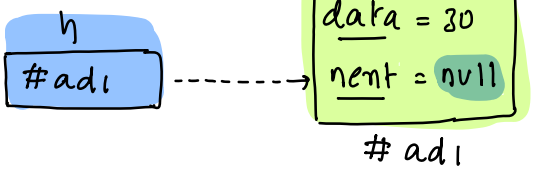
Todays Content:

- → Basic Linked List ✓
- → Insert in a sorted Linked list ✓
- → Delete k in a sorted Linked list TODO
- → Reverse LinkedList ✓
- a) Revers the first k nodes ✓
- b) Revers all k groups ✓
- → Find the middle of linked list ✓

```
Class Node{ // { In your language of choice }
    int data // variable
    Node next // object reference → Hold address of Node Object
    Node(int x){ // Constructor → used to initialize data members of Class
        data = x
        next = NULL
    }
}
```

`Node h = new Node(30);`

↳ // an object is created & h is an object reference here



Access data & next

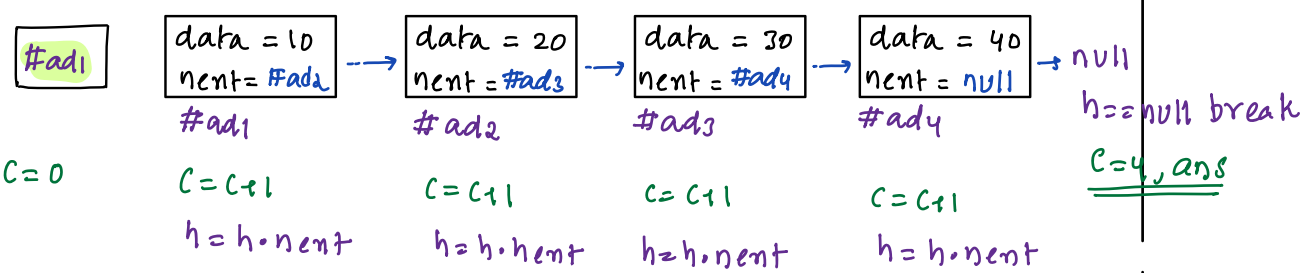
print(h.data) = 30

print(h.next) = null

Note: h is just holding reference/address of object

print(h) : #ad1

Linked list:




```
int size(Node h){ TC: O(N) SC: O(1)
    c=0
    while(h!=null){
        c=c+1
        h=h.next
    }
    return c
}
```

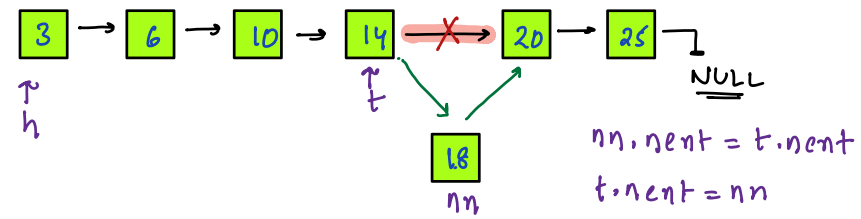
Note: Heavy Interviews:

```
if(h!=null){        null.next
    h.data / h.next
}
if(h!=NULL && h.next!=null){
    h.next.data / h.next.next
}
```

Problems:

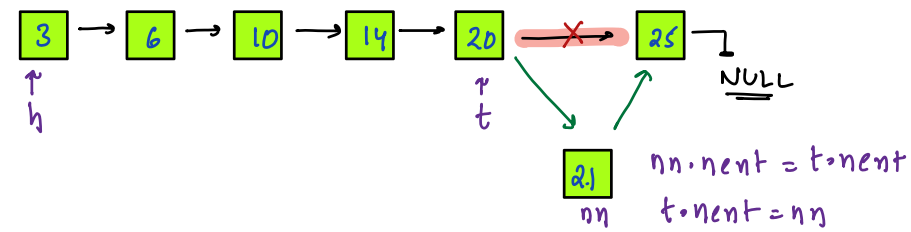
1) Given head of a sorted linked, create & insert new node with data = x

