Agenda

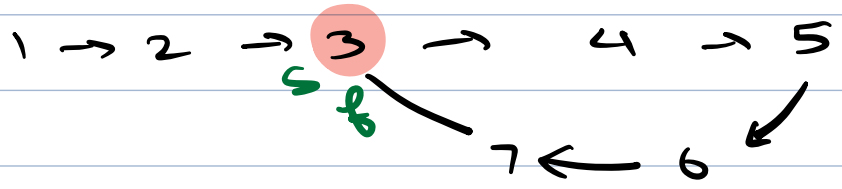
1. What is doubly LL
2. How is doubly LL different from singly LL
3. Problems
   - Insert a node
   - Delete a node
   - Cache
   - Check for Palindrome

Revision Quiz 1

TC to merge 2 sorted linked lists:

TC: O(N+M)

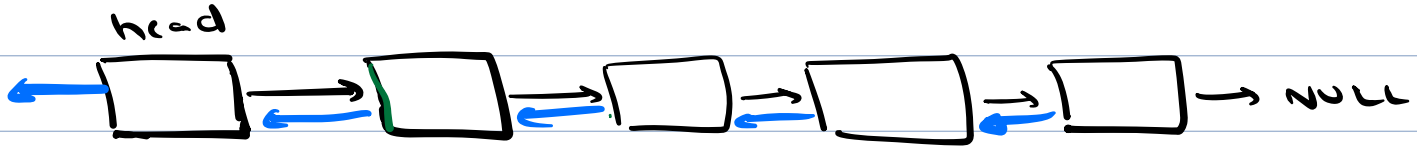
Revision Quiz 2



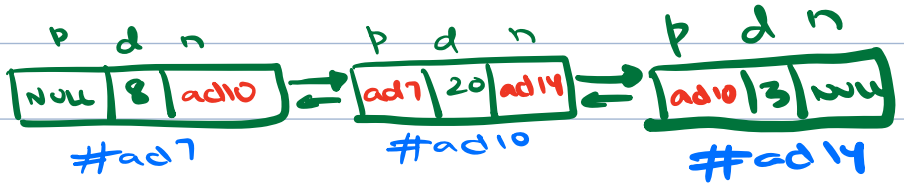
Linked list is a data structure to store and organize a collection of elements (in nodes).

Doubly Linked List

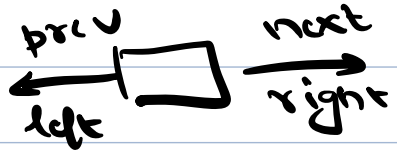
A doubly linked list is similar to a singly linked list but with an additional feature each node contains references to both next & previous nodes in the list.




```
Class Node {
    int data
    Node next
    Node prev
    Node(int x) {
        data = x
        next = NULL
        prev = NULL
    }
}
```



N ← 8 ⇄ 14 ⇄ 3 → N



① head.prev = NULL

② next of last node = NULL

Correlation with Singly Linked List

The main difference lies in the bidirectional traversal capacity of doubly linked list (which comes at a cost of increased memory usage)

