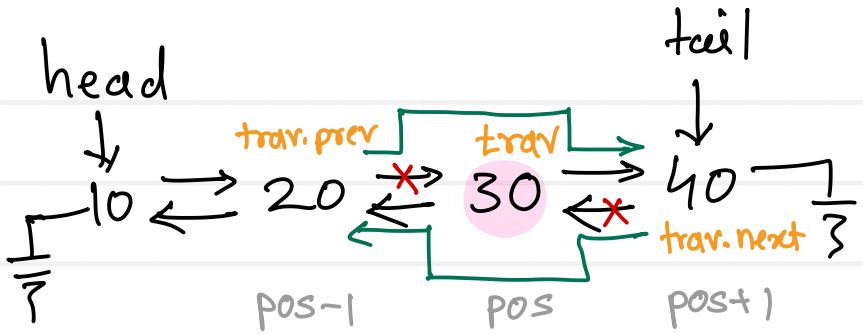
Sunbeam Institute of Information Technology
Pune and Karad

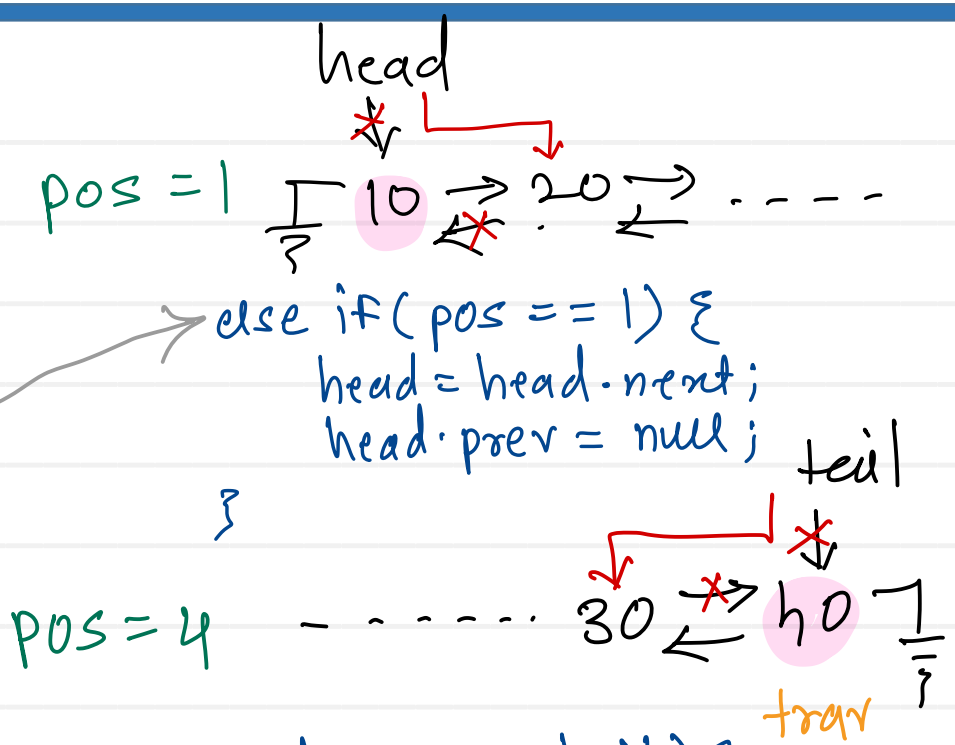
Module – Data Structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

