Sunbeam Institute of Information Technology
Pune and Karad

Algorithms and Data structures

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Bubble sort

1. Compare all pairs of consecutive elements of the array one by one
2. If left element is greater than right element , then swap both
3. Repeat above steps until array is sorted (N-1)

No. of elements = n
No. of passes = n-1

pass	comps
1	n-1
2	n-2
3	:
4	2
5	1

Total comps = 1+2+3---+n
$= \frac{n(n+1)}{2}$
Time $\propto \frac{1}{2}(n^2+n)$

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

N=6

j<N-1 j<N-i

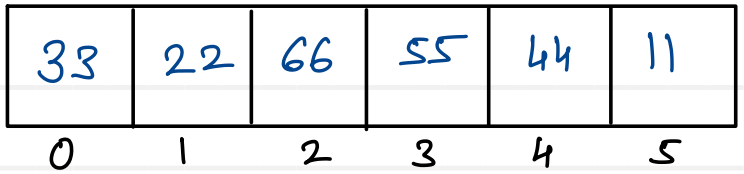
i	j	j	
1	0→4	0→4	5
2	0→4	0→3	4
3	0→4	0→2	3
4	0→4	0→1	2
5	0→4	0→0	1

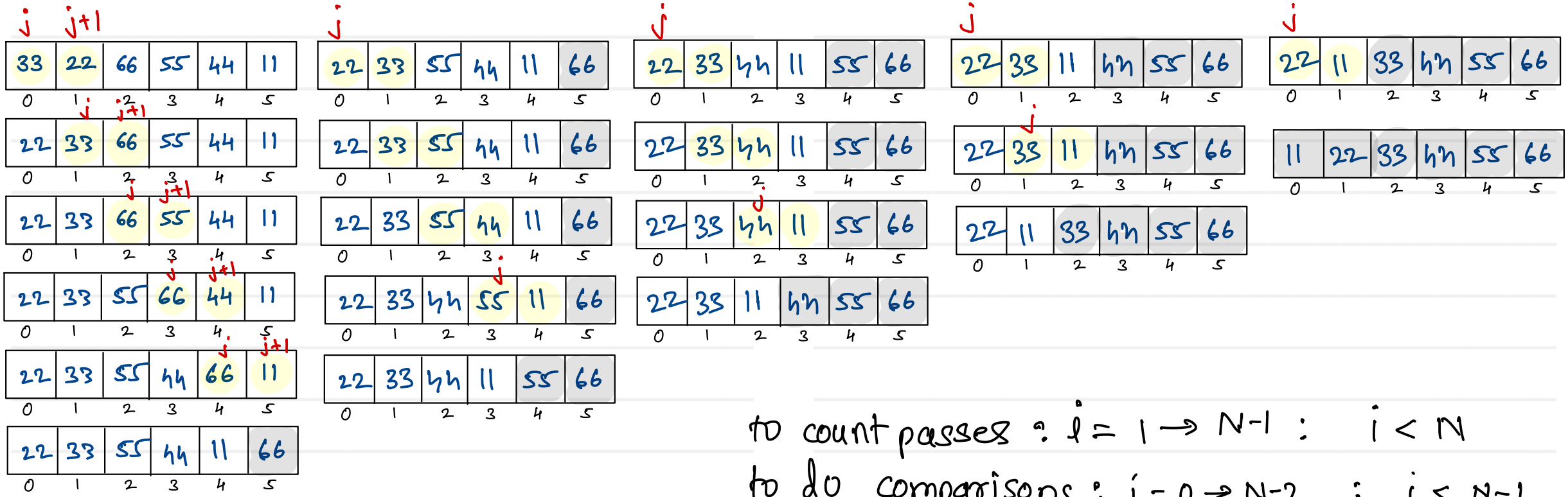
Best case :

11 22 33 44 55 66

$T(n) = O(n)$

Bubble sort





to count passes : $i = 1 \rightarrow N-1$: $i < N$

to do comparisons : $j = 0 \rightarrow N-2$: $j < N-1$

$j < N-i$

Insertion sort

1. Pick one element of the array (start from 2nd index)
2. Compare picked element with all its left neighbours one by one
3. If left neighbour is greater, move it one position ahead
4. Insert picked element at its appropriate position
5. Repeat above steps until array is sorted (N-1)

