1. https://leetcode.com/problems/convert-binary-number-in-a-linked-list-to-integer/description/?envType=problem-list-v2&envId=linked-list

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def getDecimalValue(self, head: Optional[ListNode])
-> int:
        s=""
        curr=head
        while curr:
            s=s+str(curr.val)
            curr=curr.next
        return int(s,2)
```

2. https://leetcode.com/problems/merge-two-sorted-lists/description/?envType=problem-list-v2&envId=linked-list

```
'''
l1=[2,4,6,8]
l2=[4,5,7,9]
l3=[0]*len(l1+l2)
i=0
j=0
k=0
while i<len(l1) and j<len(l2):
    if l1[i]<l2[j]:
        l3[k]=l1[i]
        k+=1
        i+=1
    else:
        l3[k]=l2[j]
        k+=1
        j+=1
while i<len(l1):
    l3[k]=l1[i]
    k+=1
    i+=1
while j<len(l2):
    l3[k]=l2[j]
    k+=1
    j+=1
```

```
print(l3)
'''

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], 
list2: Optional[ListNode]) -> Optional[ListNode]:
        list3=ListNode()
        curr1=list1
        curr2=list2
        curr3=list3
        while curr1 and curr2:
            if curr1.val<curr2.val:
                curr3.next=curr1
                curr1=curr1.next
                curr3=curr3.next
            else:
                curr3.next=curr2
                curr2=curr2.next
                curr3=curr3.next
        while curr1:
            curr3.next=curr1
            curr1=curr1.next
            curr3=curr3.next
        while curr2:
            curr3.next=curr2
            curr2=curr2.next
            curr3=curr3.next
        return list3.next
```

3. https://leetcode.com/problems/middle-of-the-linked-list/description/?envType=problem-list-v2&envId=linked-list

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> 
Optional[ListNode]:
        slow=head
        fast=head
        while fast and fast.next:
            slow=slow.next
            fast=fast.next.next
        return slow
```