LC→ 21

Merge 2 sorted linked list

I/P → 2 list di hai, unko merge karke next list banani hai sorted

L1 $$\begin{tikzcd} \textcircled{1} \arrow[r] & \textcircled{2} \arrow[r] & \textcircled{3} \end{tikzcd}$$

$$\begin{tikzcd} \textcircled{1} \arrow[r] & \textcircled{1} \arrow[r] & \textcircled{2} \arrow[r] & \textcircled{3} \arrow[r] & \textcircled{4} \arrow[r] & \textcircled{4} \end{tikzcd}$$

L2 $$\begin{tikzcd} \textcircled{1} \arrow[r] & \textcircled{3} \arrow[r] & \textcircled{4} \end{tikzcd}$$

Brute

① → Vector m store karo → O(n+m)
② → Sort → O(nlogn)
③ → newlist banao, → O(n)

} → O(nlog(n+m))

Optimised O(n+m)

① traverse both, if L1→val > L2→val.
→ overwrite L1-val.
L1→ L1→next;
till L1 & L2 are traversed fully;

```
# if (list1 == NULL) return list2;
  if (list2 == NULL) return list1;

  ListNode* newhead = NULL;
  ListNode* tail = NULL;
  while (list1 && list2)
     ListNode* newnode;
     if (list1->val < list2->val)                    else {
        newnode = new ListNode (list1->val);
        list1 = list1 -> next;                              }

     if (newhead == NULL)              // if first node added
     {   newhead = newnode;               so, it is both head/ tail
         tail = newnode;
     }
```

```
else {
        tail->next = newnode;
        tail = tail->next;
    }  }

while ( list1)                                while (list2)
    {  newnode* =                                 {
       tail->next = newnode;                        ≡
    }  tail = tail->next.                           —
       list1 = list1->next;                       }.

return newhead;

}
```

---

237 → Delete Nodes in a linked list :

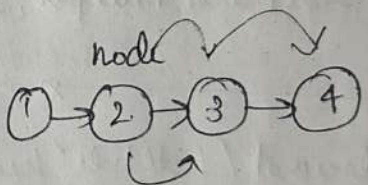
I/P → given a random node an a linked list, task is to remove that particular list node.

→ not provided with the head, as well as tail

$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 5 \end{tikzcd}$$

given node = ③ (node*)

```
{
    node->val = node->next->val;
    node->next = node->next->next;
}
```



2 ⇒ 3.
node->next = node->next->next.

→ just go froward,
→ Keep copy of the element and go foreward

→ no loops required

1290

Convert Binary Number in a Linked list to integer

I/p → given a linked list with '101', return the decimal / integral representation.

→ Convert to string
→ Convert to integer
→ return;

TC → O(n) + O(n)
SC → O(n).

```
# String ans = " ";
  while (head) {
      char ch = head→value + '0';
      ans = ans + ch;                      // String conversion
      head = head→next;
  }
  retrn stoi(ans, Nullptr, 2);             // Convert to decimal;
  }
```

```
it getdecimal (ListNode* head)
{
    int ans = 0;
    while( head) {
        ans = (ans<<1) | head→val;
        head = head→next;
    }
 return ans;
 }
};
```

Remove n^th node from End of linked list

$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \end{tikzcd}$$

I/p → head = [1, 2, 3, 4, 5], n = 2

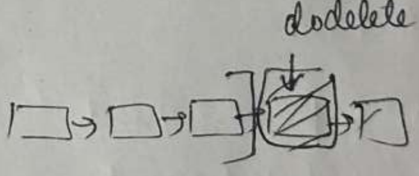
delete 2^nd node from end;

Brute: Reverse Karo.
→ fir start se for(int i=1; pos-1; i++) tk traverse, if not(null, node→next) so skip & go front;
→ reverse back.

