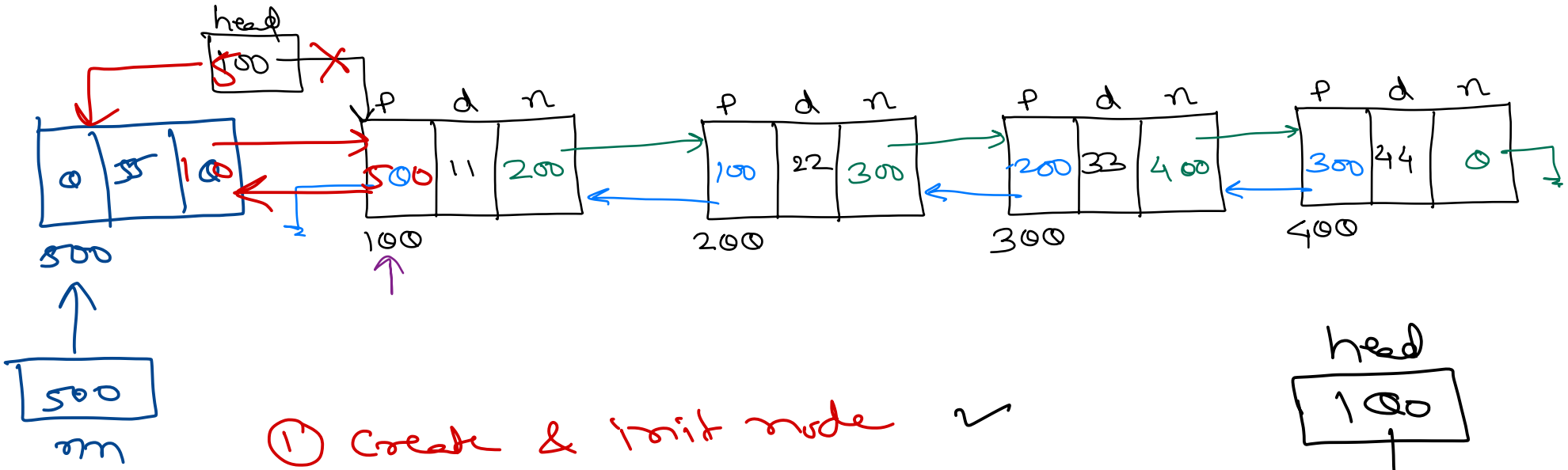

# Data Structure & Algorithms

*Sunbeam Infotech*

*Nilesh Ghule*

# Doubly Linear Linked List

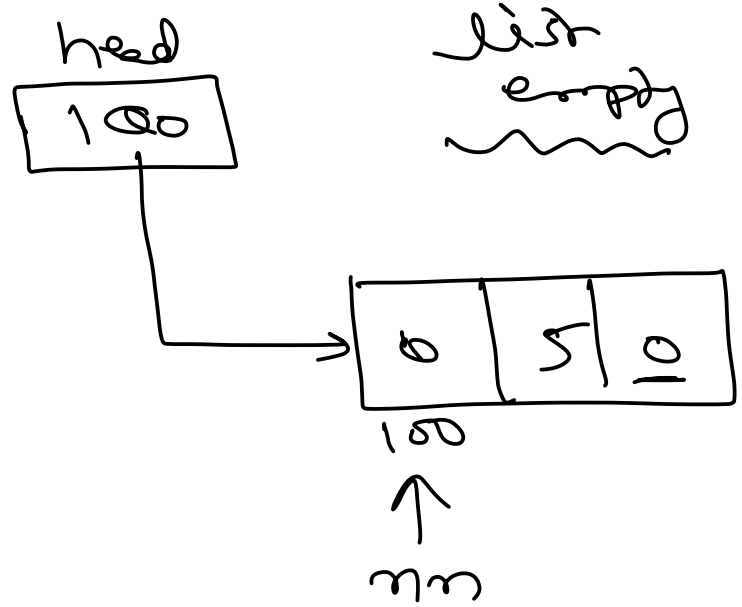



① Create & init node ✓

② nn . next = head; ✓

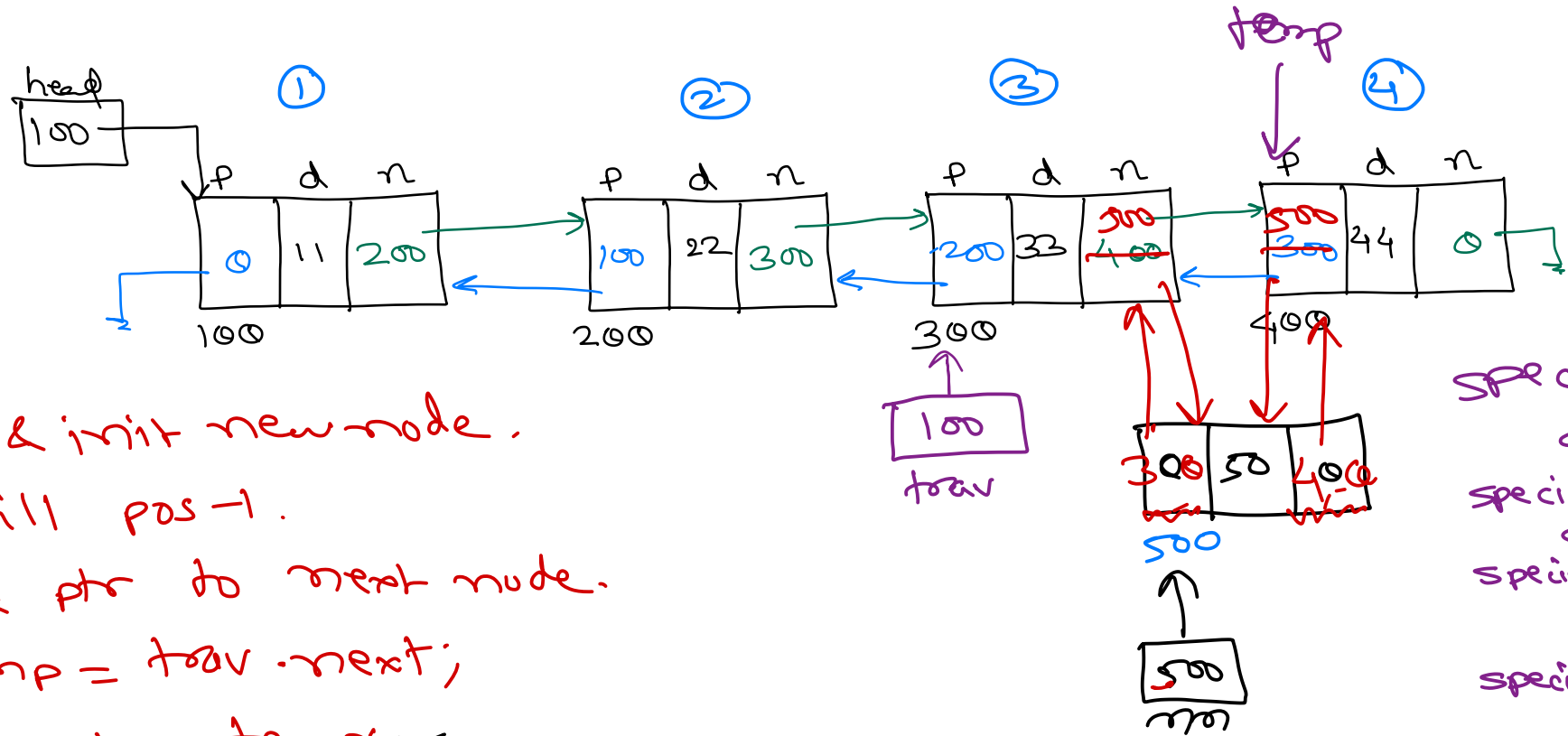
✗ ③ head . prev = nn;

④ head = nn; ✓




