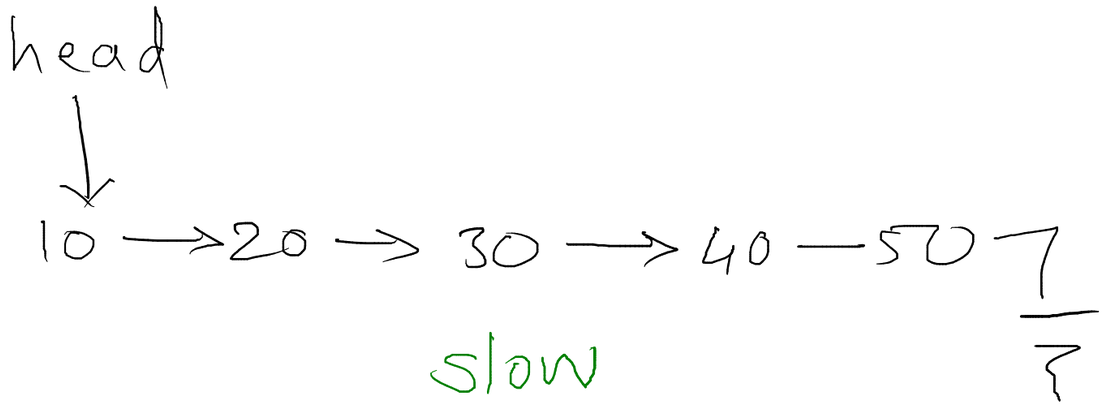
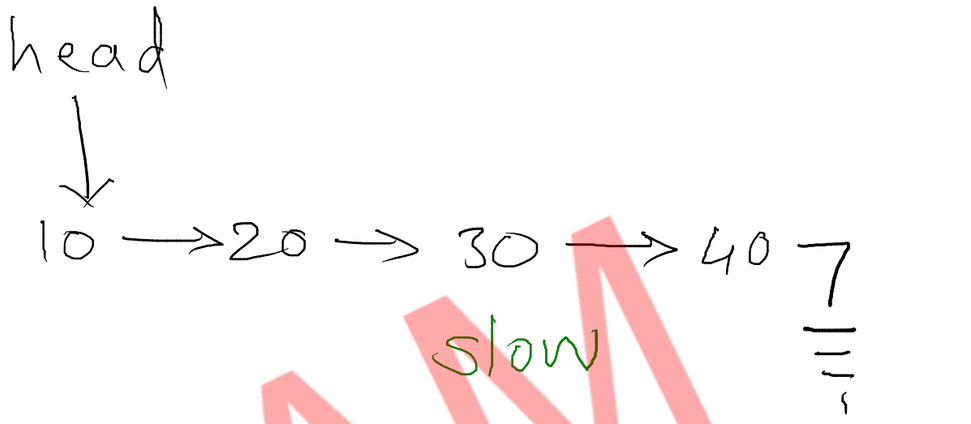
Linked List - Find Mid






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

$T(n) = O(n)$

Find middle of singly linear linked list using single pointer.

1) count number of nodes in linked list (by traversing)
2) again traverse till count/2 to find middle

