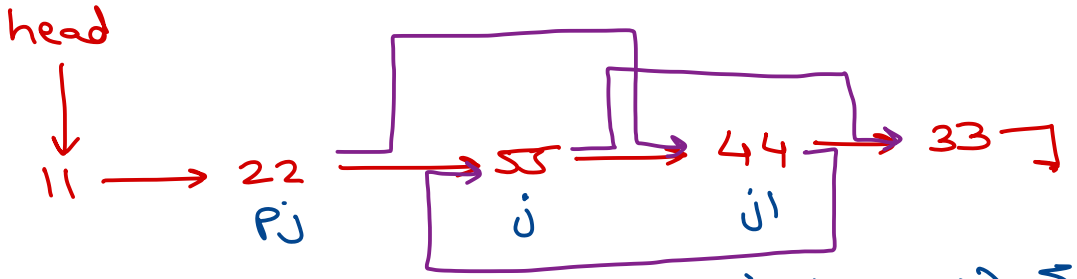
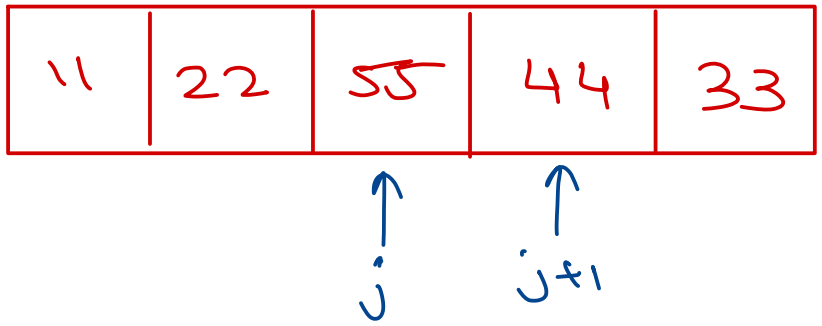
Data Structure & Algorithms

Nilesh Ghule

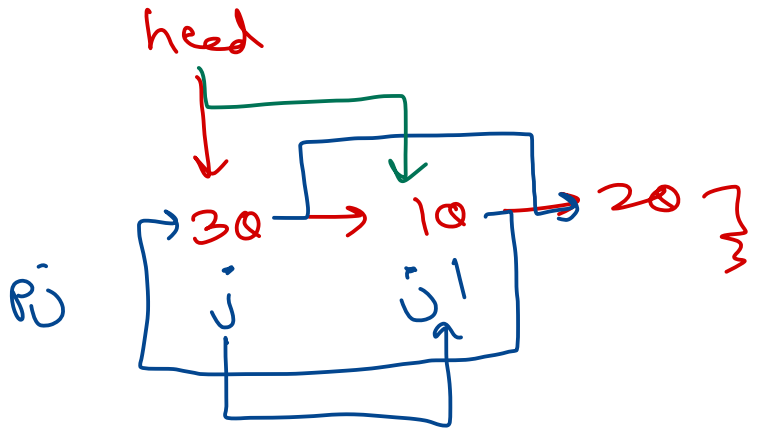
Linked List – Competitive programming

- Sort the singly linked list.