special: list empty

# Doubly Linear Linked List




① Create & init new node.

② trav till pos-1.

③ get a ptr to next node.

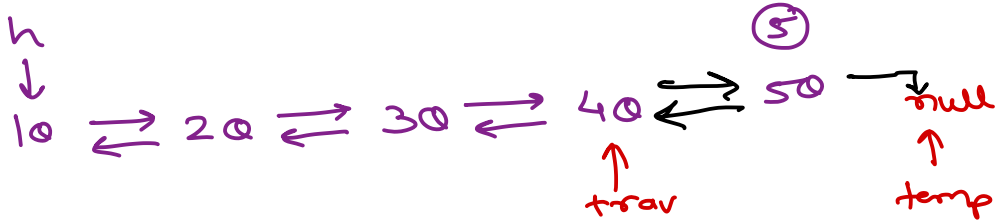
temp = trav.next;

④ nn.next = temp; ✓
nn.prev = trav; ✓

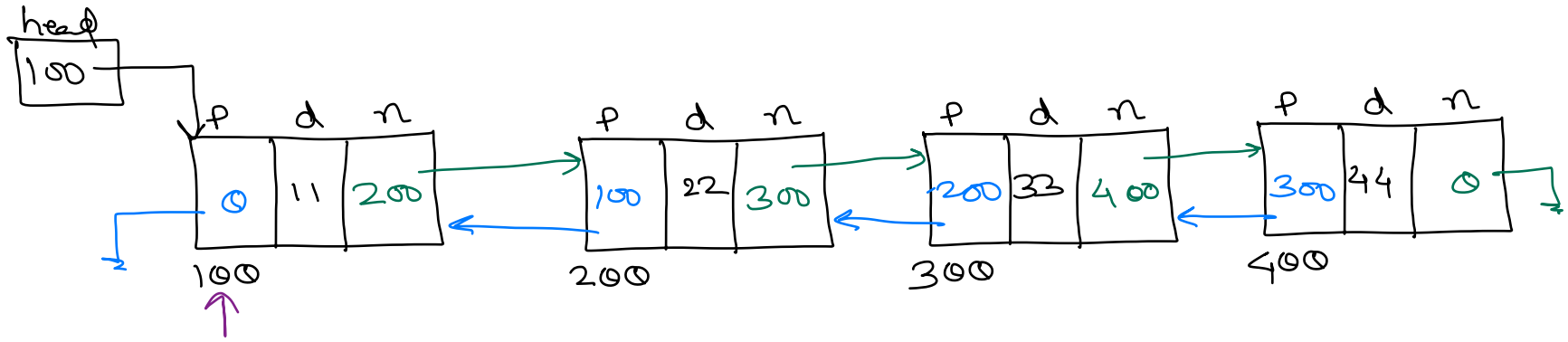
⑤ trav.next = nn; ✓

⑥ temp.prev = nn; ?

