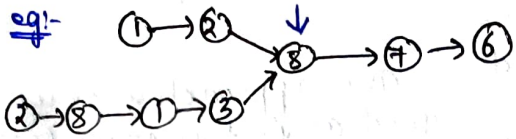
Day-6

1) Find intersection point of a LL

eg:-



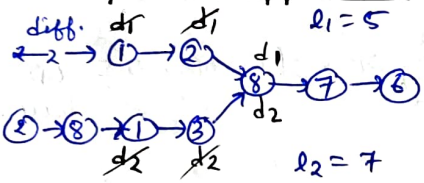
(1) Brute force :- for every node in (L1 ~~and~~ or L2), we'll check if there is the same node in (L2 ~~and~~ or L1) or not respectively.
(not value, reference of the node)

TC:- O(m×n) → we are picking one node in L1 and for that checking in L2
SC:- O(1)

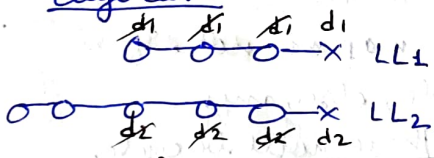
(2) Hashing :- So we we'll traverse in L1 and put the address of each node in map. Now traverse on L2 and we'll check if that address already exist in map or not. If yes then return that address. If no common address then return null.

TC:- O(n+m)
SC:- O(n)

(3) Optimal Approach :- (A) (lengths)
① Find length of Both LL (l_1, l_2)
② ~~move~~ Take P_1 and P_2 ~~on~~, put them of start of LL1 and LL2.
③ Move pointer of longer LL by ($l_2 - l_1$)
④ Now move move both pointers by 1-1 step. If $P_1 == P_2$ return P_1.



Edge case

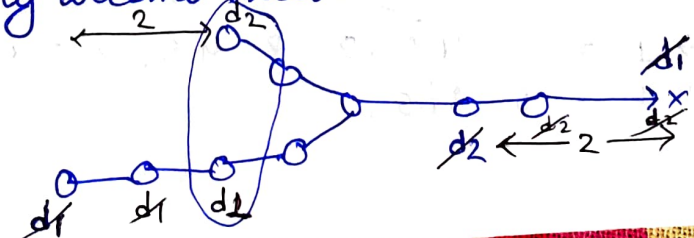


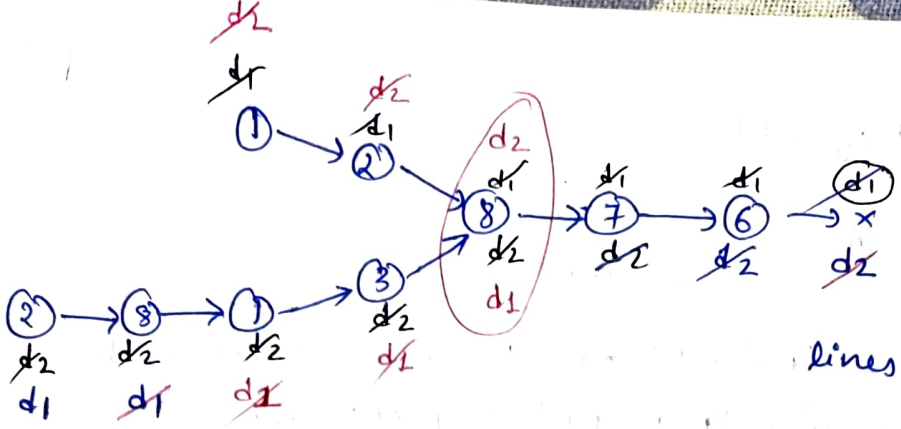
when they are not intersecting, then ($d_1 = d_2$) at null, so we need to return null.

TC:- O(M) + O(M-N) + O(N)
SC:- O(1)

(B) concise and crisp :- Take pointer P1 and P2 for LL1 and LL2,
→ move both pointer by 1 one 1 step, until any of them becomes null.
→ The moment any pointer becomes null, place that to head of another LL. and do same with another pointer.
→ Once they both have became null and traversing in another's LL, now either they meet (if they intersect) else both is simultaneously become null.

Intution :-
Both are in total travelling same length.





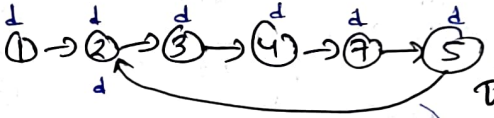
TC:- O(2n)
SC:- O(1)

If we didn't found intersection both pointer will traverse twice in L1's

This approach is very good as it has very less lines of codes.

