Sunbeam Institute of Information Technology
Pune and Karad

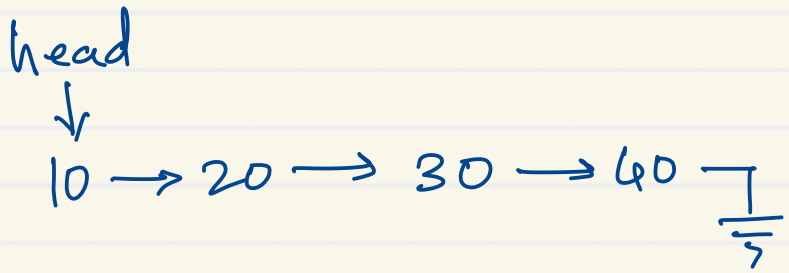
Data structures and Algorithms

Trainer - Devendra Dhande

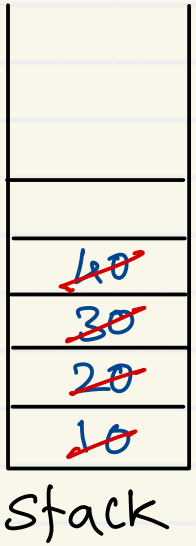
Email – devendra.dhande@sunbeaminfo.com

Linked List - competitive/interview question

- Display singly linked list in reverse order. (using stack)



Algorithm:
1. push all nodes on stack
2. while stack is not empty,
   pop from the stack & display it




```
Stack<Integer> st = new Stack<>();

Node trav = head;
while(trav != null){          <- n itrs
    s.push(trav.data);
    trav = trav.next;
}
while(!st.isEmpty()){         <- n itrs
    val = st.pop();
    sysout(val);
}
```

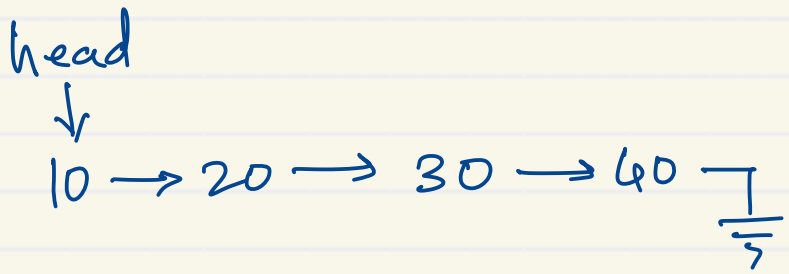
Output: 40 30 20 10

Time: itrs = n + n
$T \propto 2n$
$T(n) = O(n)$

space: (stack space)
$S \propto n$
$S(n) = O(n)$

Linked List - competitive/interview question

- Display singly linked list in reverse order. (using recursion)

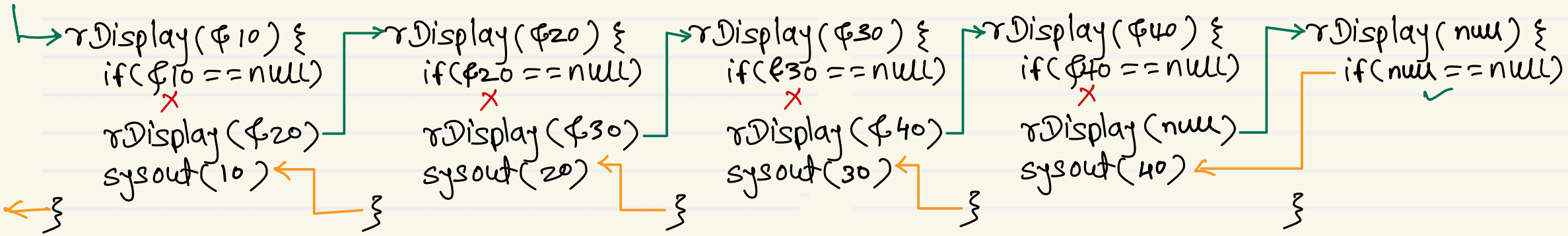


output : 40 30 20 10

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

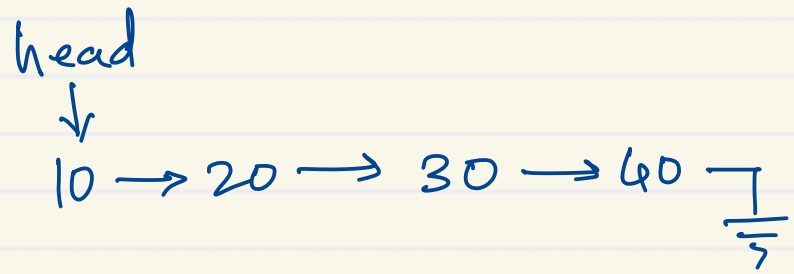
Time:
recursive call = n+1
$T \propto n$
$T(n) = O(n)$

