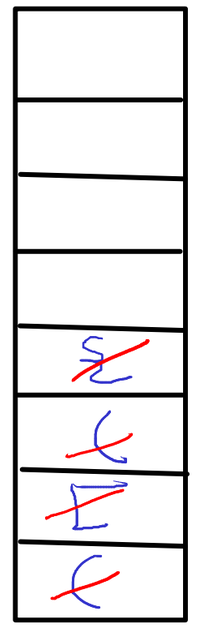
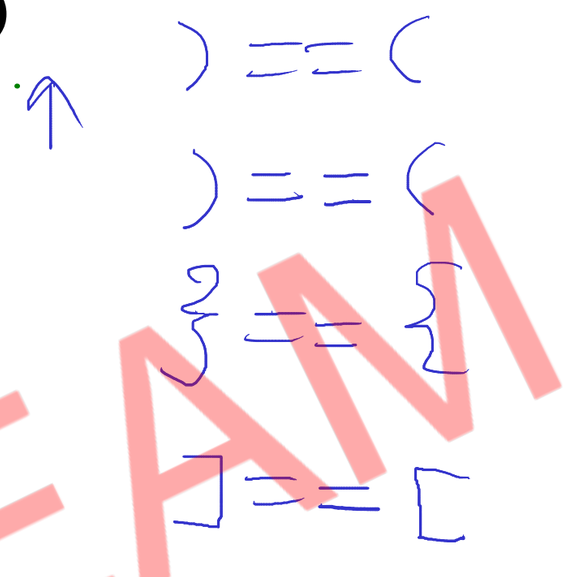
Parenthesis Balancing

5 + ([9 - 4] * (8 - { 6 / 2 }))

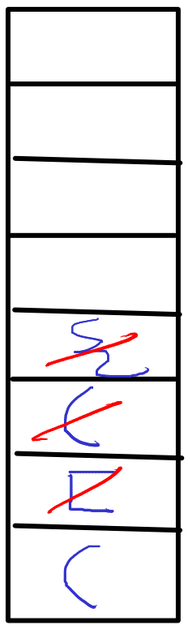
Stack

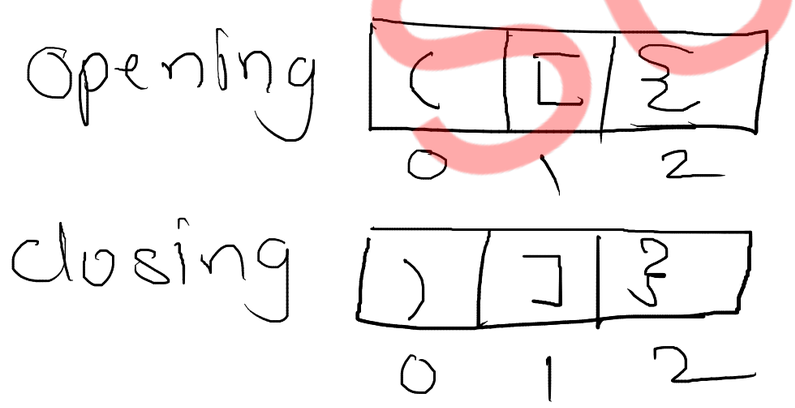


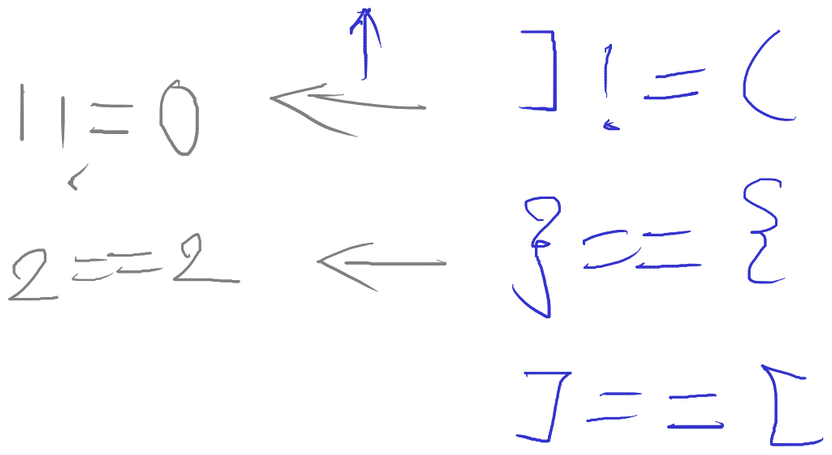


5 + ([9 - 4] * (8 - { 6 / 2 }])