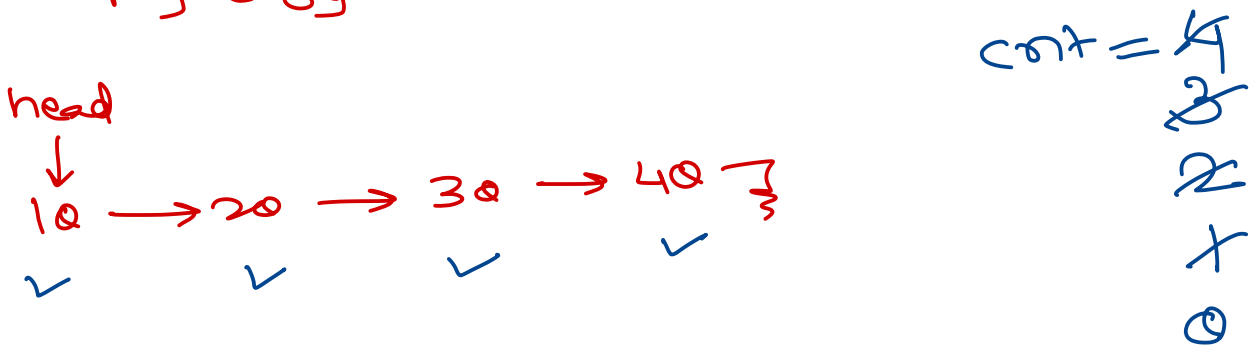



```
for(i=head; i.next!=null; i=i.next) {
    pj=null;
    for(j=head; j.next!=null; j=j.next) {
        Node j1=j.next;
        if(j.data > j1.data) {
            if(pj==null) head=j1; else pj.next=j1;
            j.next=j1.next;
            j1.next=j;
            j=j1;
        }
        pj=j;
    }
}
```



Linked List – Competitive programming

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



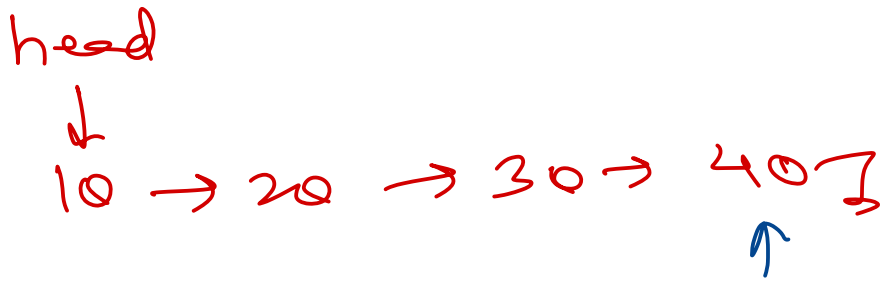
$O(n^2)$

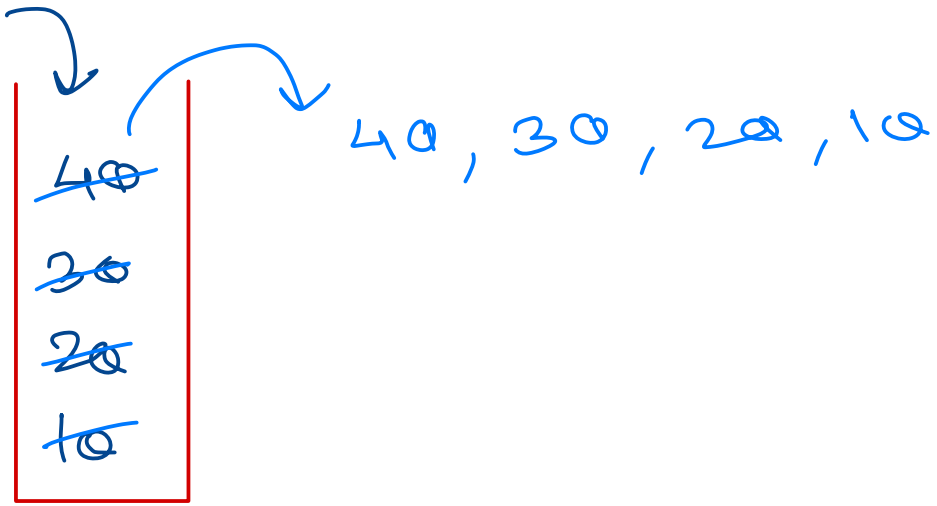
1. count num of nodes in list → cnt var.
2. traverse list till node num "cnt" & display it.
3. decrement count (cnt--).
4. repeat step 2 & 3, until cnt became zero.

