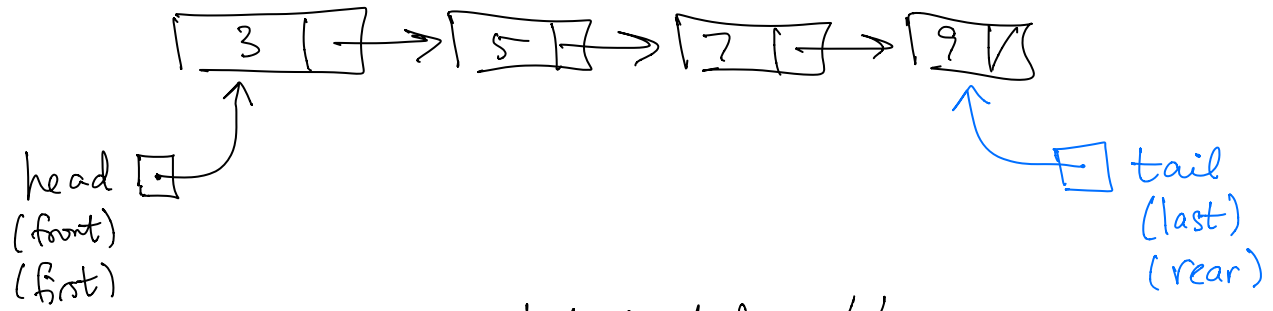


Func: print the last element in a LL.

What is the big-oh? O(n) – head pointer only

O(1) – head & tail pointers

Code for LL – DRAW A LOT OF PICTURES

Build a Singly-linked-list class to hold ints.

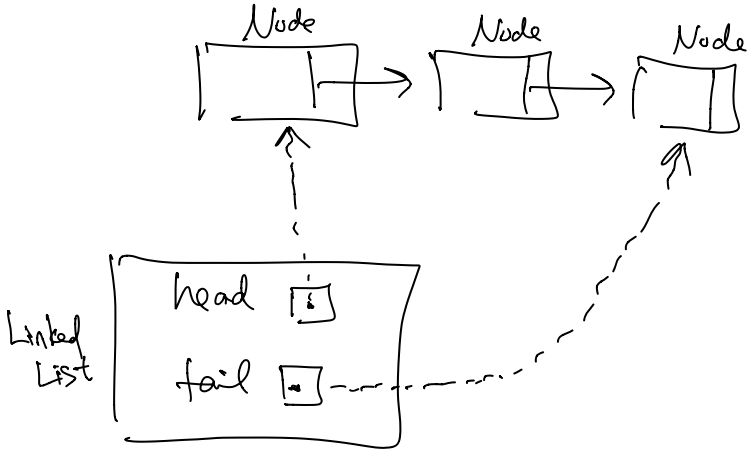
Node class 

Node 2 pieces → data
→ pointer/reference to the next item in the list.

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

← this is not a Node "inside" a Node, it is a reference to the next Node.

```
class LinkedList {
    Node head;
    (Node tail;)
}
```

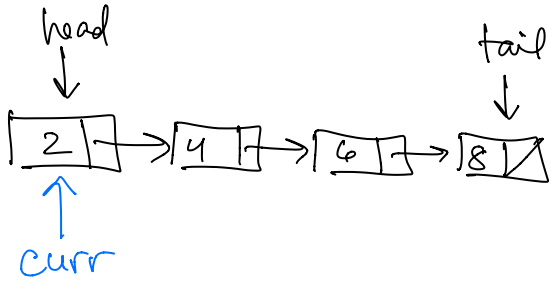


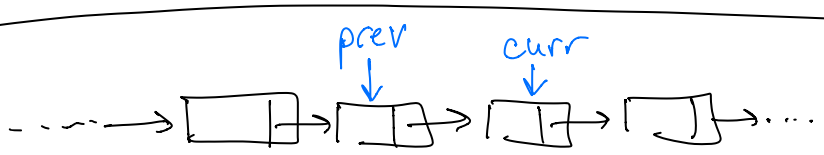
Traversals : going through the LL one element at a time.

