Leetcode Problem Links

SL No-	Problems
1	206. Reverse Linked List :
2	876. Middle of the Linked List :
3	21. Merge Two Sorted Lists:
4	237. Delete Node in a Linked List
5	83. Remove Duplicates from Sorted List
6	1290. Convert Binary Number in a Linked List to Integer
7	148. Sort List
8	2. Add Two Numbers
9	445. Add Two Numbers II
10	141. Linked List Cycle
11	19. Remove Nth Node From End of List
12	234. Palindrome Linked List
13	160. Intersection of Two Linked Lists
14	203. Remove Linked List Elements
15	82. Remove Duplicates from Sorted List II
16	1721. Swapping Nodes in a Linked List
17	328. Odd Even Linked List
18	24. Swap Nodes in Pairs
19	25. Reverse Nodes in k-Group
20	142. Linked List Cycle II
21	725. Split Linked List in Parts
22	23. Merge k Sorted Lists
23	138. Copy List with Random Pointer

24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

Linkedin

Linked List (C++)

1. Create a Singly Linked List in C++ and perform the following operations.
    - Insert elements in the beginning
    - Insert elements at the end.
    - Insert elements in between
    - Insert elements after an element (using value).
    - Insert elements after an element (using index).

```cpp
#include <iostream>
using namespace std;

struct Node
{
    int data;
    Node *next;


    Node(int val)
    {
       data=val;
        next = NULL;
    }
};


void linkedlist(Node** head)
{
    int N, value;
cin>>N;

 Node *current = NULL;
int i=0;

 Node *dummynode =  new Node(-1000);
*head=dummynode;


while(i!=N)
 {   cin>>value;
    current = new Node(value);
    dummynode->next = current;
    dummynode= dummynode->next;
    i++;
}
*head = (*head)->next;
}


void printlinkedlist(Node* head)
{
    //head = head->next;
    cout<<endl;
while(head!=NULL)
{
    cout<<head->data<<" \t";
    head=head->next;
}

}
```

```
void insertatbeg(Node** head, int newval)
{
     Node* newnode = new Node(newval);
     newnode->next= *head;
     *head= newnode;
}

void insertatend(Node** head, int newval)
{
  Node* temp = *head;
 Node* newnode = new Node(newval);
 if(temp==NULL)
  { *head = newnode;
  return;
  }
    while(temp->next!=NULL)
    {
        temp=temp->next;
    }

    temp->next=newnode;
    newnode->next = NULL;

}

void insertinbetween(Node* head, Node* n1, Node* n2, int newval)
{
  Node* temp = head;
    while(temp!=n1)
    {
            temp = temp->next;
    }
     Node* between = new Node(newval);
    between->next = n2;
    (n1)->next= between;
}


void insertafteranelement(Node* head, int after, int value)
{   Node* temp = head;

   while(temp!=NULL && temp->next!=NULL  && temp->data!=after)
   {
       temp=temp->next;
   }

if((temp==NULL) || (temp->data!=after))
{
    return;
}
else
   {  Node* newnode = new Node(value);
        newnode->next= temp->next;
       temp->next = newnode;
   }
}
```

```
void insertafteranelementwithindex(Node* prevnode, int val)
{
    if(prevnode==NULL)
      return ;

    Node* newnode = new Node(val);
    newnode->next = prevnode->next;
    prevnode->next = newnode;
}

int main()
{
Node *head = NULL;
linkedlist(&head);
cout<<" Created linked list ";
printlinkedlist(head);

 insertatbeg(&head,1000);
 cout<<endl<<"insert at beginning";
 printlinkedlist(head);
 insertatbeg(&head,5000);
 cout<<endl<<"insert at beginning";
 printlinkedlist(head);

cout<<endl<<" Insert element at the Ending ";
insertatend(&head,50);
printlinkedlist(head);
cout<<endl<<" Insert element at the Ending ";
insertatend(&head,10);
printlinkedlist(head);
insertafteranelementwithindex(head, 200);
cout<<endl<<" Insert element after index head ";
printlinkedlist(head);

 insertatbeg(&head,600);
 cout<<endl<<"insert at beginning";
 printlinkedlist(head);

  Node *n1 = head->next;
  Node *n2 = head->next->next;
insertinbetween(head,n1, n2, 25);
 cout<<endl<<"insert bewtween two pointers";
 printlinkedlist(head);

insertafteranelement(head,3,700);
 cout<<endl<<"insert after an element";
 printlinkedlist(head);

}

