

---

---

---

---

---



## # Linked List

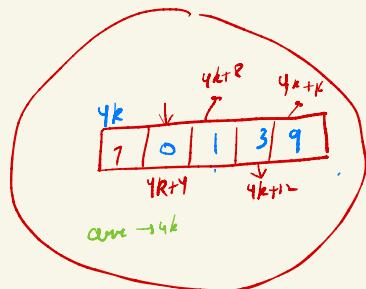
$\text{arr}(0) \rightarrow 4k$   $\text{arr} \rightarrow 7k$

$\text{int } () \quad \underline{\text{arr}_c = \text{new int } [s]}$ .

$\text{arr} \rightarrow 4k$   $5 \times 4$

$\text{arr}(0) \rightarrow$   
 $\text{arr}(1) \rightarrow 4k+4$

$4k \rightarrow 4k+4 \rightarrow$



data  
address of next  
Node object

Student s



class Node {

    → int data;

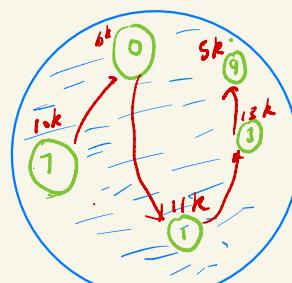
    → Node next;

}

{ Node f = new Node();  
     f.data = 7

2) Node s = new Node();  
     s.data = 0

$s = 6k$



$s = 6k$



$\text{head} = 10k$



$6k$



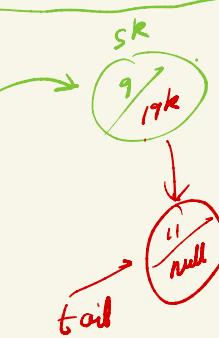
$11k$



$13k$



$15k$



$\text{tail}$

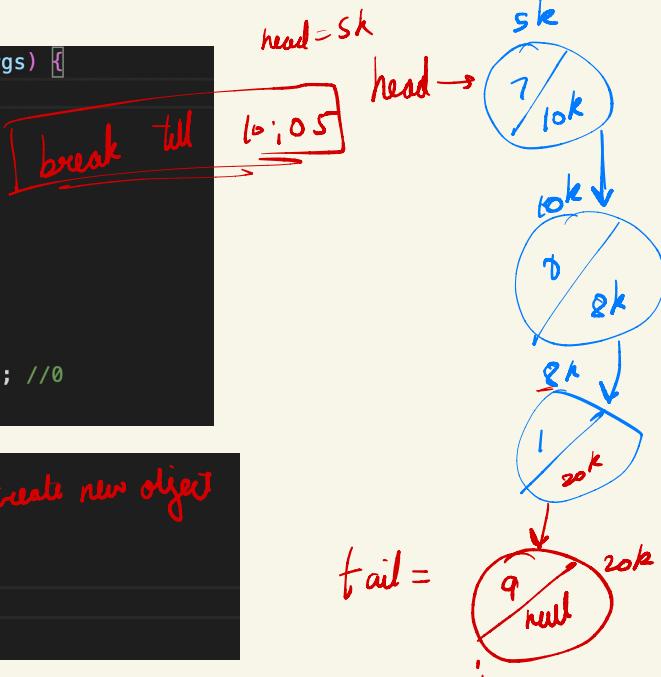
```
public void addLast(int data){
    1) Node nn = new Node(data);
    2) tail.next = nn;
    3) tail = nn;
```

```

public static void main(String[] args) {
    1) Node f = new Node(d: 7);
    2) Node s = new Node(d: 0);
    3) Node t = new Node(d: 1);

    // connecting first and second
    4) f.next = s;      head=f
    5) s.next = t;      tail=t
    6) System.out.println(f.next.data); //0
}

```



```

public void addLast(int data){ → q
    1) Node nn = new Node(data); → create new object
        tail.next = nn;
    2) tail = nn;
}