Special 1: list empty
add first.
Special 2: pos == 1
add first
Special 3: pos < 1
add first
Special 4: pos > max
skip line
temp.prev = nn

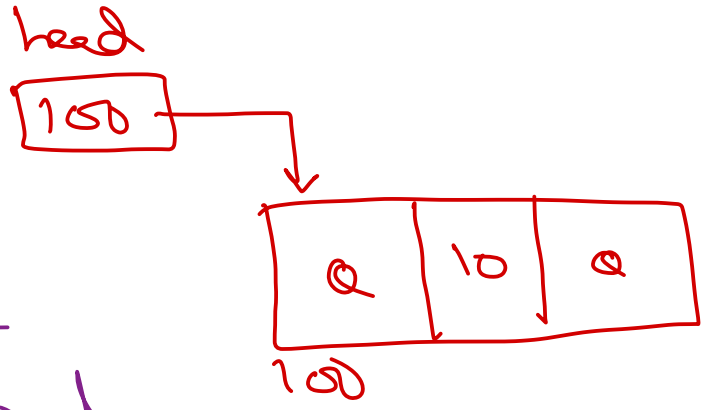


# Doubly Linear Linked List - del first

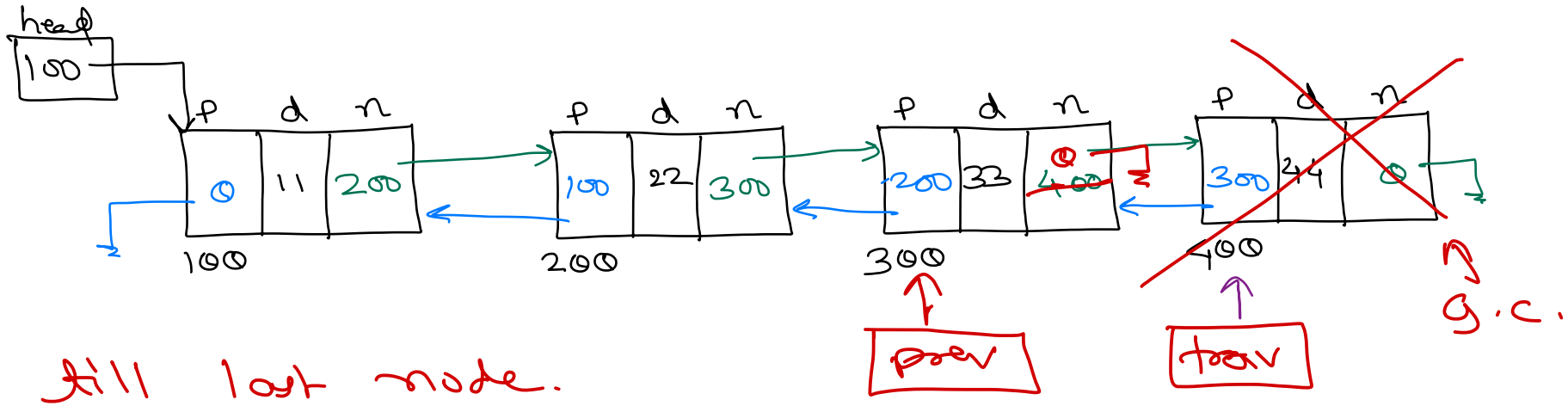



```
head = head.next;
if (head != null)
    head.prev = null;
```

← to handle special case - single node.



# Doubly Linear Linked List - del last



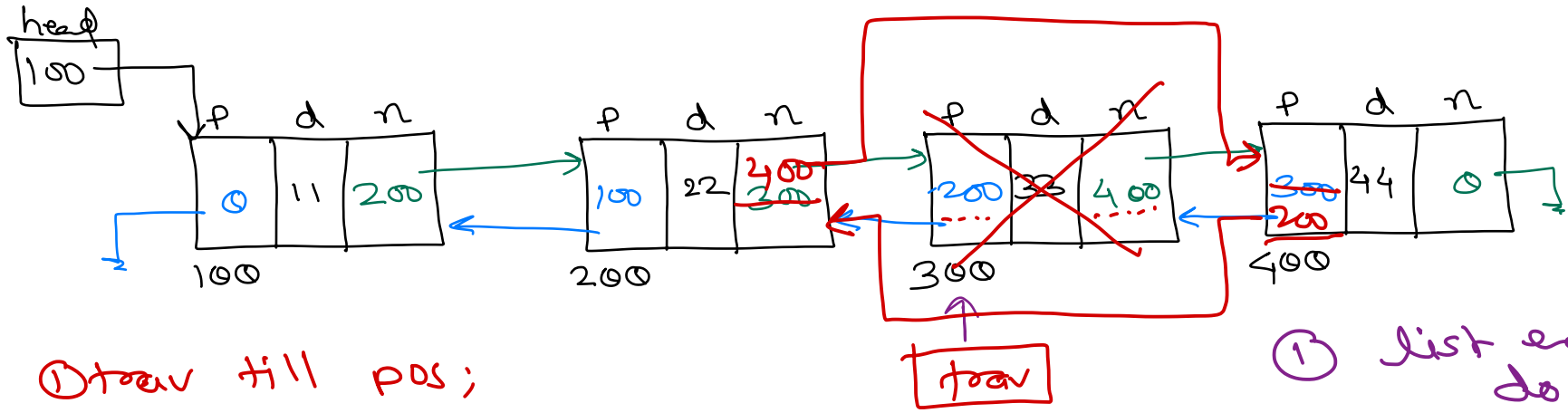

① trav till last node.