Linked List – Competitive programming

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

time : O(n)
aux space : O(n)



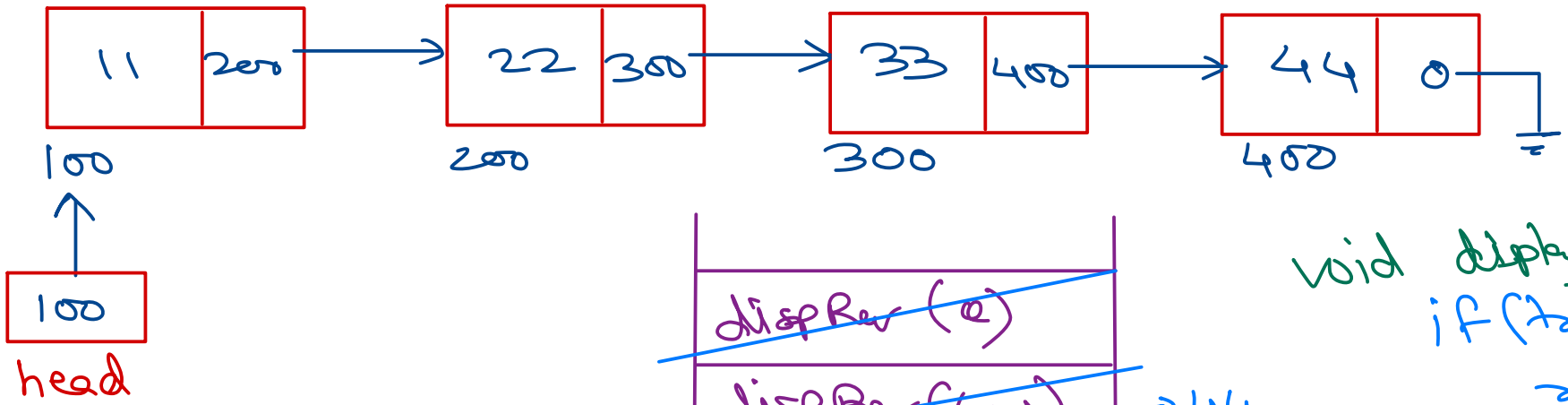


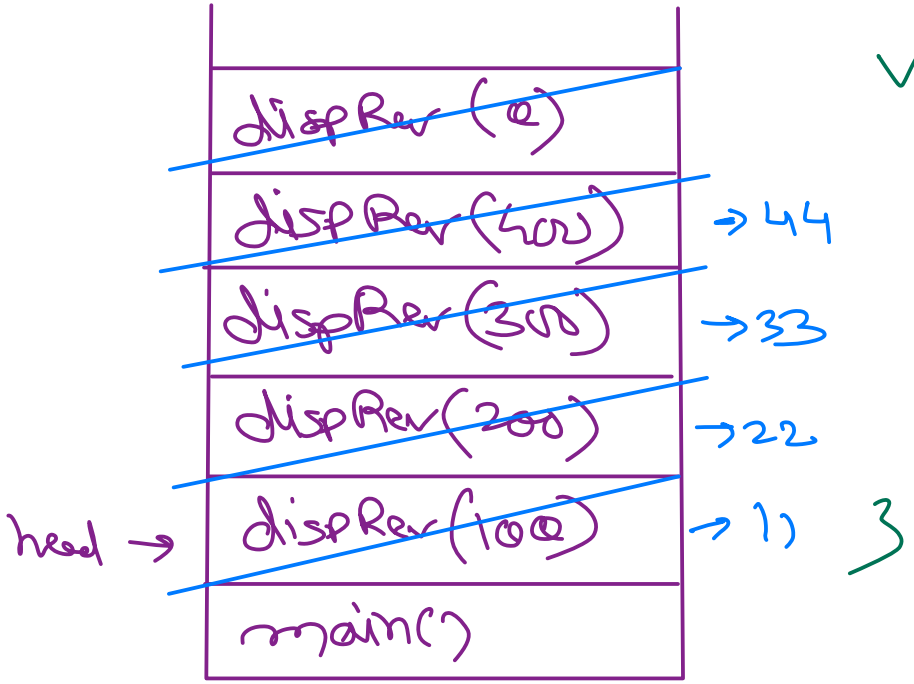

```
Stack<Integer> s = new Stack<>();
trav = head;
while(trav != null) {
    s.push(trav.data);
    trav = trav.next;
}

while(!s.isEmpty()) {
    val = s.pop();
    print(val);
}
```

