Sunbeam Institute of Information Technology

Pune and Karad

Module – Data Structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

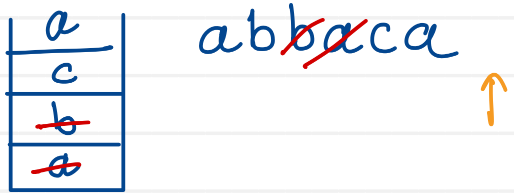
Remove all adjacent duplicates in string

You are given a string s consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them.

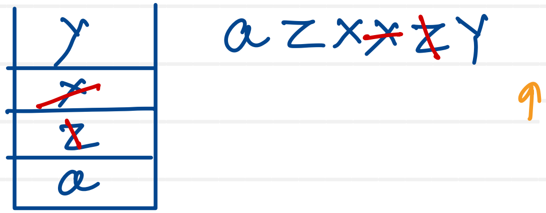
We repeatedly make duplicate removals on s until we no longer can.

Return the final string after all such duplicate removals have been made. It can be proven that the answer is unique.

Example 1:
Input: s = "abbaca"
Output: "ca"



Example 2:
Input: s = "azxxzy"
Output: "ay"



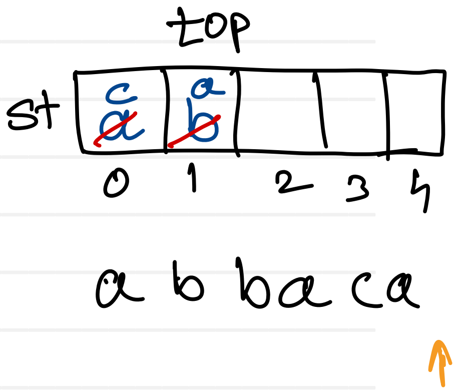
```
String removeDuplicates(String s) {
    int n = s.length();
    char []st = new char[n];
    int top = -1;

    for(int i=0; i<n; i++){
        char ch = s.charAt(i);
        if(top > -1 && ch == st[top])
            top--;
        else {
            top++;
            st[top] = ch;
        }
    }
    return new String(st, 0, top+1);
}
```