No. of elements = n
No. of passes = n-1

passes	comps
1	1
2	2
3	3
:	:
n-1	n-1

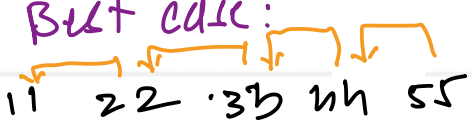
Total comps = $1+2+3+\ldots n$

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

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

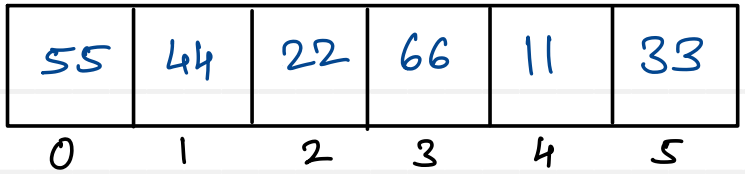
$T(n) = O(n^2)$ Worst Avg

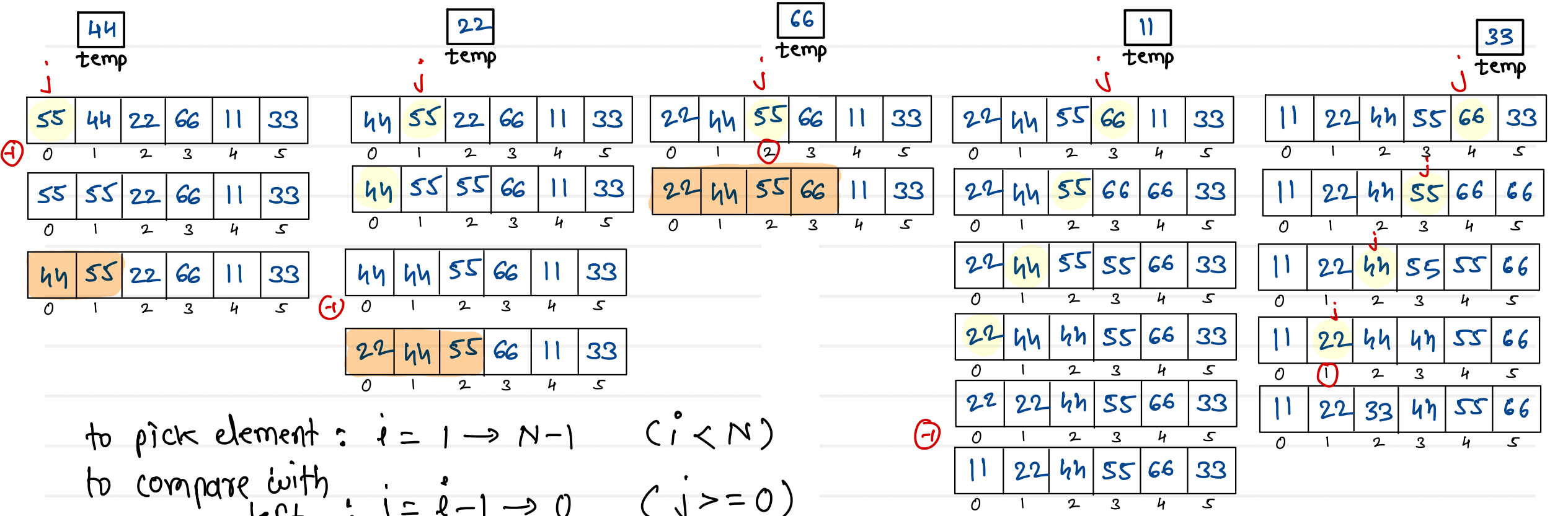
Best case:



$T(n) = O(n)$ Best

Insertion sort



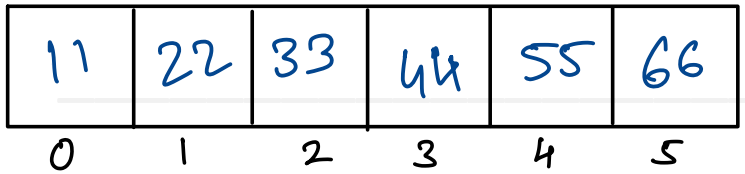


to pick element : $i = 1 \rightarrow N-1$ $(i < N)$

to compare with left : $j = i-1 \rightarrow 0$ $(j >= 0)$

Insertion sort

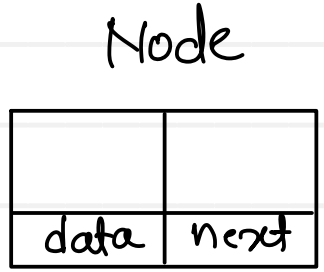
```
for(i=1; i<N; i++) {
    temp = arr[i];
    for(j=i-1; j>=0; j--) {
        if(arr[j] > temp)
            arr[j+1] = arr[j];
        else
            break;
    }
    arr[j+1] = temp;
}
```

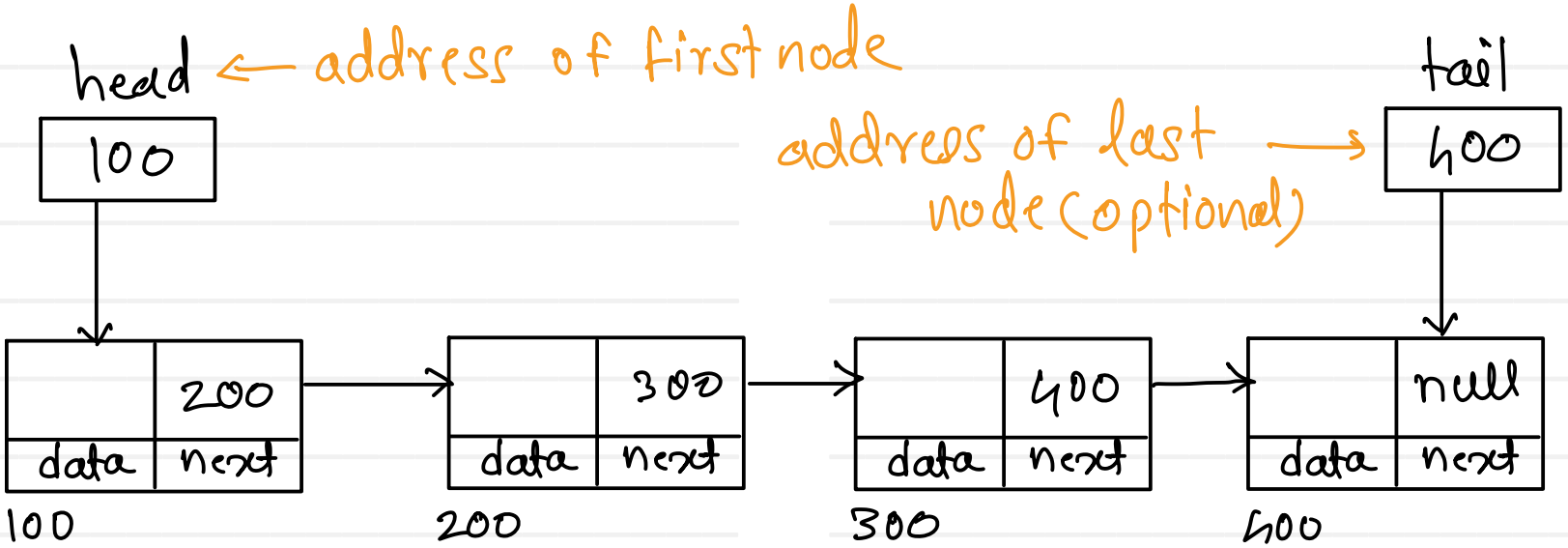


i	i<6	temp	j
1	T	44	0,-1
2	T	22	1,0,-1
3	T	66	2
4	T	11	3,2,1,0,-1
5	T	33	4,3,2,1

Linked List

- linked list is a linear data structure.
- link / address of next data is kept with its previous data.
- every element of linked list has two parts (data, link/addr) and referred as node