Linked List – Competitive programming

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

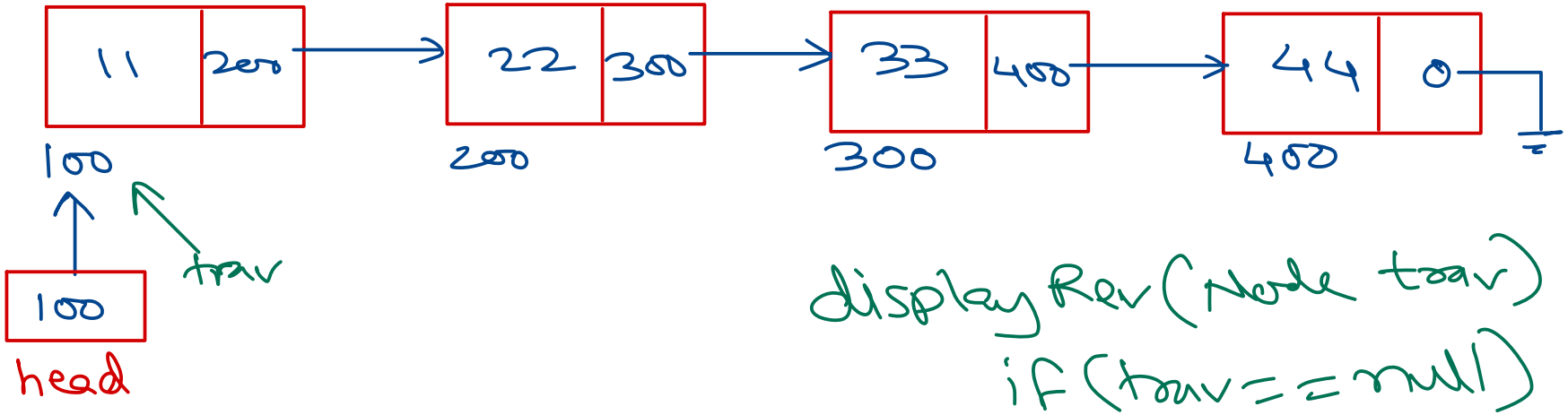





```
void displayRev(Node trav) {
    if(trav==null)
        return;
    displayRev(trav.next);
    print(trav.data);
}
```