② length nikalo, go to 'length - n' & delete

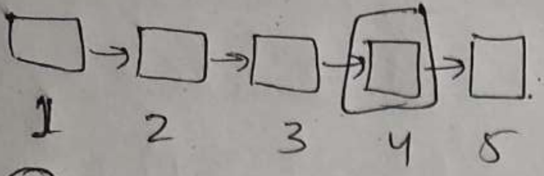
O(2n) → O(n)
[2 passes]

```
int count=0;
listNode* temp = head;
while (temp)                    // length nikalgi
 {
   count++;
   temp = temp→next;
 }
 if(n==count)            // head case;
 {
   temp = head→next;
   head = temp;
   delete temp;
  return head;
 }
else {  // go to that pos;
   for(int i = ①; i< count -n; i ++)
     { temp = temp→next;
     }
  Node* dodelete = temp→next
  temp→next = dodelete→next;
  delete dodelete;
  return head;
```



length = 5 , n = ②
(pos = length - 2)
// started from ①

[can be doun in 1 pass !
[by fast & slow pointer]



n = $\bigcirc\!\!\!\!2$.

go with ~~slow + = 2~~; (move fast n steps ahead)
~~fast + = 4~~; → than move both together +1, +1,
and delete node after slow;

```
#  listNode* slow = head;
   list Node* fast = head;
   for (int i=0; i<n; i++)
     { fast = fast→next;
     }
   if(fast = NULL) // i.e only 2 nodes        →  while(fast→next)
                        delete head                {
     {                                               S = S→next;
     LN* temp = head;                                f = f→next;
      head = head→next;          ——→              }
     delete temp;
     return head;
     }
   else { LN* temp = slow→next;
     slow→next = temp→next;
     delete temp;
     return head;
     }
};
```

LC→142

find where cycle in linked list Started :

I/P

$$\begin{tikzcd} \boxed{3} \arrow[r] & \boxed{2} \arrow[r] & \boxed{0} \arrow[r] & \boxed{-4} \arrow[ll, bend left] \end{tikzcd}$$

O/P should be ②, cause here starts.

Approach → go for Slow & fast pointer.
→ if equal break (that means cycle is present)

→ rest slow = head.
→ now iterate slow +1 , fast +1,
→ if equal, return slow;

```
# 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) ~~return head~~ break;
        }                                                      // move out
  Slow = head;
  // again with both +1
   while ( s != fast)
   {
       s = s+1;
       f = f+1;
   }
  return slow;               // starting point.
}
};
```

LC→ Two addition $\boxed{2}$

$$\begin{tikzcd} \text{I/p} \arrow[r] & 2 \arrow[r] & 4 \arrow[r] & 3 \\ & 5 \arrow[r] & 6 \arrow[r] & 4 \\ & 7 \arrow[r] & 0 \arrow[r] & 8 \end{tikzcd}$$

moving from left → right.

Approach, (carry, a variable c to check for (sum%10))

→ keep sum & move foreward; → O(n) pass

```
# int carry=0;
  ListNode* newhead = NULL;
  ListNode* tail = NULL;
  while(L1|| L2|| carry)
  {
    if(L1)
       sum = sum + L1→val;    L1=L1→next;      } suming up
    if(L2)
       sum = sum + L2→val;    L2=L2→next;
    carry = sum/10;                             } carry calculation
    newnode = new LinkNode(sum%10);             } digit into node
    if(newhead == NULL)  newhead = newnode;
                         tail = newnode;        } if 1st node
    else {  tail→next = newnode;
            tail = tail→next;                   } again further
         }
  }
  return newhead;                               } return newnode, head
}
};
```

LC→445

(Add two Numbers II)

I/P →

$$\begin{tikzcd} 7 \arrow[r] & 2 \arrow[r] & 4 \arrow[r] & 3 \\ + & 5 \arrow[r] & 6 \arrow[r] & 4 \\ 7 \arrow[r] & 8 \arrow[r] & 0 \arrow[r] & 7 \end{tikzcd}$$

Approach → reverse both the list,
l1, l2
→ perform addition.
→ return reversed(newhead);

```
listNode* addtwoNumber(l1, l2)
{
  int carry =0;
  LN* newhead = Null;
  LN* tail =Null;
  l1=reverse(l1);
  l2=reverse (l2);
  while( l1 || l2 || carry)
  {
    int sum= carry;
    if (l1)
    {
        sum + =l1→ val;
        l1 = l1 → next;
    }
    if(l2)
    {
       sum + =l2→ val;
       l2= l2→next;
    }
    carry = sum/10;
    LN newnode=new LN(sum% 10);
    if(newhead =Null)
    {
      newhead = newnode;
      tail = newnode;
    }
    else{
      tail → next =newnode;
      tail ⇒ tail → next;
    }
  }
  return reverse(newhead);
}
};
```

```
listNode* reverse(listNode* head)
{
  LN *prev=Null; next=Null;
  LN* curr=head;
  while(curr)
  {
     next= curr → next;
     curr → next= prev;
     prev = curr;
  }  curr → next;
  prev;
```