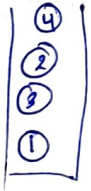
#code

```
ListNode* getIntersect(ListNode* headA, ListNode* headB){
    if(headA == Null or headB == null) return Null;
    ListNode* a = headA;
    ListNode* b = headB;
    while(a != b){
        a = a == null ? headB : a->next;
        b = b == null ? headA : b->next;
    }
    return a;
}
```

ii) **Detect cycle in a linked list** eg:-  o/p:- True.

eg:-  o/p:- false.

① **Brute force** :- (Hashing) Put the complete node (ie reference/address) in map and before puting check if it already exist in map or not.



Either pointer reaches null ⇒ no cycle, else, any node will be present in map and there will be cycle.

TC:- O(n)
SC:- O(n)

② **Optimise approach** :- (using fast & slow ptr.) Hare and tortoise

Move slow by 1 step and fast by 2 steps, if they both point at same node (which is quite certain if there is a cycle) then there is a cycle, else if fast becomes null ⇒ no cycle.

Intuition

TC:- O(n)
SC:- O(1)

So

slow →	1	2	3	4	5	2	3	4	5	2	3	4	5
fast →	1	3	5	3	5								

#code

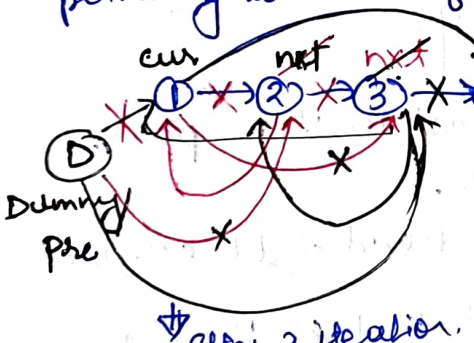
```
bool hascycle(ListNode* head){
    if(head==null or head->next==null){ return false;
    ListNode* fast, *slow;
    fast = slow = head;
    while( fast and fast->next){
        fast= fast->next->next;
        slow = slow->next;
        if( fast == slow){
            return true;
        }
    }
    return false;
}
```

iii > **Reverse Nodes in K-Group**

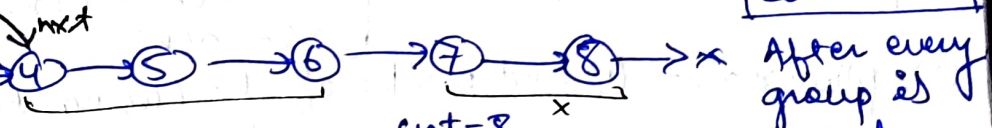
eg:- 1→2→3→4→5→6→7→8→x K=3

o/p:- 3→2→1→6→5→4→7→8→x

Approach:- Count no. of nodes in LL and take a dummy node pointing to head of LL. Perform these set of operations till Count >= K



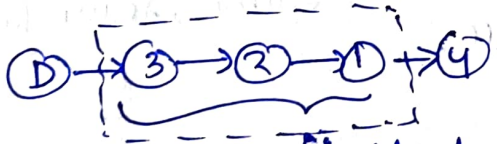
after 2 iteration.



After every group is reversed Count -= K.

```
cur->next = nxt->next.
nxt->next = pre->nxt;
pre->next = nxt
nxt = cur->next.
```

to reverse 3 node, we need to break 2 links hence in two iteration → reversed.



cnt -= 3 = 5

Similarly in after next 2 iteration.

D→3→2→1→6→5→4→7→ ∎.

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

#code

```
ListNode* reverseKgroup(ListNode* head, int K){
    if(head==null or K==1) return head;
    ListNode* dummy = new ListNode(0);
    dummy->next = head;
    ListNode* cur = dummy, *nex = dummy; *pre = dummy;
    int count = 0;
```

```
while(cur->next != NULL) {
    cur = cur->next;
    count++;
}
while(count >= K){
    cur = pre->next;
    nex = cur->next;
    for(int i=1; i<k; i++)
        cur->next = nex->next;
        nex->next = pre->next;
        pre->next = nex;
        nex = cur->next;
    }
    pre = cur;
    count -= k;
}
return dummy->next;
}
```

iv) Check if LL is palindrome or not

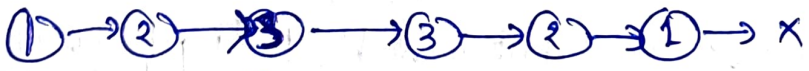
eg:- 1 → 2 → 3 → 3 → 2 → 1 → x op:- True.