Stack

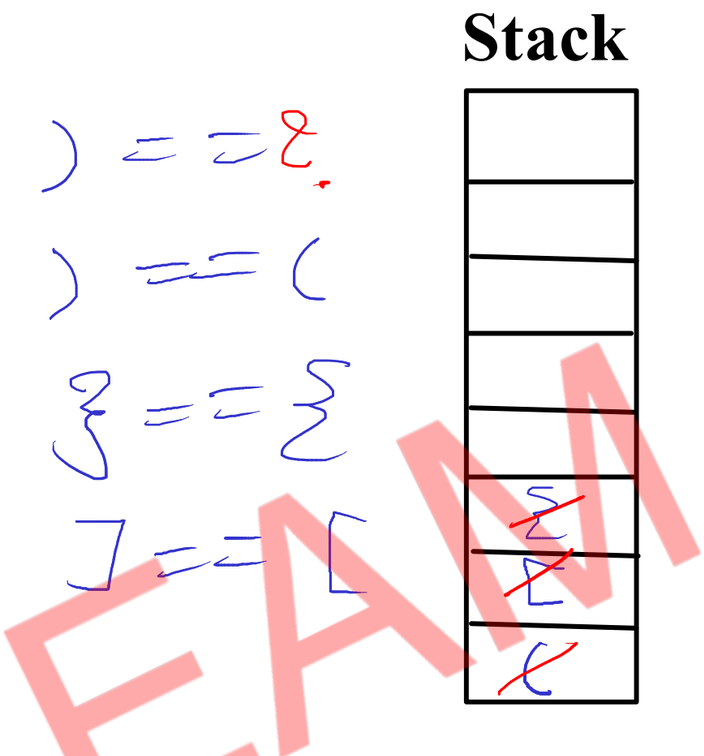




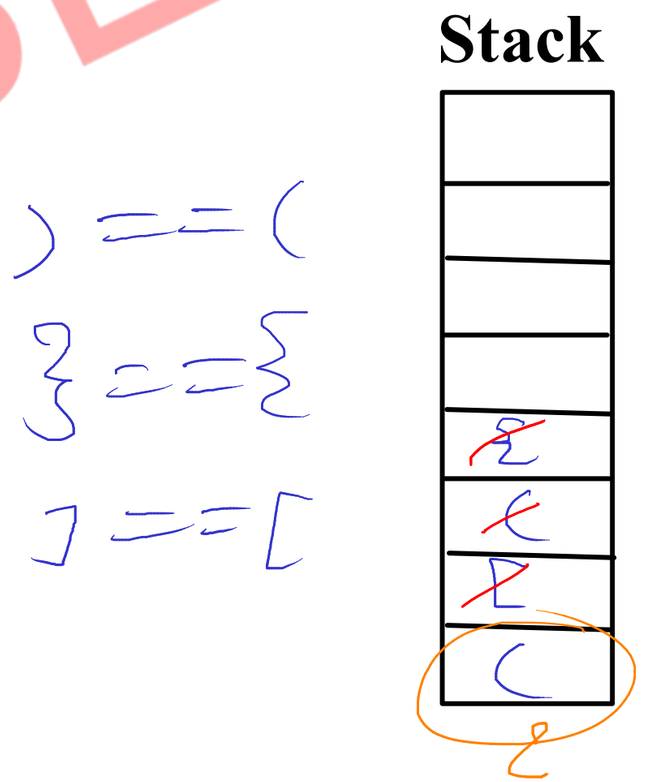


String -> indexOf( ) -> returns index of char
returns -1 if not found

5+([9-4]*8-{6/2}))