```

tail =

→ 0 size  
→ 1 size  
→ x size but

```

public class Main {
    1 static Node head;
    2 static Node tail;

    public static void addLast(int data){
        1) Node nn = new Node(data);

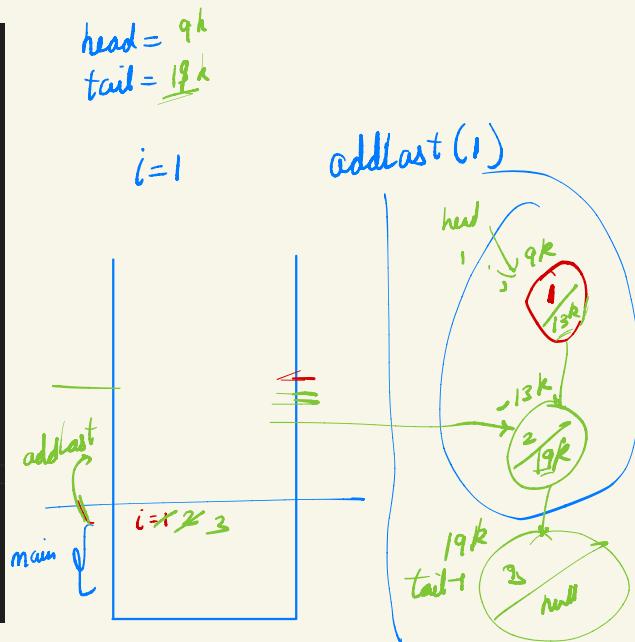
        2) if(head == null){ // in case of 0 size ll
            head = nn; // this is my first node of list
            tail = nn; // and also my last node of list
        } else {
            → tail.next = nn; // attaching
            tail = nn; // shifting tail to last
        }
    }

    Run | Debug
    public static void main(String[] args) {
        1) head =null;
        tail = null;

        2) for(int i=1; i<=3; i++){
            addLast(i);
        }

        System.out.println(head.data);
        System.out.println(tail.data);
    }
}

```



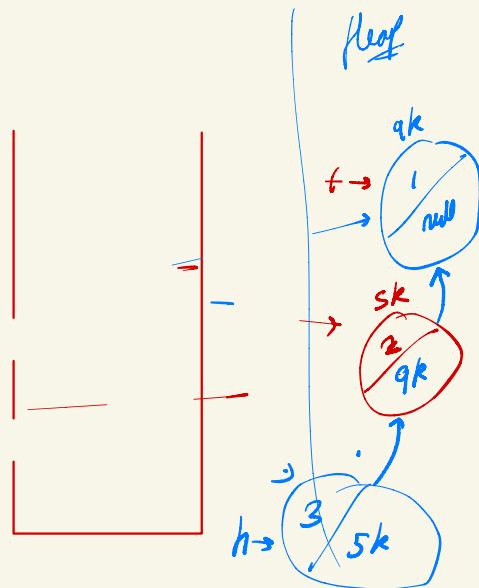
Ques Add a node at the start of ll.

- 1) addFirst(1);
- 2) addFirst(2);
- 3) addFirst(3);

head = 5k  
tail = 9k

```
public static void addFirst(int data){
    Node nn = new Node(data);

    if(head == null){
        head = nn;
        tail = nn;
    } else {
        nn.next = head;
        head = nn;
    }
}
```



# display



```
// homework |
public static void display(){
    Node curr = head;

    while(curr!=null){
        System.out.print(curr.data+" ");
        curr=curr.next;
    }
}
```

1 3 5 7 4

curr = 1/qk & tail = 4/qk null

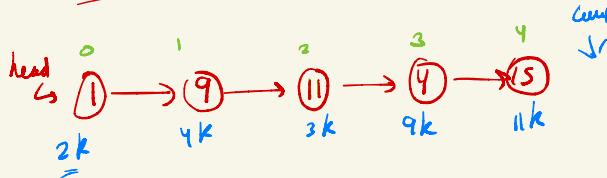
Ques Remove first element from linked list.



head = null

$\Rightarrow \underline{\underline{\text{head} = \text{head}.next;}}$

Obe Get Node at particular index.