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

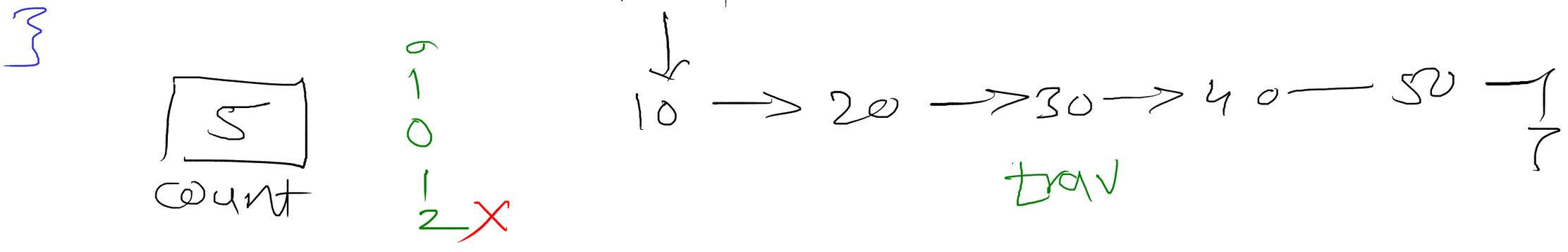
$\leftarrow n$

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

$\leftarrow n/2$

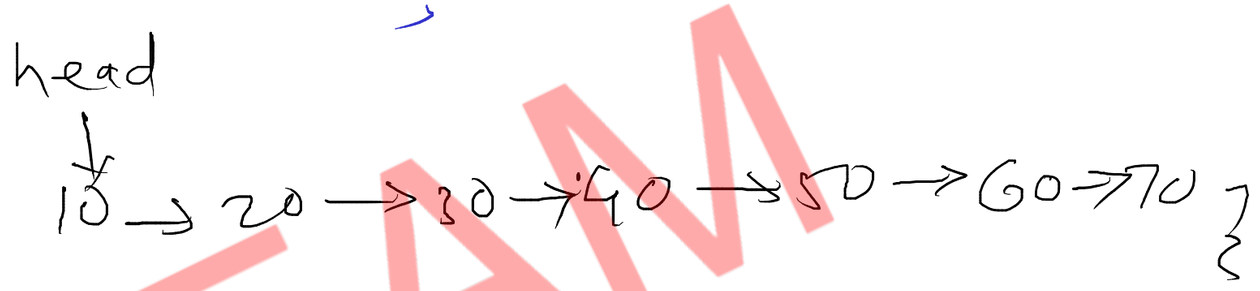
$Time \propto \frac{3}{2}n$

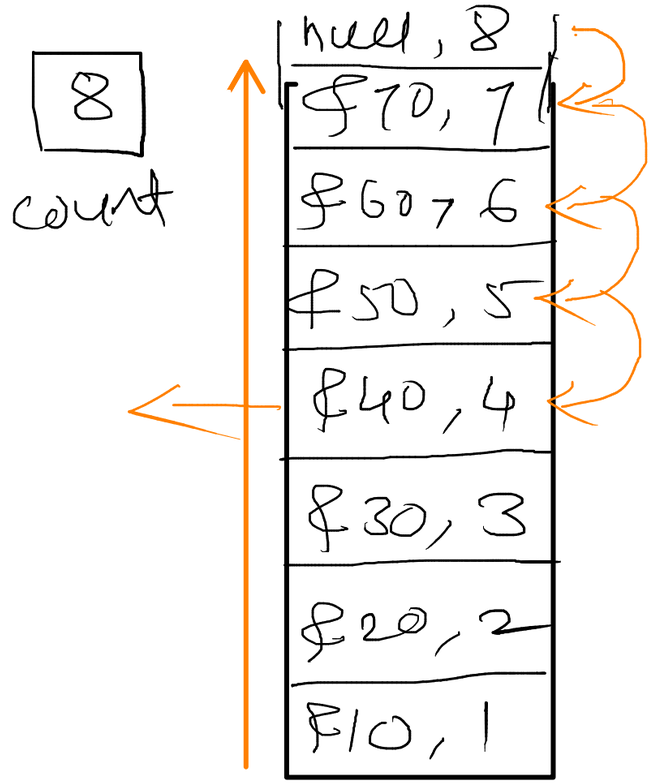
$T(n) = O(n)$



Find middle of singly linear linked list using single pointer and recursion

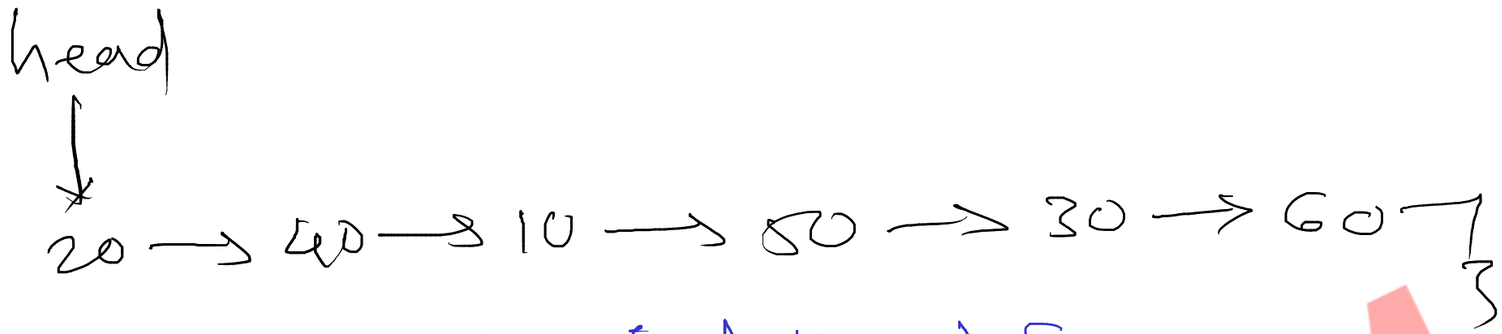
```
int count;
Node findMid(Node trav, int index) {
    if(trav == null) {
        count = index;
        return null;
    }
    last = findMid(trav.next, index+1);
    if(index == count/2)
        return trav;
    return last;
}
findMid(head, 1)
```