② get addr of prev node
prev = trav.prev;

③ In prev ' next, keep null.
prev.next = null;

④ trav node will be gc

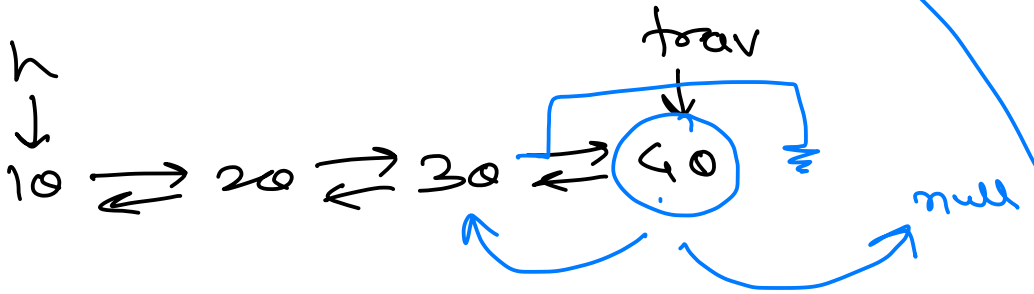
# Doubly Linear Linked List – del at pos




① trav till pos;

② trav.prev.next = trav.next;
trav.next.prev = trav.prev;

③ trav will be gc.




① list empty: do nothing

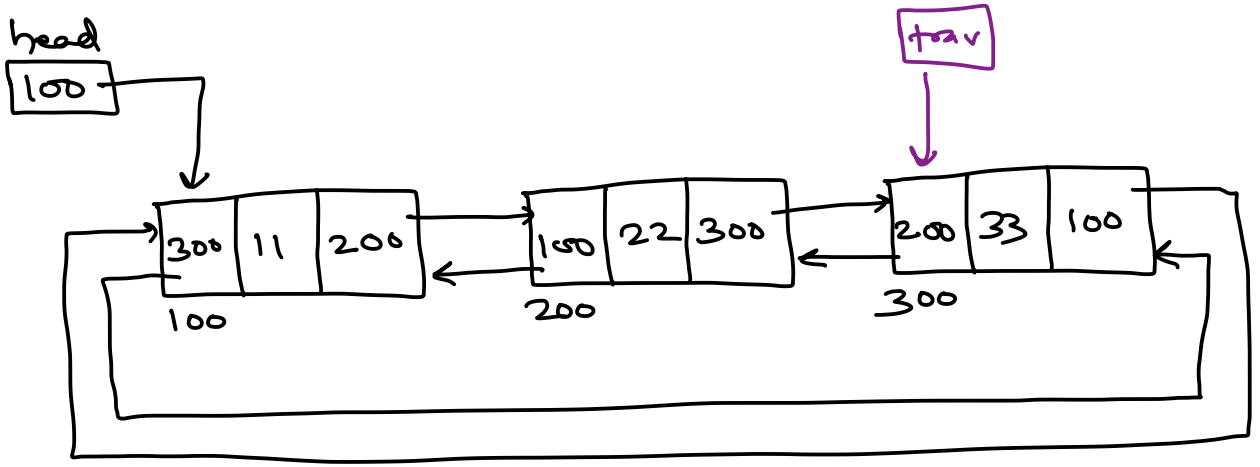
② pos == 1: del first.

③ pos < 1: do nothing

④ pos > max do nothing

⑤ pos == max
↳ skip this

# Doubly Circular Linked List – linux kernel (jobqueue, ready queue, inode list)




## display fwd → O(n)

Tan

```
trav = head;
do {
    pf(trav.data);
    trav = trav.next;
}while(trav != head);
```