* prev → left ; next → right

Implement playlist feature using doubly LL

↓

① Add song

② Skip to prev and next song

<< (11) >>

③ current song

Add (1, "Love story") ✓

Add (2, "Imagine") ✓

Play Next ✓

Current ✓ Imagine

Add (3, "California") ✓

Play Next ✓

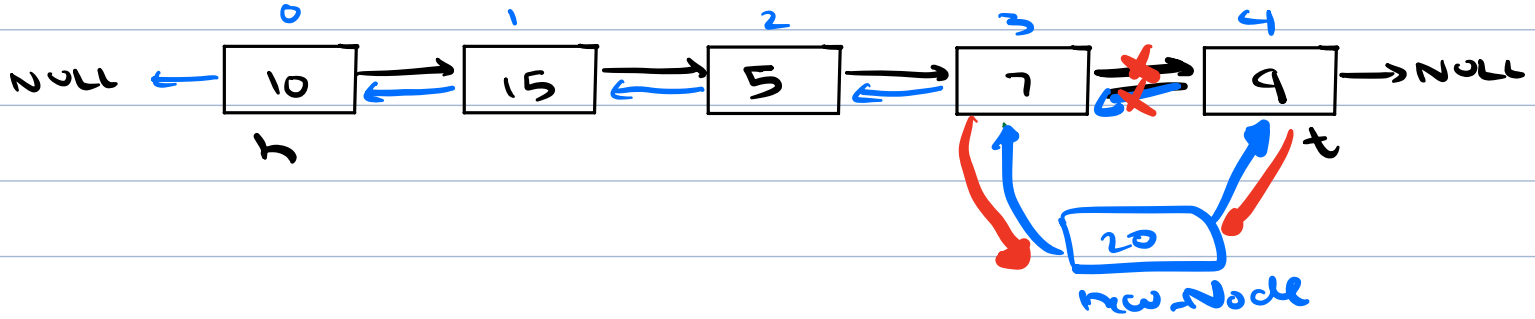
Current ✓ Hotel California

Play Previous ✓

Play Previous ✓

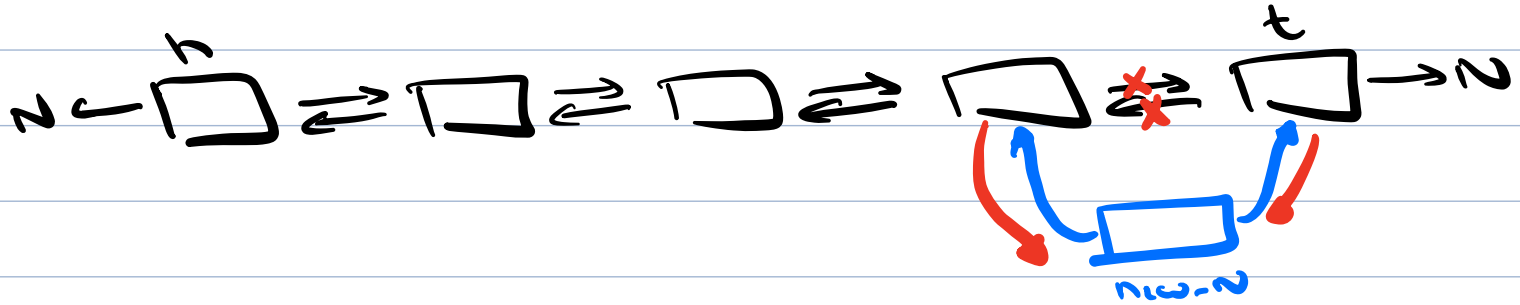
Current ✓ Love story

Q. Insert a node just before tail in DLL



1. newNode.next = t
2. newNode.prev = t.prev
3. t.prev = new_Node
4. new_node.prev.next = new_Node