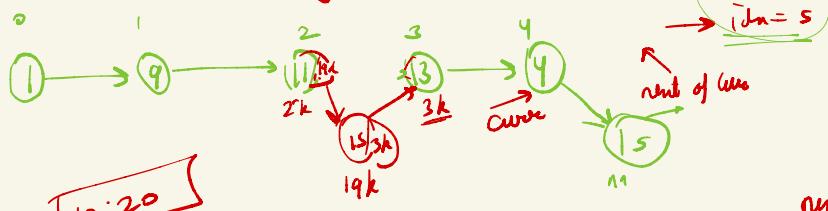
- 1)  $idx = -1$   $\rightarrow idx = 6$   
 2)  $idx = 0$   
 3)  $idx = 2$   
 4)  $idx = 4$

```
public static Node getAt(int idx){  
    if(idx < 0){  
        System.out.println("given index is out of range");  
        return null;  
    }  
  
    Node curr = head;  
    for(int i=0; i<idx; i++){  
        if(curr==null){  
            System.out.println("given index is out of range");  
            return null;  
        }  
        curr=curr.next;  
    }  
  
    return curr;  
}
```

curr = head

- $i = 0 \quad \underline{i < 6} \quad \checkmark$   
 $i = 1$   
 $i = 2$   
 $i = 3$   
 $i = 4$   
 $i = 5$

Obe <sup>use</sup> add a Node at given index.



next of curr = null

```
public static void addAt(int idx, int data){  
    Node nn = new Node(data);  
  
    Node curr = getAt(idx-1);  
    Node nextOfCurr = curr.next;  
  
    curr.next= nn;  
    nn.next = nextOfCurr;  
}
```

$i <= idx-1$

idx = 0

$2k.next = 19$   
 $NOC = 3k$

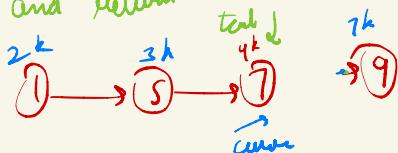
$nn.next = nextOfcurr$

$\sim 1$

$19k.next$

of ll.

Obe Remove and return the last Node of ll.



curr = 2k 3k 4k

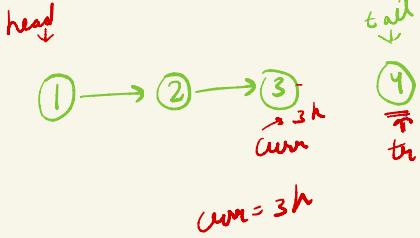
```

public static Node removeLast(){
    if(head==null){ // 0 size
        return null;
    } else if(head==tail){ // 1 size
        Node tr=tail;
        removeFirst();
        return tr;
    } else {
        Node curr = head;
        while(curr.next!=tail){
            curr = curr.next;
        }

        curr.next=null;
        // updating tail
        Node tr = tail;
        tail = curr;

        return tr;
    }
}

```

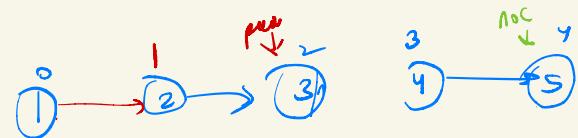


4 is deleted from ll  
but not from heap

wrote on your own code ↴ test

dry run the code

Ques Remove at given index.



```

public static void removeAt(int idx){
    int size = getLength();
    if(idx==0){
        removeFirst();
    } else if(idx==size-1){
        removeLast();
    } else {
        Node prev = getAt(idx-1);
        prev.next = prev.next.next;

        // Node nextOfCurr = prev.next.next;
        // prev.next = null;
        // prev.next = nextOfCurr;
    }
}

```

```

public static int getLength(){
    Node curr = head;
    int len = 0;

    while(curr!=null){
        curr = curr.next;
        len++;
    }

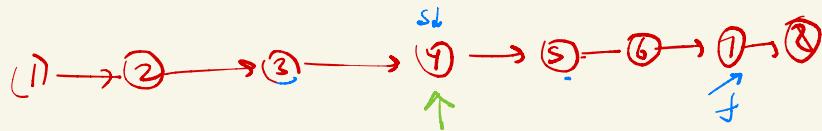
    return len;
}

```

- 1) These were the building blocks.
- 2) We have a default linked list.