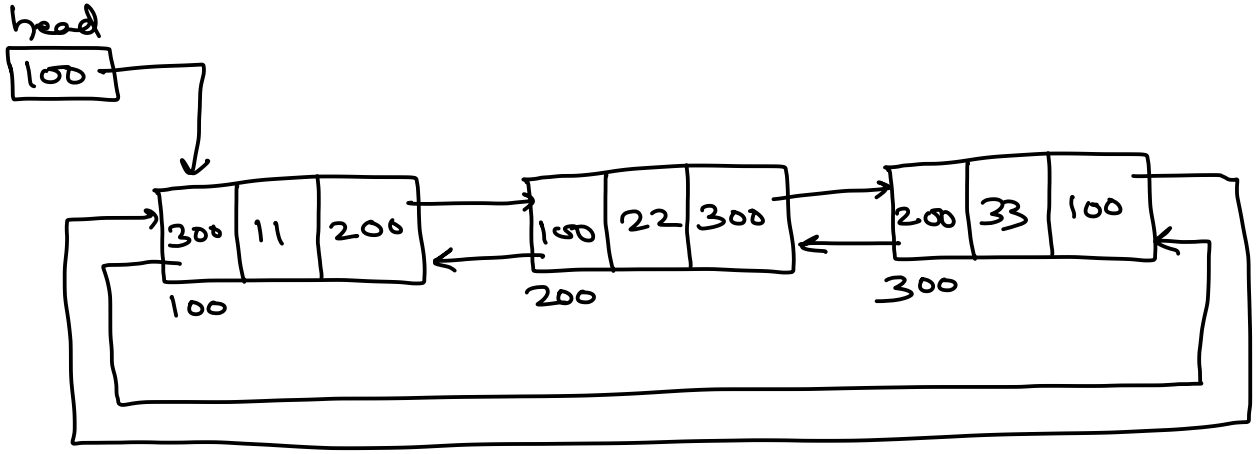
## display reverse

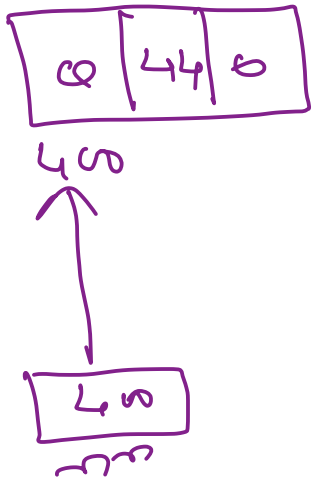
Tan

O(n)

```
// jump to last node.
trav = head.prev;
do {
    pf(trav.data);
    trav = trav.prev;
}while(trav != head.prev);
```

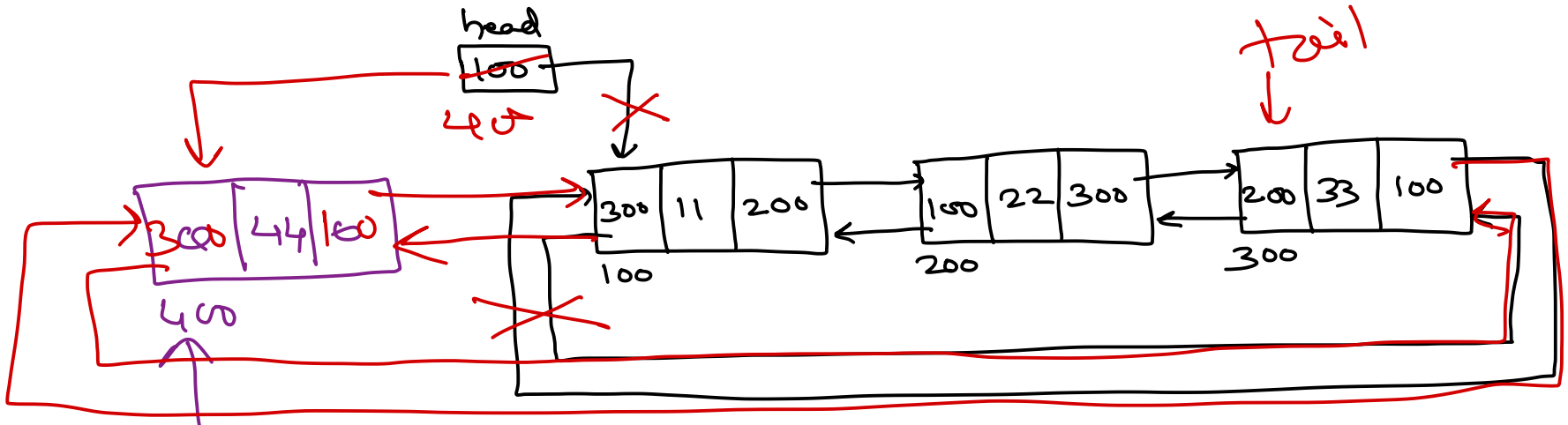
# Doubly Circular Linked List — add last → O(1)




1. create & init new node
2. get ptr to last node
   `tail = head.prev;`
3. add node betn head & tail
   `nn.next = head;`
   `nn.prev = tail;`
   `tail.next = nn;`
   `head.prev = nn;`

# Doubly Circular Linked List — add first → O(1)



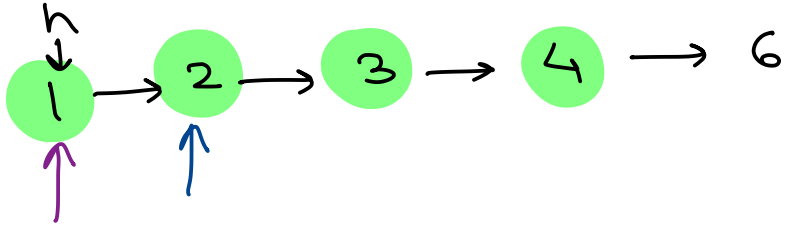

1. create & init new node
2. get ptr to last node
   `tail = head.prev;`
3. add node betn head & tail
   `nn.next = head;`
   `nn.prev = tail;`
   `tail.next = nn;`
   `head.prev = nn;`

4. `head = nn;`

# Linked List – Competitive programming

- Sort the singly linked list.

seq traversing in one direction.




