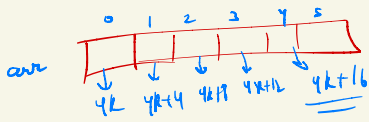
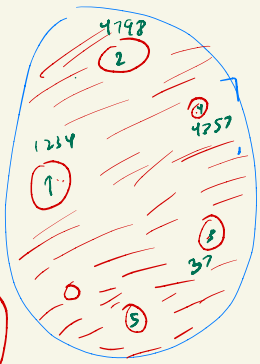
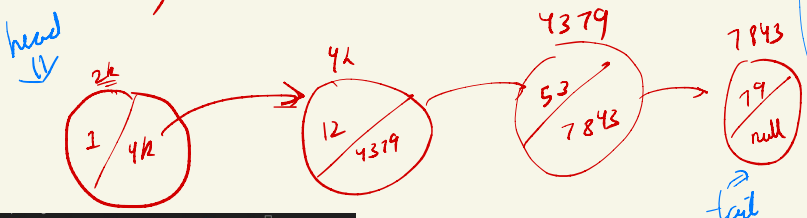
Linked List



y in memory

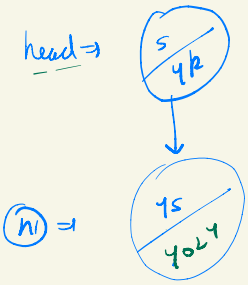
class Node {
int data;
Node next;
}






```java
public static void main(String[] args) {
    Node head=new Node(data: 5);

    Node n1=new Node(data: 45);

    head.next=n1;

    Node n2=new Node(data: 7);
    n1.next=n2;

    Node nextOfHead=head.next;
    System.out.println(n1+" fggfg "+nextOfHead);
    System.out.println(nextOfHead.data);
}
```

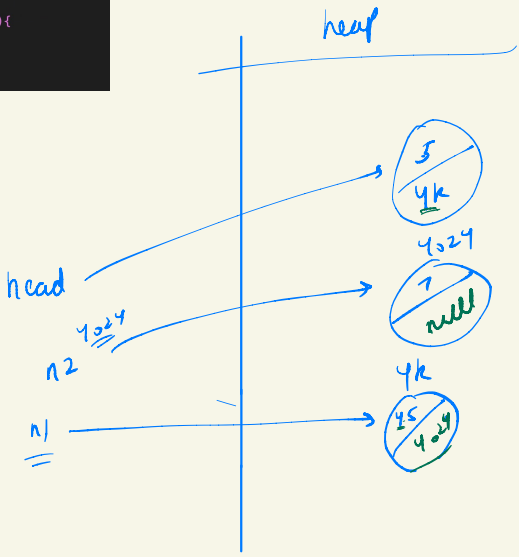
```java
class Node {
    int data;
    Node next;

    public Node(){

    }

    public Node(int data){
        this.data=data;
    }
}
```

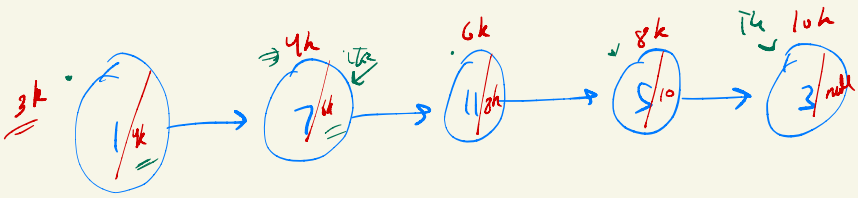


head.data
head.next = 4k

n1.data = 45
n1.next = 4024

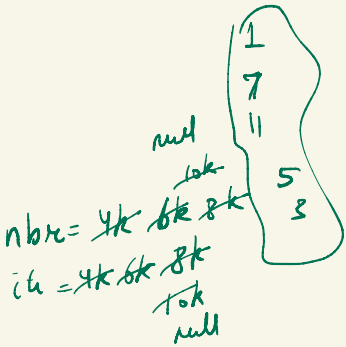
n1





head
(3k)

```java
public static void printLinkedList(Node head){
    System.out.println(x: "Printing linked list");
    Node itr=head;

    while(itr!=null){
        System.out.print(itr.data+" ");

        Node nbr=itr.next;
        itr=nbr;
    }
}
```