Ques Find middle of linked list.

breadth first  
to

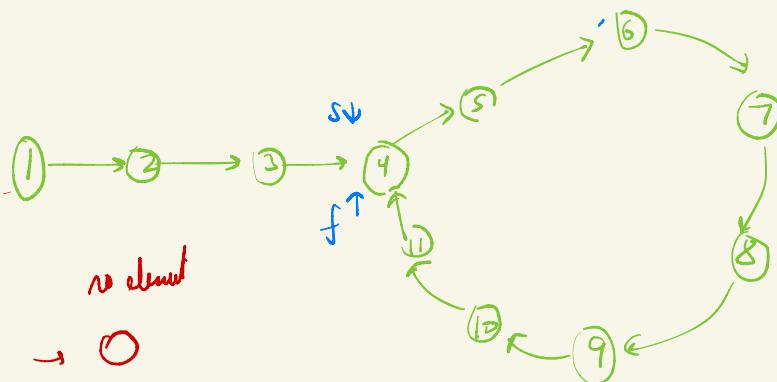


Ques what if we wanted to return first middle

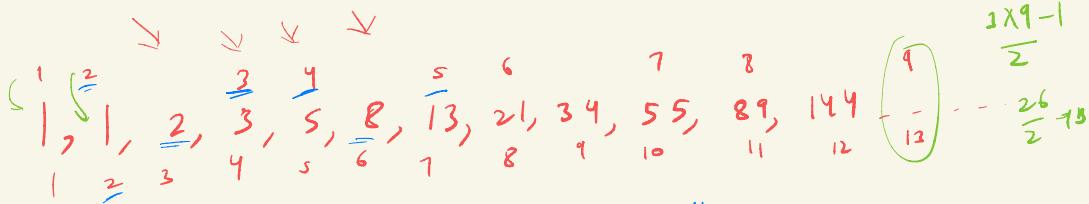
slow → 1 node per move  
fast → 2 nodes per move

[hare and tortoise method]

```
public ListNode middleNode(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;
    while(fast!=null && fast.next!=null){
        1) slow = slow.next;
        2) fast = fast.next.next;
    }
    return slow;
}
```



- 1) Check if there is a cycle
- 2) Keep fast at the intersecting node and take slow back to head
- 3) Move slow and fast at equal pace.
- 4) They will coincide again → this is the starting point.



1st odd  $\rightarrow$  1st fib

2nd odd  $\rightarrow$  2nd fib

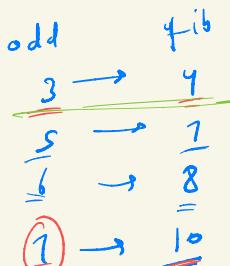
3rd odd  $\rightarrow$  4th fib

4th odd  $\rightarrow$  8th fib

5th odd  $\rightarrow$  13th fib

6th odd  $\rightarrow$  21th fib

$$\frac{3x-1}{2} \quad \frac{x^{\text{th odd}}}{\left(\frac{3x-1}{2}\right)^{\text{th fib}}}$$



3 even element

9  $\rightarrow$  13

10  $\rightarrow$  14

$$2x \Rightarrow \left( \frac{3x-1}{2} \right)$$

Find  $n^{\text{th}}$  fib



Fib

int lt = fun(n-1);  
int ltt = fun(n-2);

Expectation

my

int ans = lt + ltt;