$S \propto n$
$S(n) = O(n)$



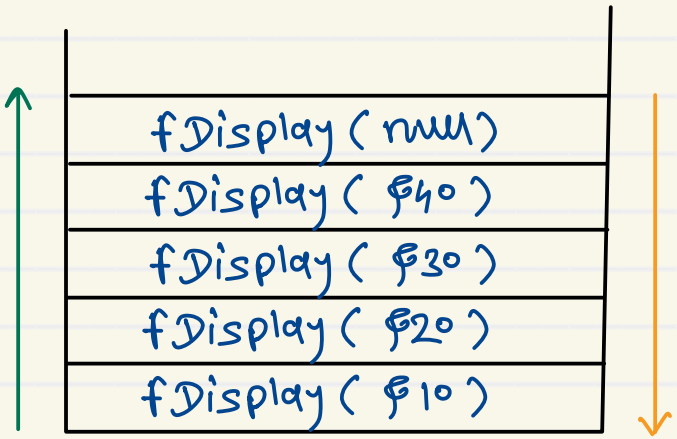
Linked List - competitive/interview question

- Display singly linked list in forward order. (using recursion)




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

Output : 10 20 30 40



Time:
recursive call = n+1
$T \propto n$
$T(n) = O(n)$

$S \propto n$
$S(n) = O(n)$

Recursion

Direct
(function itself calls function)

e.g.
```
fun( ) {
    fun( );
}
```

Tail
(Recursive call is last statement of recursive function)

Non-tail (head)
(Recursive call is not last statement of recursive function)
- after completion of recursive call, some processing is done.

Indirect
(functions are called indirectly in cyclic way)

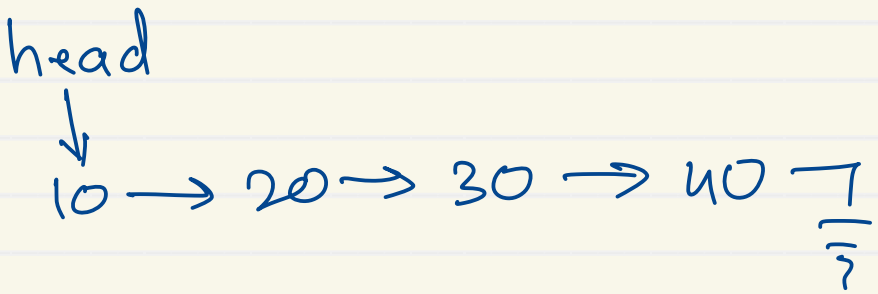
e.g.
```
fun1( ) {
    fun2( );
}

fun2( ) {
    fun1( );
}
```

Linear :
only one recursive call is done in function.
T(n) → linear

Multiple :
more than one recursive calls are done in functions
T(n) → exponential

Linked List - competitive/interview question

- Display singly linked list in reverse order. (using loops)



Count no. of node (using loop) = n

```
while (n >= 1) {
    find nth node of linked list (loop)
    display nth node
    n--;
}
```

space : no extra space
- space is constant

$S(n) = O(1)$

Time:

itrs = n + n-1 + n-2 + ... + 1

itrs = $n + (n-1) + (n-2) + \ldots + 1$

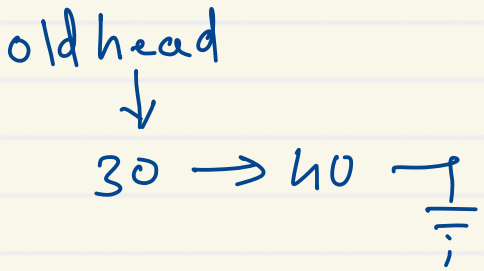
$= \frac{n(n+1)}{2}$

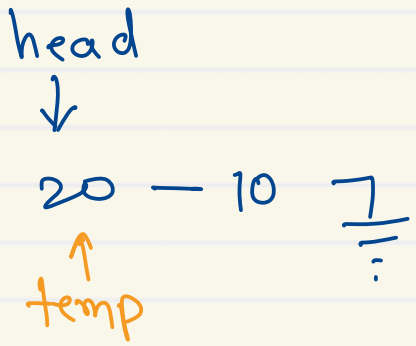
$T \propto \frac{1}{2}(n^2 + n)$

$T(n) = O(n^2)$

Linked List - competitive/interview question

- reverse singly linked list





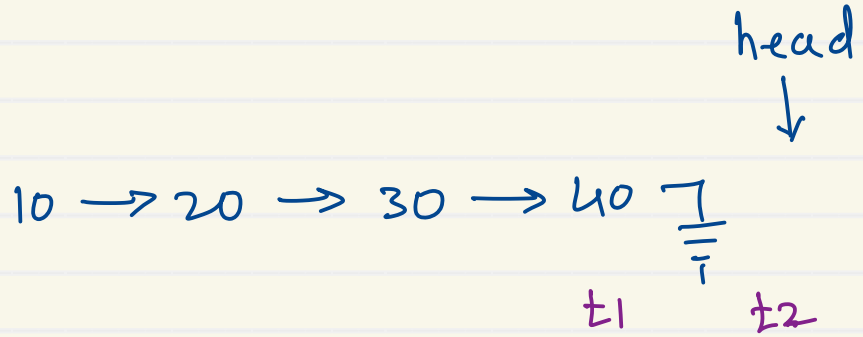
$T(n) = O(n)$

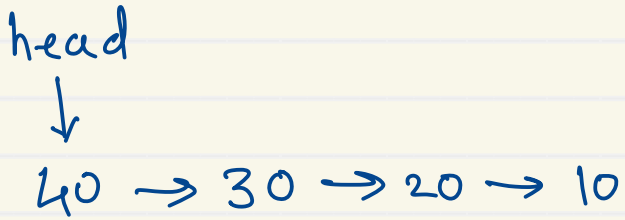
$S(n) = O(1)$

```
//consider cur list as old list
Node oldHead = head;
//consider new list as empty
head = null;
//repeat while old list is not empty
while(oldHead != null){
    //delete first node from old list
    Node temp = oldHead;
    oldHead = oldHead.next;
    //add first node into new list
    temp.next = head;
    head = temp;
}
```

Linked List - competitive/interview question

- reverse singly linked list



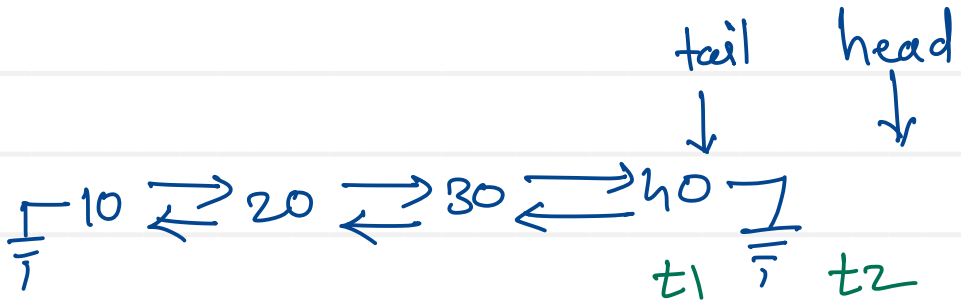


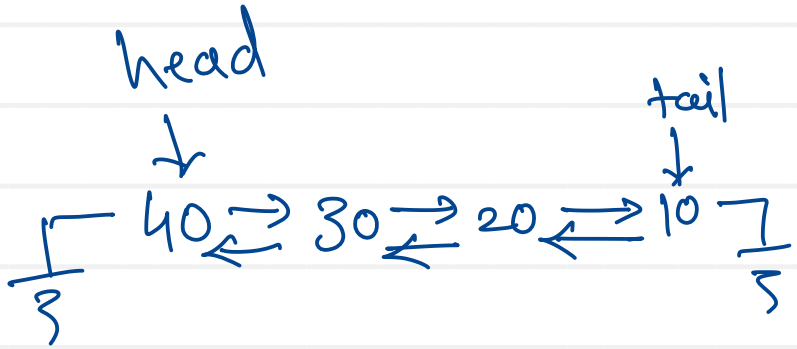

```
t1 = null;
t2 = head;
while(t2 != null){
    head = t2.next;
    t2.next = t1;
    t1 = t2;
    t2 = head;
}
head = t1;
```

$T(n) = O(n)$

$S(n) = O(1)$

Doubly linear linked list - Reverse





$t1.next = t2 \Rightarrow t2.next = t1$

$t2.prev = t1 \Rightarrow t1.prev = t2$

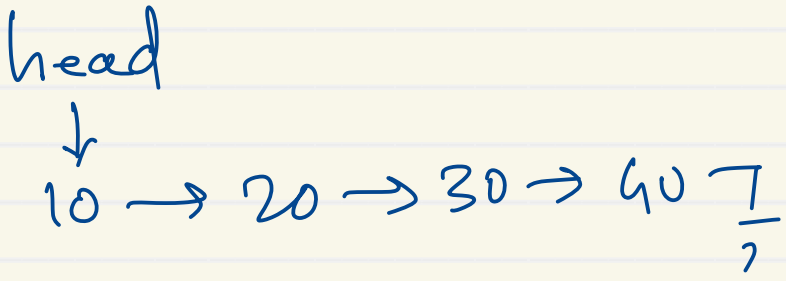
```
Node t1 = head;
Node t2 = head.next;
head.next = null;
tail = head;
while (t2 != null) {
    head = t2.next;
    t2.next = t1;
    t1.prev = t2;
    t1 = t2;
    t2 = head;
}
t1.prev = null;
head = t1;
```