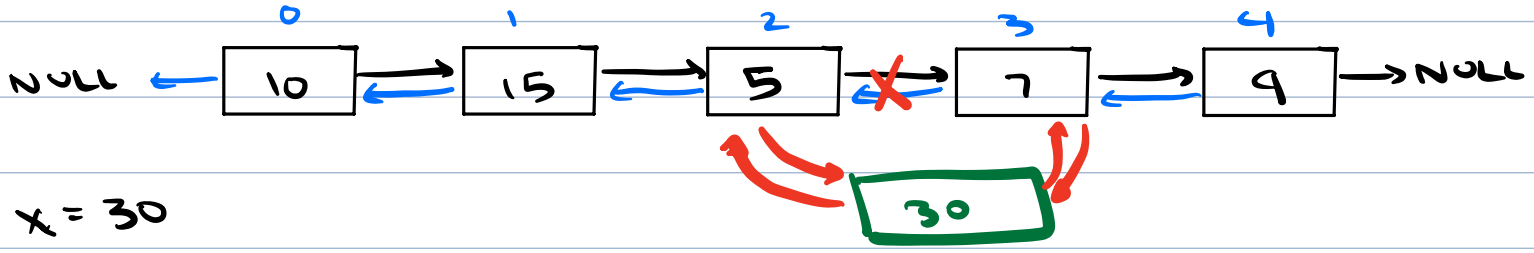
2nd last node.next = new_Node



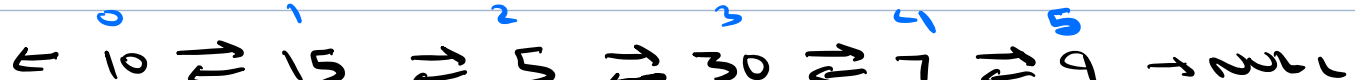
1. Insert node in a doubly linked list

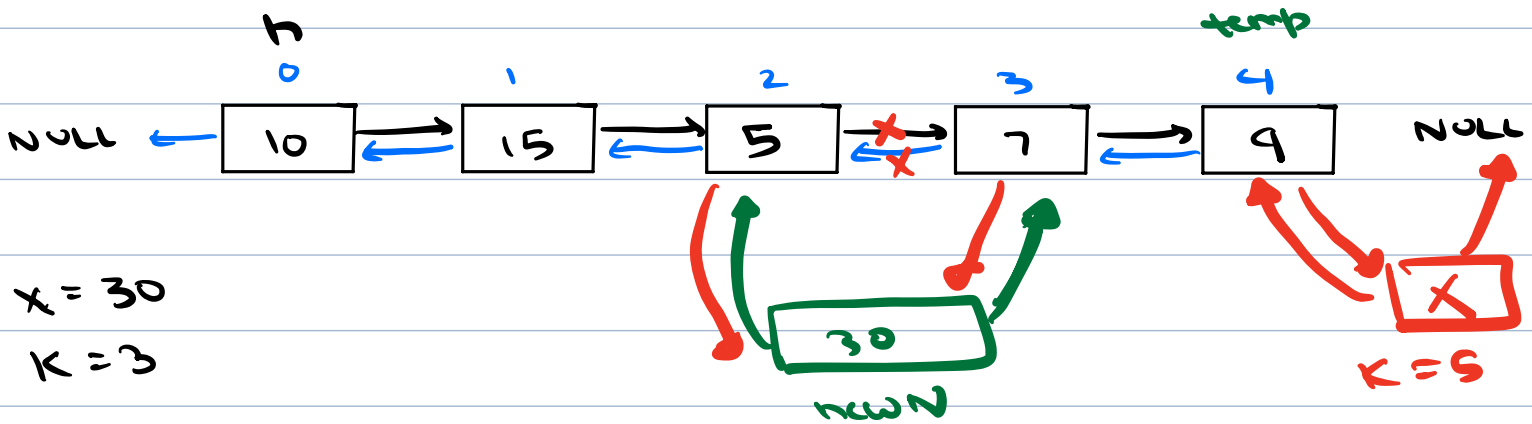
A node is to be inserted with data x at position k. Range of k is between 0 and N where N is length of doubly LL.

N = 5
0 ≤ k ≤ 5



x = 30
k = 3

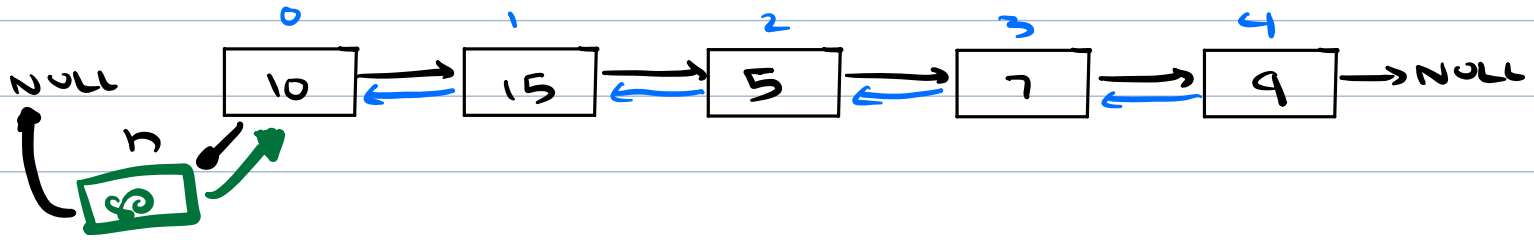




x = 30
k = 3

① Create newNode with data x
② Traverse till k-1 idx → jumps
③ newN.prev = temp
newN.next = temp.next
temp.next = newN