selection

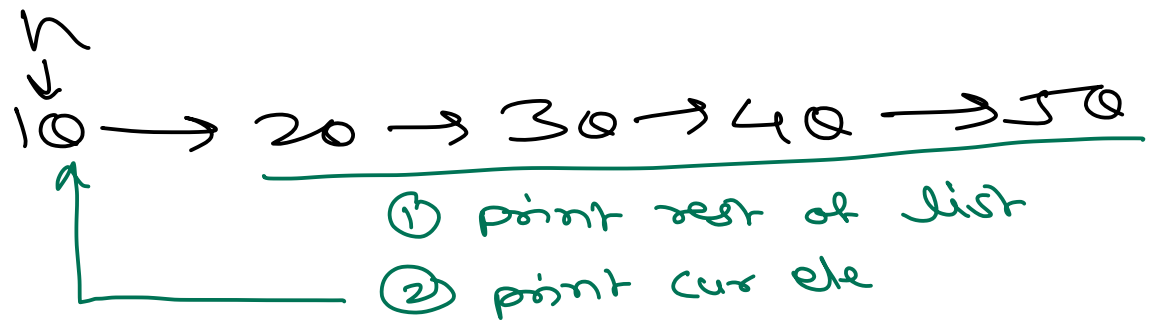
bubble

```
Node i, j;
for(i=head; i!=null; i=i.next){
    for(j=i.next; j!=null; j=j.next){
        if(i.data > j.data){
            temp=i.data;
            i.data=j.data;
            j.data=temp;
        }
    }
}
```

# Linked List – Competitive programming

- ~~Reverse~~ singly linked list. in reverse.

display




base cond: list empty.

```
void rev_display (Node cur) {
    if(cur == null)
        return;
    rev_display (cur.next);
    printf(cur.data);
}

l.rev_display (head);
```

① using stack:
- traverse list, push all ele in stack.
- while stack not empty, pop & print each ele.

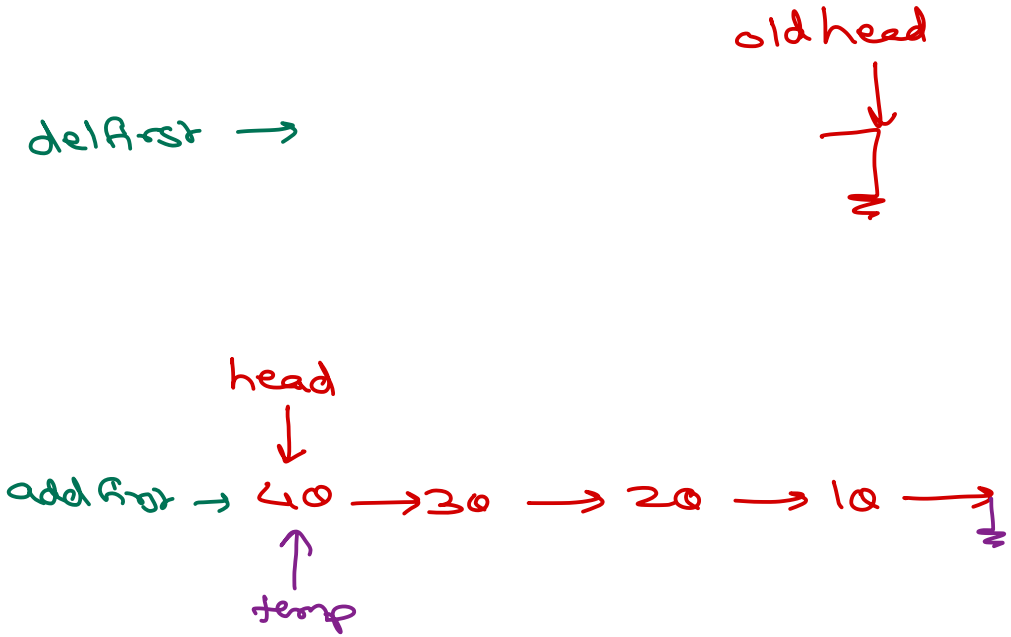
② using counter
- traverse list to count element.
- while (count > 0) {
  traverse upto count.
  print ele.
  count--;

$O(n^3)$

③ using stack/recursion.
① print rest of list
② print cur ele
base cond: list empty.

# Linked List – Competitive programming

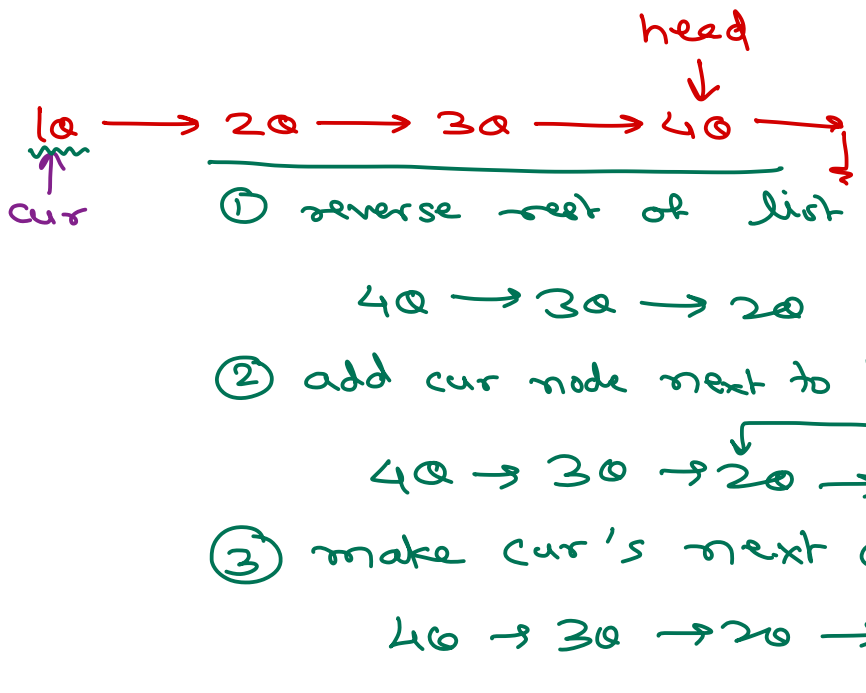
- Reverse singly linked list.




```
oldhead = head;
head = null;
while (oldhead != null)
{
    //delFirst on old list
    temp = oldhead;
    oldhead = oldhead.next;
    //addFirst on new list
    temp.next = head;
    head = temp;
}
```

# Linked List – Competitive programming

- Reverse singly linked list using recursion.