Auxillary space

array, starting index, length of string

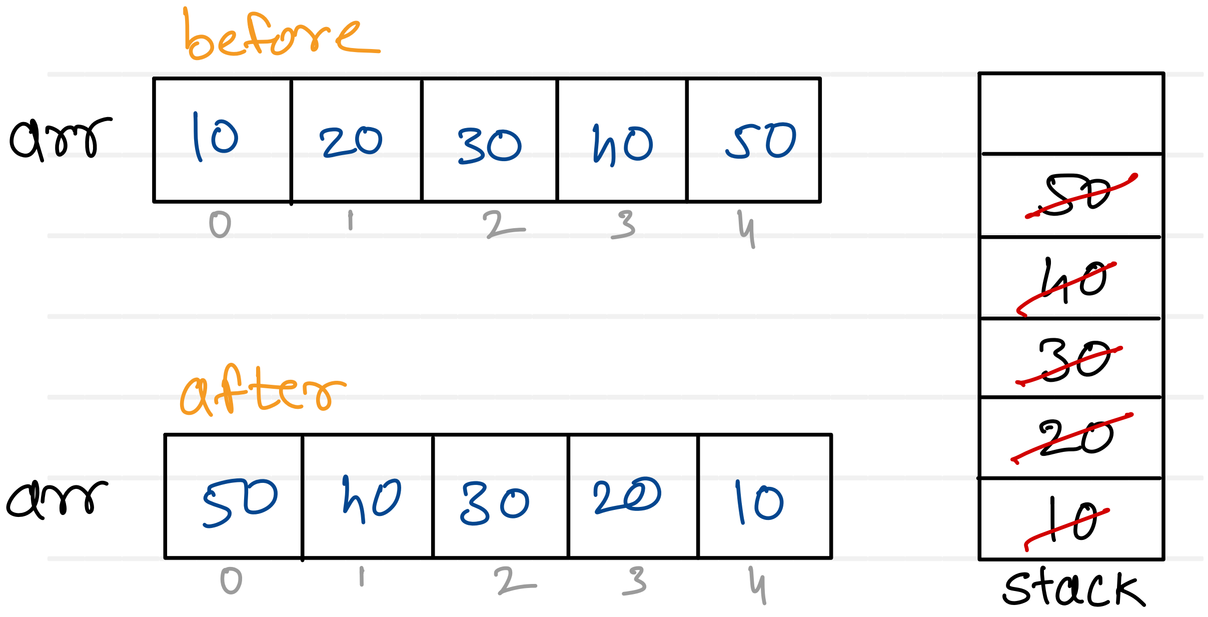
$T(n) = O(n)$

$S(n) = O(n)$



Reverse array, string or linked list using stack/queue

int arr[] = {10, 20, 30, 40, 50}



before

arr	10	20	30	40	50
	0	1	2	3	4

after

arr	50	40	30	20	10
	0	1	2	3	4

```
void reverseArray(int arr[]) {
    stack<Integer> st = new stack<>();
    for(int i = 0; i<arr.length; i++)
        st.push(arr[i]);
    for(int i=0; i<arr.length; i++)
        arr[i] = st.pop();
}
```

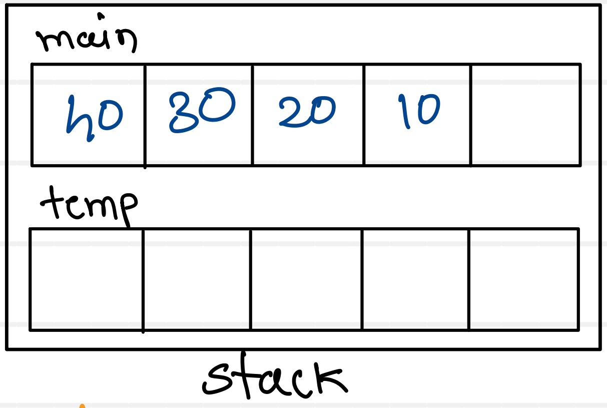
push → n

pop → n

total time = 2n $T(n) = O(n)$

stack → Auxillary space $S(n) = O(n)$

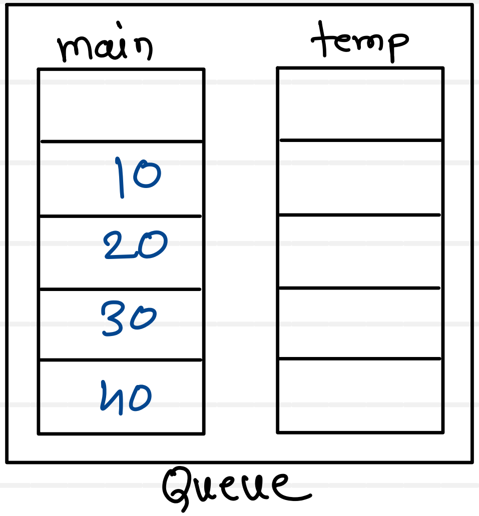
Create stack using queue



Push order : 10,20,30,40

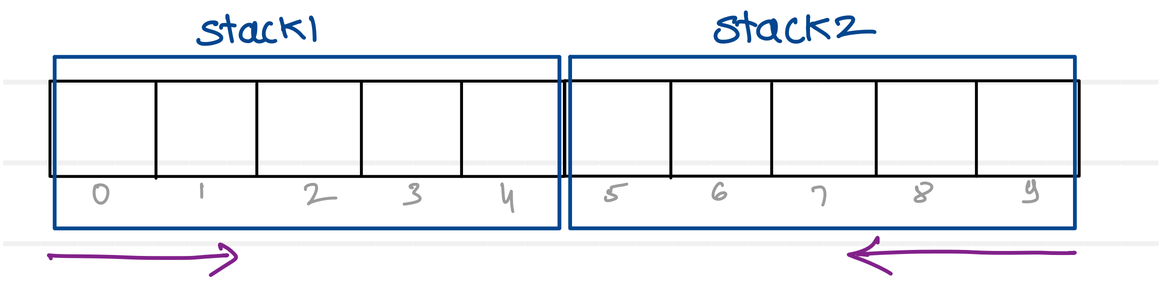
```
push :   while( ! main.isEmpty())
             temp.push(main.pop());
O(n) →   main.push(value);
         while( ! temp.isEmpty())
             main.push(temp.pop());
O(1) →  pop :   main.pop()
        peek:   main.peek()
```