$T(n) = O(n)$

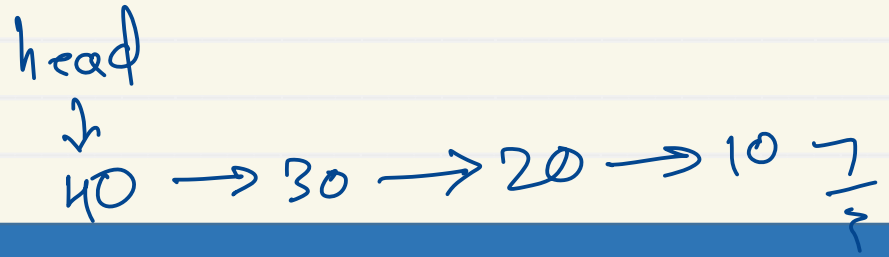
$S(n) = O(1)$

Linked List - competitive/interview question

- reverse singly linked list



	curr	last
recReverse(&10)	&10	&20
recReverse(&20)	&20	&30
recReverse(&30)	&30	&40
recReverse(&40)	&40	



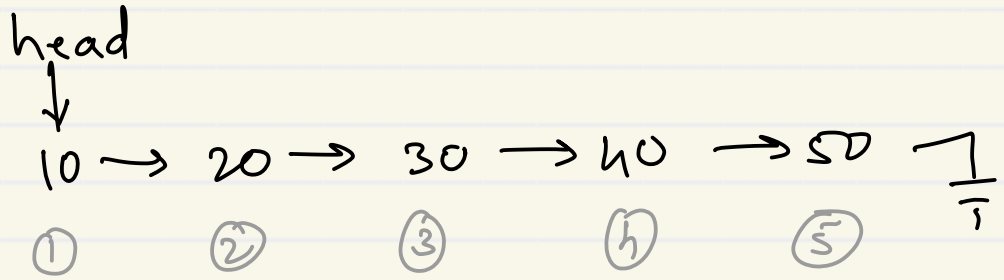

```
Node recReverse(Node curr) {
    if(curr.next == null)
        head = curr;
        return curr;
    }
    Node last = recReverse(curr.next);
    last.next = curr;
    curr.next = null;
    return curr;
}
```

$T(n) = O(n)$

$S(n) = O(n)$

Linked List - competitive/interview question

- Find middle of singly linear linked list




```
Node findMid( Node head) {
    int count= 0;
    Node trav = head;
    while(trav != null) {
        count++;
        trav =trav.next;
    }
    Node trav= head;
    for( i=0; i< count/2 ; i++)
        trav = trav.next;
    return trav;
}
```

