Q1 Check if a given list is Palindrome.

Ex1 $1 \to 3 \to 5 \to 1 \to null$

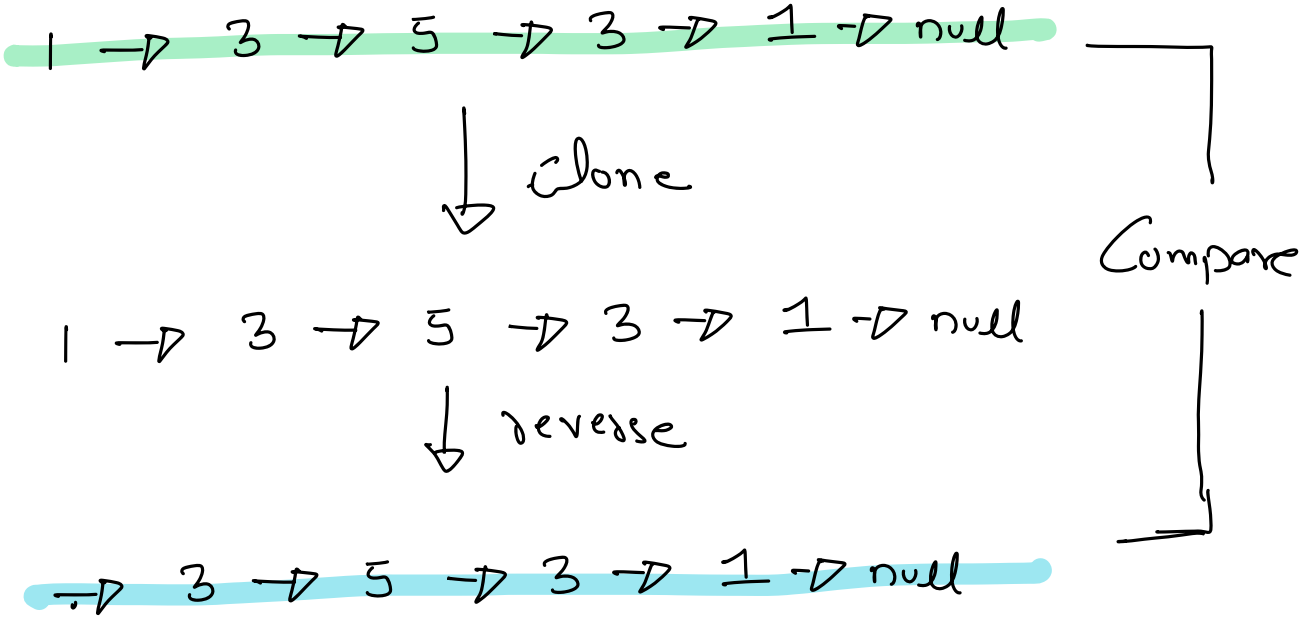
$\to$ false

Ex2 $1 \to 3 \to 5 \to 3 \to 1 \to null$

$\to$ true.

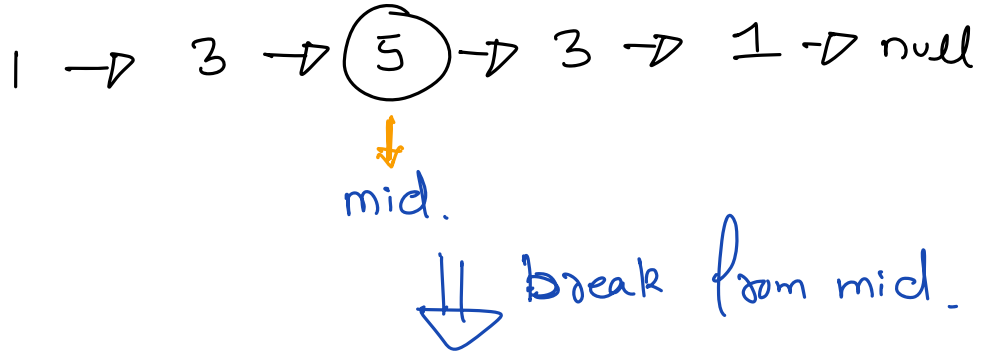
Approach 1:

1) Make a clone of linked list
2) Reverse the 2nd linked list
3) Compare both lists.