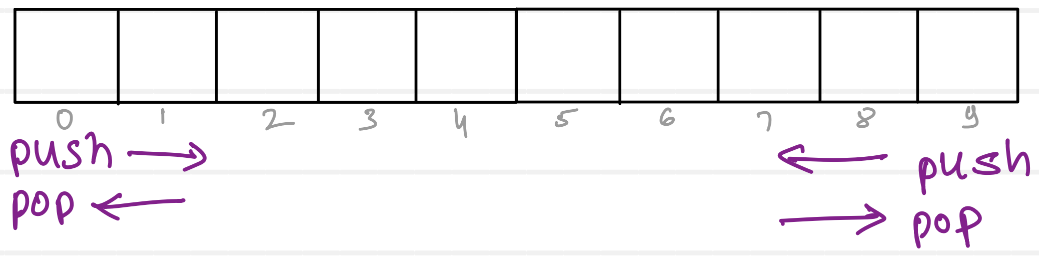
Create queue using stack



Push order : 10 ,20, 30, 40

How to Implement two stacks using array efficiently?



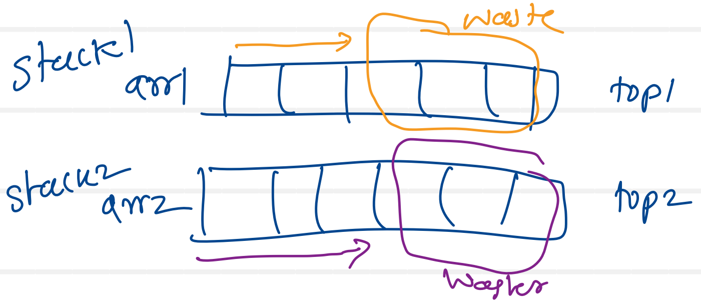


arr

	a	b	c	d	e	f	4	3	2	1	
-1	0	1	2	3	4	5	6	7	8	9	10

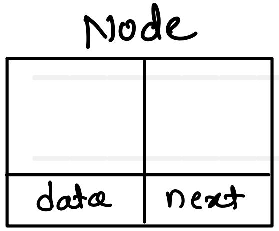
top1 top2

	stack1	stack2
push	arr[++top1] = val	arr[--top2] =val
pop	--top1	++top2
peek	arr[top1]	arr[top2]
Empty	top1 == -1	top2 == size
Full	top1 == top2-1	