count = 5

trav	i	i<2
&10	0	T
&20	1	T
&30	2	F

to count : itrs = n

to find mid : itrs = n/2

itrs = 3/2 n

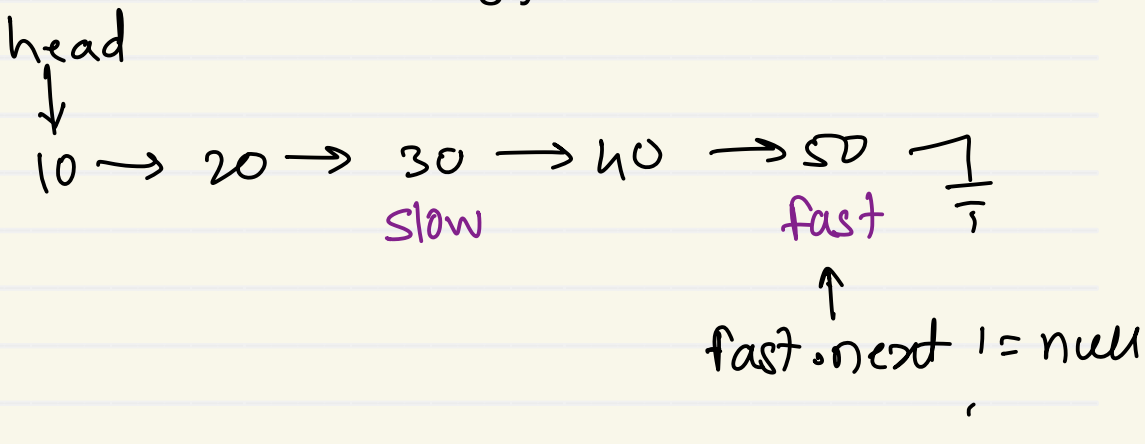
$T \propto \frac{3}{2} n$

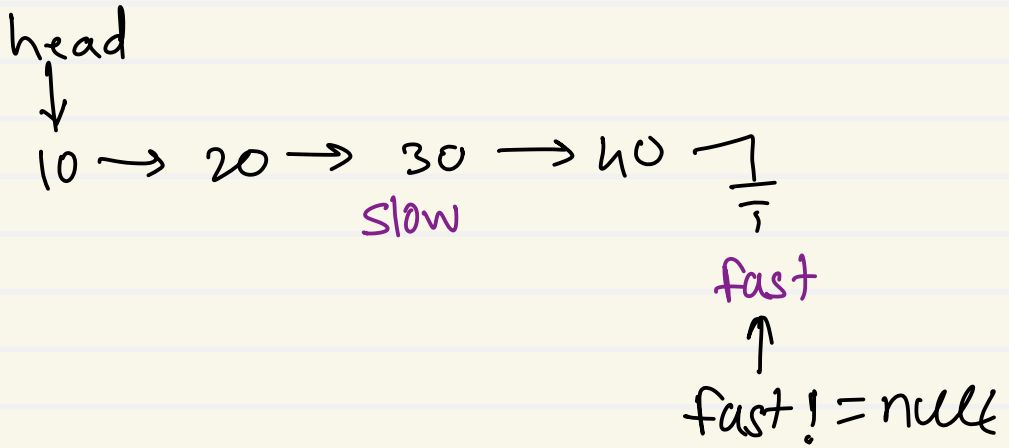
T(n) = O(n)

S(n) = O(1)

Linked List - competitive/interview question

- Find middle of singly linear linked list






```
Node findMid(Node head){
    Node slow = head, fast = head;
    while(fast != null && fast.next != null){
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
}
```

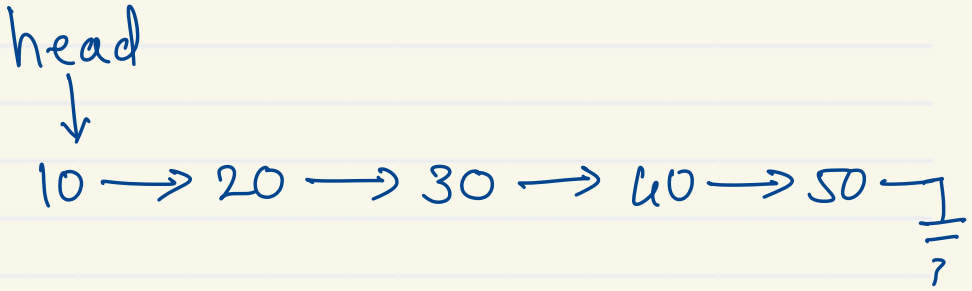
itrs = n/2

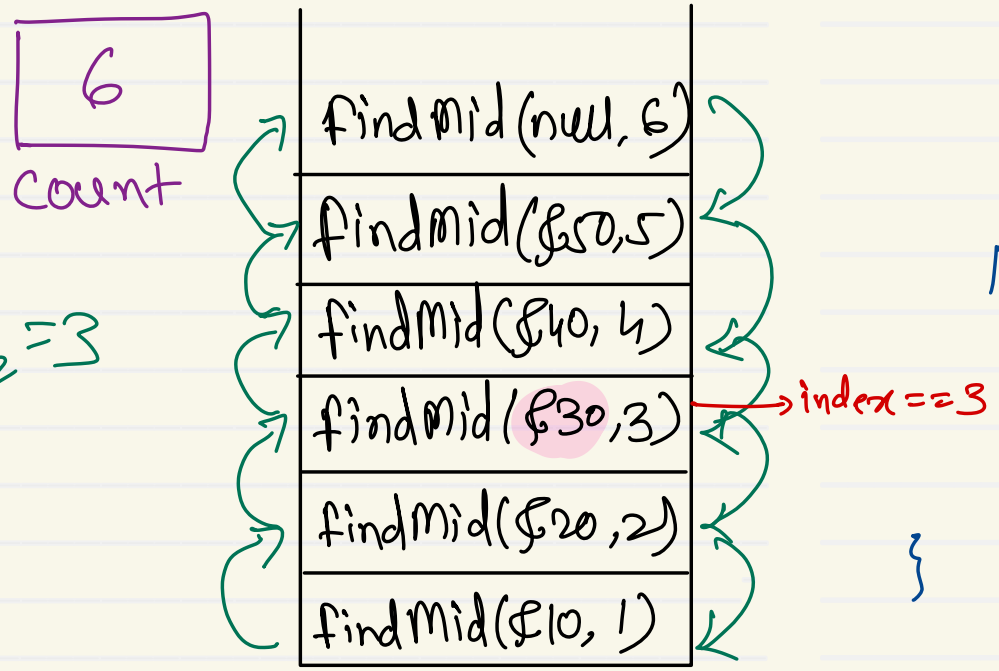
T ∝ n/2

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

Linked List - competitive/interview question

- Find middle of singly linear linked list





count/2 = 6/2 = 3

```
int count;
findMid(head, 1);

Node findMid(Node trav, int index) {
    if (trav == null) {
        count = index;
        return null;
    }
    Node ret = findMid(trav.next, index+1);
    if(index == count/2)
        return trav;
    retur ret;
}
```