if (newN.next != NULL) {
newN.next.prev = newN
}



x = 80
k = 0

① Create new Node

② newNode.next = head
head.prev = newNode

③ head = newNode

```
Node insert(Node head, int x, int k) {
    Node newN = new Node(x)
    if (head == NULL) return newN

    if (k == 0) {
        newN.next = head
        head.prev = newN
        head = newN
        return head
    }
    else {
        // Reach k-1 idx
        Node temp = head
        for (jump=1 ; jump ≤ k-1 ; jump++)
            temp = temp.next
```

```
newN.prev = temp
newN.next = temp.next
temp.next = newN
if (newN.next != NULL) {
    newN.next.prev = newN
}
return head
}
```

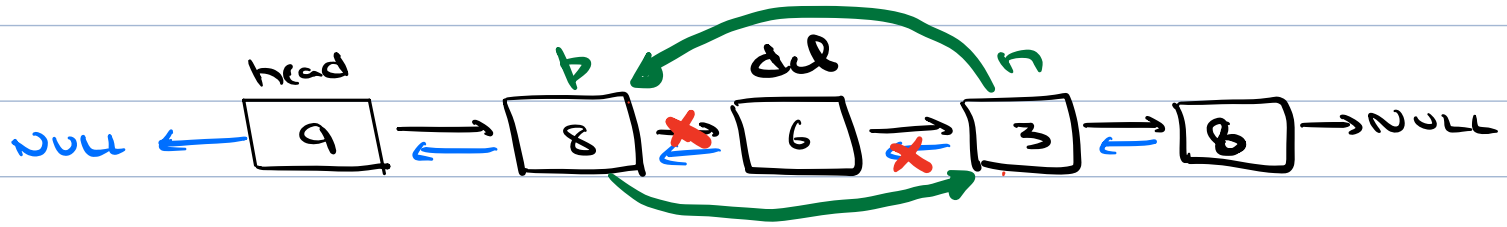
TC: O(K)

worst case → O(N)

SC: O(1)