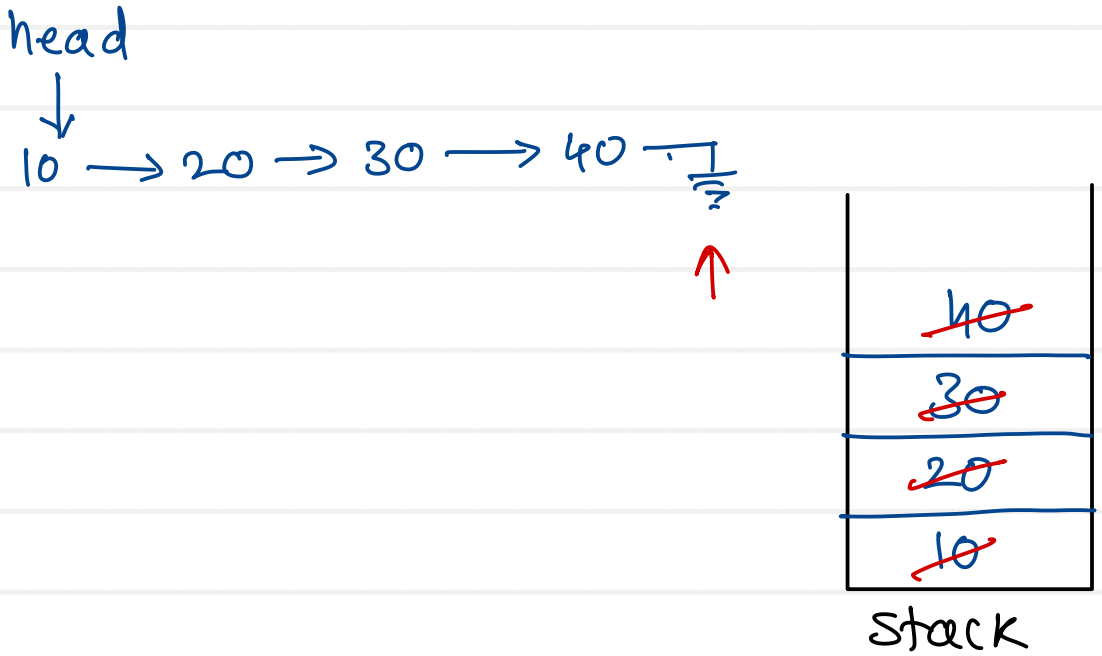



```
if ( head == null)
    return;
else if ( head == tail)
    head = tail = null;
else {
    Node trav = head;
    for(int i=1; i<pos ; i++)
        trav = trav.next;

    trav.prev.next = trav.next;
    trav.next.prev = trav.prev;
```




```
else if( pos == 1) {
    head = head.next;
    head.prev = null;
}

if( trav == tail) {
    tail = tail.prev;
    tail.next = null;
}
```

Singly linear linked list - Display reverse (using stack)



List : 40 , 30 , 20 , 10

```
void displayReverse(Node head) {
    stack<Integer> st = new stack<>();
    Node trav = head;
    while(trav != null) {
        st.push(trav.data);
        trav = trav.next;
    }
    while(!st.isEmpty()) {
        sysout(st.popC));
    }
}
```

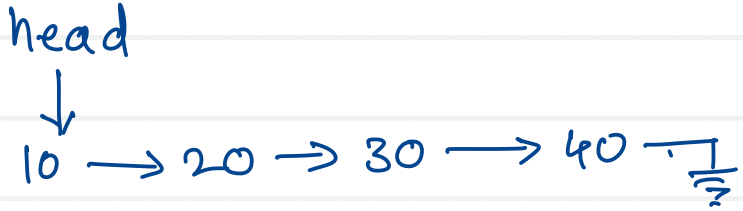
stack - Auxilliary space

$S(n) = O(n)$

to push nodes - $O(n)$
to pop nodes - $O(n)$

$T(n) = O(n)$

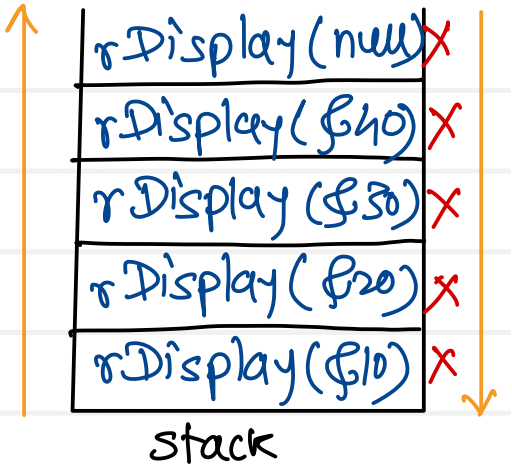
Singly linear linked list - Display reverse (using recursion)