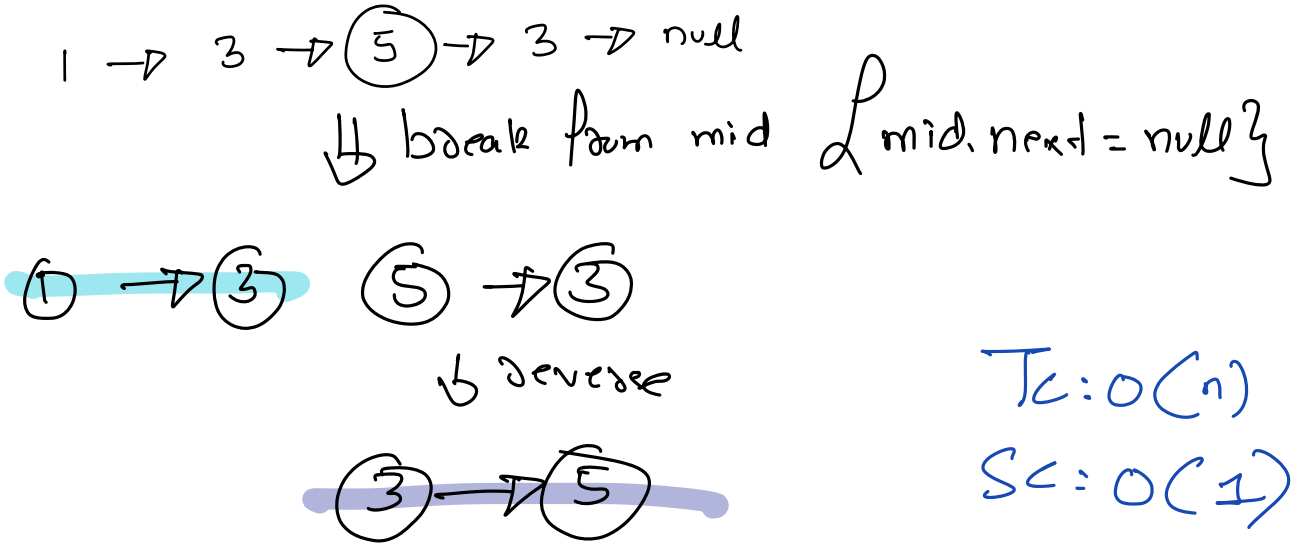


TC: O(n) SC: O(n)

Approach 2: ODD Length



EVEN Length



TC: O(n)

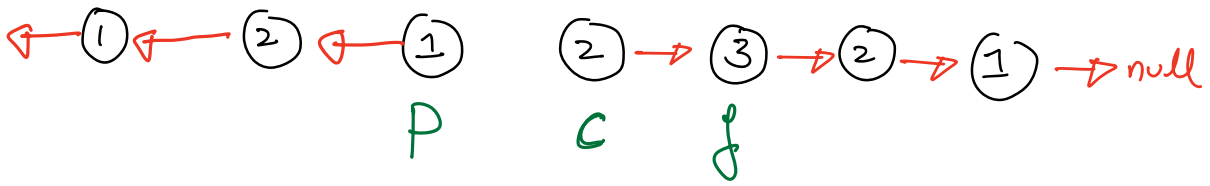
SC: O(1)

Q2 Find length of longest odd length palindromic list in a given linked list.

SC : O(1)

Ex1 

ans = 3 !



curr.next := P

P = C

C = f.

Pseudo Code !

```
if (head == null)
    return 0;
```

```
Node curr = head;
Node prev = null;
Node fut = null;
int ans = 1
while (curr != null) {
    fut = curr.next;
    ans = max(check2(prev, fut), ans);
    curr.next = prev;
    prev = curr;
    curr = fut;
}
return ans;
```

```
int check2() {
Node p1 = prev;
Node p2 = fut;
int cnt = 1
while (p1 != null &
       p2 != null) {
  if (p1.val ==
      p2.val)
    cnt = cnt + 2;
  else
    break;
  p1 = p1.next;
  p2 = p2.next;
}
return cnt;
}
```

$TC : O(n^2)$

$SC : O(1)$

TODO:

Q1. Find length of longest even length palindromic list in a given linked list.

