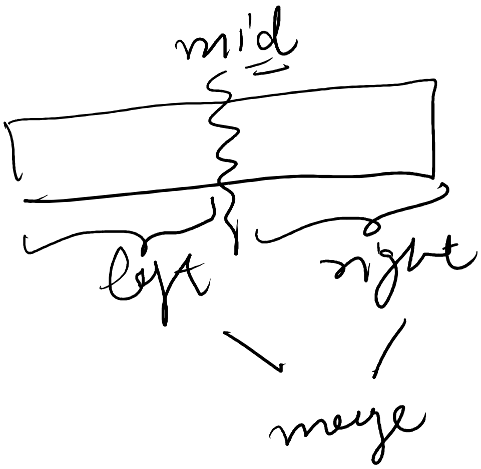
2.13 - Linked List 4 & 5

Saturday, July 5, 2025 10:41 AM



1 find mid

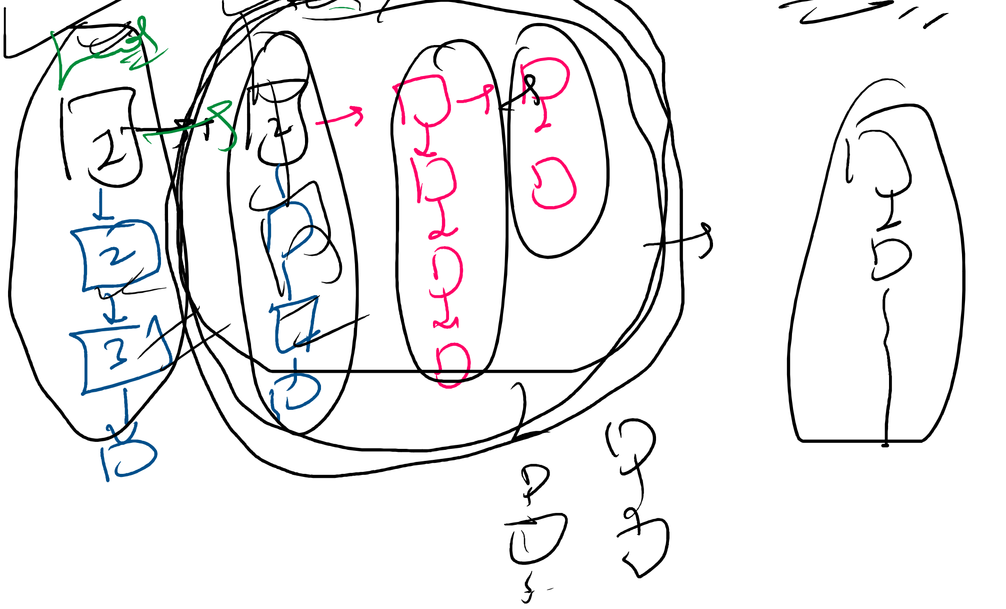
2. re left

3 rec right

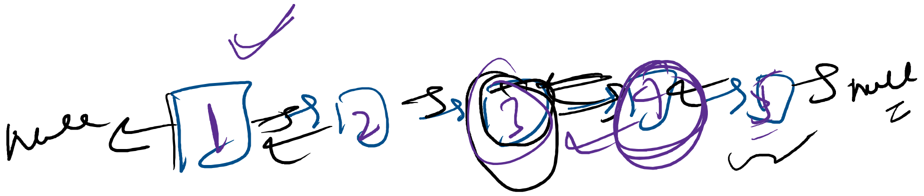
4. merge

[Recursion]

Flatten a LL



Doubly linked List

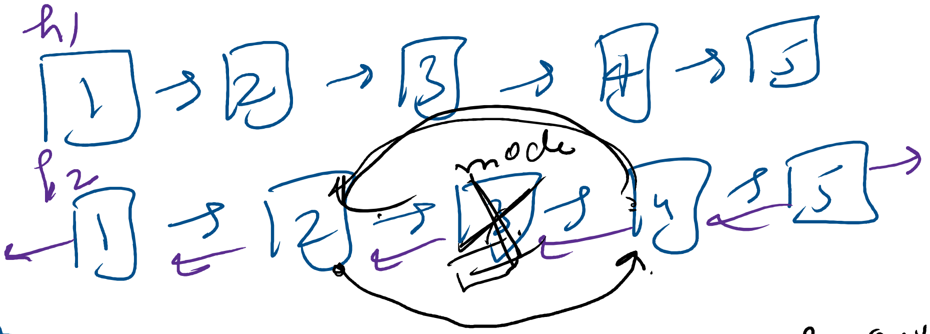


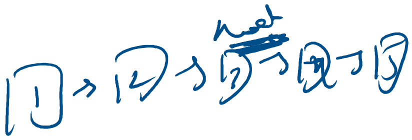
```
Node {
int data;
node next.
Node prev;
}
```