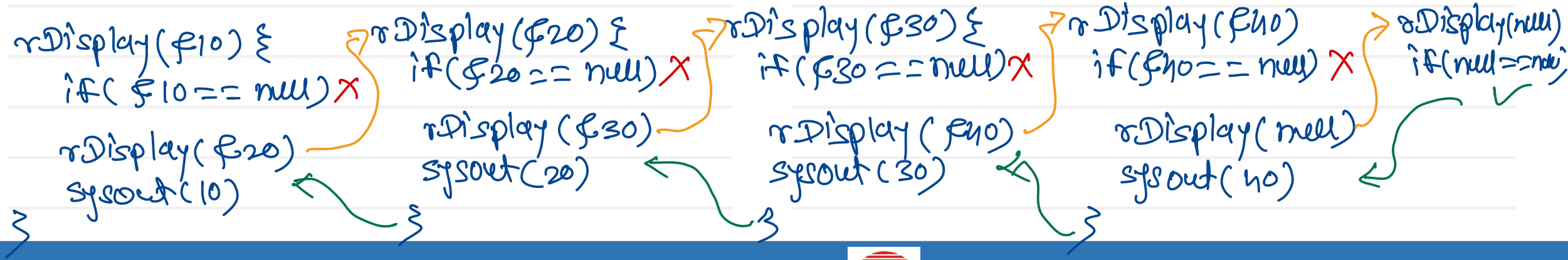
$T(n) = O(n)$

$S(n) = O(n)$

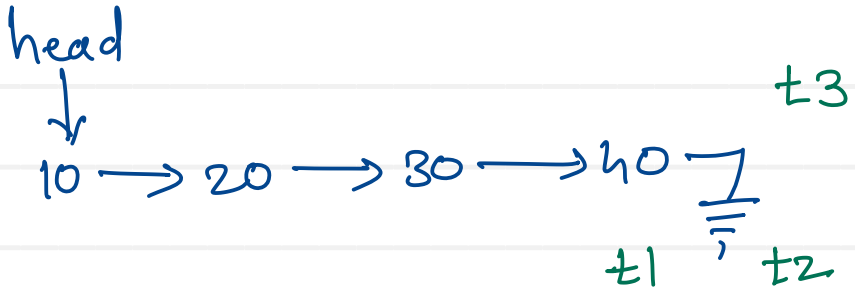
```
void rDisplay(Node trav) {
    if(trav == null)
        return;
    rDisplay(trav.next);
    sysout(trav.data);
}
```

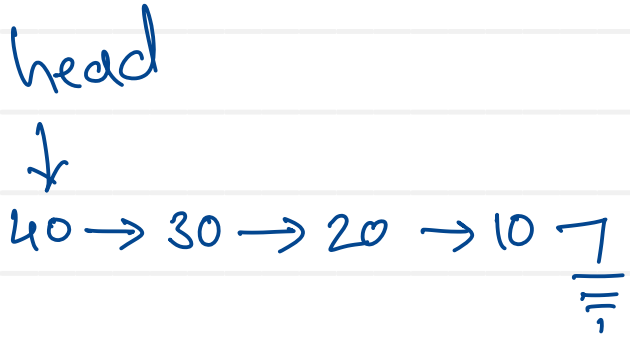


List : 40, 30, 20, 10



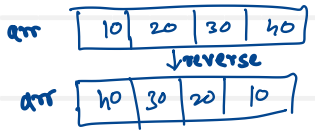
Singly linear linked list - Reverse





t1	t2	t2 != null	t3
null	&10	T	&20
&10	&20	T	&30
&20	&30	T	&40
&30	&40	T	null
&40	null		