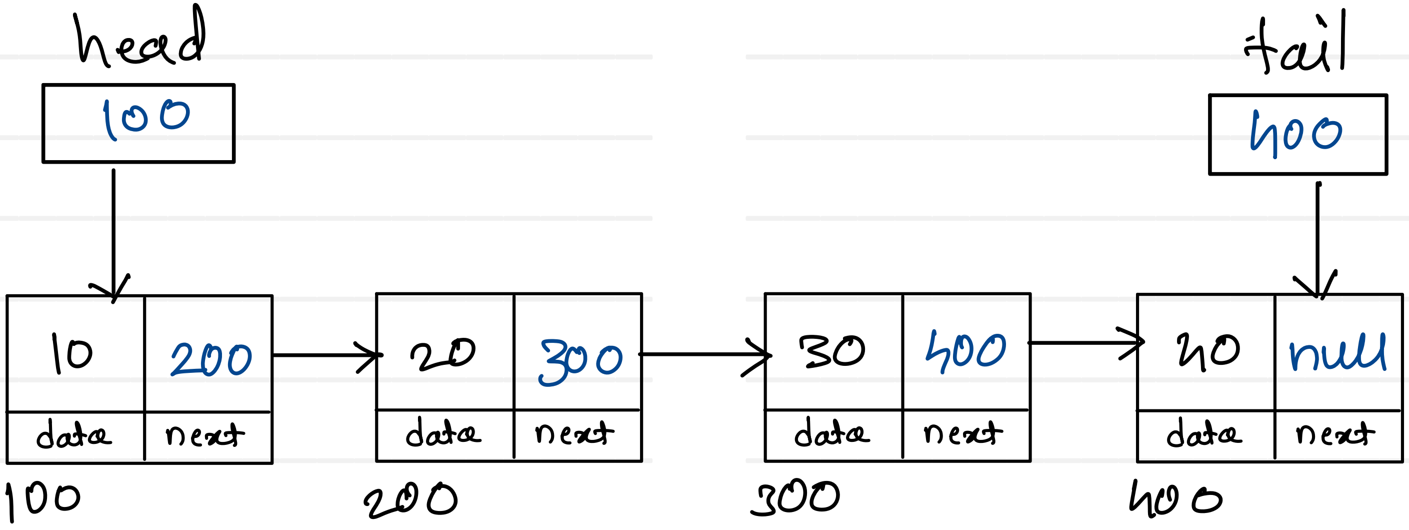


Linked List

- linked list is a linear data structure
- link/address of next data is kept with current data.
- every element of linked list has two part : data & address/link & which is referred as " node "



- Address of first node is kept into "head" reference.
- Address of last node is kept into "tail" reference. (optional)



Linked List

Operations

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

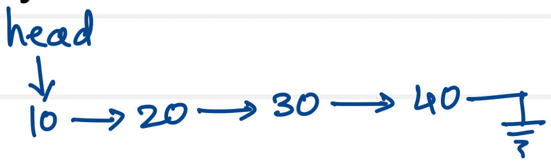
1. Delete first
2. Delete last
3. Delete position

1. Display (traverse) (forward/ backward)

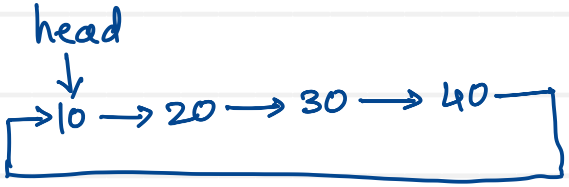
1. Search
2. Sort
3. Reverse

Types

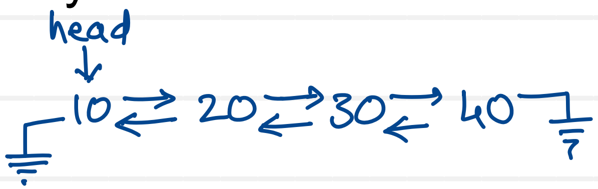
1. Singly linear linked list



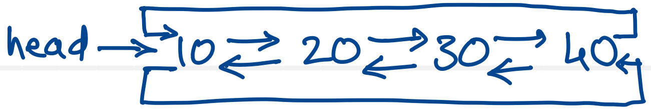
2. Singly circular linked list



3. Doubly linear linked list



4. Doubly circular linked list



Linked List

Node :

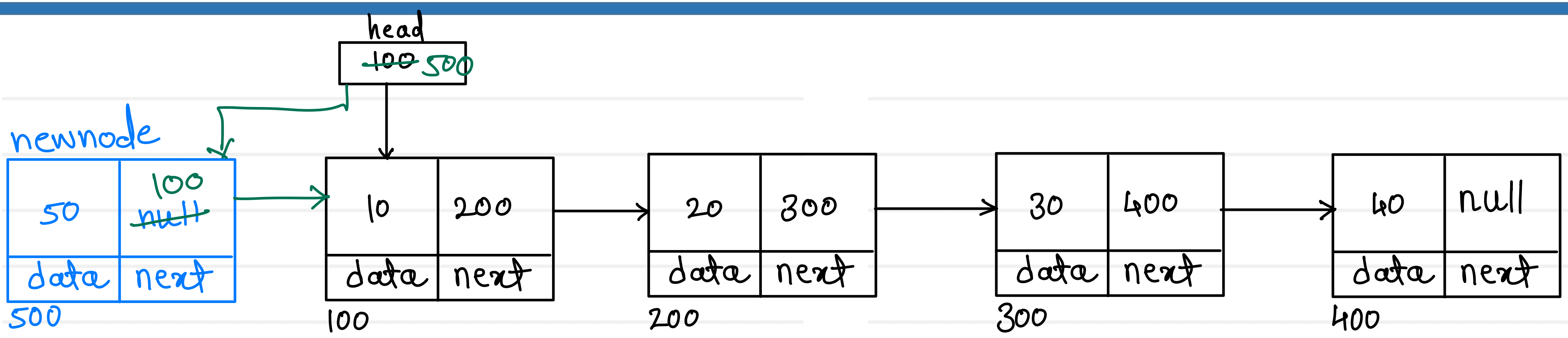
data - int, char, double, string , class