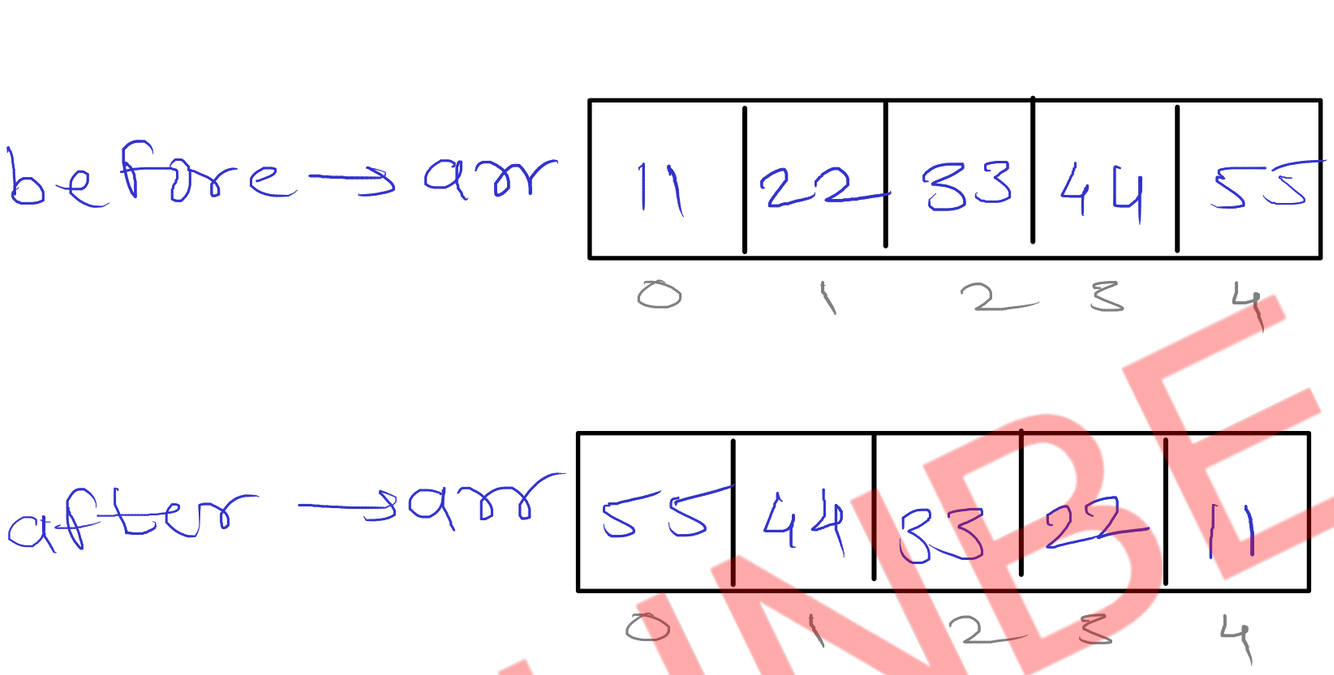
5+([9-4]*(8-{6/2})

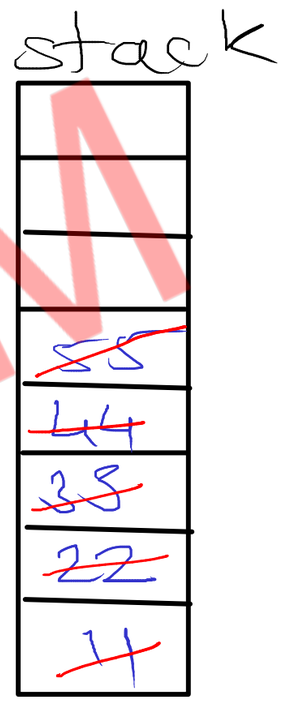


Stack / Queue - Competitive Programming

Reverse array, string or linked list using stack/queue.

int arr = {11, 22, 33, 44, 55};






```
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();
}
```

processing variable

→ n

+= 2n

→ n

T(n) = O(n)

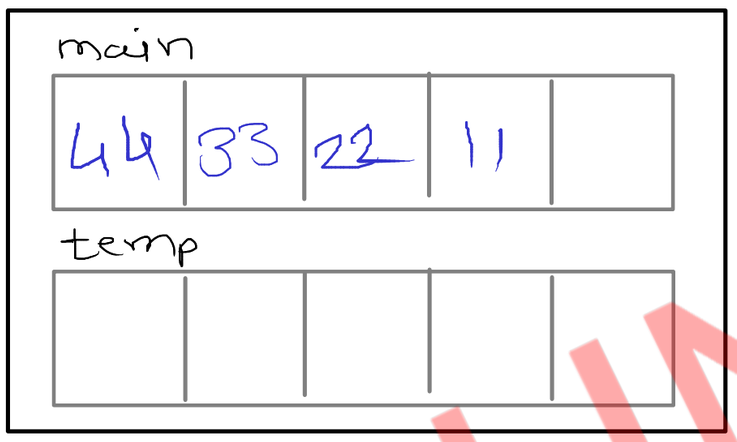
AS(n) = O(n)

Stack / Queue - Competitive Programming

Create stack using queue.

