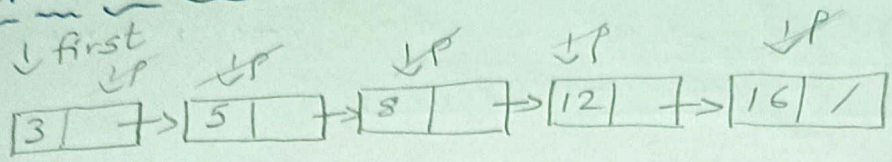
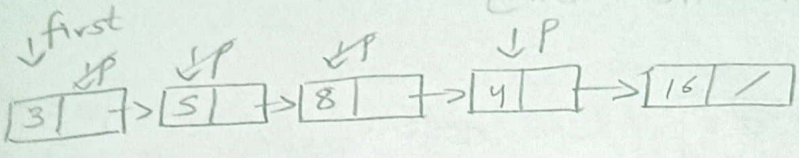
SLL continuation

Check if a LL is Sorted



x = -32768 3 5 8 12 16



x = -32768 3 5 8

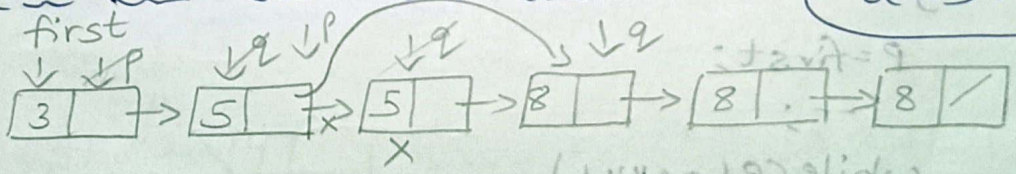
if not sorted then TC may be O(1)
if sorted Then O(n)

Code pdf ✓

```
int x = -32768;
Node *p = first;
while(p != NULL)
{ if (p->data < x)
  {
     return false;
  }
  else
  {
     x = p->data;
     p = p->next;
  }
return true;
```

O(n) TC always

Remove Duplicates from Sorted SLL



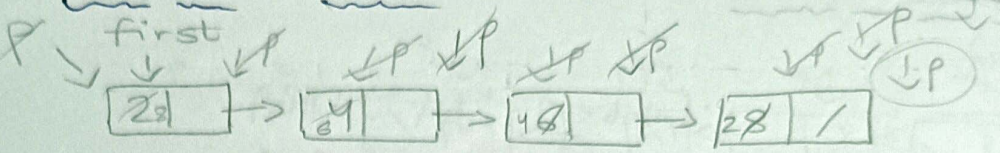
-> if p->data & q->data Then we can delete any of These Two.

```
else
{
   p->next = q->next;
   delete q;
   q = p->next;
}
}
```

```
Node *p = first;
Node *q = first->next;
while(q != NULL)
{ if (p->data != q->data)
  {
     p = q;
     q = q->next;
  }
}
```

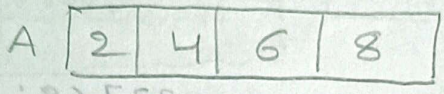
Code pdf ✓

Reversing a LL



Two methods 1. Reversing elements
2. Reversing Links.

1)

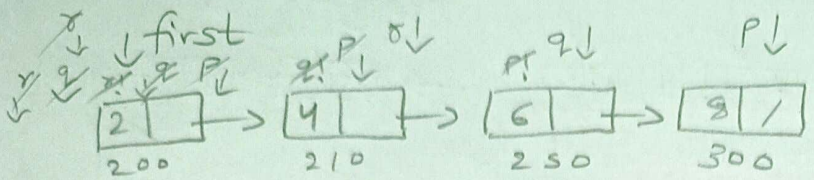


→ extra array (auxiliary space)

```
P = first;
i = 0;
while ( P != NULL)
{
    A[i] = P->data;
    P = P->next;
    i++;
}
P = first;
i--;
while (P != NULL)
{
    P->data = A[i--];
    P = P->n ;
}
```

