast = fast.next;
        }
        slow.next = slow.next.next;
        return head;
    }
```

t.c = n

27) Delete a given Node when a node is given. (0(1) solution)

```
void deleteNode(Node node)
    {
        node.data = node.next.data;
        node.next = node.next.next;
    }
```

t.c = 1, s.c = 1

28) Add two numbers as LinkedList

```
 public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(-1);
        ListNode tail = dummy;
        int c = 0;
        while(l1!=null || l2!=null){
            int sum = 0;
            if(l1==null){
                while(l2!=null){
                    sum = l2.val+c;
                    ListNode t = new ListNode(sum%10);
                    c = sum/10;
                    tail.next = t;
                    tail = t;
                    l2 = l2.next;
                }
                break;
            }
            else if(l2==null){
                while(l1!=null){
                    sum = l1.val+c;
                    ListNode t = new ListNode(sum%10);
                    c = sum/10;
                    tail.next = t;
                    tail = t;
                    l1 = l1.next;
                }
                break;
            }
            sum = l1.val + l2.val+c;
            ListNode t = new ListNode(sum%10);
            c = sum/10;
```

```
            tail.next = t;
            tail = t;
            l1 = l1.next;
            l2 = l2.next;
        }
        if(c!=0){
            ListNode t = new ListNode(c);
            tail.next = t;
            tail = t;
        }
        return dummy.next;
    }
```

t.c = Max(l1.size(), l2.size());
s.c = Max(l1.size(), l2.size());

29) Find intersection point of Y LinkedList

Return the integer data of the merging node. If no merging return -1

i. Use Two for loops and check each and every node of l1 with ever node of l2. **t.c = n*m, s.c = 1.**
ii. Use HashSet or HashMap so. Put every node of l1 into the set and while doing the same the node that is already present in the set is the answer. **t.c = n+m, s.c = n+m.**
iii.

```
int intersectionPoint(Node h1, Node h2)
{
        if(h1==null || h2==null)
                return -1;

        Node t1 = h1;
        Node t2 = h2;

        int c1=0, c2=0;

        while(t1!=null)
        {
                t1 = t1.next;
                c1++;
        }
        while(t2!=null)
        {
                t2 = t2.next;
                c2++;
        }
        t1 = h1;
        t2 = h2;

        if(c1>c2)
        {
                c1 = c1 - c2;
```

```
                while(c1- ->0)
                        t1 = t1.next;
        }
        else if(c2>c1)
        {
                c2 = c2 - c1;

                while(c2—>0)
                        t2 = t2.next;
        }

        while(t1!=null && t2!=null && t1!=t2)
        {
                t1 = t1.next;
                t2 = t2.next;
        }

        if(t1==null || t2==null)
                return -1;

        return t1.data;
}
```

t.c = n+m, s.c = 1.

30) Check if the LinkedList is palindrome or not

i. Use Stack to do this

```
 public boolean isPalindrome(ListNode head) {
       if(head==null || head.next==null)
          return true;

       Stack<Integer> stack = new Stack<>();
       ListNode temp = head;

       while(temp!=null){
          stack.add(temp.val);
          temp = temp.next;
       }

       temp = head;
       while(temp!=null){
          if(temp.val!=stack.pop())
             return false;
          temp = temp.next;
       }
       return true;
    }
```

t.c = n, s.c = n

ii. Breakdown the linked list into 2 halves

```
  public boolean isPalindrome(ListNode head) {
        if(head==null || head.next==null)
            return true;
        ListNode first = head;
        ListNode slow = head;
        ListNode fast = head;

        while(fast!=null && fast.next!=null){
            first = slow;
            slow = slow.next;
            fast = fast.next.next;
        }
        if(fast!=null)
            slow = slow.next;

        first.next = null;
        ListNode second = reverse(slow);
        first = head;

        while(first!=null && second!=null){
            if(first.val!=second.val)
                return false;
            first = first.next;
            second = second.next;
        }
        return true;
    }

    ListNode reverse(ListNode head){
        if(head==null || head.next==null)
            return head;

        ListNode ansNode = reverse(head.next);
        head.next.next = head;
        head.next = null;
        return ansNode;
    }
```

t.c = n,   s.c = 1

31) Reverse a LinkedList in groups.

