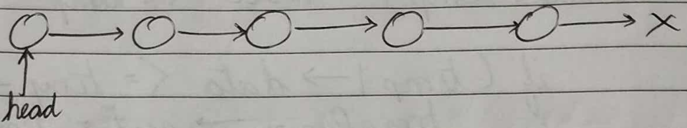
DATE

31 Linked List III

Q I/P

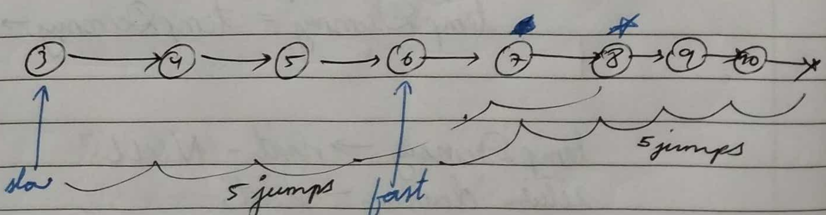


print k th node from end of LL

1. Reverse & k th
2. Print (length - k)
3. Put into array
4. Recursion
   ⇕
5. Tail Recursion

6 Gap Technique (2 pointers)

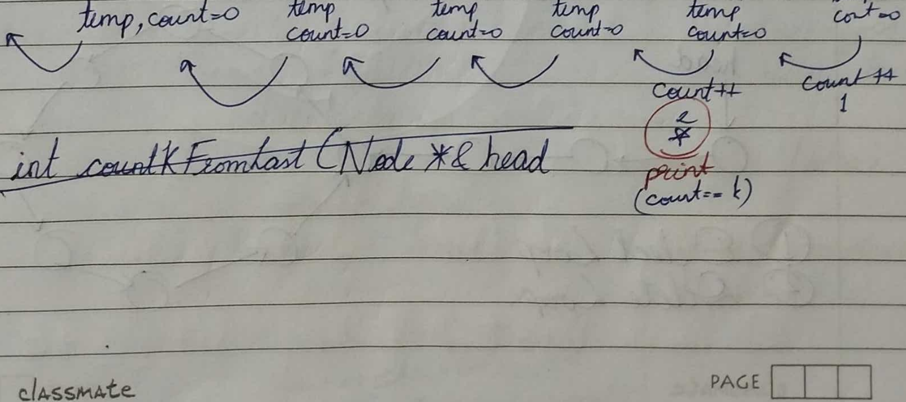
k = 3



```
① Slow → head,   fast → head+k
② while ( fast != NULL)
      fast ++
      slow ++
③ return slow.
```

```
slow = head
fast = head

while (k--)
    fast = fast → next

while ( fast != NULL) {
      slow = slow → next
      fast = fast → next;
}
return slow;
```

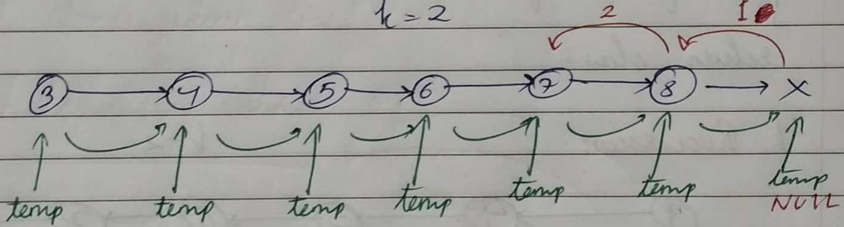
4. Recursion



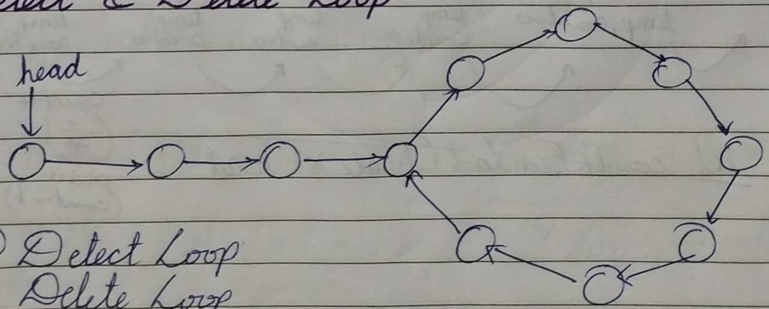
int countKFromLast (Node *& head

DATE

```
int countKFromLast(Node* &head,
                   ~~int count~~, int k) {

    if (head == NULL)
        return 0;

    int cnt = countKFromLast(head->next, cnt, k)
                                            +1;

    if (cnt == k)
        cout << head->data;

    return cnt;
}
```



Q Detect & Delete Loop



① Detect Loop
② Delete Loop

DATE

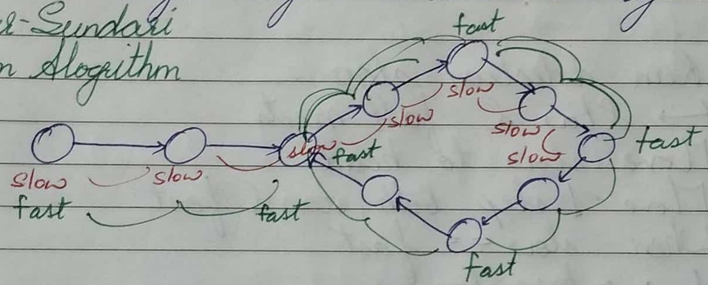
Techniques

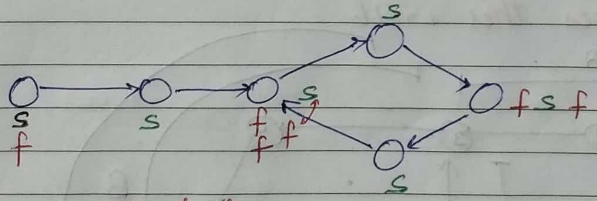
① Use map to store address
↳ if same address found again
↳ loop

② Modify given nodes
↳ if modification found
↳ loop.