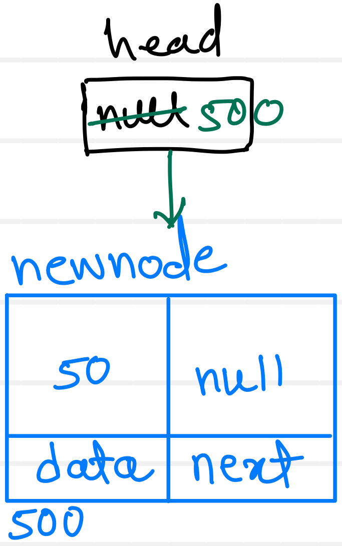
next - reference

```
class Node {
    int data;
    Node next;  <- self referential class
}
```

```
class LinkedList {
    static class Node {
        int data;
        Node next;
    }
    Node head;
    Node tail;
    int count;
    public LinkedList() {...}
    public add() {...}
    public delete() { ... }
    public display() {...-}
    public deleteALL () {...}
}
```

Singly linear Linked List - Add first



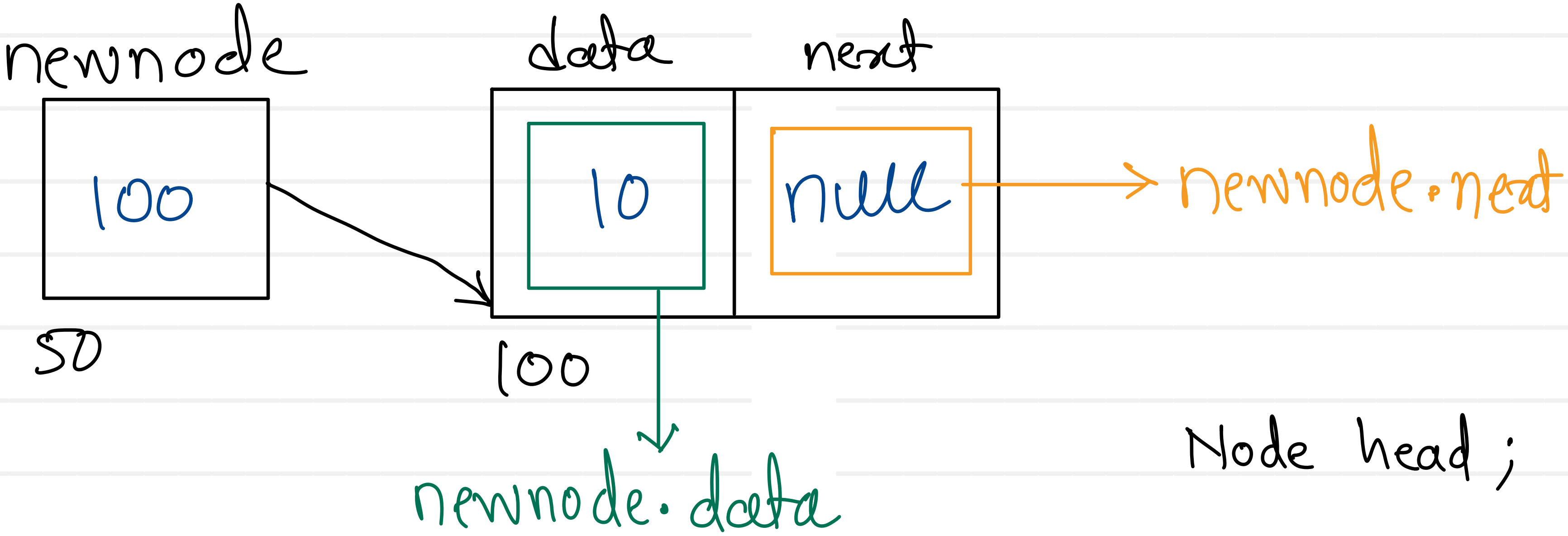


1. Create node with given data
2. add first node into next of newnode
3. move head on newnode

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

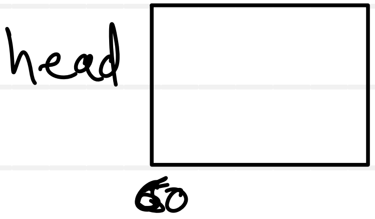
$\boxed{T(n) = O(1)}$

```
Node newnode = new Node(10);
```