$f(n-1)$   
 $f(n-2)$

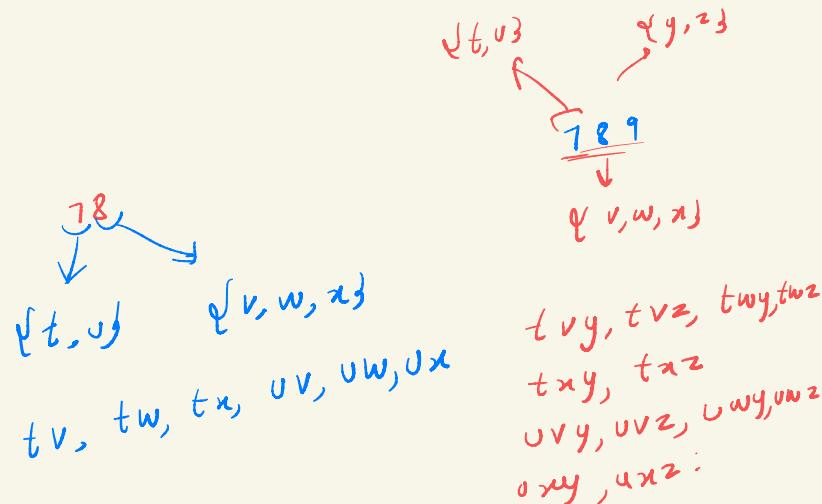
5th odd fib  $\rightarrow$  number

$\frac{3 \times 5 - 1}{2} \Rightarrow 7^{\text{th}}$

class

subsequence question

$0 \rightarrow \cdot$   
 $1 \rightarrow abc$   
 $2 \rightarrow def$   
 $3 \rightarrow ghi$   
 $4 \rightarrow jkl$   
 $5 \rightarrow mno$   
 $6 \rightarrow pqrs$   
 $7 \rightarrow tu$   
 $8 \rightarrow vwx$   
 $9 \rightarrow yz$



Fault

Return all the combinations  
for "8"  
 $\{v, w, x\}$

89

$\{vy, vz, wy, wz, xy, xz\}$   
fault

7  
↓  
tu

789  
 $\{tvy, tvz, twy, twz, txy, txz, uvy, uvz, uwy, uwz, uxy, uxz\}$

8  $\rightarrow$  v w x  
9  $\rightarrow$  y z

exception

Returns all the combinations  
when I will send the string

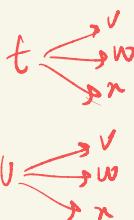
78

$\{tv, tw, tA, uv, uw, ux\}$

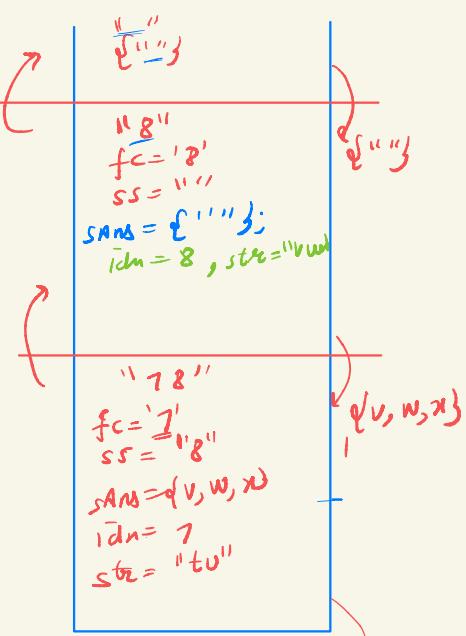
why we should not return empty ab?

```
public ArrayList<String> getKPC(String ques){  
  
    char first_char = ques.charAt(0);  
    String ss = ques.substring(1);  
    1) ArrayList<String> sans = getKPC(ss);  
  
    2) int idx = first_char - '0';  
  
    3) String str = sarr[idx]; // tu  
  
    4) ArrayList<String> ans = new ArrayList<>();  
    for(int i=0; i<str.length(); i++){  
        char ch = str.charAt(i); // t;  
        for(int j=0; j<sans.size(); j++){  
            String sansString = sans.get(j);  
            String myAnsString = ch + sansString;  
  
            ans.add(myAnsString);  
        }  
    }  
  
    return ans;  
}
```

$$\text{int idn} = '8' - '0'$$

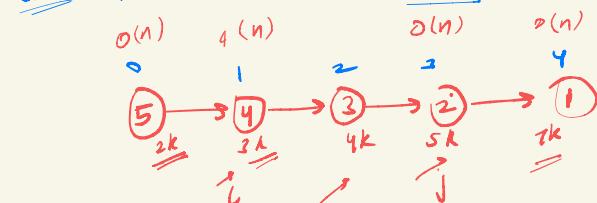


$$\begin{aligned} str &= "t \underline{v} \underline{w} \underline{n}" \\ sAns &= \{ \underline{v}, \underline{w}, \underline{n} \} \\ ans &= \{ tv, tw, tn, uv, uw, un \} \end{aligned}$$



$\{ tv, tw, tn, uv, uw, un \}$

Ans Reverses a linked list.