2. Given a doubly LL of length N, delete a given node from it.

1. Node reference is given
2. Node will not be head / tail
3. DLL is not NULL



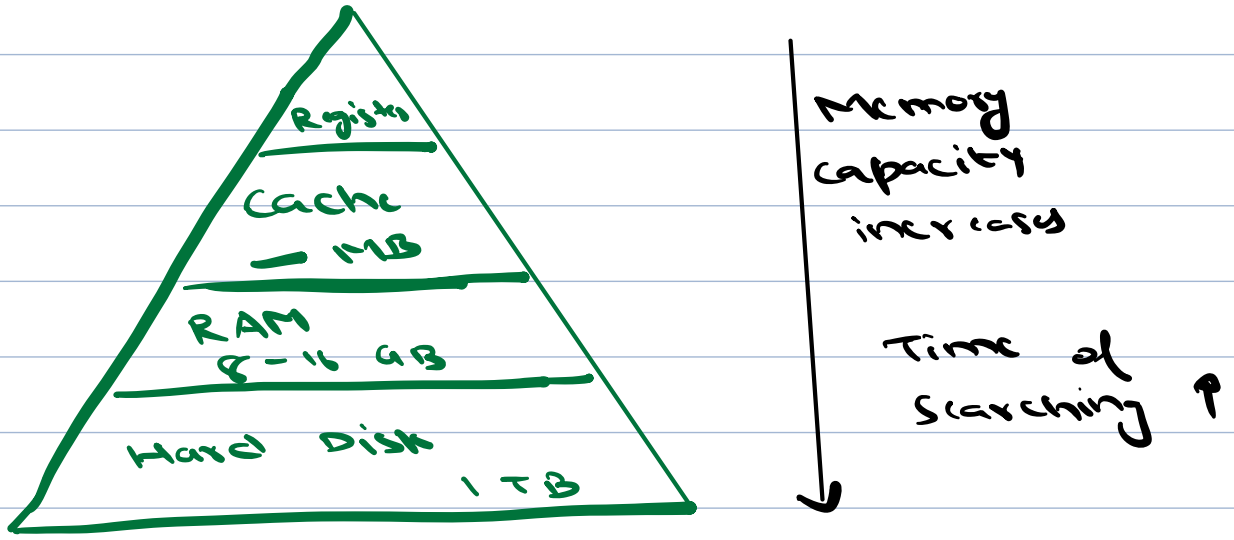
del = 6

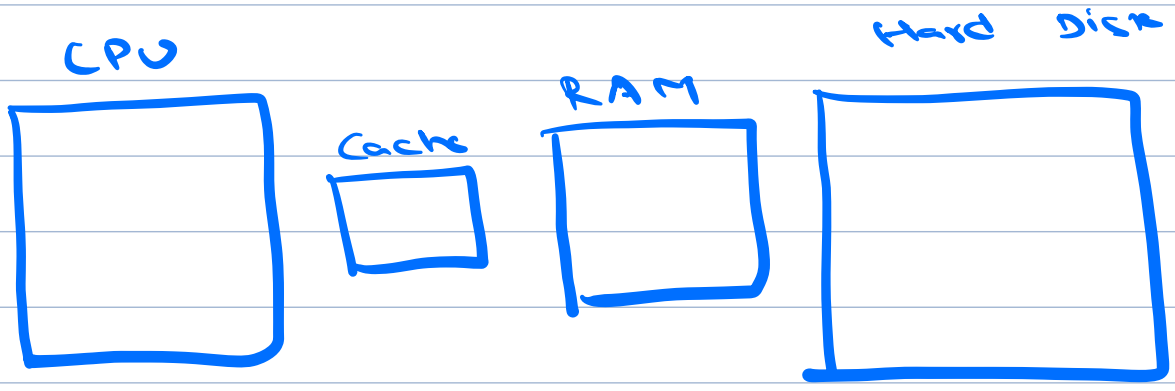
ans :


```
void deleteNode(Node del){
    ① Node p = del.prev
    ② Node n = del.next
    ③ p.next = n
    ④ n.prev = p

      free(del)
}
```

TC : O(1)





LRU Cache

Least Recently Used Cache

10:35 Break

https://notability.com/n/o4OSLf6C8RAn_sLFraqUZ

3. Given a running stream of integers and fixed memory size of M, we've to maintain most recent M elements. In case current memory is full, delete the least recent element and insert current data into the memory (as the most recent item).

7, 3, 9, 2, 6, 10, 14, 2, 10, 15, 8, 14, 10

M = 5

Least Recent

Most Recent Element

N

Search (N)

HIT

Miss

delete (N)

insert - last (N)

Size == M (capacity)

Y

N

delete_front()
insert_last(x)

insert_last(x)

	Array	Deque	LL
search(x)	O(N)	X	O(N)
insert_last(x)	O(1)	O(1)	O(1) if tail
delete_front(x)	O(N)	O(1)	O(1)
delete(x)	O(N)	X	O(N)

	DLL	DLL + HM
search(x)	O(N)	O(1)
insert_last(x)	O(1)	O(1)
delete_front(x)	O(1)	O(1)
delete(x)	O(1)	O(1)

data, reference

HM<int, Node> hm

Node h = new Node(-1)