Linked List – Competitive programming

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

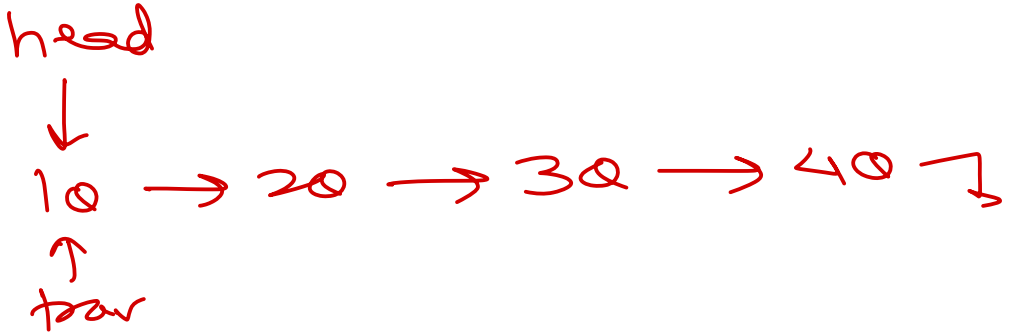


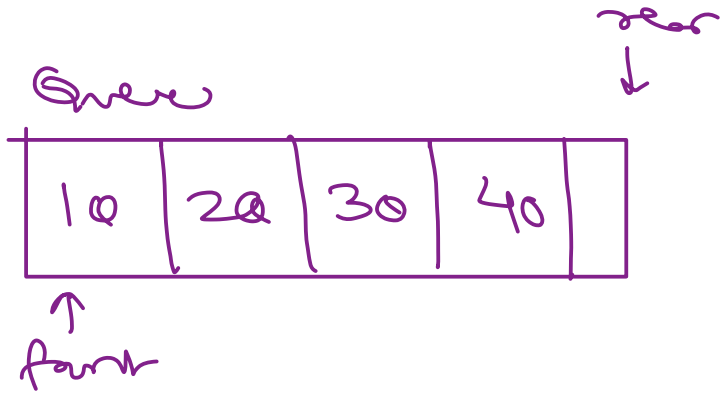
recursion:
1. explain process in terms of itself.
2. end condition

```
displayRev(Node trav) {
    if(trav == null)
        return;
    displayRev(trav.next);
    print(trav.data);
}
```

Linked List – Competitive programming

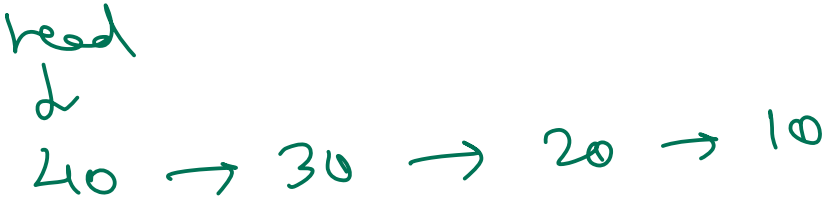
- Reverse singly linked list. – using queue





time: O(n)
aux space: O(n)

① one by one delete first node & push on queue
② pop nodes from queue. one by one and add first in list



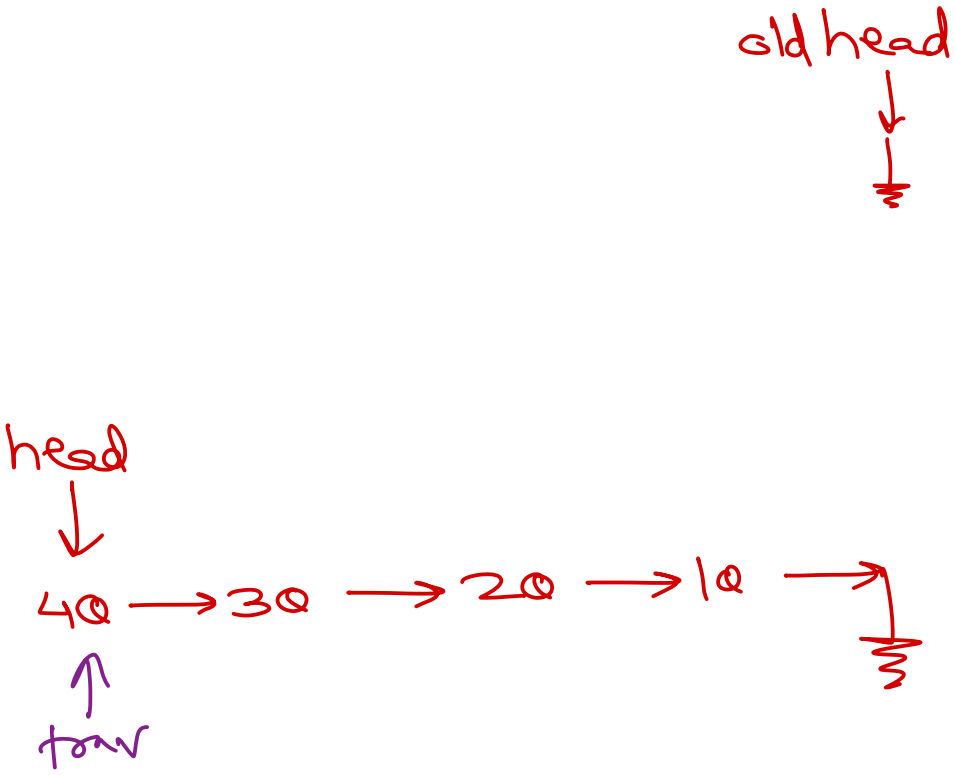

```
Queue<Integer> q = new LinkedList<>();
while(head != null) {
    val = delFirst();
    q.offer(val);
}
while(!q.isEmpty()) {
    val = q.poll();
    addFirst(val);
}
```

Linked List – Competitive programming

time : O(n) aux space : O(1)

two additional pointers oldhead, trav.