```

2. Create a Singly Linked List in C++ and perform the following operations.
    - Delete elements from the beginning
    - Delete elements from the end.
    - Delete elements at position.

```cpp
#include <iostream>
using namespace std;

struct Node
{
    int data;
    Node *next;


    Node(int val)
    {
       data=val;
        next = NULL;
    }
};


void linkedlist(Node** head)
{
    int N, value;
cin>>N;

 Node *current = NULL;
int i=0;

 Node *dummynode =  new Node(-1000);
*head=dummynode;


while(i!=N)
 {   cin>>value;
    current = new Node(value);
    dummynode->next = current;
    dummynode= dummynode->next;
    i++;
}
*head = (*head)->next;
}


void printlinkedlist(Node* head)
{
    //head = head->next;
    cout<<endl;
while(head!=NULL)
{
    cout<<head->data<<" \t";
    head=head->next;
}

}
```

```
void deletefrombeginning(Node** head)
{
    if(*head!=NULL)
    {
    Node* dummynode = new Node(-500);
         dummynode->next=*head;

           Node* temp = dummynode->next;

                dummynode->next = temp->next;
           delete temp;
         *head = dummynode->next;
    }

}

void deletefromend(Node** head)
{
             if(*head!=NULL)    // or we can use if(head==NULL) return;
    { Node* dummynode = new Node(-500);
       dummynode->next=*head;

     Node* prev = dummynode;
       Node* current = *head;

      while(current->next!= NULL)
      {
          prev= current;
          current=current->next;
      }
      delete current;
      prev->next= NULL;
      *head= dummynode->next;
    }
      }


void deleteatposition(Node** head, int position)
{    if(*head==NULL || position<0)
        return;
     Node*  dummynode = new Node(-500);
       dummynode->next=*head;

    Node* temp =*head;

        int count = 0;
        Node* prev =dummynode;
        while(count<position && temp->next!=NULL)
        {
            count++;
            prev=temp;
            temp=temp->next;
        }

        if(count<position)
          return;

        prev->next = temp->next;
        delete temp;
     *head=dummynode->next;
}
```

```cpp
int main()
{

 Node *head = NULL;
linkedlist(&head);
cout<<" Created linked list ";
printlinkedlist(head);

deleteatposition(&head,2523563);
cout<<endl<<" Delete from a particular position in the linked list ";
printlinkedlist(head);


deletefrombeginning(&head);
cout<<endl<<" Delete from beginning in the  linked list ";
printlinkedlist(head);


deletefromend(&head);
cout<<endl<<" Delete from end in the  linked list ";
printlinkedlist(head);

deletefromend(&head);
cout<<endl<<" Delete from end in the  linked list ";
printlinkedlist(head);



deletefromend(&head);
cout<<endl<<" Delete from end in the  linked list ";
printlinkedlist(head);
deletefromend(&head);
cout<<endl<<" Delete from end in the  linked list ";
printlinkedlist(head);
deletefromend(&head);
cout<<endl<<" Delete from end in the  linked list ";
printlinkedlist(head);

}

```

Leetcode Problems:

206. Reverse Linked List :

1. *Iterative approach :*

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev=NULL;
        ListNode* current = head;
        ListNode* Temp= NULL;

      while(current!=NULL)
      {
         Temp= current->next;
         current->next= prev;
          prev= current;
        current=Temp;
      }
      return prev;
    }
};