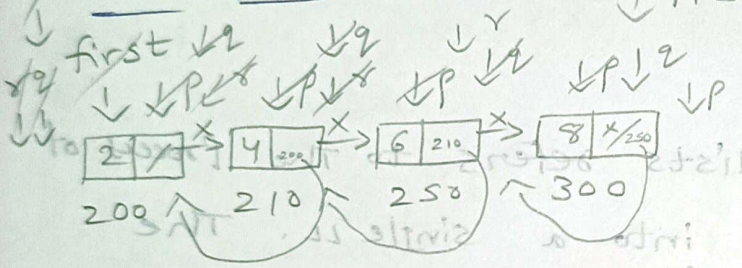
TC
2n
O(n)

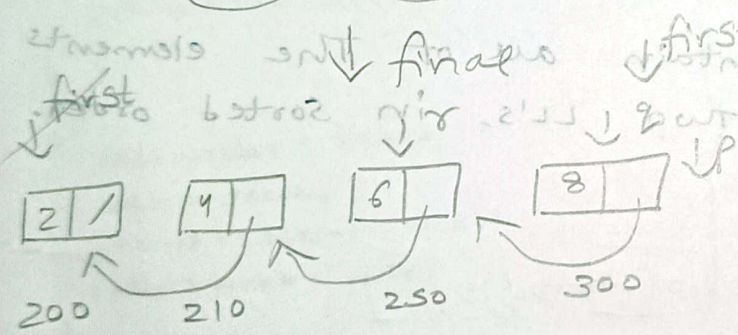
2) Sliding Pointers




```
p = first;
q = NULL;
r = NULL;
while(p != NULL)
{
    r = q;
    q = p;
    p = p->next;
}
```

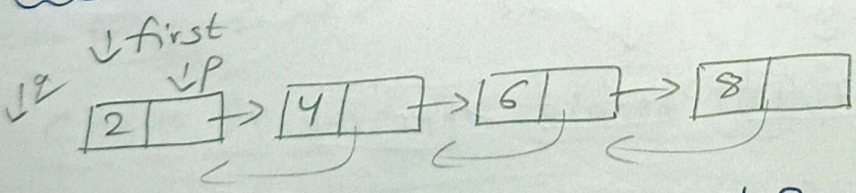
Now, for reverse code






```
p = first;
q = NULL;
r = NULL;
while(p != NULL)
{
    r = q;
    q = p;
    p = p->next;
    q->next = r;
}
first = q;
```

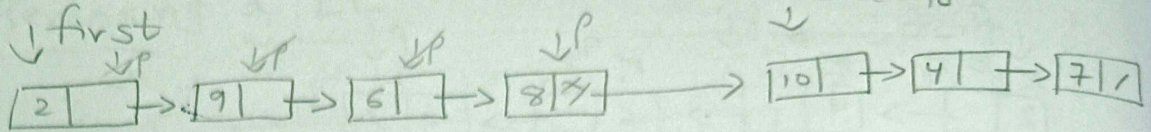
Recursive Reverse for LL




```
Void Reverse(Node *q, Node *p)
{
    if(p != NULL)
    {
        Reverse(p, p->next);
        p->next = q;
    }
    else
    {
        first = q;
    }
}
```