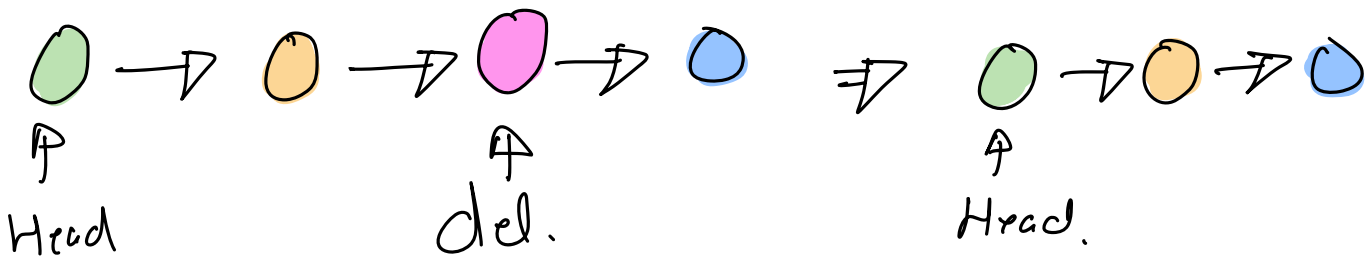
SC: O(1)

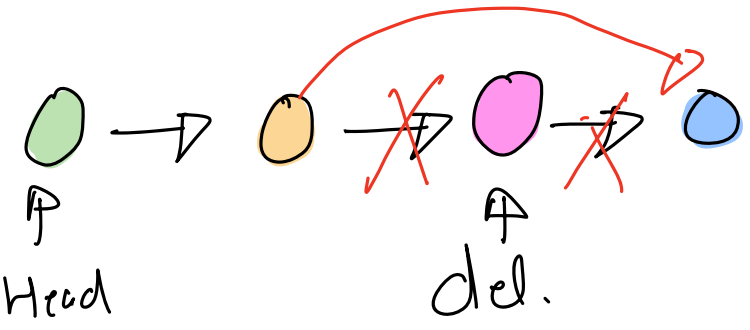
Q2. Find length of longest length palindromic list in a given linked list.

SC: O(1)

↳ Present in assignment

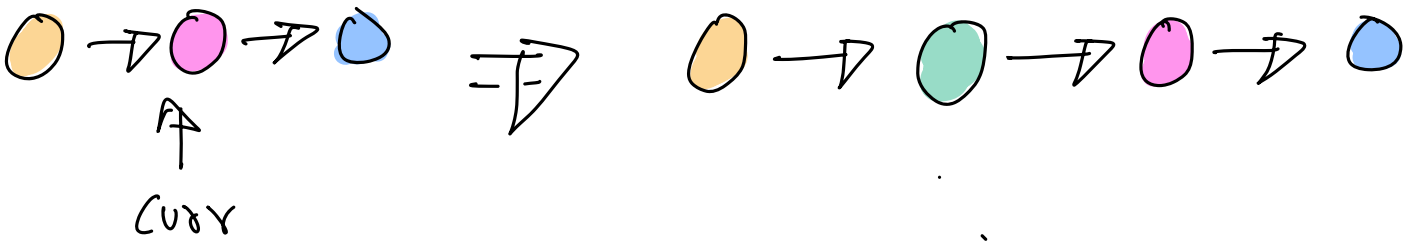
Q Given a reference to a node. Delete it:





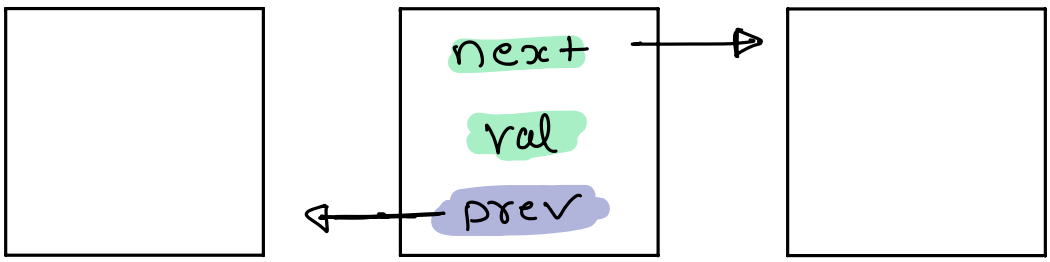
TC: O(n)

Q Given a reference to a node. Insert a new node before it.