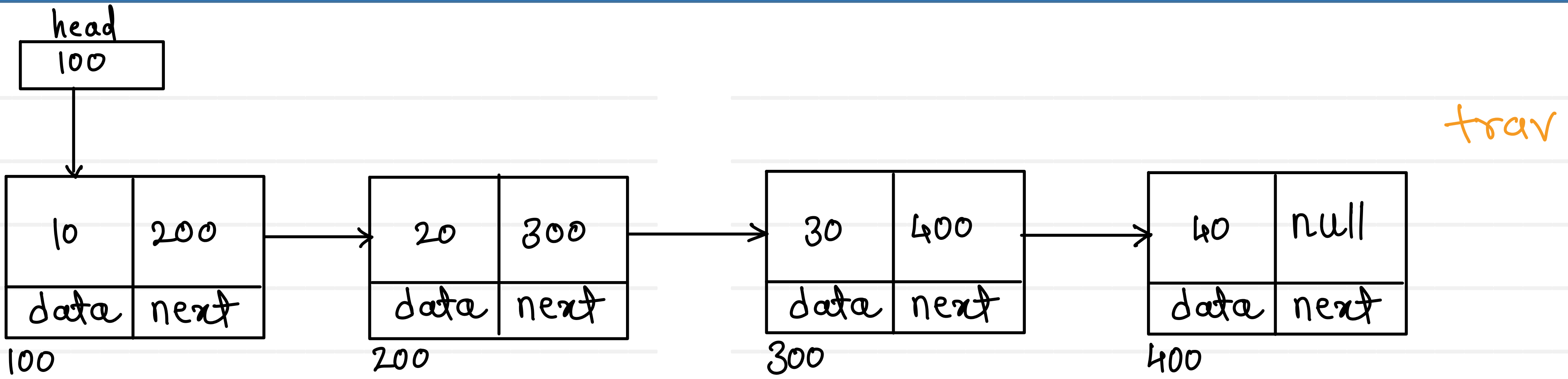


```
newnode.data = 2.    ← writing
 2. = newnode.data   ← reading
```

```
Node head;
```