$$\begin{aligned} \text{int i} &= 0 \\ \text{int j} &= 4 \end{aligned}$$

$$\begin{aligned} NAI &= 3k \\ NAJ &= 5k \end{aligned}$$

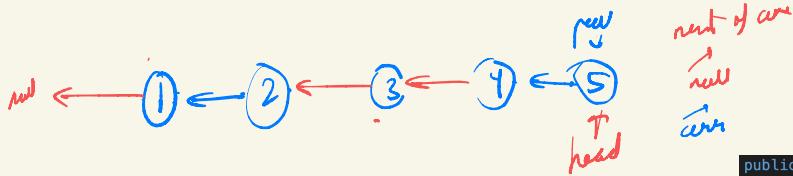
$$\begin{aligned} nxn & \\ O(n^2) & \end{aligned}$$

$$\begin{aligned} VAJ &= 2 \\ VAI &= 4 \end{aligned}$$

$n=5$

data -

```
public ListNode reverseList(ListNode head) {  
    int n = getSize(head); O(n)  
    int i=0;  
    int j=n-1;  
  
    while(i<j){  
        ListNode nodeAtI = getAt(head, i);  
        ListNode nodeAtJ = getAt(head, j);  
  
        int valAtI = nodeAtI.val;  
        int valAtJ = nodeAtJ.val;  
        swap  
        nodeAtI.val = valAtJ;  
        nodeAtJ.val = valAtI;  
  
        i++;  
        j--;  
    }  
  
    return head;  
}
```



✓

$O(n) \leftarrow$

```
public ListNode reverseList(ListNode head) {
    ListNode prev = null;
    ListNode curr = head;

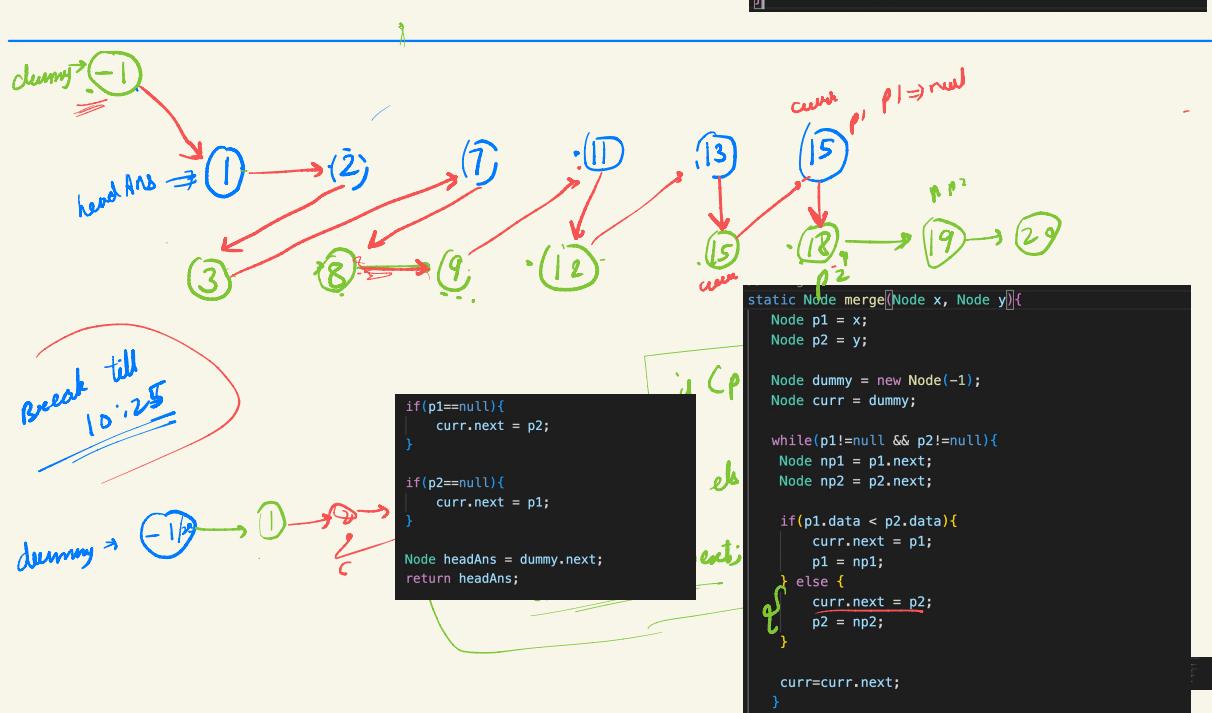
    while(curr!=null){
        // save next pointer
        1) ListNode nextOfCurr = curr.next; _____

        2) curr.next = prev; _____

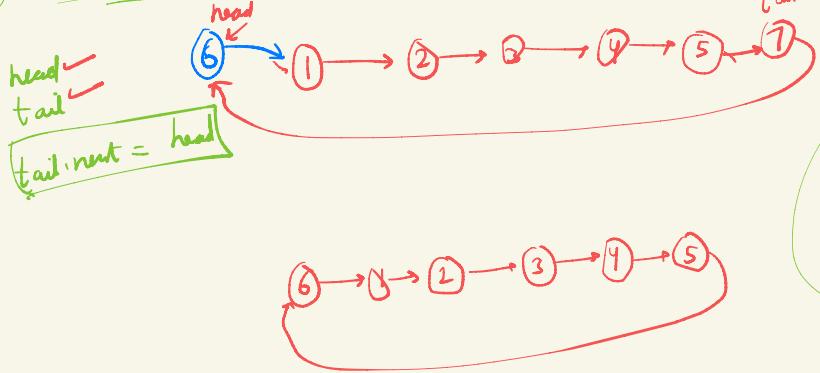
        // move pointers to the next 2 nodes
        prev = curr;
        curr = nextOfCurr;
    }

    // head is now at prev
    head = prev;

    return head;
}
```