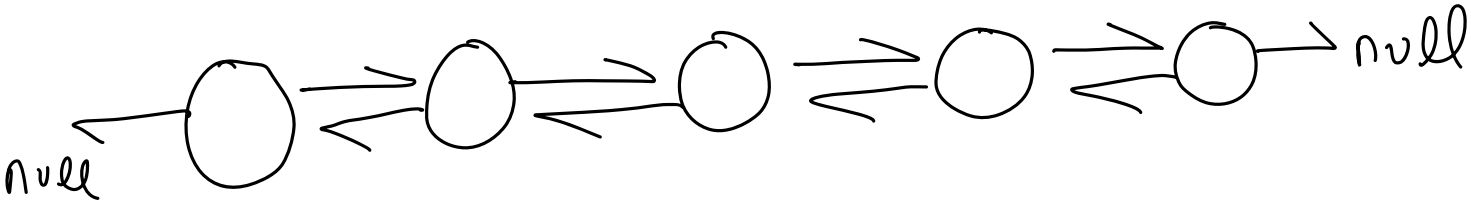


TC: O(n)

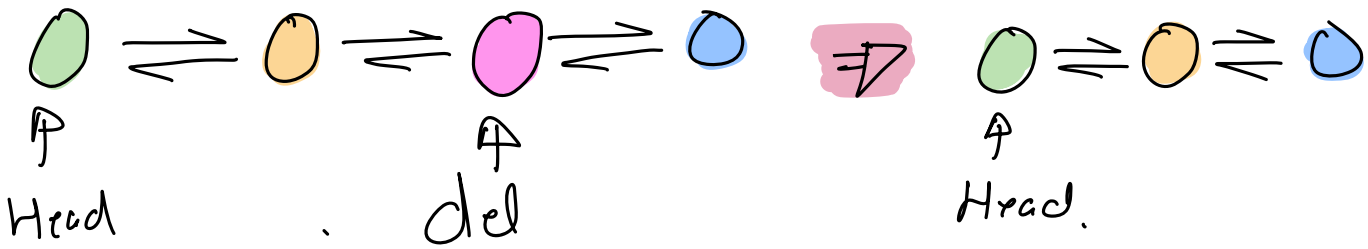
Doubly Linked List

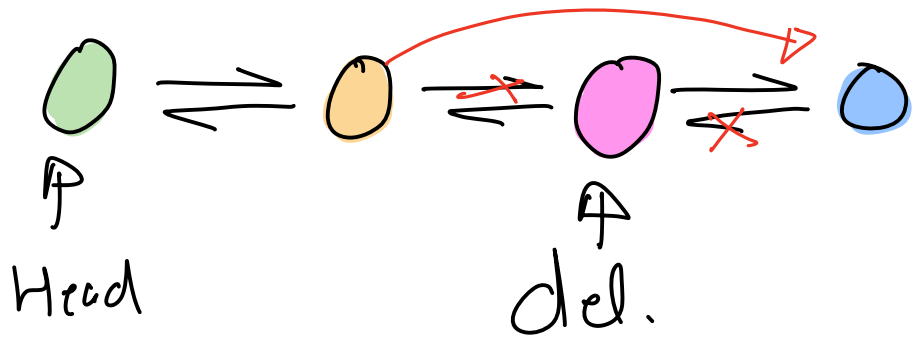



```
class Node {
    int val;
    Node next;
    Node prev;
}
```



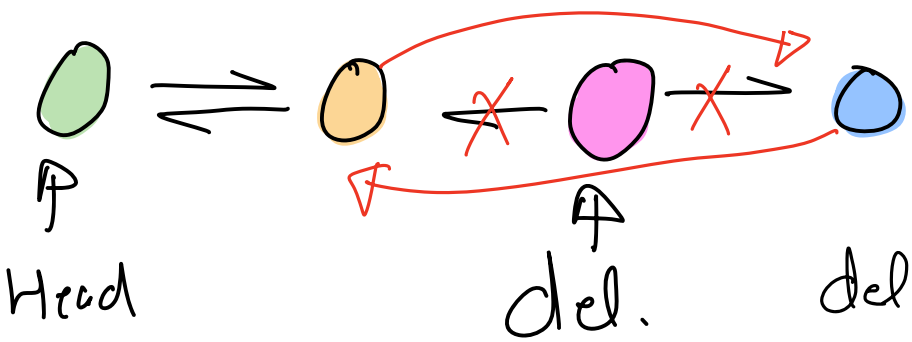
Q Given a reference to a node. Delete it:
You are given a doubly linked list.





del.prev.next = del.next

del.next.prev = del.prev.