Hint - push op will not be performed in O(1)
- can use multiple queues to creat stack

Stack



Push Order: 11, 22, 33, 44

```
push : while( !main.isEmpty() )
           temp.push(main.pop())
       main.push(value)
       while( !temp.isEmpty() )
           main.push(temp.pop())

O(n)

pop : main.pop()
peek; main.peek()

O(1)
```

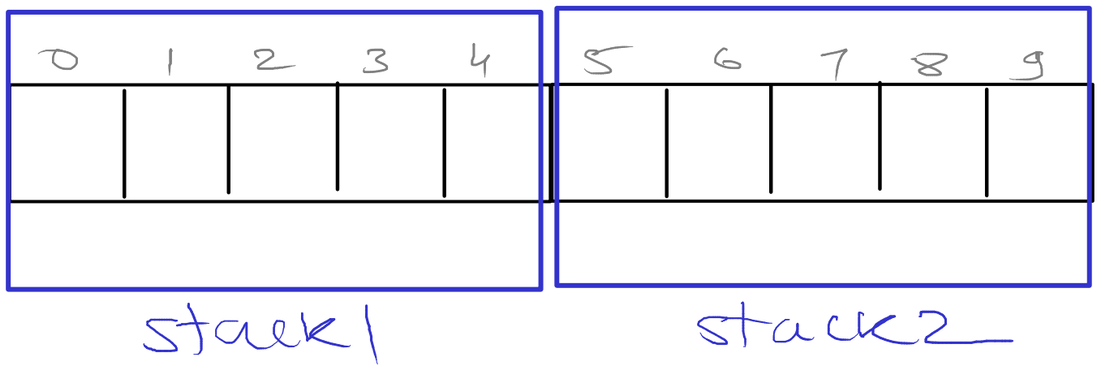
Create queue using stack.

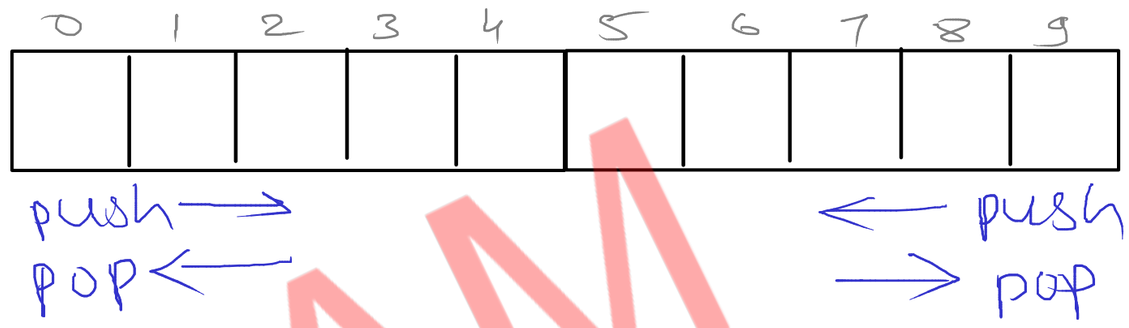
Queue

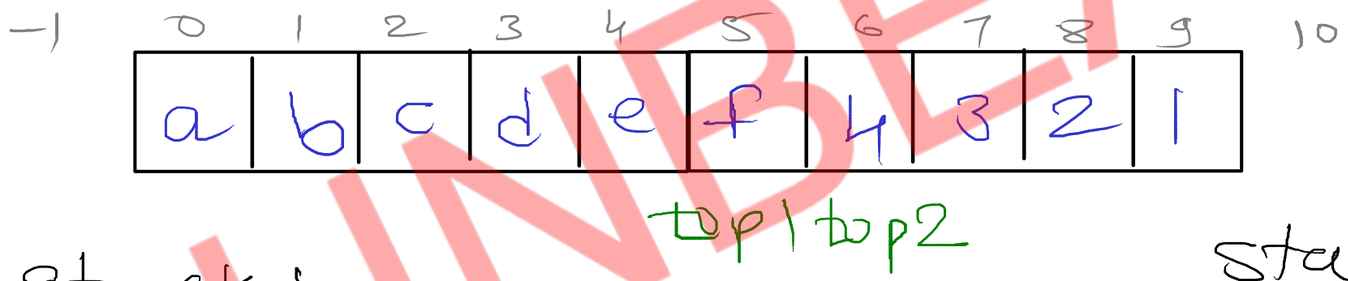
Push Order: 11, 22, 33, 44

Stack / Queue - Competitive Programming

How to implement two stacks in single array efficiently?