Linked List

Operations

1. Add first
2. Add last
3. Add position (insert)

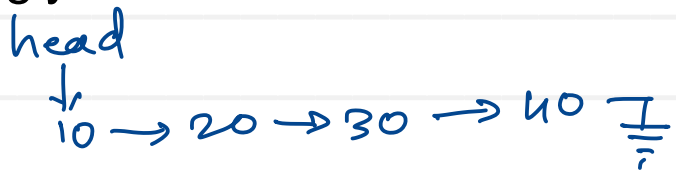
1. Delete first
2. Delete last
3. Delete position

1. Display (traverse) (forward/ backward)

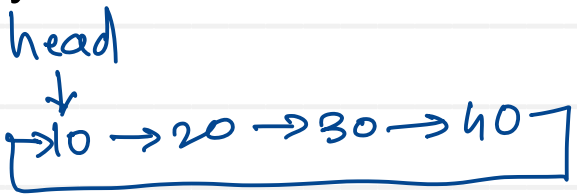
1. Search
2. Sort
3. Reverse

Types

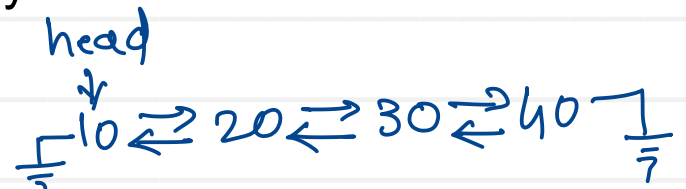
1. Singly linear linked list



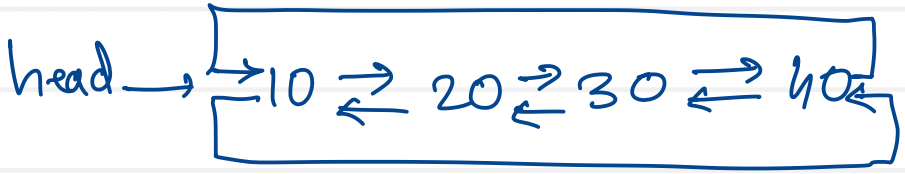
2. Singly circular linked list



3. Doubly linear linked list



4. Doubly circular linked list



Linked List

Node :
data → int, char, double, class
next → address / reference of next node

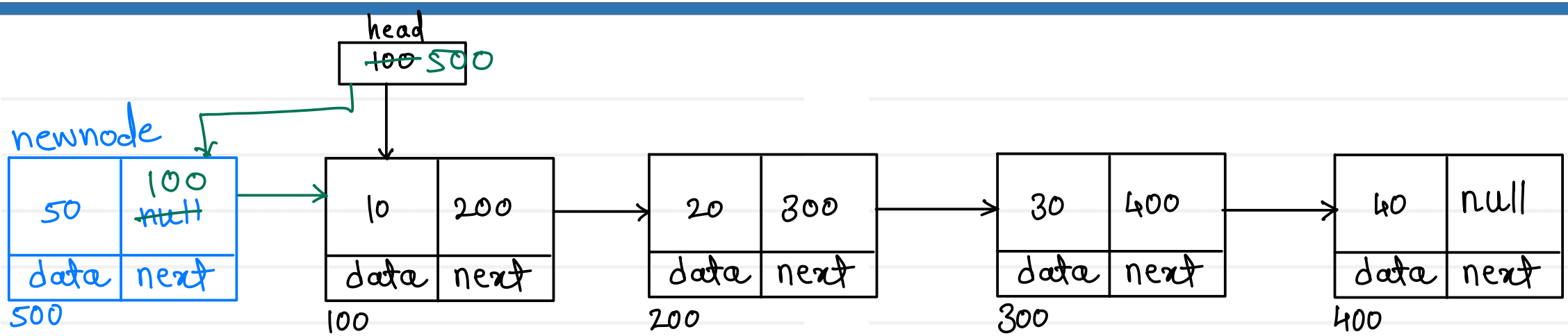
```
class Node {   ← self referntial class
    int data;
    Node next;
}
```

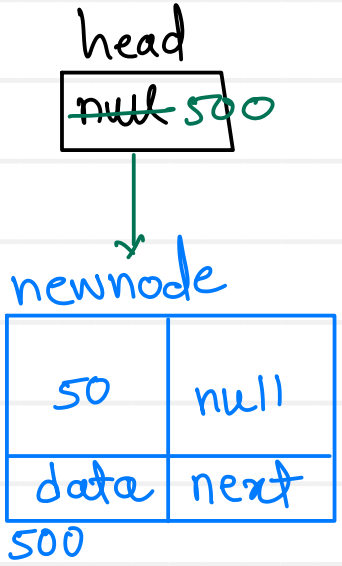
Why inner class ?
↳ to access private fields of Node class directly into List class.