del.prev = null

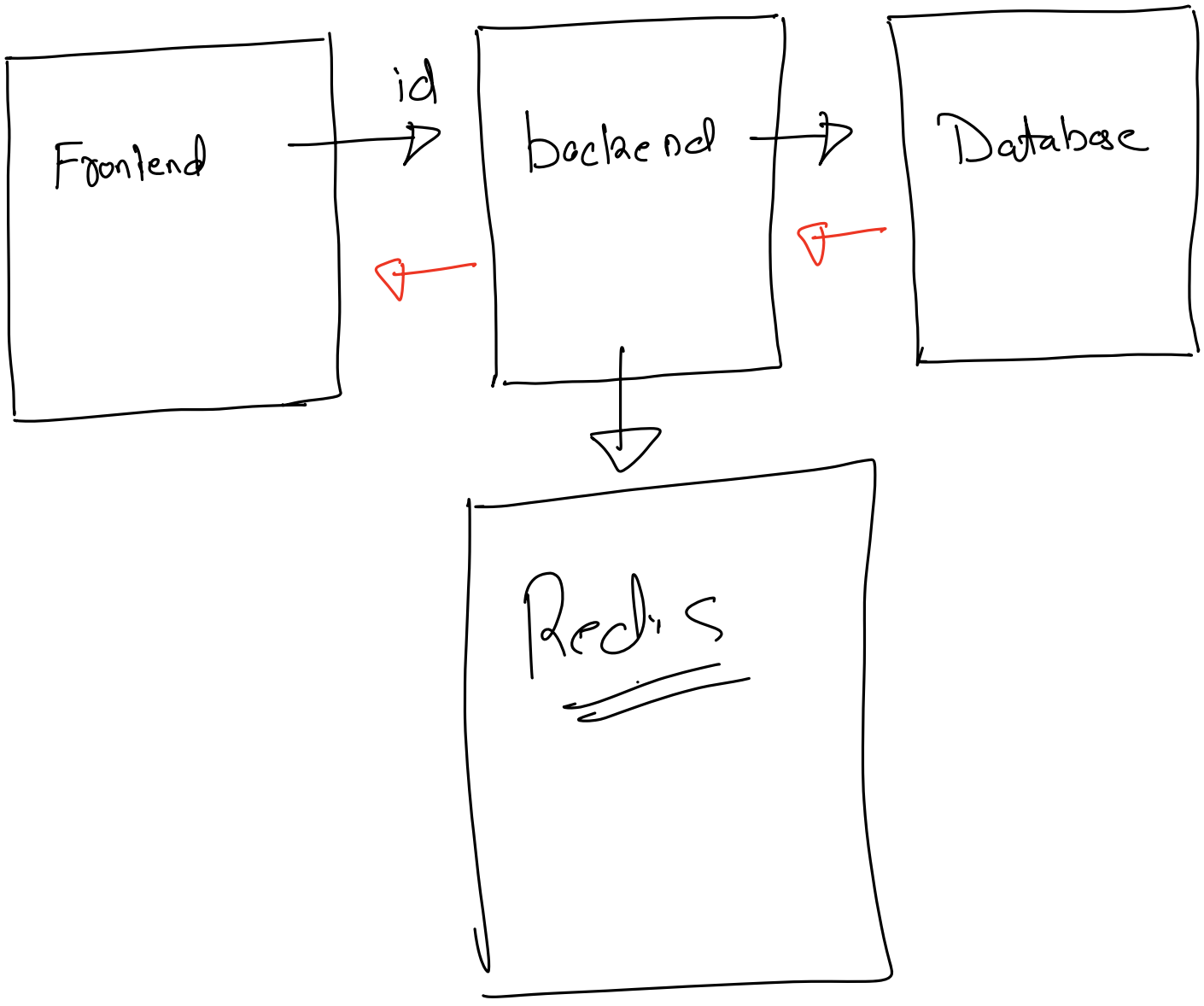
del.next = null

Pseudo Code !

```
if (del == null)
    return head.

if (del.prev != null)
    del.prev.next = del.next

if (del.next != null)
    del.next.prev = del.prev.

if (del == head)
    head = del.next

del.prev = null
del.next = null

return head;

}
```