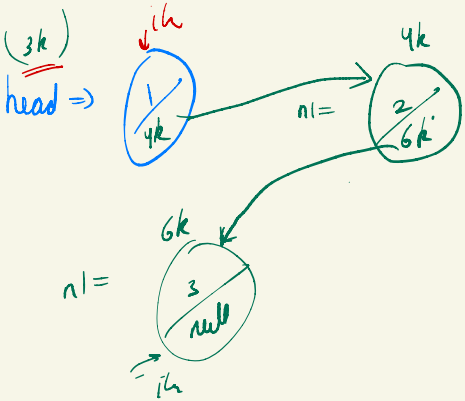
3 ⇒ 1, 2, 3

data = 2 3

f- l = 1

itr =

```java
    Scanner scn=new Scanner(System.in);
    System.out.println(x: "Enter number of inputs ");
    int n=scn.nextInt();

    int first_ele=scn.nextInt();

    Node head=new Node(first_ele);
    Node itr=head;

    for(int i=1; i<n; i++){
        int data=scn.nextInt();

        Node n1=new Node(data);
        itr.next=n1;

        Node nbr=itr.next;
        itr=nbr;
    }

    printLinkedList(head);
}
```



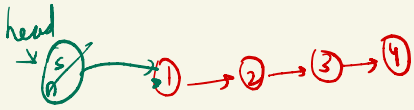
addFirst

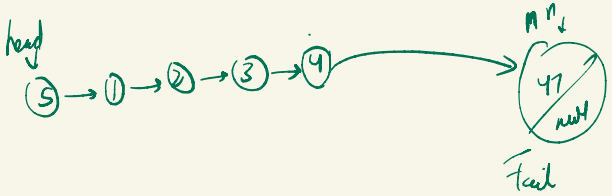
val = 5

Node nn = new Node(val);

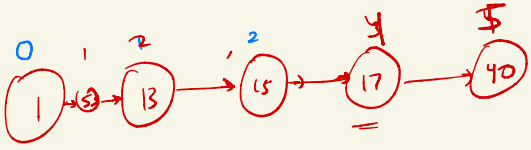
nn.next = head;

head = nn;






```
public static Node getTail(Node head){
    Node itr=head;

    while(itr.next!=null){
        // itr=itr.next;
        Node nbr=itr.next;
        itr=nbr;
    }

    return itr;
}

public static void addLast(int val, Node head){
    Node tail=getTail(head);

    Node nn=new Node(val);
    tail.next=nn;

    tail=nn;
}
```

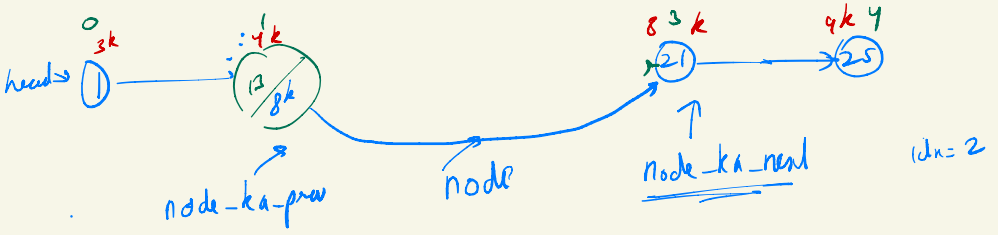


idx = 4, val = 55

i = 0 1 2 3

```
public static Node getAt(Node head, int idx){
public static Node addAt(Node head, int idx, int val){
public static Node removeAt(Node head, int idx){
    if(idx==0){
        return removeFirst(head);
    }

    Node node_ka_prev=head;

    for(int i=0; i<(idx-1); i++){
        Node nbr=node_ka_prev.next;
        node_ka_prev=nbr;
    }

    Node node=node_ka_prev.next;

    Node node_ka_next=node.next;

    node_ka_prev.next=node_ka_next;

    return head;
}
```

Ques Add a val at idx

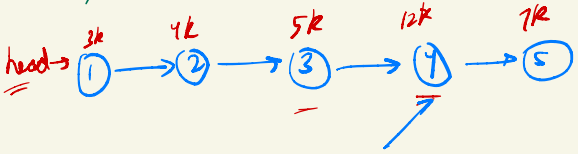


i < 1

Ques1 Find length of linked list.

Ques2 Write a recursive function to return a node in linked list with data equal to target :- (3)

tar = 4



Expectation

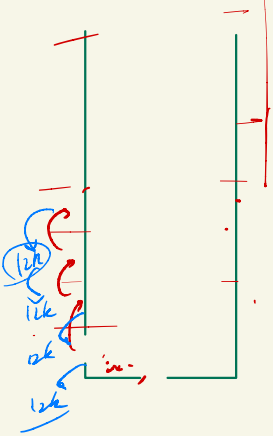
It will find the node from head to last

Faith

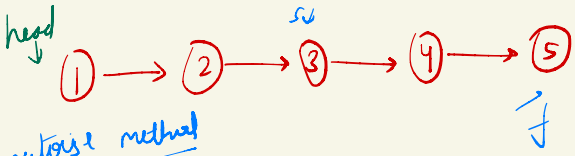
It will find node from head.next to last.

```
public static Node find_Rec(Node itr, int tar){
    if(itr==null){
        return null;
    }

    if(itr.data==tar){
        return itr;
    }

    Node ans=find_Rec(itr.next, tar);
    return ans;
}
```

Ques) Find middle of LL

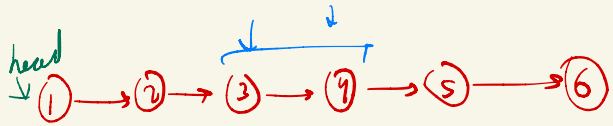


Ques Find middle of LL in one traversal.