$T(n) = O(n)$

Linked List - Searching




```
Node linear_search(int key){
    Node trav=head;
    while(trav != null){
        if(key == trav.data)
            return trav;
        trav= trav.next;
    }
    return null;
}
```

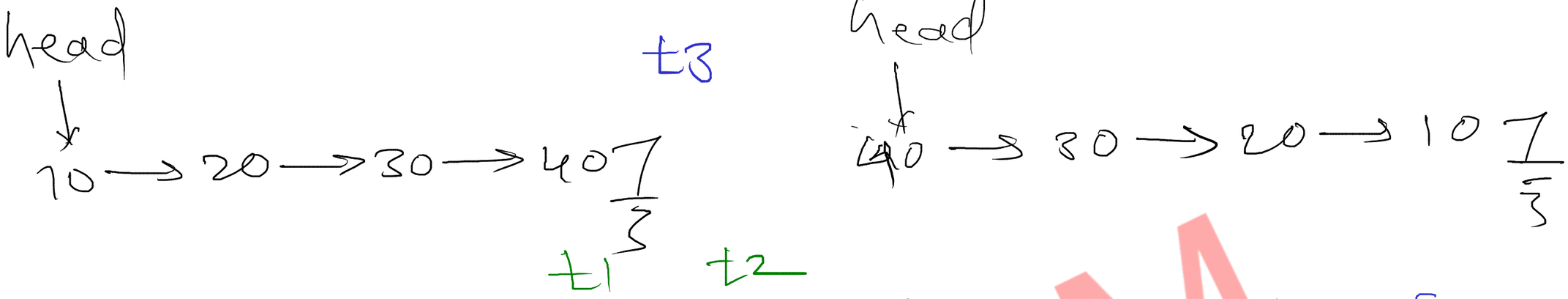
key=50

trav
$20
$40
$10
$50 ←

key=80

trav
$20
$40
$10
$50
$30
$60
null ←

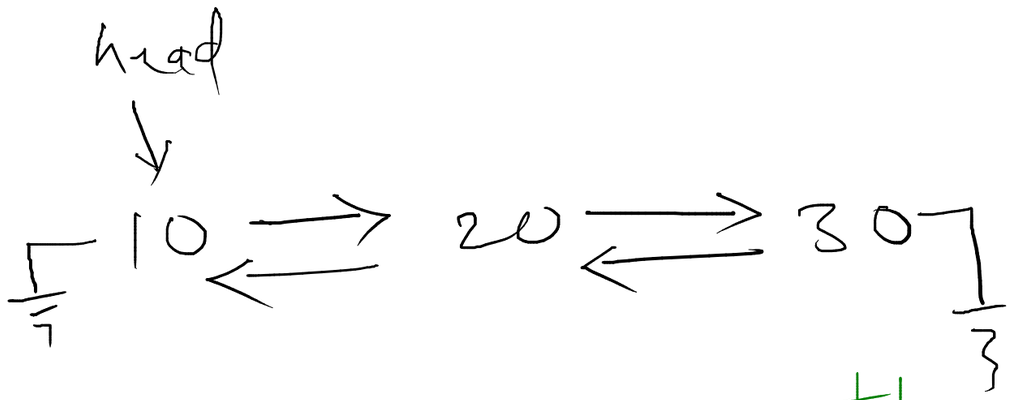
Linked List - Reverse




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

$T(n) = O(n)$

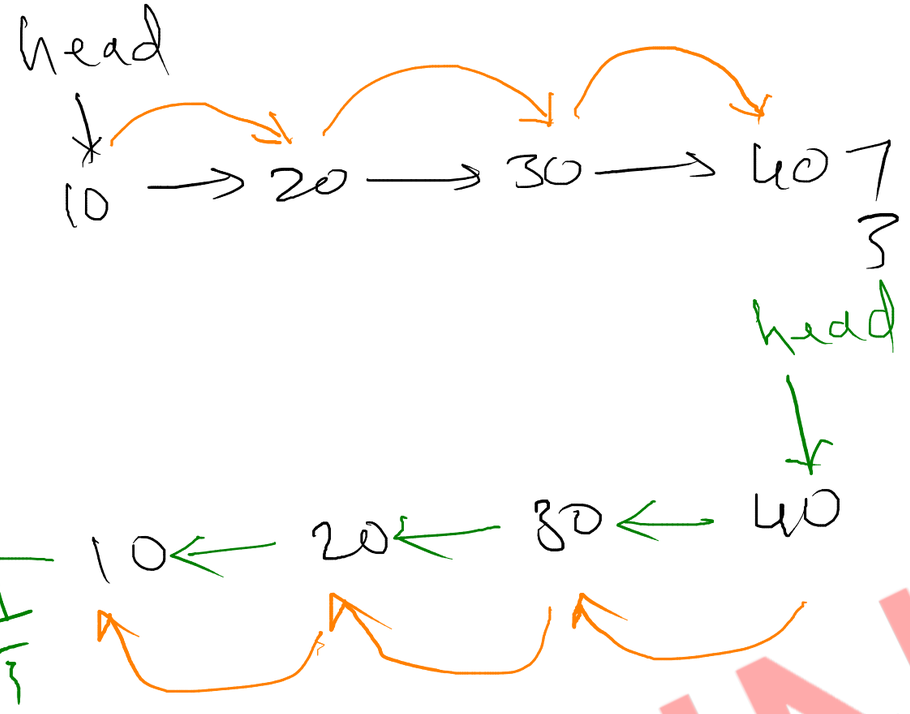
```
void reverseList( ) {
    Node t1 = head;
    Node t2 = head.next;
    head.next = null;
    while(t2 != null){
        head = t2.next;
        t2.next = t1;
        t1 = t2;
        t2 = head;
    }
    head = t1;
}
```



t1 t2

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

Reverse singly linked list using recursion.




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

main → recReverse(head)

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

Sort the singly linked list.