Node t = new Node(-

h.next = t

t.prev = h

```
void insertCache(int x) {
    if (hm.contains(x)) {          // HIT
        Node temp = hm[x]
        deleteNode(temp)
        insertAtTail(x)            <- create a nn with data x
                                      put in hashmap
    }
    else {                         // miss
        if (hm.size == M) {        // full
            hm.delete(head.next.data)
            deleteNode(head.next)
            insertAtTail(x)
        }
        else {
            insertAtTail(x)
        }
    }
}

void insertAtTail(x) {
    Node nn = new Node(x)
    hm.put(x, nn)
```

nn.next = t

nn.prev = t.prev

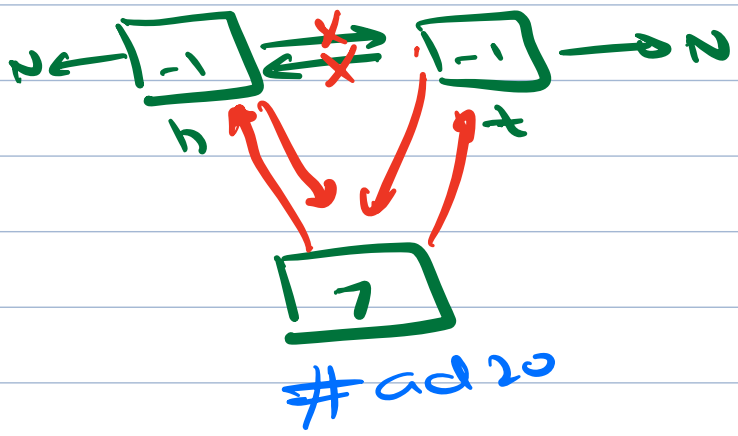
t.prev = nn

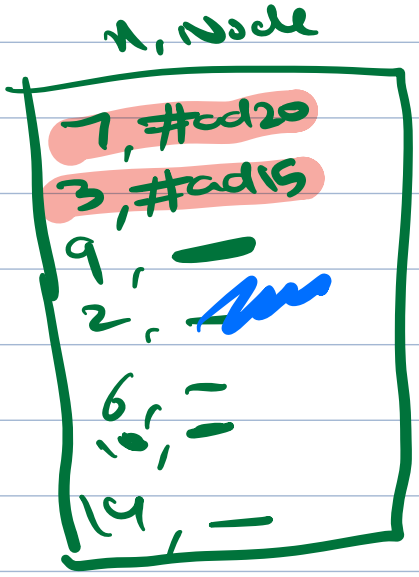
nn.prev.next = nn

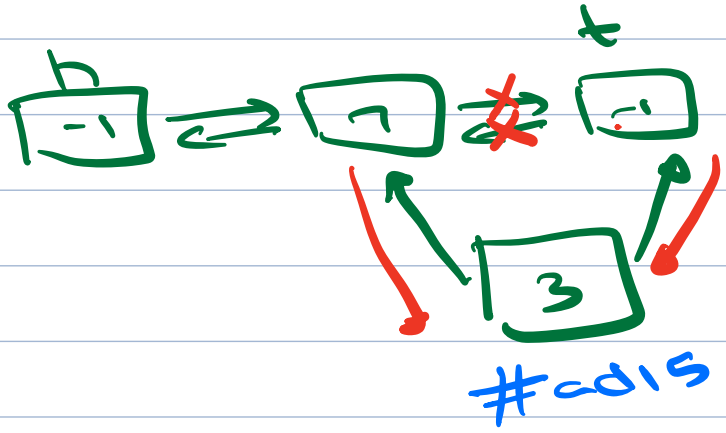
t

7, 3, 9, 2, 6, 10, 14, 2, 10, 15, 8, 14, 10

M = 5







-1 ⇄ 7 ⇄ 3 ⇄ -1

-1 ⇄ 9 ⇄ 6 ⇄ 10 ⇄ 14 ⇄ 2 ⇄ -1

Size is M, N int

Delete → TC is O(1)

TC: O(N)

SC: O(M)