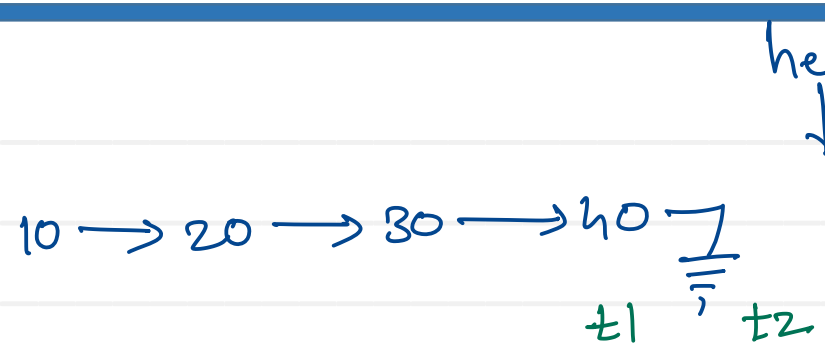
```
void reverseList( ) {
    Node t1 = null;
    Node t2 = head;
    while(t2 != null) {
        Node t3 = t2.next;
        t2.next = t1;
        t1 = t2;
        t2 = t3;
    }
    head = t1;
}
```

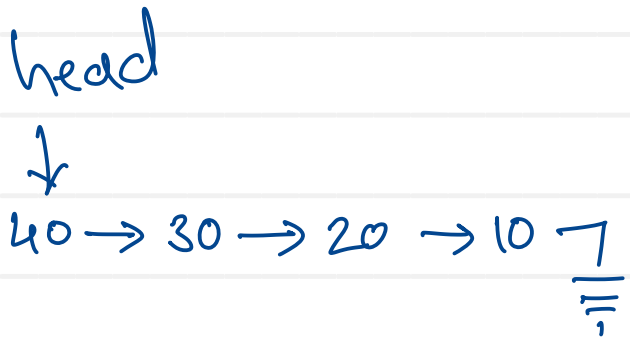


$T(n) = O(n)$

$S(n) = O(1)$

Singly linear linked list - Reverse





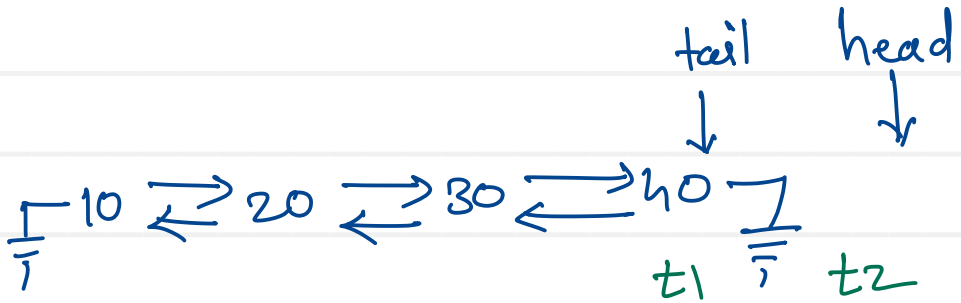
t1	t2	t2 != null	head
null	&10	T	&20
&10	&20	T	&30
&20	&30	T	&40
&30	&40	T	null
&40	null		

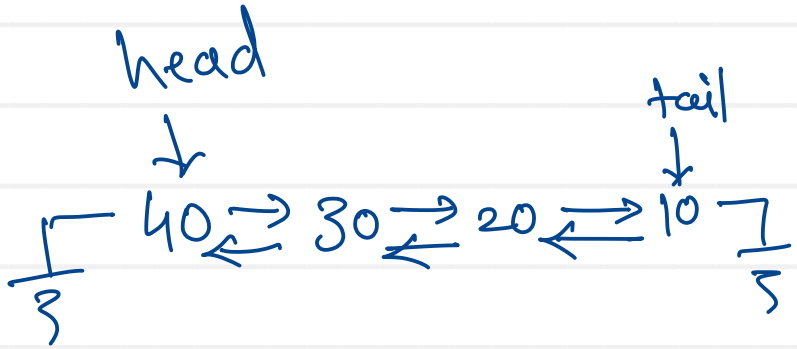
```
void reverseList( ) {
    Node t1 = null;
    Node t2 = head;
    while(t2 != null) {
        head = t2.next;
        t2.next = t1;
        t1 = t2;
        t2 = head;
    }
    head = t1;
}
```