Singly linear Linked List - Display

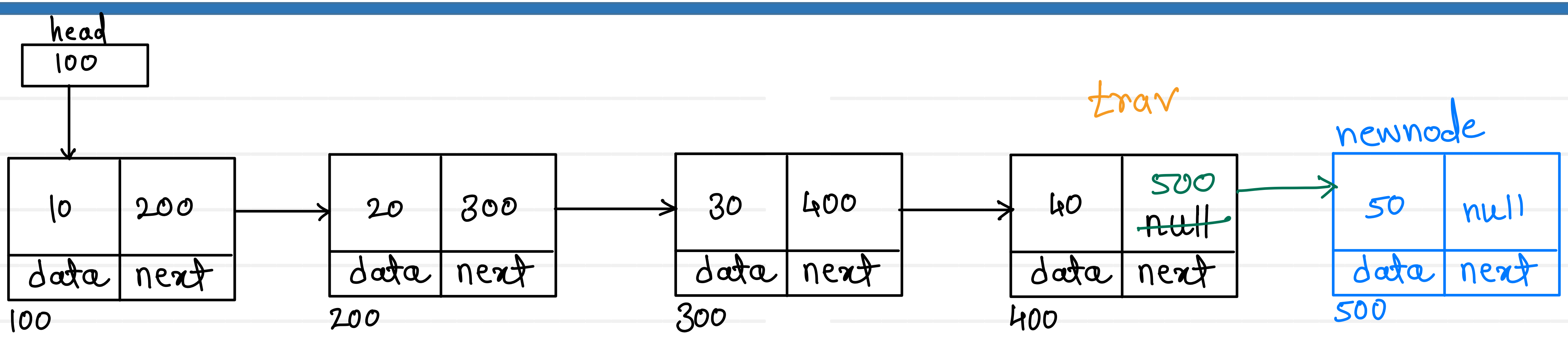


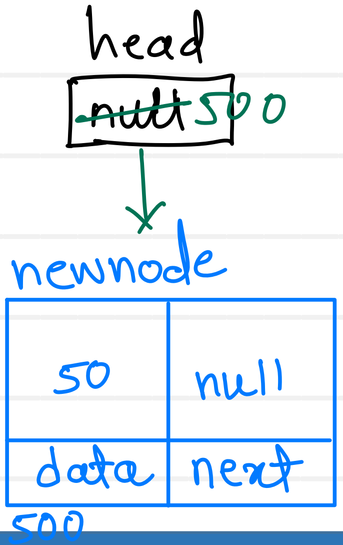
trav	trav.data	trav.next
100	10	200
200	20	300
300	30	400
400	40	null
null		

1. create trav & start at head
2. print current node data(trav.data)
3. go on next node (trav.next)
4. repeat above 2 steps for each node

$$\boxed{T(n) = O(n)}$$

Singly linear Linked List - Add last



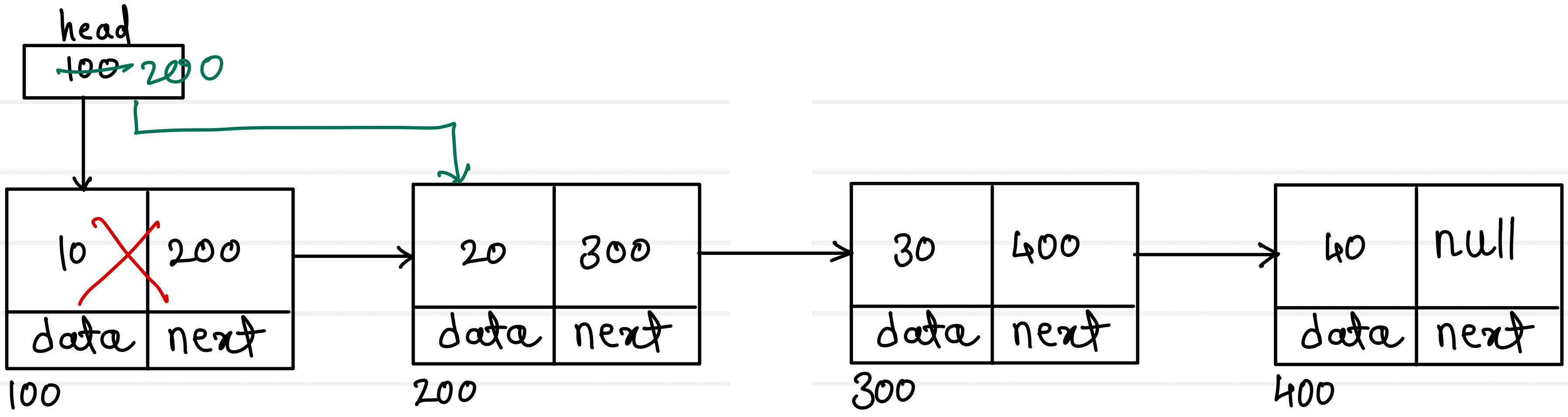


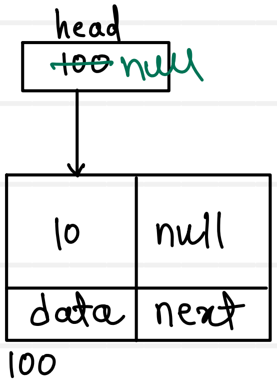
1. create node with value
2. if list is empty
    add newnode into head
3. if list is not empty
    a. traverse till last node
    b. add newnode into next of last node

```
while(trav.next != null)
    trav = trav.next;
```

$$\boxed{T(n) = O(n)}$$

Singly linear Linked List - Delete first





head = head.next;