stack1

push : arr[++top1] = value

pop : top1--

peek : arr[top1]

isEmpty : top1 == -1

isFull : top1+1 == top2

stack2

push : arr[--top2] = value

pop : top2++

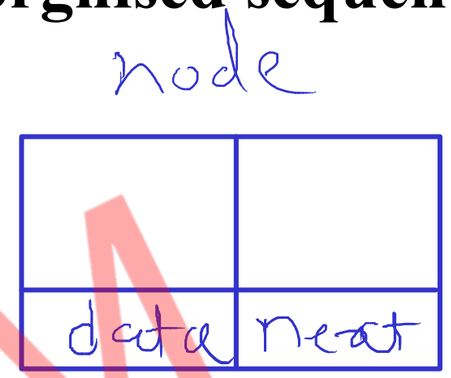
peek : arr[top2]

isEmpty : top2 == size

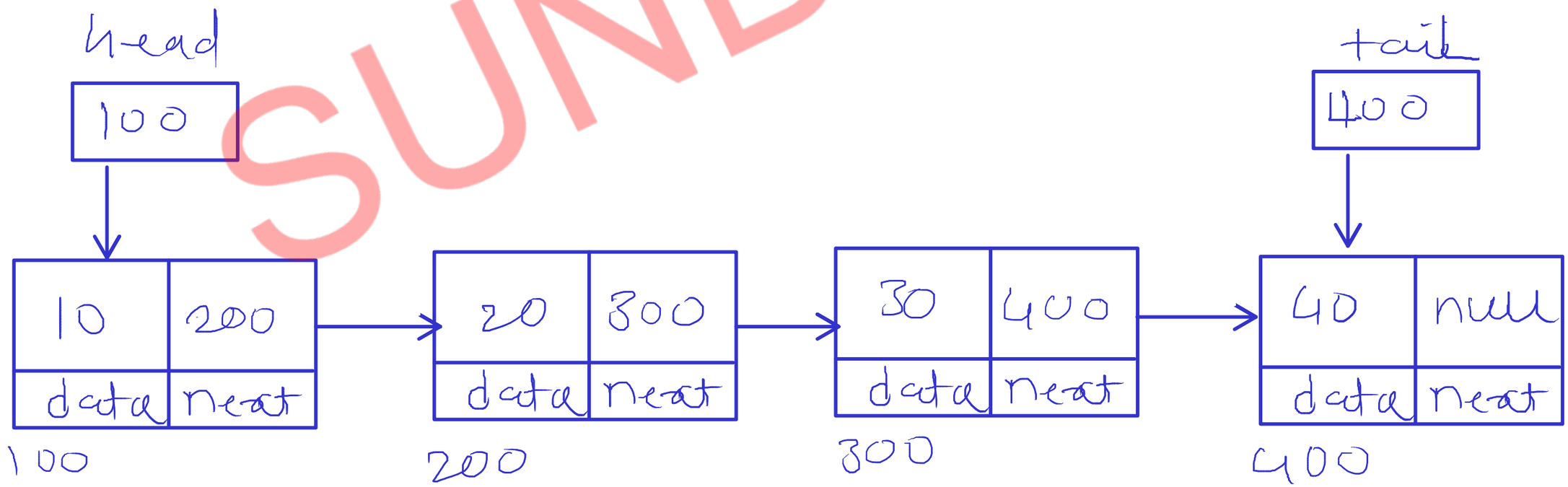
isFull : top1+1 == top2

Linked List

- Linked list is a linear data structure in which data is orgnised sequentially
- address of next data is kept with current data
- Every element of linked list is known as "node"
- Node consists of two parts
    - data : actual data of a node
    - link/next : address of next node/data