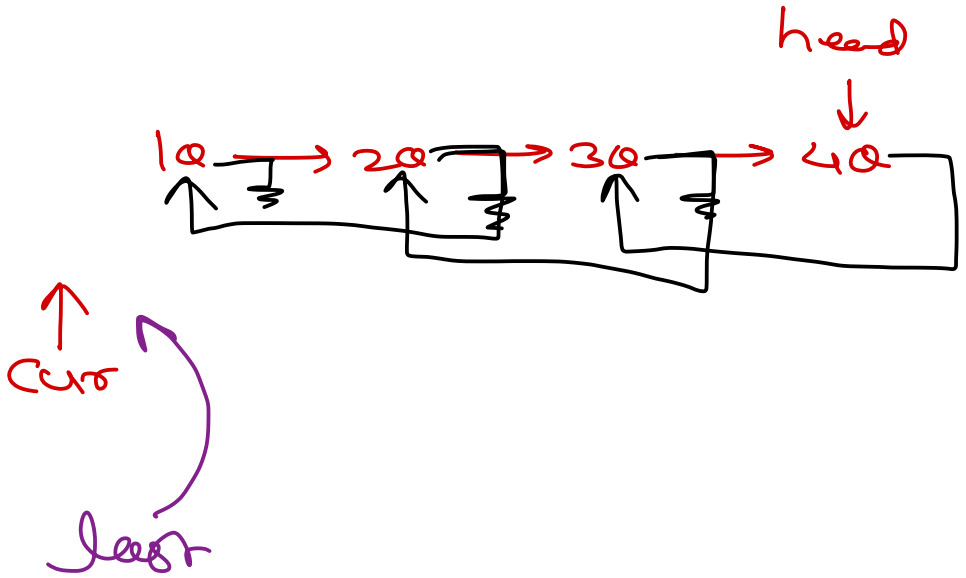


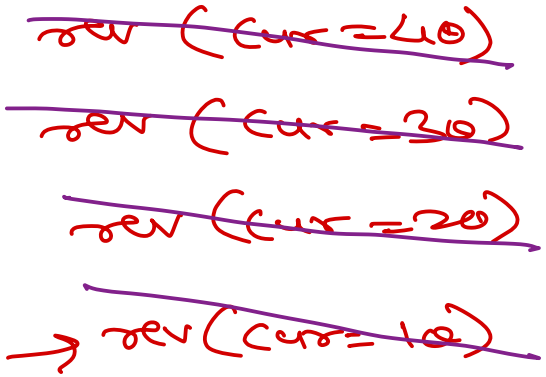

base cond: if last node, make it as head.

```
Node recReverse(Node cur) {
    if(cur.next==null) {
        head = cur;
        return cur;
    }
    last=recReverse(cur.next);
    last.next = cur;
    cur.next = null;
    return cur;
}
void recReverse() {
    if(!isempty())
        recReverse(head);
}
```

# Linked List – Competitive programming

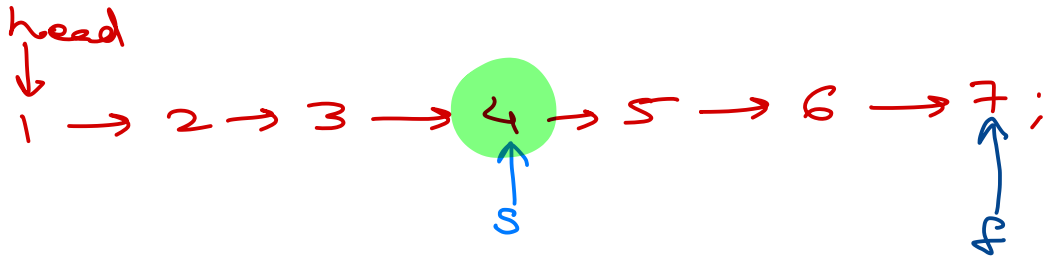
- Reverse singly linked list using recursion.



# Linked List – Competitive programming

- Find middle of singly linear linked list.




```
slow = fast = head;
while (fast != null && fast.next != null)
{
    slow = slow.next;
    fast = fast.next.next;
}
pf(slow.mid);
```

even num

odd num

use a prev behind slow ptr, so that we can implement delete/insert at middle.

① maintain count.
traverse upto count/2.

- need to maintain count.

② traverse list first time to count num of element.

traverse list second time up to count/2.