$T(n) = O(n)$

$S(n) = O(1)$

Doubly linear linked list - Reverse





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

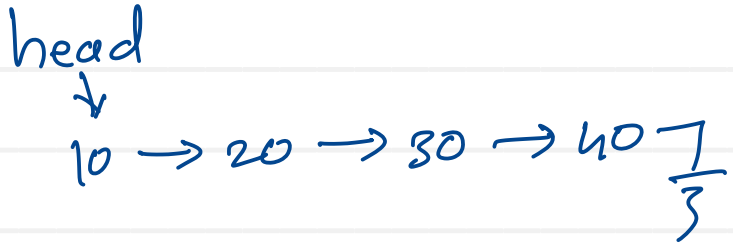
$t2.prev = t1 \Rightarrow t1.prev = t2$

```
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;
```

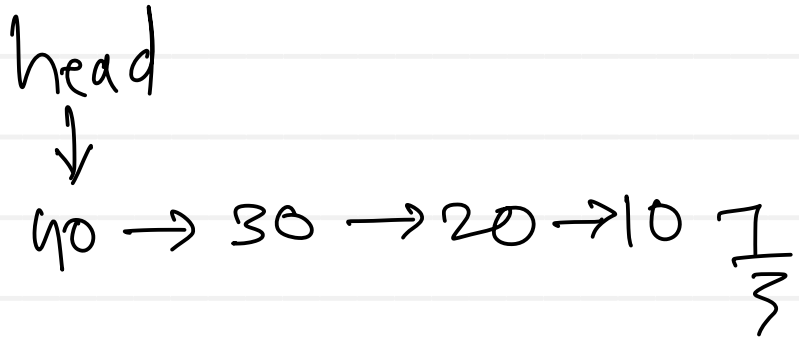
$T(n) = O(n)$

$S(n) = O(1)$

Singly linear linked list - Reverse



		current	last
1	revReverse(&10)	&10	&20
2	revReverse(&20)	&20	&30
3	revReverse(&30)	&30	&40
4	revReverse(&40)	&40	

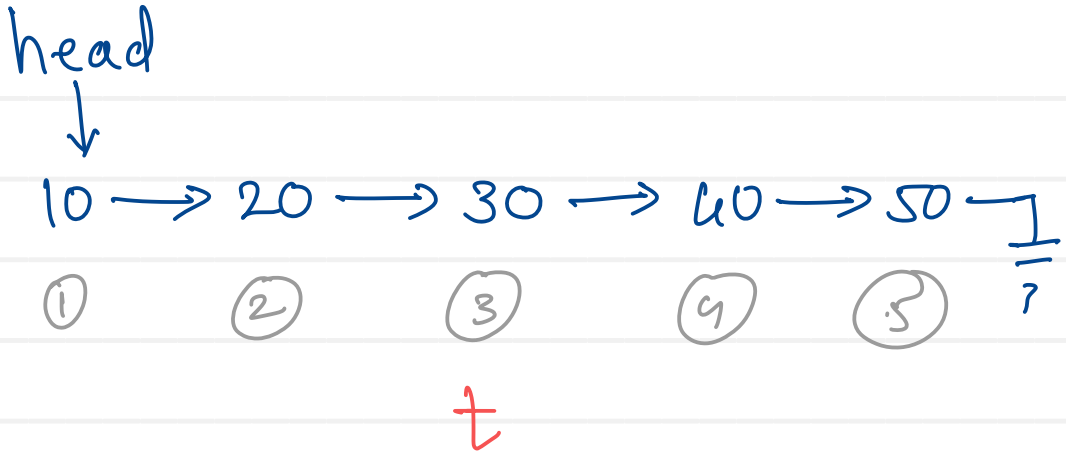



```
Node recReverse(Node current) {
    if(current.next == null) {
        head = current;
        return current;
    }
    Node last = recReverse(current.next);
    last.next = current;
    current.next = null;
    return current;
}
```