① **Brute force:** – Put all node in an array and check if it is palindrome or not.

TC :– O(n) + O(n)
SC :– O(n)

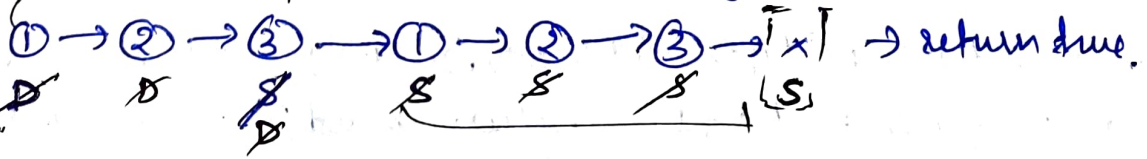
② **Optimised:** – ① find middle of LL. (using fast & slow ptr).
② Reverse the 2nd half of LL.
③ Now keep a dummy ptr. to head of LL and move slow by 1 step.
④ Now move dummy & slow by 1–1 step and check if they are same or not, at any point if they are not same, return false else if slow points to null return true.

eg:-

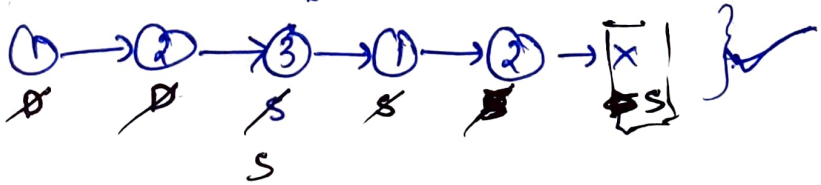


S (middle)

↓ after reversing.



eg:-



TC:- $O(n/2) + O(n/2) + O(n/2) + O(n/2) + O(n/2)$

- $O(n/2)$ → To find middle
- $O(n/2)$ → to reverse
- $O(n/2)$ → comparing
- $O(n/2)$ → again middle.
- $O(n/2)$ → again reverse.

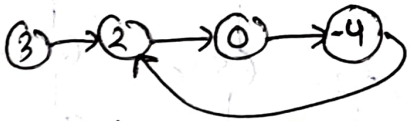
This optional, if interviewer wants the same LL back.

SC:- $O(1)$

#code

```
bool ispalin ( L* head){
    if(head==null or head->next==null) return true;
    L* slow=head, * fast=head;
middle  while( fast->next!=null and fast->next->next!=null){
            slow=slow->next;
            fast= fast->next->next;
        }
reverse slow->next= reverselist (slow->next);
    slow= slow->next;
comparing while(slow!=null) {
            if(head->val!=slow->val)
                return false;
            head=head->next;
            slow=slow->next;
        }
    return true;
}
```

v> **Find the starting point of the cycle**



o/p:- node (2)

① Brute force:- (Hashing) Create a ~~hash~~ map and put the complete map on the LL. Before putting in map check if that node already existed in map or not. If yes, then ~~that~~ return that node

TC:- O(n)
SC:- O(n)

② optimised:- using (Hare and tortoise Algo).

① find collision point ⇒ Move slow by 1 step and fast by 2 step until they point to same node.

② Now move fast ptr again to head. Now move fast and slow both by 1 step.

③ Now the point where both slow & fast ptr again meet is the starting point of cycle.

④ If at any moment fast becomes null ⇒ no ~~starting~~ cycle in LL

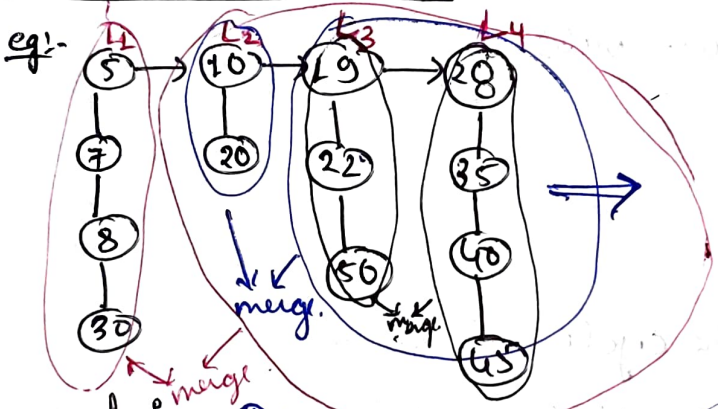
TC :- O(N)
SC :- O(1)