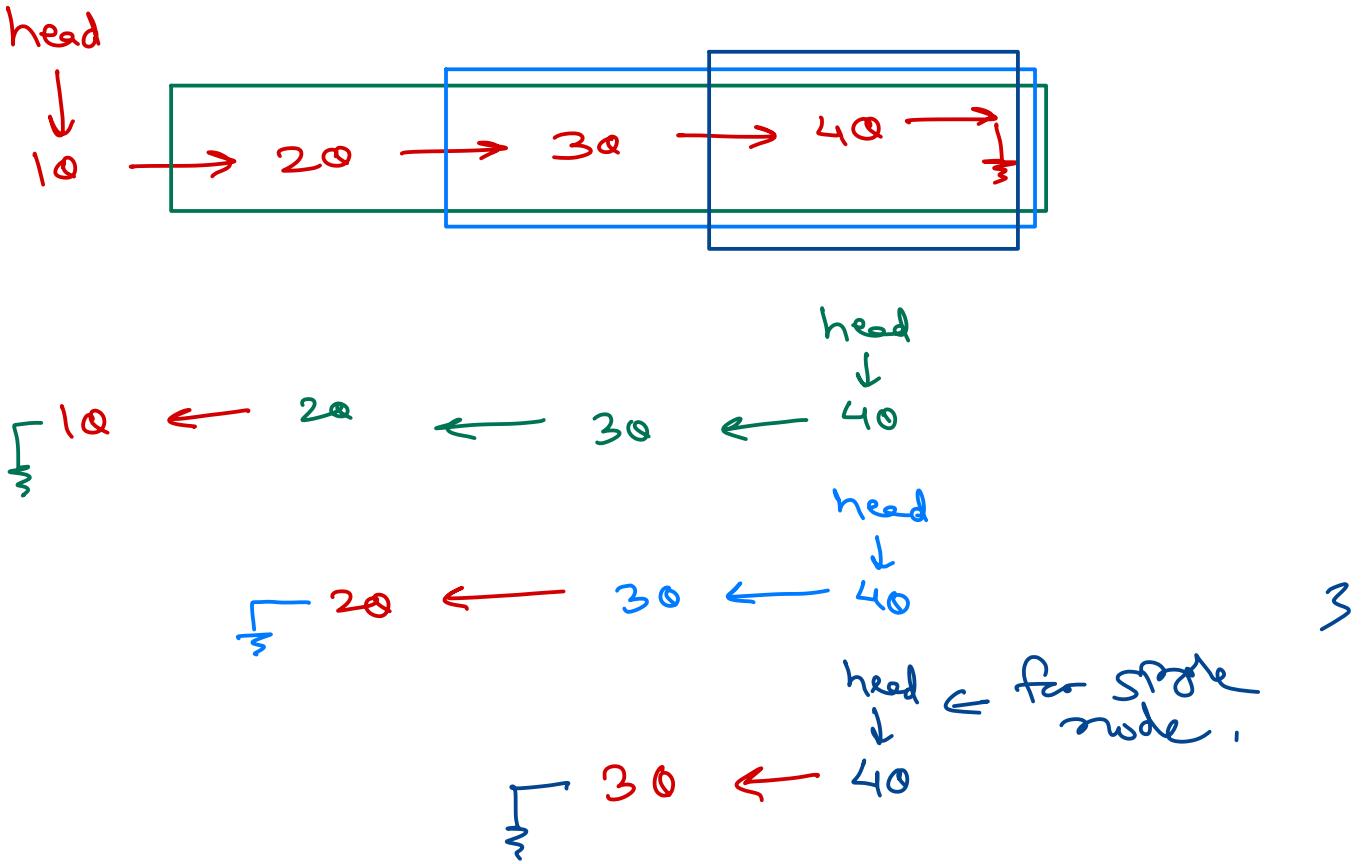
- Reverse singly linked list.




```
oldhead = head;
head = null;
while(oldhead != null)
{
    //del first from old list
    trav = oldhead;
    oldhead = oldhead.next;
    //add that node to first of list
    trav.next = head;
    head = trav;
}
```

Linked List – Competitive programming

- Reverse singly linked list using recursion.