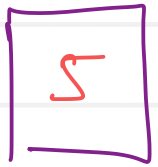
$T(n) = O(n)$

$S(n) = O(n)$

Singly linear linked list - Find mid




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

    return trav;
}
```



to count nodes – O(n)

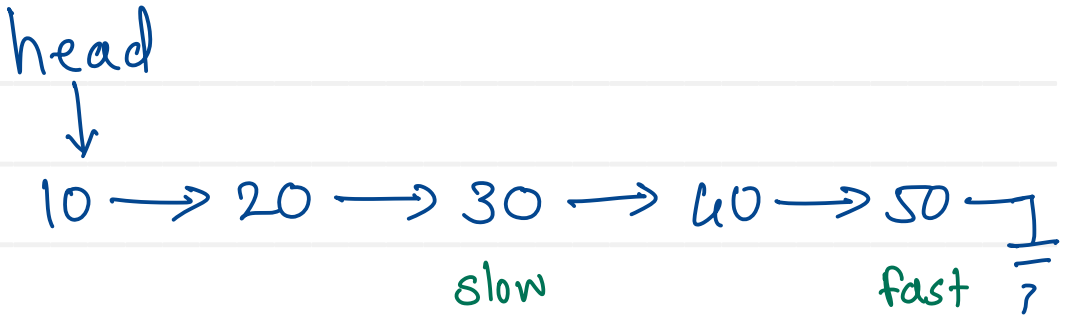
to traverse upto middle node – O(n)

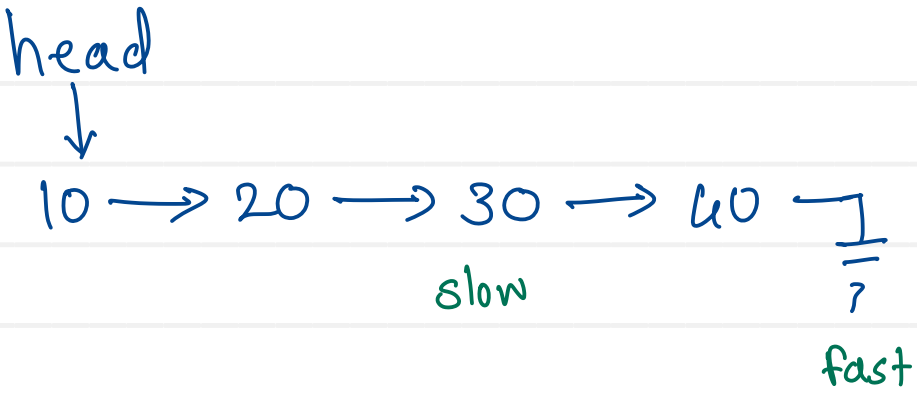
$T(n) = O(n)$

$S(n) = O(1)$

i	i<2
0	T
1	T
2	F

Singly linear linked list - Find mid



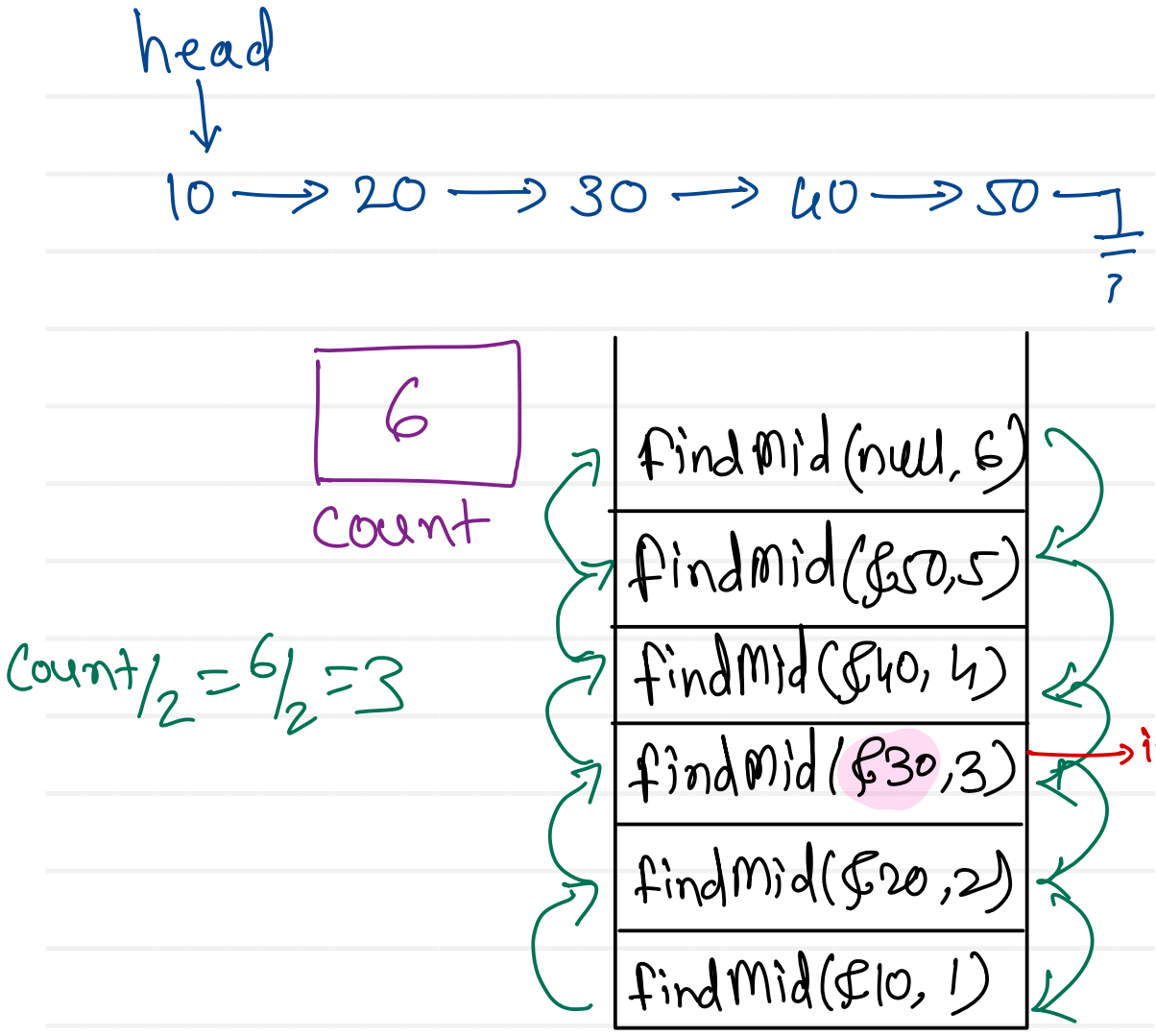