- address of first node is kept into one of the referance (head)
- address of last node is kept into one of the refeance (tail) --> optional



Linked List

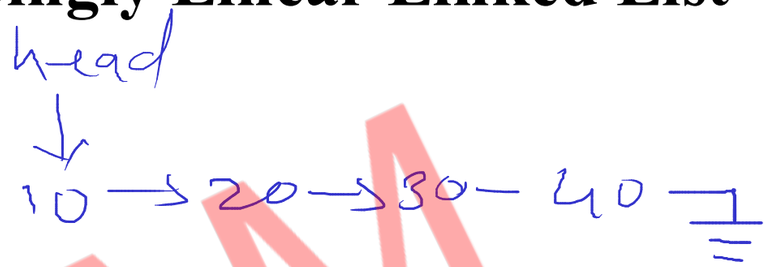
Operations:

1. Add - First, Last, Insert(pos)
2. Delete - First, Last, Remove(pos)
3. Traverse (Display)

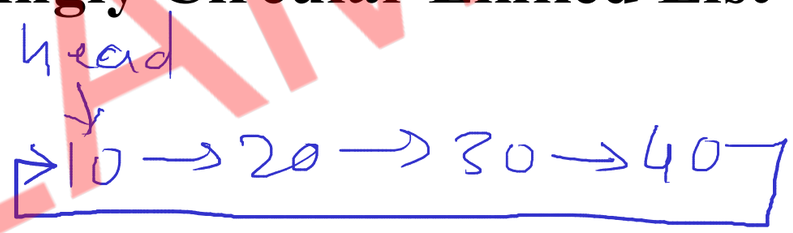
4. Search
5. Sort
6. Reverse
7. Find mid

Types:

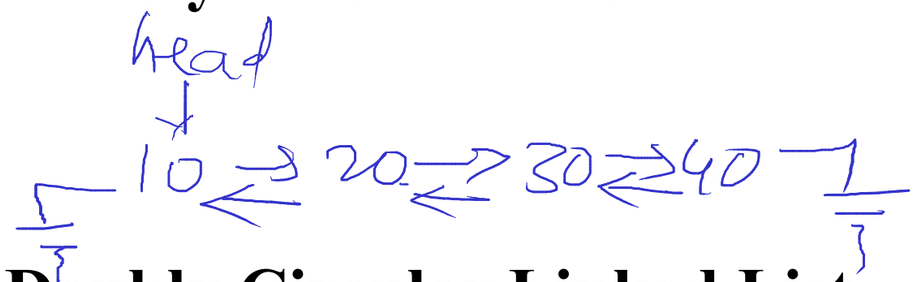
1. Singly Linear Linked List



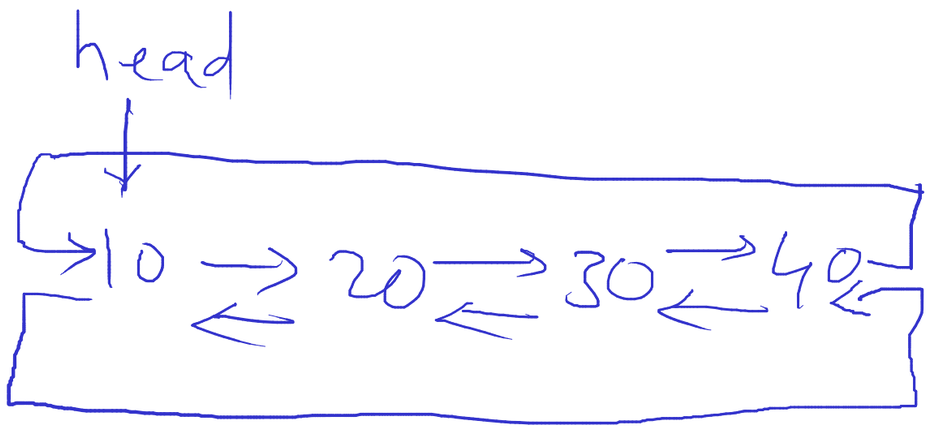
2. Singly Circular Linked List



3. Doubly Linear Linked List



4. Doubly Circular Linked List



Implentation

Node:

data - primitive, user defined data type

next - referance/pointer of next node

```
class Node{
        private int data;
        private Node next;
}
```