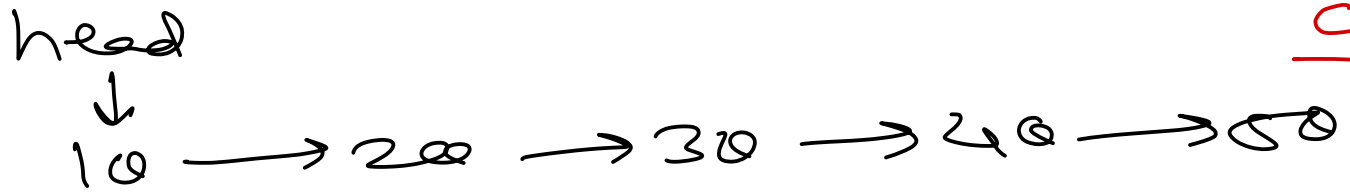
$T \propto n + \frac{n}{2}$

$T \propto 1.5\ n$

$O(n)$

# Linked List – Competitive programming

- Find middle of singly linear linked list using single pointer. & traversing only once.

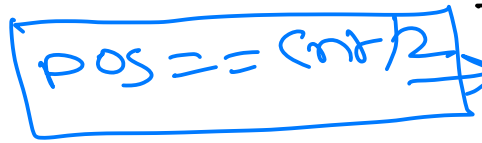



count = 6

```
int count = 0;

int mid (Node cur, int pos) {
    if (cur == null) {
        count = pos;
        return 0;
    }
    V = mid (cur.next, pos+1);
    if (pos == count/2)
        return cur.data;
    return V;
}
```

mid (head, 1);




# Linked List – Competitive programming

- Find middle of singly circular linked list.

```
slow = fast = head;
do
{
    slow = slow.next;
    fast = fast.next.next;
} while (fast != head && fast.next != head)
```

even num

odd num

# Linked List – Competitive programming

1 → 2 → 3 → 2 → 1

- Check if linked list is palindrome.

singly

① using stack.
- push all ele on stack
- pop & compare again.

T ∝ 2n
T = O(n)
S = O(n) aux

② using recursion

T = O(n)
S = O(n) aux
function stack

③ using counter approach
- similar to rev display (using cnt).

T = O(n²)  S = O(1)

④

T = O(n)
S = O(1) const

1 → 2 → 3 → 3 → 2 → 1

ⓐ split linked list in two half.
- find middle (head of 2nd half).

1st: 1 → 2 → 3
2nd: 3 → 2 → 1

ⓑ reverse 2nd half list

1st: 1 → 2 → 3
2nd: 1 → 2 → 3

ⓒ compare two list ele by ele.
if same, palindrome.
if not, not palindrome

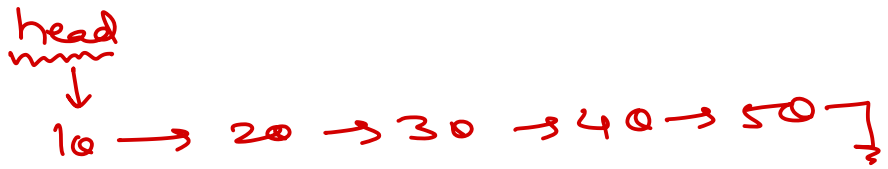
④ reverse 2nd half list

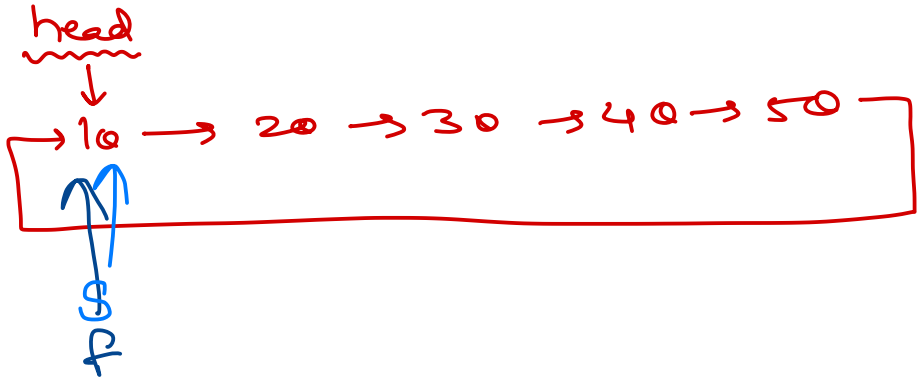
1st: 1 → 2 → 3
2nd: 3 → 2 → 1

⑤ append 2nd list to first list.

# Linked List – Competitive programming

- Check if linked list contains a loop.




```
slow = head;
fast = head;
do {
    if(fast == null ||
            fast.next == null)
        return false; → no loop.
    slow = slow.next;
    fast = fast.next.next;
} while(slow != fast)
return true; → loop.
```

# Linked List – Competitive programming

Homework: remove duplicate eles in sorted singly list.

- Remove duplicated elements from the unsorted list.

① using set/map.

- push all eles of list in set (LinkedHashSet) one by one.
- rebuild list by getting eles from set one by one.

T = O(n logn) + O(n)  S = O(n) aux → set

```
for(ele : list) ← O(n)
{
    set.add(ele); ← O(logn)
}
for(ele : set) ← O(n)
    l.addLast(ele); ← O(1)
```

② like selection sort

select an ele, compare with all ele after that; if same, delete it.

to select all eles one by one.

T = O(n²)  S = O(1) aux


*Thank you!*

Nilesh Ghule <nilesh@sunbeaminfo.com>