Ex1: x = 18

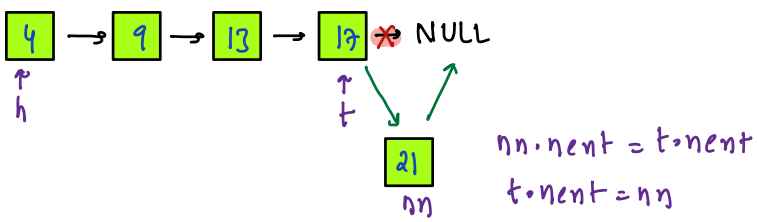


obs1: We iterate & stop at node, if it's next node data > x

Ex2: x = 21

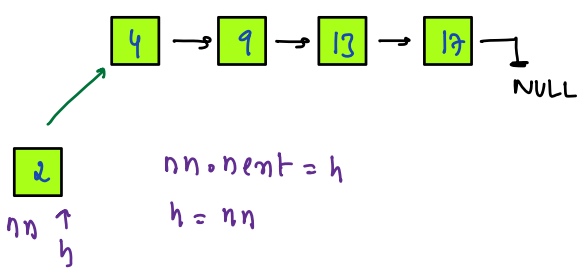


Ex3: x = 21



obs2: We stop at a node, if it's next == null

Ex4: x = 2



obs3: if (x < h.data) { Insert at start }

Since we might change head return head node of linked list

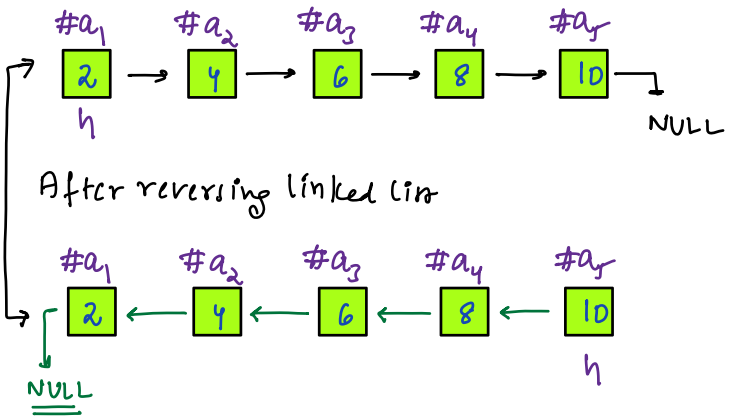
```
Node insertSorted(Node h, int x){   TC: O(N)  SC: O(1)
    Node nn = new Node(x)
    if(h==null){ h=nn, return h}
    if(x<h.data){
        nn.next=h
        h=nn
        return h
    }
    Node t=h;
    while(t.next!=NULL && t.next.data<x){
        t=t.next
    }
    // Insert new node nn after t
    nn.next=t.next
    t.next=nn
    return h
}
```

Edge Case: h=null x=20

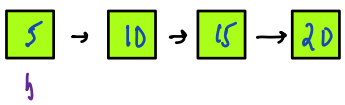
20] Given a head of linkedlist, reverse linkedlist & return head

Note: No Extra Space & We cannot change val of node TC: O(N) SC: O(1)

Eg1:

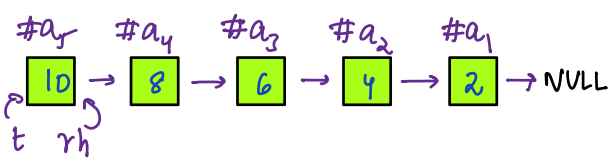


Idea: We inserted 20, 15, 10, 5 at start 1 by 1



// obs: If we insert nodes from start, we will get in reverse order

Trace:

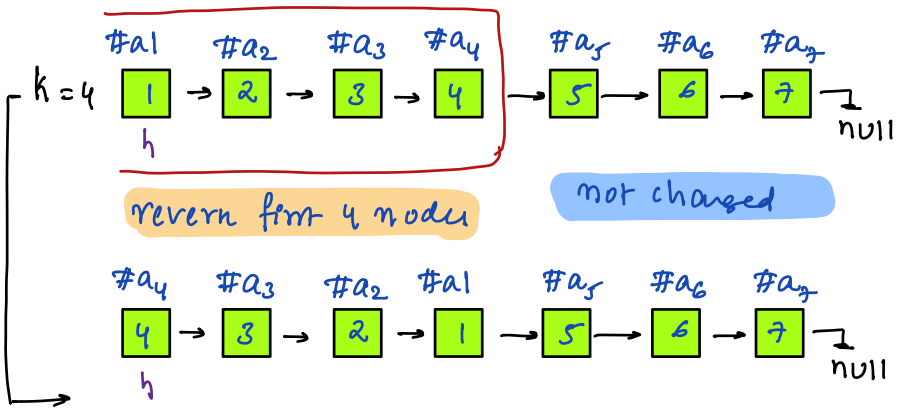



```
Node reverse(Node h){        TC: O(N)
                             SC: O(1)
    Node rh = null
    Node t = h
    while(h != null){
        t = h;
        h = h.next
        t.next = null
        t.next = rh
        rh = t
    }
    return rh; // head node
     of reversed list
}
```

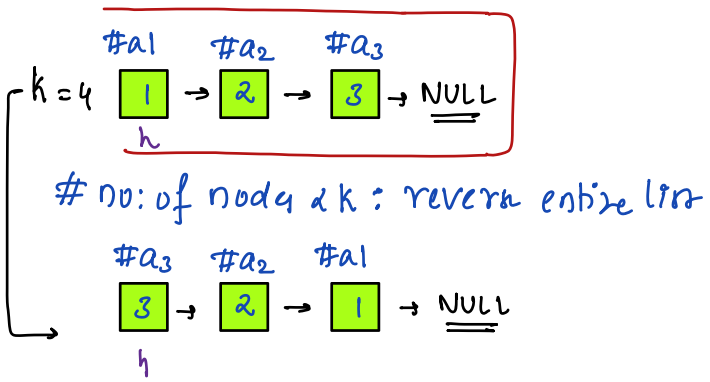
38) Given head of linked list, revern first k nodes, given $k > 0$

Note: #no: of nodes $\geq k$, still revern comple linked list

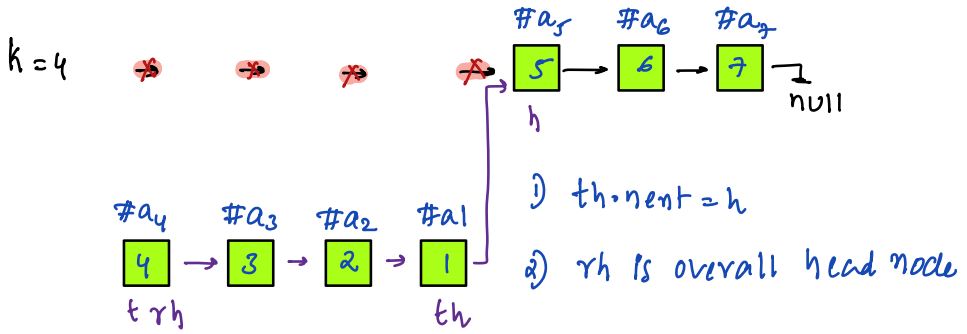
Eg1:



Eg2:



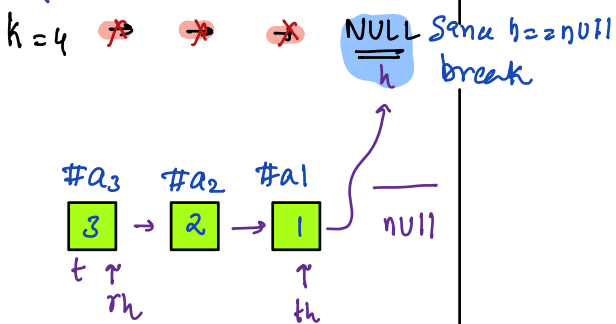
Trace: `Node th = h` // Before we revern, storing head in a temp reference




```
Node revern(Node h, int k){
    if (h == NULL || k <= 1) { return h }
    Node th = h
    Node rh = null, Node t = h;
    int c = k
    while ( c > 0 && h != null ) {
        t = h
        h = h.next
        t.next = null
        t.next = rh
        rh = t
        c = c-1;
    }
    th.next = h } Error can occur, if h == null
    return rh
}
```