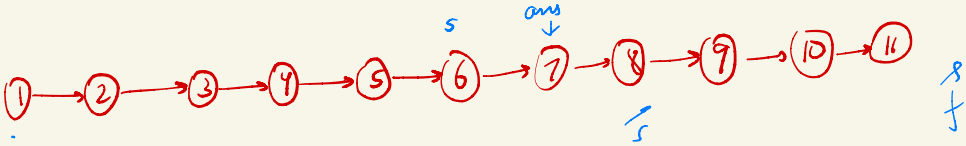


here - tortoise method

```java
public static Node getMid(Node head){
    Node slow=head;
    Node fast=head;

    while(fast.next!=null && fast.next.next!=null){
        slow=slow.next;
        fast=fast.next.next;
    }

    return slow;
}
```



Ques

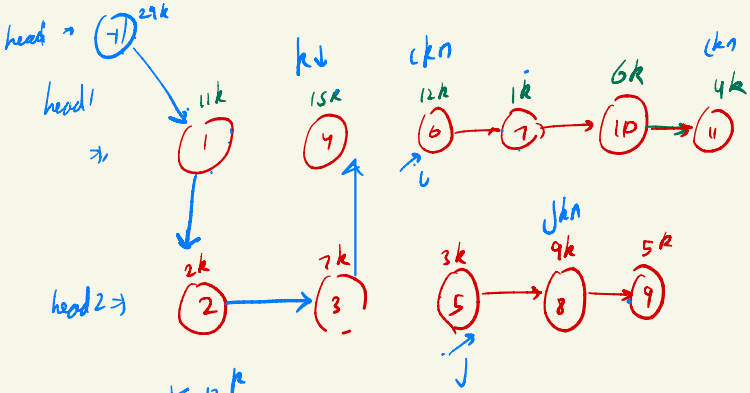



```java
public static Node getKthFromEnd(Node head, int k){
    Node slow=head;
    Node fast=head;

    for(int i=0; i<k; i++){
        fast=fast.next;
    }

    while(fast!=null){
        slow=slow.next;
        fast=fast.next;
    }

    return slow;
}
```

Run | Debug

last 5th

last 4th



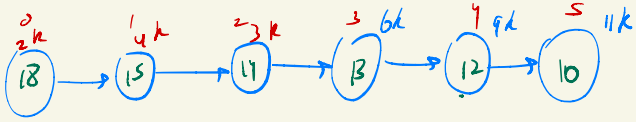
i = 12k

j = 3k

k = 15k

```java
public ListNode mergeTwoLists(ListNode head1, ListNode head2) {
    ListNode i=head1;
    ListNode j=head2;

    ListNode head=new ListNode(-1);
    ListNode k=head;

    while(i!=null && j!=null){
        ListNode ikaNext=i.next;
        ListNode jkaNext=j.next;

        if(i.val < j.val){
            k.next=i;
            i.next=null;
            i=ikaNext;
        } else {
            k.next=j;
            j.next=null;
            j=jkaNext;
        }

        k=k.next;
    }

    if(i!=null){
        k.next=i;
    }

    if(j!=null){
        k.next=j;
    }

    return head.next;
}
```

Run | Debug




```java
public static void reverse(Node head){
    int len=size(head);

    int i=0;
    int j=len-1;

    while(i<j){
        Node nodeAti=getAt(head, i);
        Node nodeAtj=getAt(head, j);

        int dataAti=nodeAti.data;
        int dataAtj=nodeAtj.data;

        nodeAtj.data=dataAti;
        nodeAti.data=dataAtj;

        i++;
        j--;
    }
}
```

O(n)

O(n)

O(n)

```java
public static Node getAt(Node head, int idx){
    Node itr=head;

    for(int i=0; i<idx; i++){
        Node nbr=itr.next;
        itr=nbr;
    }

    return itr;
}
```

nAi = 3k

nAj = 6k

dai = 1.3

daj = 14

O(n)

$\frac{n}{2} \times n \Rightarrow \frac{n^2}{2} \Rightarrow O(n^2)$


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

    while(curr!=null){
        1) ListNode currKaNext=curr.next;

        2) curr.next=prev;

        3) prev=curr;
           curr=currKaNext;
    }

    return prev;
}
```

ll

doubts