//TC = O(N)
//SC = O(1)
```

2. *Recursive Approach:*

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(head==nullptr) return nullptr;
        if(head->next ==nullptr) return head;
        ListNode* curr=head;
        ListNode* temp=reverseList(head->next);
        curr->next->next=curr;
        curr->next=nullptr;
        return temp;
    }

};

/*Recursive approach
   TC = O(N)
   SC = O(1)
*/
```

[876. Middle of the Linked List](#) :

1. *Using Two traversals :*

```cpp
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* current =head;
      int length = 0;

      while(current!=NULL)
      {
          length+=1;
          current=current->next;
        }

      int mid= length/2+1;
      for(int i=1;i<mid;i++)
      {
        head=head->next;
      }
       return head;

    }
};
```

2. Using two pointers approach :

```cpp
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* fast =head;
        ListNode* slow = head;

       while(fast->next && fast->next->next)
       {
           fast=fast->next->next;
           slow =slow->next;
       }
      if(fast->next!=NULL)
          slow=slow->next;
        return slow;
    }
};
/* TC= O(N)
   SC= O(1)
*/
```

[21. Merge Two Sorted Lists]():

1. *Using extra space for creating new nodes:*

```cpp
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {

        if(l1==NULL) return l2;

        if(l2==NULL) return l1;

    ListNode* dummy=new ListNode(-500);
    ListNode* head=dummy;

    while(l1!=NULL && l2!=NULL){
        if(l1->val<=l2->val){
            ListNode* newnode = new ListNode(l1->val);
            head->next=newnode;
            l1=l1->next;
         }
        else{
            ListNode* newnode = new ListNode(l2->val);
            head->next=newnode;
            l2=l2->next;
        }
        head=head->next;
    }
    head->next=(l1!=NULL) ? l1 : l2;

    return dummy->next;
}

  /*
   TC = O(l1+l2) or max(l1,l2)
   SC = O(l1+l2)
  */
```

2. Using O(1) extra space:

```cpp
//Using O(1) extra space
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if(l1==NULL) return l2;
    if(l2==NULL) return l1;
    ListNode* dummy=new ListNode();
    ListNode* head=dummy;
    while(l1!=NULL && l2!=NULL){
        if(l1->val<=l2->val){
            head->next=l1;
            l1=l1->next;
        }
        else{
            head->next=l2;
            l2=l2->next;
        }
        head=head->next;
    }
    head->next=(l1!=NULL) ? l1 : l2;
    return dummy->next;
}
};
/*  TC= O(l1+l2)
   SC = O(1)
*/

```

3. Using Recursion with O(1) space:

```cpp
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l1 == NULL){
            return l2;
        }
        if(l2 == NULL){
            return l1;
        }
        if(l1->val < l2->val){
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        }
        else{
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
    }
    }
};
//recursive code : tc  O(l1+l2)
```

237. Delete Node in a Linked List

```cpp
class Solution {
public:
    void deleteNode(ListNode* node) {
      node->val= node->next->val;
      node->next = node->next->next;

    }
};
```

83. Remove Duplicates from Sorted List

1. **Iterative solution:**

```cpp
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* curr = head;
        ListNode* temp = NULL;
        while(curr && curr -> next){
            if(curr -> val == curr -> next -> val){
                temp = curr -> next;
                curr -> next = curr -> next -> next;
                delete temp;
            }
            else{
                curr = curr -> next;
            }
        }
        return head;
    }
};
```

2. **Recursive solution:**

```cpp
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(!head || !(head -> next)){
            return head;
        }
        if(head -> val == head -> next -> val){
            ListNode* temp = head -> next;
            head -> next = head -> next -> next;
            delete temp;
            deleteDuplicates(head);
        }
        else{
            deleteDuplicates(head -> next);
        }
        return head;
    }
};
```

NOTE : There can be more than two duplicates so if we get the duplicate elements, we have to compare the next elements with current elements so we are calling deleteDuplicates(head) in the if condition.