(code for 1, 2, Recurreverse pdf ✓ ✓

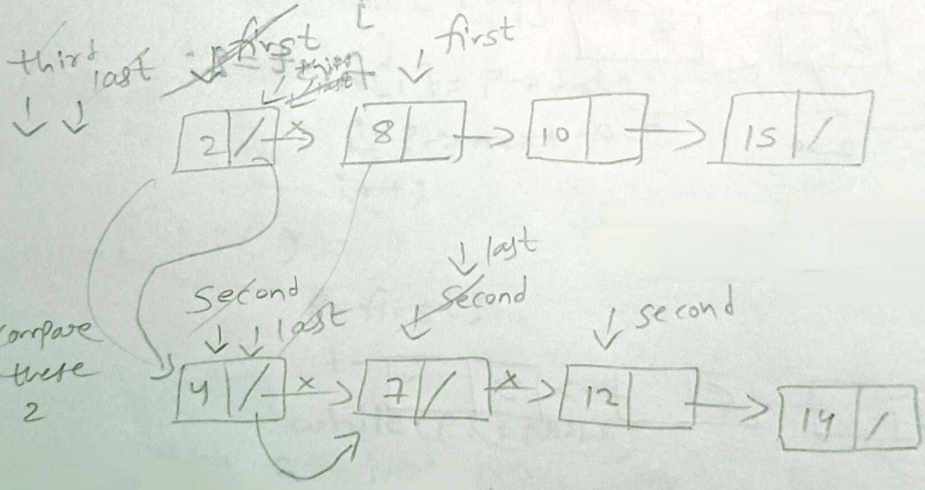
Concatenating 2 LL

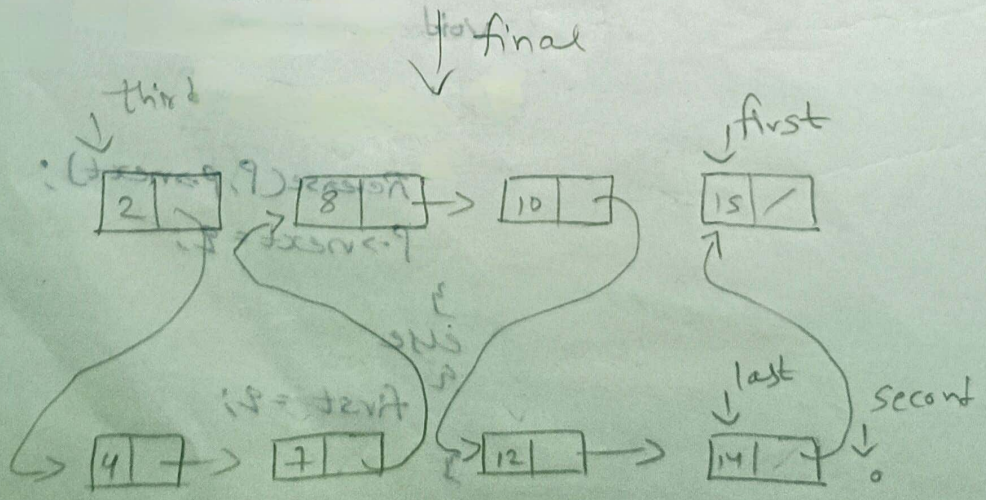



```
p = first;
while (p->next != NULL)
    p = p->next;
p->next = second;
second = NULL;
```

Merging 2 LL

Merging two sorted linked lists refers to the process of combining two LL's into a single LL. The resulting LL will contain all of the elements of the original two LL's, in sorted order.