③ Floyd Cycle Detection Algorithm

Sundar-Sundari
Detection Alogrithm





Loop Detection

```
bool detectLoop(Node * &head) {
    if (head == NULL)
        return false;

    Node * slow = head;
    Node * fast = head;

    while (fast && fast→next)
        slow = slow→next;
        fast = fast→next→next;
}
```

```
if (slow == fast)
    return true

return false;
```

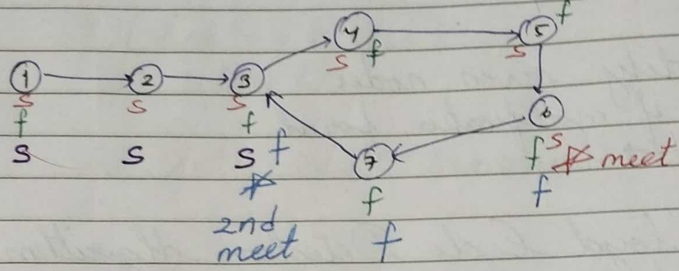
Time Complexity
$O(n/2 + k/2)$

classmate

PAGE

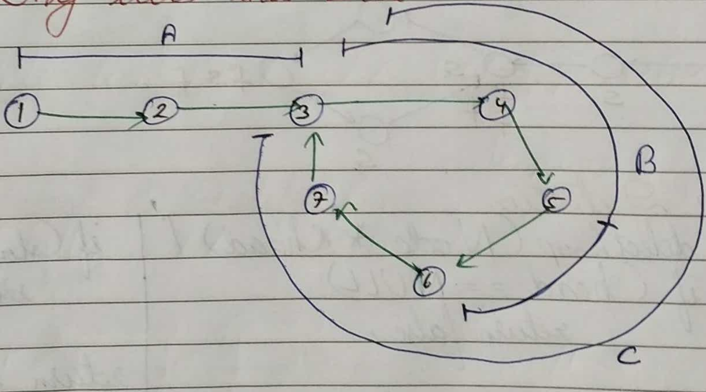
DATE

Find loop point (Intersection)



1. Run detection algo
   ↳ slow & fast meets
2. Move slow to head
3. Move 1 step both slow & fast
4. When slow == fast
   ↳ return slow.

Why does this work



Distance of fast = 2 * distance of slow

Distance travelled by fast = $A + \underbrace{xC}_{\text{x times looped in cycle}} + B$

DATE

Dist of slow · $2A + 2B + yC$

$$A + B + xC = 2A + 2B + yC$$

$$xC - yC = A + B$$

$$(x-y)C = A + B$$

$$\Rightarrow A + B = k * C$$

or $A = kC - B \Rightarrow$ head ⟼ intersection2
═ equals
intersection1 ⟼ intersection 2

Delete Loop

```
temp = intersection
temp2 = intersection
while (temp2 → next != temp)
        temp 2 = temp 2 → next ;
temp 2 → next = NULL;
```

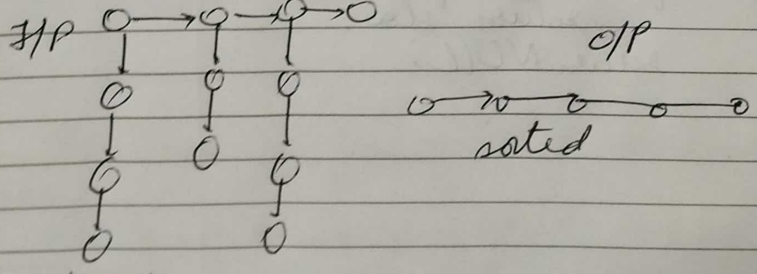
```
Node * deleteLoop(Node * ________ ) {

        ══════  same as before

        if (slow == fast)
            return slow;
        return NULL;
}
```

classmate PAGE

```
if (head == NULL)
    return false;

bool detectAndDeleteLoop (Node *&head) {

    Node * fast = ~~head~~ detectLoop(head);
    Node * slow = head;
    while ( fast != slow) {
        slow = ~~sl~~ slow → next;
        fast = fast → next;
    }

    Node * begin = slow;
    Node * temp = begin;

    while (temp → next != begin)
        temp = temp → next;

    temp → next = NULL
    return true;
}
```

H/W Intersection Point of LL (4 Approaches)
↓
loop, visitedflag,..

H/W Flatten a LL (recursion)



DATE

H/W Remove Duplicate from sorted/unsorted
H/W Add 1 to LL
H/W Add 2 LL
H/W Clone LL with random ptr
H/W Delete m nodes after n nodes.
H/W Reverse alternate k nodes.

Q Why is is quick sort preffered for arrays and why merge sort is preferred for LL?

classmate

PAGE