#code

```
L* detectcycle(L* head){
    L* fast = head, *slow = head;
    while(fast != null && fast->next != null){
        slow = slow->next;
        fast = fast->next->next;
        if(fast == slow) break;
    }
    if(fast == null or fast->next == null) return NULL;
    while(*slow != fast){
        slow = slow->next;
        fast = fast->next;
    }
    return fast;
}
```

vi> flatten of a LL (it has next and bottom pointer). (form one LL which has only bottom ptr and LL is in sorted order).

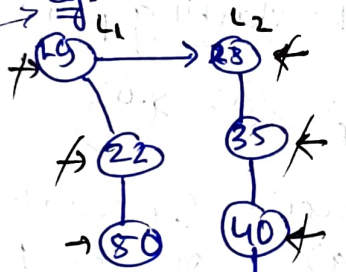
eg:-

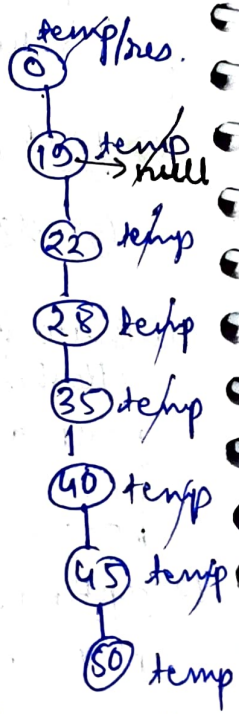


→ next of every node is null.

Approach:- Take a dummy ptr and put res/temp at dummy. Idea to solve this question is we'll take 2 node in LL (which has a LL in their bottom) now treat this as we are given 2 sorted LL and we need to merge them. Similary take 2 and merge do till we have only one node left.

eg:-



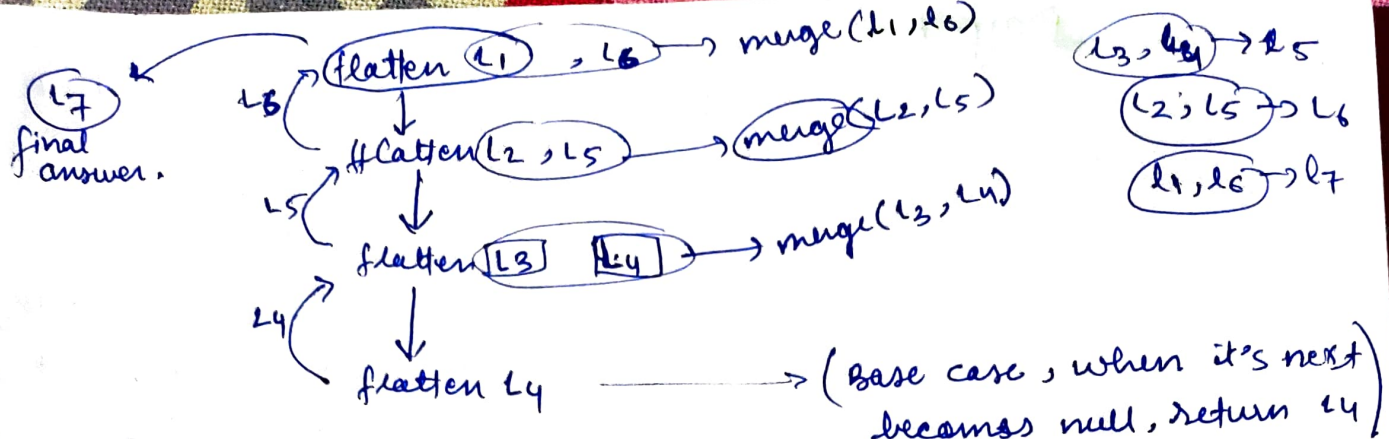


If any pointer reaches null then attach the remaing part of other pointer in it.

After this put :—

res → bottom → next = null.

We'll use recursion to solve this.



TC:- O(Total no. of nodes)
SC:- O(1)

#code

```
node* flatten(node* root){
    if(root==null or root->next==null)
        return root;
    root->next = flatten(root->next);
    root = merge(root, root->next);      //code for merging two sorted LL.
    return root;
}
```