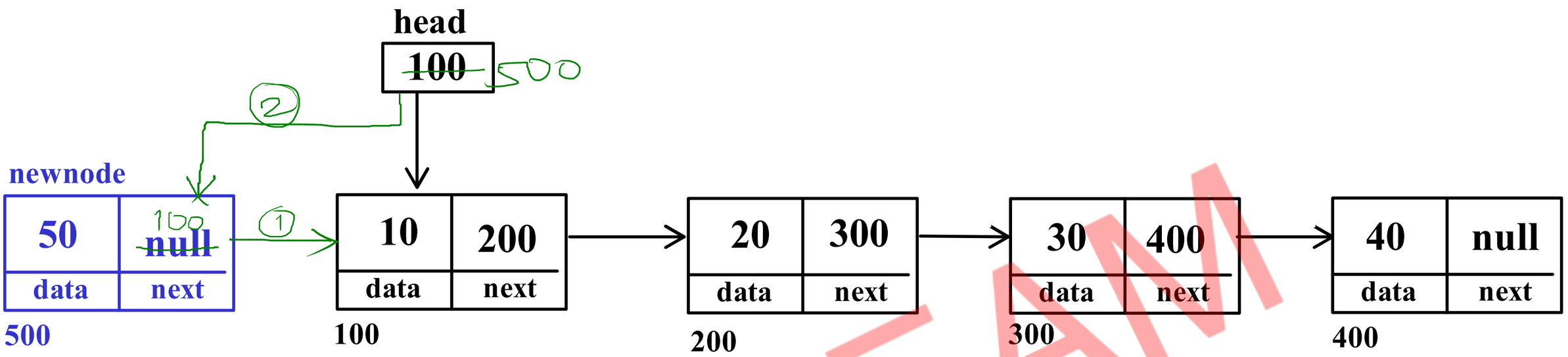
self referential class

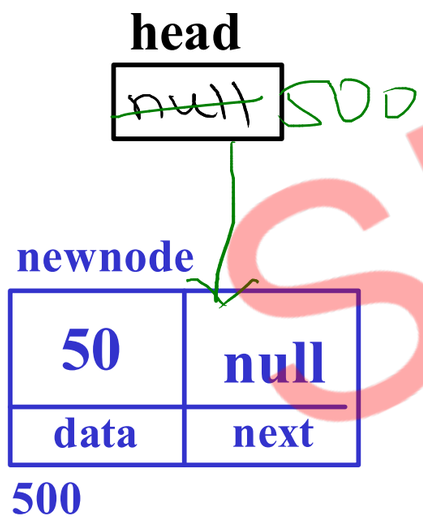
① non static fields of outer class are not accessible in inner class

② there is no dependency of outer class to create object of inner class

```
 class List{
        static class Node{
                private int data;
                private Node next;
        }
        private Node head;
        private Node tail;
        private int count;
        public List(){ }
        public Add() {      }
        public Delete() {   }
        public Display() {  )
        public DeleteAll() {      }
}
```

Singly Linear Linked List - Add First

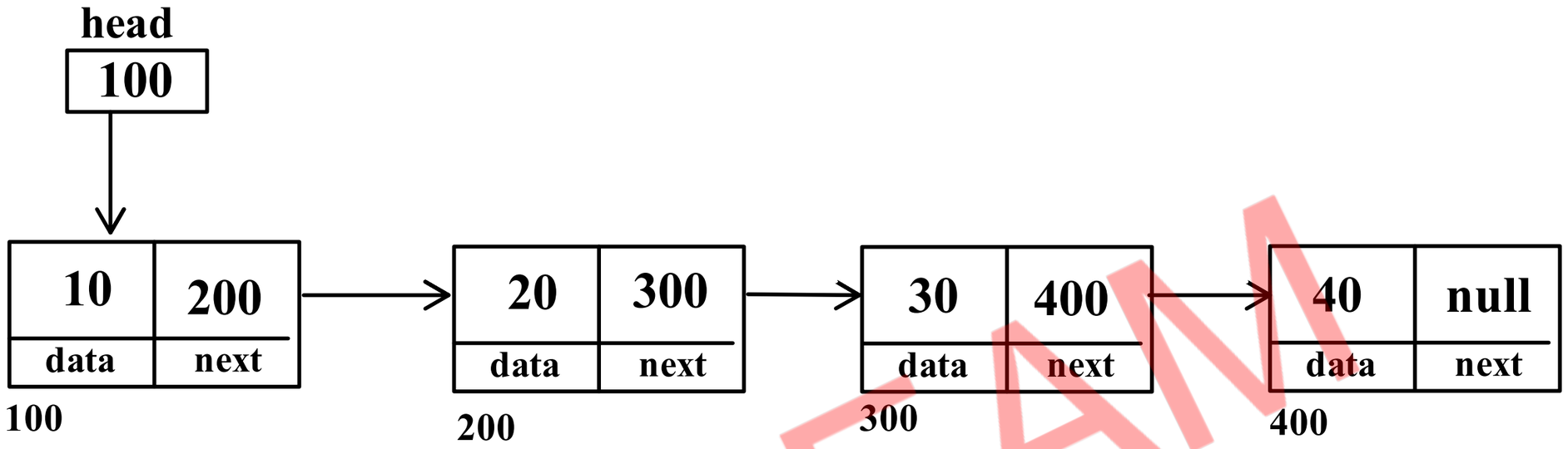





```
//1. create newnode with given data/value
//2. add first node into next of newnode
//3. move head on newnode
```