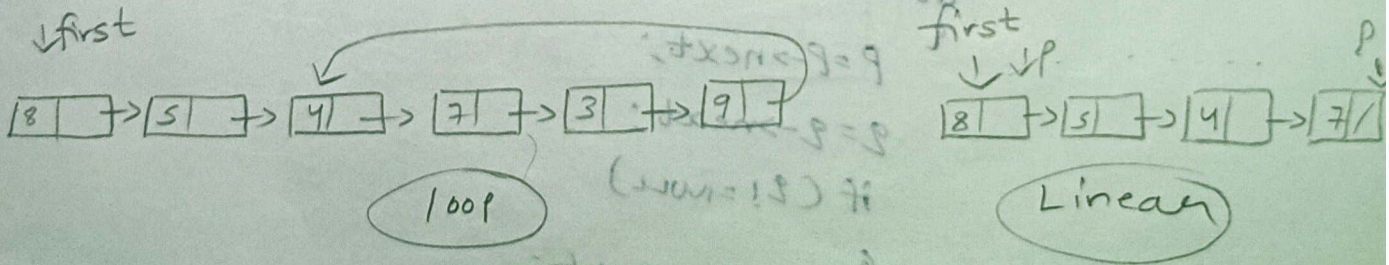




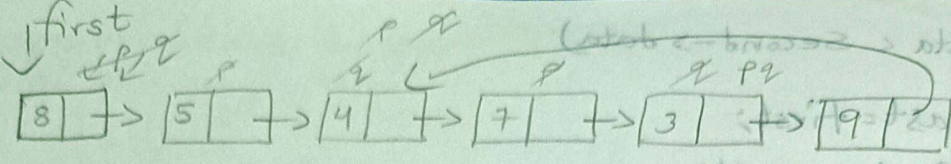
```
if (first->data < second -> data)
{   third = last = first;
    first = first -> next;
    last -> next = NULL; (or) third -> next = NULL;
}
else
{   Third = last = fecond;
    Second = second -> next;
    last -> next = NULL; (or) third -> next = NULL;
}
while (first != NULL && Second != NULL)
{ if (first -> data < second -> data)
  {  last -> next = first;
     last = first;
     first = first -> next;
     last -> next = NULL;
  }
  else
  {  last -> next = second;
     last = second ;
     second = second -> next;
     last -> next = NULL;
  }
}
if (first != NULL)
    last -> next = first;
else
    last -> next = second;
```

Code for concat pdf ✓ (easy, ok video)
Code for Merge pdf ✓

Check for Loop in Linked List.



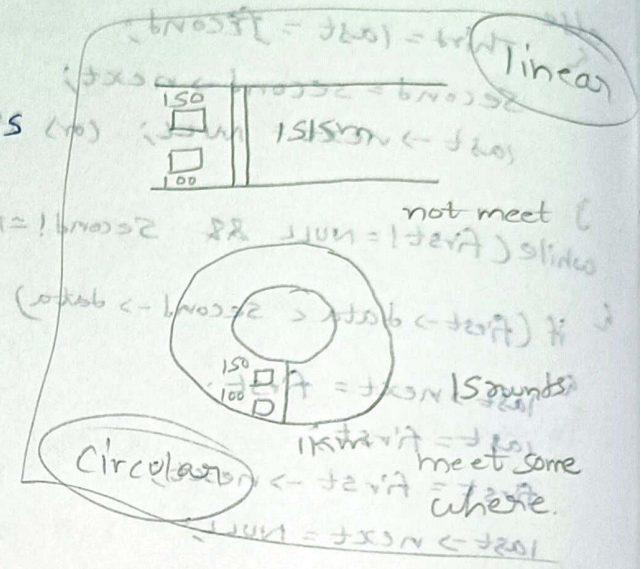
-> last node of a LL pointing on some node on LL, it's not the first one.



p & q both start from 1st node.

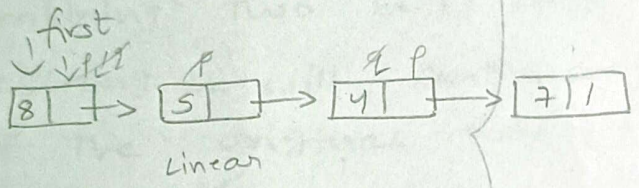
p -> move by 1 step

q -> " 2 "



* -> So, if the two pointers are meeting again after the starting point then it's a loop

-> apply p & q in linear



* if any one pointer becomes null, we should stop, then it is linear.

```
int isloop(node *f)
{
    Node *p, *q;
    p=q=f;
    do
    {
      p=p->next;
      q=q->next;
      if(q!=NULL)
      {
        q = q->next;
      }
      else
      {
        q=NULL;
      }
    }while(p&&q);
          && p!=q)
```

```
if (p==q)
    return true;
else
    return false;
}
```

Code pdfs