// Isolating 4 adding at start

Edge Case: To avoid h != NULL

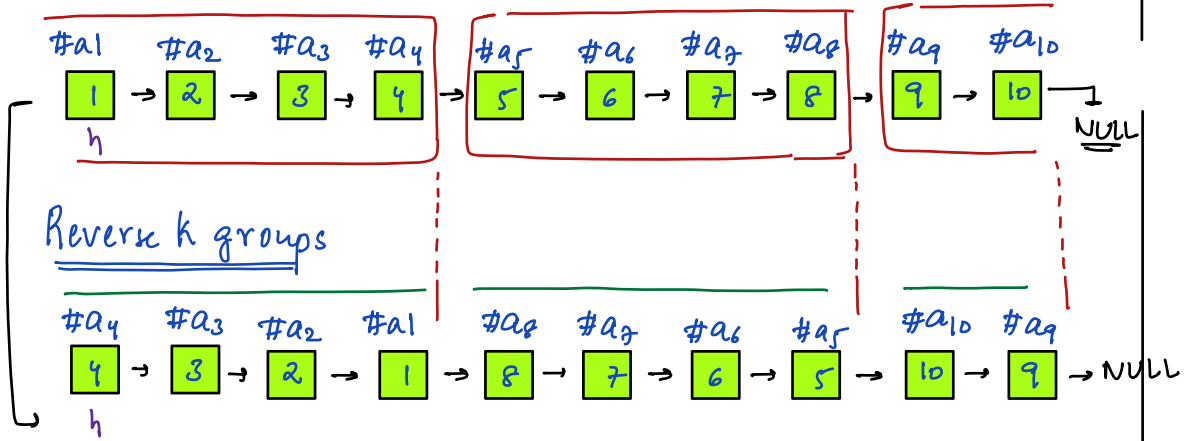


Q8) Given head of linked list, reverse all groups of size: k, k>0

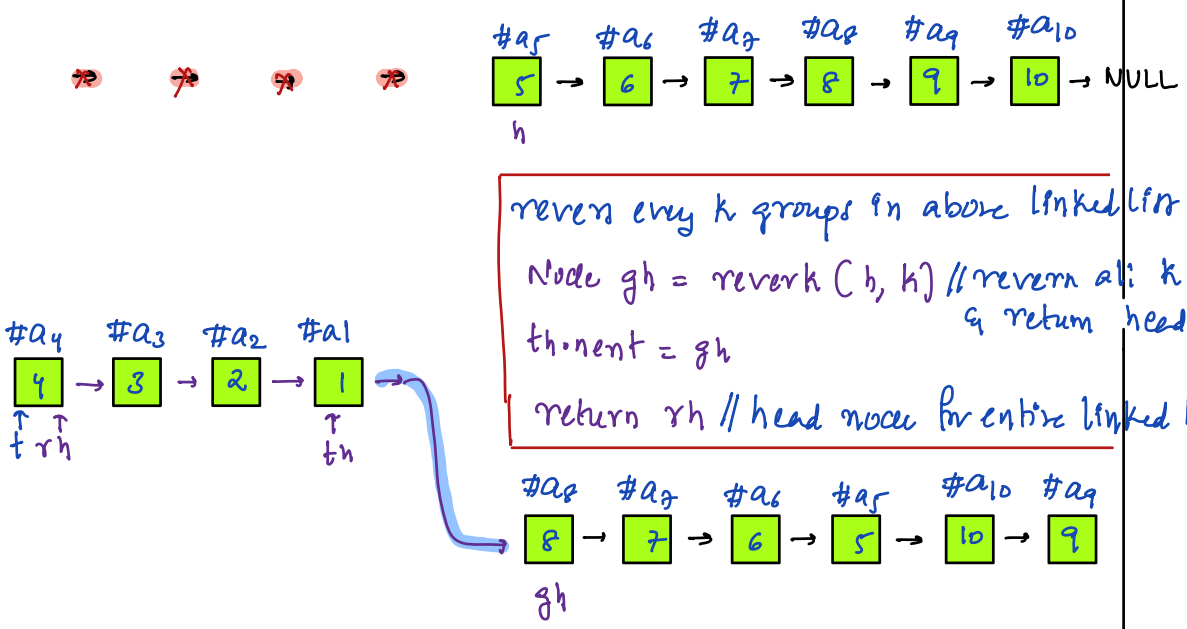
Note: If a group has less than k elements we still need reverse