```
Node findMid(Node head){
    Node slow = head, fast = head;
    while(fast != null && fast.next != null){
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
}
```

$T(n) = O(n)$

$S(n) = O(1)$

Singly linear linked list - Find mid

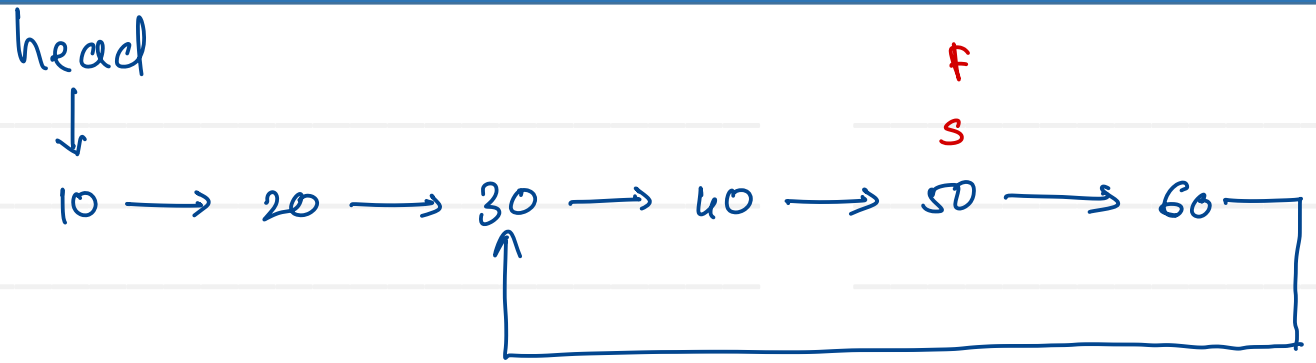