1-pointer traversal

Forward direction (left to right)

```
for(int x=0;                 Node curr = (wherever you want to start traversing)   [normally this is the head]
    x<arr.size()
    x++)
{
                             while( stopping condition is not true
                                    (normally curr != null)
                             {
                                 // processing step with
                                 // curr.data
                                 example: System.out.println(curr.data)
                                 curr = curr.next;
                             }
}
```





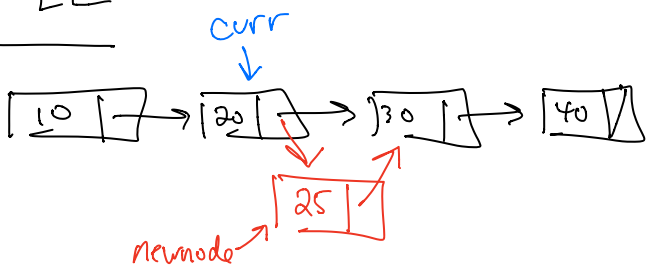
2-pointer traversal

```
Node curr = head (or somewhere)
Node prev = null

while (curr != null)
{
    // process curr.data
    .
    .
    .
    prev = curr;          <-- prev = prev.next ???
    curr = curr.next;
}
```

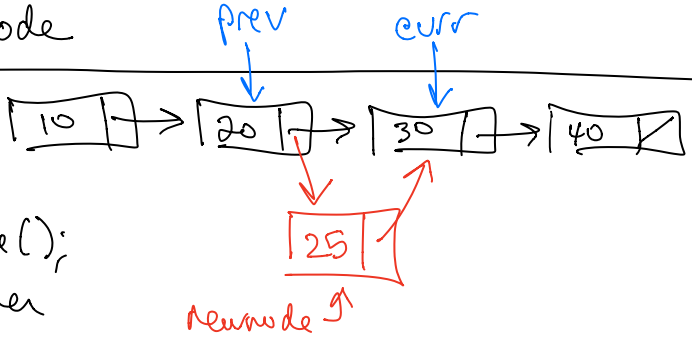
INSERTING INTO A LL

inserting after a node



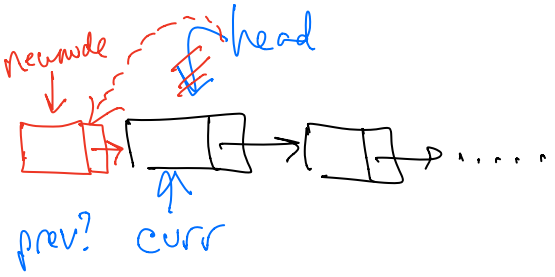

```
Node newnode = new Node();
newnode.data = whatever
newnode.next = curr.next
curr.next = newnode
```

inserting before a node



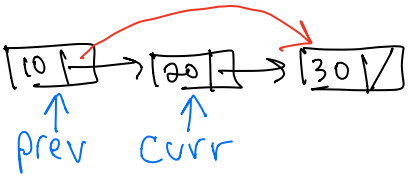

```
Node newnode = new Node();
newnode.data = whatever
newnode.next = curr
prev.next = newnode
```

insert @ the beginning of a LL



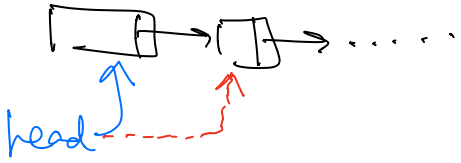

```
newnode.next = curr/head;
→ head = newnode;
```

Deletion

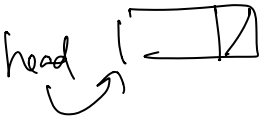



```
prev.next = curr.next
```

Deleting the head of a LL




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

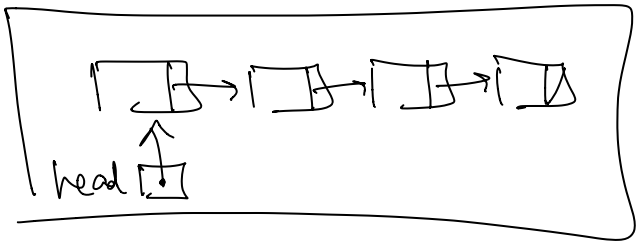


Big-oh of insert/delete

Most of the time*, insert/delete are O(1)

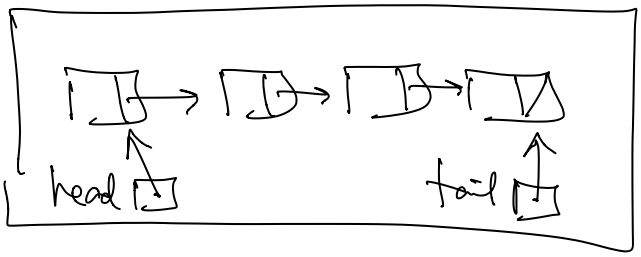
* → Takes time to traverse the list.