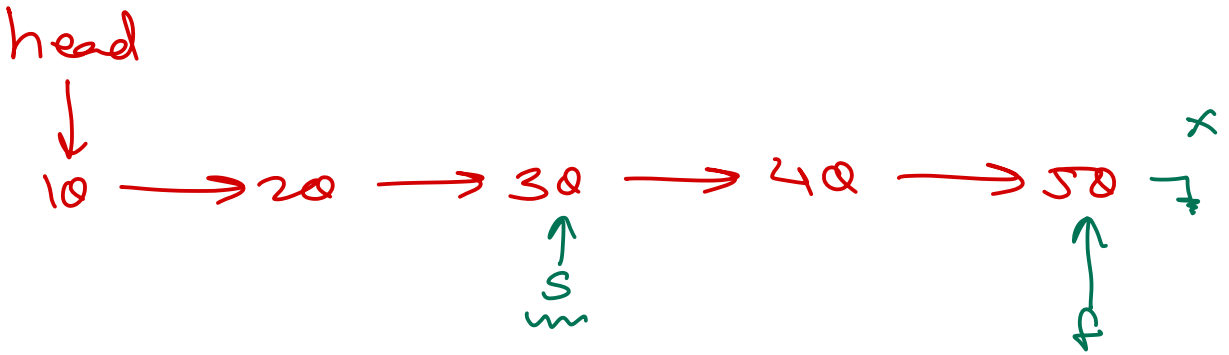


for single node.

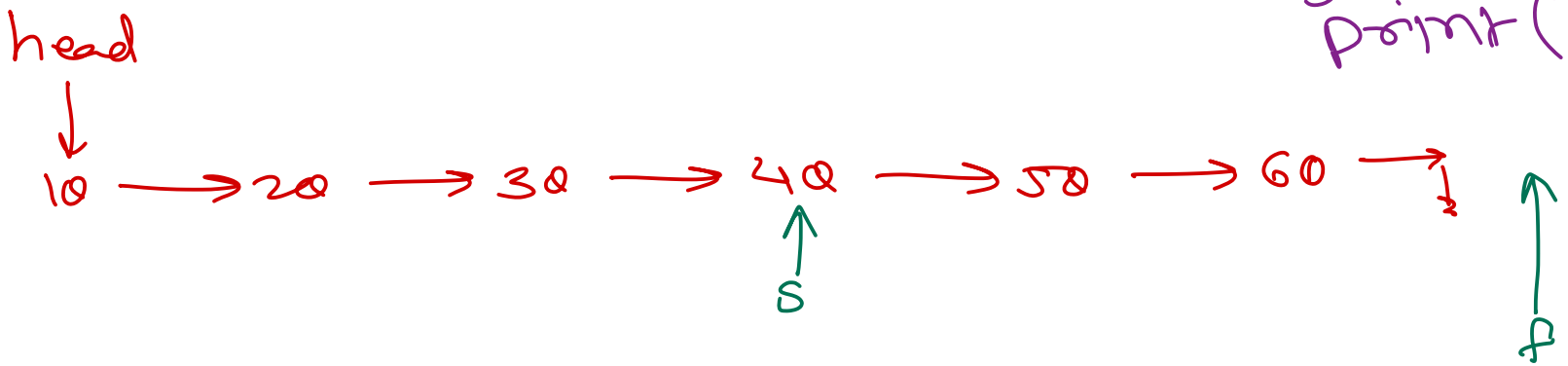
```
Node reverse(Node trav) {
    if(trav.next == null) {
        head = trav;
        return trav;
    }
    last = reverse(trav.next);
    last.next = trav;
    trav.next = null;
    return trav;
}
```

Linked List – Competitive programming

- Find middle of singly linear linked list.




```
f = head;
s = head;
while(f == null || f.next == null){
    s = s.next;
    f = f.next.next;
}
print("mid" + s.data);
```



Linked List – Competitive programming – assignment

- Find middle of singly linear linked list using single pointer.