4. next

3

Time complexity analysis of Lists



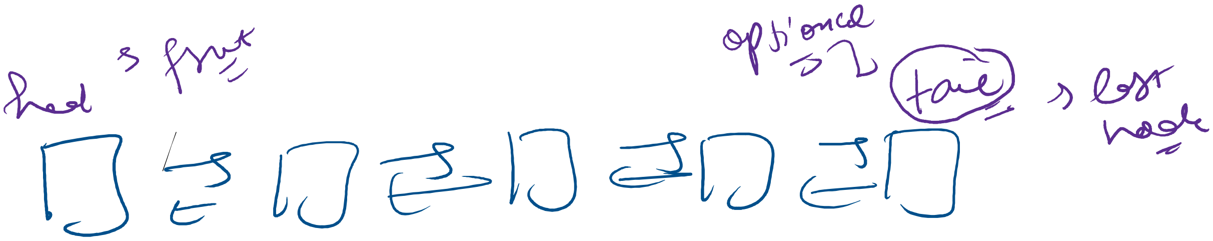


node.prev.next = node.next

node.next.prev = node.prev

→ O(1)

→ Deletion in doubly LL is O(1), and more efficient than array, array list and singly LL. But it comes with an assumption that input node is given to us.



⇒ Edge cases.

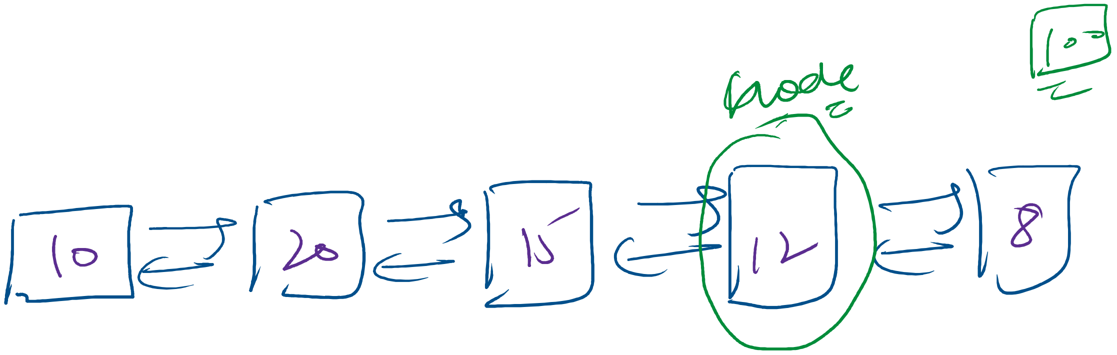
=> Edge cases.

1) head == null (empty list)

2) head == tail (single list)

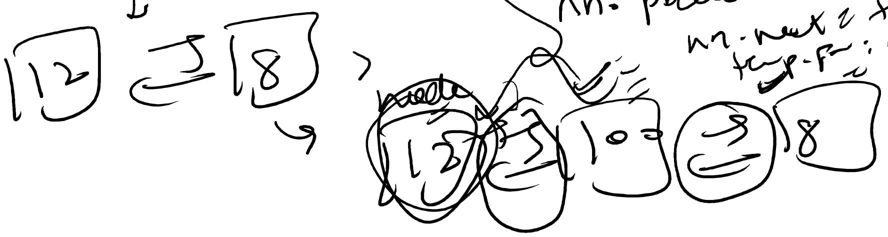
3) head (deletion first node)

4) tail (deletion of last node)



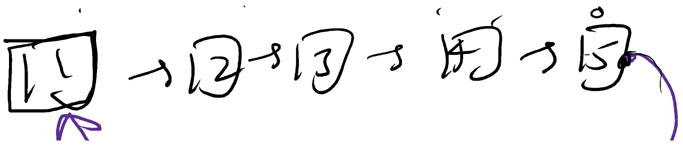
insert (node) {

```
Node nn = new Node(100);
node.next = temp
node.next = nn
nn.prev = node
nn.next = temp
temp.prev = nn
```

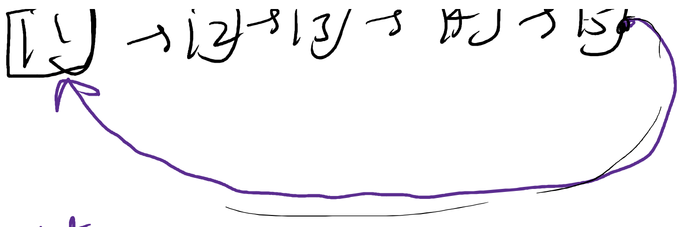


Circular_LL

1 -> 2 -> 3 -> 4 -> 5 ->



Circular



Ⓐ Last ka next pointer always points to head

SLL while (curr != null)

CLL while (curr.next != head)