- SLL w/ only a head pointer



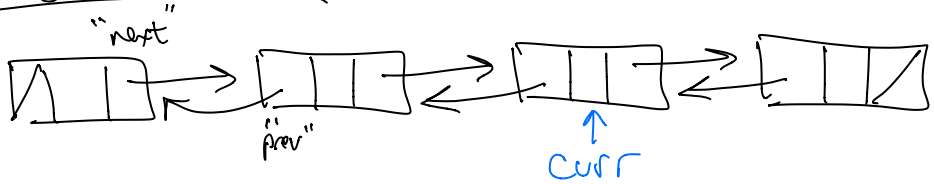
  - insert @ beginning = O(1)
  - insert @ end = O(n)
  - delete @ beginning = O(1)
  - delete @ end = O(n)
  - in general: insert/delete = O(1) + O(traversal)

- SLL w/ both head + tail pointers



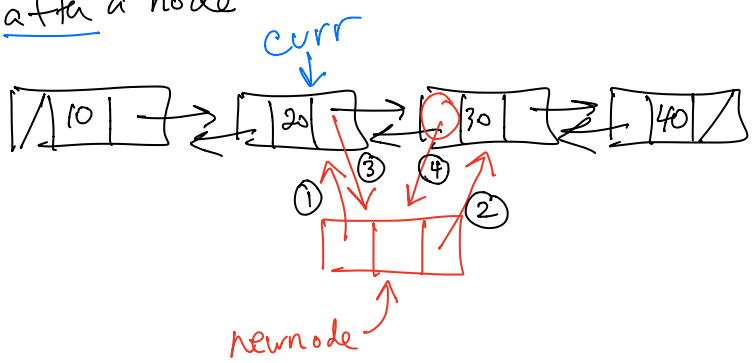
  - insert @ beg - O(1)
  - " @ end - O(1)
  - delete @ beg - O(1)
  - delete @ end - O(n)

Doubly-linked-list




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

Inserting after a node

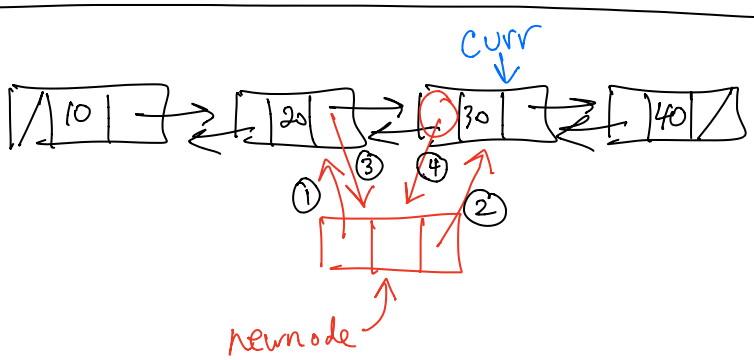


1. newnode.prev = curr
2. newnode.next = curr.next
3. curr.next = newnode
4. newnode.next.prev = newnode

|||

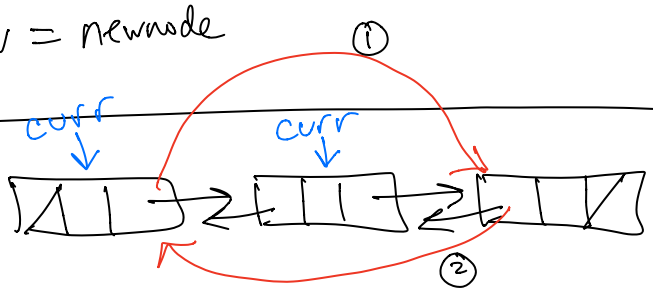
4. curr.next.prev = newnode
3. curr.next = newnode

Inserting before a node



1. newnode.prev = curr.prev
2. newnode.next = curr
3. curr.prev.next = newnode
4. curr.prev = newnode

Deletion



1. curr.prev.next = curr.next
2. curr.next.prev = curr.prev

Deletion @ head or tail

Detect that ahead of time & do special cases.

```
if (curr.prev == null)  // curr == head
if (curr.next == null) / if (curr == tail)
```

Big-oh

insert/deleting @ head/tail — O(1)

" " in middle — O(1) + O(traversal)