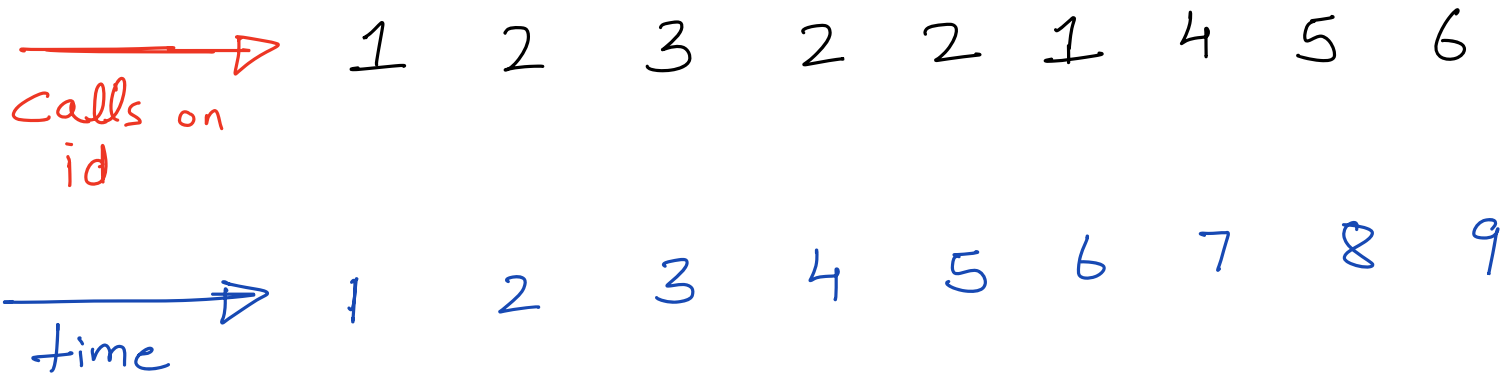
CACHE !

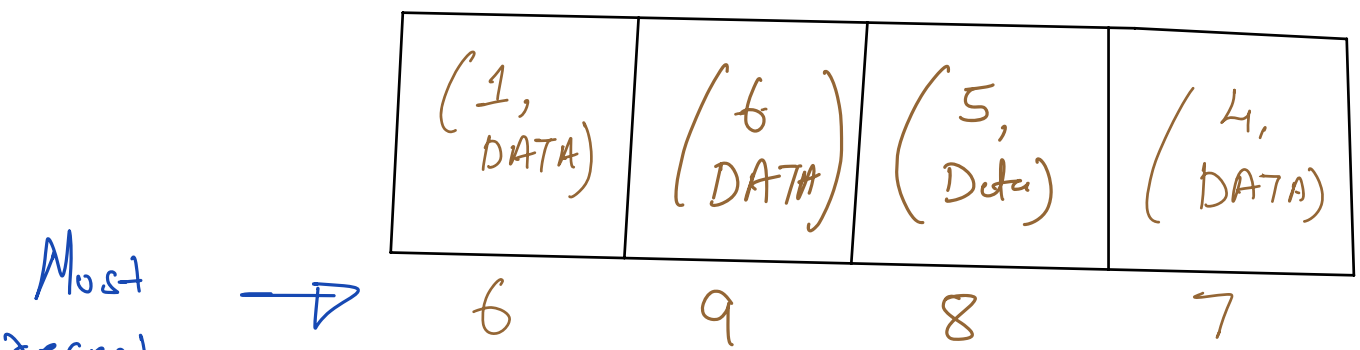
1.5 secs



LRU CACHE !

↳ {Least Recently Used}





Implement on LRU Cache

1) If the id is present in cache return Data

2) If the id is not present, insert in cache and return data

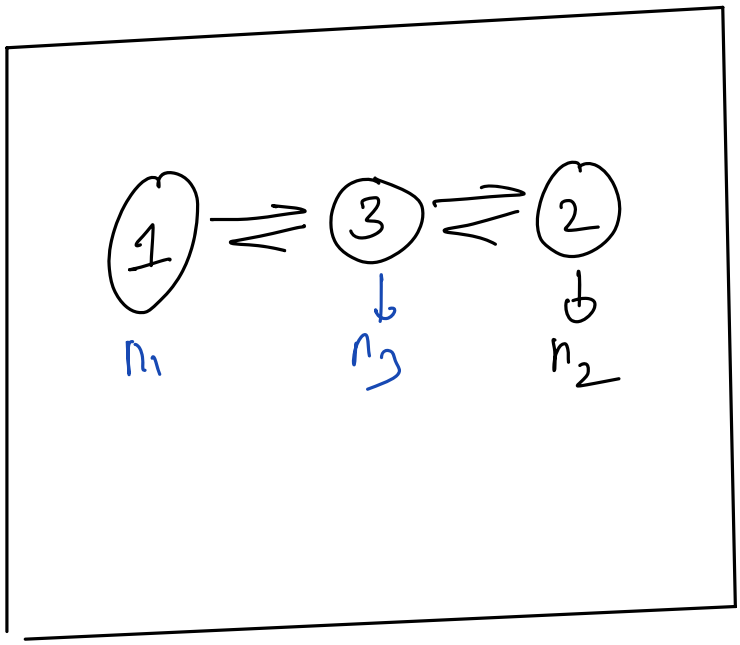
3) If the cache is full, remove least recently used id.