[1290. Convert Binary Number in a Linked List to Integer]

```cpp
class Solution {
public:
    int getDecimalValue(ListNode* head) {
        int ans=0;
        while(head!=NULL)
        {
            ans*=2;
            ans+=(head->val);
            head=head->next;
        }
        return ans;
    }
};
```

148. Sort List

Using Merge Sort:

```cpp
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if(head==NULL || head->next==NULL)
            return head;

        ListNode* fast= head;
        ListNode* slow= head;

        while(fast->next && fast->next->next)
        {
            fast=fast->next->next;
            slow= slow->next;
        }
    ListNode* list2 = slow->next;
    slow->next=NULL;
    ListNode* list1 = head;

       ListNode* left = sortList(list1);
       ListNode* right = sortList(list2);
    ListNode* answer = mergeTwoLists(left,right);
        return answer;

    }
     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if(list1==nullptr && list2==nullptr) return nullptr;
        if(list1==nullptr) return list2;
        if(list2==nullptr) return list1;
        ListNode* l1=list1;
        ListNode* l2=list2;
        ListNode* nH=new ListNode(-1);
        ListNode* nC=nH;
        while(l1!=nullptr && l2!=nullptr){
            if((l1->val)<(l2->val)){
                ListNode* curr=l1;
                l1=l1->next;
                nC->next=curr;
                nC=curr;
            }else{
                ListNode* curr=l2;
                l2=l2->next;
                nC->next=curr;
                nC=curr;
            }
        }
        if(l1==nullptr){
            nC->next=l2;
        }else if(l2==nullptr){
            nC->next=l1;
        }
        return nH->next;
    }

};
```

---

2. Add Two Numbers

```cpp
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {

        ListNode* dummynode = new ListNode(-1);
     /*   if(l1==NULL)
            return l2;
        if(l2==NULL)
            return l1;
       dummynode will handle this case
       */

        ListNode* result = dummynode;
        int carry =0;
        while(l1 && l2)
        {   int ans = (l1->val + l2->val)+carry;

         carry = ans/10;
          ans = ans%10;

            ListNode* newnode = new ListNode(ans);
             dummynode->next = newnode;
             dummynode= dummynode->next;
            l1=l1->next;
            l2=l2->next;
        }

        while(l1 || l2)
        {
             int ans= l1? l1->val +carry:  l2->val +carry;
            carry = ans/10;
            ans = ans%10;

        ListNode* newnode2 = new ListNode(ans);
        dummynode->next = newnode2;
        dummynode= dummynode->next;
            if(l1)
            l1=l1->next;
            else
            l2=l2->next;
        }

      if(carry!=0)
      { ListNode* unitnode = new ListNode(carry);
        dummynode->next = unitnode;
        dummynode= dummynode->next;
      }
        return result->next;
    }
};
```

445. Add Two Numbers II

```cpp
class Solution {
public:

    ListNode* reverseList(ListNode* head) {
        ListNode* prev=NULL;
        ListNode* current = head;
        ListNode* Temp= NULL;

      while(current!=NULL)
      {
         Temp= current->next;
         current->next= prev;
          prev= current;
        current=Temp;
      }
      return prev;
    }

    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* dummynode = new ListNode(-1);

        l1 = reverseList(l1);
        l2=reverseList(l2);

     /*   if(l1==NULL)
            return l2;
        if(l2==NULL)
            return l1;
       dummynode will handle this case
       */

        ListNode* result = dummynode;
        int carry =0;
        while(l1 && l2)
        {   int ans = (l1->val + l2->val)+carry;

         carry = ans/10;
          ans = ans%10;

            ListNode* newnode = new ListNode(ans);
             dummynode->next = newnode;
             dummynode= dummynode->next;
            l1=l1->next;
            l2=l2->next;
        }