Why static ?
↳ to restrict access of private fields of List class into Node class
↳ there is no dependancy of outer class to create object of inner class.

```
class List {
    static class Node {
        int data;
        Node next;
    }
    Node head;
    Node tail; (optional)
    int size;
    public List( ) { ... }
    public addNode() { ... }
    public deleteNode() { ... }
    public display() { ... }
    public reverse() { ... }
}
```

Singly linear Linked List - Add first



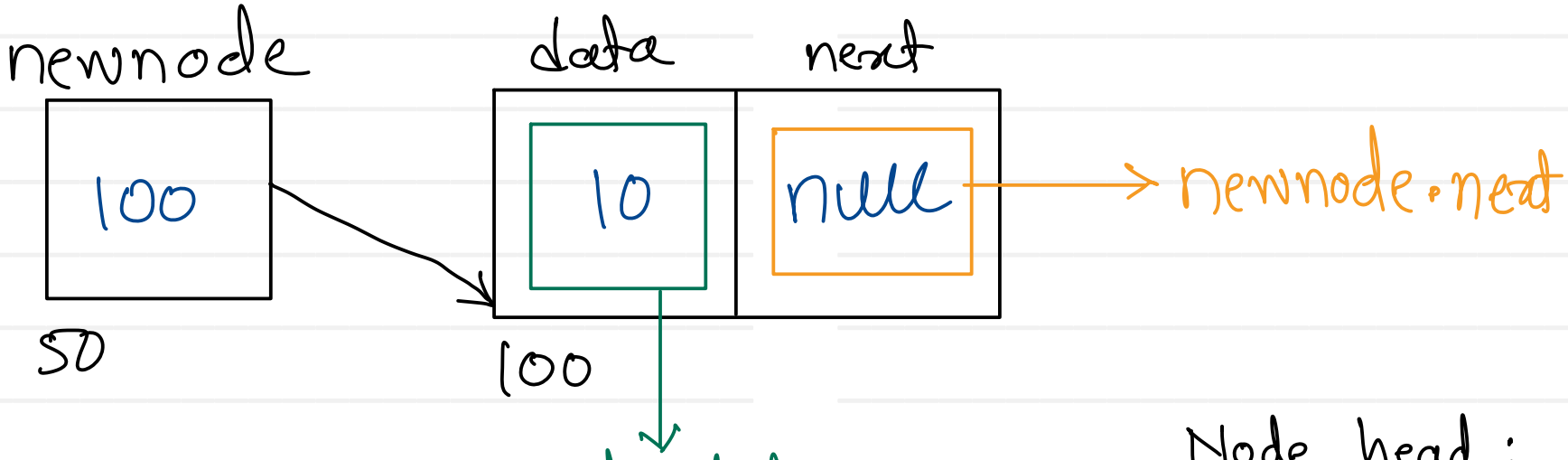



```
newnode.next = head;
head = newnode;
```

1. create node with given data
2. add first node into next of newnode
3. move head on newnode

$T(n) = O(1)$

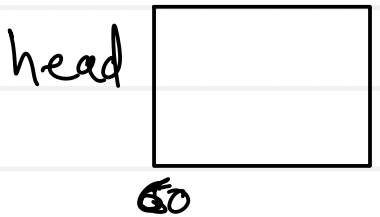
Node newnode = new Node(10);



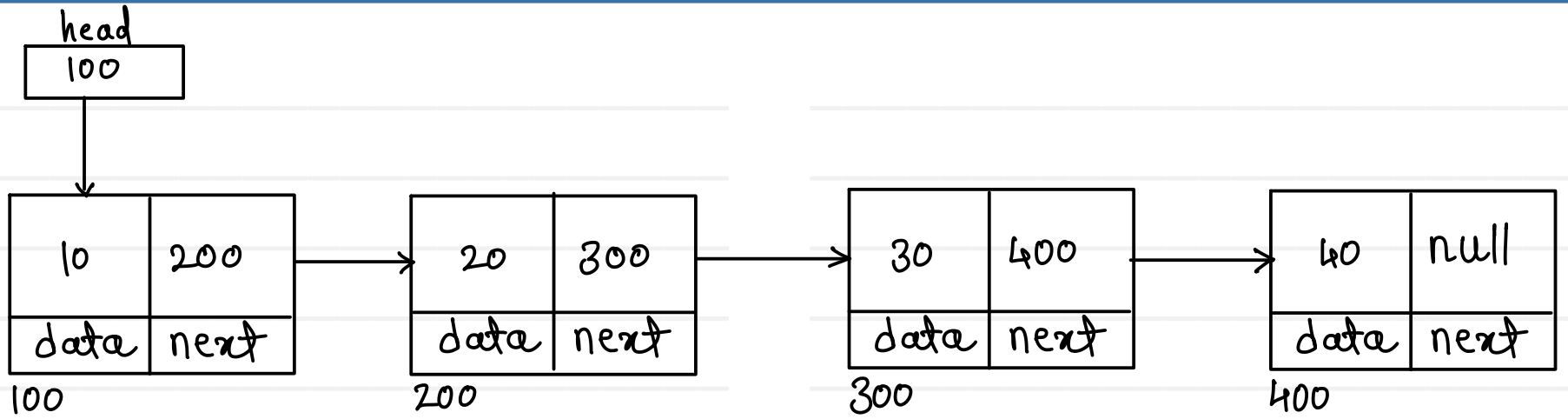
newnode.data = 2. ← writing

2. = newnode.data ← reading

Node head;