```
for(i=0; i<N-1; i++) {
    for(j=i+1; j<N; j++) {
        if(arr[i] > arr[j]) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}
```

```
void selectionSort() {
    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) {
                int temp = i.data;
                i.data = j.data;
                j.data = temp;
            }
        }
    }
}
```

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

Check if linked list is palindrome.

① traverse list & pull all data on stack.

② traverse list and pop data from stack one by one, compare them. if all elements are same, then list is palindrome otherwise not palindrome.

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

itr = 2n

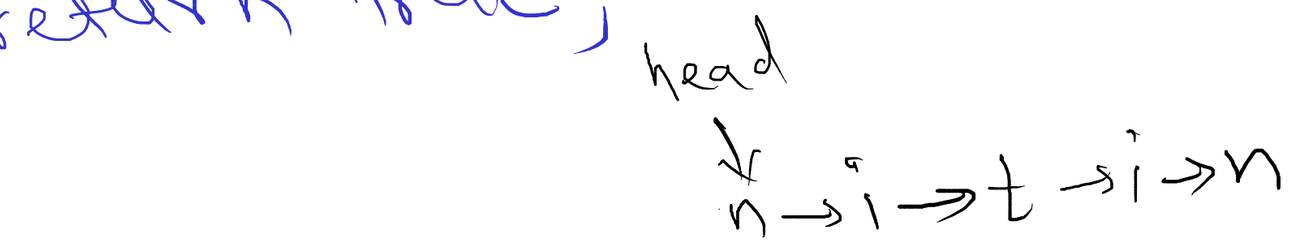
T ∝ 2n

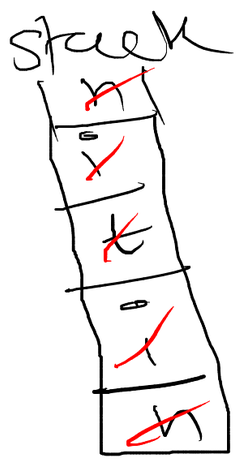
$T(n) = O(n)$

stack - auxiliary space

space ∝ n

$AS(n) = O(n)$





n != n ✗

i != i ✗

t != t ✗

i != i ✗

n != n ✗

Check if linked list contains a loop.


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