```

```
  while(l1 || l2)
        {
             int ans= l1? l1->val +carry:  l2->val +carry;
            carry = ans/10;
            ans = ans%10;

        ListNode* newnode2 = new ListNode(ans);
        dummynode->next = newnode2;
        dummynode= dummynode->next;
            if(l1)
            l1=l1->next;
            else
            l2=l2->next;
        }

      if(carry!=0)
      { ListNode* unitnode = new ListNode(carry);
        dummynode->next = unitnode;
        dummynode= dummynode->next;
      }

        result = reverseList(result->next);
        return result;
    }


};
```

[141. Linked List Cycle]

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
       if(head==NULL)
        return false;

        set<ListNode*> elements;
        while(head!=NULL)
        {
            if(elements.count(head))
                return true;
            elements.insert(head);
             head=head->next;

        }
        return false;
    }
};
```

3. Two pointer approach

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
       if(head==NULL)
           return false;

        ListNode* slow=head;
       ListNode* fast=head;


        while(fast->next && fast->next->next)
        {
            slow= slow->next;
            fast= fast->next->next;
                if(slow==fast)
                    return true;
        }

        return false;
    }
};

// 2 corner cases - 1st : if no node 2nd : if only one node
//TC = O(N)
//SC = O(1)
```

19. Remove Nth Node From End of List

```cpp
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev=NULL;
        ListNode* current = head;
        ListNode* Temp= NULL;

      while(current!=NULL)
      {
         Temp= current->next;
         current->next= prev;
          prev= current;
        current=Temp;
      }
      return prev;
    }

    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* l1 = reverseList(head);
        ListNode* dummynode = new ListNode(-100);
        dummynode->next = l1;

        ListNode* current = l1;
        ListNode* prev = dummynode;
        int count=0;
        while((count+1)!=n)
        {
            prev=current;
            current=current->next;
            count++;
        }
        prev->next = current->next;
        delete current;
         return reverseList(dummynode->next);
    }

};
```

234. Palindrome Linked List

First approach:

```cpp
class Solution {
public:

    ListNode* reverseList(ListNode* head) {
        ListNode* prev=NULL;
        ListNode* current = head;
        ListNode* Temp= NULL;

      while(current!=NULL)
      {
         Temp= current->next;
         current->next= prev;
          prev= current;
        current=Temp;
      }
      return prev;
    }

    bool isPalindrome(ListNode* head) {

        ListNode* copynode = head;
        ListNode* dummynode = new ListNode(-5000);
        ListNode* currentnode = dummynode;

        while(copynode!=NULL)
        {
            ListNode* newnode = new ListNode(copynode->val);
            currentnode->next = newnode;
            currentnode= currentnode->next;
            copynode=copynode->next;
        }


     ListNode* secondnode=dummynode->next;
     ListNode* firstnode = reverseList(head);

        while(firstnode)
        {
            if((firstnode->val)==(secondnode->val))
            {
                firstnode=firstnode->next;
                secondnode=secondnode->next;
           }
            else { return false;
            }
        }
        return true;

    }
};
```

3. Using two pointers

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* fast =head;
        ListNode* slow = head;

       while(fast->next && fast->next->next)
       {
           fast=fast->next->next;
           slow =slow->next;
       }
        return slow;
    }
    ListNode* reverseList(ListNode* head) {
        ListNode* prev=NULL;
        ListNode* current = head;
        ListNode* Temp= NULL;

      while(current!=NULL)
      {
         Temp= current->next;
         current->next= prev;
          prev= current;
        current=Temp;
      }
      return prev;
    }


    bool isPalindrome(ListNode* head) {
        if(head==NULL || head->next==NULL)
            return true;

        ListNode* middle = middleNode(head);
        ListNode* tempnode = middle->next;
        middle->next =NULL;

        ListNode* templist = reverseList(tempnode);
         return compare(head, templist);
    }

    bool compare(ListNode* l1, ListNode* l2)
    {

        while(l1 && l2)
        {
            if(l1->val!=l2->val)
            {
                return false;
            }
            l1=l1->next;
            l2=l2->next;
        }
     return true;
    }