Singly linear Linked List - Display



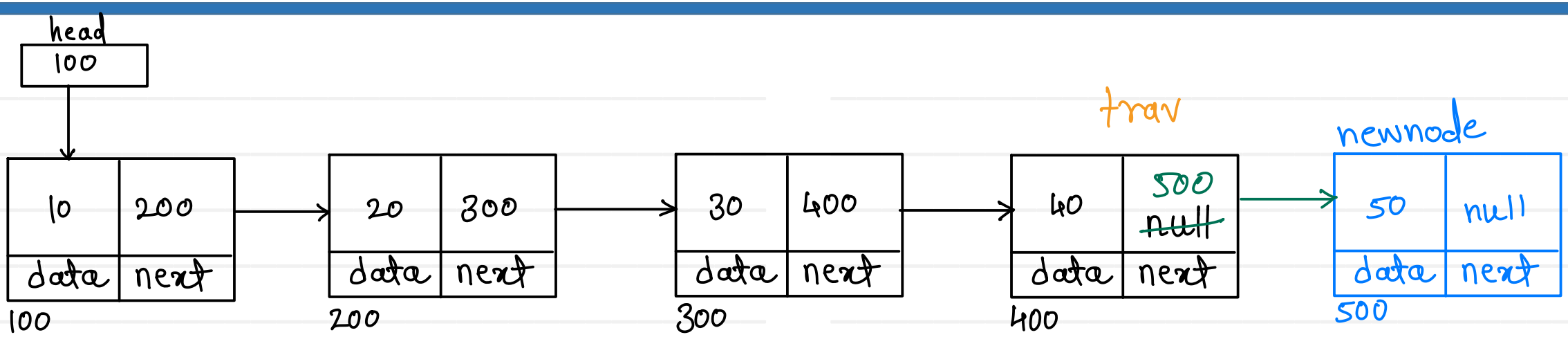

```
Node trav = head;
while(trav != null) {
    sysout(trav.data);
    trav = trav.next;
}
```

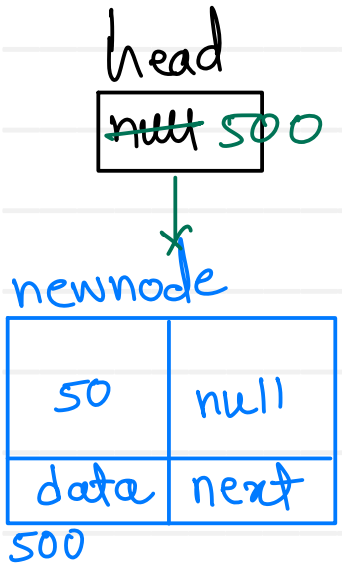
trav	trav != null	trav.data
100	T	10
200	T	20
300	T	30
400	T	40
null	F	

1. create trav & start at head
2. print/visit current node (trav.data)
3. go on next node (trav.next)
4. repeat step 2 & 3 till last node

$T(n) = O(n)$

Singly linear Linked List - Add last



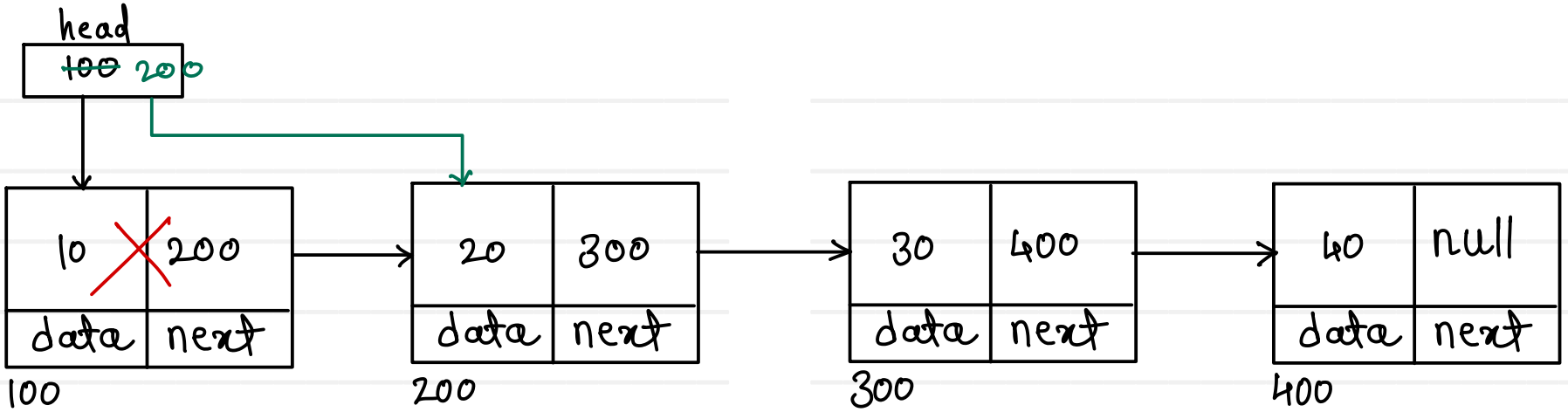


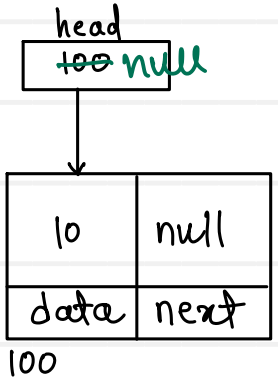
1. create node
2. if list is empty
   add newnode into head
3. if list is not empty
   a. traverse till last node
   b. add newnode into next of last node.

```
Node trav = head;
while (trav.next != null) {
    trav = trav.next;
trav.next = newnode;
```

$T(n) = O(n)$

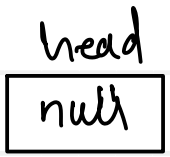
Singly linear Linked List - Delete first