4. Given a LL, check Palindrome

h
2 → 5 → 8 → 7 → 3 → NULL ans = false

h
2 → 5 → 8 → 5 → 2 → NULL ans = true

1. Go to mid, break LL from mid
2. Reverse 2nd half
3. Compare 1st and 2nd half

```
Node mid = findMiddle(head)
Node head2 = mid.next
mid.next = NULL
head2 = reverse(head2)
```

h mid h2
1 → 2 → 2 → 1

1 → 2
h2

```
Node temp = head, temp2 = head
while (temp != NULL && temp2 != NULL){
    if (temp.data == temp2.data){
        temp = temp.next
        temp2 = temp2.next
    }
```

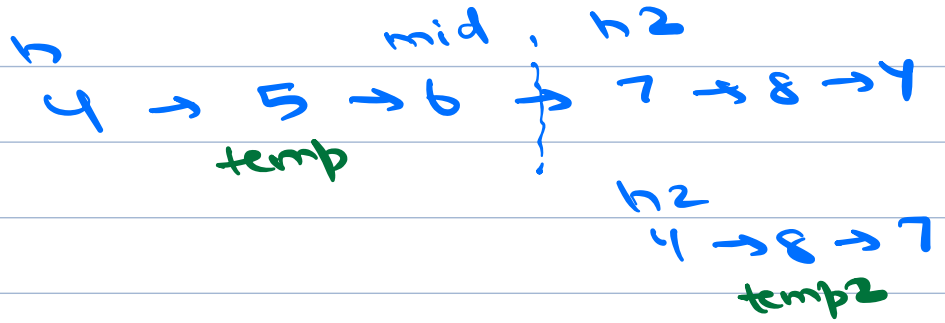
else

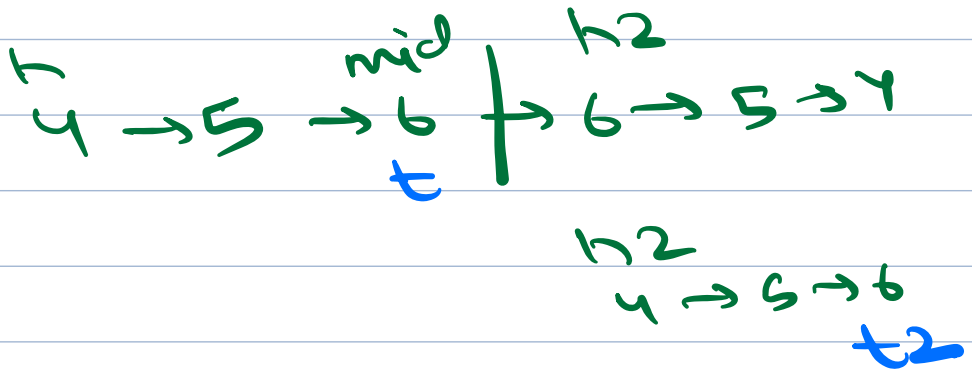
return false

return true

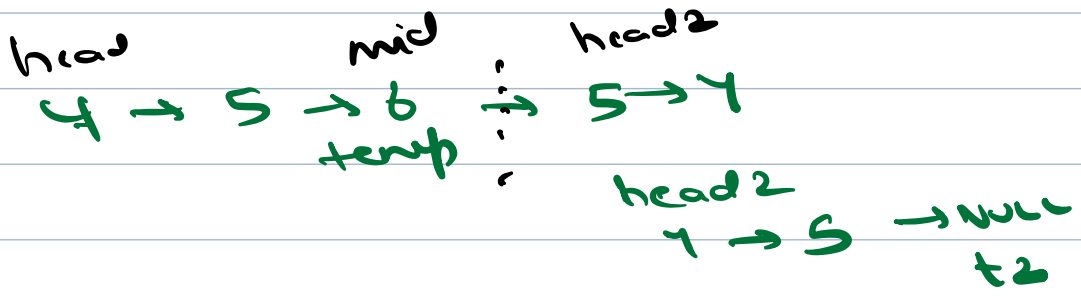
TC: O(N)
SC: O(1)

N=6





N=5



LL is even → Palindrome → not a palindrome

LL is odd → Palindrome → not a palindrome