};
```

160. Intersection of Two Linked Lists

```cpp
//brute force O(M*N)
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
       ListNode* temp;
        while(headA)
        {  temp=headB;
            while(temp)
            {   if(headA==temp)
                {
                return headA;
                }
            temp=temp->next;
            }

            headA=headA->next;
        }
       return NULL;
    }
};
```

[203. Remove Linked List Elements]

```cpp
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {

        ListNode* dummynode = new ListNode(-1000);
        dummynode->next = head;


        ListNode* current= dummynode;

        while(current->next!=NULL)
        {
            if(current->next->val==val)
            {   ListNode* temp = current->next;
                current->next = current->next->next;
                delete temp;
            }
           else { current=current->next;
                }

        }
        return dummynode->next;
    }
};
```

82. Remove Duplicates from Sorted List II

```cpp
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* dummynode = new ListNode(-101);
        dummynode->next= head;
        ListNode* prev = dummynode;
        ListNode* current =head;

        while(current)
            {
            if(prev->val==current->val)
                {
                prev->next = current->next;
                }
          else   prev = current;
            current = current->next;



        }
        return dummynode->next;
    }
};

```

```cpp
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head)
    {
     if(head==NULL||head->next==NULL) return head;
    bool isDuplicate=false;
    ListNode *curr=head,*prev=NULL;

        while(curr!=NULL&&curr->next!=NULL)
        {
            if(curr->val==curr->next->val)
            {
            ListNode *temp=curr->next;
            curr->next=curr->next->next;
            delete temp;
            isDuplicate=true;
            continue;
            }
            else
            {
                if(isDuplicate)
                {
                    if(curr->next!=NULL)
                    {
                    curr->val=curr->next->val;
                    ListNode *temp=curr->next;
                    curr->next=curr->next->next;
                    delete temp;
                    isDuplicate=false;
                    }
                }
                 else
                {
                prev=curr;
                curr=curr->next;
                }
         }
     }

    if(isDuplicate==true && curr->next==NULL)
    {
        if(prev==NULL)
        {
        delete curr;
        return NULL;
        }
        else
        {
        prev->next=NULL;
        delete curr;
        return head;
        }
    }
    return head;
}
};
```

1721. Swapping Nodes in a Linked List

```cpp
class Solution {
public:
    ListNode* swapNodes(ListNode* head, int k) {
        ListNode* dummynode = new ListNode(-1);
        dummynode->next = head;

        if(head==NULL || head->next==NULL)
            return head;

        ListNode* slow = head;
        ListNode* fast = head;

        ListNode* n1 =NULL;

         while(--k)
            {
                slow=slow->next;
            }
        n1 = slow;


        while(slow->next)
        {
            slow=slow->next;
            fast= fast->next;
        }

        ListNode* n2 = fast;


      int temp = n1->val;
        n1->val=n2->val;
        n2->val=temp;

    return dummynode->next;

    }
};
```

328. Odd Even Linked List

```cpp
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        ListNode* dummynode1 = new ListNode(-100);
         ListNode* dummynode2 = new ListNode(-1001);
        ListNode* odd=dummynode1;
        ListNode* even=dummynode2;

        int count=1;
        while(head)
        {
            if(count%2)
            {   ListNode* newnode = new ListNode(head->val);
                odd->next= newnode;
                odd=odd->next;
            }
            else
            {   ListNode* newnode = new ListNode(head->val);
                even->next=newnode;
                even=even->next;

            }
            head=head->next;
            count++;
        }
        odd->next=dummynode2->next;
        return dummynode1->next;
    }
};
```

Linkedin

24. Swap Nodes in Pairs

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head)
    {
         if(head==NULL || head->next==NULL)
    {
        return head;
    }

    ListNode* first = head;
    ListNode* second = head->next;

    ListNode* templist = swapPairs(head->next->next);

    first->next = templist;
    second->next = first;

    return second;
}

};
```

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head)
    {
    if(!head || !head->next) return head;