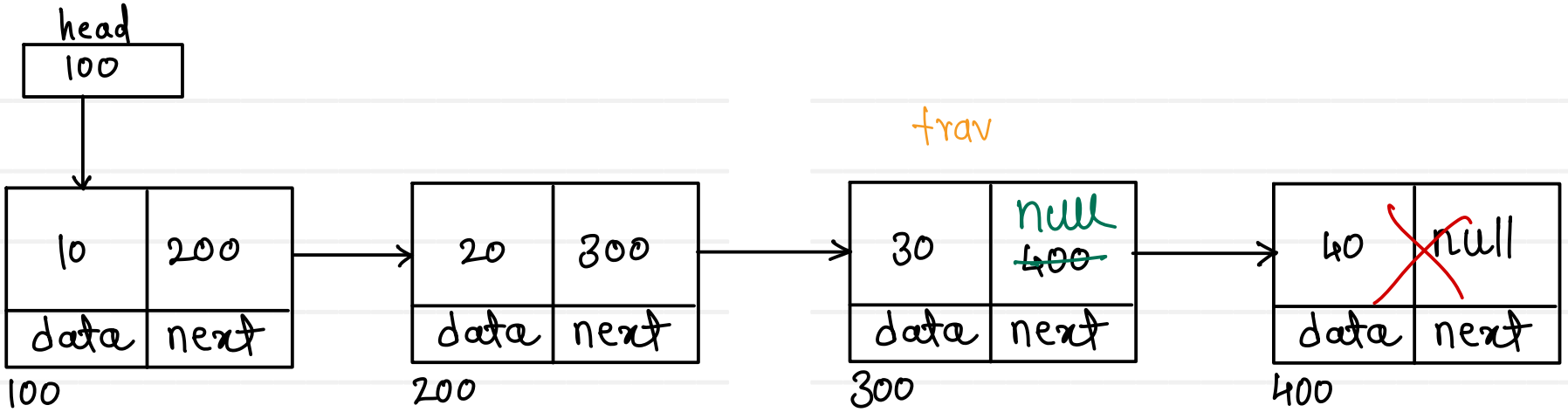


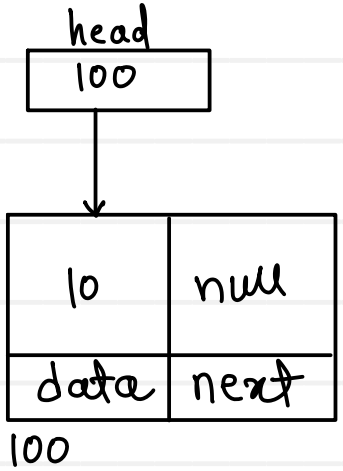
1. if list is empty
   do nothing
2. if list is not empty
   move head on second node

$T(n) = O(1)$



Singly linear Linked List - Delete last





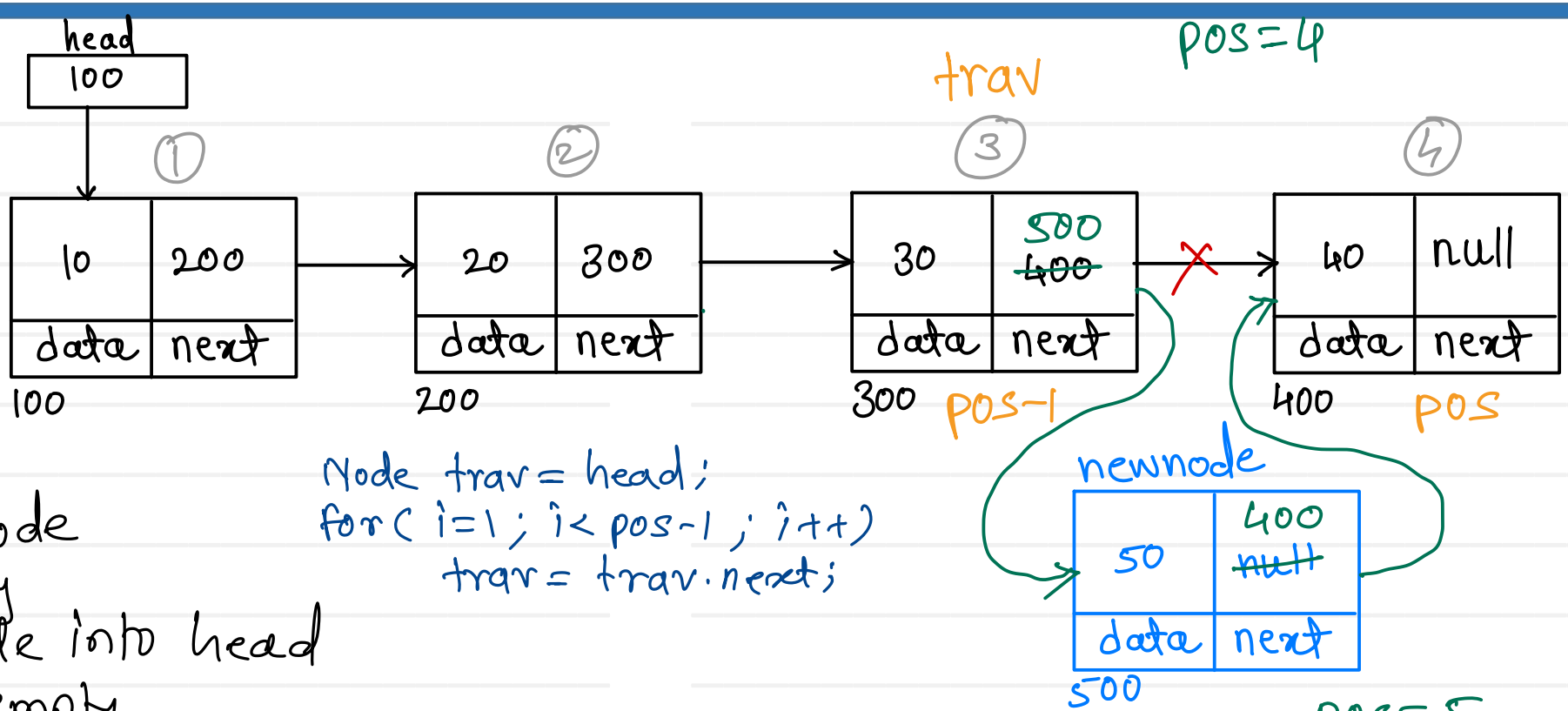
1. if list empty
   return
2. if list has single node
   head = null
3. if list has multiple nodes
   a. traverse till second last node
   b. add null into next of second last node