Note: Recursion based extra space allowed

Ex: k=4:



Trace it: k=4



Ass: Given a linked list, revern all groups of size k & return head node

```
Node revernk(Node h, int k){ TC:O(N) SC:O(N/k)
    if(h==NULL || k<=1){ return h;}
    Node th=h
    Node rh=null, Node t=h;
    int c=k
    while(c>0 && h!=null){
        t=h
        h=h.next
        t.next=null
        t.next=rh
        rh=t
        c=c-1;
    }
    Node gh=revernk(h,k) } Reversing k groups in remaining linked list
    th.next=gh
    return rh
}
```

// reversing first k nodes

// Isolating & adding at start

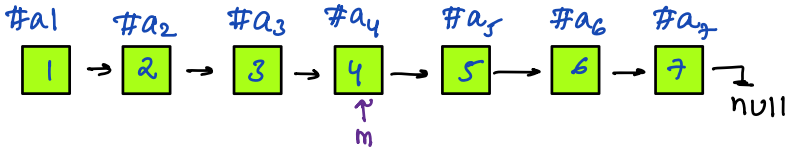
TODO: Only Iterative Code TODO

Note: reverse(h, k=2)

Swapping every 2 adj nodes

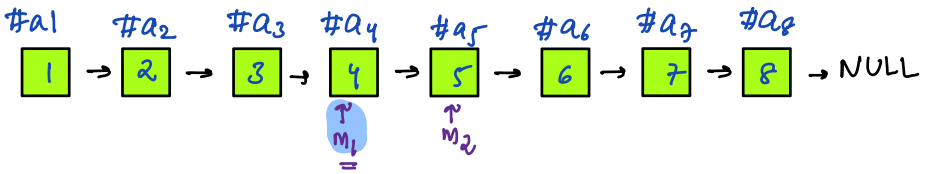
Q) Given head node find the mid of linked list

Ex1:



return 1st mid

Ex2:



Idea: → Iterate & get size of linked list : n
→ Again iterate from start & goto centre

TC: O(N)
SC: O(1)

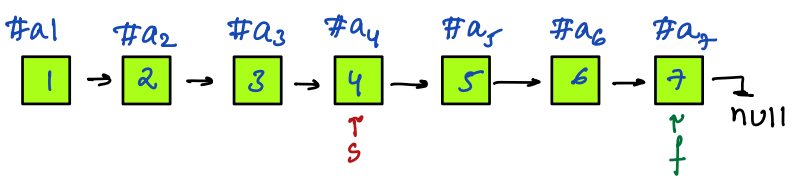
↳ 2 loops

Idea2: Find mid in a single loop:

Take s & f initialize them at head

: f = f.next.next // fast take 2 steps

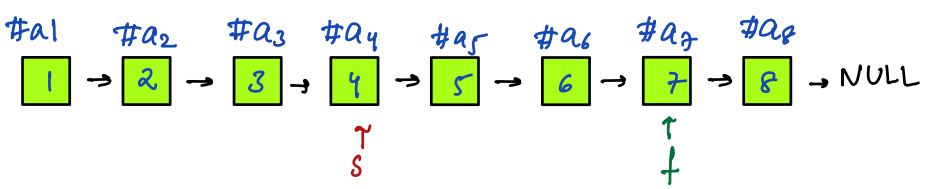
: s = s.next // slow take 1 step



Obs1: When f.next == null, stop & return s

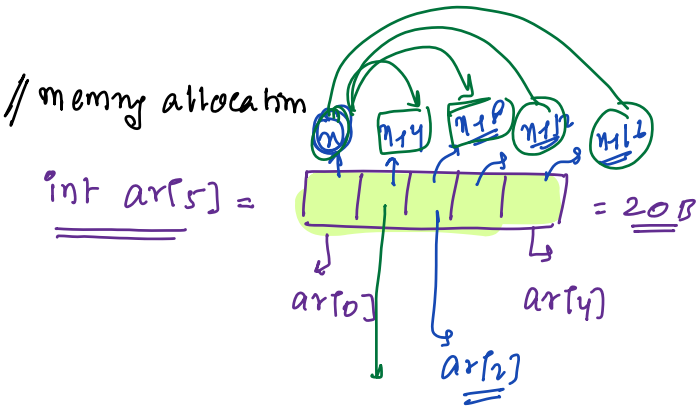
Obs2: When f.next.next == null stop & return s

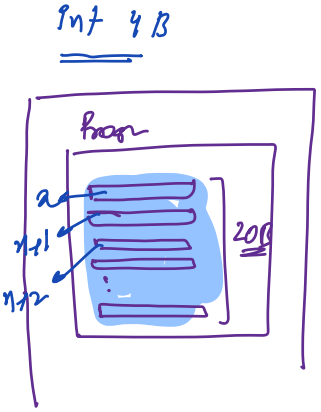
↳ Use both above conditions



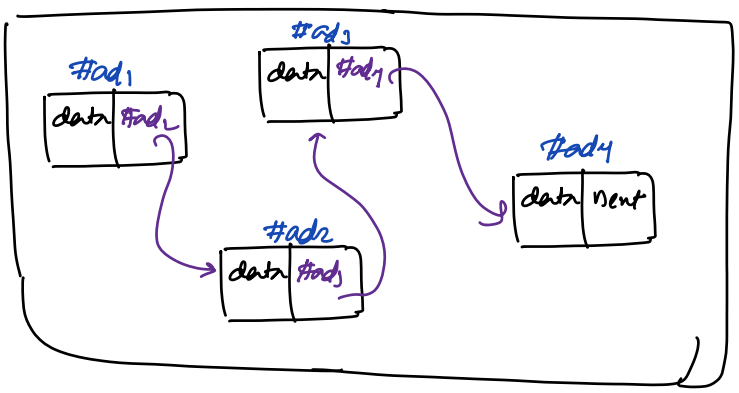
```
Node mid(Node h){ TC:O(N) SC:O(1)
    if(h==null){return h;}
    Node s=h
    Node f=h
    while(f.next!=NULL && f.next.next!=NULL){
        s=s.next
        f=f.next.next
    }
    return s
}
```

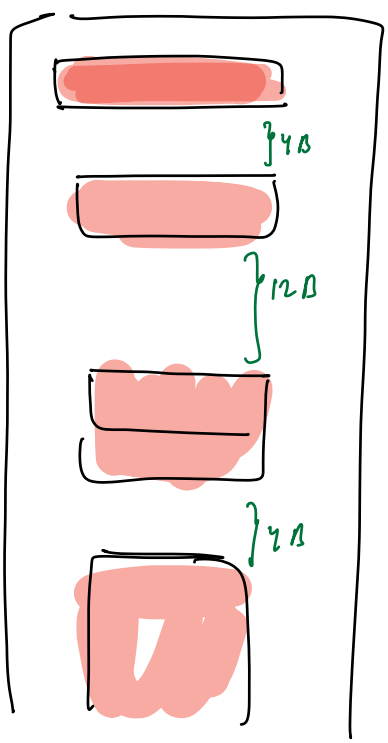
// memory allocation





Linked List:





array a[]

amortized analysis

TC: O(1) SC: O(1)