        ListNode* temp;
        temp = head->next;
        head->next = swapPairs(head->next->next);
        temp->next = head;

        return temp;
}


};
```

[25. Reverse Nodes in k-Group]

```cpp
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
     ListNode* currenthead = head;
        int count=k;
        while(count--){
            if(currenthead == nullptr) return head;
            currenthead = currenthead->next;
        }

        ListNode* current = head;
        ListNode* prev = nullptr;
        ListNode* next = nullptr;

        //reverse a linkedlist of length k
        for(int i = 0; i < k; i++){
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }
        head->next = reverseKGroup(current, k);
        return prev;
    }
};
```

142. Linked List Cycle II

```cpp
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {

        if(head==NULL || head->next==NULL)
           return NULL;

        ListNode* slow=head;
       ListNode* fast=head;

        while(fast && fast->next)
        {
            slow= slow->next;
            fast= fast->next->next;
                if(slow==fast)
                {  break;
                }
        }
        if(slow!=fast)
            return NULL;

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

[725. Split Linked List in Parts]

```cpp
class Solution {
public:
    vector<ListNode*> splitListToParts(ListNode* head, int k) {

        ListNode* current=head;
        int length=0;
        while(current)
        {   current=current->next;
            length++;
        }
        int nodesinagroup= length/k;
        int extranodes = length%k;

        vector<ListNode*> res;
        current=head;

        while(current)
        {   res.push_back(current);
            int currentlength=1;
            while(currentlength<nodesinagroup)
            {
                current=current->next;
                currentlength++;
            }
         if(extranodes>0 && length>k)
         {
             current=current->next;
             extranodes--;
         }
         ListNode* temp=current->next;
         current->next=NULL;
         current=temp;
        }

        while(length<k)
        {
            res.push_back(NULL);
            length++;
        }
        return res;

    }
};
```

Merge k Sorted Lists

```cpp
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        int m=lists.size();

        if(m==0)
            return 0;
        if(m==1)
            return lists[0];

        ListNode* l1= lists[0];
        ListNode* l2 = lists[1];
        ListNode* l= mergeTwoLists(l1,l2);

    for(int i =2;i<m;i++)
    {
        l= mergeTwoLists(l,lists[i]);
    }
    return l;
    }
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if(list1==nullptr && list2==nullptr) return nullptr;
        if(list1==nullptr) return list2;
        if(list2==nullptr) return list1;
        ListNode* l1=list1;
        ListNode* l2=list2;
        ListNode* nH=new ListNode(-1);
        ListNode* nC=nH;
        while(l1!=nullptr && l2!=nullptr){
            if((l1->val)<(l2->val)){
                ListNode* curr=l1;
                l1=l1->next;
                nC->next=curr;
                nC=curr;
            }else{
                ListNode* curr=l2;
                l2=l2->next;
                nC->next=curr;
                nC=curr;
            }
        }
        if(l1==nullptr){
            nC->next=l2;
        }else if(l2==nullptr){
            nC->next=l1;
        }
        return nH->next;
    }
};

//not optimal  TC= O(nk)
```

Linkedin

138. Copy List with Random Pointer

```cpp
class Node {
public:
    int val;
    Node* next;
    Node* random;

    Node(int _val) {
        val = _val;
        next = NULL;
        random = NULL;
    }
};


class Solution {
public:
    Node* copyRandomList(Node* head) {
        unordered_map<Node*,Node*>mp;
        Node* dummy = new Node(100001);
        Node* runner = dummy;
        Node* curr = head;
        while(curr!=NULL)
        {
            Node* newnode = new Node(curr->val);
            runner->next = newnode;
            mp[curr] = newnode;
            curr=curr->next;
            runner=runner->next;
        }
        curr = head;
        runner = dummy->next;
        while(curr!=NULL)
        {
            if(curr->random!=NULL)
                runner->random = mp[curr->random];
            runner = runner->next;
            curr = curr->next;
        }
        return dummy->next;
    }
};
```

Linkedin