```
int count;
findMid(head, 1);

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)

Singly linear linked list - Detect loop



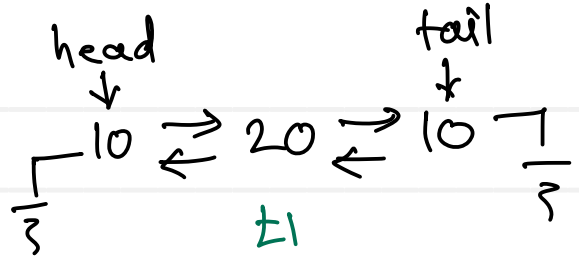
Homework:
Detect the node where loop starts.

```
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;
}
```

$T(n) = O(n)$

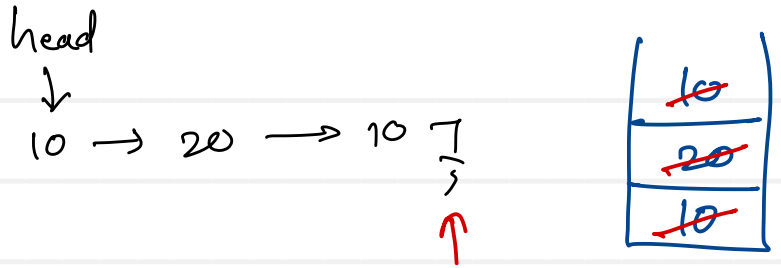
$S(n) = O(1)$

Linked list - Check palindrome




```
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;
}
```

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




```
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;
}
```

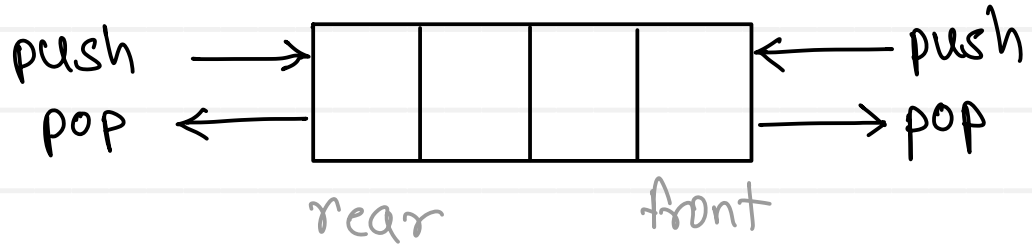
T(n) = O(n)
S(n) = O(n)

Linked list - Applications

- linked list is a dynamic data structure because it can grow or shrink at runtime.
- Due to this dynamic nature, linked list is used to implement other data structures like
  1. Stack
  2. Queue
  3. Hash table
  4. Graph

Stack	Queue
LIFO	FIFO
1. Add First Delete First	1. Add First Delete Last
2. Add Last Delete Last	2. Add Last Delete First

Deque
(Double Ended Queue)



1) Input Restricted Deque
   - one input is closed / restricted
   - both output ends are open
2) Output Restricted Deque
   - one output is closed / restricted
   - both input ends are open

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com