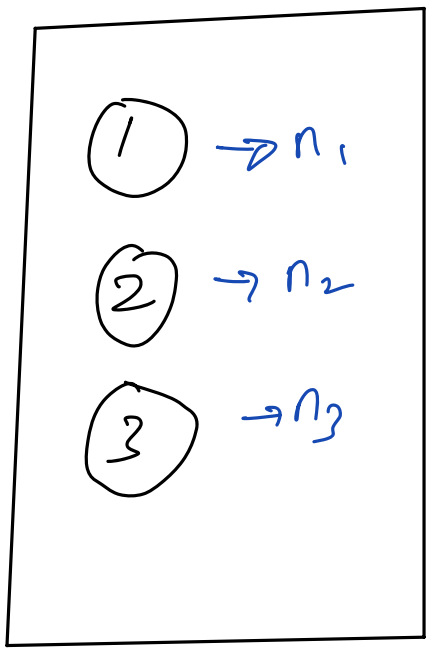
```
void set(int key, int val) {

}

int get(int key) {

}
```

Set Set Set Set

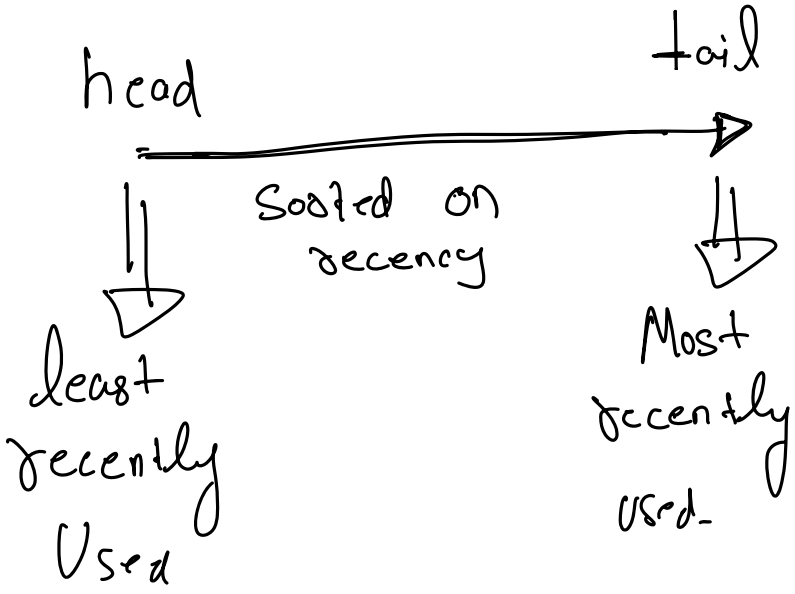
1 2 3 2 2 1 4 5 6

(2, 4)

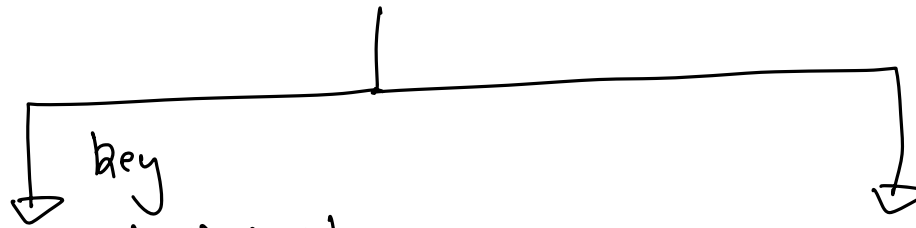




Map <int, Node>



Insert (key, value)



Key not present

1) insert key in map
2) insert node at end.

Key is present

1) Get reference from hashmap
2) Delete node from DLL and insert at end.

```
class Node {
    int key;
    int val;
    Node prev;
    Node next;
}
```