① traverse the list and count the number of nodes → cnt;

② traverse till (cnt/2) node pos & print it.

Linked List – Competitive programming – recursion.

- Find middle of singly linear linked list using single pointer.

```
int max;
void printmid(Node trav, int pos) {
    if (trav==null) {
        max = pos;
        return;
    }
    printmid(trav.next, pos+1);
    if (pos == max/2)
        print(trav.data).
}
```

Linked List – Competitive programming

- ~~Find middle of singly circular linked list.~~
  check if list is palindrome.

Soln 2 use recursion.

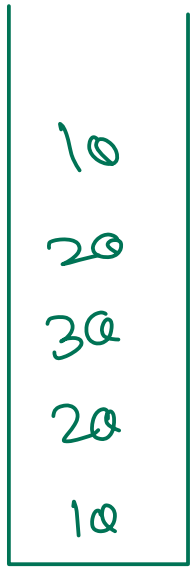
head

Soln 1

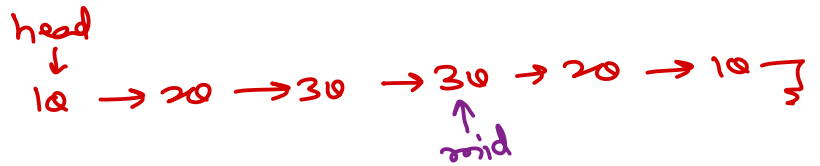
10 → 20 → 30 → 20 → 10

time: O(n)
aux space: O(n)

```
trav=head;
while(trav!=null) {
    s.push(trav.data);
    trav=trav.next;
}
trav = head;
while(trav!=null) {
    val = s.pop();
    if(val != trav.data)
        return false; //not pal.
    trav=trav.next;
}
return true; //pal.
```



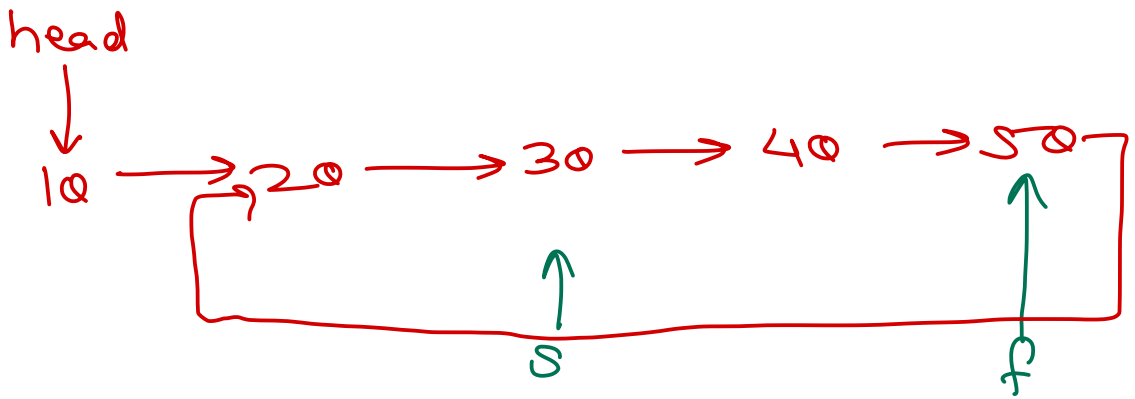
Soln 3



① find the mid (fast & slow logic)
② break list in two list (list & list2)
   list → 10 → 20 → 30
   list2 → 30 → 20 → 10
③ reverse the list 2.
   list → 10 → 20 → 30
   list2 → 10 → 20 → 30
④ compare list & list2; if they are same, list is palindrome – else not.
⑤ reverse list 2 & append to list (to reform it).

Linked List – Competitive programming

- Check if linked list contains a loop.




```
s = head;
f = head;
while(f==null || f.next==null){
    if(f==s)
        return true; // loop
    s = s.next;
    f = f.next.next;
}
return false; // no loop
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>