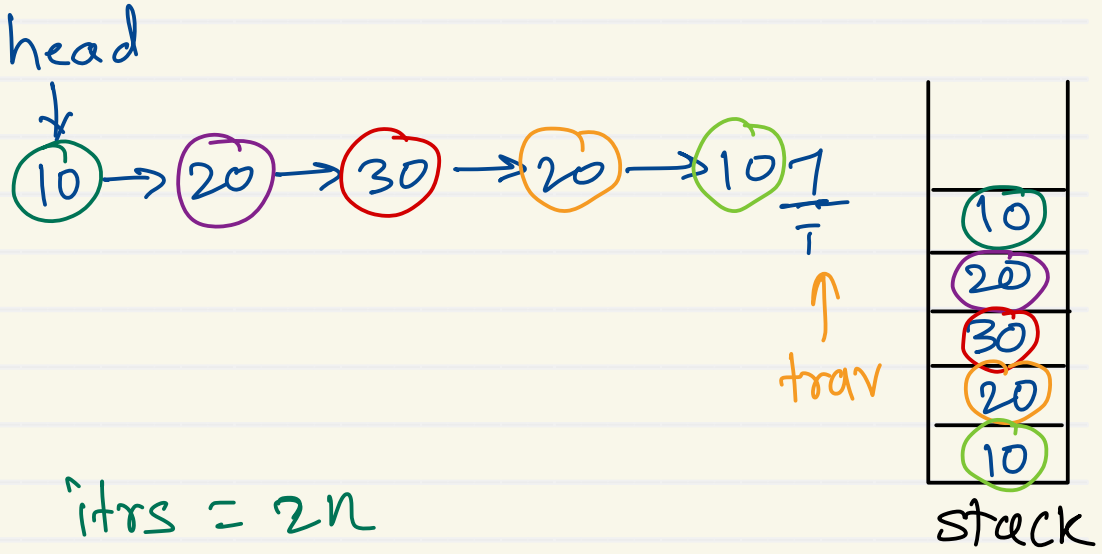
T(n) = O(n)

S(n) = O(n)

Linked List - competitive/interview question

- Check if linked list is palindrome

1. traverse list & push all nodes on stack one by one
2. pop elements from stack & compare with list elements



itrs = 2n

T ∝ 2n

T(n) = O(n)

aux space

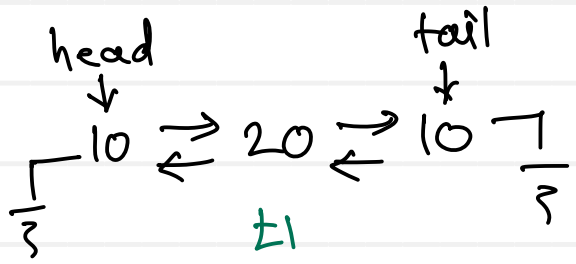
S(n) = O(n)

```
boolean isPalindrome(Node head) {
    Stack<Integer> st = new Stack<>();
    Node trav = head;
    while (trav != null) {
        st.push(trav.data);
        trav = trav.next;
    }
    Node trav = head;
    while(!st.isEmpty()) {
        if(trav.data != st.pop())
            return false;
        trav = trav.next;
    }
    return true;
}
```

T(n) = O(n)

S(n) = O(n)

Linked list - Check palindrome

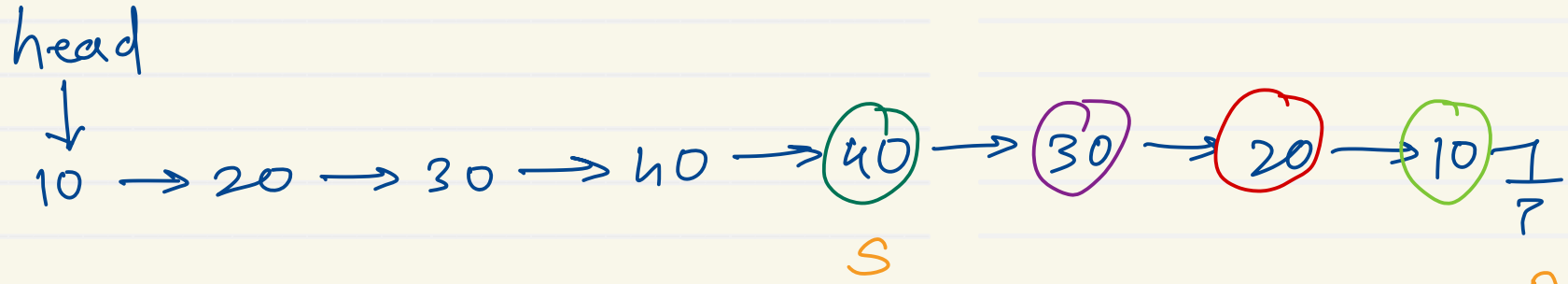


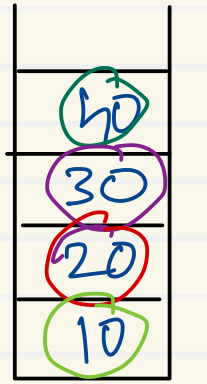
T(n) = O(n)
S(n) = O(1)

```
boolean isPalindrome( ){
    Node t1 = head, t2 = tail;
    while(t1 != null && t2 != null){
        if(t1.data != t2.data)
            return false;
        t1 = t1.next;
        t2 = t2.next;
        if(t1 == t2)
            return true;
    }
    return true;
}
```