## 1) Circular linked list



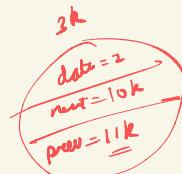
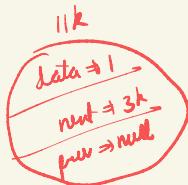
$\text{tail.next} = n^n$   
 $n^n.\text{next} = \text{head}$   
 $\text{tail} = n^n$

$n^n = 6$

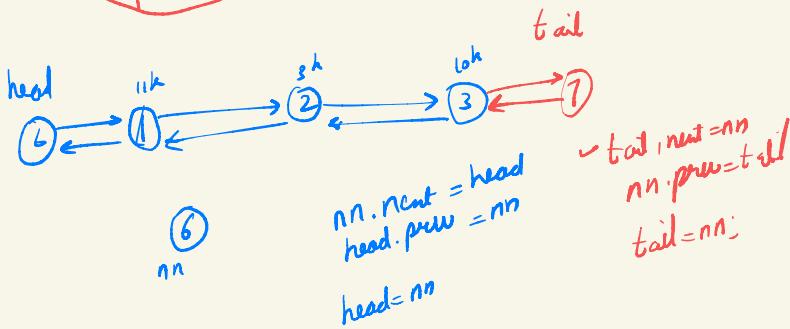
$n^n.\text{next} = \text{head}$   
 $\text{tail.next} = n^n$   
 $\text{head} = n^n$