$T(n) = O(n)$

while(trav != null) → trav = null
while(trav.next != null) → trav = last node
while(trav.next.next != null) → trav = second last node

Singly linear Linked List - Add position

make before break




```
Node trav = head;
for( i=1 ; i< pos-1 ; i++)
    trav = trav.next;
```

1. create a newnode
if list is empty
    add newnode into head
if list is not empty
    2. traverse till pos-1 node
    3. add pos node into next of newnode
    4. add newnode into next of pos-1 node

$T(n) = O(n)$

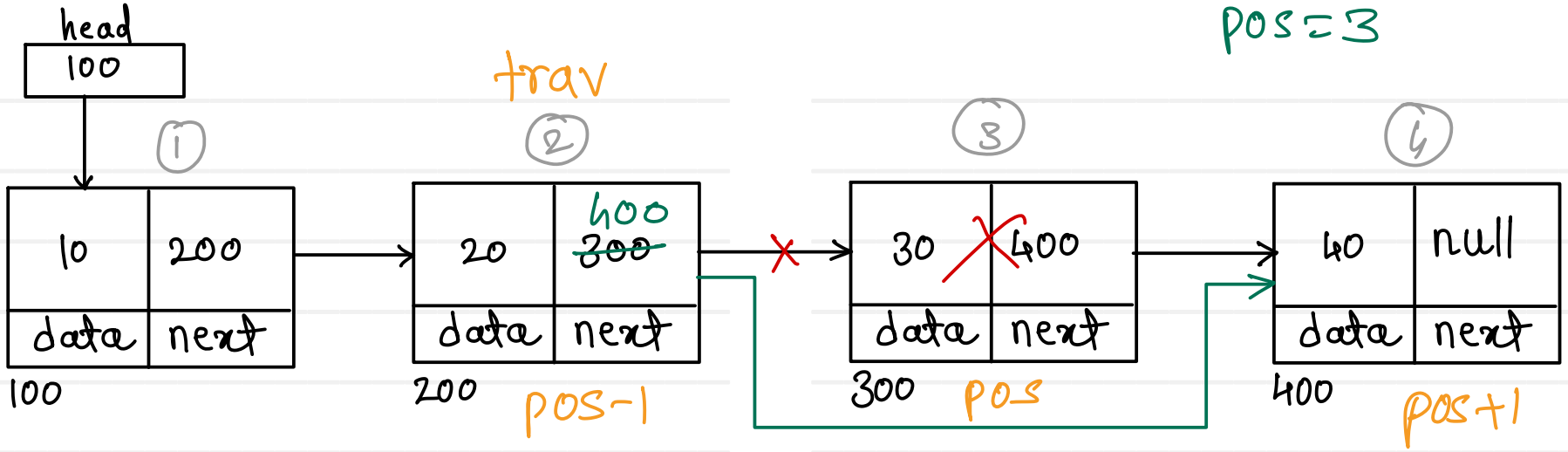
pos=4

trav	i	i<3
100	1	T
200	2	T
300	3	F

pos=5

trav	i	i<4
100	1	T
200	2	T
300	3	T
400	4	F

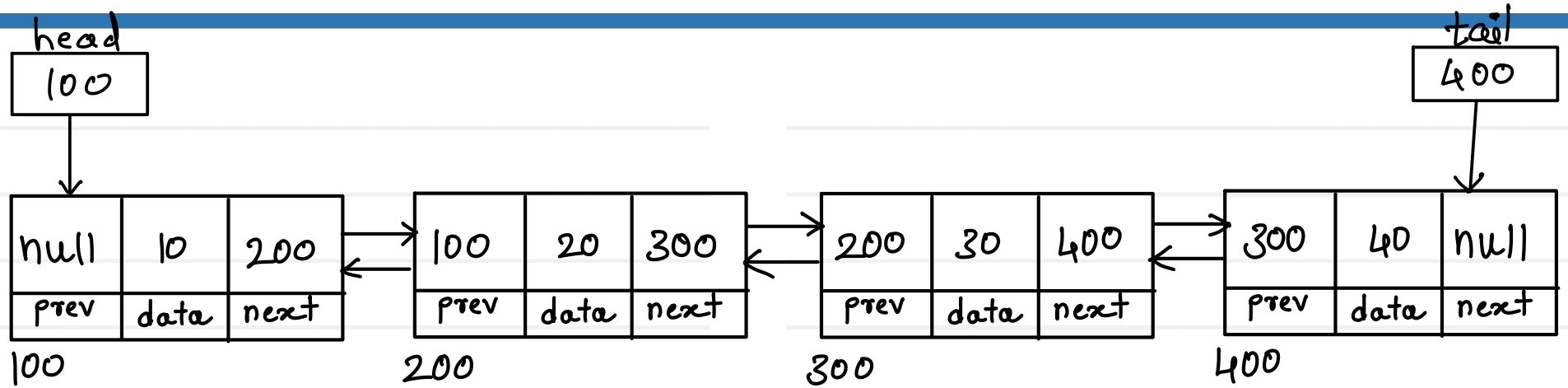
Singly linear Linked List - Delete position



1. if list is empty
   return
2. if list is not empty
   a. traverse till pos-1 node
   b. add pos+1 node into next of pos-1 node