Linked List - competitive/interview question

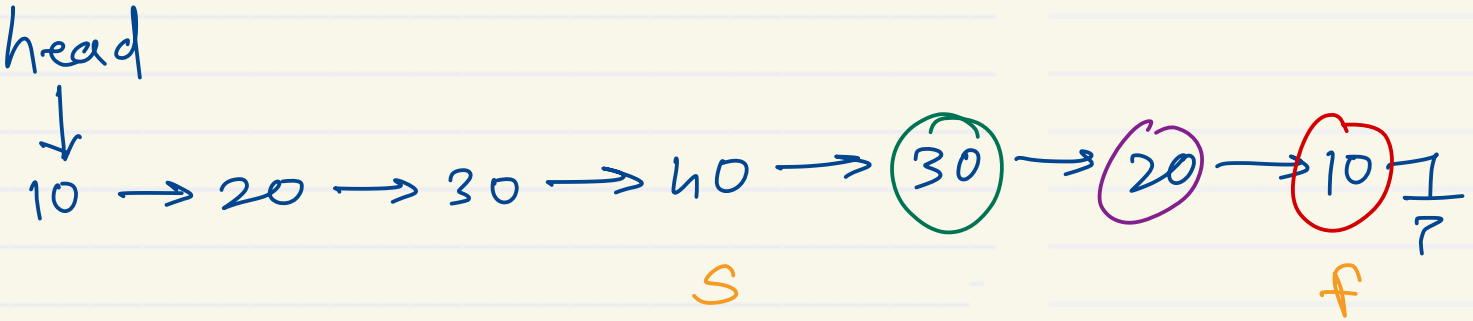
- Check if linked list is palindrome

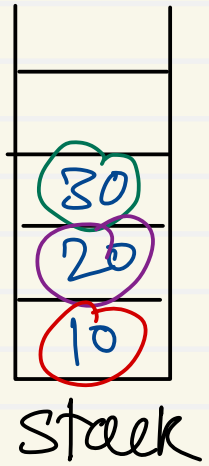




$itrs = \frac{n}{2} + \frac{n}{2} = n \rightarrow T(n) = O(n)$

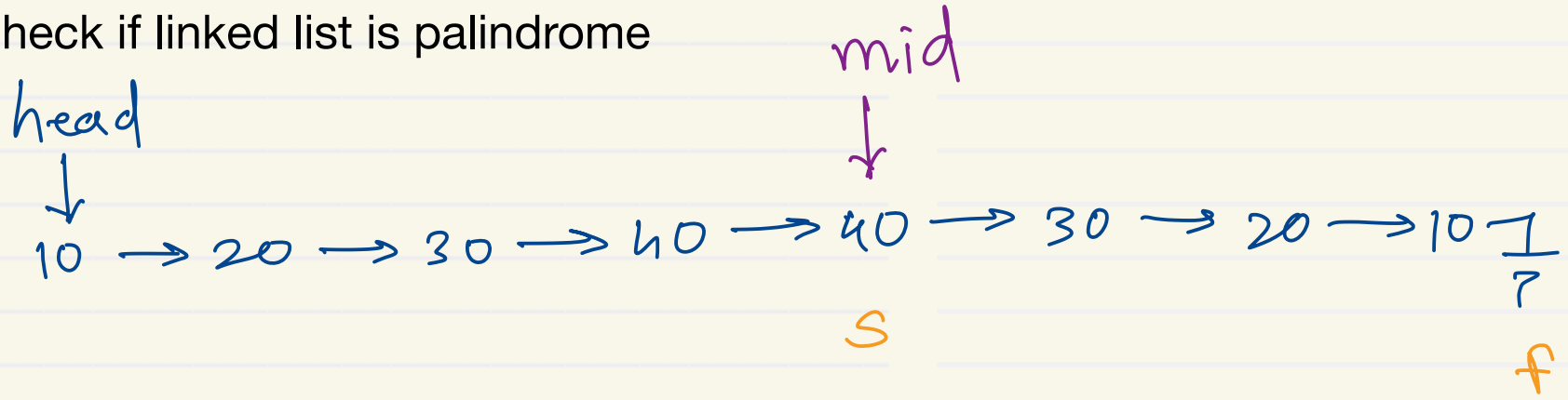
$space \propto n/2 \rightarrow S(n) = O(n)$



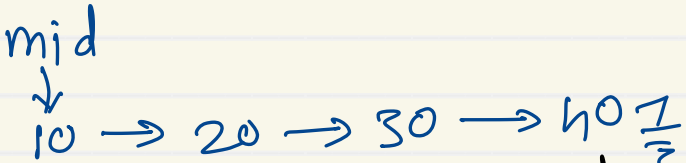


Linked List - competitive/interview question

- Check if linked list is palindrome



step 1 : find mid (use fast & slow ptrs)
step 2 : reverse 2nd half of the list



step 3 : compare 1st half & 2nd half of the list,
(head) (mid)
if same palindrome.
step 4 : reverse 2nd half of list again and append to first half → to get original list