```
public static Node reverse(Node node, int k)
    {
        if(node==null || node.next==null)
            return node;
```

```
        int c = 0;
        Node prev = null;
        Node cur = node;
        Node next = node;

        while(cur!=null && c!=k){
            next = next.next;
            cur.next = prev;
            prev = cur;
            cur = next;
            c++;
        }

        if(cur!=null)
            cur.next = reverse(cur, k);

        return prev;
    }
```

t.c = n, s.c = 1

32) Check for Cycle in the Linked List

```
  public ListNode detectCycle(ListNode head) {
        if(head==null || head.next==null)
            return null;
        ListNode slow = head;
        ListNode fast = head;

        do{
            slow = slow.next;
            fast = fast.next.next;
            if(fast==null || fast.next==null)
                return null;
        }while(slow!=fast);

        slow = head;
        while(slow!=fast){
            fast = fast.next;
            slow = slow.next;
        }
        return slow;
    }
```

t.c = n, s.c = 1

33) Detect and remove the cycle in the Linked List

```
public static void removeLoop(Node head){
        if(head==null || head.next==null)
```

```
        return;
    Node slow = head;
    Node fast = head;

    do{
        slow = slow.next;
        fast = fast.next.next;
        if(fast==null || fast.next==null)
            return;
    }while(fast!=slow);

    int c = 0;
    do{
        slow = slow.next;
        c++;
    }while(slow!=fast);

    slow = head;
    fast = head;

    while(c-->1)
        fast = fast.next;

    while(fast.next!=slow){
        slow = slow.next;
        fast = fast.next;
    }
    fast.next = null;
}
```

t.c = ~n, s.c = 1

34) Flattening of a LinkedList

Start merging from the end node.

```
Node merge(Node a, Node b)
{
        if(a==null) return b;
        if(b==null) return a;

        Node result;

        if(a.data<b.data)
        {
                result = a;
                result.bottom = merge(a.bottom, b);
        }
        else
        {
```

```
                result = b;
                result.bottom = merge(a, b.bottom);
        }
        return result;
}

Node flatten(Node root)
{
        if(root==null || root.next==null)
                return root;

        root.next = flatten(root.next);
        root = merge(root, root.next);
        return root;
}
```

t.c = O(n*m)(nexts*bottoms)
s.c = 1

35) Rotate Linked List

Given a Linked list rotate it right by k times.

i. Rotation is moving the last node to first. So do it n times.
   t.c = (numberOfTimes * LengthOfTheList), s.c = 1
ii. Other Approach is, Rotation by k times means reverse entire list once and reverse first k elements and reverse next k elements.
   If k is more than length then k = k%length. This gives us the correct output

```
  public ListNode rotateRight(ListNode head, int k) {
        if(head==null || head.next==null)
            return head;

        ListNode temp = head;
        int count = 0;
        while(temp!=null){
            temp = temp.next;
            count++;
        }
        k = k%count;
        if(k==0)
            return head;

        ListNode root = reverse(head);
        temp = root;

        while(k-->1)
            temp = temp.next;

        ListNode prev = temp;
        temp = temp.next;
        prev.next = null;
```

```
        root = reverse(root);
        temp = reverse(temp);

        head = root;
        while(root.next!=null)
            root = root.next;

        root.next = temp;

        return head;
    }
    ListNode reverse(ListNode head){
        if(head==null || head.next==null)
                return head;

        ListNode root = reverse(head.next);
        head.next.next = head;
        head.next = null;
        return root;
    }
```

t.c = n, s.c = 1

36) Clone a Linked List with random and next pointer

i. Use HashMap, **t.c = n, s.c = n**
ii. Use Inserting technique

```
Node cloneList(Node head)
{
        if(head==null)
                return null;
        Node cur = head;

        while(cur!=null)
        {
                Node clone = new Node(cur.data);
                clone.next = cur.next;
                cur.next = clone;
                cur = clone.next;
        }

        cur = head;

        while(cur!=null)
        {
                cur.next.random = (cur.random!=null)?cur.random.next:null;
                cur = cur.next.next;
        }
```

```
        Node answer = head.next;
        Node copy = answer;
        cur = head;

        while(cur!=null)
        {
                cur.next = (cur.next!=null)?cur.next.next:null;
                copy.next = (copy.next!=null)?copy.next.next:null;

                cur = cur.next;
                copy = copy.next;
        }

        return answer;
}
```

t.c = n, s.c = 1