$T(n) = O(n)$

Doubly Linear Linked List - Display



Forward traversal

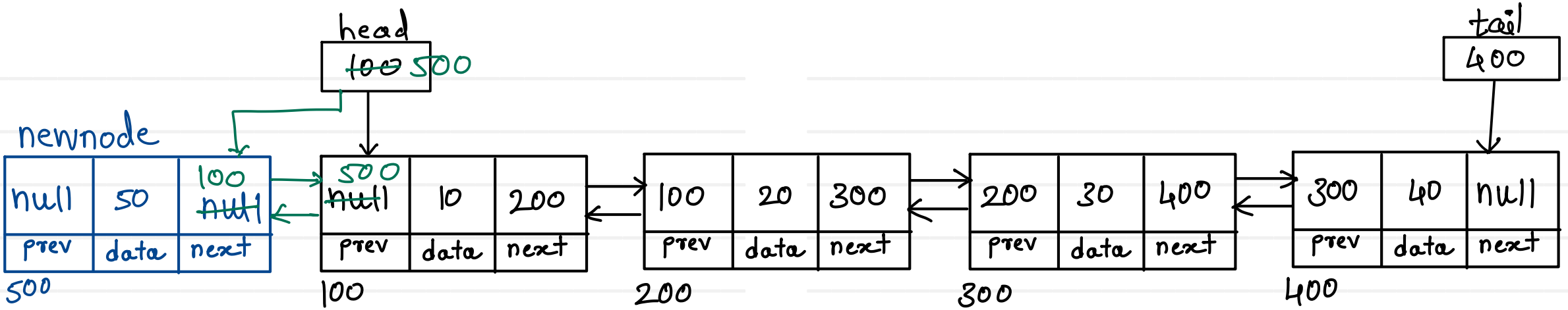
1. create trav & start at head
2. print current node data (trav->data)
3. go on next node (trav = trav->next)
4. repeat above two steps till last node

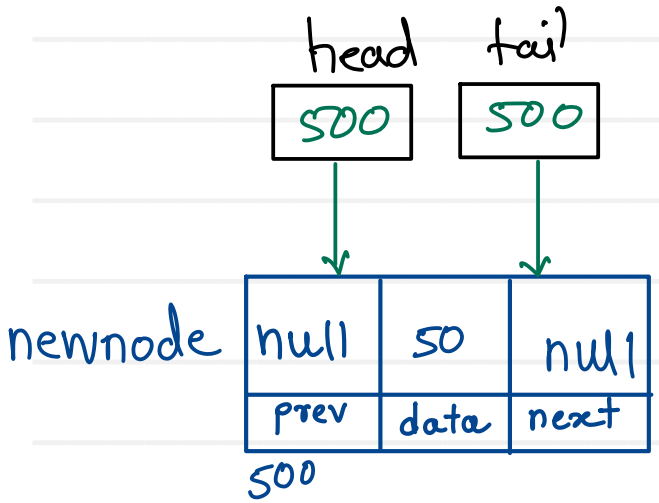
Backword traversal

1. create trav & start at tail
2. print current node data (trav->data)
3. go on prev node (trav = trav->prev)
4. repeat above two steps till first node

$$T(n) = O(n)$$

Doubly Linear Linked List - Add first

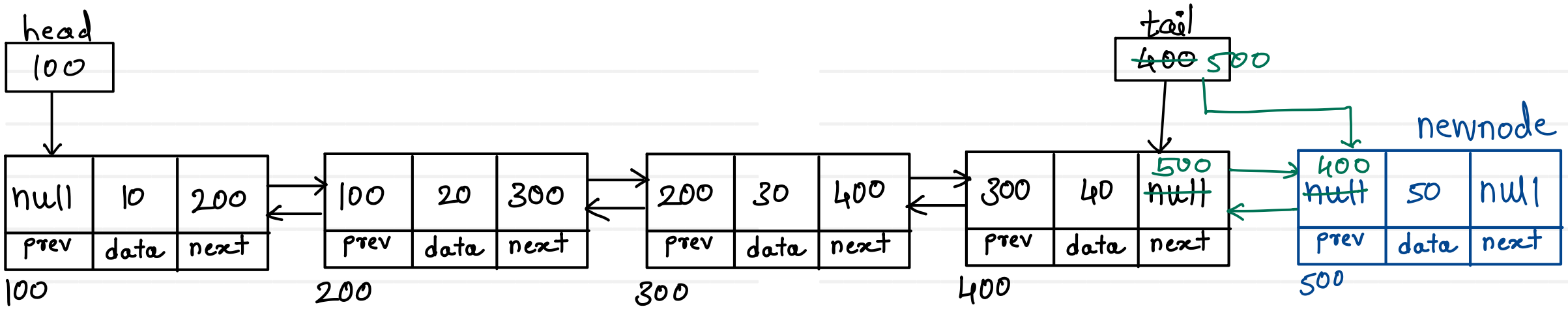


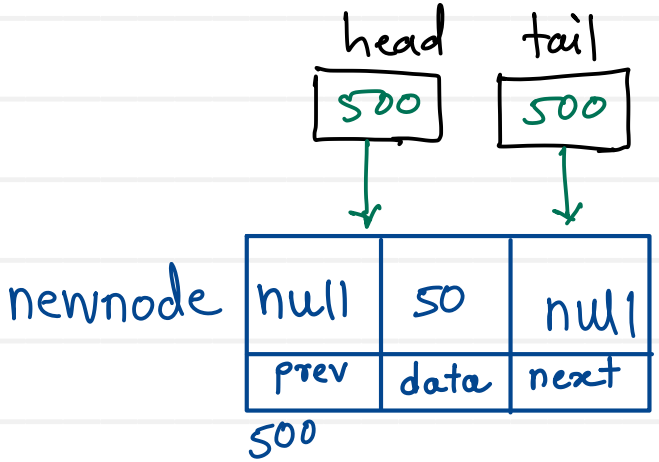


1. create node
2. if list is empty
   add newnode into head & tail
3. if list is not empty
   a. add first node into next of newnode
   b. add newnode into prev of first node
   c. move head on newnode

$T(n) = O(1)$

Doubly Linear Linked List - Add last





1. create node
2. if list is empty
   add newnode into head & tail
3. if list is not empty
   a. add last node into prev of newnode
   b. add newnode into next of last node
   c. move tail on newnode

$T(n) = O(1)$

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com