$S(n) = O(1)$

$ifrs = n/2 + n/2 + n/2 + n/2$

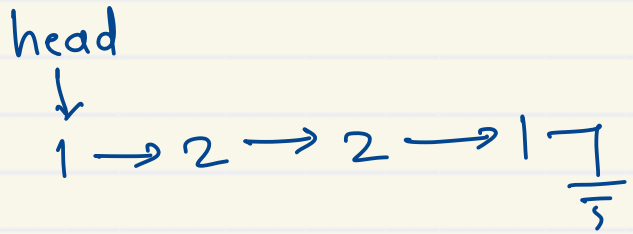
$= \frac{4n}{2} = 2n$

$T \propto 2n$

$T(n) = O(n)$

Linked List - competitive/interview question

- Check if linked list is palindrome



nodes

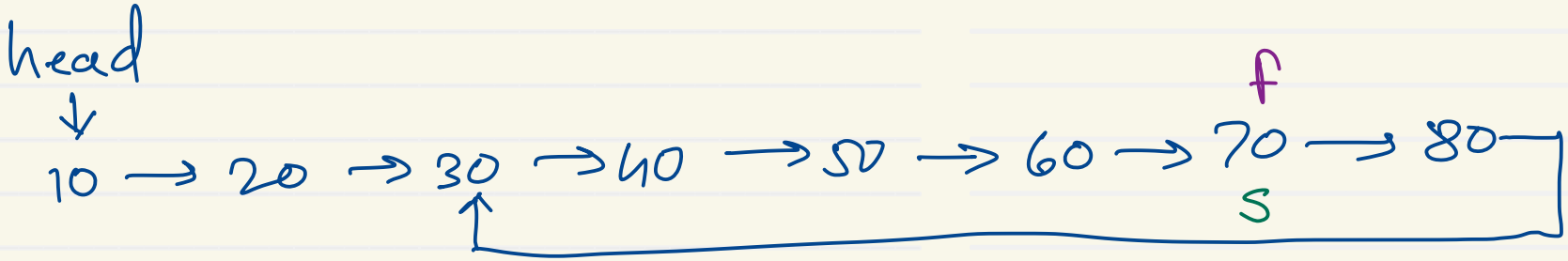
push = $\frac{n}{+} - n = \frac{3n}{2}$

pop = $2 - \frac{n}{2}$

$T(n) = O(n)$

Linked List - competitive/interview question

- Check if linked list has a loop




```
boolean hasLoop(Node head){
    Node slow=head, fast=head;
    while(fast!=null && fast.next!=null){
        fast = fast.next.next;
        slow = slow.next;
        if(fast==slow)
            return true;
    }
    return false;
}
```

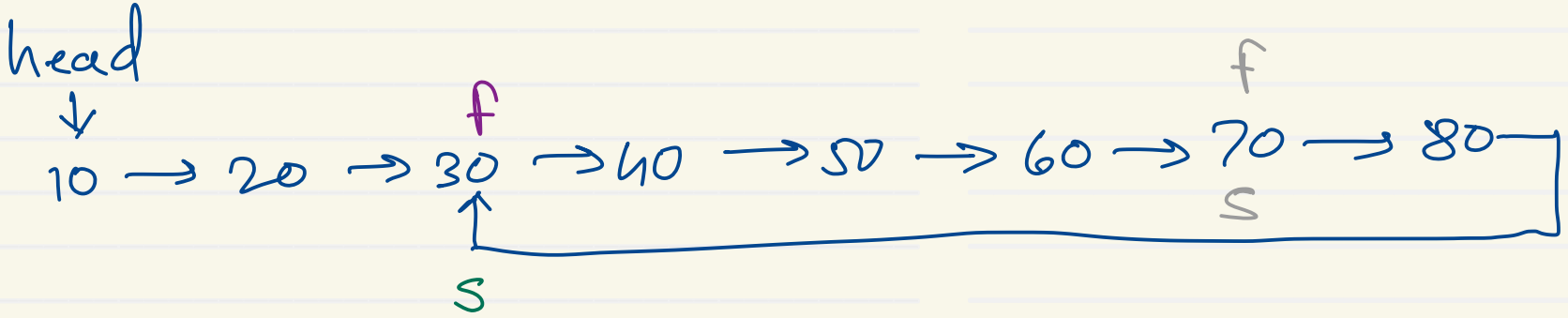
$T(n) = O(n)$

$S(n) = O(1)$

Linked List - competitive/interview question

Floyd cyclic detection algorithm

- Check if linked list has a loop



- when first time fast & slow becomes equal, cycle is detected.
- keep fast at same place & move slow to head
- now move both with same speed.
- fast & slow will be equal again, which is starting point of the loop.