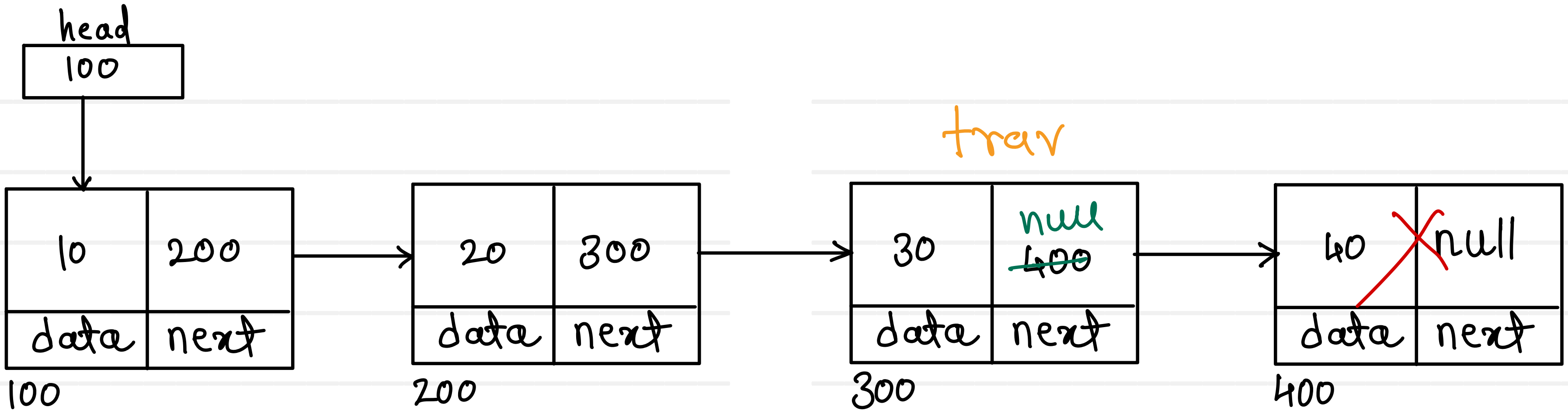
1. if list is empty
    return ;

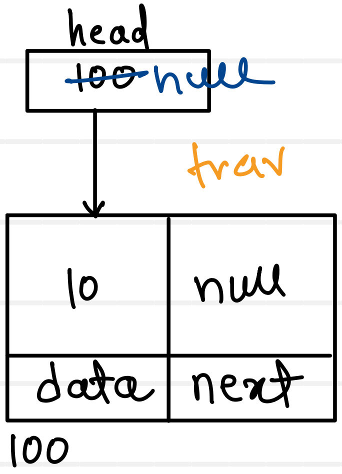
2. if list is not empty
    a. move head on second node

$$\boxed{T(n) = O(1)}$$

head = head.next; ✗

Singly linear Linked List - Delete last





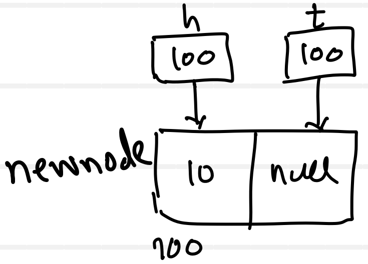
```
while(trav.next.next != null)
    trav = trav.next;
```

$$\boxed{T(n) = O(n)}$$

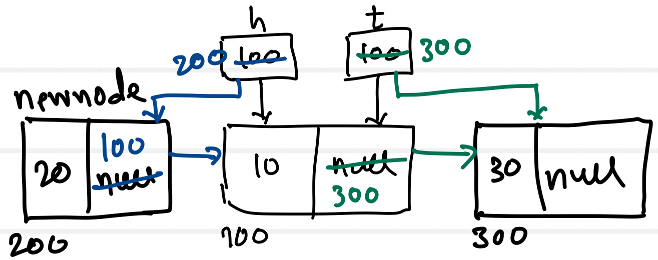
1. if list is empty
   return
2. if list has single node
   head = null;
3. if list has multiple nodes
   a. traverse till second last node
   b. add null into next of second last node

Singly Linear Linked List

	only head	head & tail
Add first	O(1)	O(1)
Add Last	O(n)	O(1)
Delete first	O(1)	O(1)
Delete last	O(n)	O(n)
Display	O(n)	O(n)



```
if(head==null)
    head=tail=newnode
```



Add first
newnode.next = head
head = newnode

Add last
tail.next = newnode
tail = newnode

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com