## 2) Doubly linked list

```
class Node {
    int data;
    Node next;
    Node prev;
}
```

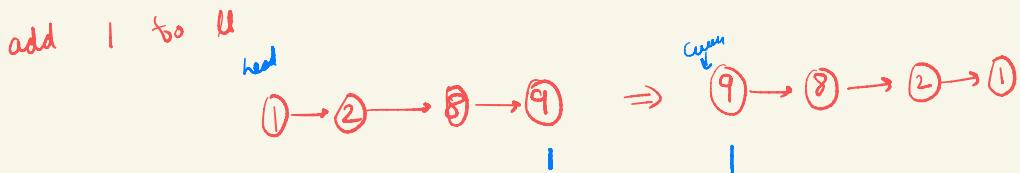


}



addFirst  
addLast

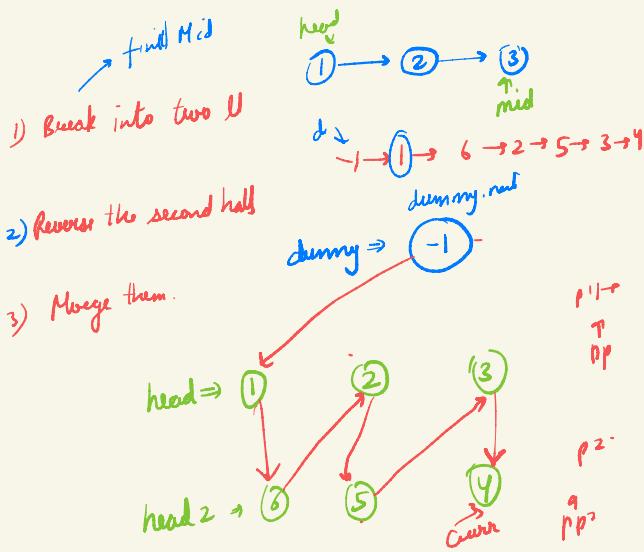
$\text{tail.next} = nn$   
 $nn.\text{prev} = \text{tail}$   
 $\text{tail} = nn$



$\text{curr} = 0$   
 $\text{curr.data} + 1$



add 2



```
public ListNode merge(ListNode head1, ListNode head2){
    ListNode dummy = new ListNode(-1);
    ListNode p1 = head1;
    ListNode p2 = head2;
    ListNode curr = dummy;

    while(p1!=null & p2!=null) {
        ListNode nextOfP1=p1.next;
        p1.next = null;

        ListNode nextOfP2 = p2.next;
        p2.next=null;

        // first add p1
        curr.next=p1;
        p1 = nextOfP1;
        curr = curr.next;

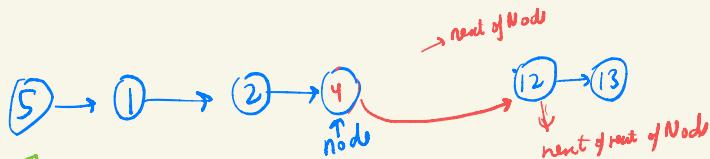
        // then add p2
        curr.next = p2;
        p2 = nextOfP2;
        curr = curr.next;
    }
}
```

```
public void reorderList(ListNode head) {
    // find mid
    ListNode mid = middleNode_(head);

    // break into two
    ListNode head2 = mid.next;
    mid.next=null;

    // reverse the second ll
    head2 = reverseList_(head2);

    // merge two lists
    head = merge(head,head2);
}
```

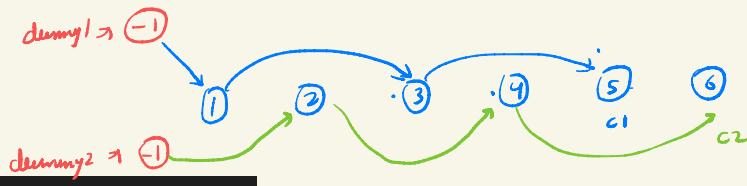


break till 9:55

160  
19

```
public void deleteNode(ListNode node) {
    // copying data of the next node to myself
    ListNode nextOfNode = node.next;
    node.val = nextOfNode.val;

    // removing node.next
    ListNode nextOfNextOfNode = nextOfNode.next;
    node.next = nextOfNextOfNode;
}
```



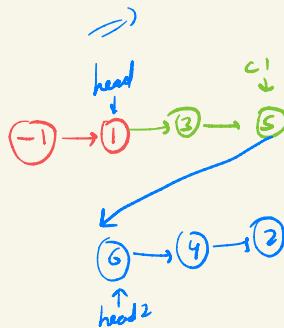
```
while(p!=null) {
    Node nextOfP = p.next;
    p.next = null;

    if(count%2 == 0){
        c1.next=p;
        p=nextOfP;
        c1=c1.next;
    } else {
        c2.next=p;
        p=nextOfP;
        c2=c2.next;
    }
    count++;
}

Node head1 = dummy1.next;
Node head2 = dummy2.next;

head2 = reverseList_(head2)
c1.next = head2;

head=head1;
```



$$\text{Count} = \cancel{0} \cancel{x^2} \underline{\equiv} 3$$