T(n) = O(1)

Singly Linear Linked List - Display (Traverse)



trav

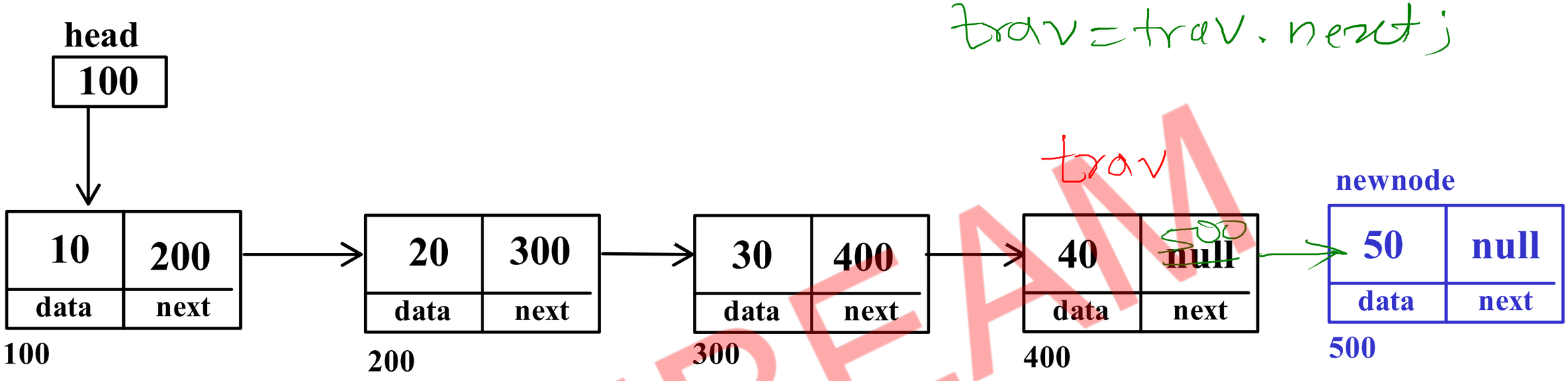
```
//1. create trav referance and start at first node
//2. print/visit current node (trav.data)
//3. go on next node (trav.next)
//4. repeat step 2 and 3 till last node
```

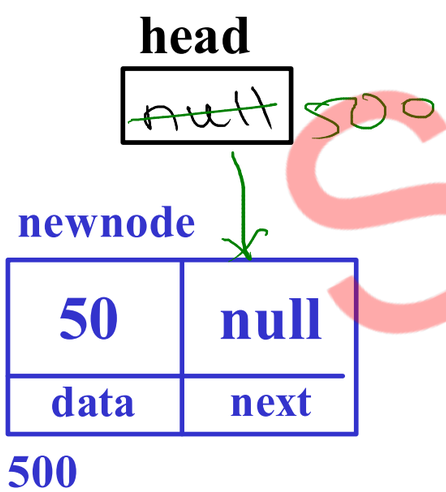
T(n) = O(n)

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

Singly Linear Linked List - Add Last

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






```
//1. create newnode with given data
//2. if list is empty
      // add newnode into head itself
//3. if list is not empty
      //a. traverse till last node
      //b. add newnode into next of last
```

T(n) = O(n)